2001 Edition

Expert Analysis | Performance | Applications | Architecture | Benchmarks

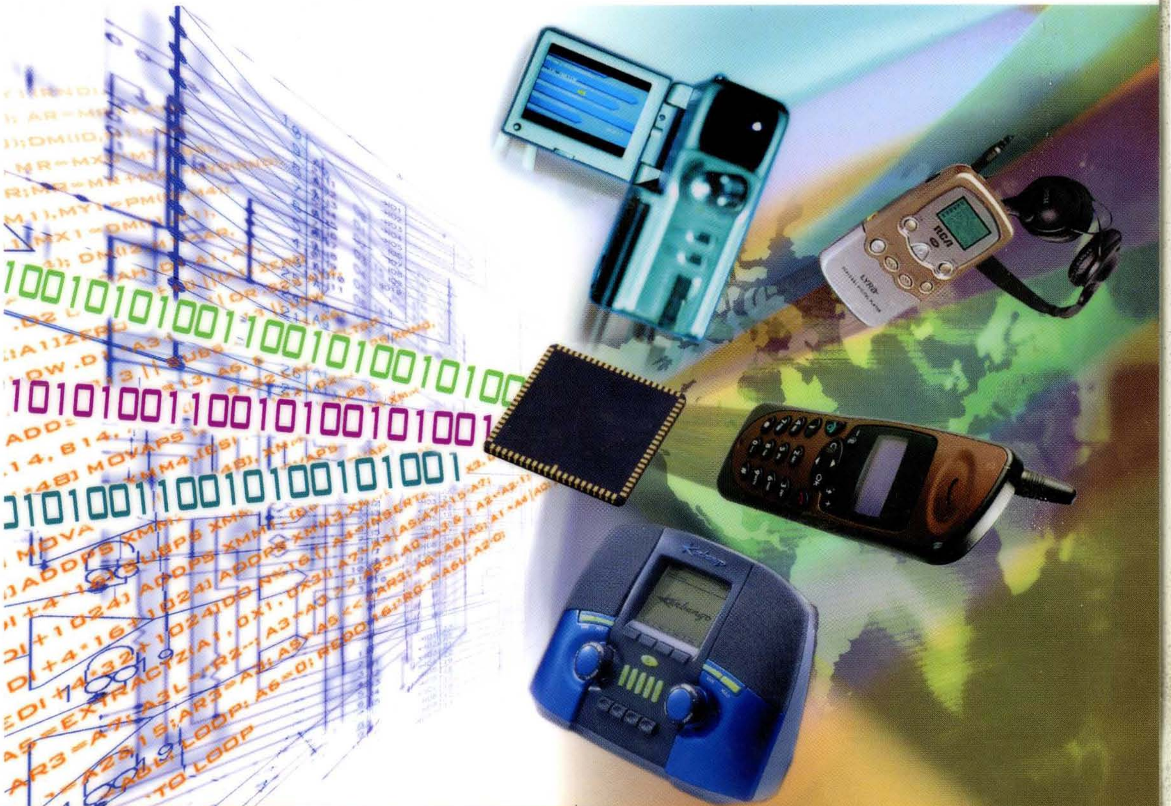
Buyer's Guide to DSP Processors

New in the 2001 Edition:

Analog Devices
ADSP-219x
TigerSHARC

Motorola
DSP56800E
StarCore SC140/
Motorola MSC8101

Texas Instruments
TMS320C55xx
TMS320C64xx



An Independent Evaluation of the Most Popular Processors Used in DSP

Includes results from the updated and expanded version of the BDTI Benchmarks™

BDTi

Berkeley Design Technology Inc

While every reasonable precaution has been taken in the preparation of this work to ensure accuracy of the information contained herein, Berkeley Design Technology, Inc., and its agents by necessity assume no responsibility for accuracy, errors or omissions, or for the uses made of any information and/or material contained herein or the decisions based on such use. NO WARRANTIES ARE MADE, EXPRESS OR IMPLIED, WITH REGARD TO THE CONTENTS OF THIS WORK, ITS MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. BERKELEY DESIGN TECHNOLOGY, INC., AND ITS AGENTS SHALL NOT BE RESPONSIBLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE AND/OR RELIANCE ON THE CONTENTS OF THIS WORK.

0101

Throughout this report, trademark names are used. Rather than place a trademark symbol at every occurrence, we hereby state that we are using the names only in an editorial fashion, with no intention of infringement of the trademark.

Berkeley Design Technology, Inc.
2107 Dwight Way, Second Floor
Berkeley, CA 94704 USA

Telephone: +1 (510) 665-1600
Fax: +1 (510) 665-1680
E-mail: info@BDTI.com
WWW: http://www.BDTI.com

Support

For questions or comments regarding this report and information on updates and related reports and services, contact:

Berkeley Design Technology, Inc.
2107 Dwight Way, Second Floor
Berkeley, CA 94704 USA

Telephone: +1 (510) 665-1600
Fax: +1 (510) 665-1680
Email: info@BDTI.com
WWW: http://www.BDTI.com

Please complete and return a photocopy of the registration form on the following page to ensure that you receive timely information regarding updates to this report.

Excerpt Policy

Please contact us for information about our established policy regarding the use of excerpts from this work.

Registration Form

To ensure that you receive timely information regarding updates to this report, please photocopy, complete, and return this form by mail or fax to:

Berkeley Design Technology, Inc.
2107 Dwight Way, Second Floor
Berkeley, CA 94704 USA

Fax: +1 (510) 665-1680

Your Name	
Title	
Company Name	
Mail Stop	
Street Address	
City/State/Postal Code	
Country	
Telephone	
Fax	
Email Address	
DSP Application(s)	
DSP Processors Used (If Any)	
How did you learn of this report?	
Comments	
Report Title	*Buyer's Guide to DSP Processors (0101)*

Table of Contents

1. Introduction

This report is a comprehensive technical guide to programmable digital signal processors. Programmable digital signal processors (often called DSPs, PDSPs, or DSP processors) are microprocessors that are specialized to perform well in digital signal processing-intensive applications. Since the introduction of the first commercially successful DSP processors in the early 1980s, dozens of new processors have been developed, offering system designers a vast array of choices. According to the market research firm Forward Concepts, sales of user-programmable DSPs will total roughly US $4.4 billion in 2001, with a projected annual growth rate of 40% [Stra98]. With semiconductor manufacturers vying for bigger shares of this booming market, designers' choices will broaden even further in the next few years.

Scope and Purpose

This report is intended for anyone who is evaluating or comparing DSP processors. Our emphasis is on completeness, in-depth analysis, objectivity, and consistency. We present each of the key elements of DSP processor technology and examine current product offerings with a critical eye. We expect that this report will be especially useful for electronic systems designers, processor developers, engineering managers, product planners, and marketing managers. It will aid in choosing or designing the DSP processor or processors that are best suited to a given application and in developing an understanding of how the capabilities of DSP processors can be used to meet the needs of the application.

Changes in the 2001 Edition

This is the 2001 Edition of *Buyer's Guide to DSP Processors.* The most significant changes from the 1999 edition include:

- **New Processors**
  We have added analysis and benchmark results for five new DSP processor families: the Texas Instruments TMS320C55xx, Texas Instruments TMS320C64xx, Analog Devices ADSP-219x, Motorola DSP5685x, and the StarCore SC140. We have also added an extensive qualitative analysis of Analog Devices' TigerSHARC architecture (benchmark results were not available at the time this report was published).
- **New Benchmarks**
  This is the first edition of *Buyer's Guide to DSP Processors* to use the new version of the BDTI Benchmarks™, released in 1999. The new edition of the benchmark suite adds three new benchmarks (Viterbi Decoder, Control, and Bit Unpack), eliminates two benchmarks (Convolutional Encoder and FSM), and revises the specifications for several of the original BDTI Benchmarks, such as the Two-Biquad IIR and 256-Point FFT benchmarks.

- **BDTImark2000™ Scores**
  With the introduction of the new version of the BDTI Benchmarks, a new version of the BDTImark has also been created: the BDTImark2000. Like the original BDTImark, the BDTImark2000 provides a convenient estimate of processor's DSP speed and is much more realistic that traditional simplified metrics like MIPS and MFLOPS.
- **Revised Analyses for Previously Evaluated Processors**
  We have expanded and updated our analysis for previously evaluated processor families with new insights and new family members, as well as updates on prices, package options, speeds, operating voltages, tools, and many other critical details. We have implemented the latest version of the BDTI Benchmarks for older processors that are included in the benchmark analysis in this report.
- **Coverage of Older or Highly Specialized Processors**
  For some older processors we have either eliminated coverage entirely or limited coverage to a description of the architecture without including detailed benchmark results. Additionally, we have discontinued coverage of processors that are highly specialized for niche applications. Excluding these processors allowed us to focus our evaluation on processors of the most interest to the largest number of users.
- **Scope of Coverage of DSP Processor Architectural Concepts**
  In this edition of *Buyer's Guide*, we have eliminated the detailed introduction to DSP processor architectures. This introduction can be found in BDTI's textbook, *DSP Processor Fundamentals* [BDTI96].

Organization

This report is organized as follows:

- **Authors**
  Chapter 2 provides brief background information on the authors of this report.
- **Digital Signal Processing and DSP Systems**
  Chapter 3 provides a high-level overview of digital signal processing, including DSP system features and applications.
- **DSP Processor Embodiments and Alternatives**
  Chapter 4 provides a brief introduction to DSP processors and then discusses the different forms that DSP processors take, including chips, multi-chip modules, and SoC cores. In this chapter we also briefly touch on alternatives to DSP processors, such as fixed-function DSP integrated circuits and general-purpose processors.
- **Guidance for Choosing a Processor**
  In Chapter 6 we present detailed run-time profiling data to illustrate the kinds of demands that typical applications make on DSP processors.

- **Information on Other Processors**
  In Chapter 5 we provide information on where to find analyses on processors that are not covered in this report.
- **In-Depth Processor Analyses**
  In Chapter 7 we provide in-depth analyses of seventeen DSP processors and processor families, highlighting the distinctive features, strengths, and weaknesses of each, including factors such as the quality of development tools and applications engineering support.
- **Benchmark Results**
  To provide a basis for fair comparisons of DSP processor performance, we have developed the BDTI Benchmarks™. Chapter 8 presents benchmark results for fourteen processors and processor families, and examines the number of cycles required to execute the benchmarks, execution speed, cost-performance, energy efficiency, and memory usage.
- **Conclusions**
  In Chapter 9 we present our conclusions on strategies for comparing processors, the current state-of-the-art in DSP processors, and likely future developments in DSP processor technology.
- **Vendor Contact Information**
  This appendix contains contact information for companies that sell DSP processors and related products and services covered in this report.
- **References, Bibliography, Glossary, and Index**
  A glossary of DSP processor-related terms provides definitions of technical terminology used in this report. The bibliography lists useful sources of information for those interested in delving more deeply into the topics covered here. The References section lists related publications referenced in this document. References are denoted with square brackets, as in [BDTI97]. To help you quickly find the information you need, an extensive index is included at the end of this report.

Related Resources

BDTI offers a number of products and services related to this report.

- BDTI publishes technical evaluations which provide in-depth analyses of specific processors. These reports provide detailed benchmark results, along with comparisons to competing processors from other vendors. Technical evaluations published to date include *Inside the StarCore SC140, Inside the Infineon Carmel, Inside the Siemens TriCore*, and *Inside the Lucent DSP16000.*
- BDTI offers the Benchmark Analysis Tool (BAT), a software complement to *Buyer's Guide to DSP Processors*. The BAT simplifies processor selection and competitive analysis by allowing easy customization of BDTI's detailed DSP benchmark analyses to study specific scenarios.

- BDTI licenses the BDTI Benchmark specification and methodology to processor developers and users to facilitate in-house benchmarking.
- BDTI provides a variety of training and consulting services which are further described in Chapter 2.
- BDTI's website, *www.BDTI.com*, offers a variety of information about DSP, including the *comp.dsp* newsgroup FAQ, summary descriptions of nearly all currently available DSP processors and cores, a pocket guide to DSP processors, and articles by BDTI authors.

2. About the Authors

Berkeley Design Technology, Inc. (BDTI) was founded in 1991 to assist companies in creating, selecting, and using DSP technology. The technical staff of BDTI has extensive experience in the development of DSP-intensive software and hardware for commercial applications. In addition, each of the founders of BDTI has been a key contributor to pioneering research in the field of DSP design tools and methodologies at the University of California at Berkeley.

Berkeley Design Technology, Inc. offers a variety of services, including:

- **Published reports on DSP technology.** BDTI publishes a variety of unique technical reports and books. *Buyer's Guide to DSP Processors* is BDTI's comprehensive technical analysis of programmable digital signal processors. Nearly 900 pages in length, *Buyer's Guide* contains in-depth evaluations of the architecture, instruction set, peripherals, development tools, and applications support of every major commercial DSP processor. The evaluations are quantified with processors' scores on the BDTI Benchmarks™, a suite of critical DSP algorithms that have become the industry-standard measure of DSP performance. Every benchmark implementation is coded in assembly language and painstakingly optimized to reveal each processor's true performance potential.

  BDTI also publishes a series of smaller, more focused reports that cover single processors. These reports include *Inside the StarCore SC140*, *Inside the Infineon Carmel, Inside the Siemens TriCore*, and *Inside the Lucent DSP16000.* New reports are added to the series regularly; contact BDTI for information.

- **Development of DSP software.** BDTI applies its unique expertise in processor architectures and DSP applications to provide DSP software development services. BDTI develops highly optimized DSP software for component libraries, modules, and complete applications, especially for audio and telecommunications applications. BDTI has experience programming a wide variety of target processors.
- **Consulting and processor evaluation services.** BDTI provides consulting services to leading companies that develop and use DSP technology. BDTI's consulting expertise is in the evaluation and specification of processor architectures, DSP design tools, DSP algorithms, and other DSP technology. BDTI's processor evaluation methodology, which features the BDTI Benchmark specification, is available for license to processor developers and users.
- **Training.** BDTI offers courses to help engineers, marketers, and managers develop their knowledge of DSP technology. Courses are available for on-site delivery or in electronic format. Course descriptions are posted on BDTI's website at *www.BDTI.com.* BDTI also develops custom courses for customers' specific needs.

BDTI's customers include major semiconductor, consumer electronics, telecommunications, and software companies who are leaders in the development and application of DSP technology.

The authors welcome your comments. Please forward your corrections and suggestions to the authors in care of BDTI.

BDTI can be reached by telephone at +1 (510) 665-1600, by fax at +1 (510) 665-1680, by electronic mail at info@BDTI.com, and on the World Wide Web at *www.BDTI.com.*

Primary Authors

Jeff Bier is a founder and General Manager of BDTI. Mr. Bier is the author of numerous industry reports on DSP technology and is a member of the IEEE Design and Implementation of Signal Processing Systems and Industry DSP technical committees. Mr. Bier received his B.S. and M.S. degrees from Princeton University and U.C. Berkeley, respectively.

Laurent Bonetto is a DSP Engineer with BDTI, where he works on evaluating processors and on developing DSP software. Mr. Bonetto received his M.E. degree from the Georgia Institute of Technology, where he worked on Synthetic Aperture Radars with Prof. James McClellan. Mr. Bonetto also holds degrees in Mathematics and in Engineering from the Electrical and Computer Engineering School, Supelec, France, where his principal focus was DSP theory.

Brian Cavagnolo is a DSP Engineer with BDTI, where he is involved in processor evaluation and DSP software development. He holds a B.S. degree in electrical engineering from the U.C. Berkeley.

Jennifer Eyre is BDTI's Manager of Analysis and Publications. She is responsible for overseeing BDTI's analyses of processor architectures. Ms. Eyre holds an MSEE from UCLA and is author or co-author of numerous reports and articles on DSP processor technology.

Bjorn Hori is a DSP Engineer with BDTI, where he focuses primarily on digital audio applications and analysis of DSP processors. Mr. Hori received his B.S. degree in electrical engineering and computer science from U.C. Berkeley.

Phil Lapsley is a co-founder of BDTI. Mr. Lapsley is an expert on real-time DSP techniques and has extensive experience in the analysis of DSP processors, tools, and related technology. Mr. Lapsley received his B.S. and M.S. degrees from U.C. Berkeley.

Adam Lins is BDTI's Engineering Manager. Mr. Lins oversees BDTI's software development activities, and contributes to BDTI's processor analysis activities. Mr. Lins holds an M.S. degree in electrical engineering from the University of Canterbury, New Zealand, where his principal focus was digital communication systems.

Frantz Lohier is a DSP Engineer with BDTI, where he focuses primarily on evaluating processors and developing DSP software. Mr. Lohier holds M.S. and Ph.D. degrees from the Pierre et Marie Curie University, France. His Ph.D. thesis was on DSP Architectures and Programming Methodologies. Mr. Lohier also holds a French Computer Engineering Diploma.

Amit Shoham is a Senior DSP Engineer with BDTI, where he focuses primarily on benchmarking DSP processor performance and evaluating DSP design tools. His technical interests include digital audio and music synthesis. Prior to joining BDTI, Mr. Shoham developed factory diagnostics for digital audio hardware at Silicon Graphics. He holds a B.S. degree in computer systems engineering and an M.S. degree in electrical engineering, both from Stanford University.

Contributors

Daniel Ash, of Ashcan Engineering, contributed to the analysis of the TMS320C54xx and TMS320C55xx.

Cynthia Keller, BDTI's Office Manager, managed project administration and contributed to research and document production.

John Strawn, of S Systems, Inc., contributed to the analysis of the ADSP-TS0xx, DSP563xx, DSP568xx, and DSP5685x.

Acknowledgments

This report would not have been possible without the help of many people.

We wish to especially acknowledge the assistance of those outside BDTI who assisted with benchmark coding, processor evaluations, and project administration: Maria Tagliaferro and David Starr of Analog Devices; Andy Soukup, Henry Wiechman, David Hoyle, Jim Larimer, and Martin Burgos of Texas Instruments; Yuval Ronen and Joe Gergen of Motorola; Diana Yannes and John Sweeney of Lucent Technologies. All gave generously of their time and expertise.

3. Digital Signal Processing and DSP Systems

For the purposes of this report, we define a DSP system to be any electronic system making use of digital signal processing. Our informal definition of digital signal processing is the application of mathematical operations to digitally represented signals. Signals are represented digitally as sequences of *samples*. Often, these samples are obtained from physical signals (for example, audio signals) through the use of *transducers* (such as microphones) and *analog-to-digital converters*. After mathematical processing, digital signals may be converted back to physical signals via *digital-to-analog converters* and transducers (such as speakers).

In some systems, the use of DSP is central to the operation of the system. For example, modems and digital cellular telephones rely very heavily on DSP technology. In other products, the use of DSP is less central, but often offers important competitive advantages in terms of features, performance, and cost. For example, manufacturers of analog consumer electronics devices such as audio amplifiers widely employ DSP technology to add features such as simulation of concert hall acoustics.

This chapter presents a high-level overview of digital signal processing. We first discuss the advantages of DSP over analog systems. We then describe some salient features and characteristics of DSP systems in general. We conclude with a brief look at some important classes of DSP applications.

This chapter is not intended to be a tutorial on DSP theory. For a general introduction to DSP theory, we recommend one of the many textbooks available on DSP, such as *Discrete-Time Signal Processing* by Oppenheim and Schafer [Oppe89] or *Understanding Digital Signal Processing* by Richard G. Lyons [Lyon97].

Advantages of DSP

Digital signal processing enjoys several advantages over analog signal processing. The most significant of these is that DSP systems are able to accomplish tasks inexpensively that would be difficult or even impossible using analog electronics. Examples of such applications include speech synthesis, speech recognition, and high-speed data communication using error-correction coding. All of these tasks involve a combination of signal processing and control (e.g., making decisions regarding received bits or operating conditions) that is extremely difficult to implement using analog techniques.

DSP systems also enjoy two additional advantages over analog systems:

- **Insensitivity to environment.** Digital systems, by their very nature, are considerably less sensitive to environmental conditions than analog systems. For example, an analog circuit's behavior depends on its temperature. In contrast, barring catastrophic failures, a DSP system's operation does not depend on its environment. Whether in the snow or the desert, a DSP system delivers the same response.
- **Insensitivity to component tolerances.** Analog components are manufactured to particular tolerances—a resistor, for example, might be guaranteed to have a resis-

tance within one percent of its nominal value. The overall behavior of an analog system depends on the actual values of all of the analog components used. As a result, two analog systems of exactly the same design will have slightly different behaviors due to slight variations in their components. In contrast, barring a malfunction, two identical digital systems will always produce the same outputs given the same inputs.

These two advantages combine synergistically to give DSP systems an additional advantage over analog systems:

- **Predictable, repeatable behavior.** Because a DSP system's output does not vary due to environmental factors or component variations, it is possible to design systems having exact, known responses that do not vary.

Finally, some DSP systems have two other advantages over analog systems:

- **Reprogrammability.** If a DSP system is based on a programmable processor or other programmable device, it can be reprogrammed—even in the field—to perform other tasks. In contrast, analog systems usually require physically different components to perform different tasks.
- **Size.** The size of analog components varies with their values; for example, a 100-microfarad capacitor used in an analog filter is physically larger than a 10-picofarad capacitor used in a different analog filter. In contrast, DSP implementations of both filters might well be the same size—indeed, might even use the same hardware, differing only in their filter coefficients—and might be smaller than either of the two analog implementations.

These advantages, coupled with the fact that DSP can take advantage of the rapidly increasing densities and speeds enabled by more advanced digital IC manufacturing processes, make DSP the solution of choice for an expanding range of signal processing applications.

Characteristics of DSP Systems

In this section we describe a number of characteristics common to all DSP systems, including algorithms, sample rate, clock rate, and arithmetic types.

Algorithms

DSP systems are often characterized by the *algorithms* used. The algorithms specify the arithmetic operations to be performed but do not specify how those operations are to be implemented. They might be implemented in software on an ordinary microprocessor or programmable signal processor, or they might be implemented in custom integrated circuits. The selection of an implementation technology is determined in part by the required processing speed and arithmetic precision. Table 3.0-1 lists some common types of DSP algorithms and some applications in which they are typically used.

Classes of DSP Algorithms	System Application
Speech coding and decoding	Digital cellular telephones, voice-over-Internet, digital cordless telephones, multimedia computers, secure communications, tapeless answering machines
Speech encryption and decryption	Digital cellular telephones, personal communications systems, secure communications
Speech recognition	Advanced user interfaces, multimedia computers, robotics, automotive applications, cellular telephones, personal communications systems, voice response systems
Speech synthesis	Advanced user interfaces, robotics, voice response systems
Speaker identification	Security, multimedia computers, advanced user interfaces
High-fidelity audio encoding and decoding	Consumer audio, consumer video, digital audio broadcast, professional audio, multimedia computers, Internet audio
Modem algorithms	Digital cellular telephones, personal communications systems, digital cordless telephones, digital audio broadcast, digital signaling on cable TV, multimedia computers, wireless computing, navigation, data/facsimile modems, secure communications
Noise cancellation	Professional audio, advanced vehicular audio, industrial applications
Audio equalization	Consumer audio, professional audio, advanced vehicular audio, music, hearing aids
Ambient acoustics emulation	Consumer audio, professional audio, advanced vehicular audio, music, games
Audio mixing and editing	Professional audio, music, multimedia computers
Sound synthesis	Professional audio, music, multimedia computers, advanced user interfaces, games
Vision	Security, manufacturing, advanced user interfaces, instrumentation, robotics, navigation
Image compression and decompression	Digital photography, digital video, multimedia computers, videoconferencing, consumer video
Image compositing	Multimedia computers, consumer video, advanced user interfaces, navigation
Beamforming	Navigation, medical imaging, radar, sonar, signals intelligence, cellular base stations
Echo cancellation	Speakerphones, hands-free cellular telephones
Spectral estimation	Signals intelligence, radar, sonar, professional audio, music

TABLE 3.0-1. Common DSP algorithms and typical applications.

Sample Rates

A key characteristic of a DSP system is its *sample rate:* the rate at which samples are consumed, processed, or produced. Combined with the complexity of the algorithms used in the system, the sample rate determines the required speed of the implementation technology. A familiar example is the digital audio compact disc (CD) player, which produces samples at a rate of 44.1 kHz on two channels.

Of course, a DSP system may use more than one sample rate; such systems are said to be *multirate DSP systems*. An example is a converter from the CD sample rate of 44.1 kHz to the digital audio tape (DAT) rate of 48 kHz. Because of the awkward ratio between these sample rates, the conversion is usually done in stages, typically with at least two intermediate sample rates. Another example of a multirate algorithm is a filter bank, used in applications such as speech, audio, and video encoding and some signal analysis algorithms. Filter banks typically consist of stages that divide a signal into high- and low-frequency portions. These new signals are then downsampled (i.e., their sample rate is lowered by periodically discarding samples) and divided again. In multirate applications, the ratio between the highest and the lowest sample rates in the system can become quite large, sometimes exceeding 100,000 to 1.

The range of sample rates encountered in signal processing systems is huge. In Figure 3.0-1 we show the rough positioning of a few classes of applications with respect to algorithm complexity and sample rate. Sample rates for applications range over 12 orders of magnitude! Only at the very top of that range is digital implementation rare. This is because the cost and difficulty of implementing a given algorithm digitally increases with the sample rate. For this same reason, DSP applications that use at higher sample rates tend to use less complex algorithms than those using lower sample rates.

Clock Rates

Digital electronic systems are often characterized by their *clock rates*. The clock rate usually refers to the rate at which a system or sub-system performs its most basic unit of work. Often, systems use different clock rates for different purposes. Within a single chip, clock rates of 500 MHz and higher are becoming common in mass-produced, commercial products. Between chips, clock rates of up to 100 MHz are common, with faster rates found in some high-performance products. For DSP systems, the ratio of clock rate to sample rate is one of the most important characteristics used to determine how the system will be implemented. The relationship between the clock rate and the sample rate in a system or component partially determines the amount of hardware needed to implement a given algorithm in real-time. As the ratio of sample rate to clock rate increases, so does the amount and complexity of hardware required to implement the algorithm.

Numeric Representations

Arithmetic operations such as addition and multiplication are at the heart of DSP algorithms and systems, and signal fidelity (i.e., range and precision) is usually a key performance metric. As a result, the numeric representations and type of arithmetic used can have a profound influence on the behavior and performance of a DSP system. The most important choice for the designer is between *fixed-point* and *floating-point* arithmetic. Fixed-point arithmetic represents numbers in a fixed range (e.g., -1.0 to +1.0) with a finite number of bits of precision (called the *word width*). For example, an eight-bit fixed-point number provides a resolution of 1/256 of the range over which the number is allowed to vary. Numbers outside of the specified range cannot be represented; arithmetic operations that would result in a number outside this range either *saturate* (that is, are limited to the largest positive or negative representable value) or *overflow* (that is, the extra bits resulting from the arithmetic operation are discarded).

Floating-point arithmetic greatly expands the representable range of values. Floating-point arithmetic represents every number in two parts: a mantissa and an exponent. The mantissa is, in effect, forced to lie between -1.0 and 1.0, while the exponent keeps track of the amount by which the mantissa must be scaled (in terms of powers of two) in order to create the actual value represented. That is:

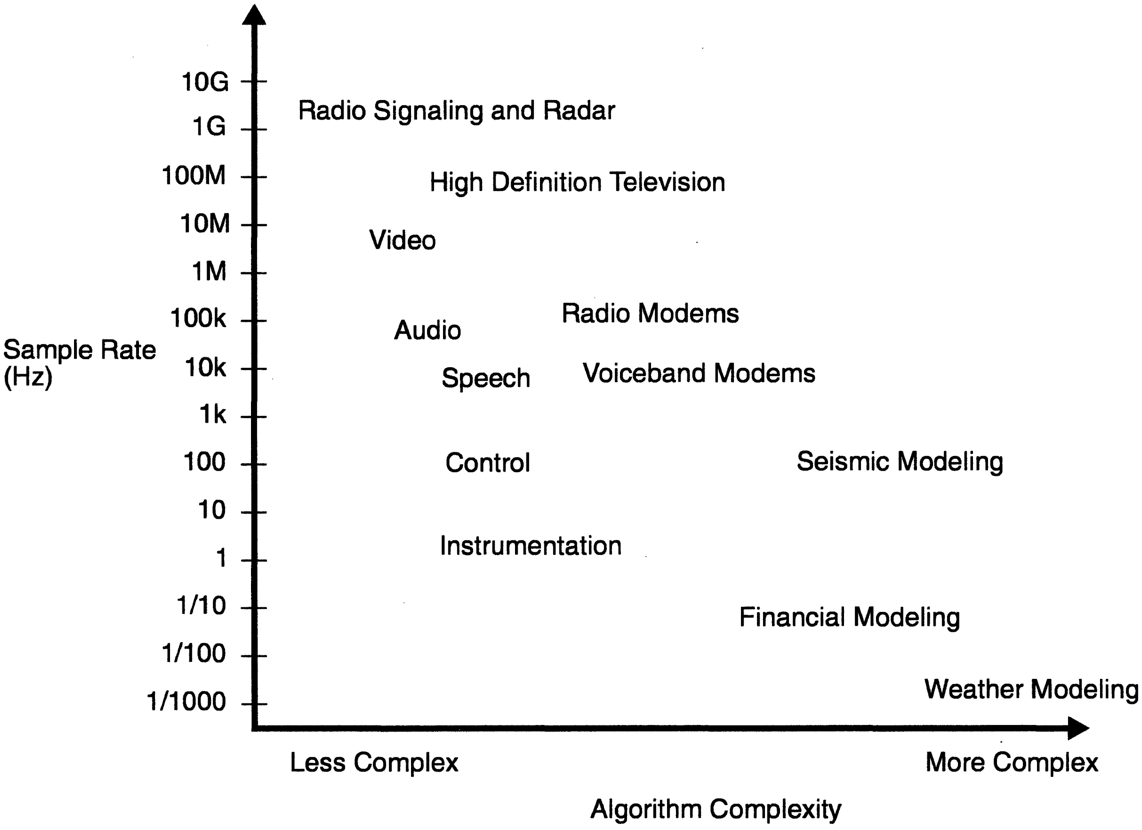


FIGURE 3.0-1. A rough illustration of sample rates and relative algorithm complexities for a variety of DSP application classes.

$$value = mantissa \times 2^{exponent}$$

Floating-point arithmetic provides much greater dynamic range (that is, the ratio between the largest and smallest values that can be represented) than fixed-point arithmetic using the same number of bits. Because it reduces the probability of overflow and the necessity of scaling, floating-point arithmetic can considerably simplify algorithm and software design. Unfortunately, processors that use floating-point arithmetic are generally slower and more expensive than processors that use fixed-point arithmetic, because floating-point arithmetic is more complicated to implement in hardware.

Arithmetic and numeric formats are discussed in more detail in BDTI's textbook, *DSP Processor Fundamentals*.

Execution-Time Predictability

Many DSP systems are subject to *hard real-time constraints*, meaning that the system must process or respond to inputs within a specified amount of time in every instance. In such systems, failure to meet real-time deadlines may cause malfunctions ranging from a reduction in signal quality to a loss of data to failure of a communications link. This type of performance constraint differs from, for example, performance requirements of personal computers. In a personal computer, the system is expected to respond to inputs within a reasonable amount of time on average, but exceeding the maximum desired response time is generally not considered a failure.

To ensure that hard real-time constraints are met, the programmer must be able to predict how much time is required to execute time-critical sections of the application software. Hence, a processor's *execution-time predictability* is often a significant consideration in real-time DSP applications.

In many cases, programmers writing real-time DSP applications can execute their software on a development board and measure the execution time. Unfortunately, measuring execution time does not guarantee that the worst-case scenario is known; in some architectures, the execution timing of a specific segment of software may change depending on the instructions that preceded it, or depending on the locations of the instructions and associated data in memory. Execution timing may also be data dependent. Hence, measuring the execution time on hardware often does not solve the problem.

Even in the absence of hard real-time constraints execution-time predictability can be important, because it plays a role in software optimization. If it is difficult to predict how long a given section of software will take to execute, it can be difficult to determine the effect of changes to the software—will the modified software require more, less, or the same amount of time as the original software?

Of course, all processors are fundamentally deterministic; that is, given enough information about the processor's architecture and state, it is possible to predict the exact number of clock cycles required to execute a specific segment of software. However, the ease with which execution times can be predicted varies widely. Most DSP processors have relatively straightforward architectures and are supported by tools to help the programmer predict execution times, such as software simulators that accurately report elapsed instruction cycles. In contrast, most high-performance general-purpose processors have very complex architectures and lack tool support to aid programmers in predicting execution times. These factors may make it extremely difficult to predict how long a section of DSP software will take to execute, thus complicating DSP software development and optimization on high-performance CPUs.

The complexity of DSP algorithms coupled with high data rates means that in many DSP applications, programmers develop (or optimize) software in assembly language in order to squeeze the maximum performance out of the processor. In such cases the application programmer must understand the intricacies of the processor's architecture (including execution timing) in order to effectively select a processor, predict performance, and optimize software. Where performance is not critical, developers sometimes make use of high-level language compilers to quickly generate application software. However, poor execution-time predictability is often a challenge for the compiler as well as the assembly-language programmer; if the rules governing the execution time of a small block of software are complicated, it may be difficult for the compiler to generate optimized software.

A few of the latest architectures targeting DSP applications have begun to incorporate dynamic features traditionally found only in high-end general-purpose processors, in an effort to boost performance. For this reason, we include a brief discussion of some of the dynamic features and their impact on execution-time predictability.

Caches

High-performance DSP processors often use on-chip instruction caches, and in a few cases have recently also begun to incorporate data caches. When the required instructions and data are contained in the on-chip caches, the processor executes at full speed. Otherwise, the processor may be stalled while instructions and data are loaded into the caches from main memory. In real-time applications, caches can be problematic because they complicate the task of predicting software execution times. For this reason, many DSP processors allow programmers to manually control cache segments, thereby ensuring that critical instructions and data are present in the caches when needed. The cost of this control is that of degraded performance in other sections of the program.

Dynamic Memory

To reduce costs, some systems rely on dynamic RAM (DRAM) for their main memory. Depending on the type of DRAM devices chosen and the details of the system design, accesses to DRAM-based main memory may degrade execution-time predictability because the DRAM may temporarily be unavailable while data is refreshed, and because DRAM requires variable access times, for example access times increase when crossing memory page boundaries.

Branch Prediction

Program branches can be very costly in terms of execution time, because instructions following the branch that have already entered the processor's pipeline must be flushed, and the pipeline must be reloaded. (See BDTI's textbook, *DSP Processor Fundamentals,* for a discussion of pipelines.) One approach to decreasing this execution-time penalty is to provide hardware in the processor that attempts to predict the outcome of upcoming branches. The processor then fetches instructions based on the outcome of this prediction, in an attempt to avoid fetching unneeded instructions and flushing the pipeline. Branch prediction schemes on high-performance general-purpose processors are often quite complicated. For example, they often feature sophisticated branch prediction mechanisms that keep a record of branch statistics and attempt to detect patterns of taken and not-taken branches. This can be a very effective tool for increasing performance; however, complicated branch prediction schemes adversely affect execution-time predictability. Among processors for DSP, the Analog Devices TigerSHARC and the Infineon TriCore are examples of processors that include branch prediction. Their branch prediction schemes are simple, however, and do not detract from their execution-time predictability.

Dynamic Instruction Scheduling

Processors designed for DSP rely heavily on parallelism to achieve strong performance. Many DSP processors achieve parallelism by encoding several operations in a single instruction. Some processors achieve high parallelism by employing a *superscalar* architecture, in which several instructions are issued and executed in parallel. Superscalar processors dynamically select sequential instructions for parallel execution, depending on the available execution units and on dependencies between instructions. Run-time scheduling of instructions in superscalar processors can be quite a complex process, making execution timing difficult to predict.

Some superscalar processors, such as the Motorola PowerPC 604e and the Intel Pentium II and Pentium III processors, use *out-of-order* execution. When one of these processors fetches instructions from memory, the instructions are stored temporarily in a buffer in the processor and are not issued to their respective execution units until their operands become available. The processors can buffer dozens of instructions waiting to be issued. Since operands for the instructions may become available in a different order than the order in which the instructions occur in memory, the instructions may be issued to their respective execution units in a different order than they were fetched. Processors that employ out-of-order execution contain hardware that is responsible for committing results to registers and memory so that the processors appear to execute the instructions in the same order in which they occur in the program.

Some processors, such as the Texas Instruments TMS320C6xxx families, achieve high parallelism by employing a *VLIW* (very long instruction word) architecture. VLIW architectures are similar to superscalar architectures; several instructions are issued and executed in parallel. These instructions are fetched as part of one long super-instruction.

In a VLIW architecture, however, the programmer (or software-generation tool) explicitly specifies which instructions will be executed in parallel; this determination takes place before the program is executed, and does not affect the execution-time predictability of the processor.

Tools

Strong software and hardware development tools are essential for efficient application development in general, but are especially important for development of performance-critical, real-time applications. In such applications, developers need to be able to analyze and predict performance in detail, to perform real-time debugging, and to thoroughly optimize critical sections of software.

DSP processor tools generally include clock-cycle-accurate instruction-set simulators. Such simulators allow programmers to observe, cycle by cycle, software execution on the target processor for purposes of performance analysis, optimization, and debugging. For processors with difficult-to-predict execution times, the availability of a cycle-accurate simulator is an essential tool for software development and optimization. Development tools are discussed further in BDTI's textbook, *DSP Processor Fundamentals*.

Classes of DSP Applications

Digital signal processing is used in an extremely diverse range of applications, from radar systems to consumer electronics. Naturally, no one processor can meet the needs of all or even most applications. Therefore, the first task for the designer selecting a processor is to weigh the relative importance of performance, cost, integration, ease of development, power consumption, and other factors for the application at hand. Here we briefly touch on the needs of just a few categories of DSP applications. Table 3.0-2 summarizes these categories.

Category	Example Applications
Low-Cost Embedded Systems	Modems, radar detectors, pagers, cellular telephones, cordless telephones, disk drives, automotive real-time control
High-Performance Applications	Radar, sonar, seismic imaging, speaker identification
Personal Computer-Based Multimedia	Modems, voice mail, music synthesis, speech synthesis; speech, audio, and video compression and decompression

TABLE 3.0-2. Example DSP processor application types.

Low-Cost Embedded Systems

The largest applications (in terms of dollar volume) for digital signal processing are inexpensive, high-volume embedded systems, such as cellular telephones, disk drives (where DSPs are used for servo control), and modems. In these applications, cost and integration considerations are paramount. For portable, battery-powered products, power consumption is also critical. In these high-volume, embedded applications, performance and ease of development considerations are often given less weight, even though these applications usually involve development of custom software to run on the processor and custom hardware that interfaces with the processor. These products often must conform to published interface standards, such as the ITU-T V.90 modem standard.

Low-cost general-purpose processors and microcontrollers are very common in embedded applications. In some of these applications, such as modems and digital cellular telephones, it is common to find a microcontroller and a DSP processor working together, sometimes integrated in the same chip. Typically, the microcontroller handles overall control, user interface, and some top-level protocol processing, while the DSP handles the computationally intensive signal processing tasks. Many microcontroller vendors, recognizing the benefits of a single-processor solution, offer DSP-enhanced versions of their microcontroller architectures. These hybrid DSP/microcontroller architectures typically include some (or many) of the architectural features common among DSP processors.

High-Performance Applications

Another important class of applications involves processing large volumes of data with complex algorithms for specialized needs. This includes uses like sonar and seismic exploration, where production volumes are lower, algorithms more demanding, and product designs larger and more complex. As a result, designers favor processors with maximum performance, ease of use, and support for multiprocessor configurations. In some cases, rather than designing their own hardware and software from scratch, designers of these systems use off-the-shelf boards and ease their software development tasks by using existing software libraries.

High-performance floating-point DSP processors are common in these applications, as are high-performance general-purpose processors. Often, multiple processors are employed. It is common for a personal computer or workstation to be part of such systems, providing the user interface, access to mass storage, and other functions. Since personal computers and workstations are based on general-purpose processors, these processors have a foothold in such applications.

Personal Computer-Based Telecommunications and Multimedia

An important class of DSP applications is personal computer- and workstation-based telecommunications and multimedia functions. Increasingly, PCs are incorporating such capabilities as telephony, voice mail, data and facsimile modems, music and speech synthesis, and image compression. As with other high-volume, embedded applications, PC multimedia demands low cost and high integration. Unlike some other embedded applications, PC multimedia also demands high performance, since a multimedia PC may be called on to perform multiple functions simultaneously. Furthermore, the multitasking nature of such applications means that in addition to performing each function efficiently, the processor must have the ability to efficiently switch between functions. Memory capacity may also be an issue in these applications because many multimedia applications manipulate large amounts of data.

The first implementations of these integrated applications used separate DSP processors to handle real-time signal processing tasks. For example, in the late 1980s, NeXT workstations incorporated a Motorola DSP560xx processor. Later, some Apple Macintosh models incorporated the now obsolete Lucent Technologies DSP32xx. These approaches were innovative in that they replaced the multiple, fixed-function, ROM-programmed DSPs that would normally be found in a modem, a sound card, etc., with a single reprogrammable DSP.

In 1994, Intel announced an initiative called "Native Signal Processing," or "NSP," through which they planned to facilitate the implementation of real-time DSP functions on existing Intel processors in PCs. Intel's original NSP initiative was based entirely on software, including a version of Texas Instruments' SPOX real-time operating system and software libraries provided by Intel. This original software-only initiative had no significant impact on the marketplace, partly due to Microsoft's reluctance to allow SPOX to become a central component of the Windows PC software environment, but it did serve to raise awareness of the concept of using computer system host processors for signal processing. With Intel's addition of DSP hardware support to their processors via the MMX and SSE architecture extensions, products are beginning to emerge that employ NSP. In the meantime, other vendors of PC and workstation host processors have begun, with varying degrees of commitment, their own efforts in this direction. AMD, for example, offers its K6 processor, which is compatible with Intel's MMX extensions, and more recently added 3DNow! instructions to its processors. Other vendors have announced similar instruction set extensions; for example, Motorola offers the AltiVec instruction set extensions in its G4 PowerPC processor.

General-purpose processors are present in all PCs and workstations, and are among the highest-value components in these systems. Thus it is natural for manufacturers of these chips to take steps to defend their positions by minimizing the need for a second processor to implement telecommunications and media functions in computer systems.

4. Processor Architectures and Performance

Chapter 3, *Digital Signal Processing and DSP Systems*, described digital signal processing in general terms, focusing on DSP fundamentals, systems, and application areas. In this chapter we begin to examine specific characteristics of processors intended for use in DSP applications, starting with a high-level description of the features common to virtually all DSP processors. We then describe classes of architectures for DSP, including those used in dedicated DSP processors and those used in general-purpose processors. BDTI's textbook, *DSP Processor Fundamentals,* provides a more detailed treatment of DSP processor architectures and features.

Architectural Features for DSP

Most DSP applications require high performance in repetitive computation- and data-intensive tasks. The most important processor architecture features that support these kinds of tasks are introduced briefly here and summarized in Table 4.0-1.

Fast Multiply-Accumulate

The most often-cited feature of DSP processors is the ability to perform a *multiply-accumulate* operation (often called a *MAC*) in a single instruction cycle. The multiply-accumulate operation is useful in algorithms that use vector dot products, such as digital filters, correlation, and Fourier transforms. To achieve this functionality, DSP processors include one or more multipliers and accumulators integrated into the main arithmetic processing unit (called the *data path*) of the processor. In addition, to allow a series

Feature	Use
Fast Multiply-Accumulate	Most DSP algorithms, including filtering and transforms, are multiplication-intensive.
Multiple-Access Memory Architecture	Many data-intensive DSP operations can be accelerated by reading a program instruction and multiple data items during each instruction cycle.
Specialized Addressing Modes	Efficient handling of data arrays and other common data types in DSP applications.
Specialized Program Control	Efficient control of loops is important for many iterative DSP algorithms. Fast interrupt handling is important for applications with frequent I/O operations.
On-Chip Peripherals and Input/ Output Interfaces	On-chip peripherals, like analog-to-digital converters, allow for small, low-cost system designs. Similarly, I/O interfaces tailored for common peripherals allow simple interfaces to off-chip I/O devices.

TABLE 4.0-1. Basic features common to virtually all DSP processors.

of multiply-accumulate operations to proceed without the possibility of arithmetic overflow, DSP processors generally provide extra bits in their accumulator registers to accommodate growth of the accumulated result. These bits are often referred to as "guard bits." Multiply-accumulate features are discussed in detail in BDTI's textbook, *DSP Processor Fundamentals.*

Multiple-Access Memory Architecture

A second feature shared by most DSP processors is the ability to complete several accesses to memory in a single instruction cycle. This allows the processor to fetch an instruction while simultaneously fetching operands for a previously fetched instruction or storing the result of a previous instruction to memory. High bandwidth between the processor and memory is essential for good performance if repetitive data-intensive operations are required in an algorithm, as is common in many DSP applications.

In many DSPs, parallel memory accesses are subject to restrictions. Typically, all but one of the memory locations accessed must reside on-chip, and multiple memory accesses can take place only with certain instructions. To support simultaneous accesses of multiple memory locations, DSP processors provide multiple on-chip buses, multi-ported on-chip memories, and in some cases multiple independent memory banks. DSP processor memory structures are often quite distinct from those of general-purpose processors.

Specialized Addressing Modes

To allow arithmetic processing to proceed at maximum speed while accessing common DSP data structures, DSP processors incorporate dedicated address generation units. Once the appropriate addressing registers have been configured, the address generation units operate in parallel with the processor's data path, forming the addresses required for operand accesses in parallel with the execution of arithmetic instructions. Address generation units typically support a selection of addressing modes tailored to DSP applications. The most common of these is register-indirect addressing with post-increment, which is useful in algorithms where a repetitive computation is performed on a series of data stored sequentially in memory. Special addressing modes called *circular* or *modulo* addressing are often supported to simplify the use of data buffers. Some processors support *bit-reversed* addressing, which eases the task of implementing the fast Fourier transform (FFT) algorithms.

Specialized Execution Control

Because many DSP algorithms involve repetitive computations in small loops, most DSP processors provide special support for efficient looping. Often, a special loop or repeat instruction is provided that allows the programmer to implement a *for-next* loop without expending any instruction cycles for updating and testing the loop counter or for jumping back to the top of the loop.

Some DSP processors provide other execution control features to improve performance, such as fast context switching and low-latency/low-overhead interrupts for fast input/output data handling.

Peripherals and Input/Output Interfaces

To allow low-cost, high-performance input and output (I/O), most DSP processors incorporate one or more serial or parallel I/O interfaces, and specialized I/O handling mechanisms such as direct memory access (DMA). DSP processor peripheral interfaces are often designed to interface directly with common peripheral devices like analog-to-digital and digital-to-analog converters.

As integrated circuit manufacturing techniques have improved in terms of density and flexibility, DSP processor vendors have begun to include not just peripheral interfaces, but complete peripheral devices on-chip. Examples of this include chips designed for digital answering machine applications, several of which incorporate a digital-to-analog and analog-to-digital converter on-chip.

Classes of Processors for DSP

Most DSP processors include the features outlined in the previous sections, enabling them to perform well on DSP algorithms. These features can be implemented within different architectural styles, and as the demand for DSP-capable processors has grown, the variety of styles of DSP processor architectures has widened. In addition, there are a growing number of processors that while not, strictly speaking, "DSP processors" are nonetheless capable of strong DSP performance. In the following sections, we provide an overview of the classes of processors commonly used to implement DSP. These processors can be grouped as follows:

- Conventional DSP processors
- Enhanced conventional DSP processors
- VLIW processors
- Superscalar processors
- General-purpose processors
- Hybrid processors

Note that these processor classes are not all mutually exclusive; for example, general-purpose processors are often superscalar.

Conventional DSP Processors

The first commercially successful programmable DSP processors were introduced in the early 1980's. For over a decade, virtually all subsequent DSP processors were based on the same style of architecture as the earliest DSPs, albeit with higher instruction execution rates, more powerful execution units, and larger address spaces. We refer to processors with this type of architecture as *conventional DSP processors.*

Conventional DSP processors typically provide most of the features outlined in the previous sections. They typically use 16- to 24-bit fixed-point arithmetic or 32-bit floating-point arithmetic, and contain a single ALU, a single multiplier, and at least one shifter. These architectures generally allow two data memory accesses in a single instruction cycle while executing instructions from within a hardware loop. Instructions are executed at a rate of one instruction per instruction cycle. Conventional DSP architectures typically use very specialized DSP-oriented instructions, and allow multiple operations (such as a multiply-accumulate and data move) to be encoded in each one. Conventional DSP architectures typically have limited on-chip memory; common peripherals include serial ports, host ports, bit I/O, timers, and parallel ports.

Conventional DSP processors are designed with an emphasis on low cost, low power consumption, and low memory usage. Most DSP processors in widespread use today are based on conventional DSP architectures. Current examples of conventional DSP processors include the Analog Devices ADSP-218x, the Motorola DSP563xx, and the Texas Instruments TMS320C54xx, to name just a few.

Enhanced Conventional DSP Processors

Enhanced conventional DSP processors are fundamentally similar to conventional DSP processors, but have additional execution units and, in some cases, wider buses. Such features allow increased parallelism, since more operations can potentially be performed simultaneously. The wider buses facilitate higher on-chip memory bandwidth to provide the additional execution units with data and to allow wider instructions that encode more parallel operations.

Enhancements may also include specialized hardware or co-processors for accelerating performance on specific tasks. For example, Lucent Technologies' enhanced conventional processor, the DSP16xxx, includes hardware acceleration for Viterbi decoding. In some cases, enhancements are more subtle than adding an extra execution unit; for example, the DSP16xxx also includes specialized hardware support for specific applications—such as specialized shifting capabilities to support the extended-precision multiplication used in the enhanced full-rate GSM application.

Such enhancements enable execution speed improvements over conventional processors executing at the same clock rate. For example, Lucent's DSP16xxx at 120 MHz is able to execute many DSP tasks faster than its predecessor, the DSP16xx, at the same clock speed. This increased efficiency is achieved by virtue of the higher degree of parallelism in the DSP16xxx. Major enhancements in the DSP16xxx include a second multiplier, a three-input adder (separate from the ALU), an expanded register file, increased memory bandwidth, and instructions that allow more parallel operations to be specified.

Not surprisingly, in addition to the enhancements mentioned earlier, enhanced conventional DSP processors generally feature peripherals similar to those of conventional DSP processors, except that they often have additional features such as support for new communication protocols. In most other respects, these enhanced processors are identical to conventional DSP processors.

By making modest improvements over conventional DSP processors, enhanced conventional DSP processors can achieve higher performance while maintaining similar cost, power consumption, and memory usage.

In some cases, enhanced conventional DSP architectures are software compatible with their conventional predecessors—or at least very similar. For example, in some cases, assembly-language software written for a conventional processor can be reassembled and executed on a successor enhanced conventional processor without any changes. Although DSP16xx assembly-language software cannot be reassembled for the DSP16xxx, the two languages are so similar that a translation is relatively uncomplicated. Regardless of compatibility, existing software will require further optimization to make use of the additional capabilities of the enhanced conventional processor.

VLIW and Superscalar Processors

Traditionally, DSP processors have issued and executed one instruction per instruction cycle. A number of newer processors targeting DSP applications, however, are based on *multi-issue architectures*; that is, they are capable of issuing and executing more than one instruction per cycle. The advantages of multi-issue architectures include increased performance and better compilability. Potential disadvantages include increased power consumption and increased program memory requirements. The two styles of multi-issue architectures currently used in processors for DSP are *VLIW* (very long instruction word) and *superscalar.*

Unlike conventional and enhanced conventional DSP processors, which are specialized for DSP, DSP-specificity is not an inherent feature of VLIW and superscalar processors. In fact, superscalar architectures are used far more often by general-purpose processors (discussed shortly) than by processors specialized for DSP. Currently, there is only one commercially available DSP processor that is based on superscalar execution: LSI Logic's LSI401Z. Among general-purpose processors, the Compaq Alpha AXP 21264, the Intel Pentium processors, and the Motorola PowerPC family are all examples of superscalar processors.

VLIW architectures generally issue and execute several simple, RISC-like instructions in each instruction cycle. These instructions are concatenated to form a single macro-instruction; thus, the term "very long instruction word." In VLIW processors, the programmer (or code-generation tool) specifies, in the source program, which instructions will be executed in parallel[1]. Examples of VLIW processors include the Analog Devices

[1] Note that some vendors use the term *static superscalar* architecture to refer to what we term a VLIW architecture. The nomenclature in this area has not been standardized; we will adhere to the definitions presented here throughout this report.

ADSP-TS0xx, the StarCore SC140, and the Texas Instruments TMS320C62xx, TMS320C64xx, and TMS320C67xx.

Like VLIW architectures, superscalar architectures are capable of executing several instructions per instruction cycle. However, in superscalar architectures, the determination of which instructions will be executed in parallel takes place at run-time. Superscalar processors include specialized hardware that determines which instructions will be executed in parallel (a task referred to as *instruction scheduling*) based on availability of execution units and dependencies between instructions. Hence, in a superscalar architecture, it is the processor (rather than the programmer or compiler) that groups instructions for parallel execution.

Processors with VLIW and superscalar architectures typically have a relatively large number of independent execution units (e.g., ALUs and multipliers), and sufficient instruction decoders, buses, and register file or memory access ports to allow simultaneous issue and execution of multiple instructions. In addition, they must have sufficient memory bandwidth to support the transfer of multiple instructions and data words per cycle.

The most important advantage of VLIW and superscalar architectures is that, by executing multiple, simple instructions per instruction cycle, the processor can achieve high parallelism while retaining the benefits of using a simple instruction set: high clock speeds and ease of programming. Regularity of instructions benefits compilers as well as programmers, making these processors better compiler targets than conventional and enhanced conventional DSP processors. However, processors with simple instructions often require more instructions to perform a given task than processors that use complex, compound instructions. Thus, comparing the relative number of instructions executed per instruction cycle is unlikely to yield an accurate comparison of relative performance. In addition, VLIW processors often require more program memory than processors using complex instructions, because each task requires more instructions, and the instructions are often wider than those used in other architectures. The additional instruction width allows greater regularity in the instruction set and may also be used to include information about which execution unit will execute each of the parallel instructions.

Although most VLIW and superscalar processors use simple, RISC-like instructions, the Intel Pentium, a general-purpose processor, provides a counter-example. When the Pentium processor was introduced, binary compatibility with preceding architectural family members was essential, and these predecessors used very complicated instructions. While clock speeds could not easily be increased significantly in the legacy architecture, converting to a superscalar design allowed much of the software written for older processors to run at a significantly higher speed on the Pentium.

A somewhat similar approach was taken in the Texas Instruments TMS320C55xx. The TMS320C55xx maintains assembly source code compatibility with its popular predecessor, the TMS320C54xx, but increases performance by adding additional execution units and executing two instructions per cycle instead of one. The TMS320C55xx instruction set is a superset of that of the TMS320C54xx. Since the TMS320C54xx instruction set was not RISC-like, neither is that of the TMS320C55xx. Thus, the TMS320C55xx provides a second counter-example to the general rule that VLIW-based processors use simple, RISC-like instructions.

On virtually all architectures, it is necessary for the assembly language programmer or compiler writer to understand all of the capabilities and limitations of the architecture in order to generate optimized software and, in some cases, to avoid writing programs that produce erroneous results. In VLIW processors, software complexity is increased because of the need to manually schedule multiple instructions for parallel execution. The risk of writing erroneous programs is increased because of the need to juggle multiple instructions and execution units. In addition, assembly-language programmers and high-level language compilers must often restructure algorithms so that parallelism inherent in an algorithm is expressed as explicit parallelism that can take advantage of the VLIW architecture.

On superscalar architectures, it is easier for the programmer or compiler to generate programs that function correctly, because the responsibility for scheduling parallel instructions is shouldered by the processor. However, the process of generating *optimized* software and predicting software execution times can still be very complicated. As explained in Chapter 3, superscalar architectures introduce challenges for real-time DSP applications, which typically require a very high level of optimization and a very high level of execution-time predictability.

VLIW and superscalar architectures are generally designed with an emphasis on speed, and with less emphasis on cost, power consumption, and memory usage.

General-Purpose Processors

General-purpose processors are designed to efficiently perform control and protocol-oriented tasks, and thus have traditionally included very few of the DSP-oriented features described previously. Recently, however, general-purpose processors have begun to incorporate DSP-oriented features, and have assumed responsibility for performing DSP tasks in some applications. For this reason, we include a discussion of general-purpose processors here, although this report does not include this class of processor.

General-purpose processors serve in a vast range of products, from television remote controls to supercomputers. Not surprisingly, these processors have evolved in many different directions, creating an extremely diverse array of architectures and products. Today, general-purpose processors span four orders of magnitude in price, and a wide spectrum in performance as well.

For DSP tasks, the most important categorization of general-purpose processors divides processors intended for use as host processors in computers ("PC processors") and those intended for use in embedded applications. High-end PC processors are typified by the Intel Pentium III, the Motorola/IBM PowerPC 604/604e, and the Compaq Alpha AXP

21264. Embedded general-purpose processors are typified by IDT's R4650 and Hitachi's SH-2.

The following list describes the key attributes of PC general-purpose processors that distinguish them from DSP processors:

- **Instruction sets**. Instruction sets are general-purpose in nature. They are often simple and orthogonal, especially in RISC architectures. Where specialized, these instruction sets tend to be specialized in ways that do not address the needs of DSP applications. For example, the Intel Pentium architecture provides support for character string operations, which are not particularly useful for DSP. This distinction is becoming less clear, however, as many general-purpose processors have instruction-set extensions designed to address the needs of DSP and multimedia tasks.
- **Memory architectures**. General-purpose processors either rely on simple, Von Neumann memory architectures, or use on-chip caches with Harvard architectures. Most DSPs use Harvard architectures and multiple-access on-chip RAM, and DSPs usually do not use caches. DSPs also tend to have smaller address spaces.
- **High-level languages**. Software development for general-purpose processors is almost exclusively performed in high-level languages. In contrast, most performance-critical software for DSP processors is developed in assembly language. Software development tools for general-purpose processors reflect this difference; development tools are heavily focused on high-level language software development.
- **Performance**. PC general-purpose processors were originally designed to optimize performance and cost-performance on general computing applications, such as spreadsheets, word processors, and databases. DSP processors, in contrast, were optimized for signal processing and embedded applications. In recent years, this distinction has begun to blur, as general-purpose processor vendors are now adding DSP features to their processors, and DSP processors are adding control-oriented functionality.
- **Arithmetic**. PC general-purpose processors provide hardware support for both integer and floating-point arithmetic. Surprisingly, performance on floating-point arithmetic is sometimes better than performance on integer arithmetic. In contrast, typical DSP processors support either floating-point or fixed-point arithmetic, but do not support both well.
- **Compatibility**. PC general-purpose processors usually must be designed to run object code written for previous-generation processors, and to do so efficiently. This is necessary because of users' large investments in operating system and application software. In contrast, object-code compatibility has not been considered critical for DSP processors and for embedded general-purpose processors;

programmers often expect to invest significant effort rewriting, porting, or re-optimizing software for a new DSP processor.

- **Peripherals**. High-end PC general-purpose processors do not provide on-chip peripherals or peripheral interfaces. DSP processors often provide a variety of on-chip peripherals or peripheral interfaces to facilitate integration.
- **Dynamic execution**. High-end PC processors often rely on superscalar architectures where instructions are dynamically scheduled at run time. Caches, data-dependent instruction execution times, and branch prediction add additional layers of run-time decision-making to the processors' execution. In contrast, DSP processors have traditionally relied on simple, highly static execution models. As previously discussed, however, some high-end DSP processors have begun to incorporate some of the dynamic features more commonly found in PC processors, such as superscalar execution and data caches.

Embedded general-purpose processors can have similar architectures to PC processors (e.g., the R4650 is similar to other MIPS architecture processors), but typically aim for different objectives. Some of these differences are summarized below.

- **Cost and performance**. Embedded processors aim for a wider range of cost and performance targets than PC processors, since the needs of their target applications are more varied.
- **Power consumption**. Many embedded applications require low power consumption, typically to conserve battery life or to minimize the size of the power supply.
- **Peripherals**. In many cases, embedded general-purpose processors provide on-chip peripherals and peripheral interfaces, such as timers and serial ports.

Processors that begin life as PC processors sometimes evolve into embedded processors. The IDT R4650 is such an example, and Intel is promoting Pentium II processors for embedded applications.

While both DSPs and general-purpose processors have continuously achieved significant performance gains, it appears that in recent years the rate of advancement in general-purpose processor performance has outpaced that of DSPs, at least at the high end of performance. If this trend persists, it will raise serious questions about the future of dedicated DSP processors in many applications.

Hybrid Processors

Most DSP applications require a mixture of control-oriented processing and DSP. In many applications, these requirements are addressed by using a microcontroller for control-oriented software and a DSP processor for DSP algorithm software. In cases where the microcontroller or DSP processor is an established architecture, the dual-processor approach may offer the advantages of an existing software base and familiarity of architectures. However, the dual-processor approach also has important disadvantages, such as the complexity of multi-processor software development and the inefficiency of duplicated resources.

To avoid the need for two processors, some processor vendors attempt to combine microcontroller features with DSP features in a single chip. Some DSP processor vendors have added microcontroller functionality to some of their DSP processors; similarly, vendors of general-purpose processors have added DSP capabilities to some of their general-purpose processors. Current DSP processors with microcontroller features include Motorola's DSP568xx processor family and Texas Instruments' TMS320C27xx family. Both processors have modest DSP performance but include more microcontroller features than other DSP processors. Virtually all of the major microcontroller architectures offer some support for DSP, in the form of extra hardware or additional instructions. For example the IDT R4650 is an enhanced version of IDT's basic microcontroller architecture, and includes a multiply-accumulate instruction.

Other general-purpose processor vendors have performed more extensive renovations on existing architectures. Hitachi, with its SH-DSP and SH3-DSP processors, has added a complete DSP data path to the existing SH-2 and SH-3 microcontrollers. Intel, MIPS, and Motorola have all enhanced their general-purpose processor architectures with DSP and multimedia instruction-set extensions.

Yet another approach taken by some general-purpose processor vendors is to add a co-processor that helps with DSP tasks. For example, ARM offered the Piccolo DSP co-processor for use with its ARM7 core, and NEC has added a "media co-processor" to the V830 processor. (ARM is no longer actively marketing the Piccolo design.) Massana offers a DSP co-processor, the FILU-200, that can be used in conjunction with any general-purpose architecture.

In addition to the hybrid architectures that are retrofits of existing architectures, there are a number of entirely new architectures that include both DSP and microcontroller features. These architectures, which usually combine a RISC-based microcontroller with a substantial complement of DSP hardware and instructions, are free of the limitations of legacy architectures, but generally do not offer the benefits of retrofitting (such as software compatibility). Processors based on new hybrid architectures include Infineon's TriCore and Hyperstone's E1-16X and E1-32X processors.

Processor Embodiments

The most familiar form of processor is the single-chip processor, which is incorporated into a printed circuit board design by the system designer. However, with the widespread proliferation of processors in many new kinds of applications, the increasing levels of integration in all kinds of electronic products, and the development of new packaging techniques, processors can now be found in many different forms. In this section we briefly discuss some of the forms that processors take.

Multi-Chip Modules

Rather than packaging a single integrated circuit (IC) die in a ceramic or plastic package as is done with conventional ICs, multi-chip modules (MCMs) combine multiple, bare (i.e., unpackaged) dies into a single package. One advantage of this approach is higher packaging density—more circuits per square inch of printed circuit board. This, in turn, results in increased operating speed and reduced power dissipation. As MCM packaging technology has advanced in the past few years, vendors have begun to offer MCMs containing processors.

Multiple Processors on a Chip

As IC manufacturing technology becomes more sophisticated, processor designers can squeeze more features and performance into a single-chip processor, and they can consider combining multiple processors on a single IC. For example, both Motorola and Texas Instruments offer devices that combine a DSP and a microcontroller on a single chip, a natural combination for many applications. As with MCMs, multi-processor chips provide increased performance and reduced power compared with designs using multiple, separately packaged processors. However, the selection of multi-processor chip offerings is currently limited to only a few devices.

Chip Sets

While some manufacturers combine multiple processors on a single chip, and others use MCMs to combine multiple chips into one package, another variation on processor packaging is to divide the processor into two or more separate packages. This is the approach Sharp Microelectronics has taken with its Butterfly DSP chip set, which consists of the BDSP9320 address generator and the BDSP9124 processor. Dividing the processor into two or more packages may make sense if the processor is very complex and if the number of I/O pins is very large. Splitting functionality into multiple ICs may allow the use of much less expensive IC packages and may thereby provide cost savings. This approach also provides added flexibility, allowing the system designer to combine the individual ICs in the configuration best suited for the application. For example, with the Sharp Microelectronics chip set, multiple address generator chips can be used in conjunction with one processor chip. Finally, chip sets have the potential of providing more I/O pins than individual chips. In the case of the Sharp Microelectronics chip set, the use of separate address generator and processor chips allows the processor to have eight 24-bit external data buses, many more than provided by more common single-chip processors.

Processor Cores

An interesting approach for high-volume designs is the coupling of a processor with user-defined or user-selected circuitry on a single chip. This approach combines the benefits of an off-the-shelf processor (such as programmability, development tools, and software libraries) with the benefits of custom circuits (e.g., low production costs, small size, and low power consumption). In this section, we briefly describe one variant of this design style: core-based system-on-chip designs (core-based SoCs).

A processor core is a processor intended for use as a building block in creating a chip, as opposed to being packaged by itself as an off-the-shelf chip. A core-based SoC is a SoC that incorporates a processor core as one element of the overall chip. The core-based SoC approach allows the system designer to integrate a processor core, interface logic, peripherals, memory, and other custom elements onto a single integrated circuit. Figures 4.0-1 and 4.0-2 illustrate the core-based SoC concept.

Many vendors of DSP and microcontroller processors use the core-based approach to create versions of their standard processors targeted at application areas. In our discussion of core-based SoCs, we focus on the case where a chip user wishes to create a core-based SoC for a specific application, or where a semiconductor vendor wishes to obtain a core in order to create new chip products.

The most significant providers of DSP cores are DSP Group, Texas Instruments, STMicroelectronics, IBM Microelectronics, Clarkspur Design, and Infineon. General-purpose and microcontroller cores with DSP functionality are available from ARM, ARC, and Lexra, among others. The services that core vendors provide can be broadly divided into two categories. In the first category, the vendor of the core is also the provider of

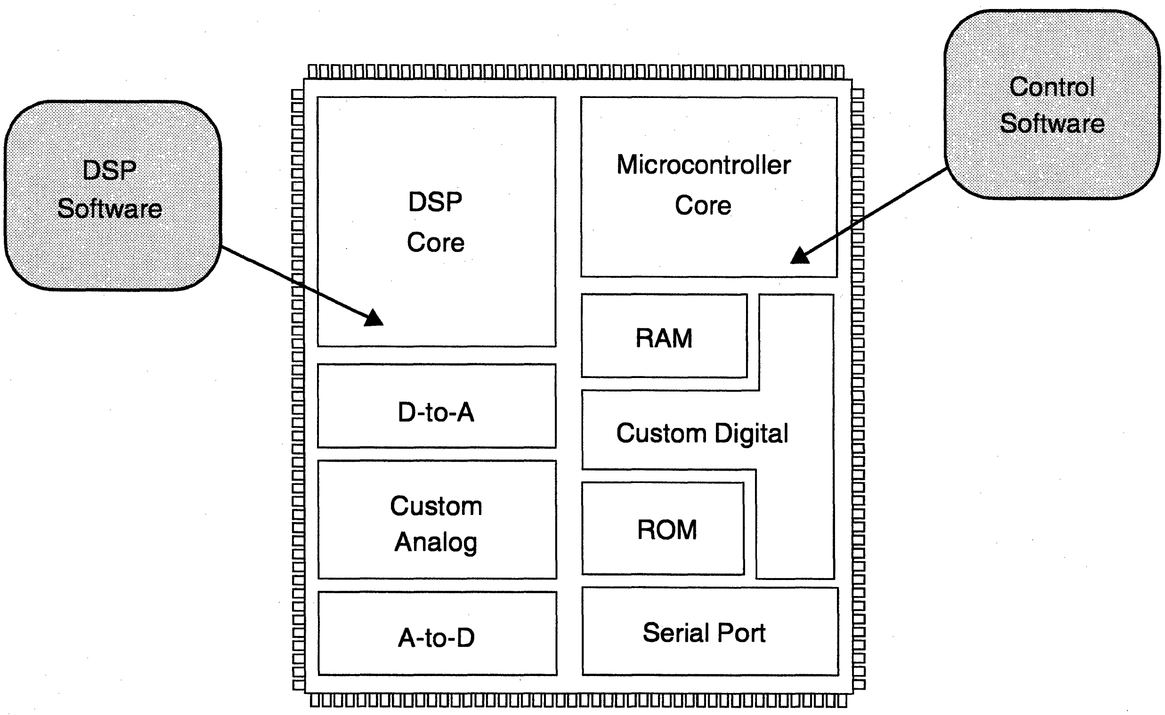


FIGURE 4.0-1. Core-based SoCs allow the integration of multiple processor types and analog circuitry, in addition to memories, peripheral interfaces, and custom digital circuitry.

foundry services used to fabricate the SoC containing the core; we refer to this category as *foundry-captive*. Texas Instruments and STMicroelectronics are providers of foundry-captive DSP cores, for example.

In the second category, the core vendor licenses the core design to the customer, who is then responsible for selecting an appropriate foundry. We call this category *licensable*. Companies such as DSP Group, Clarkspur Design, and Infineon offer licensable DSP cores. In exchange for a license fee and (in some cases) royalties, the customer receives a complete design description of the processor core. This core can then be fabricated as part of a SoC using the IC foundry of the customer's choice. Additionally, the customer may be able to modify the core processor if desired, since the complete design is provided.

Licensable cores are usually provided in the form of synthesizable VHDL or Verilog HDL design descriptions.

Note that vendors differ in their definitions of exactly what is included in a processor core. For example, some vendors' cores include not only the processor but memory and peripherals as well. Others include only the processor and no peripherals or memory.

Multiprocessors

No matter how fast and powerful processors become, the computational demands of a significant class of applications cannot be met by a single processor. Some of these applications may be well suited to custom integrated circuits. Or, if programmability is important, a multiprocessor based on commercial processors may be an effective solution.

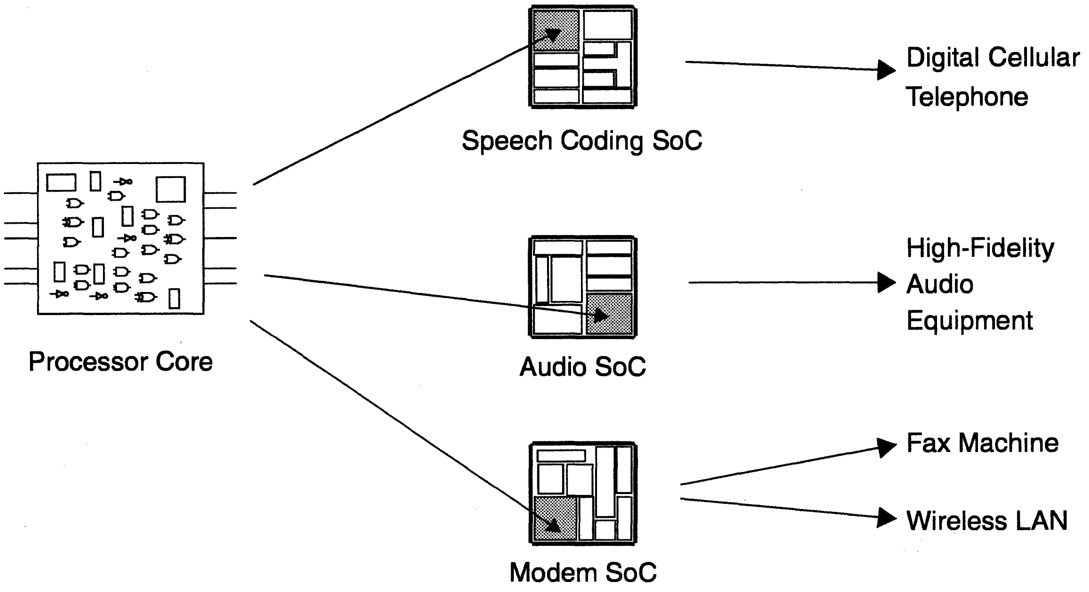


FIGURE 4.0-2. A processor core is intended to be used in SoCs customized for different applications or classes of applications.

Although any processor can be used in a multiprocessor design, some manufacturers have made special efforts to create processors that are especially well suited to multiprocessor systems. For example, the Analog Devices ADSP-2106x and ADSP-2116x provide extensive support for multiprocessor configurations.

Processor Boards

Often a processor is one component of a new printed circuit board design created for a particular product. The processor may be combined with other components, such as custom-designed chips. However, a custom printed circuit board is not required for every application. For low-volume products or for prototyping, one of the hundreds of commercial off-the-shelf processor boards may be attractive. Each of the popular processors is available in many board configurations. Such boards interface with many different host buses, such as VME, ISA, and PCI. In addition, many of these boards provide expansion buses to interface with custom boards.

Each of the embodiments just discussed is really just a different way of packaging and using processors. The architectural and performance analysis that comprises the later sections of this report focuses on single-chip, off-the-shelf DSP processors. For the most part, this analysis can be applied equally well to processors packaged in different forms. However, adjustments in the analysis may have to be made to account for customizations made in the architecture, and users will want to weigh certain processor characteristics differently depending on the form in which the processor is being used. For example, effective application engineering support is vital for users contemplating a core-based SoC design.

Custom Hardware

There are two important reasons why custom-developed hardware is sometimes a better choice than a processor-based implementation: performance and production cost. In virtually any application, custom hardware can be designed that provides better performance than a programmable processor. Custom hardware is also likely to be cost-effective because of its specialized nature. In applications with high sampling rates (for example, higher than 1/100th of the system clock rate), custom hardware may be the only reasonable approach.

For high-volume products, custom hardware may also be less expensive than a processor. A custom implementation places only those functions needed by the application in the hardware, whereas a processor requires every application to pay for the full functionality of the processor, even if a given application uses only a small subset of the processor's capabilities. Of course, developing custom hardware has some serious drawbacks. Most notable among these are the effort and expense associated with custom hardware development, especially for custom chip design.

Custom hardware can take many forms. It can be a simple, small printed circuit board using off-the-shelf components, or it can be a complex, multi-board system, incorporating custom integrated circuits. The aggressiveness (in terms of complexity and level of customization) of the design approach depends on the needs of the application. In the remainder of this section we very briefly mention some of the more popular approaches.

One of the most common approaches for custom hardware for DSP applications is to design custom printed circuit boards that incorporate a variety of off-the-shelf components. These components may include standard logic devices, fixed-function or configurable arithmetic units, field-programmable gate arrays (FPGAs), and function- or application-specific integrated circuits (FASICs). As their name implies, FASICs are chips that are designed to perform a specific function, perhaps for a single application. Examples of FASICs are digital filter chips, which can be configured to work in a range of applications; and facsimile modem chips, which are designed specifically to provide the signal processing functions for a fax modem and are not useful for anything else.

Many off-the-shelf FASICs sold by semiconductor vendors for DSP applications are really DSP processors containing custom, mask-programmed software in on-chip ROM. Some are based on proprietary processor architectures; perhaps the most prominent examples of the latter approach are Conexant's data and fax modem chips.

As tools and techniques for creating custom chips improve and more engineers become familiar with them, more companies are developing custom chips for their applications. Designing a custom chip provides the ultimate in specialization, since the chip can be tailored to the needs of the application, down to the level of a single logic gate or transistor.

Of course, the benefits of custom chips and other custom-hardware-based implementation approaches come with important trade-offs. Perhaps most importantly, the complexity and cost of developing custom hardware can be high, and the time required can be long. Also, if the hardware includes a custom programmable processor, new software development tools are needed to enable software development targeting the new processor.

It is important to point out that the implementation options discussed here are not mutually exclusive. In fact, it is quite common to combine many of these design approaches in a single system, choosing different techniques for different parts of the system. One such hybrid approach, core-based SoCs, was mentioned above. Others, such as the combination of an off-the-shelf DSP processor with custom ICs, FPGAs, and a general-purpose processor, are very common.

Performance Issues

System designers selecting a processor for use in a product face many choices. Although it is just one of many factors in processor selection, execution speed on DSP applications is an important concern and is often the first criteria used to narrow the field of contenders.

Gauging execution speed of a DSP processor is not as straightforward as it may sound. Typically, many DSP processor vendors quote their processors' instruction execution rates in millions of instructions per second (MIPS), or in millions of operations or millions of floating-point operations per second (MOPS or MFLOPS). General-purpose processor vendors often simply quote their processors' clock rates in MHz. Within a particular class of similar architectures, such as conventional DSPs, performance comparisons based on MIPS and MHz are moderately accurate. Unfortunately, when comparing processors with dissimilar architectures or instruction sets, these simplified metrics can be extremely misleading.

As we have seen in the previous sections, processor architectures have become more diverse in recent years. As a result, the amount of signal processing work accomplished by one instruction or one operation varies widely between different processor architectures, rendering the simplistic MIPS, MFLOPS, and MHz metrics virtually useless.

Some processor vendors quote the number of multiply-accumulate operations as the performance metric of interest. While the millions of multiply-accumulates per second (MMACS) metric might be a meaningful predictor of performance in some algorithms, a processor's DSP performance generally cannot be exclusively determined by its multiply-accumulate throughput; there is more to DSP algorithms than the multiply-accumulate operation. For example, MMACS does not include data moves and other operations.

To illustrate how misleading simplified performance metrics such as MIPS can be, consider the inner loop of a dot product implemented on two different processors: the Lucent Technologies DSP16xxx and the Texas Instruments TMS320C62xx.

Table 4.0-2 lists the differences in implementations of the inner loop of a dot product on the DSP164xx and the TMS320C62xx, together with the vendor-quoted speed ratings. If the MIPS ratings were used as indicators of the speed of the two processors, one might conclude that the TMS320C62xx would be 14 times faster than the DSP16xxx. In reality, however, as indicated in the table, both processors perform the inner loop at a rate of two vector elements per processor clock cycle, and only by virtue of its higher clock speed does the TMS320C62xx (at 300 MHz) perform the inner loop roughly twice as fast as the DSP164xx (at 170 MHz). This is significantly faster, but does not approach a speed difference of 14 times as suggested by the MIPS ratings of the two processors. Even this comparison is of limited use, however, since it is based on MAC throughput—which, as we discussed earlier, neglects important performance differences. In an effort to provide more meaningful comparisons of speed, BDTI has developed its own metric, the BDTImark2000, shown in the final row of Table 4.0-2 and described briefly below.

The BDTImark2000™

MIPS, FLOPS, and other simplistic measures are poor measures of processors' DSP performance, but have traditionally been the only measures readily available. In recognition of this fact, the authors of this report have proposed an alternative metric in an attempt to provide a more meaningful basis for comparing processors' DSP performance. This metric, the BDTImark2000™, combines the execution-time results from a suite of DSP algorithm benchmarks (such as FIR filters, FFTs, etc.) into a single number. The underlying DSP benchmarks were developed and implemented in assembly language and conform to a strict specification that governs their required functionality and allowable optimizations. Because the BDTImark2000 is based on actual DSP algorithms, it is a far more accurate measure than MIPS or MOPS or other simplified metrics. The BDTImark2000 is mainly useful for making quick comparisons of processor speed, however; serious system designers will want more detailed analysis to make their design choice. BDTImark2000 scores for a number of processors are posted on BDTI's website, *www.BDTI.com.*

	Lucent Technologies DSP164xx[a]	Texas Instruments TMS320C62xx	Ratio of Metrics
MIPS Rating According to Vendor	170 MIPS	2,400 MIPS	1:14.1
Processor Clock Speed	170 MHz	300 MHz	1:1.8
Number of Instructions in Inner Loop of Vector Dot Product	1 instruction	8 instructions	1:8
Number of Vector Elements Per Processor Clock Cycle	2	2	1:1
Number of Vector Elements Per Second	340 million	600 million	1:1.8
BDTImark2000 score at clock speeds shown above	810	1920	1:2.4

TABLE 4.0-2. Relative performance of the Lucent Technologies DSP164xx and the Texas Instruments TMS320C62xx.

[a] All information for the DSP164xx is for one of two on-chip cores. The DSP164xx is used for this analysis rather than the DSP16xxx, since the DSP164xx supports some instructions not supported by other DSP16xxx family members.

5. Processors Not Covered in This Report

The next chapter contains analyses of the most popular DSP processors currently available. However, there are many more processors of potential interest for DSP applications than can be included in this report. These processors include DSP cores, certain older or more specialized DSPs, and general-purpose processors (many with DSP enhancements).

Brief overviews of many of these other processors are available at BDTI's website at *www.BDTI.com.* Below we list the processors that are currently included on the website. As information becomes available on new processors, we will also publish brief overviews of them on the website.

Detailed analyses of some of the processors listed below are available in published reports from BDTI or by separate arrangement with BDTI. Contact BDTI for details.

General-Purpose Processors and Hybrids

Over the last few years, there has been expanded interest in performing DSP tasks on general-purpose processors. Some high-performance general-purpose processors, such as the Motorola/IBM PowerPC 604e and Intel Pentium, achieve impressive DSP performance despite their lack of DSP-oriented features. Other general-purpose processors, such as the Hitachi SH-DSP and Intel Pentium III, have been significantly enhanced to boost their DSP capabilities. As a result, many general-purpose processors are now capable of strong DSP performance. However, implementing DSP applications on general-purpose processors presents some unique challenges.

Brief overviews of some general-purpose processors are available at BDTI's website.

DSP Cores

DSP cores are becoming increasingly important as improved IC fabrication technology and the need for smaller, less expensive, and more energy-efficient products expands. Brief overviews of the following licensable and foundry-captive DSP cores are available on BDTI's website:

- Clarkspur Design CD2400 (licensable)
- Clarkspur Design CD245x (licensable)
- DSP Group OakDSPCore (licensable)
- DSP Group PineDSPCore (licensable)
- Infineon Carmel Core
- SGS-Thomson D950 CORE
- Texas Instruments T320C2xLP Core
- 3Soft M320C25 (licensable)

- 3Soft M320C50 (licensable)

We plan to publish reports focusing on DSP cores in the future. Contact BDTI for information about forthcoming reports.

A few of the DSP processors analyzed in this report are also available as cores. Most Texas Instruments cores are available for use in customer designed SoCs fabricated by Texas Instruments.

Other DSP Processors

Several DSP processors and processor families that were covered in previous editions of *Buyer's Guide to DSP Processors* have been omitted from this edition due to space constraints. We have omitted the following older or more specialized DSPs:

- Analog Devices ADSP-21cspxx
- Analog Devices ADSP-21020
- LSI Logic LSI40xZ
- Lucent Technologies DSP16xx
- Lucent Technologies DSP32C
- Lucent Technologies DSP32xx
- Motorola DSP560xx
- Motorola DSP561xx
- Motorola DSP566xx
- NEC µPD7701x
- Texas Instruments TMS320C1x
- Texas Instruments TMS320C2x
- Texas Instruments TMS320C27xx
- Texas Instruments TMS320C4x
- Texas Instruments TMS320C8x
- Zoran ZR38xxx

Brief overviews of these processors are available at BDTI's website. More detailed analyses of these processors may be obtained by contacting BDTI.

6. Choosing a Processor

In this chapter, we present our philosophy and methodology for choosing a processor for a particular application. After discussing the approach in general, we present profiling data illustrating the importance of different kinds of functions and instructions in a variety of applications.

The "Best" Processor

There is no single "best" DSP processor for all applications; the best processor depends completely on the specifics of the application at hand. Even within a narrowly defined application area like data modems, one can't identify a single best choice for all designs. One modem design team may be most concerned with time-to-market, in which case quality development tools, extensive application engineering support, and the availability of software libraries become dominant considerations. Another modem design team (or even the same one six months later) may place the priority on low production cost, low power consumption, high integration, or other criteria.

Because the selection criteria for a DSP processor depend so much on the specific needs of the product being designed, we don't attempt in this report to rank processors or provide recommendations on which processor is best for a given application. Instead, we present in a consistent fashion as much objective and subjective information as possible about each processor, so that once one determines which criteria are important for a design, one can quickly and accurately ascertain which processor or processors best meet those criteria.

Selection Methodology

In broad outline, here is our suggested approach for selecting a DSP processor:

- Identify those features, capabilities, and performance aspects that may be important for your application. This will almost certainly require some initial system design work to determine the kinds of algorithms that will be used (and therefore the kinds of computation required of the DSP processor); the types of devices the processor will be interfacing with; and the cost, power, and printed circuit board area that can be allocated to the DSP processor and associated components like memory chips. Also consider issues such as development tools, technical support, and documentation.
- Assign weights to each feature, capability, or performance aspect to reflect its relative importance for your application. Researching previous designs with similar priorities may help.
- Scrutinize the evaluations in Chapters 7 and 8 that relate to the processor aspects you have identified as important. Using the weights you have assigned, you can then begin to form a picture of which processors may be a good match for your design.

- Once you have narrowed the field to a few candidates, we recommend investigating those processors and their manufacturers in greater detail than is provided in this report. Each processor vendor provides user's manuals that explain many important details not covered here. You will probably want to meet with the vendors' applications engineers so that you can get their ideas about how their processor can be used to meet the needs of your application, and so that you can evaluate the quality and availability of each vendor's support. You may want to try calling the vendors' telephone support lines with technical questions.
- Depending on your application, you may also want to investigate the availability of third-party development tools, software libraries, and hardware. You will certainly want to gather updated speed and pricing information, based on quantities that are appropriate for your application.

Performance Measures

It is particularly important to recognize that summary performance measures such as benchmark results tell only part of the story of a processor's abilities, and can be very misleading. Before using such ratings to select a processor, you should understand what is being measured, how the measurements are performed, and how the resulting rating relates to the needs of your design. The fact that a given processor can perform a convolutional encoding twice as fast as the nearest competitor is likely to be irrelevant if your application consists mostly of filtering algorithms. Likewise, even if your application makes heavy use of filtering operations, other aspects of processor performance that are seldom benchmarked, such as input/output handling, can be equally important. In Chapter 8 we discuss the pitfalls of processor speed and power ratings and provide recommendations on interpreting these results in more detail.

To assist you in understanding the performance measures that may be meaningful for your application, we have included example application profiling data in Sections 6.2 and 6.3.

6.1 Application Profiling

Profiling refers to a set of techniques used to measure or estimate the amount of time an application program spends executing its various subsections, or the frequency with which different instructions or operations are executed as an application runs.

While profiling is primarily used by application programmers to help them optimize their software, profiling data is also useful to understand the demands made by an application on a processor and to help relate benchmark performance to application performance. In this section we briefly introduce several profiling approaches, and then examine profiling data from a variety of example applications implemented on a range of DSPs.

Profiling Approaches

We divide profiling approaches into four main categories:

- **Function-level profiling** analyzes the percentage of execution time that an application program spends in each of its major sub-functions during a period of operation. For example, for a voiceband data modem, profiling data might show the percentage of time spent in the decoder, echo cancellation, carrier recovery, clock recovery, and other major functions.

  DSP application developers most commonly use function-level profiling. After determining which sub-functions account for the largest portions of an application's execution time, application developers can focus their execution-speed optimization efforts on these sub-functions.

  Some DSP processor vendors provide instruction-set simulators or emulators that can be used to collect profiling data. For example, Texas Instruments provides profiling support in their in-circuit emulators for certain DSPs. Typically, the programmer divides the application into sub-functions by specifying ranges of program memory containing each sub-function. As the program executes, the simulator or emulator collects information about how much time is spent in each sub-function. If a processor does not have profiling support via its simulator or emulator, the application programmer may still be able to collect profiling data, for example, by saving a trace of all program addresses accessed or by setting up a series of breakpoints with associated automated scripts. This, however, can be awkward and time consuming.

  Later in this chapter we present several examples of function-level profiling of common DSP applications.

- **Instruction-level profiling** analyzes the relative frequency of execution of each instruction or type of instruction during a period of normal operation of the application; for example, the relative number of times MAC instructions were executed, the number of times bit manipulation instructions were executed, etc.

By determining which kinds of instructions or combinations of instructions are most commonly executed in applications, instruction-level profiling data can help processor designers determine where to focus their efforts to improve the processor's performance. Developers and users of high-level language compilers and other code-generation tools can use instruction-level profiling data to help understand how well these tools are making use of the processor's capabilities.

For application developers, instruction-level profiling data is most useful as an aid in understanding the demands a particular application makes on a particular processor; it is less useful for optimizing software.

Later in this chapter we present several examples of instruction-level profiling of common DSP applications.

- **Algorithm analysis** is another approach that can help in predicting processor performance in a particular application and in interpreting benchmark results. Algorithm analysis reveals the relative frequency of execution of basic algorithmic operations or sub-functions in an application, without regard to implementation on a particular processor. For example, such profiling might reveal that a particular audio coding algorithm requires execution of a bank of eight 16-tap FIR filters on a data stream with a sample rate of 16 kHz.

  Algorithm-level profiling data is usually developed through manual analysis of the algorithms involved in a given application. This data can be useful in estimating whether an application can achieve the necessary speed executing on a DSP processor, how fast a processor may be required, and what kinds of processor operations are likely to have the greatest impact on the execution speed of an application.

- **Power consumption profiling** focuses on power consumption instead of execution time. Low power consumption is a critical design goal in many applications. Power consumption profiling measures the power consumed by a processor as it executes an application and determines how much power is consumed by various sub-functions of the application. These sub-functions can then be targeted for power-reduction optimizations.

There are other types of profiling data as well. For example, in some applications it is useful to collect statistics on the frequency of accesses to different areas of program or data memory, or on the maximum time taken to respond to a particular type of interrupt.

Limitations of Profiling Techniques

Although profiling is an important tool in relating benchmark performance to application performance, it has a few serious shortcomings. The most important of these are:

- **Processor choice**. Since different processors perform differently on the same type of function, profiling results can vary significantly depending upon the processor used. (Algorithm analysis is not subject to this limitation, since it analyzes algorithms independent of processor choice.)

- **Optimization**. Application programmers tend to optimize their programs with specific performance goals in mind. The profile of an application varies depending on the optimization approach taken by the programmer and on the success of this approach. For example, developers normally focus on optimizing those sub-functions that account for the largest share of execution time. This has the effect of reducing the execution time of these sub-functions relative to others. In addition, it is often possible to substitute one algorithm for another to take advantage of the capabilities of a particular processor. For example, an FIR filter can be implemented using FFTs.
- **Implementation techniques**. The profile of an application also varies depending upon the implementation techniques used. For example, an implementation of a given application written in C typically results in a very different profile from an assembly language implementation of the same application, since the C compiler generates more efficient code for some kinds of operations than for others. Similarly, application profiles often change significantly between fixed-point and floating-point implementations, because some (but not all) functions require more cycles on a fixed-point processor than on a floating-point processor.

While each type of profiling data has its limitations, these types of data can be very instructive when viewed with their limitations in mind. Therefore, in the sections that follow we present a variety of function- and instruction-level application profiling examples.

Benchmarks and Profiling Data

Application profiling data can be useful in helping to relate DSP processor benchmark performance to application performance. For reasons that will be explored in Chapter 8, our DSP processor benchmarks, like most, are based on small- or medium-sized building-block functions, such as FIR filters and FFTs. Application profiling data can help the DSP processor user to understand which of these benchmark functions are important to the application, and their relative importance. This is our main motivation for presenting examples of application profiling data in this chapter.

Both function-level and algorithm-level profiling data can be used to relate DSP processor benchmark performance to application performance. Instruction-level profiling data is generally not useful in this context, though it has other important uses. Since function-level profiling data is most readily available, we focus on this type of profiling data in the remainder of this chapter. We also provide instruction-level profiling examples.

Application Profiling Examples

With the help of several processor vendors, third-party application software developers, and researchers, we have gathered and analyzed profiling data on several example applications. In the following sections we present examples of both function-level and instruction-level profiling data. The source of the profiling data is noted in each example. For information on contacting the organizations mentioned, please see Appendix A, *Vendor Contact Information.*

Our intention in presenting this profiling data and analysis is to illustrate the use of profiling and the characteristics of particular implementations of the example applications. For the reasons outlined above, it is important to recognize that the data and analysis presented here is not definitive and cannot necessarily be used to draw broad conclusions about applications and processors.

6.2 Function-Level Profiling

In this section we present function-level profiling examples for several DSP processor applications: two V.32bis data modem implementations, a V.17 fax modem, a G.728 speech encoder, two CELP speech encoders, a video coder, an adaptive beamforming algorithm, and three GSM speech encoder and decoder implementations. The function-level profiling data shows how much time the DSP processor spends in various activities for a particular implementation of each application.

DSP systems often have multiple modes of operation. For example, a modem has an idle mode where it waits for commands, a handshake mode where it establishes a connection, a training mode where it adapts to the conditions of the current connection, and a communication mode where data is transferred. Obviously, the activities of the DSP processor vary significantly depending upon the mode of operation. The profiling data presented here is for the main steady-state operating mode of each application.

For each example, we list the eight to ten most important functions in terms of relative processor execution time, indicating the percentage of processor execution time accounted for by each function. We provide a brief description of each function and an indication of the types of basic DSP processor operations that are typically found in each function. In some cases, these operations are identical or very similar to the functions found in the BDTI Benchmarks™, discussed in Chapter 8. In other cases, the types of DSP processor operations used are very complex or are unique to the application. In these cases, we either omit the listing of DSP processor operations, or we list familiar DSP processor operations that are similar to those used in the function.

More detailed technical information on each of these applications can be found in texts and journals on communications and signal processing, a sampling of which are listed in the *References* section at the end of this report.

V.32bis Modem

V.32bis refers to a 14,400 bit/second modem protocol that is widely used for asynchronous data modems. V.32bis data modems are capable of communicating at slower speeds using older protocols ("fallback modes"). The profiling data presented here is for full 14,400 bit/second operation.

Like most modems, the V.32bis modem transmits and receives simultaneously. Therefore, the profiling data reflects operation of both the transmitter and receiver, even though these are distinct operations within the DSP processor.

Table 6.2-1 and Figure 6.2-1 give function-level profiling data for one V.32bis modem implementation. The modem was implemented on a Texas Instruments TMS320C3x floating-point processor. The implementation approach began with coding the entire modem in the C language and then replacing critical sections of C code with carefully optimized assembly language. This application profiling data was provided by DSP Software Engineering, Inc., a provider of DSP software used in telecommunications and multimedia applications such as video conferencing, wireless communications, and ISDN communications. DSP Software Engineering's software development business was recently acquired by Tellabs.

Table 6.2-2 and Figure 6.2-2 give function-level profiling data for another V.32bis modem implementation. This modem was implemented on a fixed-point Texas Instruments TMS320C5x in assembly language. This application profiling data was provided by ILLICO, a design and consulting firm specializing in the application of DSP to telecommunication product development, especially modems.

For information on contacting ILLICO, please see Appendix A, *Vendor Contact Information.*

Table 6.2-1. Profiling Data for:

Application: V.32bis Modem
Language: C and Assembly
Processor: Texas Instruments TMS320C3x
Data From: DSP Software Engineering

Abbreviation	Function	Related DSP Processor Operations	Percent of Time in Function
Decoder	Decision and symbol/Viterbi decoder	Vector subtract, branch and control	27.65
Echo	Echo canceller processing	Complex LMS filter	16.26
Carrier recovery	Equalize, jitter predict, demod., and carrier recovery	IIR filter, voltage-controlled oscillator	15.82
Symbol clock	Symbol clock recovery	IIR filter, voltage-controlled oscillator	11.78
Shaper	Multirate pulse shaper	FIR filter	8.63
Byte pack	Convert modem symbols to/from bytes; synchronize	Bit manipulation, branch and control	8.30
Multi filter	Front-end multirate filter	FIR filter	3.76
Near blk. update	Echo canceller near-end block update	LMS Filter	2.77
Far blk. update	Echo canceller far-end block update	LMS Filter	2.77
Encoder	(Symbol or trellis) encoder, modulate	Convolutional encoder	0.83
Gain control	Automatic gain control	Vector square	0.66
Other	Other		0.77

Figure 6.2-1. Profiling Data for V.32bis Modem on TMS320C3x

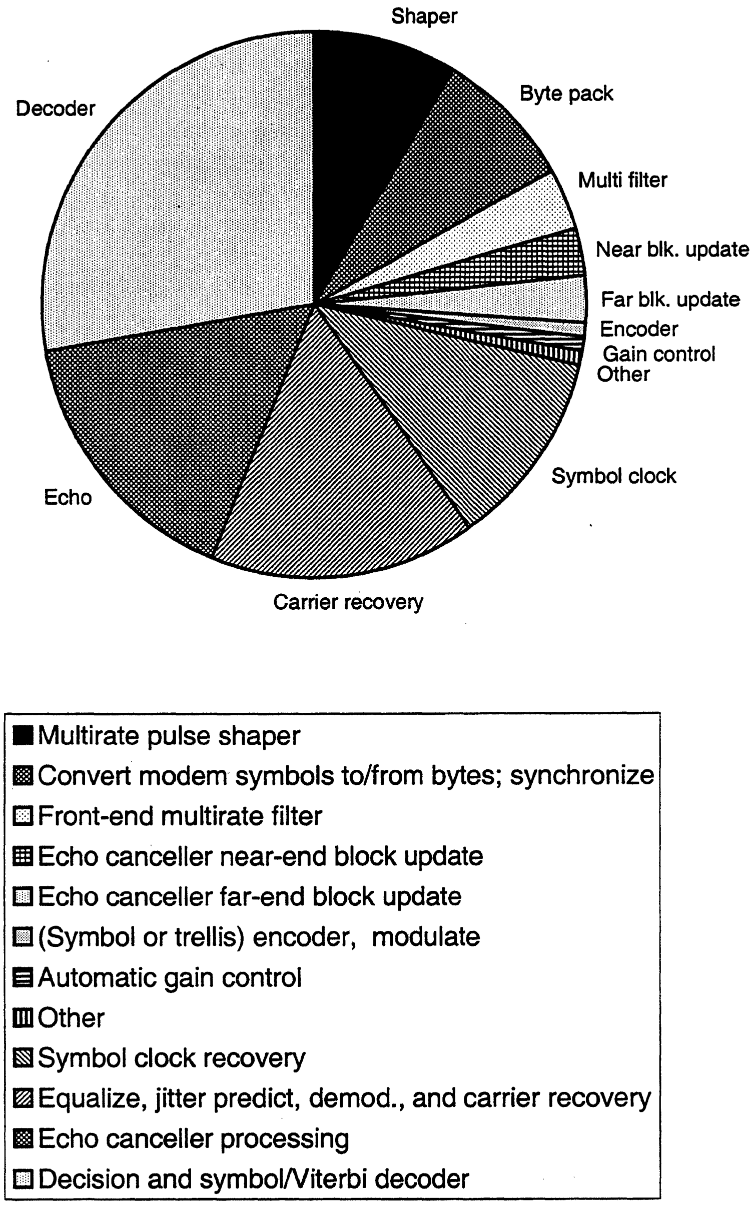
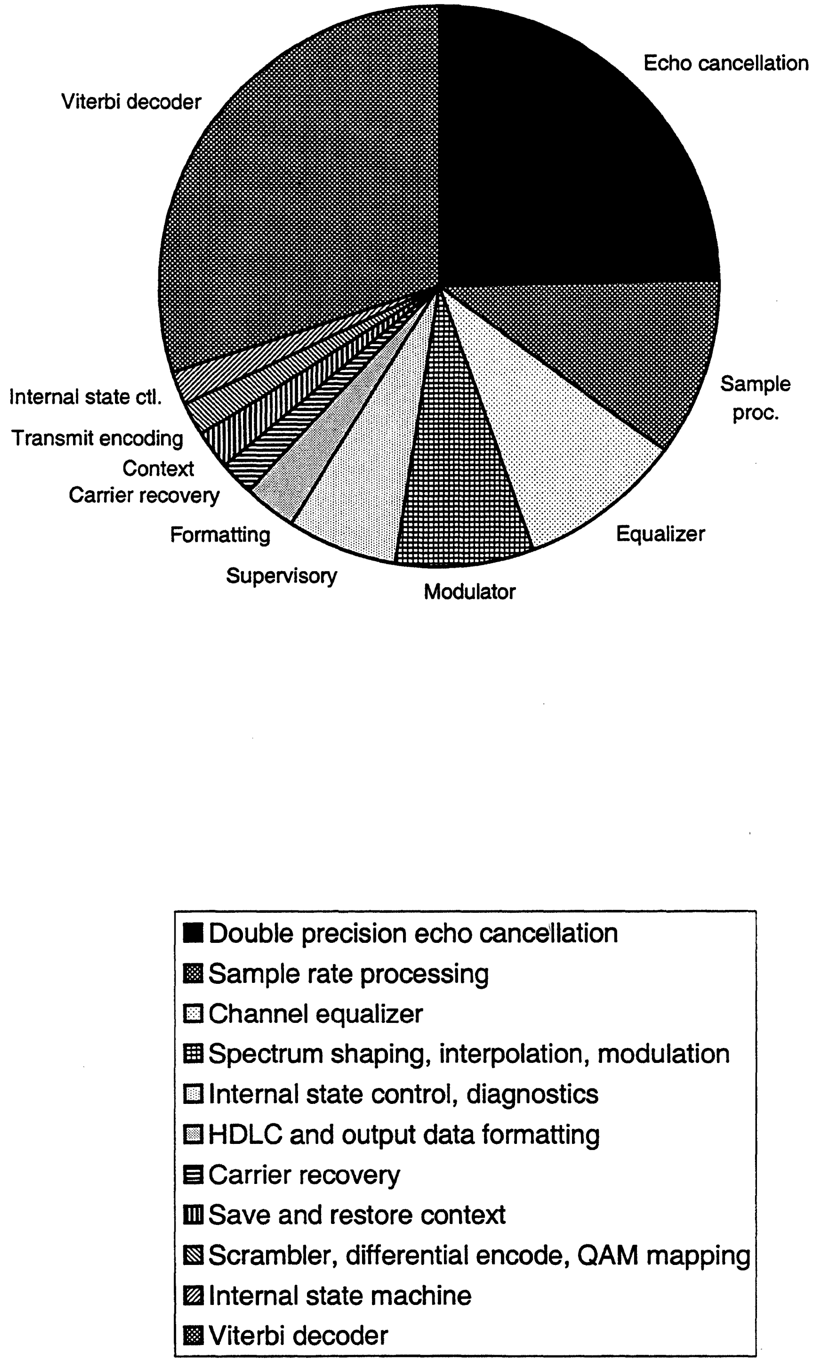


Table 6.2-2. Profiling Data for:

Application: V.32bis Modem
Language: Assembly
Processor: Texas Instruments TMS320C5x
Data From: ILLICO

Abbreviation	Function	Related DSP Processor Operations	Percent of Time in Function
Viterbi decoder	Viterbi decoder	Viterbi decoder	29.00
Echo cancellation	Double precision echo cancellation	Adaptive real LMS FIR filter	24.00
Sample proc.	Sample rate processing	Hilbert transform, IIR filter, digital PLL, interpolating FIR filter	10.00
Equalizer	Channel equalizer	Adaptive complex LMS FIR filter	9.00
Modulator	Spectrum shaping, interpolation, modulation	FIR filtering	8.00
Supervisory	Internal state control, diagnostics	Decision-making control	6.00
Formatting	HDLC and output data formatting	Bit manipulation	3.00
Carrier recov.	Carrier recovery	Digital PLL, FIR, IIR	2.00
Context	Save and restore context	Memory-memory data moves	2.00
Transmit encoding	Scrambler, differential encode, QAM mapping	Bit manipulation	2.00
Transmit State Ctl.	Internal state machine	Program control	2.00

Figure 6.2-2. Profiling Data for V.32bis Modem on TMS320C5x



V.17 Modem Receiver

V.17 refers to a 14,400 bit/second modem protocol that is standard for Group III facsimile transmission. V.17 fax modems are also capable of communicating at slower speeds using older protocols ("fallback modes"). The profiling data presented here is for full-speed operation.

The modem was implemented on a Motorola DSP560xx processor in assembly language. The profiling data reflects operation of the modem receiver only. Since it is used for facsimile, a V.17 modem operates in half-duplex mode; that is, it is either transmitting or receiving at any given time, but not both. As with most modem types, the receiver is significantly more complex than the transmitter. Table 6.2-3 and Figure 6.2-3 give function-level profiling data for this V.17 modem receiver implementation.

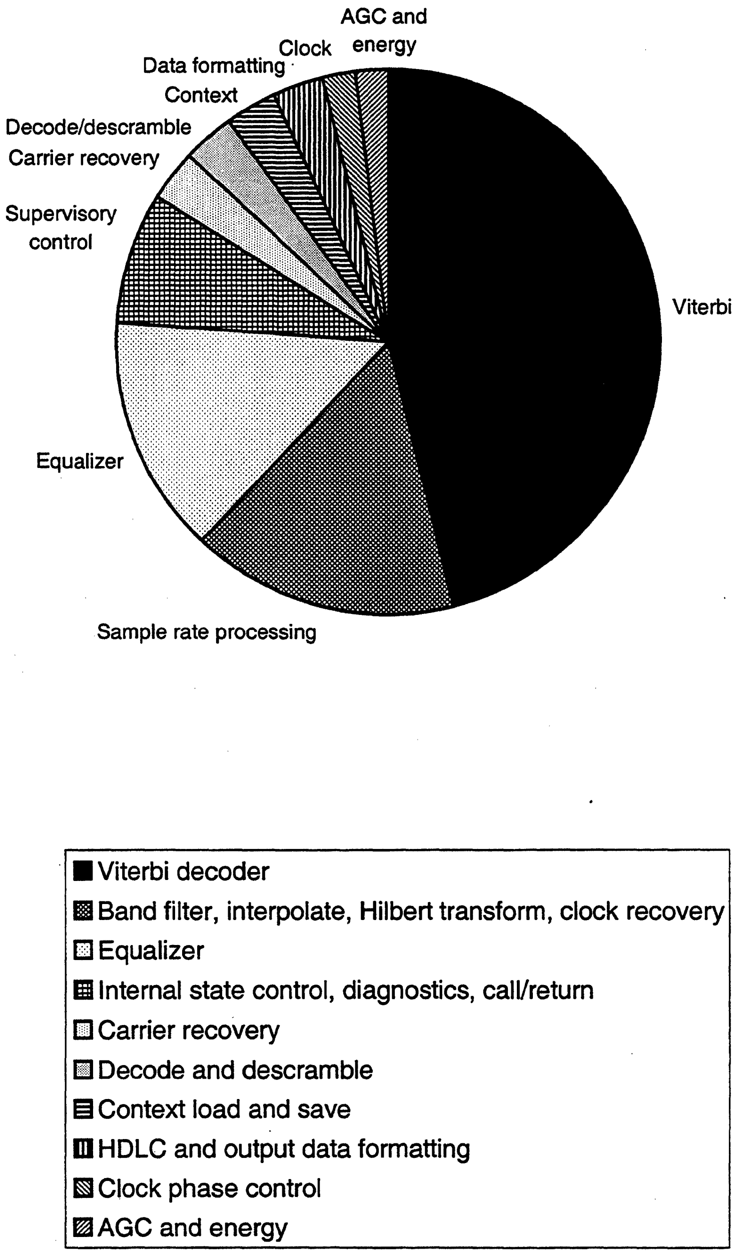
This application profiling data was provided by ILLICO, a design and consulting firm specializing in the application of DSP to telecommunication product development, especially modems. For more information on ILLICO, please see Appendix A, *Vendor Contact Information.*

Table 6.2-3. Profiling Data for:

Application: V.17 Modem Receiver (for Group III Facsimile)
Language: Assembly
Processor: Motorola DSP560xx
Data From: ILLICO

Abbreviation	Function	Related DSP Processor Operations	Percent of Time in Function
Viterbi decoder	Viterbi decoder	Viterbi decoder	46.0
Sample rate processing	Band filter, interpolate, Hilbert transform, clock recovery	Hilbert transform, IIR filter, digital PLL, interpolating FIR filter	16.0
Equalizer	Equalizer	Adaptive complex LMS FIR filter	14.0
Supervisory control	Internal state control, diagnostics, call/return	Decision-making control	8.0
Carrier recovery	Carrier recovery	Digital PLL, FIR, IIR	3.0
Decode/descramble	Decode and descramble	Bit manipulation, differential encoding	3.0
Context	Context load and save	Memory-memory data moves	3.0
Data formatting	HDLC and output data formatting	Bit manipulation	3.0
Clock	Clock phase control		2.0
AGC and energy	AGC and energy	Sum of squares	2.0

Figure 6.2-3. Profiling Data for V.17 Modem Receiver on DSP560xx



G.728 Speech Encoder

G.728 refers to a standard format for the encoding and compression of digital speech signals. This standard defines a low-delay CELP speech encoding scheme. CELP stands for *code excited linear predictor* and refers to a technique for the compression of digital speech signals based on creating a parameterized model of the human vocal tract and transmitting the parameters of the model in place of the speech signal itself. CELP is used in applications where speech signals need to be transmitted digitally using low bit rates, such as secure telephones. Low-delay CELP is a variation of CELP speech coding that has a shorter intrinsic time delay than other CELP variations. The G.728 encoding scheme has been rigorously tested and found to have comparable voice transmission quality to conventional techniques that use higher bit rates, such as G.726, which uses ADPCM (adaptive differential pulse code modulation). One current application of G.728 is compressing voice signals for video conferencing. Other variations of CELP are used in other applications.

This G.728 speech encoder was implemented on a Texas Instruments TMS320C3x processor by DSP Software Engineering. The implementation was done completely in assembly language. As with many speech coding techniques, the encoder (sometimes called the *analyzer*) is more computationally demanding than the decoder (sometimes called the *synthesizer*). The encoder accounts for about twice the computational load of the decoder.

Table 6.2-4 and Figure 6.2-4 give function-level profiling data for this G.728 encoder implementation.

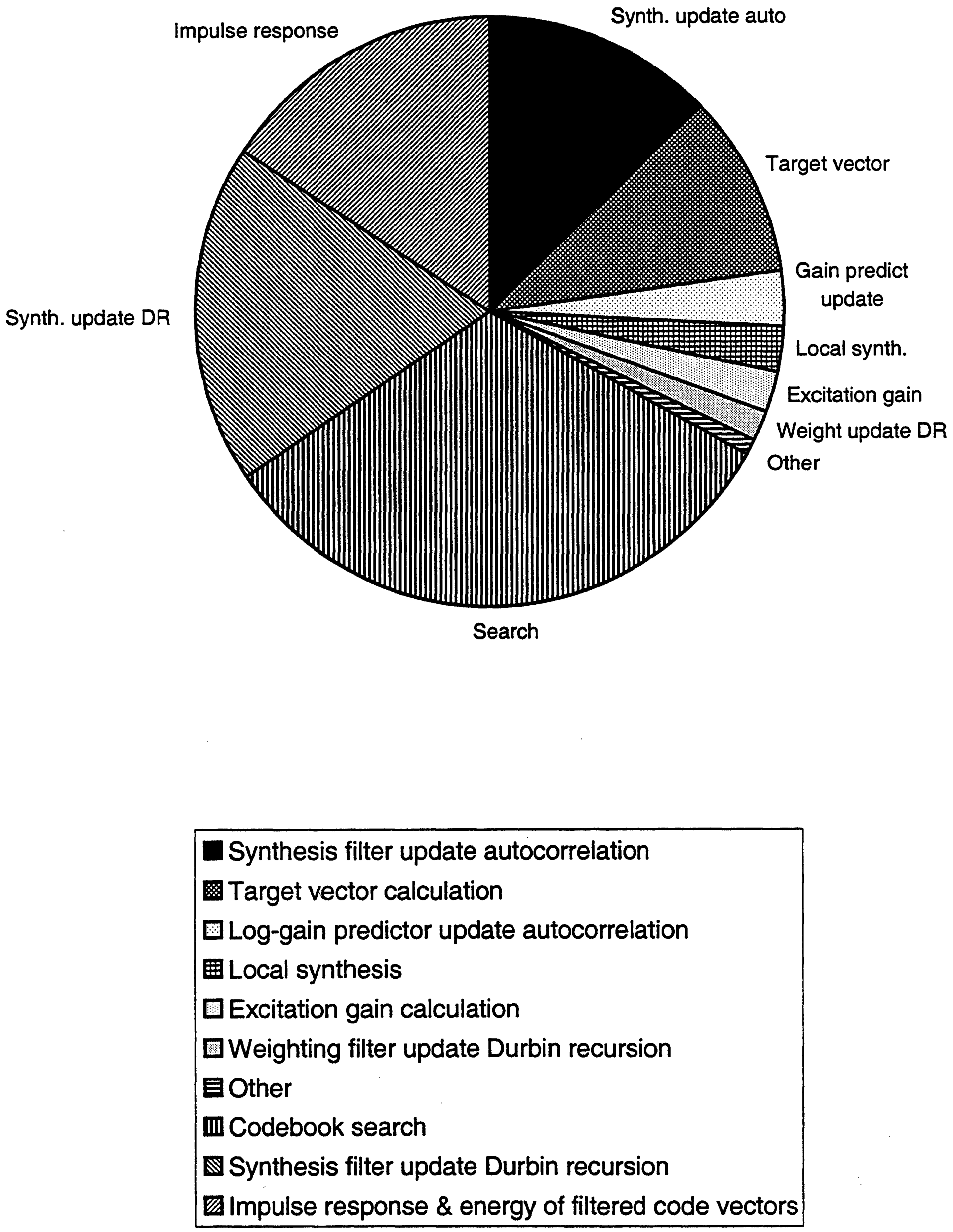
In the next example, we show profiling results for a G.728 encoder and decoder implemented on the Analog Devices ADSP-2171.

Table 6.2-4. Profiling Data for:

Application: G.728 Encoder
Language: Assembly
Processor: Texas Instruments TMS320C3x
Data from: DSP Software Engineering

Abbreviation	Function	Related DSP Processor Operations	Percent of Time in Function
Search	Codebook search	FIR filter	32.48
Synth. update DR	Synthesis filter update Durbin recursion	50th order Durbin recursion	18.52
Impulse response	Impulse response & energy of filtered code vectors	Convolution	15.88
Synth. update auto	Synthesis filter update autocorrelation	Autocorrelation, specialized windowing	12.86
Target vector	Target vector calculation	IIR filter, vector subtraction	9.80
Gain predict update	Log-gain predictor update autocorrelation	Autocorrelation	3.14
Local synth.	Local synthesis	IIR filter	2.66
Excitation gain	Excitation gain calculation	Vector squaring, autocorrelation, specialized windowing	2.11
Weight update DR	Weighting filter update Durbin recursion	10th order Durbin recursion	1.64
Other	Other		0.91

Figure 6.2-4. Profiling Data For G.728 Encoder on TMS320C3x



G.728 Speech Encoder and Decoder

This profiling example is for a G.728 encoder and decoder implemented on the Analog Devices ADSP-2171. Please refer to the previous profiling example for a discussion of the G.728 algorithm. This implementation was written in assembly language by Analogical Systems (now Voice Pump, Inc.), a firm specializing in the development of DSP software for telecommunications applications, especially speech compression. For more information, please see Appendix A, *Vendor Contact Information.*

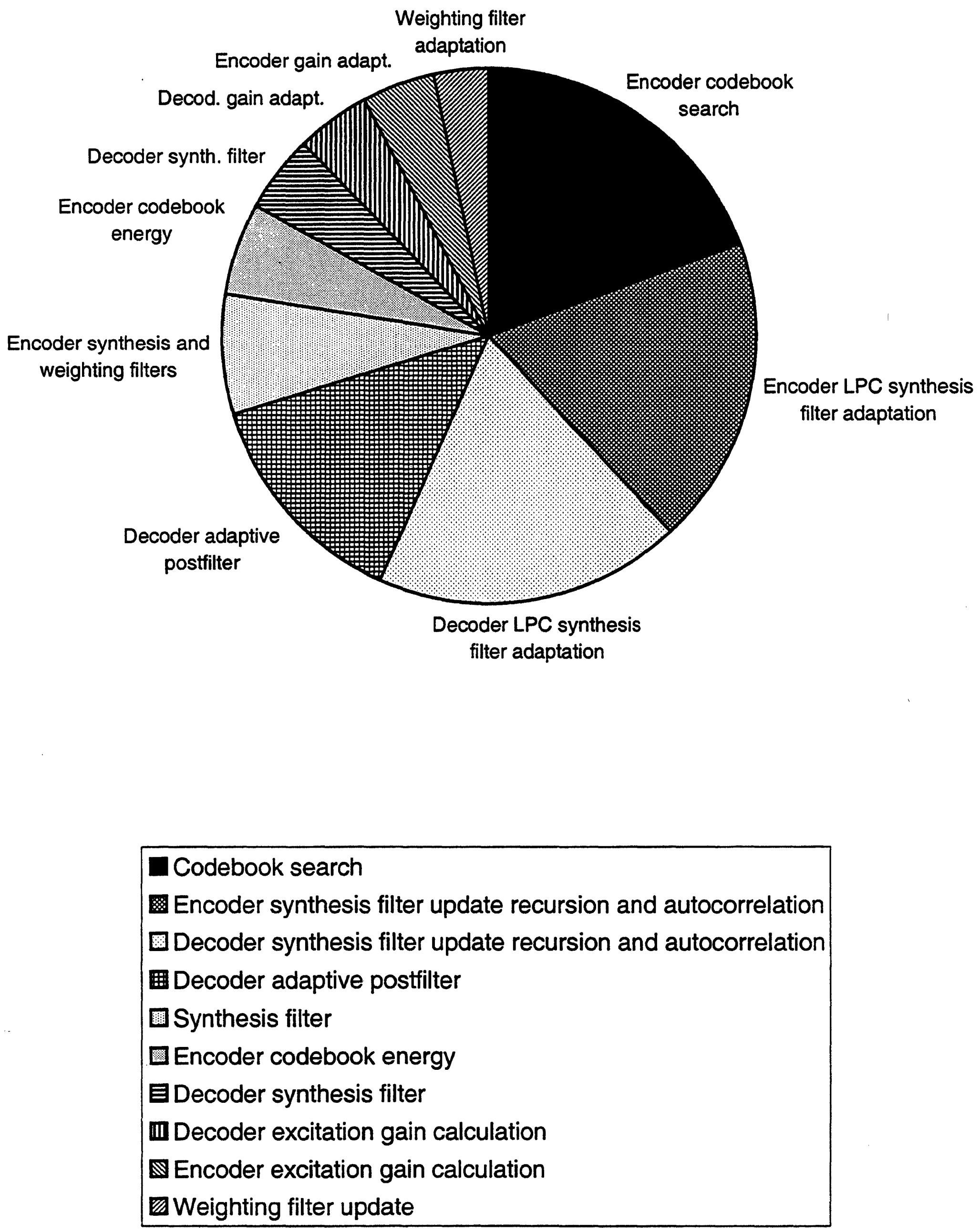
Table 6.2-5 and Figure 6.2-5 give function-level profiling data for this G.728 encoder and decoder implementation.

Table 6.2-5. Profiling Data for:

Application: G.728 Encoder and Decoder
Language: Assembly
Processor: Analog Devices ADSP-2171
Data From: Analogical Systems

Abbreviation	Function	Related DSP Processor Operations	Percent of Time in Function
Encoder codebook search	Codebook search	FIR filter, maximum search, branching	18.4
Encoder LPC synthesis filter adaptation	Encoder synthesis filter update recursion and autocorrelation	Durbin recursion, division, autocorrelation, windowing, LMS filter	17.4
Decoder LPC synthesis filter adaptation	Decoder synthesis filter update recursion and autocorrelation	Durbin recursion, division, autocorrelation, windowing, LMS filter	17.4
Decoder adaptive postfilter	Decoder adaptive postfilter	Data moves, FIR filter, maximum search	13.0
Encoder synthesis & weighting filters	Synthesis filter	IIR filter, FIR filter	6.6
Encoder codebook energy	Encoder codebook energy	MACs, data moves	5.4
Decoder synthesis filter	Decoder synthesis filter	FIR filter	4.4
Decoder gain adaptation	Decoder excitation gain calculation	Durbin recursion, division, autocorrelation, windowing, LMS filter	4.1
Encoder gain adaptation	Encoder excitation gain calculation	Durbin recursion, division, autocorrelation, windowing, LMS filter	4.1
Encoder weighting filter adaptation	Weighting filter update	Durbin recursion, division, autocorrelation, windowing, LMS filter	3.2

Figure 6.2-5. Profiling Data for G.728 Encoder and Decoder on ADSP-2171



USFS 1016 CELP Speech Encoder

In this section we examine profiling results for a speech coding application similar to that of the previous section.

As discussed earlier in this chapter, CELP stands for *code excited linear predictor* and refers to a technique for the compression of digital speech signals based on creating a parameterized model of the human vocal tract and transmitting the parameters of the model in place of the speech signal itself. CELP is used in applications where speech signals need to be transmitted digitally using low bit rates, such as secure telephones. USFS 1016 CELP is a United States federal government standard defining one variation on CELP speech coding. USFS 1016 CELP encodes telephone-quality speech into a 4,800 bits/second digital bit stream. Other variations of CELP are used in various applications.

The CELP speech encoder was implemented on a Motorola DSP560xx processor by Analogical Systems (now Voice Pump, Inc.). The implementation was done completely in assembly language. As with many speech coding techniques, the encoder (sometimes called the *analyzer*) is much more computation-intensive than the decoder (sometimes called the *synthesizer*).

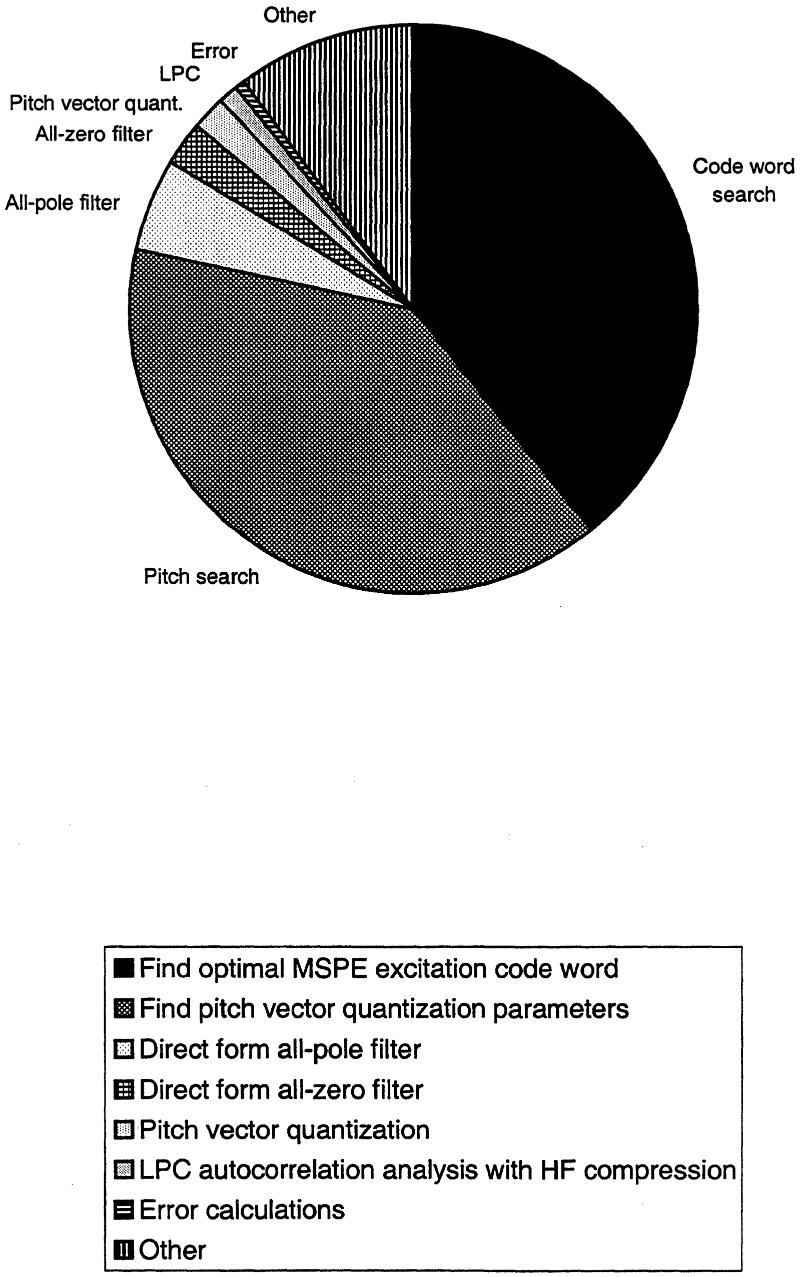
Table 6.2-6 and Figure 6.2-6 give function-level profiling data for this CELP speech encoder implementation.

Table 6.2-6. Profiling Data for:

Application: US Federal Standard 1016
Code Excited Linear Predictor (CELP)
Language: Assembly
Processor: Motorola DSP560xx
Data From: Analogical Systems

Abbreviation	Function	Related DSP Processor Operations	Percent of Time in Function
Code word search	Find optimal MSPE excitation code word	Correlation, convolution, scalar division, vector scaling	39.25
Pitch search	Find pitch vector quantization parameters	Correlation, convolution, scalar division, vector scaling, normalization	39.04
All-pole filter	Direct form all-pole filter	IIR filter	5.20
All-zero filter	Direct form all-zero filter	FIR filter	2.59
Pitch vector quant.	Pitch vector quantization	Specialized IIR filter	2.08
LPC	LPC autocorrelation analysis with HF compression	Autocorrelation, Durbin recursion	0.94
Error	Error calculations	FIR, IIR filters	0.73
Other	Other		10.16

Figure 6.2-6. Profiling Data For US Federal Standard 1016 CELP on DSP560xx



H.261 Video Encoder and Decoder

H.261 refers to a standard format for the encoding and compression of digital video signals for video conferencing applications. H.261 is a sub-standard of the video conferencing standard H.320. H.261 combines adaptive differential pulse code modulation, frequency-domain techniques, run-length coding, and Huffman coding.

This H.261 video encoder and decoder was implemented on a Texas Instruments TMS320C80 single-chip multiprocessor. The implementation was done completely in assembly language and uses three of the TMS320C80's four DSP processors. The chip's fourth DSP processor and RISC controller can be used to implement the remaining sub-standards comprising the complete H.320 standard (audio compression and system layer processing).

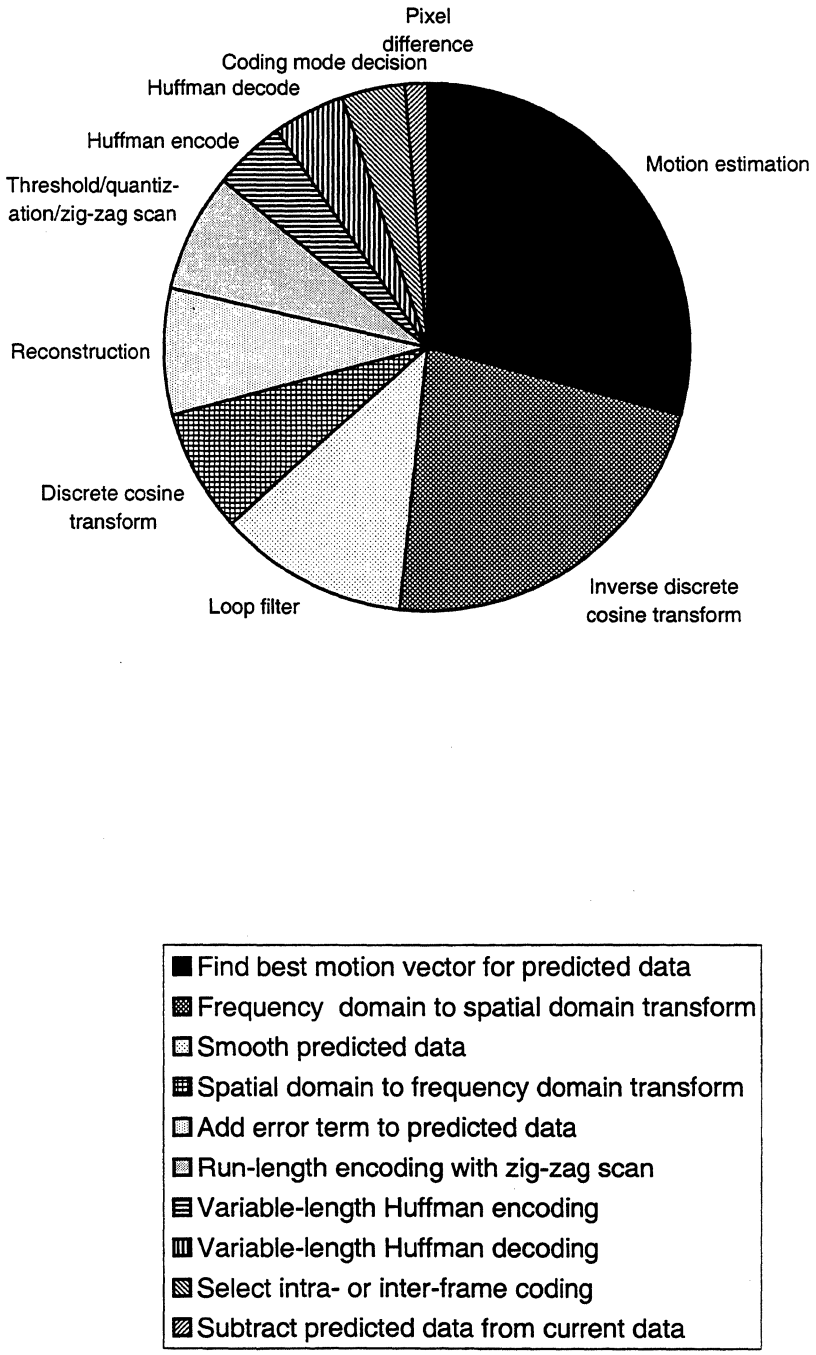
Table 6.2-7 and Figure 6.2-7 give function-level profiling data for this H.261 encoder implementation. Note that unlike our other profiling examples, this application was implemented on three processors. Therefore, the percentages of execution time shown in the table and figure correspond to percentages of the three processors' combined execution time.

Table 6.2-7. Profiling Data for:

Application: H.261 Video Encoder and Decoder
Language: Assembly
Processor: Texas Instruments TMS320C80 (using 3 processors)
Data From: Texas Instruments

Abbreviation	Function	Related DSP Processor Operations	Percent of Time in Function
Motion estimation	Find best motion vector for predicted data	Sum of absolute differences	29.0
Inverse discrete cosine transform	Frequency domain to spatial domain transform	DCT, FFT	22.6
Loop filter	Smooth predicted data	FIR filter	11.6
Discrete cosine transform	Spatial domain to frequency domain transform	FFT, DCT	7.7
Reconstruction	Add error term to predicted data	Add with saturate	7.7
Threshold/quantization/zig-zag scan	Run-length encoding with zig-zag scan	Thresholding, quantization, run-length coding	7.1
Huffman encode	Variable-length Huffman encoding	Huffman encoding	4.5
Huffman decode	Variable-length Huffman decoding	Huffman decoding	4.5
Coding mode decision	Select intra- or inter-frame coding	Autocorrelation	3.9
Pixel difference	Subtract predicted data from current data	Vector subtraction	1.3

Figure 6.2-7. Profiling Data for H.261 Video Encoder and Decoder on TMS320C80



Adaptive Beamformer

Beamforming refers to a class of techniques that involve processing and combining signals from a two- or three-dimensional array of sensors to direct the sensitivity of the sensor array in a particular direction (forming a beam). Similarly, signals sent by an array of transmitting elements can be processed so that the combined effect of the entire array is to direct a signal in a particular pattern or beam. Thus, beamforming techniques allow a physically immobile sensor or antenna array to "look" in different directions. Such techniques are used in radar, sonar, and medical imaging products, among other applications.

Adaptive beamforming is an extension of standard beamforming techniques in which the beamforming signal processing automatically adapts to compensate for distortion introduced elsewhere in the system. Our example adaptive beamformer was implemented on the Texas Instruments TMS320C3x processor by DSP Software Engineering. This implementation was written completely in assembly code.

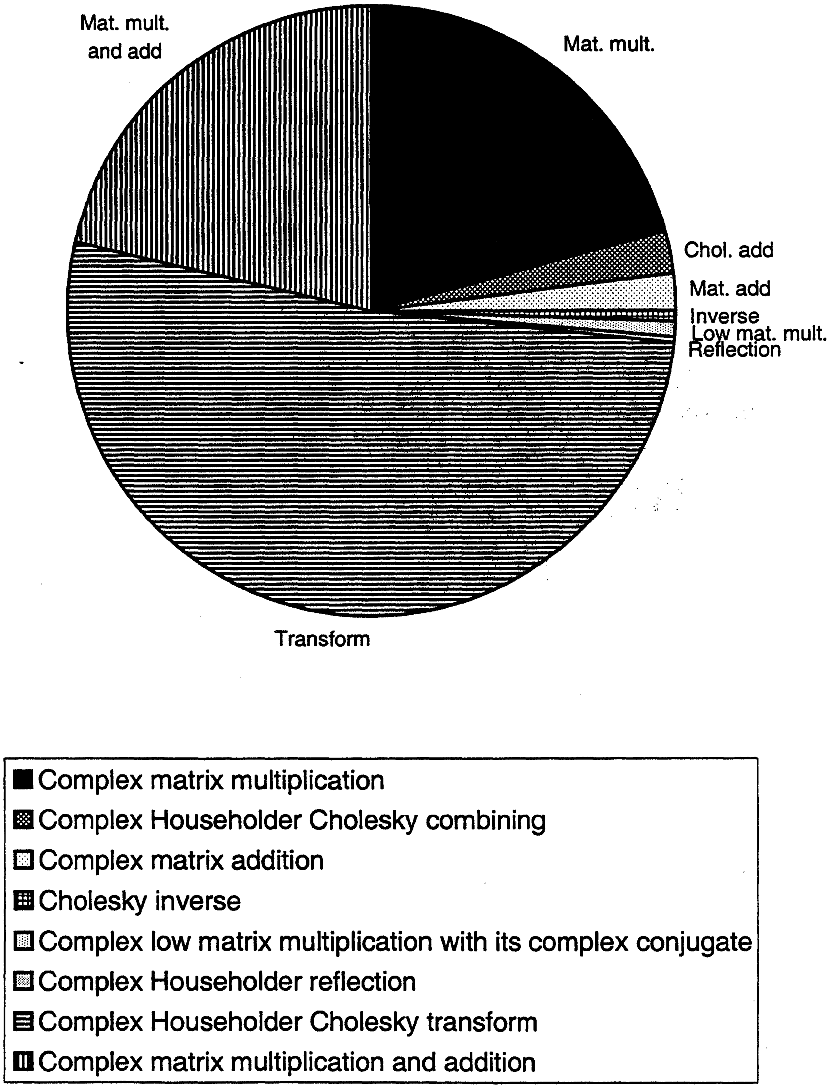
Table 6.2-8 and Figure 6.2-8 give function-level profiling data for this beamforming application implementation.

Table 6.2-8. Profiling Data for:

Application: Adaptive Beamforming Algorithm
Language: Assembly
Processor: Texas Instruments TMS320C3x
Data from: DSP Software Engineering

Abbreviation	Function	Related DSP Processor Operations	Percent of Time in Function
Transform	Complex Householder Cholesky transform	Matrix decomposition, diagonalizing; find eigenvalues	51.99
Mat. mult. and add	Complex matrix multiplication and addition	Matrix multiplication (vector product), matrix addition (vector sum)	21.42
Mat. mult.	Complex matrix multiplication	Matrix multiplication (vector product)	20.91
Chol. add	Complex Householder Cholesky combining	Matrix addition (vector sum)	2.24
Mat. add	Complex matrix addition	Matrix addition (vector sum)	1.76
Inverse	Cholesky inverse	Simplified matrix inversion	0.72
Low mat. mult.	Complex low matrix multiplication with its complex conjugate	Matrix multiplication (vector product)	0.71
Reflection	Complex Householder reflection	Matrix data move	0.25

Figure 6.2-8. Profiling Data For Adaptive Beamforming Algorithm on TMS320C3x



GSM Speech Coder

GSM is currently the prevalent standard for digital mobile telephony in Europe. These profiling examples are for the speech coder portions of a GSM digital cellular telephone.

The profiled implementation shown in Table 6.2-9 and Figure 6.2-9 reflects normal execution of the GSM enhanced full-rate encoder. The profiled implementation of the GSM enhanced full-rate decoder is shown in Table 6.2-10 and Figure 6.2-10. This encoder and decoder were both developed in assembly language by Motorola India Electronics, Ltd. for the Motorola DSP563xx.

The profiling data shown in Table 6.2-11 and Figure 6.2-11 reflects normal execution of both the encoder and decoder for an implementation on the Texas Instruments TMS320C62xx. This implementation was developed in assembly language by Texas Instruments.

Table 6.2-9. Profiling Data for:

Application: GSM Enhanced Full-Rate Speech Encoder
Language: Assembly
Processor: Motorola DSP563xx
Data From: Motorola India Electronics Ltd.

Abbreviation	Function	Related DSP Processor Operations	Percent of Time in Function
Search	Search best algebraic code vector	Recursive calculation of correlations, maximum search	30.80
Correlation	Compute filter coefficient correlation	Vector scaling, generate 40x40 correlation matrix	8.10
LSP quant.	LSP quantization	Euclidean distance, minimum search	7.80
Convolution	Convolution during closed-loop search	Convolution, vector dot product	6.30
Az to LSP	Az to LSP	Polynomial addition and multiplication, interpolation, polynomial evaluation	5.10
Norm corr.	Normalized correlation during closed loop	10th order FIR filter, correlation	5.00
IIR synth.	IIR synthesis filter	10th order IIR filter	4.80
LPC analysis	LPC analysis	Auto-correlation, IIR filter, Durbin recursion, vector scaling, division	4.80
Max. lag	Open-loop maximum lag	Correlation, maximum search	4.70
Other	Other		22.60

Figure 6.2-9. Profiling Data for GSM Enhanced Full-Rate Speech Encoder on DSP563xx

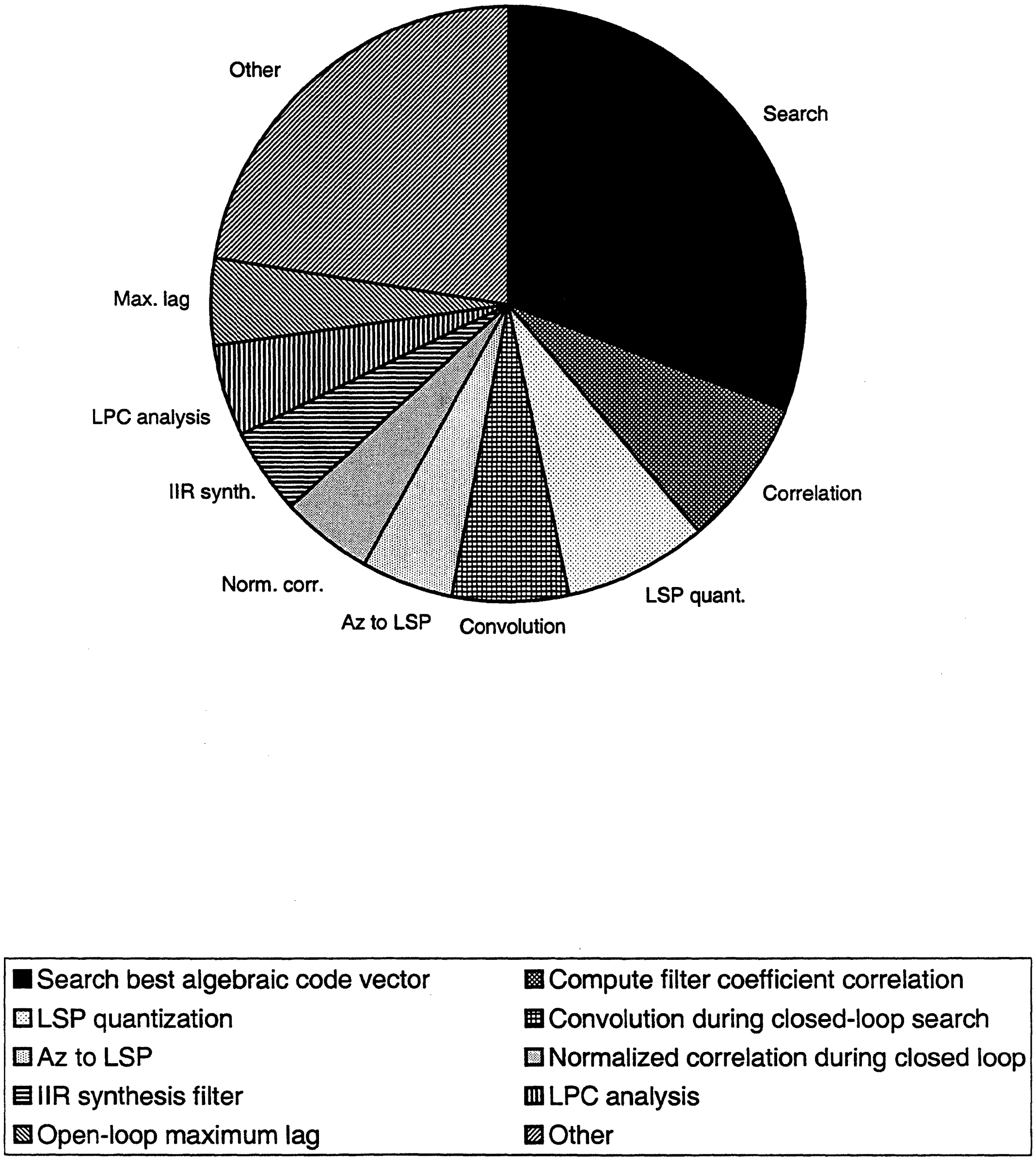


Table 6.2-10. Profiling Data for:

Application: GSM Enhanced Full-Rate Speech Decoder
Language: Assembly
Processor: Motorola DSP563xx
Data From: Motorola India Electronics Ltd.

Abbreviation	Function	Related DSP Processor Operations	Percent of Time in Function
Codebook	Adaptive codebook excitation	Interpolating FIR filter	26.00
Synthesis	Synthesis filter	10th order IIR filter	17.10
Post-process	Post-processing filter	Vector scaling, FIR filter, IIR filter	11.10
LSP polynomial	Get LSP polynomial	Polynomial multiplication	6.20
Decode alg.	Decode algebraic code	Generate vector from coded indices	3.20
Decode gain	Decode gain code	Logarithm, FIR filter	3.00
Pre-emphasis	Pre-emphasis	1st order FIR filtering	2.10
LSP to az	LSP to az	Vector add	1.60
Other	Other		29.70

Figure 6.2-10. Profiling Data for GSM Enhanced Full-Rate Speech Decoder on DSP563xx

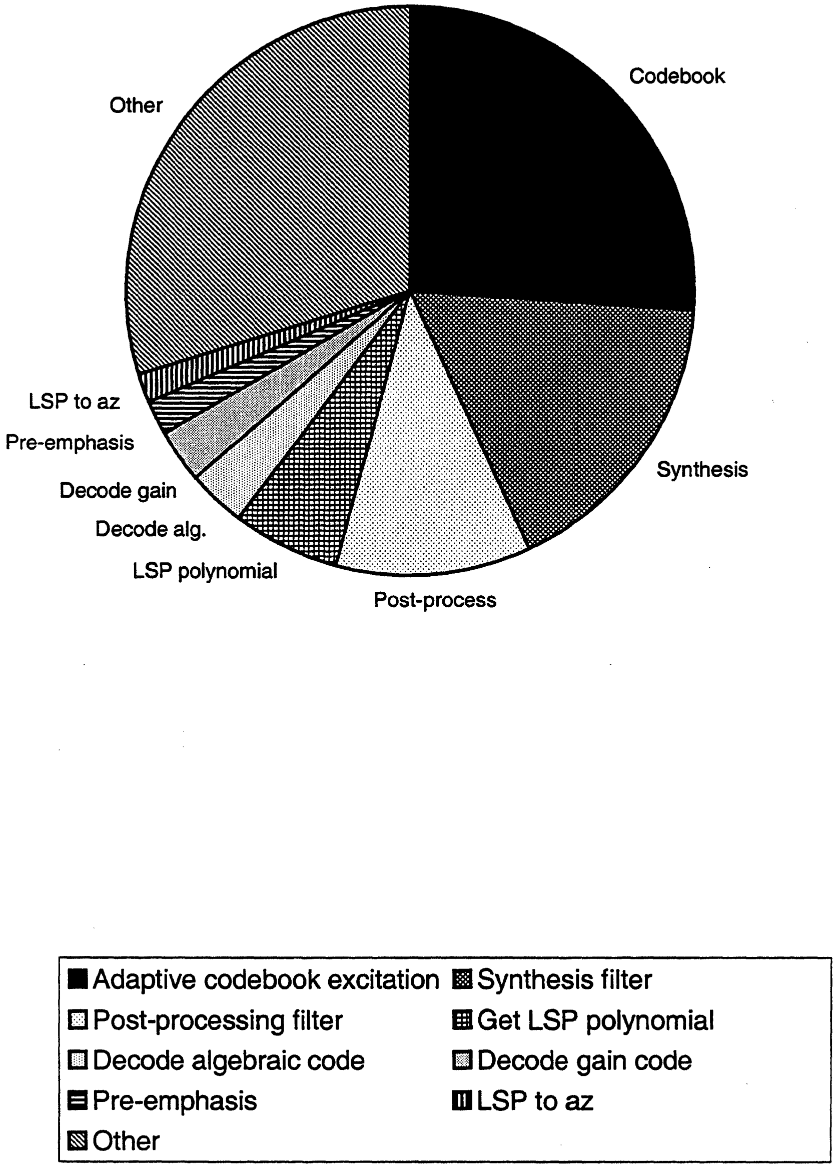
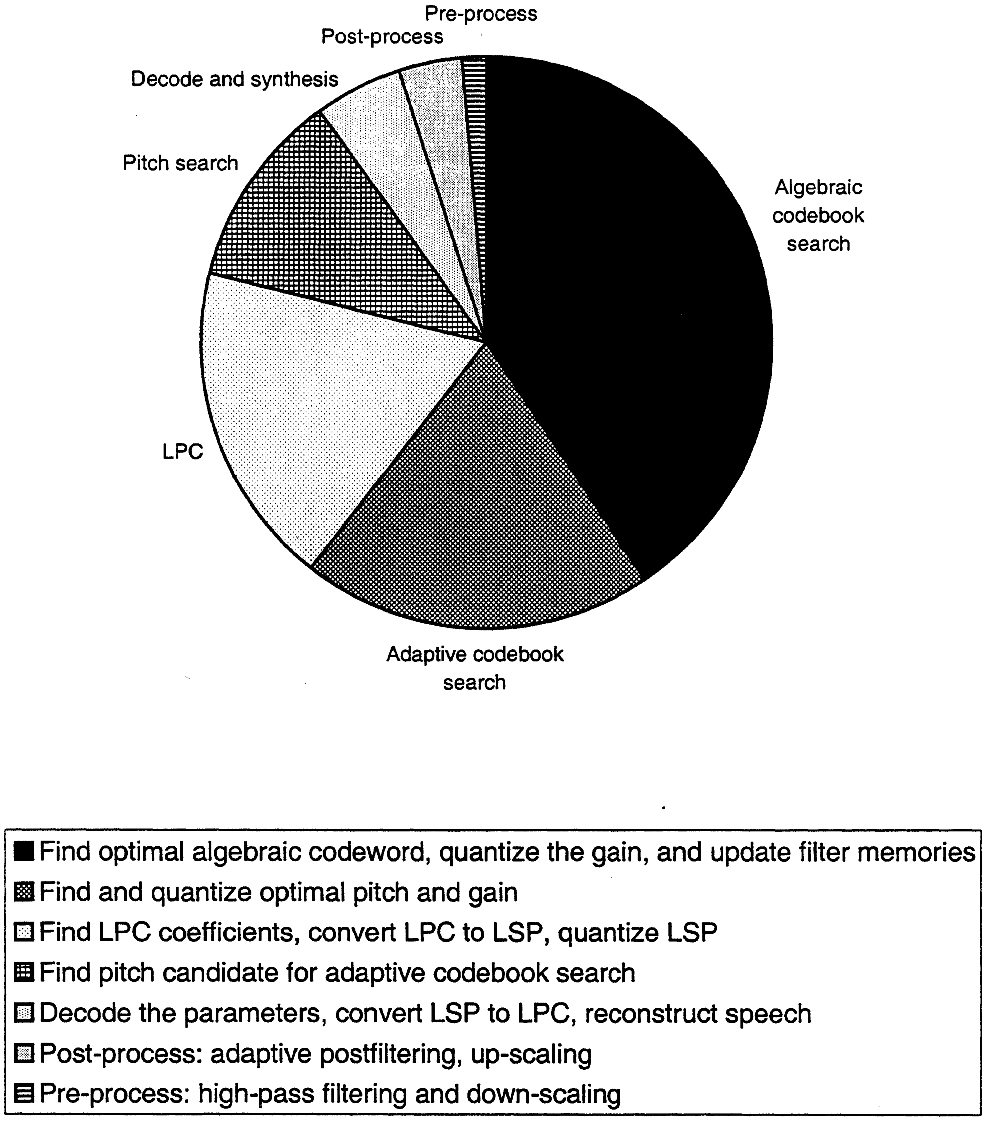


Table 6.2-11. Profiling Data for:

Application: Enhanced Full-Rate GSM Vocoder
Language: Assembly
Processor: Texas Instruments TMS320C62xx
Data From: Texas Instruments

Abbreviation	Function	Related DSP Processor Operations	Percent of Time in Function
Algebraic codebook search	Find optimal algebraic codeword, quantize the gain, and update filter memories	Correlation, vector sum, scalar quantization	39.90
Adaptive codebook search	Find and quantize optimal pitch and gain	IIR & FIR filters, convolution, vector sum, interpolation, scalar quantization	19.70
LPC	Find LPC coefficients, convert LPC to LSP, quantize LSP	Autocorrelation, find roots of polynomial, vector quantization, interpolation	18.10
Pitch search	Find pitch candidate for adaptive codebook search	IIR and FIR filters, cross-correlation	11.00
Decode and synthesis	Decode the parameters, convert LSP to LPC, reconstruct speech	IIR filter, interpolation, vector sum	5.00
Post-process	Post-process: adaptive postfiltering, up scaling	IIR and FIR filters, dot product	3.50
Pre-process	Pre-process: high-pass filtering and down-scaling	IIR and FIR filters	1.30

Figure 6.2-11. Profiling Data for GSM Enhanced Full-Rate Vocoder on TMS320C62xx



6.3 Instruction-Level Profiling

In this section, we present instruction-level profiling data for several example applications: various portions of a GSM cellular telephone and ADPCM speech compression. Instruction-level profiling data reveals the percentage of the total instruction cycles attributable to different categories of processor instructions while running a given application. It therefore provides a sense of the relative importance of different instruction types in the application.

As in the function-level profiling results presented earlier in this chapter, the profiling data presented here is for the main, steady-state operating mode of each application.

For each application, we list the ten most frequently executed instruction types. We provide a brief description of each instruction type and the percentage of execution time accounted for by each type.

More detailed technical information on each of these applications can be found in texts and journals on communications and signal processing, a sampling of which are listed in the *References* section at the end of this report.

GSM Channel Coder

GSM is currently the prevalent standard for digital mobile telephony in Europe. This profiling example is for the channel coder portion of a GSM digital cellular telephone. The channel coder is responsible for a variety of tasks including bit packing, interleaving, encryption, error control coding, and error detection and correction. The instruction profiling data here reflects normal execution of both the encoder and decoder over several frames of data.

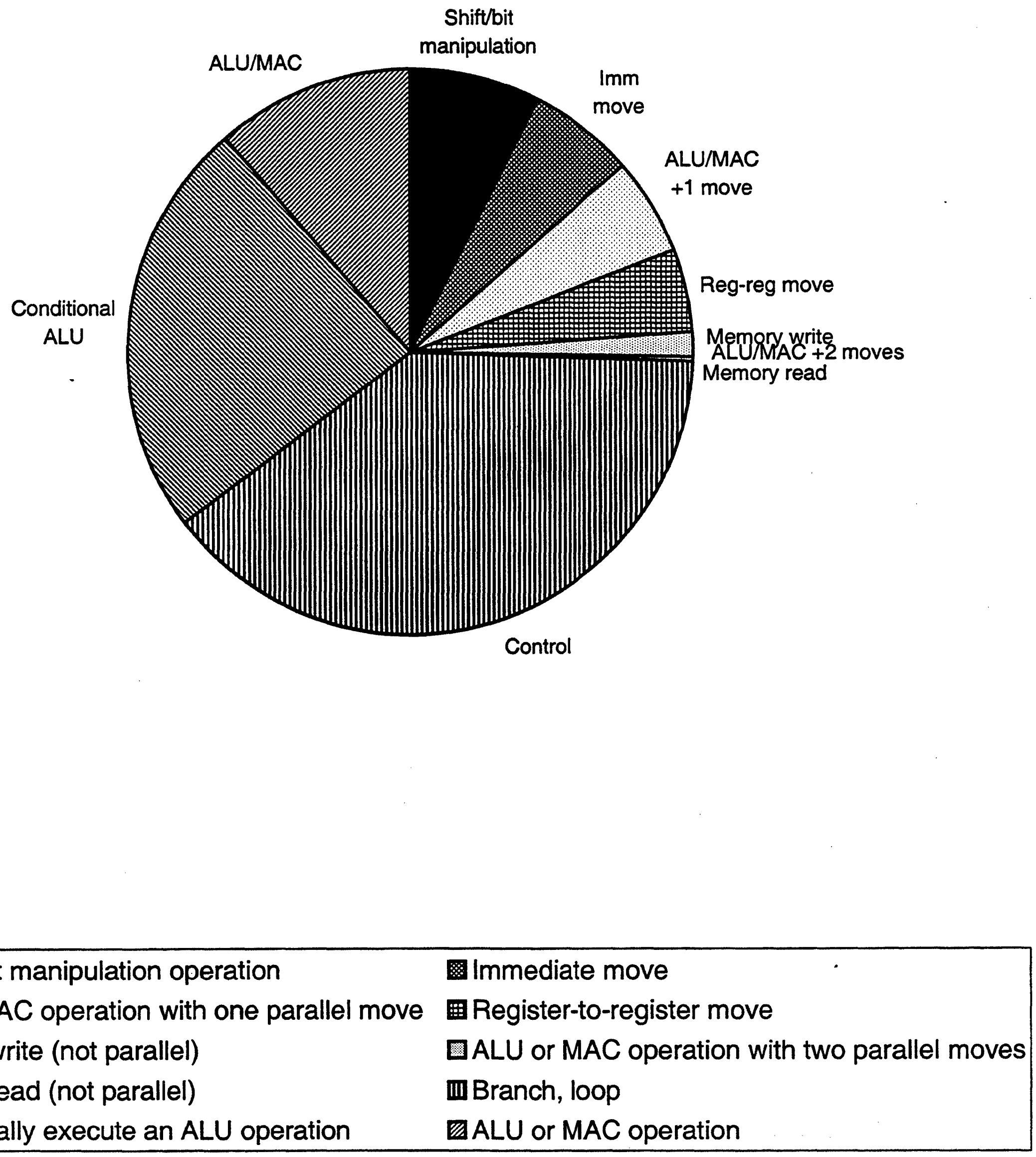
The profiled implementation was developed in assembly language by Lucent Technologies for the Lucent Technologies DSP1618. Table 6.3-1 and Figure 6.3-1 display instruction-level profiling data for this GSM channel coder implementation.

Table 6.3-1. Profiling Data for:

Application: GSM Channel Coder
Language: Assembly
Processor: Lucent Technologies DSP1618
Data From: Lucent Technologies

Abbreviation	Instruction Type	Percent of Total Inst. Cycles
Control	Branch, loop	39.2
Conditional ALU	Conditionally execute an ALU operation	23.8
ALU/MAC	ALU or MAC operation	11.4
Shift/bit manipulation	Shift or bit manipulation operation	7.6
Imm move	Immediate move	6.1
ALU/MAC + 1 move	ALU or MAC operation with one parallel move	5.6
Reg-reg move	Register-to-register move	4.5
Memory write	Memory write (not parallel)	1.4
ALU/MAC + 2 moves	ALU or MAC operation with two parallel moves	0.4
Memory read	Memory read (not parallel)	0.1

Figure 6.3-1. Profiling Data For GSM Channel Coder on DSP1618



GSM Digital Receiver

This profiling example is for the digital receiver portion of a GSM digital cellular telephone. The digital receiver is responsible for a variety of tasks including received signal strength estimation, synchronization, frequency error estimation, and demodulation.

The profiled implementation was developed in assembly language by Lucent Technologies for the Lucent Technologies DSP1618.

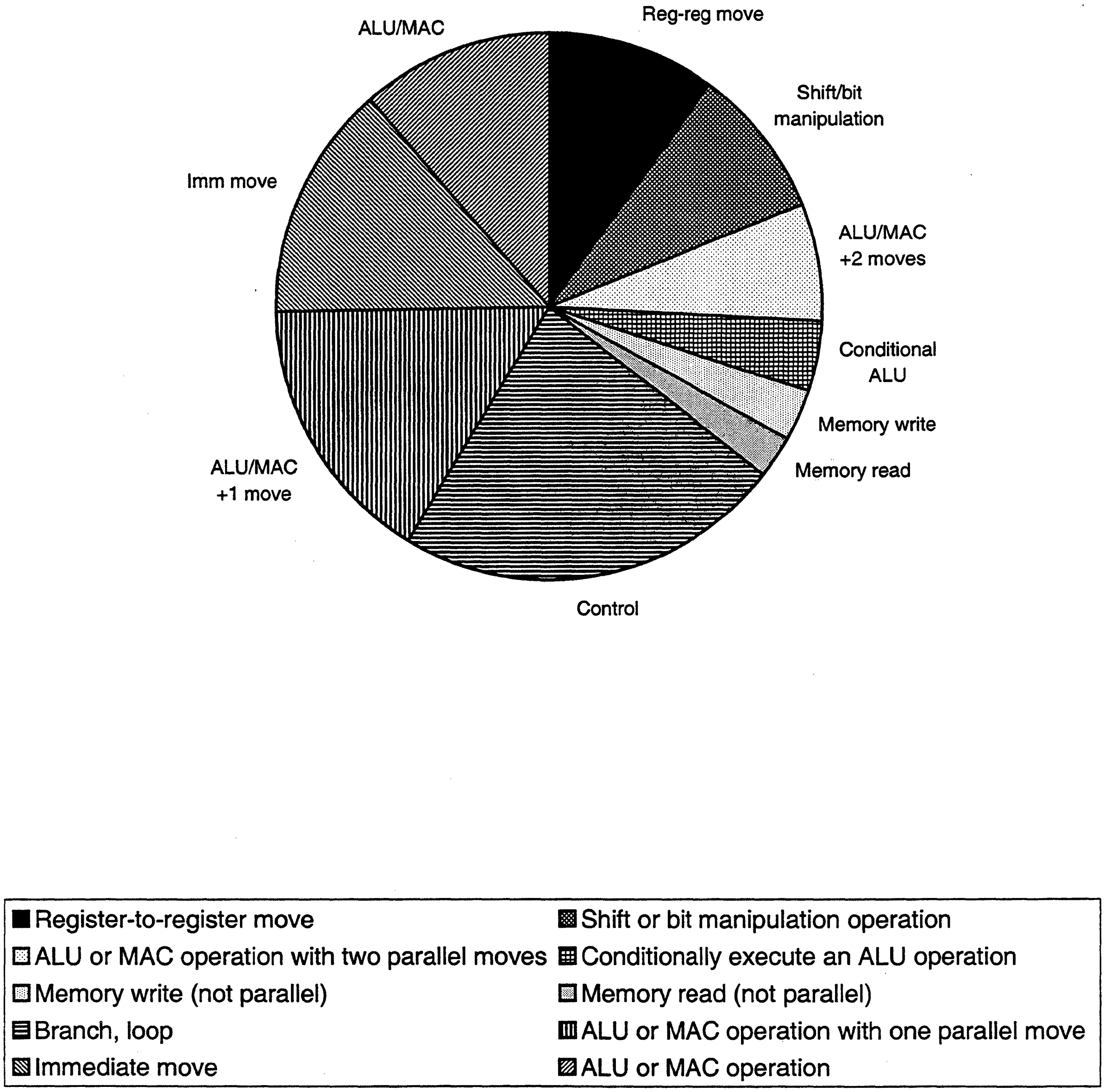
Table 6.3-2 and Figure 6.3-2 display instruction-level profiling data for this GSM digital receiver implementation.

Table 6.3-2. Profiling Data for:

Application: GSM Receiver
Language: Assembly
Processor: Lucent Technologies DSP1618
Data From: Lucent Technologies

Abbreviation	Instruction Type	Percent of Total Inst. Cycles
Control	Branch, loop	22.9
ALU/MAC + 1 move	ALU or MAC operation with one parallel move	16.2
Imm move	Immediate move	13.8
ALU/MAC	ALU or MAC operation	11.5
Reg-reg move	Register-to-register move	9.9
Shift/bit manipulation	Shift or bit manipulation operation	9.0
ALU/MAC + 2 moves	ALU or MAC operation with two parallel moves	7.0
Conditional ALU	Conditionally execute an ALU operation	4.2
Memory write	Memory write (not parallel)	3.0
Memory read	Memory read (not parallel)	2.5

Figure 6.3-2. Profiling Data For GSM Receiver on DSP1618



GSM Speech Coder

Table 6.3-3 and Figure 6.3-3 display instruction-level profiling data for a full-rate GSM speech coder implementation. The speech coder performs compression and decompression of digital speech signals. The profiled implementation was developed in assembly language by Lucent Technologies for the Lucent Technologies DSP1618. The profiling data presented here represent normal operation of both the encoder and decoder for one frame of speech data.

The instruction-level profiling example shown in Table 6.3-4 and Figure 6.3-4 is for the full-rate GSM speech decoder portion of a GSM digital cellular telephone. The profiling example shown in Table 6.3-5 and Figure 6.3-5 is for the enhanced full-rate speech decoder portion of a GSM digital cellular telephone. Both of these profiled implementations were developed in assembly language by Motorola India Electronics Ltd. for the Motorola DSP563xx.

Table 6.3-3. Profiling Data for:

Application: Full-Rate GSM Speech Coder
Language: Assembly
Processor: Lucent Technologies DSP1618
Data From: Lucent Technologies

Abbreviation	Instruction Type	Percent of Total Inst. Cycles
ALU/MAC + 1 move	ALU or MAC operation with one parallel move	54.6
ALU/MAC + 2 moves	ALU or MAC operation with two parallel moves	11.7
Conditional ALU	Conditionally execute an ALU operation	8.4
Imm move	Immediate move	6.3
Control	Branch, loop	5.6
Shift/bit manipulation	Shift or bit manipulation operation	4.9
Reg-reg move	Register-to-register move	3.4
ALU/MAC	ALU or MAC operation	2.9
Memory write	Memory write (not parallel)	1.1
Memory read	Memory read (not parallel)	1.1

Figure 6.3-3. Profiling Data For Full-Rate GSM Speech Coder on DSP1618

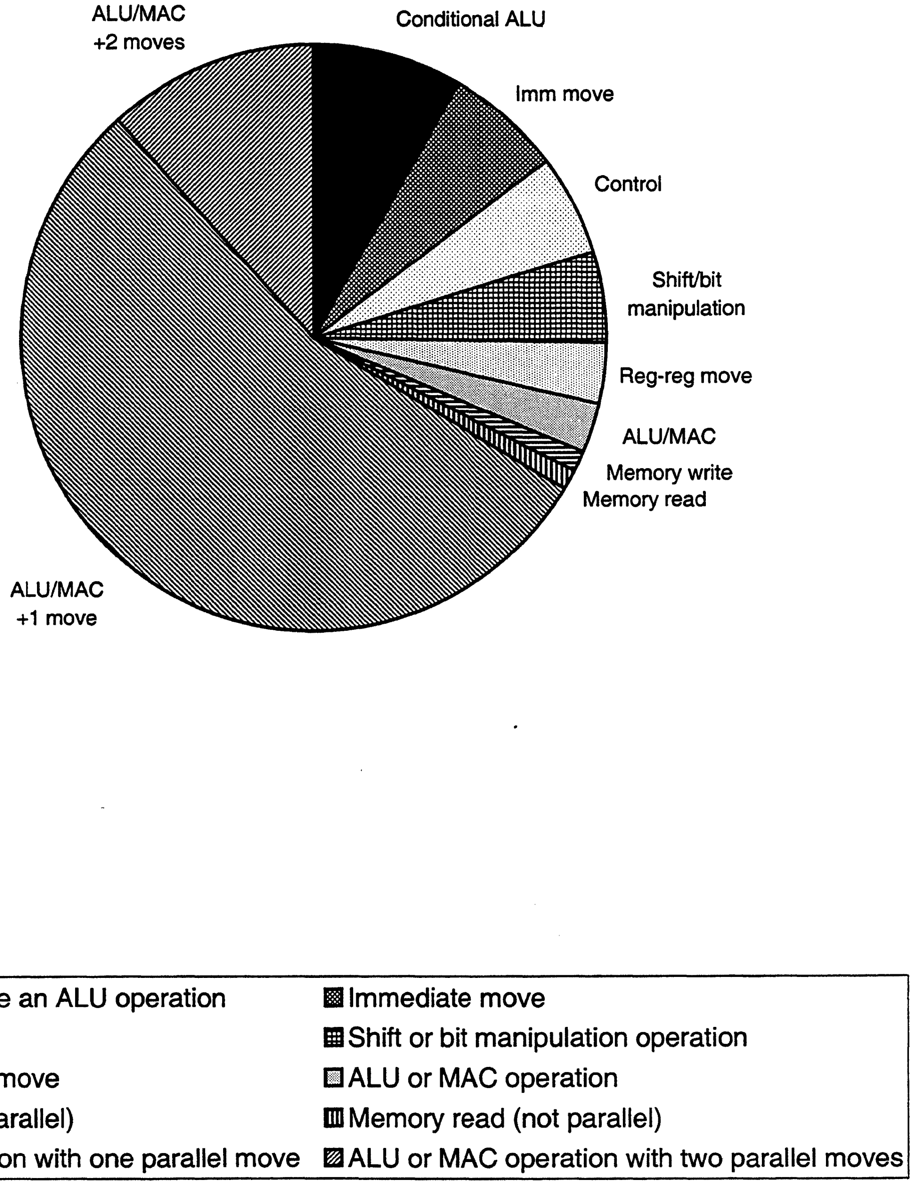


Table 6.3-4. Profiling Data for:

Application: GSM Full-Rate Speech Decoder
Language: Assembly
Processor: Motorola DSP563xx
Data From: Motorola India Electronics Ltd.

Abbreviation	Instruction Type	Percent of Total Inst. Cycles
Move	Memory-to-register or register-to-memory move	35.7
MAC, 2 moves	MAC with two parallel moves	34.7
ALU	ALU operations	16.7
Bit manipulation	Bit manipulation	6.2
Control	Control code	3.5
MAC, 1 move	MAC with one parallel move	2.6
Reg-reg move	Register-to-register move	1.1

Figure 6.3-4. Profiling Data For GSM Full-Rate Speech Decoder on DSP563xx

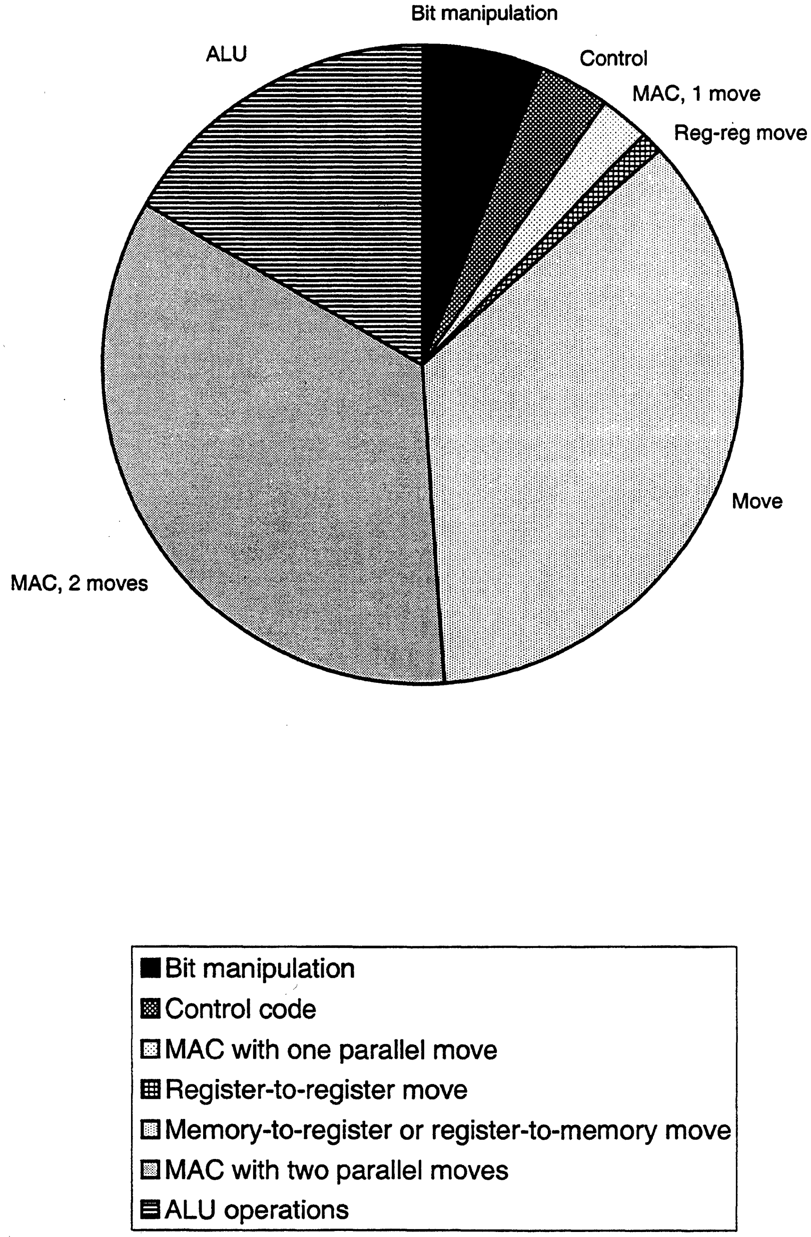
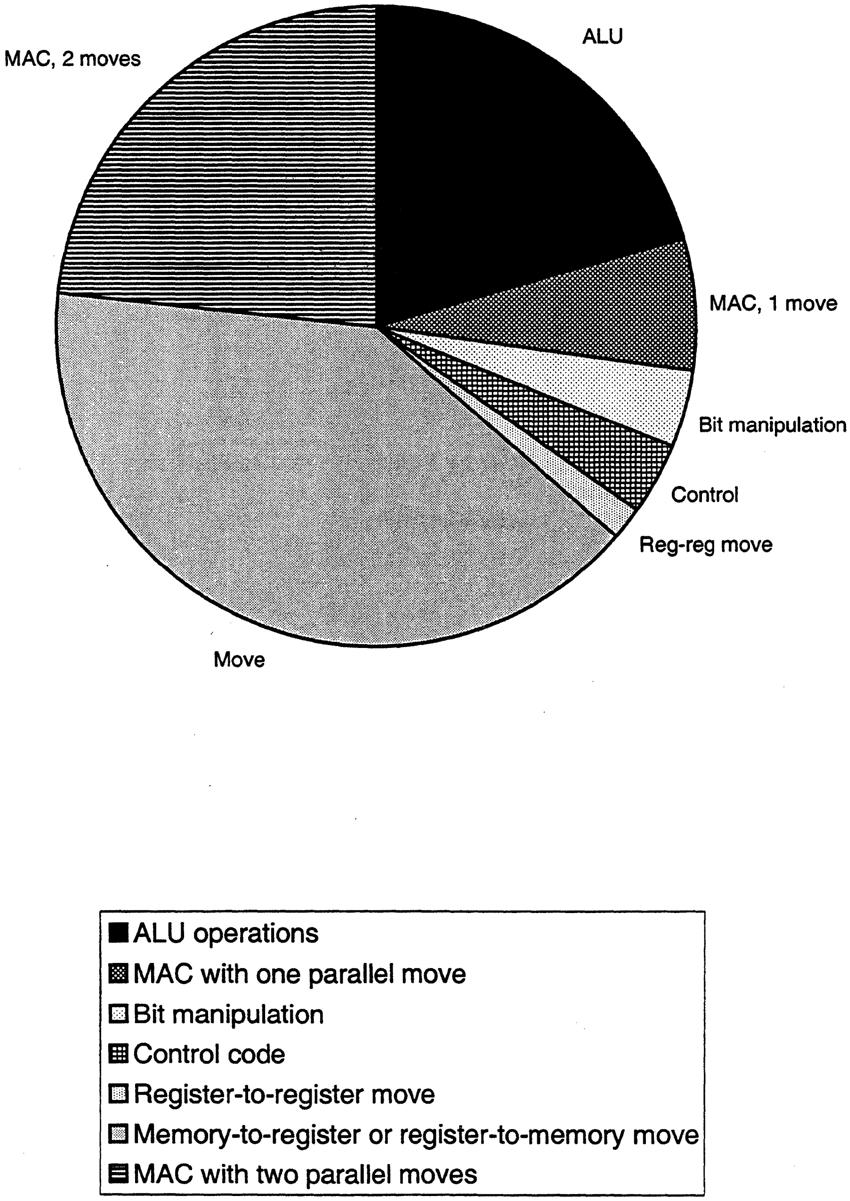


Table 6.3-5. Profiling Data for:

Application: GSM Enhanced Full-Rate Speech Decoder
Language: Assembly
Processor: Motorola DSP563xx
Data From: Motorola India Electronics Ltd.

Abbreviation	Instruction Type	Percent of Total Inst. Cycles
Move	Memory-to-register or register-to-memory move	40.2
MAC, 2 moves	MAC with two parallel moves	23.3
ALU	ALU operations	20.5
MAC, 1 move	MAC with one parallel move	6.6
Bit manipulation	Bit manipulation	4.1
Control	Control code	3.5
Reg-reg move	Register-to-register move	1.8

Figure 6.3-5. Profiling Data For GSM Enhanced Full-Rate Speech Decoder on DSP563xx



ADPCM Speech Coder

ADPCM stands for *adaptive differential pulse code modulation*, a speech coding technique used to compress eight-bit speech samples to four-bit samples with minimal loss of fidelity. ITU-T standard G.721 defines standard ADPCM encoder and decoder algorithms that are common in telecommunications applications.

The following two instruction profiling examples show two implementations of the G.721 ADPCM algorithm on the Analog Devices ADSP-21xx processor: the first was implemented in assembly code, the second in C. This data was provided by the Institute for Integrated Systems in Signal Processing (ISS) of the Aachen University of Technology, Germany. The ISS (led by Professor Heinrich Meyr) is concerned with the analysis and synthesis of complex information processing systems. The DSP Tools Group, a part of ISS, focuses on the DSP system design methodology, as well as on development of appropriate design automation tools.

Table 6.3-6 and Figure 6.3-6 display instruction-level profiling data for the assembly-language ADPCM implementation.

Table 6.3-7 and Figure 6.3-7 display instruction-level profiling data for the C-language ADPCM implementation. Note in particular that the C implementation makes much heavier use of non-parallel data move instructions than the assembly-language implementation. This is symptomatic of the inefficiency of C compilers for many DSP processors, especially for conventional fixed-point processors such as the ADSP-21xx.

Table 6.3-6. Profiling Data for:

Application: ADPCM Speech Encoder and Decoder
Language: Assembly
Processor: Analog Devices ADSP-21xx
Data From: Aachen University, ISS

Abbreviation	Instruction Type	Percent of Total Inst. Cycles
ALU with move(s)	ALU operation with one or two parallel moves	30.2
ALU only	ALU operation with no parallel moves	18.9
Single move only	Single data move operation	18.5
Shift with move(s)	Shift operation with one or two parallel moves	9.7
Shift only	Shift operation with no parallel moves	8.1
MAC with move(s)	MAC operation with one or two parallel moves	5.6
Control	Control flow operation (e.g., branch, call)	5.5
Dual move(s)	Two data moves in parallel	2.9
Misc	Stack and other miscellaneous operations	0.4
MAC only	MAC operation with no parallel moves	0.2

Figure 6.3-6. Profiling Data For ADPCM Speech Encoder and Decoder on ADSP-21xx

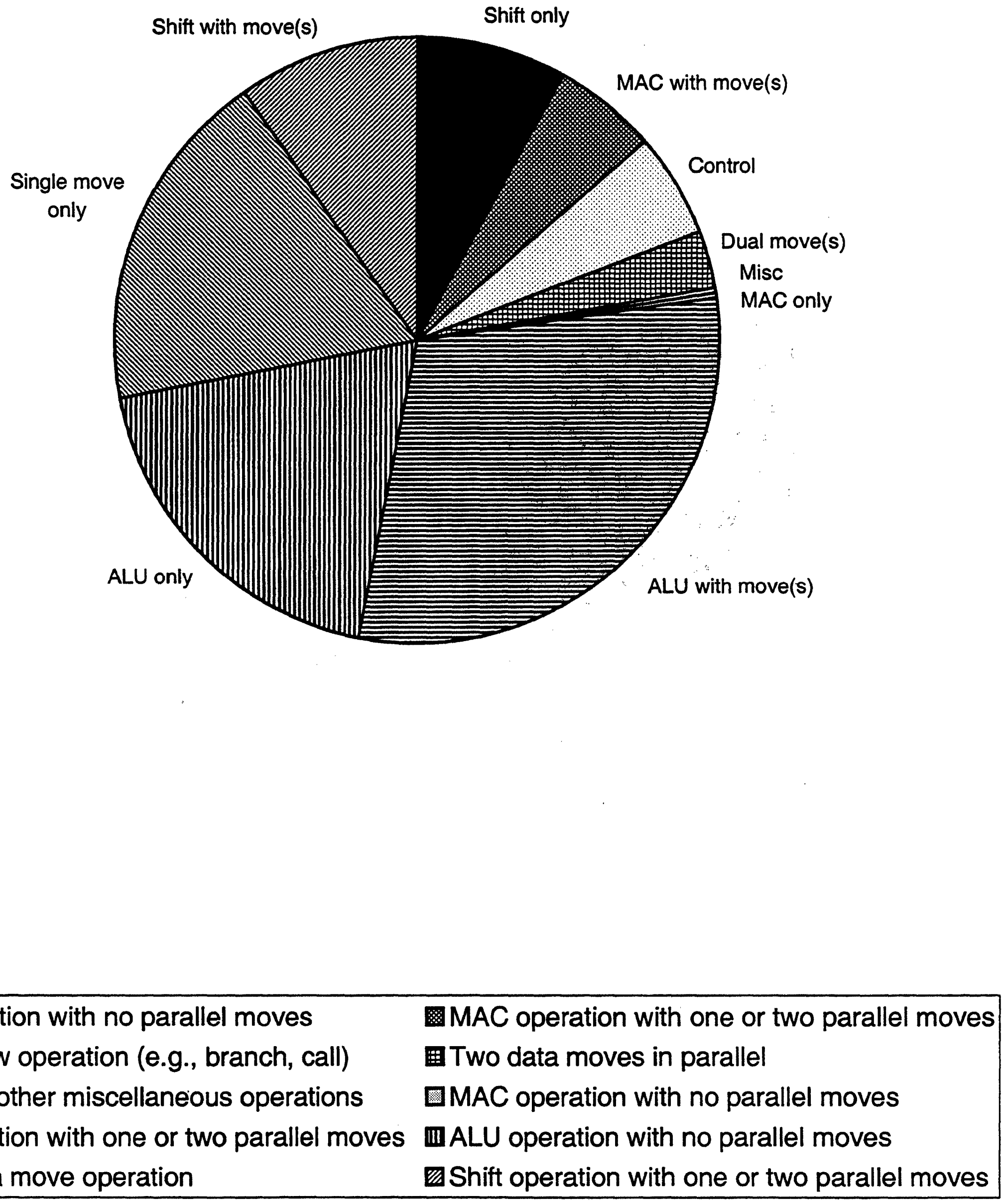
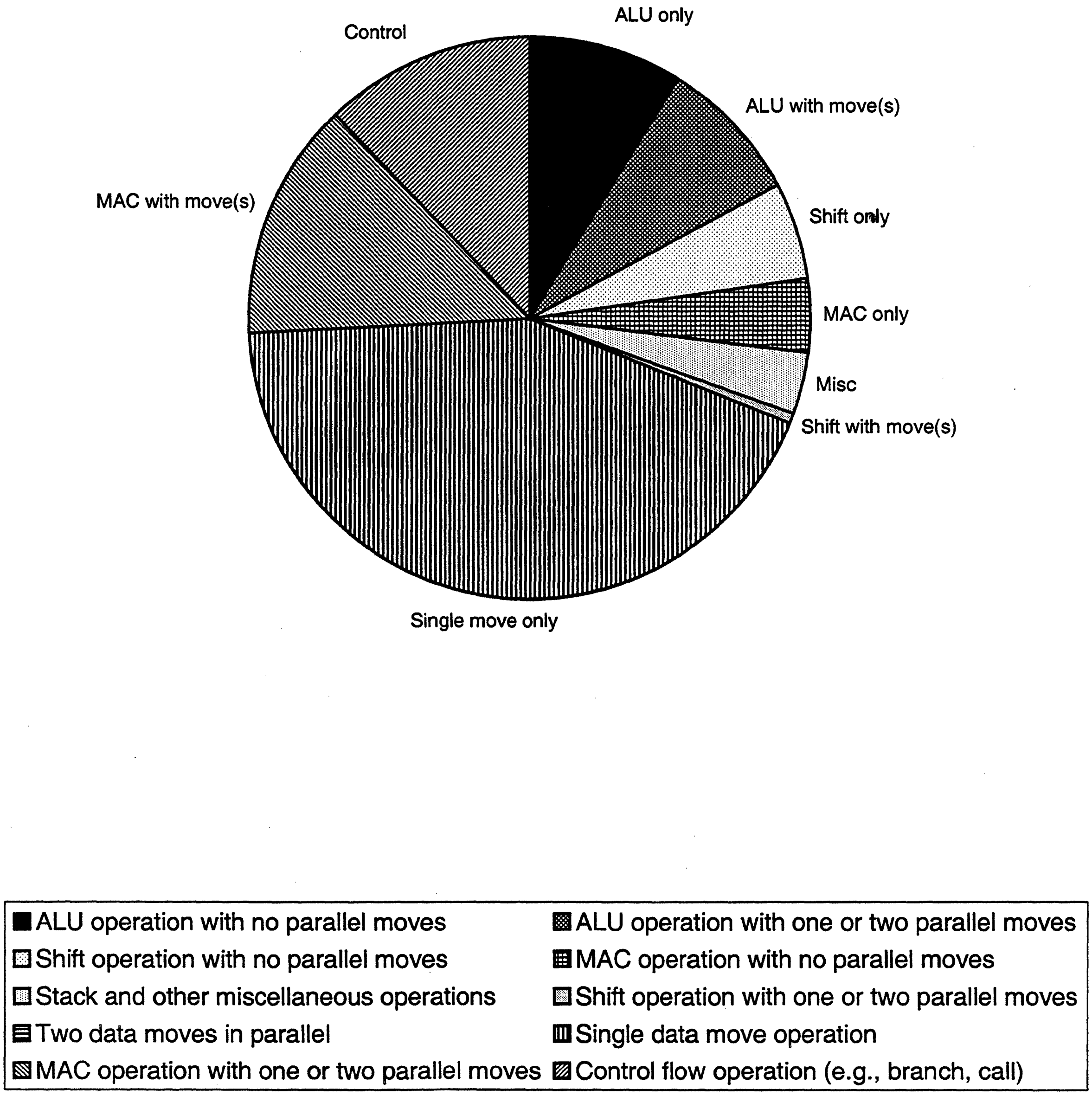


Table 6.3-7. Profiling Data for:

Application: ADPCM Speech Encoder and Decoder
Language: C
Processor: Analog Devices ADSP-21xx
Data From: Aachen University, ISS

Abbreviation	Instruction Type	Percent of Total Inst. Cycles
Single move only	Single data move operation	43.1
MAC with move(s)	MAC operation with one or two parallel moves	13.6
Control	Control flow operation (e.g., branch, call)	12.3
ALU only	ALU operation with no parallel moves	9.0
ALU with move(s)	ALU operation with one or two parallel moves	8.3
Shift only	Shift operation with no parallel moves	5.5
MAC only	MAC operation with no parallel moves	4.3
Misc	Stack and other miscellaneous operations	3.6
Shift with move(s)	Shift operation with one or two parallel moves	0.4
Dual move(s)	Two data moves in parallel	0.0

Figure 6.3-7. Profiling Data For ADPCM Speech Encoder and Decoder on ADSP-21xx



7. Processor Analyses

In this chapter we examine 17 processor families from four different vendors. In a consistent manner, we describe in detail many important aspects of these processors, including architecture, peripherals, power management, technical support, and development tools. We also evaluate the strengths and weaknesses of each processor to provide insight into which processor best fits the needs of a given application. Unique and outstanding features that could be advantageous in particular applications are noted, as well as omissions of features and limitations that could be detrimental.

> *Indented, italicized paragraphs (such as this paragraph) used in this chapter represent subjective comments of engineers at BDTI. These comments are based on extensive experience with DSP systems, processors, and algorithms. Remember, however, that these comments are opinions, not facts, and should be read in that light.*

We begin each processor analysis with a high-level overview, presenting a brief outline of the processor's characteristics, its history, and noteworthy details. Then, over the course of many subsections, we delve into the internals of the processor and its functionality. We cover the following topics in detail since we consider them to be among the most relevant considerations in evaluating a processor for a given set of application requirements:

- **Architecture**
  Provides an overview of the processor's architecture, explains which execution units are found in the processor, specifies the data and instruction word widths, etc.
  - Data Path
    Discusses the main aspects of the data path; i.e., integer ALUs and/or floating-point units; support for guard bits, saturation, and rounding; and capabilities of shifters, multipliers, and other special function units. We also consider registers and their widths.
  - Memory System
    Includes an explanation of the processor's program and data memory organization, on-chip memory configurations, address spaces, and cache capabilities. Load/store capabilities and on-chip bandwidth restrictions are also discussed.
  - External Memory Interface
    Explains the processor's external memory interface in terms of bus width, bus speed, off-chip bandwidth restrictions, and support for wait states. If available, we also discuss bus protocols such as split-transaction or pipelined buses and support for special types of memory, such as DRAM.
  - Address Generation Units
    Discusses the processor's addressing modes; e.g., register-indirect addressing, support for immediate data, etc. Explains how addresses are generated; and comments on the number of address registers.

Chapter 7

  - Pipeline
    Outlines and explains pipeline stages where necessary for understanding software performance and optimization on the processor. Pipeline properties such as interlocking, hazards, and sub-pipelined execution units are also explored here.

- **Instruction Set**
  Summary of the processor's instruction set and registers and discussion of the properties of the instruction set.
  - Assembly Language Format
    Discusses the style of the processor's assembly language syntax, such as opcode-operand or algebraic with separate fields for parallel moves.
  - Parallel Move Support
    Discusses whether parallel moves are supported and whether they are operand-related or operand-unrelated, relevant restrictions, and the maximum number of parallel loads and stores that can be performed per instruction cycle.
  - Orthogonality
    Assesses the orthogonality of the processor's instruction set as judged by BDTI engineers and explains what makes the instruction set orthogonal or not.
  - Execution Time
    Describes the number of instruction cycles required for each class of instructions as well as the number of instructions that can be dispatched and executed per instruction cycle.
  - Instruction Set Highlights
    Summarizes noteworthy instructions that may ease programming or increase performance in certain applications.
- **Execution Control**
  Exploration of clocking, hardware looping, interrupts, stacks, and bootstrap loading.
  - Clocking
    Discusses processor's clock generation options.
  - Hardware Looping
    Explores hardware looping capabilities and how they affect performance.
  - Interrupts
    Lists interrupt sources, interrupt handling mechanisms, and interrupt latency.
  - Stack
    Explains stack implementation.
  - Bootstrap Loading
    Outlines how the processor begins executing when reset.

- **Peripherals**

  Discusses on-chip peripherals such as timers, serial ports, host ports, bit I/O, and more specialized peripherals.
- **On-Chip Debugging Support**

  Outlines on-chip debugging support, which can range from none to sophisticated on-chip emulation with support for IEEE-1149.1 (JTAG) scan-based debugging.
- **Power Consumption and Management**

  Discusses power consumption, supply voltage, and power management features particularly as they affect suitability for low-power applications. Unless otherwise noted, nominal (not minimum) voltages are used. Processors can operate at the indicated speeds with supply voltages at least 10% higher or lower than their nominal voltages.
- **Cost and Packaging**

  Summarizes price and packaging options available as of mid-2000.
- **Benchmark Performance**

  Discusses the most notable BDTI Benchmark results for the processor. Detailed benchmark analyses can be found in Chapter 8, *BDTI Benchmark™ Results.*
- **Fabrication Details**

  Describes the fabrication technology used to fabricate members of the processor family. If the processor is available as a core, this is also discussed.
- **Development Tools**

  Discusses availability and features of tools that can significantly ease development: For example, assemblers, compilers, linkers, library archivers, instruction-set simulators, emulators, and debuggers.
- **Applications Support**

  Discussion of vendor and third-party support, such as the availability of books or on-line tutorials, applications engineers, telephone hotlines, and websites.

We also identify processor variants available within the family. Additionally, we conclude each analysis with a summary of the processor's noteworthy advantages and disadvantages. These serve as convenient overviews of the analyses.

For readers seeking addition background on the terms and concepts discussed in this chapter, BDTI offers an introductory textbook, *DSP Processor Fundamentals.* Contact BDTI directly or visit BDTI's website (*www.BDTI.com*) for details. Chapter 8, *BDTI Benchmark™ Results* presents the results of the BDTI Benchmarks, which provide details on the performance of most of the processors discussed in this chapter.

The analyses presented in this chapter are organized alphabetically by manufacturer. The processor or family name is indicated within the header of each page for your convenience.

Chapter 5, *Processors Not Covered in This Report* gives information on where to find analyses of processors not included in this report, including analyses of the DSP capabilities of general-purpose processors.

7.1 Analog Devices ADSP-21xx Family

**BDTImark2000 Score:
230 at 75 MHz**

Introduction

Analog Devices' ADSP-21xx family consists of a large number of processors based on a common 16-bit, fixed-point conventional DSP architecture with a 24-bit instruction word. The fastest members of the family operate at 75 MHz at 2.5 volts, 52 MHz at 3.3 volts, and 40 MHz at 5.0 volts. ADSP-21xx processors are targeted at modem, audio, speech processing, and digital cellular applications.

The ADSP-21xx is the first DSP processor family from Analog Devices. The ADSP-2100A, the first member of the family, was introduced in 1986. Subsequently, Analog Devices has introduced many family members with various peripherals and speeds, and new members continue to be added to the family. Recent additions to the ASP-21xx family include new ADSP-218x members, such as the 2.5-volt ADSP-2186M operating at up to 75 MIPS. ADSP-21xx family processors are summarized in Table 7.1-1 and Table 7.1-2. In addition, Analog Devices offers specialized versions of ADSP-21xx devices with application-specific peripherals for such applications as motor control and speech processing. These specialized devices are not covered in this report.

In October 1999, Analog Devices announced the ADSP-219x, a new architecture derived from the ADSP-21xx. Section 7.2 provides an analysis of the ADSP-219x core and focuses on the differences between the ADSP-219x and the ADSP-21xx.

> *The ADSP-21xx family is notable for its dissimilar instruction and data word sizes (24 and 16 bits, respectively) and for its many family variants. Only a few of these family members (seven at present) execute at 50 MHz or higher, however.*
>
> *The top clock speed of the ADSP-21xx family has not increased since the last edition of* Buyer's Guide to DSP Processors *was published, in late 1998. It appears likely that Analog Devices will encourage its customers to migrate to the newer ADSP-219x architecture if they require performance beyond that of existing ADSP-21xx family members, rather than increase the speed of the older architecture.*

Architecture

The core architecture of the ADSP-21xx consists of a 16-bit, fixed-point data path, two data address generators, a program control unit, and separate program and data memories with dedicated buses. Figure 7.1-1 illustrates the architecture of the ADSP-2181.

Data Path

The ADSP-21xx data path consists of three separate arithmetic execution units: an arithmetic/logic unit (ALU), a multiplier/accumulator (MAC), and a barrel shifter. Each unit is capable of single-cycle execution, but only one of these units can be active during a single instruction cycle. Each unit has dedicated input registers and dedicated result registers. The ADSP-21xx implements a load-store architecture; each unit requires data to be loaded into input registers before computation.

Part	Core Voltage (V)	Max. Speed (MIPS)	On-Chip Memory		I/O Ports			Bit Oper-ations
			Prog. RAM/ ROM	Data RAM	Ser.	Host	Other	
ADSP-2100A	5.0	12.5	0/0	0	0	No	—	No
ADSP-2101	5.0	25.0	2Kx24/0	1Kx16	2	No	—	No
ADSP-2103	3.3	10.2	2Kx24/0	1Kx16	2	No	—	No
ADSP-2104	5.0	20.0	512x24/0	256x16	2	No	—	No
ADSP-2104L	3.3	13.8	512x24/0	256x16	2	No	—	No
ADSP-2105	5.0	20.0	1Kx24/0	512x16	1	No	—	No
ADSP-2109*	5.0	20.0	0/4Kx24	256x16	2	No	—	No
ADSP-2109L*	3.3	13.8	0/4Kx24	256x16	2	No	—	No
ADSP-2111	5.0	20.0	2Kx24/0	1Kx16	2	Yes	—	No
ADSP-2115	5.0	25.0	1Kx24/0	512x16	2	No	—	No
ADSP-2141L	3.3	40.0	16Kx24/0	16Kx16	2	No	—	No
ADSP-2161*	5.0	16.7	0/8Kx24	512x16	2	No	—	No
ADSP-2162*	3.3	10.2	0/8Kx24	512x16	2	No	—	No
ADSP-2163*	5.0	25.0	0/4Kx24	512x16	2	No	—	No
ADSP-2164*	3.3	10.2	0/4Kx24	512x16	2	No	—	No
ADSP-2165*	5.0	20.0	1Kx24/ 12Kx24	4Kx16	2	No	—	No
ADSP-2166*	3.3	16.7	1Kx24/ 12Kx24	4Kx16	2	No	—	No

TABLE 7.1-1. ADSP-21xx family variants (continued on next page).
* Minimum order quantity is 10,000.

The ALU and MAC units each have four input registers. If these registers are not needed to store operands for their associated arithmetic unit, they can be used as general-purpose registers. The output registers of any arithmetic unit may be used as inputs to any arithmetic unit. All ALU, MAC, and barrel shifter registers are shadowed by a set of secondary registers. The secondary registers can be switched in or out under program control in a single instruction cycle.

Shadowing of the data path registers is a very useful feature for context switching during subroutines and interrupt service routines.

The ALU operates on 16-bit data. It includes four input registers (two for each memory space, PM and DM), a feedback register (to return results to the input of the ALU

Part	Core Voltage (V)	Max. Speed (MIPS)	On-Chip Memory		I/O Ports			Bit Operations
			Prog. RAM/ ROM	Data RAM	Ser.	Host	Other	
ADSP-2171	5.0	33.3	2Kx24/0	2Kx16	2	Yes	—	Yes
ADSP-2173	3.3	20.0	2Kx24/0	2Kx16	2	Yes	—	Yes
ADSP-2181	5.0	40.0	16Kx24/0	16Kx16	2	IDMA	BDMA	Yes
ADSP-2183	3.3	52.0	16Kx24/0	16Kx16	2	IDMA	BDMA	Yes
ADSP-2184	5.0	40.0	4Kx24/0	4Kx16	2	IDMA	BDMA	Yes
ADSP-2184L	3.3	40.0	4Kx24/0	4Kx16	2	IDMA	BDMA	Yes
ADSP-2185	5.0	33.3	16Kx24/0	16Kx16	2	IDMA	BDMA	Yes
ADSP-2185M	2.5	75.0	16Kx24	16Kx16	2	IDMA	BDMA	Yes
ADSP-2185L	3.3	52.0	16Kx24/0	16Kx16	2	IDMA	BDMA	Yes
ADSP-2186	5.0	40.0	8Kx24/0	8Kx16	2	IDMA	BDMA	Yes
ADSP-2186L	3.3	52.0	8Kx24/0	8Kx16	2	IDMA	BDMA	Yes
ADSP-2186M	2.5	75.0	8Kx24/0	8Kx16	2	IDMA	BDMA	Yes
ADSP-2187L	3.3	52.0	32Kx24/0	32Kx16	2	IDMA	BDMA	Yes
ADSP-2188M	2.5	75.0	48Kx24/0	32Kx16	2	IDMA	BDMA	Yes
ADSP-2189M	2.5	75.0	32Kx24/0	48Kx16	2	IDMA	BDMA	Yes

TABLE 7.1-2. ADSP-21xx family variants (continued from previous). Note: IDMA = Internal DMA, BDMA = Byte DMA.

for further processing), and a result register. In addition to the usual ALU operations, the ALU provides increment/decrement, absolute value, and add-with-carry functions. ALU results are saturated upon overflow if the appropriate configuration bit is set by the programmer.

The fact that the ALU operates on 16-bit data is a disadvantage. Many 16-bit processors provide ALUs that operate on 32-bit data.

The MAC unit includes a 16 × 16 → 32-bit multiplier, four input registers (two for each memory space, PM and DM), a feedback register, a 40-bit adder, and a single 40-bit result register/accumulator providing eight guard bits. Unlike the ALU, the MAC unit does not have status bits to indicate if results are zero, positive, or negative (only an overflow flag is provided). MAC unit results must be passed through the ALU to obtain such status information. The MAC unit provides an optional automatic one-bit left shift of results to allow either fractional or integer multiplication. Besides signed operands, the multiplier can operate on unsigned/unsigned or on signed/unsigned operands, thus supporting multi-precision arithmetic.

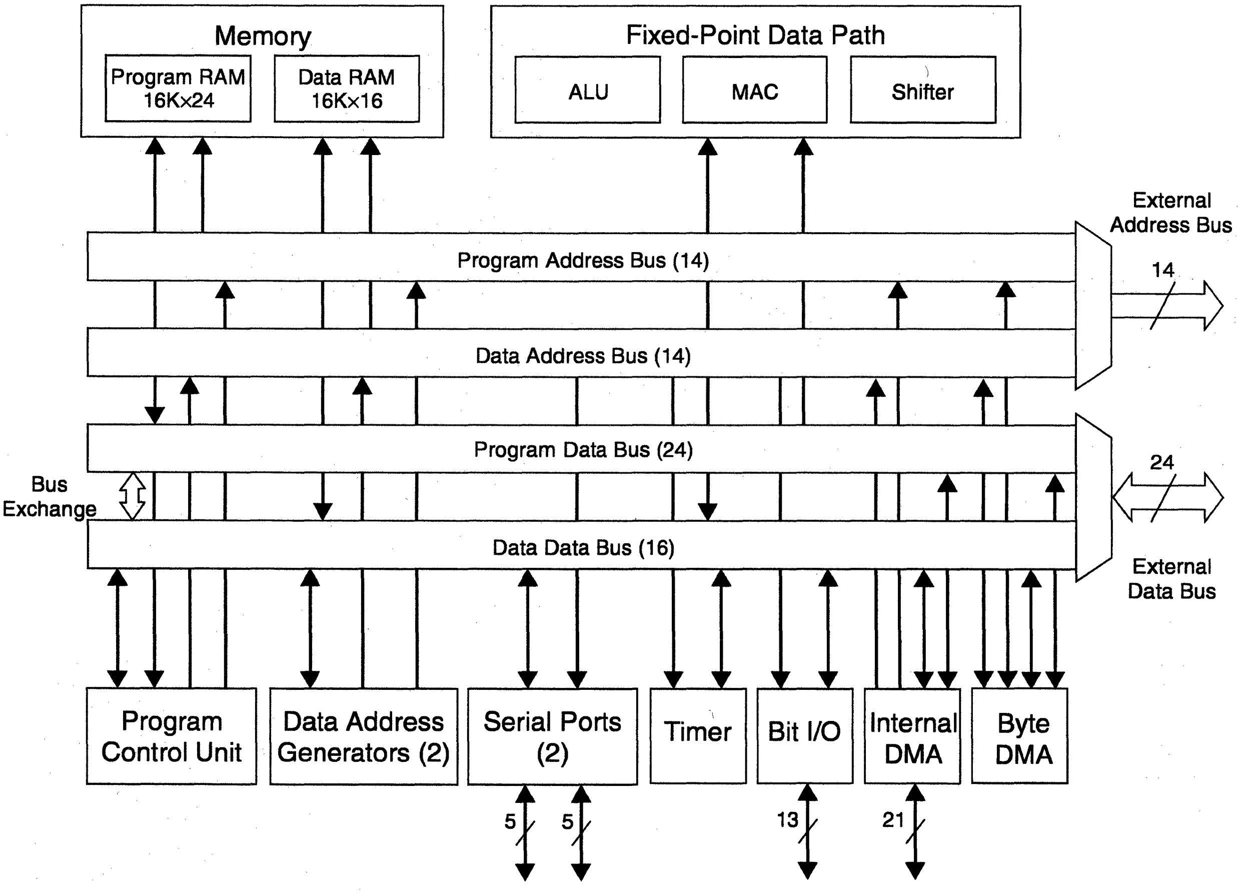


FIGURE 7.1-1. ADSP-2181 architecture.

The single accumulator is a limitation of the ADSP-21xx data path. It makes the processor poorly suited for arithmetic using complex numbers, for example. While the shadow register set provides a second accumulator, accessing it requires exchanging all arithmetic registers with their shadow registers. Thus, the shadow registers are more useful for context switching than as additional arithmetic registers for use within a single algorithm.

The fact that values must be transferred from the shifter or MAC unit to the ALU in order to set the status bits (for example, to test whether a result is negative) introduces extra overhead for some decision-making operations.

The 40-bit accumulator register can be addressed as two 16-bit and one 8-bit register, each of which can be individually copied to memory or to another register if extended precision is needed. The accumulator supports convergent rounding on bits 39-16 via dedicated instructions. In addition, round-to-nearest (biased) rounding is supported on the ADSP-217x and ADSP-218x.

A special instruction checks whether the value in the accumulator has overflowed and saturates the accumulator if overflow has occurred. Because the MAC unit does not contain a shifter, overflow is defined to occur whenever results fall outside the range of values that can be represented without the use of the accumulator guard bits (i.e., in the lower 32 bits of the accumulator). There is no overflow detection or saturation operation for the upper eight bits of the accumulator.

The fact that saturation requires an explicit instruction means that extra cycles are used in applications that must saturate results before moving them from the accumulator.

If the output of the accumulator is to be scaled before being stored to memory, the accumulator value must be passed to the barrel shifter unit 16 bits at a time. Typically, this means two shift operations must be performed, requiring a total of two instruction cycles.

The barrel shifter shifts 16-bit inputs from an input register or from the ALU/MAC/barrel shifter result registers into a 32-bit result register. Logical and arithmetic left and right shifts of up to 32 bits are supported. The shift count (number of bit positions by which the operand will be shifted) can be specified as immediate data in a shift instruction or by loading a value into the shifter control register.

Shift results can be merged (bitwise *or*'d) with previous shift results. This feature is necessary when using the shifter to scale accumulator results, since the accumulator must be scaled in two sections (or three sections in certain situations). This feature is also useful for packing and unpacking small bit fields into and out of a 16-bit word. The barrel shifter also supports block floating-point arithmetic via a block exponent detect instruction (which can be used iteratively to determine the maximum exponent of a block of data), plus single-word exponent detect, normalize, and exponent adjust instructions.

Compared to other fixed-point DSPs with barrel shifters, the ADSP-21xx barrel shifter is limited in that it can only accept 16-bit input operands, and thus, shifting 32- or 40-bit operands takes multiple instruction cycles.

The ADSP-21xx's support for multi-precision arithmetic somewhat makes up for the fact that the ALU and shifter can only process 16-bit inputs. The multiplier can multiply any combination of signed and unsigned integers or signed and unsigned fractional numbers, and the ALU supports add-with-carry and subtract-with-borrow operations.

Memory System

ADSP-21xx processors use a modified Harvard architecture with separate memory spaces and bus sets for program and data. All processors in the ADSP-21xx family, except the ADSP-2100A, include on-chip program RAM or ROM and on-chip data RAM. The ADSP-2100A has no on-chip memory; instead it has a 16-word instruction cache and two external bus sets. Please refer back to Table 7.1-1 and Table 7.1-2 for a summary of the various memory configurations and peripherals available in the ADSP-21xx family.

On-chip program memory can be used for both instructions and data. It is accessed via a 14-bit address bus and a 24-bit data bus. The program memory bus runs at twice the speed of the data memory bus to allow the fetching of a data operand and the next instruction in a single instruction cycle. The on-chip data memory is accessed via a 14-bit address bus and a 16-bit data bus. One access to the on-chip data memory can be performed in a single instruction cycle. Some ADSP-218x devices have more than 16 Kwords of on-chip program and data memory. Only 16 Kwords are directly user-accessible, however, with the remainder used to support the overlay capability described in the next section.

Three memory accesses (one instruction and one data operand from program memory, plus one data operand from data memory) can be performed in one instruction cycle (except on the ADSP-2100A, which has no on-chip memory). On a 75 MIPS ADSP-2189M, this results in a maximum sustainable on-chip data memory bandwidth of 150 million 16-bit words/second.

When a data word is read from the 24-bit program/data memory, the 16 MSBs are used to load the destination register, and the eight LSBs are automatically loaded to a dedicated PX register. Similarly, upon data writes to program memory, the 16 MSBs are written from the destination register and the eight LSBs come from the PX register. The eight-bit PX register can be read or written by the data path of the ADSP-21xx. This feature can be used to extend precision in some applications. It can also be used to improve memory efficiency when storing 16-bit data in 24-bit memory.

External Memory Interface

All ADSP-21xx processors except the ADSP-2100A have one external memory interface, providing a 14-bit address bus and a 24-bit data bus. This external interface is multiplexed between program and data memory accesses.

Programmed wait states (from zero to seven) can be separately specified for each of three regions of external data memory and for one region of external program memory. Externally requested wait states are supported only on the ADSP-2100A via the DMACK pin. External devices can request control of the processor's external memory interface through a bus request pin. Depending upon the value of a configuration bit, the processor may continue executing after relinquishing control of its external memory interface as long as no external memory accesses are required.

Assuming zero wait states, one external memory access can be performed in a single instruction cycle. The accessed data can be either a 24-bit instruction word or a 16-bit data word. For a 75 MHz ADSP-2189M, this results in a maximum sustainable external memory bandwidth of 75 million words/second for program or data.

The ADSP-2100A has two external memory bus sets, one for program memory with a 14-bit address bus and a 24-bit data bus, and one for data memory with a 14-bit address bus and a 16-bit data bus. The ADSP-2100A supports one access to each of the off-chip memory spaces per instruction cycle. Normally, one of these accesses is an instruction fetch, and the other is a data access. On the ADSP-2100A, when instructions are executed from the cache, two data accesses can be completed in a single instruction cycle using the two external bus sets.

The ADSP-218x adds several enhancements to the ADSP-21xx family's external memory interface, including program memory overlays, I/O address space, a "bus grant hung" pin, and two DMA ports. These enhancements are described below.

The ADSP-218x extends the ADSP-21xx family's 16 Kword program address space and 16 Kword data address space by supporting 8 Kword overlays. Using overlays, the programmer can map an 8 Kword segment of off-chip program memory into the upper half of the processor's program address space, replacing the upper 8 Kwords of on-chip program memory in the address map. Similarly, the programmer can map an 8 Kword segment of off-chip data memory into the lower half of the processor's data address space, replacing the lower 8 Kwords of on-chip data memory. Overlays are enabled via a configuration register. Up to two overlays can be provided for each memory space, although only one per memory space can be active simultaneously. During an external memory access, the MSB of the external address bus indicates which overlay is in use, if any. When a program memory overlay is in use, the upper 8 Kwords of on-chip program memory are not accessible. Similarly, when a data memory overlay is in use, the lower 8 Kwords of on-chip data memory are not accessible.

> *The ADSP-21xx family's 14-bit (16 Kword) address space is comparatively small and may be a serious constraint for some applica-*

tions. The ADSP-218x overlays partially overcome this shortcoming, but at the expense of programming complexity.

The ADSP-218x also adds an I/O address space of 2,048 16-bit words that can be used to access off-chip peripherals or other resources. The I/O space is accessed via dedicated read and write instructions. Programmed wait states (from zero to seven) can be separately specified for each of four regions of I/O space. Externally requested wait states are not supported. The $\overline{\text{IOMS}}$ pin is used to indicate that an I/O space access is underway.

The ADSP-218x also includes two DMA controllers. The IDMA ("internal DMA") allows an external device to access the processor's on-chip memory directly. The BDMA ("byte memory DMA") supports transfers to and from an external "byte memory space." These are discussed below in the section on peripherals.

Like other ADSP-21xx processors, the ADSP-218x provides a separate memory select output pin for each of its memory spaces (program memory, data memory, byte memory, and I/O memory). These pins indicate which memory space is being accessed during an external memory read or write. The ADSP-218x also provides a "composite memory select" ($\overline{\text{CMS}}$) pin that can be used to simplify designs where a single off-chip physical memory is mapped into multiple processor memory spaces. The $\overline{\text{CMS}}$ pin can be configured to be the logical *or* of any subset of the processor's four other memory space select pins. The $\overline{\text{CMS}}$ pin can then be used as a chip select signal for an external memory that is mapped into multiple memory spaces.

The ADSP-217x and ADSP-218x "bus grant hung" ($\overline{\text{BGH}}$) pin is asserted by the processor when it is ready to execute an instruction but cannot do so because the external bus has been granted to another device.

Address Generation Units

The ADSP-21xx supports immediate data and register-direct, memory-direct, and register-indirect addressing modes.

Register-indirect addressing makes use of two data address generators (DAGs). One DAG generates addresses only for data memory and supports bit-reversed addressing. The second DAG can generate addresses for program memory (which can also contain data) or data memory, but does not support bit-reversed addressing. Each data address generator contains four address registers. In addition, each data address generator has four buffer-length registers used for modulo addressing and four modifier registers that can be used to post-increment or decrement the address registers. Within a DAG, any of the four modifier registers can be used to post-increment any of the address registers. All DAG registers are 14 bits wide.

The fact that any of the four modifier registers can be used to post-increment any of the four address registers in each DAG provides flexibility for the programmer. However, the fact that the DAG does not provide built-in post-increment by -1, 0, and +1 means *that if these common post-increment values are needed, they must be loaded into modifier registers. Most other processors provide these post-increment values in hardware.*

The ADSP-21xx supports circular addressing via dedicated buffer-length registers associated with each address register. Thus, a maximum of eight circular buffers (four for the program and four for the data address space) can be active at a time. Circular addressing for any of the address registers can be enabled by loading its associated buffer length register with the size of the circular buffer. The starting address of a circular buffer must be aligned on a power-of-two boundary larger than the buffer size. The maximum size of a circular buffer is 16 Kwords. Circular addressing for an address register can be disabled by loading its buffer length register with zero.

Support for eight simultaneous circular buffers is superior to most other DSP processors. However, alignment requirements complicate the use of circular buffers.

The data address generator used for program memory data access can also be used for indirect jumps and subroutine calls. This is useful for reconfigurable interrupt handlers and jump tables.

Pipeline

The ADSP-21xx uses a two-stage instruction pipeline comprised of fetch/decode and execute stages. The pipeline is transparent and since instructions are executed on the cycle after they are fetched, delayed branches are not needed.

The shallow ADSP-21xx pipeline does not create challenges for the programmer. This is unusual among DSP processors.

Instruction Set

The ADSP-21xx instruction set and registers are summarized in Tables 7.1-3 and 7.1-4. ADSP-21xx instructions are 24 bits wide.

Assembly Language Format

The ADSP-21xx assembly language uses an algebraic syntax. Instructions are typically divided into one to three parts: a computation field and one or two data move fields. The computation field specifies an operation to be carried out by the ALU, MAC unit, or barrel shifter, and the data move fields specify a register-register data move or up to two register-memory data moves. For example, the instruction

```
MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M5);
```

multiplies the values in the MX0 and MY0 registers (the “SS” specifies that both multiply operands should be treated as signed numbers), adds the product to the value in the accumulator MR, and places the sum back into the accumulator MR. In parallel with this multiply-accumulate computation, two memory moves are performed. The value in the data memory ("DM") location pointed to by address register I0 is moved into MAC register MX0, and the value in the program memory ("PM") location pointed to by address register I4 is moved into MAC register MY0. After these moves are completed, address register I0 is post-incremented by adding the value in modifier register M1, and address register I4 is post-incremented by adding the value in modifier register M5. Note that the values of MX0 and MY0 used in the multiply-accumulate operation are *not* the values moved from memory in this instruction; rather, the values that were in the registers prior to the execution of these instruction fields are used.

Parallel Move Support

The ADSP-21xx supports operand-unrelated parallel moves, as illustrated by the preceding example instruction. MAC unit computation instructions support either two parallel data reads (one from program and one from data memory) or one parallel write (to data or program memory). Parallel moves are also allowed with virtually all unconditional

Class	Instructions
Arithmetic	Absolute value, add, add with carry, clear, increment, decrement, subtract, subtract with carry, negate, pass value, set ALU status (*ADSP-218x only*)
Multiplier-Accumulator	Multiply, multiply-accumulate, multiply-subtract (all support any combination of signed and unsigned operands), clear accumulator, saturate result, convergent round, square, square-accumulate, round-to-nearest (*ADSP-217x and ADSP-218x only*)
Logic	*And, or, exclusive-or, not*
Shifting	Arithmetic/logical shift left/right 1-32 bits, shift with *or*
Rotation	*None*
Conditional Execution	All arithmetic, logical, and shift instructions can be conditionally executed
Comparison	Compare (*ADSP-218x only*)
Looping	Single- or multi-instruction hardware loop
Branching	Conditional and unconditional branch, branch on pin set/clear
Subroutine Call	Conditional and unconditional call, call on pin set/clear
Bit Manipulation	Bit set, clear, test, toggle (*ADSP-217x and ADSP-218x only*)
Special Function	Scalar exponent detect, block exponent detect, normalize, division iteration

TABLE 7.1-3. ADSP-21xx instruction set summary. Underlined instructions are present only on some of the more recently introduced variants, as noted.

shifter and ALU operations. Alternatively, the ADSP-21xx allows a single register-register move in parallel with most computations.

The ADSP-21xx's support for register-to-register parallel moves is useful, especially because the MAC unit, ALU, and shifter use separate operand registers.

Unit	Registers	Width	Shadowed	Purpose
ALU	AX0, AX1, AY0, AY1	16 bits	Yes	Input registers
	AR	16 bits	Yes	Result register
	AF	16 bits	Yes	Feedback register
MAC	MX0, MX1, MY0, MY1	16 bits	Yes	Input registers
	MR2:MR0	40 bits	Yes	Accumulator; can be treated as two 16-bit registers (MR1, MR0) and one 8-bit register (MR2)
	MF	16 bits	Yes	Feedback register
Shifter	SI	16 bits	Yes	Input register
	SE	8 bits	Yes	Exponent register
	SB	5 bits	Yes	Block exponent register
	SR1:SR0	32 bits	Yes	Result register
DAG1	I0-I3	14 bits	No	Address registers
	L0-L3	14 bits	No	Circular buffer length registers
	M0-M3	14 bits	No	Modifier registers
DAG2	I4-I7	14 bits	No	Address registers
	L4-L7	14 bits	No	Circular buffer length registers
	M4-M7	14 bits	No	Modifier registers

TABLE 7.1-4. ADSP-21xx register summary. The processor can switch between primary and shadowed registers under program control.

Orthogonality

The ADSP-21xx 24-bit instructions provide reasonable orthogonality. The fact that the MAC unit, ALU, and shifter unit use separate register sets is the main factor detracting from the processor's orthogonality.

Execution Times

All ADSP-21xx instructions, including branches and subroutine calls, execute in a single instruction cycle, assuming no external memory wait states or multiple external memory accesses in a single instruction.

Uniform single-cycle instruction execution simplifies programming.

Instruction Set Highlights

The ADSP-21xx provides special support for the following operations:

- Division iteration
- Single-cycle scalar or block exponent detect to support normalization operations
- Conditional execution (depending on the ALU status flags) of most multiplier-accumulator, shifter, and ALU instructions (not available when using parallel moves)
- On the ADSP-217x and ADSP-218x only, bit set, clear, test, and toggle operations, plus square and square-accumulate instructions
- On the ADSP-218x only, compare and ALU status set operations

Execution Control

Clocking

The ADSP-210x, ADSP-211x, and ADSP-216x processors operate from a master clock running at the instruction execution rate. For example, a 20 MIPS ADSP-2101 uses a 20 MHz master clock. The ADSP-214x, ADSP-217x, and ADSP-218x operate from a 1/2-X master clock. For example, a 52 MIPS ADSP-2183 uses a 26 MHz master clock. On all but the ADSP-2100A, the master clock can be generated from an on-chip clock oscillator with the use of an external crystal, or an externally generated clock signal can be used. The master clock signal is available on an output pin of the processor.

Hardware Looping

The ADSP-21xx provides zero-overhead program looping through its DO instruction. Sequences of instructions of any length can be contained in a hardware loop, and up to 16,384 repetitions are supported. The loop exit condition can be a zero value in the special-purpose loop counter or arithmetic conditions generated by the ALU. The ADSP-21xx supports nesting of hardware loops up to four levels deep. A dedicated hardware stack is provided to store hardware loop variables for nested loops. Hardware loops are interruptible.

> *The ability of the ADSP-21xx to exit a hardware loop based on the value of an arithmetic condition is an advantage; some other processors provide conditional break instructions, but these require extra cycles to execute.*

Interrupts

Most ADSP-21xx processors provide up to three external interrupt lines. The pins for two of the external interrupt lines are shared with serial port pins, so if the serial port is used, only one external interrupt is available. On these processors, external interrupts can be configured as edge-sensitive or level-sensitive. The ADSP-218x provides six external interrupt lines: one edge-sensitive, two level-sensitive, and three that can be configured as edge-sensitive or level-sensitive. Of these six, two share pins with one of the serial ports.

Internal interrupt sources include the on-chip timer (all variants except the ADSP-2100A), serial ports (separate transmit and receive interrupts for each serial port on variants containing one or more serial ports), the host port (transmit and receive for variants with a host port), byte DMA controller (ADSP-218x only), and an interrupt that occurs prior to entering power-down mode (ADSP-217x and ADSP-218x).

Interrupts are individually maskable, prioritized and automatically nestable. Interrupts are prioritized by source. Interrupt and arithmetic status are automatically saved to a hardware status stack upon interrupt. The four-deep hardware loop stacks and sixteen-deep program counter stack allow nested interrupts to use zero-overhead loops without context save if the interrupted program does not use the hardware loop stacks in their entirety. Each interrupt source has its own vector that resides in low memory.

The ADSP-21xx responds to an interrupt by pushing the program counter onto the PC stack and the processor status (which includes the interrupt mask register) onto the status stack, then executing the first instruction at the corresponding interrupt vector location. If interrupt nesting is enabled, the processor sets the interrupt mask register so that lower-priority interrupts are masked, and higher-priority interrupts are not masked. If interrupt nesting is disabled, the processor sets the mask register so that all interrupts are masked. In all ADSP-21xx variants except the ADSP-2100A, each interrupt vector location contains four words of program memory, so that short interrupt service routines can reside entirely within the interrupt vector table with no need for a branch to reach the interrupt service routine.

Interrupt latency is five instruction cycles from the time when the external interrupt line is asserted to the execution of the first instruction at the interrupt vector location, assuming the processor is in an interruptible state. If a short interrupt service routine is used, then the first instruction at the interrupt vector location is the first instruction of the service routine. In other cases, this instruction is a branch to the interrupt service routine.

In such cases, the total latency to the execution of the first word of the interrupt service routine is six instruction cycles, assuming the processor is in an interruptible state.

Hardware loops are interruptible. Conditions that can delay interrupt processing include external memory wait states, external bus grant, and serial port autobuffer or DMA port cycle stealing (discussed below).

> *The ADSP-21xx interrupt latency is more predictable than those of most fixed-point DSPs, since hardware loops are interruptible and all instructions nominally execute in a single cycle.*

As mentioned above, the ADSP-21xx provides shadow registers for ALU, multiplier-accumulator, and shifter registers. Under program control, the processor can switch between the primary and shadow registers in a single instruction cycle.

The ADSP-21xx family supports the ability to clear or force an interrupt under software control, but not to test for the presence of an interrupt.

> *The ADSP-21xx has flexible and powerful interrupt handling mechanisms. In particular, the ability to execute four-instruction interrupt service routines without the need for a branch instruction, support for automatically nestable interrupts, and the availability of shadow registers are noteworthy.*

Stack

The ADSP-21xx provides four hardware stacks to allow loop, subroutine, and interrupt nesting. The PC stack is a 16-deep hardware stack for the program counter, used for subroutine calls and interrupts. In addition, separate hardware stacks are provided for the loop counter and for the end-of-loop address and loop termination condition, to support nestable hardware loops (discussed above). Finally, a fourth stack is provided for saving interrupt masks and status registers.

Bootstrap Loading

Upon reset, most ADSP-21xx processors can load an initial program into internal memory one byte at a time via the host port or via the external memory interface, which provides a separate boot memory space. An input pin selects the boot behavior. ADSP-21xx family members with program ROM begin execution at a fixed memory location. The ADSP-218x can boot from internal memory location 0, or can load a boot program via the BDMA or IDMA ports (discussed below).

The external boot memory space consists of eight pages, each comprised of up to 8K of eight-bit words. Upon reset, the processor loads a boot program from page 0 into on-chip program memory one byte at a time and executes it. Since addressing eight 8 Kbyte pages requires 16 address bits and the ADSP-21xx address bus is only 14 bits wide, during booting the two MSBs of the external data bus are used to provide the two most-significant bits of the address.

Accesses to boot memory are made using three or seven wait states by default, depending on the processor variant. After booting for the first time (upon reset), boot memory accesses can be configured under software control to use zero to seven wait states. After booting for the first time, the processor can be configured to reboot using one of the other seven boot memory pages.

Peripherals

The ADSP-2100A has no on-chip peripherals. All other family variants incorporate one or more serial ports, a timer, a bit I/O port, and in some cases a host port, and/or two DMA controllers. Refer back to Tables 7.1-1 and 7.1-2 for specific configurations.

- **Serial ports**

  With the exception of the ADSP-2105 (which has one serial port) and ADSP-2100A (which has none), ADSP-21xx family DSPs have two synchronous, bidirectional serial ports. These serial ports support word lengths from 3 to 16 bits. Each serial port has its own clock signal, but the transmit and receive sections of each serial port operate from the same clock. Separate transmit and receive frame sync signals are used for each direction of each port. Serial clock and framing pins can be configured as outputs if derived from the DSP's master clock via a programmable divider, or as inputs if these signals are generated by another device. Each port's built-in serial clock generator can be programmed to generate a serial clock by dividing the master chip clock by two, and then dividing again by a programmable 16-bit value. The maximum serial clock frequency is half of the processor's master clock rate. Thus the maximum serial port data rate for a 52 MIPS ADSP-2183 is 26 Mbits/second. However, if an external serial clock is used, the maximum clock rate is half of the processor's instruction cycle rate up to a maximum of 20 MHz.

  The serial ports offer an autobuffering mode, a kind of cycle-stealing DMA. In autobuffering mode, data is transferred between a serial port and memory without interrupts or explicit instruction execution. Instead, one set of data address generator registers (one address register, one circular buffer length register, and one modifier register) are used to manage a circular buffer in data memory. When the serial port is ready for a transfer (transmit or receive), the processor automatically moves the data word between the serial port and the circular buffer in data memory, suspending normal program execution for one instruction cycle (or more if wait states are required). Assuming zero wait states, one overhead cycle is incurred for each data word transfer. Interrupts are generated only when the end of the buffer is reached.

  The serial ports provide optional hardware µ-law or A-law companding. In addition, one of the two serial ports supports 24- or 32-channel time division multiplexed operation (except on the ADSP-2105).

Autobuffering is an elegant and efficient mechanism for serial port data transfer. The ADSP-21xx built-in companding support is also noteworthy.

In ADSP-21xx processors with serial ports, one of the serial ports can be disabled and its pins used instead to provide two external interrupt lines and two bit I/O lines (discussed below).

- **Bit I/O**

  Most ADSP-21xx processors provide one input and one output bit I/O pin, called "flags" by Analog Devices. The flag pins share the serial port 1 pins; i.e., these pins can be used for bit I/O when the serial port is not used. The bit I/O input can be used to control conditional jump and call instructions. The output can be set, reset, or toggled.

  The ADSP-2111, ADSP-214x, ADSP-217x, and ADSP-218x provide three additional bit I/O pins. These pins are separate from the serial port 1 pins and can only be used as outputs. In addition to the three output pins, the ADSP-218x provides eight bit I/O pins, which can be programmed either as inputs or outputs.

- **Timer**

  Except for the ADSP-2100A, all ADSP-21xx family processors incorporate a 16-bit interval timer. The timer clock is generated by dividing the master chip clock by the value stored in an eight-bit programmable linear prescaler register. The timer generates an interrupt when its 16-bit programmable counter reaches zero.

- **Host port**

  The ADSP-2111 and ADSP-217x processors include a host port interface. The host port can be configured to use 8- or 16-bit data, separate or multiplexed address and data buses, and a common read/write signal or separate read and write signals. The host port has six data registers and two control registers, each of which can be read or written by the DSP or the host processor. Host write interrupts are generated when the host CPU writes to a host port data register, and read interrupts are generated when the host CPU reads a host port data register. Status register bits indicate which data registers have been read from or written to by the host. In addition, the processor can be configured to boot from the host port.

- **IDMA port**

  The ADSP-218x provides an "IDMA" (internal DMA) port similar to the host port found on other ADSP-21xx variants, but which allows an external processor to read and write the ADSP-218x on-chip memory directly. The IDMA port consists of 16-bit multiplexed address/data bus and five control pins.

  Transfers are initiated by the external processor. The ADSP-218x then moves data between the IDMA port and on-chip memory via cycle-stealing DMA. Block transfers are supported, wherein the external processor specifies a starting address and then reads or writes a block of ADSP-218x on-chip memory via the IDMA port. Transfers to and from 24-bit internal program memory are performed in two successive steps. The processor can be configured to load its boot program via the IDMA port.

- **BDMA port**

  The ADSP-218x provides a "BDMA" (byte memory DMA) port. This port uses the processor's main external memory interface to transfer 8-, 16-, or 24-bit instructions or data into or out of the processor eight bits at a time. External accesses are made to a separate "byte memory space" consisting of 256 16-Kbyte pages.

  Since addressing 256 16-Kbyte pages requires 22 address bits and the ADSP-21xx address bus is only 14 bits wide, during BDMA port accesses the eight MSBs of the external data bus are used to provide the most significant bits of the address.

  Software on the ADSP-218x can initiate a BDMA transfer by loading a configuration register to specify the word width, direction, internal and external starting addresses, and number of words to transfer. The BDMA controller then moves data between the external memory interface and on-chip memory via cycle-stealing DMA. The processor can be configured to load its boot program via the BDMA port.

> *The ADSP-218x's two DMA ports are unusual for a low-cost fixed-point processor, and are likely to be helpful in simplifying system designs and reducing costs in some applications.*

On-Chip Debugging Support

Most ADSP-21xx family DSPs do not have scan-based debugging/emulation ports. The ADSP-218x does provide scan-based emulation support via a proprietary 13-pin emulation port. Through this interface the user may examine or modify DSP registers or memory and set hardware breakpoints. Boundary scan is not supported.

> *The lack of scan-based serial debugging/emulation ports on most ADSP-21xx processors complicates debugging in many situations and puts the ADSP-21xx at a disadvantage compared to most current fixed-point DSPs.*
>
> *While the functionality provided by the ADSP-218x debugging interface is useful, the use of a JTAG-compatible interface would have increased functionality and interoperability.*

Power Consumption and Management

ADSP-21xx family DSPs except for the ADSP-2100A provide a basic low-power mode, entered by executing the IDLE instruction. An unmasked interrupt awakens the processor.

Analog Devices quotes "typical" power consumption based on the processor spending 20% of its time executing IDLE instructions. The power consumption figures listed below are estimates that should reflect the power consumption of the processor when executing 100% DSP instructions. These estimates are derived from the typical power consumption and the power consumption of the processor executing IDLE instructions quoted by Analog Devices.

The estimated power consumption for the ADSP-2103, ADSP-2162, and ADSP-2164 is 33 mW at 10 MIPS and 3.3 volts.

The estimated power consumption for the ADSP-2101, ADSP-2105 and ADSP-2115 is 151 mW at 20 MIPS and 5.0 volts. The estimated power consumption of the ADSP-2161 and ADSP-2163 is 116 mW at 10 MIPS and 5.0 volts.

Power consumption for the 2.5-volt ADSP-2185M, ADSP-2186M and ADSP-2188M has not yet been characterized by Analog Devices. However, based on Analog Devices initial estimates, the power consumption for these devices will be approximately 113 mW at 75 MIPS and 2.5 volts.

The ADSP-217x and ADSP-218x support two additional low-power options. The first option, called "slow idle," places the processor in IDLE mode and also slows the on-chip master clock by a user-selectable factor of 16, 32, 64, or 128. According to Analog Devices, the power consumption for a 5.0-volt ADSP-2171 at 33.3 MIPS is 31 mW in slow idle mode when the user-selectable factor is 16.

The second option, called "power-down," shuts off all internal clocks and the internal clock oscillator. There is a 100-cycle wake-up latency upon exiting this power-down mode if the on-chip clock oscillator is not used. If the on-chip oscillator is used, wake-up latency is 4,196 cycles. Power-down mode is entered using a dedicated pin or by setting a configuration register. The processor can be awakened only via the dedicated pin or the reset pin. According to Analog Devices, the ADSP-21xx power consumption in power-down mode drops below 1 mW.

Benchmark Performance

The ADSP-218x has been benchmarked with the BDTI Benchmarks™. Overall benchmark results for all benchmarked processors are presented in Chapter 8, *BDTI Benchmark™ Results*. We summarize and analyze ADSP-218x benchmark performance in the paragraphs below. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not reflect the processor's instruction cycle rate; therefore, lower instruction cycle counts imply a more powerful architecture, but do not imply faster speed. Next we discuss benchmark execution times and cost-execution time products, indicating processor speed and cost-performance, respectively. We then discuss the processor's energy consumption, which reflects the energy consumed by the processor in order to perform a task. Finally, we discuss the processor's memory usage. We divide the memory usage discussion into three parts: Control benchmark memory usage, overall benchmark program memory usage, and benchmark data memory usage.

> *Note that instead of the entire ADSP-21xx family, the ADSP-218x is used in this analysis. The reason for this is that some of the BDTI Benchmarks use instructions only available on the ADSP-217x and ADSP-218x. Therefore the results presented here do not necessarily apply to other ADSP-21xx family members.*

Execution Performance

- **Instruction cycle counts:** As illustrated in Figure 8.1-13, the ADSP-218x total normalized instruction cycle count is roughly 30% higher than the average for benchmarked processors.

  On the **Vector Maximum** benchmark, the ADSP-218x has the second-highest instruction cycle count; the processor with the highest instruction cycle count is the ADSP-219x. The primary reason for these processors' high cycle counts is that neither the ADSP-218x nor the ADSP-219x support maximum (or minimum) instructions.

  The ADSP-218x has the lowest instruction cycle count among all benchmarked processors on the **Control** benchmark. Note, however, that the Control benchmark is optimized for minimum memory usage rather than for minimum cycle count. On this benchmark, the ADSP-218x instruction cycle count is about 25% lower than the average for benchmarked processors. The ADSP-218x achieves its low instruction cycle count on this control-oriented benchmark through the use of register-to-register parallel moves and single-cycle branches and subroutine calls.

  On the **Viterbi** benchmark, the ADSP-218x has the second-highest instruction cycle count. The ADSP-218x instruction cycle count is approximately twice as high as the average for the benchmarked DSP processors. The main reason for its high cycle count result on this benchmark is that the ADSP-218x lacks a shift-through-carry operation. This disadvantage costs many instruction cycles in the bit-interleaving section of this benchmark. In addition, due to the 16-bit width of the processor's barrel shifter, shifting 32-bit words requires two cycles.
- **Execution times:** The ADSP-2186M's somewhat slow instruction cycle rate of 75 MHz coupled with its higher-than-average instruction cycle counts, gives it a total normalized execution time that is almost twice as slow as the average for all benchmarked fixed-point DSP processors, as shown in Figure Figure 8.2-13. The 75 MHz ADSP-2186M has a BDTImark2000 score of 230.
- **Cost-execution time:** The low cost ($8.50) of the ADSP-2186M gives it the third-lowest total normalized cost-execution time product among all of the benchmarked processors (presented in Figure 8.3-13).
- **Energy consumption:** As presented in Figure 8.4-13A and B, the total normalized energy consumption for the ADSP-2186M is roughly 50% higher than the average

for benchmarked fixed-point DSP processors. The ADSP-2186M's relatively high power consumption (the lowest core voltage available in the ADSP-21xx family is 2.5 volts compared to less than 2.0 volts for many other fixed-point processors) combined with its slow execution time gives it poor energy consumption compared to many other fixed-point processors.

Memory Usage

The focus in the memory usage analysis is on Control benchmark memory usage. Unlike other benchmarks, the Control benchmark is optimized for minimum memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP code. Finally we discuss constant and non-constant data memory usage.

- **Control benchmark memory usage:** Despite its 24-bit instruction word width, the ADSP-218x achieves fair code density in control-oriented applications as indicated by the total Control benchmark memory usage in Figure 8.5-9. The total memory usage of the ADSP-218x on this benchmark is roughly 5% below the average for all benchmarked fixed-point processors.
- **Program memory usage:** As Figure 8.5-13 illustrates, the total normalized program memory usage of the ADSP-218x is roughly 40% lower than the average for all benchmarked fixed-point DSP processors. This result is partially explained by the fact that newer processors require loop unrolling in order to achieve optimal performance, which the ADSP-218x does not, in part because of its shallow pipeline. Also, VLIW processors often use multiple instances of the same instruction to make use of parallel execution units, resulting in larger programs—again, the ADSP-218x does not require this technique.
- **Data memory usage**: The ADSP-218x constant and non-constant benchmark data memory usage is as expected for 16-bit fixed-point DSP processors.

> *The ADSP-21xx is aging in comparison to state-of-the-art processors. The most recent major changes to the ADSP-21xx family consisted of reducing the core voltage and the pricing of some of the members; the highest clock speed available (75 MHz) hasn't increased in the past two years. As a result, the ADSP-218x doesn't achieve high scores for the execution time. ADSP-21xx family members are, however, good candidates for low cost applications where low processor cost and simplicity of assembly language programming are key elements.*
>
> *As indicated by the Control benchmark and the program memory usage analysis, the ADSP-218x achieves good code density despite its relatively wide instruction words.*

Cost

Price and packaging options for some ADSP-21xx family members are summarized in Table 7.1-5 through Table 7.1-9. Refer to Table 7.1-1 and Table 7.1-2, which appeared earlier in this section, for information on each family member's memory and peripheral configuration. Most ADSP-21xx processors are available in commercial, industrial, and military temperature grades. The prices presented here are for commercial temperature grade, except where indicated.

Fabrication Details

According to Analog Devices, the ADSP-210x, ADSP-211x, and ADSP-217x devices are fabricated in two-metal-layer 0.6 μm CMOS technology. The ADSP-2100A is fabricated in two-metal-layer 1.0 μm CMOS technology. The ADSP-2141L is fabricated in two-metal-layer 0.35 μm CMOS technology. ADSP-216x devices are fabricated in two-metal-layer 0.5 μm CMOS technology. ADSP-218x processors are fabricated in 0.5, 0.45, 0.35, or 0.25 μm CMOS technology depending on device.

Development Tools

Analog Devices provides "VisualDSP," an integrated development environment for the ADSP-21xx that includes an assembler, C compiler (C++ is also supported on recent versions of VisualDSP), linker, ROM splitter, and cycle-accurate instruction-set simulator. VisualDSP runs under Windows 9x or Windows NT on IBM PC-compatible computers.

> *VisualDSP is a quite sophisticated and programmer-friendly integrated development environment.*

Analog Devices offers versions of its EZ-LAB development hardware for some ADSP-21xx family processors: the ADSP-2101, ADSP-2111, and ADSP-2171. The EZ-LAB system is a stand-alone evaluation system enabling real-time application testing and debugging. The EZ-LAB board interfaces to the host computer via an RS-232 serial interface (the EZ-LAB board for the ADSP-2171 allows the board to be connected to the I/O bus of PC-compatible computers). The EZ-LAB board includes an external EPROM memory for booting, an analog interface, a serial interface, and an expansion connector for interfacing external devices.

The "EZ-KIT" starter package from Analog Devices includes the EZ-LAB evaluation board and application development software for PC-compatible computers.

Analog Devices offers EZ-ICE in-circuit emulators for ADSP-21xx processors. For all variants except for the ADSP-217x and ADSP-218x, the emulators are pod based; that is, they provide a connector that attaches to the target system in place of the ADSP-21xx processor. For the ADSP-217x, the emulator connects to the processor in the target board through a set of dedicated connectors that must be designed into the target system. These connectors provide access to many of the processor's pins. For the

Part	Speed (MIPS)	Voltage (V)	Package	Price (Qty. 10,000)
ADSP-2100A	12.5	5.0	100 PGA	$67.00
	12.5	5.0	100 MQFP	$60.00
	10.2	5.0	100 MQFP	$67.00
	10.2	5.0	100 MQFP	$56.00
	8.0	*5.0*	*100 PGA*	*$244.00*
ADSP-2101	25.0	5.0	68 PLCC	$22.68
	25.0	5.0	68 PGA	$34.02
	25.0	5.0	80 LQFP	$29.99
	20.0	5.0	68 PLCC	$19.72
	20.0	5.0	68 PGA	$29.58
	20.0	5.0	80 MQFP	$26.08
	16.7	5.0	68 PLCC	$16.42
	16.7	5.0	68 PGA	$24.62
	16.7	5.0	80 MQFP	$18.18
ADSP-2103	10.2	3.3	68 PLCC	$19.72
	10.2	3.3	80 MQFP	$22.68
ADSP-2104	20.0	5.0	68 PLCC	$4.99
ADSP-2104L	13.8	3.3	68 PLCC	$5.25
ADSP-2105	20.0	5.0	68 PLCC	$10.82
	13.8	5.0	68 PLCC	$10.38
ADSP-2109	20.0	5.0	68 PLCC	$8.55
ADSP-2109L	13.8	3.3	68 PLCC	$8.55

TABLE 7.1-5. ADSP-21xx price and package summary. Prices as of June, 2000. Italic numbers refer to parts only available in military or industrial temperature grades. (continued)

ADSP-218x, the emulator is scan based; it connects to the processor in the target system through the use of the chip's dedicated, proprietary 13-pin debugging interface. The target system must make these 13 signals accessible to the emulator.

The emulators use the same VisualDSP user interface as the instruction-set simulator. They allow the user to inspect and modify memory or registers. C language source-level debugging, single stepping, software breakpoints, and symbolic code disassembly are supported.

Part	Speed (MIPS)	Voltage (V)	Package	Price (Qty. 10,000)
ADSP-2111	20.0	5.0	100 PGA	$68.25
	20.0	5.0	100 MQFP	$31.68
	16.7	5.0	100 PGA	$60.38
	16.7	5.0	100 MQFP	$27.51
ADSP-2115	25.0	5.0	68 PLCC	$14.32
	25.0	5.0	80 MQFP	$16.47
	25.0	5.0	80 LQFP	$18.94
	20.0	5.0	68 PLCC	$12.45
	20.0	5.0	80 MQFP	$14.32
	20.0	5.0	80 LQFP	$16.47
	16.7	5.0	68 PLCC	$12.02
	16.7	5.0	80 MQFP	$14.66
ADSP-2141L	40.0	3.3	208 MQFP	$65.00
ADSP-2161	16.7	5.0	68 PLCC	$16.70
	16.7	5.0	80 MQFP	$19.20
	10.2	5.0	68 PLCC	$15.18
	10.2	5.0	80 MQFP	$17.45
ADSP-2162	10.2	3.3	68 PLCC	$15.18
	10.2	3.3	80 MQFP	$17.45

TABLE 7.1-6. ADSP-21xx price and package summary. Prices as of June, 2000. (continued)

Analog Devices offers an inexpensive EZ-KIT Lite evaluation board based on the ADSP-218x and bundled with an assembler, linker, and simulator. It connects to the host PC via an RS-232 serial interface. A Microsoft Windows-based monitor program is also provided for downloading programs and communicating with the ADSP-218x.

Applications Support

Analog Devices supports the ADSP-21xx with *The ADSP-2100 Family User's Manual*, data sheets, application handbooks, and a quarterly newsletter, *DSPatch*. A

Part	Speed (MIPS)	Voltage (V)	Package	Price (Qty. 10,000)
ADSP-2163	25.0	5.0	68 PLCC	$12.57
	25.0	5.0	80 MQFP	$14.45
	16.7	5.0	68 PLCC	$10.93
	16.7	5.0	80 MQFP	$12.57
	10.2	5.0	68 PLCC	$8.73
	10.2	5.0	80 MQFP	$10.04
ADSP-2164	10.2	3.3	68 PLCC	$8.73
	10.2	3.3	80 MQFP	$10.04
ADSP-2165	20.0	5.0	80 MQFP	$26.00
	16.7	5.0	80 MQFP	$26.00
ADSP-2166	16.7	3.3	80 MQFP	$27.00
ADSP-2171	33.3	5.0	128 MQFP	$27.31
	33.3	5.0	128 LQFP	$31.41
	26.0	5.0	128 MQFP	$24.58
	26.0	5.0	128 LQFP	$32.51
ADSP-2173	*20.0*	*3.3*	*128 MQFP*	*$24.94*
	20.0	*3.3*	*128 LQFP*	*$28.50*

TABLE 7.1-7. ADSP-21xx price and package summary. Prices as of June, 2000. Italic numbers refer to parts only available in military or industrial temperature grades. (continued)

Part	Speed (MIPS)	Voltage (V)	Package	Price (Qty. 10,000)
ADSP-2181	40.0	5.0	128 MQFP	$15.50
	40.0	5.0	128 LQFP	$17.00
	33.3	5.0	128 MQFP	$13.50
	33.3	5.0	128 LQFP	$15.50
	28.8	5.0	128 MQFP	$13.50
	28.8	5.0	128 LQFP	$13.50
ADSP-2183	52	3.3	128 LQFP	$16.69
	52	3.3	144 miniBGA	$21.00
	40.0	3.3	128 LQFP	$17.00
	33.3	3.3	128 LQFP	$15.50
	28.8	3.3	128 LQFP	$13.50
ADSP-2184	*40.0*	*5.0*	*100 LQFP*	*$7.50*
ADSP-2184L	*40.0*	*3.3*	*100 LQFP*	*$7.50*
ADSP-2185	33.3	5.0	100 LQFP	$17.00
	28.8	5.0	100 LQFP	$13.50
ADSP-2185L	52.0	3.3	100 LQFP	$18.70
	52.0	3.3	144 miniBGA	$22.57
	40.0	3.3	100 LQFP	$18.70
	33.3	3.3	100 LQFP	$15.50
	28.8	3.3	100 LQFP	$13.50
ADSP-2185M	75.0	2.5	100 LQFP	$8.25
	75.0	2.5	144 miniBGA	$10.75

TABLE 7.1-8. ADSP-21xx price and package summary. Prices as of June, 2000. Italic numbers refer to parts only available in military or industrial temperature grades. (continued)

World Wide Web page is available which allows users to access product information, data sheets, and applications notes (*http://www.analog.com/industry/dsp*).

Analog's user's manuals and application handbooks are thorough, clear, and well organized. The application handbooks present applications in a consistent, understandable way and include source code on diskette.

Part	Speed (MIPS)	Voltage (V)	Package	Price (Qty. 10,000)
ADSP-2186	40.0	5.0	100 LQFP	$13.00
	40.0	5.0	144 miniBGA	$15.75
	33.3	5.0	100 LQFP	$12.00
	28.8	5.0	100 LQFP	$11.00
ADSP-2186L	40.0	3.3	100 LQFP	$15.75
	40.0	*3.3*	*144 miniBGA*	*$13.00*
	33.3	3.3	100 LQFP	$12.00
	28.8	3.3	100 LQFP	$11.00
ADSP-2186M	75.0	2.5	100 LQFP	$6.00
	75.0	2.5	144 miniBGA	$8.50
ADSP-2187L	52.5	3.3	100 LQFP	$27.00
	40.0	3.3	100 LQFP	$23.50
ADSP-2188M	75.0	2.5	100 LQFP	$24.00
	75.0	2.5	144 miniBGA	$26.00
	66.7	*2.5*	*100 LQFP*	*$24.00*
ADSP-2189M	75.0	2.5	100 LQFP	$20.00
	75.0	2.5	144 miniBGA	$22.00
	66.7	*2.5*	*100 LQFP*	*$20.00*

TABLE 7.1-9. ADSP-21xx price and package summary. Prices as of June, 2000. Italic numbers refer to parts only available in military or industrial temperature grades.

> *Because of the proliferation of family variants, it is difficult to keep track of which variants include which features. An up-to-date summary document would be helpful in this regard.*

A telephone and fax hot-line is available between 8 AM and 5 PM Eastern time. Analog Devices DSP technical support group can also be reached via the Web or email. Applications engineers specializing in DSP support are available worldwide.

> *There is a moderate level of third-party support for the ADSP-21xx processors in the form of development boards, software function libraries, and application libraries.*

Advantages

- Rich instruction set
- Conditional instruction execution of most multiplier-accumulator and ALU instructions (when no parallel moves are used)
- Invisible pipeline
- Large number of family variants provide many performance, memory, and peripheral options
- Good multi-precision arithmetic support
- C-like assembly language
- Exponent detect and block exponent detect instructions
- Nestable, interruptible hardware loops (including single-instruction)
- Shadowing of arithmetic registers with single-level, single-cycle register set context switch
- Support for operand-unrelated parallel moves, including register-to-register moves; up to two parallel data reads per instruction cycle
- Large number of address registers (eight)
- Flexible use of modifier registers with address registers
- Conditional execution of most instructions
- Short interrupt latency
- Automatically nestable interrupts
- Two serial ports on most variants except ADSP-2100A, which has none, and ADSP-2105, which has one. All serial ports support autobuffering, TDM mode (except ADSP-2105), and μ-law/A-law companding
- Good documentation
- Flexible bootstrap modes on some family members
- Bit manipulation operations (ADSP-217x and ADSP-218x)

- Ability to use 1X externally generated clock (1/2-X on the ADSP-217x and ADSP-218x)
- On-chip DMA controller (ADSP-218x only)
- Low-cost family variants available (e.g., $7.50 for a 40 MIPS ADSP-2184, quantity 10,000)

Disadvantages

- Only one accumulator; complicates programming for many applications, particularly those that use complex numbers
- No shifter/limiter on MAC results to scale the 40-bit accumulator; shifting typically takes two instruction cycles
- ALU status bits not set by MAC or shifter; values must be passed through the ALU to set status bits
- No immediate post-increment capabilities
- Small address space (16 Kwords for program, 16 Kwords for data)
- Dissimilar program and data word sizes makes sharing off-chip memory between data and program spaces less efficient
- Larger instruction word than other 16-bit processors may increase system cost if external program memory is required
- No support for externally requested wait states (except for ADSP-2100A)
- Lack of a scan-based emulation port complicates debugging (except on ADSP-218x)
- Slow execution time results on the BDTI Benchmarks
- Poor energy consumption results on the BDTI Benchmarks

7.2 Analog Devices ADSP-219x Family

BDTImark2000 Score:
Not Available

Introduction

Analog Devices' ADSP-219x family is a 16-bit fixed-point conventional DSP processor family with 24-bit instructions. It is based on the ADSP-21xx family (described in Section 7.1, *Analog Devices ADSP-21xx Family*), but has a number of architectural enhancements. The ADSP-219x is mostly, but not completely, assembly source-code upward compatible with the ADSP-21xx. Compared with the ADSP-21xx, architectural changes on the ADSP-219x include the addition of new addressing modes, an expanded address space, an instruction cache, and a deeper pipeline.

Immediately prior to publication of this report (October 2000), ADI announced the first product based on the ADSP-219x architecture, the ADSP-2192. The ADSP-2192 contains two ADSP-219x cores, each of which will operate at 160 MHz, according to Analog Devices. Analog Devices states that the ADSP-2192 is currently sampling. Because BDTI has not yet verified the ADSP-2192 clock speed, however, there is no BDTImark2000 score currently available for this processor. Check BDTI's website (*www.BDTI.com*) for updated BDTImark2000 scores. This chapter focuses primarily on the ADSP-219x core and does not cover the ADSP-2192.

The ADSP-219x targets low-cost, low-power applications, such as motor control, and applications that demand high performance, such as cellular base stations and voice-over-IP gateways. For the latter application class, Analog Devices expects to offer chips containing multiple ADSP-219x cores.

> *With the ADSP-219x, ADI has enhanced the ADSP-21xx architecture to allow greater speeds while maintaining some software compatibility with the earlier ADSP-21xx architecture. With this combination, ADI hopes to leverage the success of the ADSP-21xx and expand the range of applications it can target with its fixed-point architectures.*

Because the ADSP-219x family is very similar to the ADSP-21xx family, this analysis covers only the differences between the ADSP-219x and the ADSP-21xx family. Readers should refer to Section 7.1, *Analog Devices ADSP-21xx Family*, for a full discussion of the ADSP-21xx architecture and for details of features that are identical between the two families.

Architecture

The ADSP-219x is based on the same 16-bit, fixed-point data path that is used in the ADSP-21xx. The ADSP-219x architecture includes two data address generators, a program control unit, an instruction cache, and a unified program/data memory with two buses. The architecture of the ADSP-219x is shown in Figure 7.2-1.

Data Path

The ADSP-219x data path consists of the same three separate arithmetic execution units used in the ADSP-21xx data path: an arithmetic/logic unit (ALU), a multiplier/accumulator (MAC), and a barrel shifter. Each unit is capable of single-cycle execution, but only one of these units can be active during a single instruction cycle.

The ADSP-219x includes sixteen 16-bit data registers. In contrast to the ADSP-21xx family, all of these registers can be used by all three execution units (with a few limitations for conditional ALU/MAC instructions). Although the registers are no longer dedicated to the individual execution units for which they are named, the register names are the same as those used in the ADSP-21xx to aid assembly code compatibility.

> *Allowing the execution units to share a common set of registers rather than dedicating specific registers to each execution unit eases assembly programming and compiler development, and is a significant improvement over the ADSP-21xx family.*

Outputs from the MAC unit can be deposited either in the 40-bit multiplier result register/accumulator, MR, or the 40-bit shifter output register, SR. The SR and MR registers are formed via a concatenation of three 16-bit registers, MR0-MR2 and SR0-SR2.

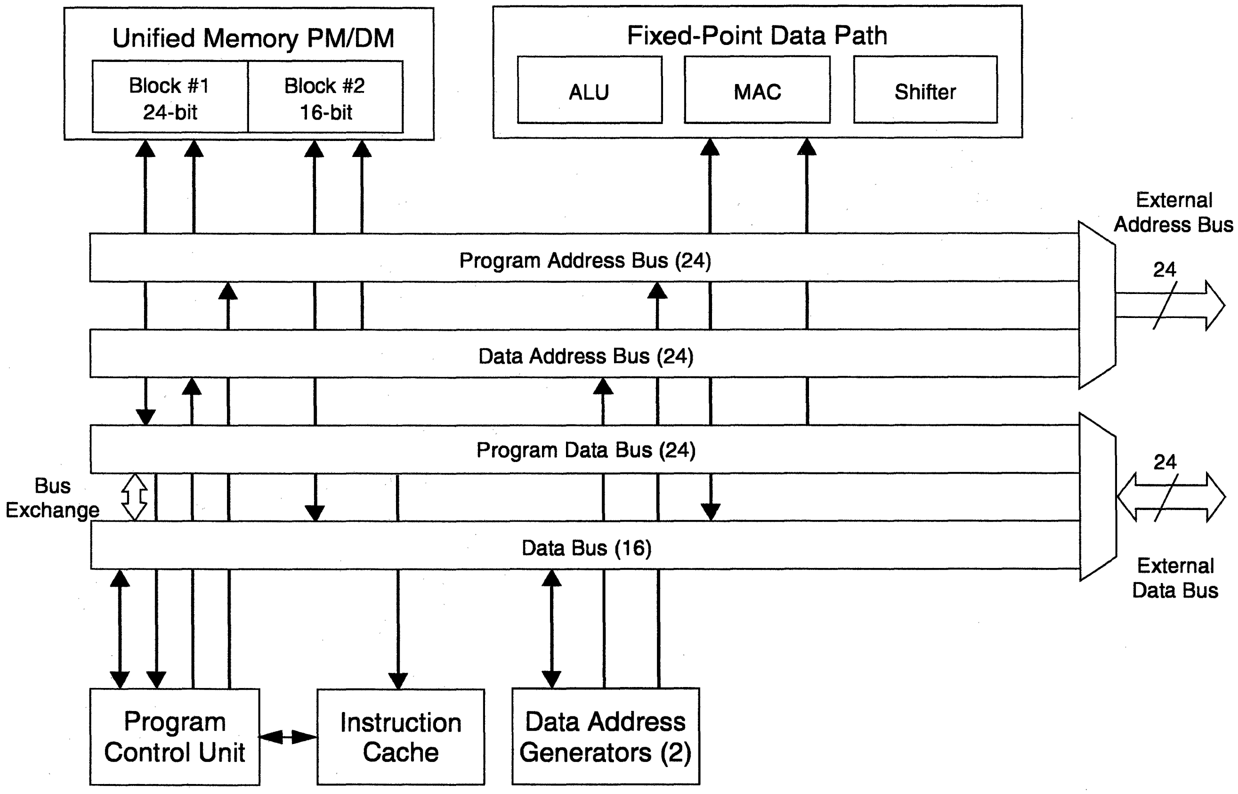


FIGURE 7.2-1. ADSP-219x architecture.

The lower 32 bits of a 40-bit multiply-accumulate result are stored in registers MR1 and MR0 or SR1 and SR0, and the higher 8 bits are stored in the lower 8 bits of registers MR2 or SR2. The ADSP-21xx includes a feedback register, MF, for the MAC unit; this register has been eliminated in the ADSP-219x. The functionality of the MF register has been replaced with the SR register.

> *A significant disadvantage of the ADSP-21xx was its single accumulator, which complicated programming and made the processor ill suited for arithmetic using complex numbers. By extending the SR register and allowing the ADSP-219x MAC unit to use this register as an additional accumulator, ADI has addressed this deficiency.*
>
> *The elimination of the MF register is one reason for the lack of complete assembly source code compatibility between the ADSP-21xx and ADSP-219x; software that uses the MF register will need to be rewritten to execute on the ADSP-219x by using the SR register instead.*

The barrel shifter functionality remains unchanged from that on the ADSP-21xx, but the size of the result register has been increased from 32 bits to 40 bits. This extended width is accommodated in the lower 8 bits of a new 16-bit data register, SR2. SR2 can also be used as a general-purpose 16-bit register.

> *The ADSP-219x barrel shifter remains limited in that it can only accept 16-bit input operands; thus, shifting 32- or 40-bit operands requires multiple instructions and instruction cycles.*

Memory System

Unlike the ADSP-21xx, the ADSP-219x does not segregate program and data memory. The ADSP-219x memory system provides a unified, word-addressable address space which contains instructions and data. The internal and external memory is physically organized as 256 pages of 64 Kwords each. Each memory page contains a block of 24-bit memory that can contain instructions or data, and a block of 16-bit memory that can contain only data. The distribution of memory between 16-bit and 24-bit memory will vary depending on the family member.

> *The address space has been extended from 14 bits to 24 bits, providing for considerable additional memory in comparison to the earlier ADSP-21xx family. The unified address space simplifies programming.*

The ADSP-219x provides a 64-word instruction cache. The ADSP-219x cache is similar to the ADSP-2106x cache, in that it only caches instructions that access data memory using the PM buses. When an instruction is cached on the ADSP-219x, the number of cycles required to execute the instruction is the same as on the ADSP-21xx. If the instruction is not cached, the ADSP-219x will require an additional cycle to execute the instruction. Since the instruction cache is similar to the one used in the Analog Devices' floating-point ADSP-2106x family, readers should refer to *Memory System* in Section 7.3 for details.

> *The ADSP-219x instruction cycle allows the processor to achieve two data memory accesses per cycle in many cases, without requiring double-clocked memory as used in the ADSP-21xx. This helps the ADSP-219x achieve a significantly higher clock rate than that of the ADSP-21xx.*

External Memory Interface

The ADSP-219x supports one external memory interface. The characteristics of the external memory interface are the same as those of the ADSP-21xx external memory interface, except that the address bus has been extended from 14 bits to 24 bits.

Address Generation Units

Like the ADSP-21xx, the ADSP-219x has two data address generators, DAG1 and DAG2. Each address generator handles four address pointers (also called index registers) and four modifier registers (I0-I3 and M0-M3 for DAG1, and I4-I7 and M4-M7 for DAG2). Any index register can be used with any modifier register from the same DAG. In contrast to the ADSP-21xx, DAG1 is no longer limited to generating addresses for data memory only; both DAGs on the ADSP-219x can generate data and program memory addresses. Note that the ADSP-219x instruction set still uses the terminology "PM" and "DM" but these names are strictly semantic and no longer indicate a true distinction among memory areas.

To support the new 24-bit address size, the existing DAG registers have been extended from 14 bits to 16 bits, and two new 8-bit registers (called DMPGs, or "page registers") have been added to hold the upper 8 bits of the 24-bit address. The existing index registers hold the lower 16 bits of the address.

> *The fact that the memory addressing is not linear but paged makes moving among pages awkward. However, the large size of each page (65,536 16- or 24-bit words) mitigates this drawback.*

The ADSP-219x supports several new addressing modes relative to the ADSP-21xx. The new addressing modes are listed below.

- **Indexed addressing.**
  In indexed addressing operations, the sum of the address register and the modify register provides the address of the memory access. The value of the index register is not permanently changed. For example,

```
AX0=DM(I0+M0);          // AX0=@(I0+M0); I0 not modified
```

- **Address post-increment with immediate data.**
  The address of a data word transferred between memory and registers over the DM bus or the PM bus can be post-modified with an 8-bit immediate two's complement offset value. However, instructions that perform a data transfer in parallel with an ALU or MAC operation (a common combination in DSP algorithm kernels) do not support an immediate 8-bit address offset.

  > *The addition of support for immediate modify operations is an improvement over the ADSP-21xx. The programmer is no longer required to use the modify registers (M0-M7) for modification values such as -2, -1, 0, +1, or +2, which are commonly used in DSP algorithms. Unfortunately, immediate modify operations are not supported for data transfers that are executed in parallel with ALU or MAC operations, a significant limitation in the usefulness of this feature.*
  >
  > *The added support for immediate modify operations did not significantly improve the efficiency of BDTI Benchmarks when they were ported from the ADSP-21xx to the ADSP-219x.*

- **Write immediate data to memory.**
  The ADSP-219x adds support for a 24-bit immediate data write to memory.

The support for modulo addressing on the ADSP-219x is similar to that on the ADSP-21xx. Each of the eight address registers can be independently used for linear or modulo addressing, and the maximum size of a circular buffer is still 16 Kwords. A set of eight memory-mapped base registers, B0-B7, has been added to the ADSP-219x. Before using a circular buffer with a given address pointer, the programmer must initialize the corresponding base register to the circular buffer starting address in addition to initializing the modulo buffer length register (L0-L7), as on the ADSP-21xx. There is no restriction on the starting address; i.e., no alignment is required on the ADSP-219x for circular buffers.

> *Support for eight simultaneous circular buffers is more than is available on most other DSP processors. The addition of base registers significantly simplifies the use of circular buffers on the ADSP-219x in comparison to the ADSP-21xx.*

Pipeline

In order to achieve higher clock speeds, the three-stage instruction pipeline of the ADSP-21xx family has been extended to six stages: look-ahead, pre-fetch, fetch, address decode, decode and execute stages. The pipeline is fully interlocked.

> *Increasing the pipeline depth allows higher clock speeds, but introduces instruction latencies that may affect software efficiency. Soft-*

ware developed for the ADSP-21xx may require re-optimization to eliminate pipeline stalls on the ADSP-219x.

Instruction Set

The ADSP-219x instruction set is almost identical to that of the ADSP-21xx, with a few modifications. The most important modifications to the instruction set are:

- **NEG and POS arithmetic conditions removed**
  The status conditions NEG (X Input Sign Negative) and POS (X Input Sign Positive) have been removed. The user must set CCODE to the appropriate value and check the SWCOND flag instead, as shown in Table 7.2-1.
- **MAC feedback register eliminated**
  The MF feedback register of the MAC has been eliminated. Instructions that use this register must be rewritten to use the SR registers instead.

  For a complete listing of all instruction set modifications, readers should refer to the documentation provided by Analog Devices. A run-time option of the ADSP-219x assembler provides instruction diagnostics to help guide the user in converting ADSP-21xx software to run on the ADSP-219x.

  Despite ADI's effort towards assembly code compatibility between the ADSP-219x and the ADSP-21xx family, some software modifications may be required in order to port ADSP-21xx assembly code to the ADSP-219x. In reviewing the BDTI Benchmark implementations for the ADSP-21xx, however, we noted that none of the benchmarks used NEG, POS, or the MF register. Hence, it is likely that the removal of these features from the ADSP-219x instruction set will have a limited impact when porting ADSP-21xx code to the ADSP-219x.
- **New instructions**
  The ADSP-219x includes several new instructions that were not available on the ADSP-21xx. Data transfer instructions have been enhanced to support new addressing modes, as described in the *Address Generation Units* section.

ADSP-21xx Syntax	ADSP-219x Syntax	Description

AX0=DM(I0,M0);
IF POS AR=AX0+AY0;
	CCODE = 0x08;
	AX0=DM(I0,M0);
	IF SWCOND AR=AX0+AY0;
		Set AR to AX0+AY0 if AX0>0

TABLE 7.2-1. Example of ADSP-219x new syntax for testing the sign of AX0.

The ADSP-219x also adds support for long branches and calls. These instructions support a 24-bit immediate value, which enables programs to access any location in the memory address space.

The ADSP-219x instruction set and registers are summarized in Table 7.2-2 and Table 7.2-3. Most instructions are 24 bits wide, but some of the new ADSP-219x instructions that support an indirect memory write of immediate data or conditional long branches or calls require an additional instruction word.

Class	Instructions
Arithmetic	Absolute value, add, add with carry, clear, increment, decrement, subtract, subtract with carry, negate, pass value, set ALU status
Multiplier-Accumulator	Multiply, multiply-accumulate, multiply-subtract (all support any combination of signed and unsigned operands), clear accumulator, saturate result, convergent round, square, square-accumulate, round-to-nearest
Data Transfer	Indirect 16/24-bit memory read/write with address pointer pre/post-modify, memory read/write with address-pointer immediate pre/post-modify, 16/24-bit immediate data write to memory
Logic	*And, or, exclusive-or, not*
Shifting	Arithmetic/logical shift of a 16-bit quantity left/right by 1-32 bits into a 40-bit result register, shift with *or*
Rotation	*None*
Conditional Execution	Most arithmetic, logical and shift instructions can be conditionally executed
Comparison	Compare
Looping	Single- or multi-instruction hardware loop
Branching	Conditional and unconditional non-delayed and delayed branch, branch on pin set/clear, conditional non-delayed absolute long branch
Subroutine Call	Conditional and unconditional non-delayed and delayed call, call on pin set/clear, 24-bit unconditional non-delayed absolute long call
Bit Manipulation	Bit set, clear, test, toggle
Special Function	Scalar exponent detect, block exponent detect, normalize, division iteration

TABLE 7.2-2. ADSP-219x instruction set summary. Underlined instructions are new or changed relative to the ADSP-21xx.

Assembly Language Format

The ADSP-219x assembly language format is mostly identical to the ADSP-21xx, which is described in Section 7.1. The assembler is now case-sensitive for label names and some assembler directives and operators have been modified. Table 7.2-4 illustrates these modifications. These syntax differences are handled by the ADSP-219x assembler when the code is assembled with the option "-legacy" (which forces the assembler to support old ADSP-21xx directives, rather than the ADSP-219x directives) and the option "-c" (which makes the assembler case-sensitive).

Parallel Move Support

Support for parallel moves is largely unchanged from that of the ADSP-21xx. There is, however, one new multifunction instruction that performs a register store in parallel with a register transfer.

Registers	Width	Shadowed	Purpose
DREG	16 bits	Yes	General-purpose input registers
AR	16 bits	Yes	ALU result register; general-purpose input register
AF	16 bits	Yes	Feedback register
MR2*, MR1, MR0	16 bits	Yes	Result registers. Can be combined to form a 40-bit accumulator using MR1, MR0, and 8 bits of MR2
_SR2_*, SR1, SR0	16 bits	Yes	Result registers. Can be combined to form a 40-bit accumulator using SR1, SR0, and 8 bits of SR2
SE*	8 bits	Yes	Exponent register
SB*	5 bits	Yes	Block exponent register
I0-I3, I4-I7	16 bits	Yes	Address registers
L0-L3, L4-L7	16 bits	Yes	Circular buffer length registers
M0-M3, M4-M7	16 bits	Yes	Modifier registers
_B0-B3, B4-B7_	16 bits	No	Circular buffer base registers
DMPG1-DMPG2	8 bits	No	Page registers (one for each address generator)

TABLE 7.2-3. ADSP-219x main register summary. The processor can switch between primary and shadow registers under program control. DREG represents any of these data registers: AX0-AX1, AY0-AY1, AR, MX0-MX1, MY0-MY1, MR0-MR2, SR0-SR2, SI. Underlined registers have been extended to 16 bits. Italicized registers have been added to the ADSP-219x. Registers marked with an asterisk can also be used as 16-bit general-purpose registers.

For example:

```
DM(I4+=M5)=I5, I5=I4;    // data transfer with post-increment
                         // and register transfer
```

Orthogonality

The ADSP-219x instructions provide reasonable orthogonality. The MAC, ALU, and shifter units use the same input registers, providing more flexibility and orthogonality compared to the ADSP-21xx family. There are still some limitations on which registers can be used as destination registers, however; for example, the AR register can be used as a destination only for ALU results.

Execution Times

With its six-stage instruction pipeline, the ADSP-219x does not retain the uniform single-cycle instruction execution of the ADSP-21xx. Branches and calls have a four-cycle latency if the branch/call is taken and a one cycle latency if not taken. DO UNTIL loops generally incur a three-cycle latency when entering the loop. There may be

ADSP-21xx syntax	ADSP-219x syntax	Description
.INIT cos: <x.dat>;	.INIT cos = "x.dat";	Load a file in memory

.VAR/DM/RAM group	.section/dm data;
	.VAR group;
		Allocate space to variables
		The ADSP-219x linker lets the user define code sections and place variables in the different sections

I1=^cos;	I1=cos;	Set I1 to point to array cos
L1=%cos;	L1=length(cos);	Set L1 equal to length of array cos
I0=h#02;	I0=0x02;	Set I0 to an immediate hexadecimal
0001 0002 0003	0x01 0x02 0x03	Content of an initialization file <file.dat> (prefix 00 has been replaced by prefix 0x)
.CONST N=256,M=10;	#define N 256 #define M 10	Constant definitions

.MACRO simple(%1);
I0=0x80; M0=5;
%1=DM(I0,M0);
.ENDMACRO;
	#DEFINE simple(x)\
	I0=0x80; M0=5;\
	x=DM(I0,M0);
		Macro definitions

TABLE 7.2-4. ADSP-219x assembler directive and operator syntax examples.

some additional latency (up to seven cycles) when the loop contains a multifunction ALU/MAC instruction in parallel with a dual read.

> *Software written for the ADSP-21xx that contains branch/call or DO UNTIL instructions will generally need to be reoptimized when converting to the ADSP-219x, due to differences in latencies. In addition, even after converting ADSP-21xx code to the ADSP-219x and hand-optimizing it again, code (with cache pre-loaded) generally requires more cycles to execute on the ADSP-219x than on the ADSP-21xx. For example, the hand-optimized BDTI Benchmarks take an average of 13% more cycles to execute on the ADSP-219x-C in comparison to the cycle counts of the ADSP-21xx versions.*

Instruction Set Highlights

ADSP-219x instruction set highlights are the same as those of the ADSP-21xx. Examples of new instructions available on the ADSP-219x are summarized in Table 7.2-5.

Execution Control

Clocking

The ADSP-219x operates from a 1X master clock. The master clock can be generated with the use of a crystal oscillator, a sine wave input, or an external system clock oscillator.

Example	Instruction description
DM(I4+=M5)=I5, I5=I4;	Data transfer in parallel with register transfer
AX0=DM(I0+M1); PM(I4+M6)=AX2	Read/write with pre-modify addressing
DM(I0,M0)=0x123456	Immediate 16/24-bit write

DMPG1=0x12;
I0=0x3456;M1=2;
AX0=DM(I0+=M1);
PM(I0+=M1)=AX2;
	24-bit read/write
	Set 8 bits of page register DMPG1 (DAG1)
	Set 16 bits of I0 and M1
	Read at address 0x123456
	Write at address 0x123458

IF GT JUMP 0x123456;
CALL LABEL1;
	Long branch/call

TABLE 7.2-5. Example of ADSP-219x specific instructions.

The ADSP-219x has a phase locked loop clock generator (PLL), which allows the input clock frequency to be multiplied by one of 63 different factors ranging from 1 to 31.

Unlike the ADSP-21xx, the ADSP-219x is fully static, meaning that the core retains its state when the clock is stopped.

Hardware Looping

Support for hardware looping on the ADSP-219x is similar to that on the ADSP-21xx. The deeper pipeline of the ADSP-219x complicates the use of the DO instruction, however. In general, the DO instruction incurs a three-cycle latency when entering the loop. If the loop contains a multifunction ALU/MAC instruction with a parallel dual read, however, the DO instruction will incur additional latency. Because of the relatively complex pipeline, the exact latency of the DO instruction is difficult to predict.

> *The difficulty in determining the exact latency of some instances of the DO instruction complicates programming, and forces the programmer to rely on simulation measurements to optimize software.*

Unlike on the ADSP-21xx, on the ADSP-219x there are a few restrictions on the instructions contained within a DO loop. The last or next-to-last instruction cannot be a delayed branch, and the last instruction in the loop cannot be a function call.

The ADSP-219x supports more levels of nested hardware loops than the ADSP-21xx; maximum nesting depth has been increased from four to eight levels. A dedicated hardware stack is provided to store hardware loop variables for nested loops. Hardware loops are interruptible. Up to 65,536 repetitions are supported.

As with the ADSP-21xx, the loop exit condition can be a zero value in the special-purpose loop counter or it can be based on arithmetic conditions generated by the ALU or multiplier-accumulator.

> *The ADSP-219x retains the ability to exit a hardware loop based on the value of an arithmetic condition. This is an advantage compared to some other processors, which provide conditional break instructions but require extra cycles to execute them.*

Interrupts

The ADSP-219x does not include an interrupt controller as part of the core. Interrupt controller functionality will be chip-specific, though some features will be common to all family members. The core itself supports up to twelve user interrupts. Each of the twelve user interrupts has a fixed priority (unlike the ADSP-21xx, whose interrupts were not prioritized). A chip-specific interrupt controller will allow the programmer to assign interrupts generated by peripherals to the twelve user interrupts, thus allowing indirect prioritization of each peripheral interrupt.

The interrupt controller assigns each priority level to a dedicated interrupt vector specifying the location in memory of its service routine. This assignment cannot be changed by the programmer; however, the programmer can change the priority of an interrupt (and thus change the interrupt's associated service routine) as mentioned above.

> *The fact that the priority determines the service routine is unusual among DSP processors. It complicates programming if interrupt peripheral priorities need to be changed during program execution.*

The ADSP-219x also supports four high-priority interrupts that are not user-assignable. These are the EMU interrupt (which causes a hardware reset), the PWDN interrupt (which turns off the core clock), the SSTEP interrupt (which causes the processor to execute a single instruction during emulation) and the STACK interrupt (generated when a stack overflow or underflow occurs). In total, the ADSP-219x supports up to sixteen automatically nested interrupts.

The ADSP-219x has a 19-cycle latency from the time an interrupt is received until the interrupt service routine begins execution.

The length of each interrupt vector location (in words) is chip-specific. ADI states that this length (as yet undisclosed) will be sufficient to enable short interrupt services routines to reside entirely within the interrupt vector table, with no need for a branch to reach the interrupt service routine. In addition, the ADSP-219x provides one set of shadow registers for both arithmetic and address registers—the ADSP-21xx only provides shadow registers for arithmetic registers—to support faster context switching. Table 7.2-3 shows which registers are shadowed.

> *The ADSP-219x has flexible and powerful interrupt handling mechanisms. In particular, the availability of shadow registers and the potential ability to execute short interrupt service routines without the need for a branch instruction are noteworthy.*

Stack

The ADSP-219x provides five hardware stacks to support loop, subroutine, and interrupt nesting. The PC hardware stack can store 33 24-bit words (compared to 16 24-bit words for the ADSP-21xx family) for the program counter, and is used for subroutine calls and interrupts. The loop counter store can stack eight 16-bit words, and holds the current values of the loop counters. Two separate hardware stacks, each containing eight 24-bit words, are provided for storing the start-of-loop and end-of-loop addresses to support nestable hardware loops (discussed earlier). Finally, a stack is provided for saving the interrupt masks and status registers upon interrupt.

Bootstrap Loading

Bootstrap loading features will vary depending on the family member.

Benchmark Performance

The ADSP-219x has been benchmarked with the BDTI Benchmarks™. Overall benchmark results for all benchmarked processors are presented in Chapter 8, *BDTI Benchmark™ Results*. We summarize and analyze ADSP-219x benchmark performance in the paragraphs below. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not reflect the processor's instruction cycle rate; therefore, lower instruction cycle counts imply a more powerful architecture, but do not imply faster speed. Next we discuss benchmark execution times, indicating processor speed. Finally, we discuss the processor's memory usage. We divide the memory usage discussion into three parts: Control benchmark memory usage, overall benchmark program memory usage, and benchmark data memory usage.

Because the ADSP-219x uses an instruction cache, the benchmark analysis presented in Chapter 8, *BDTI Benchmark™ Results*, treats the ADSP-219x as two distinct processor families. One family (referred to as the ADSP-219x) assumes that the cache is empty before each benchmark is executed. The other family (referred to as the ADSP-219x-C) assumes that the processor's instruction cache has been preloaded by executing the benchmark code once prior to measuring the instruction cycle counts for that benchmark.

For the execution time metric, we use the projected speed of 160 MHz provided by Analog Devices. We have not verified this clock speed on ADSP-219x hardware.

Execution Performance

- **Instruction cycle counts:** As illustrated in Figure 8.1-13, the ADSP-219x-C total normalized instruction cycle count is roughly 45% higher than the average for all benchmarked processors and is the highest (except for the ADSP-219x) among all of the processors benchmarked in this report.

  Since the pipeline has been extended, the ADSP-219x often has longer latencies than the ADSP-218x when using DO loops and branch instructions. Thus, the instruction cycle counts of the ADSP-219x and ADSP-219x-C are generally higher than those of the ADSP-218x. As a result, the ADSP-219x-C total normalized instruction cycle count is about 15% higher than that of its predecessor, the ADSP-218x. The ADSP-219x has cycle counts roughly 5% higher overall than the ADSP-219x-C. Most of this difference is a result of non-cached instructions that perform dual memory accesses and incur a one-cycle penalty the first time they are executed. Refer to *Memory System* for details of the cache operation.

  On the **LMS** benchmark, the ADSP-219x-C cycle count is significantly higher than that of its predecessor, the ADSP-218x. This is largely due to the multiple cycles required to launch a hardware loop on the ADSP-219x, and to the fact that some instructions that execute in one cycle on the ADSP-218x require multiple cycles on the ADSP-219x.

On the **Vector Maximum** benchmark, the ADSP-219x-C has the highest cycle count, twice as high as the average. This high cycle count is mostly due to the lack of a maximum instruction, which is offered on most current DSP processors. Also, the ADSP-219x benchmark implementation, unlike that of the ADSP-218x, does not use a conditional branch instruction in the inner loop, since branches on the ADSP-219x incur a high cycle count penalty. Instead, the ADSP-219x implementation replaces the conditional branch with conditional execution of several instructions that immediately follow the (removed) branch. This results in an inner loop that requires more cycles than that of the ADSP-218x (because the conditional instructions each consume a cycle regardless of the outcome of the condition), but fewer than would be required by using a conditional branch instruction.

On the **Control** benchmark, on which the older ADSP-218x has the lowest cycle count (tied with the SC140), the instruction cycle count for the ADSP-219x-C is roughly 40% higher, and is roughly equal to the average. This is mainly a result of the deeper pipeline on the ADSP-219x, which causes branch instructions to incur multiple cycle penalties.

On the **Viterbi** benchmark, the ADSP-219x-C has the second-highest instruction cycle count (the same as that of the ADSP-218x). The ADSP-219x instruction cycle count is approximately twice as high as the average for the benchmarked processors. The main reason for this high cycle count is that the ADSP-21xx and ADSP-219x processors lack a shift-through-carry operation. This disadvantage costs many instruction cycles in the bit-interleaving section of this benchmark. In addition, due to width of the processors' barrel shifter (16 bits), shifting a 32-bit word requires two cycles.

- **Execution times:** The ADSP-219x's relatively fast instruction cycle rate of 160 MHz compensates for its high instruction cycle counts and results in a projected total normalized execution time that is roughly equal to the average of all benchmarked fixed-point DSP processors. The ADSP-219x total normalized execution time is also almost twice as fast as that of the ADSP-2186M. This is presented in Figure 8.2-13.
- **Cost-execution time:** At the time of BDTI's benchmark analysis, no pricing information has been disclosed for ADSP-219x products. Thus, the ADSP-219x is excluded from this metric.
- **Energy consumption:** At the time of BDTI's benchmark analysis, no power consumption data had been disclosed for ADSP-219x products. Thus, the ADSP-219x is excluded from this metric.

Memory Usage

The focus in the memory usage analysis is on Control benchmark memory usage. Unlike other benchmarks, the Control benchmark is optimized for minimum memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP code. Finally we discuss constant and non-constant data memory usage.

- **Control benchmark memory usage:** Despite its relatively wide 24-bit instruction word width, the ADSP-219x achieves reasonable code density in control-oriented tasks as indicated by the processor's total memory usage on the Control benchmark shown in Figure 8.5-9A. The total memory usage of the ADSP-219x on this benchmark is roughly 15% above the average for all benchmarked fixed-point processors.
- **Program memory usage:** As Figure 8.5-13 illustrates, the total normalized program memory usage of the ADSP-219x is roughly 40% lower than the average for all benchmarked fixed-point DSP processors. The average is somewhat skewed by the fact that some newer processors such as the TMS320C64xx require loop unrolling in order to achieve optimal performance and have very high total normalized program use results. Compared to processors with 24-bit instructions, the ADSP-219x's total normalized program memory usage is roughly average. Note that since ADSP-218x code sometimes needs to be unrolled and further modified tooptimize it for the ADSP-219x, the ADSP-219x code is roughly 10% larger than that of the ADSP-218x even though the two instruction sets are very similar.
- **Data memory usage:** The ADSP-218x constant and non-constant benchmark data memory usage is roughly average for a 24-bit fixed-point DSP processor.

> *The relatively simple architecture of the ADSP-219x results in good code density; for example, extensive loop unrolling is not required compared to other more complex DSP processors. However, the extension of the pipeline makes it necessary to rewrite some ADSP-21xx code in order to obtain optimized ADSP-219x code. The delays caused by the pipeline give the ADSP-219x higher cycle count results than those of the ADSP-218x. However, the much faster projected instruction cycle rate of the ADSP-219x (160 MHz compared to 75 MHz for the ADSP-218x) provides the ADSP-219x with a projected execution time roughly twice as fast as the ADSP-218x, and average in comparison to other fixed-point DSP processors.*

Development Tools

Analog Devices has released the Version 7.0 of its VisualDSP development tool suite. Version 7.0 supports the ADSP-21xx and the ADSP-219x, and includes an assembler and a linker, a C and C++ compiler, a ROM splitter, and a cycle-accurate instruction-set simulator. VisualDSP 7.0 runs under Windows 95 OSR B, Windows 98, Windows NT 4.0, Windows 2000, and Solaris.

The ADSP-219x simulator (beta release) marks each instruction that is in the pipeline with a symbol indicating the pipeline stage.

Making the different pipeline stages visible allows the programmer to more easily determine the causes of pipeline stalls.

According to Analog Devices, the C compiler libraries provided with the ADSP-219x will include libraries already available with the ADSP-218x compiler. The main libraries are the math library (trigonometric functions, FFTs, filters, etc.) and the signal library (interrupt, SPORT, system interfaces).

Applications Support

The key documentation for the ADSP-219x includes the *Instruction Set Reference Manual* and the *ADSP-219x Core Manual.*

The main documentation for the ADSP-219x family consists of the *ADSP-219x Instruction Set Reference Manual* and datasheets for individual processors. In addition, Analog Devices offers documents on porting software from the ADSP-21xx to the ADSP-219x and on optimizing software for the ADSP-219x.

A telephone and fax hotline is available between 8 AM and 5 PM Eastern time. Analog Devices' DSP technical support group can also be reached via the website or email. Applications engineers specializing in DSP support are available worldwide.

Advantages

- Rich instruction set
- Conditional instruction execution of most multiplier-accumulator and ALU instructions (when no parallel moves are used)
- Good multi-precision arithmetic support
- C-like assembly language
- Exponent detect and block exponent detect instructions
- Nestable, interruptible hardware loops (including single-instruction)
- Shadowing of arithmetic registers and address registers with single-level, single-cycle register set context switch
- Support for operand-unrelated parallel moves, including register-to-register moves; up to two parallel data reads per instruction cycle

- Flexible use of modifier registers
- Automatically nestable interrupts
- Good documentation
- Flexible bootstrap modes
- Compatibility with previous generation
- Ability to use externally generated clock with a wide variety of multiplier ratios

Disadvantages

- High latency for starting DO UNTIL loops
- No shifter/limiter on MAC results to scale the 40-bit accumulator; shifting typically takes two instruction cycles
- ALU status bits not set by MAC or shifter; values must be passed through the ALU to set status bits
- Dissimilar program and data word sizes makes sharing off-chip memory between data and instructions less efficient
- Larger instruction word than other 16-bit processors may increase system cost if external program memory is required
- The ALU supports only 16-bit operands. Many 16-bit processors provide ALUs that operate on 32-bit data

7.3 Analog Devices ADSP-2106x Family

**BDTImark2000 Score:
250 at 66 MHz**

Introduction

The ADSP-2106x "SHARC" is a family of 40-bit floating-point conventional DSP processors with 48-bit instruction words from Analog Devices. The ADSP-2106x is targeted at military, audio, voice recognition, imaging, and telephony applications, especially those requiring multiprocessor systems. The fastest members of the ADSP-2106x family run at 66 MIPS at 3.3 volts and 50 MIPS at 5.0 volts.

The first ADSP-2106x family members were introduced in September 1994. The ADSP-2106x is based on Analog Devices' earlier ADSP-21020 architecture and is object-code compatible with the ADSP-21020. Compared to the ADSP-21020, the ADSP-2106x adds up to 512 Kbytes of on-chip SRAM, an external memory interface designed for multiprocessor systems, a DMA controller, two serial ports, and up to six communications ("link") ports useful for multiprocessor configurations.

In 1998, Analog Devices introduced the next generation of the SHARC architecture, the ADSP-2116x. This processor is covered in Section 7.4, *Analog Devices ADSP-2116x Family*. Any code written for the ADSP-2106x can be re-assembled and executed without modification on the ADSP-2116x.

The ADSP-2106x is noteworthy for its powerful and orthogonal instruction set and shadowing of all arithmetic and address registers. The ADSP-21060, ADSP-21060L, ADSP-21062, and ADSP-21062L are also noteworthy for their large on-chip memory and multiprocessor system support. ADSP-2106x family members are summarized in Table 7.3-1. In the remainder of this analysis, "ADSP-21060" will be used to refer to both

Part	Max. Speed (MHz)	On-Chip Memory	Comments
ADSP-21060 ADSP-21060L	40	512 Kbytes	6 communication ports, host interface, timer, 10 DMA channels, 2 serial ports
ADSP-21061 ADSP-21061L	50	128 Kbytes	No communication ports, host interface, timer, 6 DMA channels, 2 serial ports
ADSP-21062 ADSP-21062L	40	256 Kbytes	6 communication ports, timer, host interface, 10 DMA channels, 2 serial ports
ADSP-21065L	66	68 Kbytes	No communication ports, host interface, 2 timers, 10 DMA channels, 2 serial ports

TABLE 7.3-1. ADSP-2106x family summary. On-chip memory sizes are listed in bytes since on-chip memory can be divided into 48-bit-wide program and 32-bit-wide data memory in many different configurations.

the ADSP-21060 and the low-power ADSP-21060L, except when the discussion relates to voltage or power consumption. Similarly, "ADSP-21061" will be used to refer to both the ADSP-21061 and the ADSP-21061L, and "ADSP-21062" will be used to refer to both the ADSP-21062 and the ADSP-21062L.

Analog Devices provides a multi-chip module, the AD14160, which includes four ADSP-21060 processors. The processors are connected to a 32-bit program bus and a 48-bit data bus that is extended off-chip. Peripherals include 16 link ports and 8 serial ports, which are connected to the module pins.

Architecture

ADSP-2106x processors are based on a common 40-bit floating-point data path, program control unit, bus structure, and I/O interfaces, but feature different amounts of on-chip memory and different peripheral configurations, as shown in Table 7.3-1. The on-chip serial ports, interprocessor communication ports, and DMA controller are grouped together in what Analog Devices calls an "I/O Processor." The ADSP-2106x uses a 32-bit address space and 48-bit instruction word.

The 48-bit instruction word is unusually large for a conventional DSP processor.

Figure 7.3-1 illustrates the ADSP-2106x architecture, exemplified by the ADSP-21060.

Data Path

As in Analog Devices' ADSP-21xx fixed-point DSP processors (discussed in detail in Section 7.1), the ADSP-2106x provides three distinct arithmetic units: a multiplier-accumulator, a shifter, and an ALU, all of which perform arithmetic and logical operations in a single instruction cycle.

The ADSP-2106x supports four data types: 40-bit IEEE floating-point, 32-bit IEEE-754 and IEEE-854 floating-point, 16-bit floating-point, and 32-bit fixed-point. A mode bit determines whether 32-bit or 40-bit floating-point is used. The ADSP-2106x provides nearly complete hardware support for IEEE-754 and IEEE-854 single-precision floating-point format and arithmetic; the ADSP-2106x additionally supports the IEEE-754 and IEEE-854 single-extended precision floating-point format (a 40-bit format that extends the single-precision mantissa by eight bits for added precision). A few special cases of the IEEE-754 and IEEE-854 standards are not supported by these processors. The 16-bit floating-point format, called "short float," can be used for storage but not computation. This format has a smaller range and lower precision than 32-bit or 40-bit floating-point formats. The ADSP-2106x includes instructions that can convert between 32- or 40-bit floating-point and short-float formats or between 32- or 40-bit floating-point and fixed-point formats in a single instruction cycle.

Unlike the ADSP-21xx, on the ADSP-2106x the multiplier-accumulator, the shifter, and the ALU all access a common register file containing sixteen 40-bit registers. Inputs to all arithmetic operations come from the register file, and results of all arithmetic operations are delivered to the register file. If only one arithmetic operation is performed in a given instruction, then any of the registers in the register file can be accessed. If multiple arithmetic operations are performed in a single instruction, each operation can access only a restricted group of registers for its input operands, but results can be deposited into any register.

In addition to the usual fixed- and floating-point arithmetic operations, the ALU supports average ([A+B]/2), minimum, maximum, compare, clipping (saturation where the limit is specified as an operand of the instruction), and simultaneous add/subtract ([A+B],[A-B]). The latter operation is useful for FFT implementations. A special shift reg-

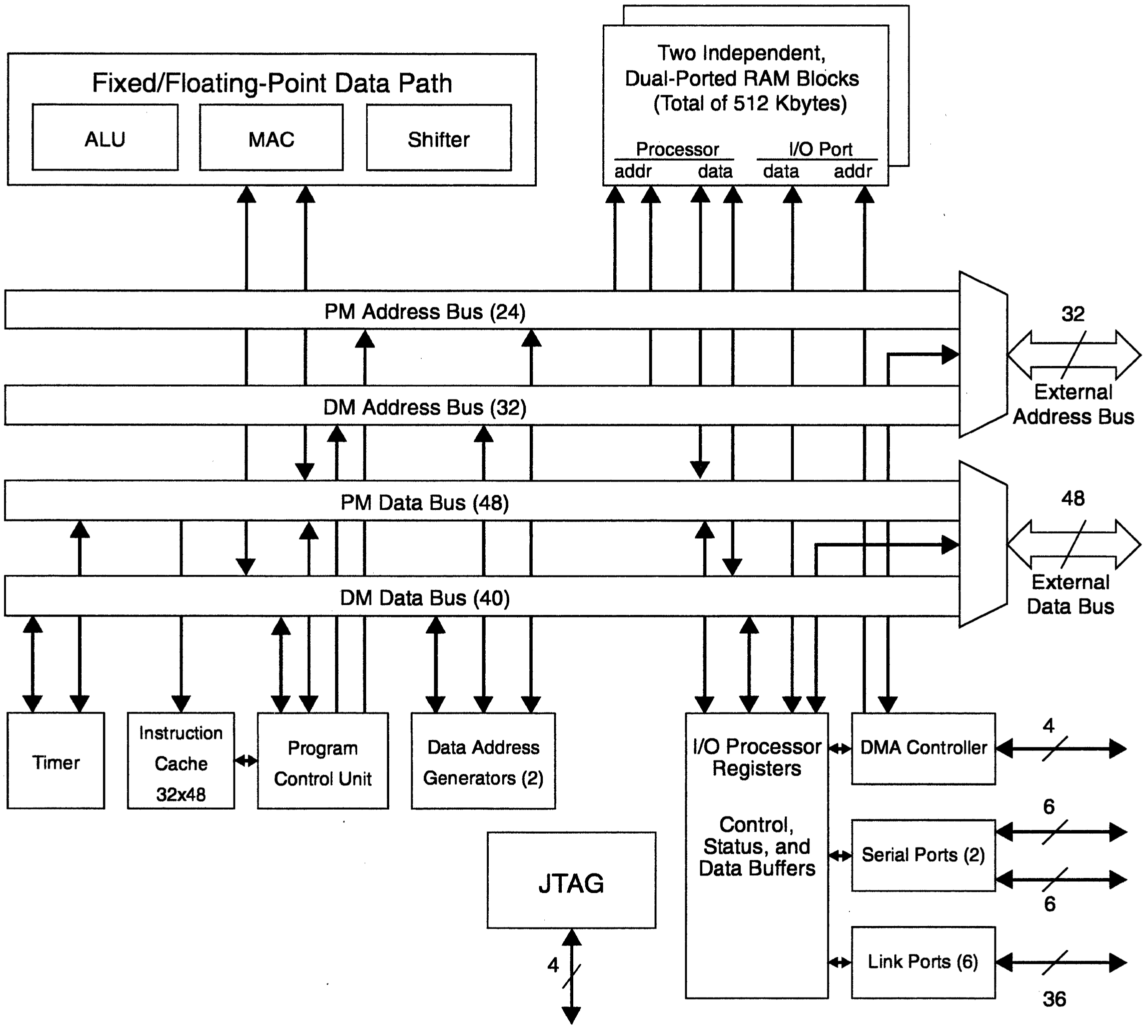


FIGURE 7.3-1. ADSP-21060 architecture.

ister stores the results of the previous eight compare operations, so that they can be inspected by a single instruction. Scaled conversions between 32-bit fixed-point and IEEE floating-point formats are also provided.

The ALU provides the following status flags dependent on the result of the last operation: zero, negative, overflow, underflow, invalid, carry, and floating-point (indicates whether the last operation was a floating-point or fixed-point operation). A sign flag is also provided and is set if the input operand is negative. The sign flag is set only by the ABS and MANT instructions.

The shifter unit performs single-bit manipulation, bit-field manipulation, rotation, and logical or arithmetic shifting operations. The shifter does not make use of the carry bit. Bit field instructions allow a group of bits to be extracted from one register and deposited into another position in another register using two instructions. Arithmetic, logical, and rotational shifting of registers of up to 32 bits left or right is supported. The shifter also supports single-cycle exponent detection.

The multiplier unit performs floating-point multiplication and fixed-point integer or fractional multiplication. Fixed-point integer and fractional operands may be signed/signed, unsigned/unsigned or signed/unsigned. These features provide support for multi-precision arithmetic. When operating on floating-point data, the multiplier unit performs $32 \times 32 \rightarrow 40$-bit or $40 \times 40 \rightarrow 40$-bit floating-point multiplications. When operating on fixed-point data, the multiplier unit performs $32 \times 32 \rightarrow 64$-bit multiplications and provides an 80-bit accumulator sub-unit (which yields 16 guard bits). The accumulator is used only for fixed-point operations; the ALU is used to perform accumulation on floating-point results. The accumulator sub-unit supports rounding and 64-bit saturation. 80-bit accumulator values can be moved to and from registers in 32-bit blocks.

The ADSP-2106x allows parallel operation of the multiplier and ALU for a subset of the ALU operations. In those cases, both operations must be fixed-point or floating-point. When performing floating-point multiply-accumulate operations, the multiplier unit performs multiplication and the ALU performs accumulation. In this case, the contents of two registers are multiplied and the product is written back to the register file. In the same instruction cycle, the ALU accumulates the product from the previous multiply (which is held in the register file) with the value in another register and writes the accumulated result back into the register file.

When both the ALU and multiplier are used, only registers from groups F0-F3 and F4-F7 can be used for the multiplier inputs, and only registers from groups F8-F11 and F12-F15 can be used for the ALU inputs.

The register file and the accumulator in the multiplier unit are shadowed by a secondary set of registers. The processor can switch between the primary and secondary set of registers under program control in one instruction cycle. In addition, the shadow accumulator can be accessed directly by fixed-point arithmetic operations without switching the entire register context.

The ADSP-2106x's shadowing of its arithmetic registers (and its address registers, discussed below) is unusual and provides good support for interrupt servicing and multitasking environments.

All ADSP-2106x data path operations execute in a single instruction cycle.

Memory System

The ADSP-2106x memory system consists of either 512 Kbytes, 256 Kbytes, 128 Kbytes, or 68 Kbytes (depending on the family member) of on-chip memory, up to 2.5 Mbytes of off-chip multiprocessor memory (memory that physically resides in other ADSP-2106x processors), and up to 24 Gbytes of general-purpose off-chip memory. The ADSP-21065L is limited to 68 Kbytes of off-chip multiprocessor memory (since it can connect to only one other ADSP-21065L in a multiprocessor system) and 256 Mbytes of external memory. Memory is arranged in a unified, word-addressable address space that contains both instructions and data. Additionally, the ADSP-2106x has a 32-word on-chip instruction cache.

On-chip memory is divided into two independent blocks. On the ADSP-21060, ADSP-21061, and ADSP-21062, each memory block is 256-, 64-, or 128-Kbytes in size, respectively. On the ADSP-21065L, there is one 32-Kbyte block and one 36-Kbyte block. On-chip memory as a whole can be configured into a 48-bit-wide area and 32-bit-wide area. 48-bit instruction words and 40-bit data words can be stored in the 48-bit-wide area, and 32-bit and 16-bit data words can be stored in the 32-bit-wide area. This feature allows on-chip memory to be used more efficiently than if all on-chip memory was configured as full-width, 48-bit-wide memory. To simplify addressing and memory implementation, the 48-bit-wide area always resides at lower addresses than the 32-bit-wide area.

The ADSP-21060 and ADSP-21062 contain significantly more on-chip memory than most commercial DSP processors. The ADSP-2106x's ability to arrange on-chip memory into different word widths allows the processor to use on-chip memory efficiently despite the processor's disparate instruction and data word widths.

Separate address generators, address buses, and data buses allow both memory blocks to be accessed by the core processor in a single instruction cycle. Each set of buses—called the "PM" and "DM" buses—can access either block of on-chip memory once per instruction cycle. However, both buses cannot access the same memory block in a single cycle. Unlike the ADSP-21020, the PM and DM address and data buses in the core of the ADSP-2106x are not restricted to accessing separate memory blocks. That is, the PM and DM buses can access the same physical memory locations. However, memory is accessed at the highest rate when the PM and DM buses address different blocks of on-chip memory, or when one accesses on-chip memory and the other accesses off-chip memory.

Each block of the ADSP-2106x on-chip memory is dual-ported. The PM and DM buses are both connected to the first port of each memory block. The second port of each block is connected to a single I/O bus. This means that a total of three memory accesses are possible per instruction cycle between the two blocks: one PM access, one DM access, and one I/O access. As a result, the core processor can perform computations on data in the on-chip memory while the I/O Processor transfers data into or out of one block without contention.

The PM bus set is also used by the program sequencer to fetch instructions. Therefore, the PM bus is nominally occupied with an instruction fetch in each instruction cycle and is unavailable for accessing data. However, if the instruction to be fetched is contained in the instruction cache, then the PM bus is free to perform a data fetch, and the processor can complete two data accesses in a single instruction cycle.

The ADSP-2106x instruction cache operates differently than a traditional cache in which an instruction is loaded whenever a cache miss occurs. The ADSP-2106x cache only caches instructions that conflict with data memory accesses using the PM buses; all other instructions are accessed directly from on-chip or off-chip memory. The cache contains two sets of 16 words each. Each instruction address can be mapped into two locations, one in each of the two sets. The least-significant four bits of the instruction address determine which word within each set can be used for a particular instruction. When an instruction is loaded into the cache, it is loaded into the word that was least recently accessed. The cache can be disabled or locked (in which case its contents are frozen) under software control.

> *The ADSP-2106x caching scheme makes good use of the small cache by only storing instructions where necessary to allow two data accesses in one instruction cycle.*

The I/O Processor on the ADSP-2106x provides an on-chip DMA controller that allows transfer between internal memory and any one of external memory, multiprocessor memory, a host processor, a serial port, or a link port. The DMA controller also allows transfers between external memory and external peripherals. More information on the DMA controller is in the *Peripherals* section below.

The ADSP-21065L at 66 MIPS achieves a peak and maximum sustainable data bandwidth of 132 million 32-bit words/second to or from on-chip memory when executing instructions from the instruction cache. When executing instructions from on-chip memory, the maximum data bandwidth is 66 million 32-bit words/second. The on-chip DMA controller allows one additional access to on-chip memory per cycle, corresponding to an additional 66 million 32-bit words/second on a 66 MIPS ADSP-21065L.

An ADSP-2106x can access the on-chip memories of other ADSP-2106x processors on a shared bus via its external memory interface. Each processor includes arbitration logic to coordinate external accesses. Up to six ADSP-2106x processors (or two if the ADSP-21065L is used), a host processor, and external memory can be connected using the shared bus scheme. A processor can access the memory of any other single processor or can perform a broadcast write to all other processors simultaneously. Such multiprocessor accesses use the I/O Processor of the accessed ADSP-2106x and require it to release its I/O bus and external memory interface. Thus, multiprocessor accesses compete for access to the I/O bus with I/O operations on the accessed processor. However, ADSP-2106x arbitration logic allows the accessed ADSP-2106x to reclaim control of I/O resources if needed. Also, multiprocessor accesses do not disrupt operations that use only the core processor and on-chip memory. On the ADSP-21060, ADSP-21061, and ADSP-21062, any memory location or I/O Processor register can be directly accessed through the external memory interface. On the ADSP-21065L, only I/O Processor registers can be accessed directly without interrupting the processor. Any memory location can still be indirectly accessed on the ADSP-21065L by programming the on-chip DMA controller through the I/O Processor

External Memory Interface

External memory consists of off-chip RAM, ROM, or multiprocessor memory (memory that physically resides in other ADSP-2106x processors) that is accessed via the ADSP-2106x external memory interface. The ADSP-21065L has special support for SDRAM, including refresh generation. The on-chip PM, DM, and I/O buses share access to the external memory interface. On the ADSP-21060, ADSP-21061, and ADSP-21062, the external address bus is 32 bits wide, which allows the processor to access 4 Gwords of external memory. The external data bus is 48 bits wide to accommodate the 48-bit instruction width. On the ADSP-21065L, the external address bus is 26 bits wide with the highest two bits decoded and provided only as four bank select lines, and the external data bus is 32 bits wide. The ADSP-21065L can access 40- and 48-bit data and instructions stored in external memory over the 32-bit data bus, but requires two memory accesses to do so.

The external memory interface on the ADSP-21060, ADSP-21061, and ADSP-21062 can perform accesses at a maximum rate of one per instruction cycle. Thus, the peak and maximum sustainable off-chip memory bandwidth is 50 Mwords/second of either 48-bit instructions or 40- or 32-bit data on a 50 MIPS ADSP-21061. On the ADSP-21065L, the external memory interface is only capable of one access every two instruction cycles. Thus, the maximum sustainable off-chip memory bandwidth is 33 million 32-bit words/second on a 66 MIPS ADSP-21065L.

> *Unlike the other members in the ADSP-2106x family, the external data bus of the ADSP-21065L is 32 bits wide. Since the external memory bandwidth is effectively halved when 40- or 48-bit data or instructions are accessed, the ADSP-21065L must execute instructions from on-chip memory for good performance.*

Unlike the ADSP-21020, the ADSP-2106x stores instructions and data together in off-chip memory. In a multiprocessor system, external memory can be shared by up to six ADSP-2106x processors and a host processor utilizing the arbitration scheme mentioned previously for multiprocessor memory.

External memory is divided into five sections: four "banked" memories of equal size and one "non-banked" memory. On the ADSP-21060, ADSP-21061, and ADSP-21062, the size of the banked memories is set by a parameter in a control register and can range in size from 8 Kwords to 256 Mwords in powers of two. Non-banked memory consists of any remaining memory in the memory space. The processor has four memory select pins and asserts one of the pins during banked memory accesses to indicate which bank is being accessed. These pins can be used to drive the chip select inputs of external RAM devices, eliminating the need for external address decoding circuitry in many applications. These memory select pins are unused during non-banked memory access. On the ADSP-21065L, external memory is divided into banks, but the size of the banked memories is fixed at 16 Mwords and there can be no non-banked memory.

Wait state configuration is independently specified for each bank of memory. The processor supports both externally requested wait states and zero to seven programmed wait states. If externally requested and programmed wait states are both used, the processor can be configured to wait until *both* wait state sources (internal count or external pin) expire before completing the memory access, or to wait only until *one* of the wait state sources expires.

In addition, the ADSP-21060, ADSP-21061, and ADSP-21062 can be configured to use page-mode DRAM with an external DRAM controller in memory bank zero. When a memory access in bank zero crosses a page boundary, an output pin is asserted and an additional wait state can be automatically inserted. Page lengths can be independently specified for program and data memory and can be any power of two from 256 to 32,768 words, inclusive. The ADSP-21065L provides an internal SDRAM controller that can be enabled for memory accesses in bank zero. This internal controller allows up to 16 32-bit Mwords of external SDRAM memory to be used with no external logic.

The ADSP-2106x provides unusually flexible support for external memory.

An arbitration mechanism is provided that allows a host processor or another ADSP-2106x in a multiprocessor system to gain control of the external memory bus. Multiple ADSP-2106x processors can share an external bus without additional arbitration circuitry. Either fixed- or rotating-priority arbitration can be selected for the ADSP-2106x processors. Under the fixed-priority scheme, the programmer assigns each ADSP-2106x a different priority. Under the rotating-priority scheme, the priorities are automatically re-assigned after every transfer of bus mastership so that all ADSP-2106x processors in turn have different priorities assigned. When a processor requests control of the bus from another processor that has a lower priority, the lower-priority processor completes the current instruction, places its external memory interface in a high-impedance state, and asserts its bus grant output pin. The higher-priority processor then uses the external bus. When that processor finishes, a lower-priority processor can gain control of the bus. The ADSP-2106x has conditional instructions that depend upon whether an ADSP-2106x is the current bus master in a multiprocessor system. These instructions can be used to perform an atomic read-modify-write operation.

Address Generation Units

The ADSP-2106x supports register-direct, memory-direct, and register-indirect addressing modes. Most arithmetic operations use register-direct addressing exclusively. Immediate operands and memory-direct addressing can be used for loading values into memory or registers. Register-indirect addressing is used for moving data between registers and memory, including operand-unrelated parallel moves.

Like the ADSP-21xx, the ADSP-2106x has two data address generators, one (DAG1) associated with the DM address bus and the other (DAG2) associated with the PM address bus. The PM bus can only access internal memory and the lowest 12 Mwords of external memory in the ADSP-2106x memory space.

Each data address generator contains eight address registers and eight modifier registers. Values stored in the modifier registers are used to pre- or post-increment address register contents. Within each address generator, any modifier register can be used to modify the address held in any address register. Post-increment values of -1, 0, and +1 are not built into hardware; if these values are needed, they must be loaded into modifier registers or specified as immediate values. Immediate pre- and post-increment values can be specified as part of the instruction word, but when this feature is used only one data access is allowed in the instruction. This feature is useful in C compilers for stack maintenance.

> *The fact that any of eight modifier registers can be used to post-increment any of the eight address registers in an address generator provides flexibility for the programmer. However, the fact that the address generators do not provide built-in post-increment by -1, 0, and +1 means that these post-increment values must be loaded into modifier registers if they are needed in an instruction that does not allow immediate increment values. Most other DSP processors provide these post-increment values in hardware.*

Indexed addressing is also supported (confusingly, Analog Devices calls it "pre-modification"). With indexed addressing, the effective address for the access is formed by adding the value in a modifier register to the value in an address register; the value in the address register is not changed.

The ADSP-2106x supports circular addressing with no alignment constraints by giving each address register a companion base address register and buffer length register. One pair of base and length registers in each of the program and data memory address generators can be configured to generate an interrupt when the address reaches the end of a programmer-defined circular buffer.

Bit-reversed addressing is supported by one address register in each of the data address generators. (In addition, the ADSP-2106x offers a bit-reversal instruction which can be used to modify and then bit-reverse the contents of any address register. Note that this bit-reversal instruction does not access the memory referenced by the bit-reversed address.)

The data address generator used for PM data access can also be used for indirect jumps and subroutine calls; this is useful for reconfigurable interrupt service routines and jump tables.

All address, base address, buffer length, and modifier registers are shadowed by a secondary set of registers. The processor can switch between the primary and secondary set of registers under program control in one instruction cycle.

Pipeline

The ADSP-2106x uses a three-stage pipeline comprised of fetch, decode, and execute stages. The pipeline is partially interlocked and invisible to the user except for a few cases, the most important of which are:

- When an instruction that loads an address generator register is immediately followed by an instruction that uses the same address generator for addressing, the processor inserts a one-instruction-cycle delay between the two instructions.
- An instruction that writes to a modulo addressing buffer length register or base address register must not be immediately followed by an instruction that reads directly from the corresponding address register, or incorrect results will be produced.
- Non-delayed branch, call, and return instructions take three instruction cycles to execute. Delayed branches are available, however, so that the programmer has the option of executing two useful instructions during the two-instruction-cycle delay slot.

Although programmers must be aware of the ADSP-2106x pipeline to create optimal code, the impact of the pipeline on the programmer is minor compared to other floating-point DSP processors.

Instruction Set

The instruction set and registers of the ADSP-2106x are summarized in Tables 7.3-2 and 7.3-3. The ADSP-2106x uses a 48-bit instruction word, and its instruction set is a superset of the ADSP-21020 instruction set; the ADSP-2106x is object-code compatible with the ADSP-21020 (although the object code must be re-linked because the ADSP-21020 and ADSP-2106x use different memory maps). The assembly language of the ADSP-2106x is very similar to that of the ADSP-21xx. Also, ADSP-2106x assembly-language code can be re-assembled for the ADSP-2116x without modification.

With its unusually large 48-bit instruction word, the ADSP-2106x provides an exceptionally rich and regular instruction set.

Class	Instructions
Floating-Point Arithmetic	Absolute value, add, add with carry, clear, increment, decrement, round, subtract, subtract with borrow, negate, set status, simultaneous add and subtract, average two values, minimum, maximum, clip, absolute value of sum, absolute value of difference, round, truncate, scale, extract mantissa, extract exponent, copy sign
Fixed-Point Arithmetic	Absolute value, add, add with carry, clear, increment, decrement, subtract, subtract with borrow, negate, pass value, simultaneous add and subtract, average two values, minimum, maximum, clip
Floating-Point Multiply	Multiply; can be executed in parallel with floating-point arithmetic instructions including add, subtract, absolute value
Fixed-Point Multiply	Multiply, multiply-accumulate, multiply-negate-and-accumulate, multiply-add, multiply-subtract, clear accumulator, saturate, round. All multiply operations can operate on fractional or integer, signed or unsigned values, and can optionally round their result.
Logic	*And*, *or*, *exclusive-or*, *not*
Shifting	Arithmetic/logical shift left/right 0-32 bits with optional logical *or* operation
Rotation	Rotate left/right 0-32 bits
Conditional Execution	Most instructions can be conditionally executed
Comparison	Floating-point and fixed-point compare
Looping	Single- or multiple-instruction hardware loop
Branching	Conditional and unconditional relative or absolute branch, delayed or non-delayed, with optional arithmetic operation
Subroutine Call	Conditional and unconditional call, delayed or non-delayed, with optional arithmetic operation
Bit Manipulation	Bit set, clear, test, toggle; bit field deposit, bit field extract
Special Function	Division iteration, reciprocal of square root seed, fixed-point to floating-point and floating-point to fixed-point conversion with optional scaling, conversion between "short float" and 32-or 40-bit floating-point, compiler stack support, bit-reversal of address in a register

TABLE 7.3-2. ADSP-2106x instruction set summary. Underlined instructions are additions to the ADSP-21020 core instruction set.

Assembly Language Format

The ADSP-2106x uses an algebraic assembly language format, where instructions are typically divided into three parts: a computation field and one or two data move fields. The computation field specifies the operation(s) to be carried out by the ALU, multiplier-accumulator, or shifter, and the data move fields specify a register-register data move or up to two register-memory data moves. In the case of floating-point multiply-accumulate instructions, there are two computation fields: one for the multiplication and one for the accumulation. For example, consider the instruction:

```
F12=F8+F12, F8=F0*F4, F0=DM(I0,M1), F4=PM(I8,M9);
```

The previous value in register F8 (possibly the result of the previous multiply instruction) is summed with the value in register F12, and the result written to register F12. The previous contents of registers F0 and F4 are multiplied, and the product is written to register F8. Registers F0 and F4 are loaded with new input data from data memory and program memory, respectively. Finally, the address registers I0 and I8 are post-incremented by adding to them the values in the modifier registers M1 and M9, respectively.

To distinguish between fixed-point and floating-point operations, the arithmetic registers are referred to as R0-R15 for fixed-point and F0-F15 for floating-point operations.

Registers	Width	Purpose
R0-R15	40 bits	Arithmetic registers; referred to as F0-F15 for floating-point operations
MR2-MR0	80 bits	Fixed-point accumulator; can be treated as two 32-bit registers (MR1, MR0) and one 16-bit register (MR2)
I0-I7	32 bits	Address generator 1 address registers
B0-B7	32 bits	Address generator 1 modulo base registers
L0-L7	32 bits	Address generator 1 modulo length registers
M0-M7	32 bits	Address generator 1 modifier registers
I8-I15	24 bits	Address generator 2 address registers
B8-B15	24 bits	Address generator 2 modulo base registers
L8-L15	24 bits	Address generator 2 modulo length registers
M8-M15	24 bits	Address generator 2 modifier registers

TABLE 7.3-3. ADSP-2106x register summary. The processor can switch between these primary registers and a secondary set of shadow registers (not listed here) under program control.

Parallel Move Support

The ADSP-2106x supports operand-unrelated parallel moves with all arithmetic operations. For example, multiply-accumulate instructions support one data access (read or write) via the DM bus and one data access (read or write) via the PM bus.

If two parallel moves are used, conditional instruction execution is not allowed. However, with one parallel move, conditional execution is available. In addition, the ADSP-2106x allows a single register-to-register move in parallel with most computations.

The ADSP-2106x can perform up to two reads, two writes, or one read and/or write in parallel with an arithmetic operation.

> *The ADSP-2106x provides good flexibility in the use of parallel data moves.*

Orthogonality

The ADSP-2106x instruction set is very orthogonal by DSP processor standards. This is made possible by its 48-bit instruction word size. For example, nearly all instructions can be conditionally executed, and most arithmetic instructions operate on the same set of registers. Also, most arithmetic instructions support both fixed-point and floating-point formats.

Execution Times

Nearly all ADSP-2106x instructions execute in one instruction cycle in the absence of wait states or external memory access conflicts (which may occur, for example, when both instructions and data are located off-chip). Instructions with PM bus data accesses execute in two instruction cycles unless the instruction fetched (not the instruction being executed) in parallel with the program memory data access is held in the cache, in which case they execute in one instruction cycle.

Non-delayed branches, calls, and returns execute in three cycles. Delayed branches have a latency of three cycles; when a delayed branch, call, or return is executed, the two instructions following the branch instruction are also executed before the branch takes effect.

Instruction Set Highlights

Noteworthy features of the ADSP-2106x instruction set include:

- Multiply with parallel dual add/subtract, useful for computing FFT and DCT kernels
- Bit-field insertion and extraction
- Any combination of two parallel register-memory moves (two loads, one load and one store, or two stores) or one register-register move with a computation
- Conditional execution of nearly all instructions

- Hardware looping with a wide range of loop termination conditions (discussed below)
- Branch with loop abort option (discussed below)
- Delayed branch, call, and return

Execution Control

Clocking

The ADSP-21060, ADSP-21061, and ADSP-21062 processors operate from an externally generated clock at the instruction execution rate; that is, a 40 MIPS ADSP-21060 operates from a 40 MHz master clock. No on-chip oscillator is provided for clock signal generation. The internally buffered instruction clock is made available on an output pin.

The ADSP-21065L uses an input clock with a frequency equal to half the instruction rate; that is, a 66 MIPS ADSP-21065L operates from a 33 MHz master clock. The ADSP-21065L also provides internal clock generation oscillator if an external clock is not available. The internally buffered instruction clock is available through the SDRAM interface.

Hardware Looping

The ADSP-2106x provides zero-overhead hardware looping through its DO instruction. A sequence of instructions of any length can be contained in a hardware loop. The loop exit condition can be a zero value in the special-purpose loop counter, an arithmetic condition generated by the ALU or multiplier-accumulator, the value of a bit I/O pin, or loss of mastership of the external memory interface. Hardware loops can be nested up to six levels and are interruptible. The maximum number of repetitions is 2^{32} when using the special-purpose loop counter.

In the common case where the loop termination condition is based on a counter that is automatically decremented each time through the loop, the initial counter value is specified in the DO instruction, so that initializing the loop requires a total of one instruction cycle. However, loops containing fewer than three instructions incur extra setup overhead.

In addition, the ADSP-2106x provides a branch instruction with optional "loop abort." This instruction is used to exit a hardware loop before the loop termination condition is reached. The instruction adjusts the loop stack so that the loop is cleanly terminated, even though the termination condition was not reached.

The ADSP-2106x provides unusually powerful hardware looping features.

Interrupts

The ADSP-2106x has 29 interrupt sources. Interrupts are individually prioritized (each has a separate but fixed priority) and optionally nestable. External interrupts include the reset pin and three interrupt pins. Internal hardware interrupt sources include stack overflow, timer expiration, circular buffer overflow (two buffers with separate interrupts), arithmetic exceptions (fixed-point overflow; floating-point overflow, underflow, and invalid operand), link port service request, and DMA activity. In a multiprocessor configuration, one ADSP-2106x processor can force another ADSP-2106x processor to execute a "multiprocessor vector interrupt." In addition, four user software interrupts are provided. All interrupts except the reset pin are individually maskable. The timer has two interrupt locations, both corresponding to timer expiration, but having different priorities. By masking the high-priority timer interrupt, the programmer can lower the priority of the timer interrupt.

The ADSP-2106x responds to an interrupt by pushing the program counter onto the PC stack. If the interrupt is an external interrupt or a timer interrupt, the processor status is pushed onto the status stack. Next, the processor branches to the first instruction at the corresponding interrupt vector location. Each interrupt vector location contains four words of program memory, so that short interrupt service routines can reside entirely within the interrupt vector table, with no need for a branch to reach the interrupt service routine.

Latency from the time when an interrupt is asserted as pending to completion of execution of the first interrupt vector instruction is five instruction cycles, assuming the processor is in an interruptible state.

The ADSP-2106x has the ability to clear, force, or test for the presence of interrupts. In addition, as mentioned above, the ADSP-2106x provides shadow registers for arithmetic and address registers. Under program control, the processor can be switched between the primary and shadow registers in a single instruction cycle.

> *The ADSP-2106x has flexible and powerful interrupt handling mechanisms. In particular, the availability of shadow registers and ability to execute four-instruction interrupt service routines without the need for a branch instruction are noteworthy. The multiprocessor vector interrupt is useful for interprocessor commands in a system with multiple ADSP-2106x processors.*

Stack

The ADSP-2106x provides a 30-word hardware stack for the program counter, used for subroutine calls and hardware loops. In addition, a separate six-word hardware stack is provided for the loop counter, end-of-loop address, and loop termination condition, to support six levels of nestable hardware loops. The ADSP-2106x generates a stack-overflow exception if any of the stacks are exhausted.

The ADSP-2106x provides specialized stack frame instructions to implement a software stack for programs written in C.

> *The use of a small hardware stack for storing the PC in a processor with a large address space is surprising. However, since large programs are likely to be written in C, the specialized stack frame instructions mitigate this problem.*

The ADSP-2106x also provides a five-word hardware status stack where the processor status is stored during external interrupts or timer interrupts.

Bootstrap Loading

On reset, the ADSP-2106x can begin execution at a fixed location in memory, or it can bootstrap load over the host port, the link ports, or a byte-wide ROM attached to the external memory interface.

Peripherals

The ADSP-2106x on-chip peripherals include a timer (two timers on the ADSP-21065L), two synchronous serial ports, six "link ports" (not available on the ADSP-21061 or the ADSP-21065L), a ten-channel DMA controller (six-channel on the ADSP-21061), four bit-I/O pins (12 pins on the ADSP-21065L), and a host port. The serial ports, link ports, and DMA controller are grouped together and designated the I/O Processor. Peripherals on the ADSP-2106x are generally independent and do not share processor pins. However, on some family members peripherals share DMA channels so that not all peripherals may be configured for DMA at the same time.

- **Serial Ports**

  The serial I/O ports of the ADSP-21060, ADSP-21061, and ADSP-21062 support bit rates equal to the clock speed of the processor. That is, each serial port on a 40 MHz ADSP-21060 can transfer data at a maximum rate of 40 Mbits/second. Each serial port supports both transmit and receive channels with independent serial clock and framing signals. Clock and framing signals can be generated on-chip or can come from an off-chip source. The serial ports can transfer words from 3 to 32 bits long; words can be converted to serial data (or vice versa) using either most-significant-bit-first or least-significant-bit-first serial bit order. A multi-channel mode allows each serial port to interpret the serial data stream as time-division-multiplexed (TDM) words using up to 32 separate channels. Each serial port can select or ignore any of these TDM channels. A-law and μ-law companding are available. Single-word transfers can be interrupt-serviced by the processor core, and multiword block transfers can be serviced by the DMA controller without intervention by the processor core.

  The ADSP-21065L serial ports are generally similar to the other family members, although there are some differences. The 21065L serial ports only support a maximum bit rate equal to half the instruction clock speed of the processor. However, each serial port also has two transmit and two receive channels. The two transmit channels share a transmit clock and framing signals and the two receive channels share similar receive clock and framing signals. The four channels of each serial port are otherwise independently enabled and configured. Each serial port on a 66 MHz ADSP-21065L can transfer data at a maximum rate of 66 Mbits/second, 33 Mbits/second through each channel. The ADSP-21065L serial ports also support the I^{2}S protocol.

The serial ports on the ADSP-2106x have many useful features.

- **Timer**

  The ADSP-21060, ADSP-21061, and ADSP-21062 devices feature a 32-bit interval timer. The timer generates an interrupt and pulses the TIMEXP output pin when its counter reaches zero. The timer is clocked with the instruction clock with no prescaler. The timer interrupt can be assigned as a low- or high-priority interrupt.

  The ADSP-21065L has two 32-bit timers with the same features as the timer found on the other ADSP-2106x devices; however, the ADSP-21065L timers use two sets of timer registers and have two associated output pins. Also, the ADSP-21065L timers can use an externally-supplied clock.

  Because the ADSP-2106x timers use 32-bit counters, the lack of a prescaler is not a drawback.

- **Link Ports**

  The link ports are 4-bit bidirectional parallel communications ports that can be connected to peripheral devices or to the link ports on other ADSP-2106x processors. Each link port consists of four bidirectional data lines, a bidirectional clock line, and a bidirectional acknowledge line. The ports can be clocked twice per processor instruction cycle, allowing each port to transfer up to 8 bits of data per cycle. The direction of data transfer for each link port is specified by the link port control register in the I/O Processor. Data is automatically packed or unpacked into 32-bit or 48-bit words. Single-word transfers can be interrupt-serviced by the processor core, and multiword block transfers can be serviced by the DMA controller without intervention by the processor core. There are six independent buffer registers called "link buffers" that are shared between the six link ports. Each link buffer can be assigned to any single link port and is used to double-buffer that link port.

- **DMA**

  The DMA controller on the ADSP-2106x supports ten channels of DMA (six on the ADSP-21061). These channels can be used to perform block transfers of data between on-chip memory and the link ports, serial ports, or external memory interface. Additionally, these channels can be used to perform block transfers between external memory and external peripherals. The DMA controller allows the core processor or external peripherals to specify block-data transfer operations and then return to normal operation while the DMA controller carries out those transfers independently. All DMA transfers are executed using the internal I/O bus of the ADSP-2106x. DMA transfers never conflict with the processor core for on-chip RAM access.

Each ADSP-2106x DMA channel utilizes a dedicated on-chip data buffer for data transfers. Each data buffer is allocated to one or more physical I/O resources, such as serial ports, as listed in Table 7.3-4. Most buffers are allocated to only one I/O resource. However, on the ADSP-21060, ADSP-21061, and ADSP-21062, buffers for channels 1, 3, 6, and 7 can be allocated to a fixed resource or one of the link buffers (not link *ports*). For example, the data buffer for DMA channel 1 can be allocated to "serial port 1 receive" or "link buffer 0." DMA on these four shared channels is only enabled for the allocated resource, and the unallocated resource cannot use DMA transfers. For example, if DMA channel 1 were allocated to "serial port 1 receive," then link buffer 0 would be unavailable. However, each link buffer can be assigned to any link port, so five other link buffers would be available to assign to the six link ports. Conversely, if DMA channel one were allocated to "link buffer 0," then serial port 1 receive would be unavailable. Unfortunately, the serial ports have no alternate buffers.

> *The ADSP-2106x has an extremely flexible and elaborate DMA controller. However, the relationship between DMA channels, data buffers, and link ports is complicated. Conflicts between I/O resources may not be obvious to the programmer.*

The DMA channels have a priority schedule which the DMA controller uses to determine which channel can drive the I/O bus on each cycle. The priority schedule, shown in Table 7.3-4, is fixed except for the four external memory interface channels. Those four channels can be enabled with a rotating priority schedule. There is no throughput loss incurred when the controller switches channels. For example, reading four words into memory from link buffer 0 and then reading four words into memory from link buffer 1 can execute as quickly as reading eight words into memory from link buffer 0 alone.

The ADSP-2106x DMA controller allows DMA chaining. In this scenario, the ADSP-2106x automatically configures a new DMA transfer when the entire contents of the current buffer have been transmitted or received. The parameters for successive DMA transfers are called a "transfer control block" (TCB) and are read from internal memory.

- **Bit I/O**

  The ADSP-2106x has 4 bit-I/O pins (except the ADSP-21065L, which has 12 pins), which can be individually configured as inputs or outputs.

- **Host Port**

  The ADSP-2106x external memory interface doubles as a host port. The host port can be configured for 16- or 32-bit synchronous or asynchronous transfers. Via the host port, a host processor can directly access the on-chip memory of the ADSP-2106x. The host processor can also initiate DMA transfers and issue commands to the ADSP-2106x by triggering interrupts via the host port.

On-Chip Debugging Support

ADSP-2106x DSPs have a JTAG serial debugging interface with boundary scan and on-chip debugging support. Analog Devices offers "EZ-ICE" in-circuit emulators for the ADSP-2106x family that make use of this serial interface. The EZ-ICE emulator is an IBM PC plug-in card with a cable that connects to the DSP's JTAG port through a connector in the target system. Through the JTAG interface, memory and registers can be inspected, and breakpoints can be set based on individual addresses or address ranges in program or data memory. Instruction and bus-tracing capabilities are not provided.

DMA Channel Number and Priority (0 = Highest)	Associated Resources	
	ADSP-21060, ADSP-21061, and ADSP-21062	ADSP-21065L
0	Serial port 0 receive	Serial port 0 receive, A data
1	Serial port 1 receive or link buffer 0	Serial port 1 receive, A data
2	Serial port 0 transmit	Serial port 0 receive, B data
3	Serial port 1 transmit or link buffer 1	Serial port 1 receive, B data
4	Link buffer 2	Serial port 0 transmit, A data
5	Link buffer 3	Serial port 1 transmit, A data
6	External memory interface buffer 0 or link buffer 4	Serial port 0 transmit, B data
7	External memory interface buffer 1 or link buffer 5	Serial port 1 transmit, B data
8	External memory interface buffer 2	Host port buffer 0
9	External memory interface buffer 3	Host port buffer 1

TABLE 7.3-4. The ten DMA channels on the ADSP-21060, ADSP-21062, and ADSP-21065L. The ADSP-21061 eliminates channels 4, 5, 8, and 9. "A" and "B" data refer to the two channels supported by each serial port on the ADSP-21065L.

JTAG-based in-circuit emulators that provide similar functionality to the EZ-ICE and connect to the host debugger through either USB, ethernet, or PCI-card interfaces are available from third-party vendors.

Power Consumption and Management

According to Analog Devices, the ADSP-2106x devices (excluding the ADSP-21065L) typically consume 3.4 watts at 5.25 volts when executing at 40 MIPS (this corresponds to a power consumption of 3.1 watts at 5.0 volts at the same speed). For the low-voltage variants the power consumption is 1.5 watts at 3.3 volts and 40 MIPS. The typical power consumption was measured for an FFT butterfly with two memory accesses and an instruction fetch from the cache. Analog Devices quotes the ADSP-21065L power consumption as 0.87 watts at 3.3 volts and 60 MIPS.

The ADSP-21065L power consumption of 0.87 watts at 3.3 volts and 60 MIPS is relatively low for a floating-point DSP processor.

The ADSP-2106x provides an IDLE instruction for power management. Execution of the IDLE instruction places the processor in a low-power mode. An external, timer, or DMA interrupt wakes the processor and causes it to execute the appropriate interrupt service routine. The wake-up latency from low-power mode is two instruction cycles used to fetch and decode the instructions in the interrupt service routine. The ADSP-21061 and 21065L provide an additional IDLE16 instruction that reduces the master clock to 1/16 of the input clock rate. To resume full-speed operation, an external interrupt must occur. Full-speed operation is resumed two instruction cycles after the interrupt has been recognized. In the IDLE16 low-power mode, the DMA, link ports, serial ports, and external port cannot be used.

According to Analog Devices, ADSP-2106x IDLE-mode power consumption is 1.1 watts typically at 5.25 volts and 40 MIPS, corresponding to a power consumption of 1.0 watts at 5.0 volts at the same speed. IDLE-mode power consumption at 3.3 volts and 40 MIPS is 0.39 W.

Benchmark Performance

The ADSP-2106x has been benchmarked with the BDTI Benchmarks™. Overall benchmark results for all benchmarked processors are presented in Chapter 8, *BDTI Benchmark™ Results*. We summarize and analyze ADSP-2106x benchmark performance in the paragraphs below. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not reflect the processor's instruction cycle rate; therefore, lower instruction cycle counts imply a more powerful architecture, but do not imply faster speed. Next we discuss benchmark execution times and cost-execution time products, indicating processor speed and cost-performance, respectively. We then discuss the processor's energy consumption, which reflects the energy consumed by the processor in order to perform a task. Finally, we discuss the processor's memory usage. We divide the memory usage discussion into three parts: Control benchmark memory usage, overall benchmark program memory usage, and benchmark data memory usage.

Because the ADSP-2106x uses an instruction cache, the benchmark analysis presented in Chapter 8, *BDTI Benchmark™ Results*, treats the ADSP-2106x as two distinct processor families. One family (referred to as the ADSP-2106x) assumes that the cache is empty before each benchmark is executed. The other family (referred to as the ADSP-2106x-C) assumes that the processor's instruction cache has been preloaded by executing the benchmark code once prior to measuring the instruction cycle counts for that benchmark. The total normalized cycle count is only about 5% lower for the ADSP-2106x-C than for the ADSP-2106x. For this reason, we have focused on the ADSP-2106x-C in the analysis below. However, the effect of cache preloading is more significant on single-sample benchmarks, where the cycle counts are about 20% lower for the ADSP-2106x-C than for the ADSP-2106x. When using these benchmark results to gauge the performance of the ADSP-2106x family in a potential application, we urge readers to carefully consider how the processor's cache will perform in that application.

Execution Performance

- **Instruction cycle counts**: Total normalized instruction cycle counts are found in Figure 8.1-13. The ADSP-2106x-C's total normalized instruction cycle count is roughly average for all benchmarked processors but about 15% above average for floating-point DSPs.

  The ADSP-2106x-C tends to have lower cycle counts than other conventional DSPs due to the processor's quite powerful instruction set that allows most instructions to be conditionally executed and provides flexibility for specifying multiple unrelated operations to take place within a single instruction. These features are made possible in part by the processor's unusually wide 48-bit instruction word. However, ADSP-2106x-C cycle counts tend to be higher than those of processors with modern VLIW or SIMD architectures, such as the TMS320C67xx and ADSP-2116x.

  For most block-oriented benchmarks, the wide instruction word of the ADSP-2106x does not reduce the number of instructions in the inner loop (as compared to other benchmarked processors), but does allow for slightly more efficient housekeeping outside the inner loop. On many of the benchmarks, this reduces the ADSP-2106x-C cycle counts slightly relative to other conventional DSPs. This is visible in particular on the **Single-Sample FIR**, where code outside the inner loop accounts for a significant portion of the total cycle count. Since the ADSP-2106x-C incurs little overhead on initialization code, it has a low cycle count for a conventional DSP on the Single-Sample FIR filter benchmark. A similar tendency can be seen on the **LMS Adaptive FIR**. However, the main reason that the ADSP-2106x-C has a relatively low cycle count (for a conventional DSP) on the LMS benchmark is that the coefficient update and convolution can be combined efficiently in a single loop.

On the **Two-Biquad IIR** filter benchmark, the ADSP-2106x-C has a lower cycle count than most conventional DSPs. The processor takes advantage of its flexible parallel move support and its ability to perform an arithmetic operation in parallel with a multiplication. This enables the processor to perform an IIR filter biquad in four instruction cycles in the inner loop. The same features help the ADSP-2106x-C on the **FFT** benchmark, where it achieves very low cycle counts for a conventional DSP. A specialized instruction that allows simultaneous multiply and dual add/subtract operations (taking somewhat restricted operands) with up to two operand-unrelated data moves, together with a large register set, enable the processor to implement the FFT using a compact radix-4 butterfly loop.

The **Control** benchmark does not provide much opportunity for executing several operations in parallel; instead, the ADSP-2106x-C takes advantage of conditional instruction execution and optionally delayed branches to keep cycle counts down.

The ADSP-2106x-C's cycle count on the **Viterbi** benchmark is roughly average, although it is well below average for a conventional DSP. The processor's ability to perform the add-compare-select phase of Viterbi decoding efficiently is aided by its flexible parallel move support and its dual add/subtract operation, which is useful for calculating path metrics. However, the ADSP-2106x-C has a higher cycle count than the benchmarked DSPs with SIMD or VLIW architectures, and than the TMS320C54xx, which has specialized Viterbi decoding support.

On the **Bit Unpack** benchmark, the ADSP-2106x-C has lower cycle counts than most benchmarked DSPs, due in particular to its powerful bit-field manipulation capabilities and support for delayed branches. However, the benchmarked DSPs with VLIW architectures—the SC140, the TMS320C55xx, and the TMS320C6xxx families—make use of their greater parallelism to perform this benchmark in fewer cycles than the ADSP-2106x-C.

- **Execution times**: In terms of total normalized execution time, the ADSP-21065L-C is significantly slower than the other floating-point processors benchmarked—the Analog Devices ADSP-21160 and the Texas Instruments TMS320C6701—as shown in Figure 8.2-13. The ADSP-21065L-C is roughly 65% slower than the ADSP-21160-C, and roughly three times as slow as the TMS320C6701. In both cases, much of the difference is attributable to differences in clock speed, with the remainder a result of having a less-powerful architecture than the SIMD and VLIW architectures of the ADSP-21160 and the TMS320C6701. The 66 MHz ADSP-21065L has a BDTImark2000 score of 250.
- **Cost-execution time**: With a price tag of $25.00 (quantity 10,000), the ADSP-21065L is aggressively priced for a floating-point processor. As a result, the ADSP-21065L-C takes first place among floating-point processors, with a cost-execution time product that is significantly better than that of its closest float-

ing-point rival, the Texas Instruments TMS320C6701. It should be noted, however the TMS320C67xx family has reduced-cost, reduced-performance family members that would have significantly better cost-execution times than the TMS320C6701. Also, the ADSP-21065L-C has a cost-execution time that is slightly better than two fixed-point DSP processors: the Motorola DSP56311 and the TMS320C6203. The total normalized cost-execution time products are illustrated in Figure 8.3-13.

- **Energy consumption**: The ADSP-21065L-C has a typical power consumption of 0.93 watts at 3.3 volts and 66 MIPS, which is quite low compared to the other floating-point processors benchmarked. However, in terms of energy consumption, the ADSP-21065L-C places second among the floating-point processors benchmarked, with an energy consumption that is slightly more than double that of the Texas Instruments TMS320C6701. The ADSP-21065L-C's low power consumption is not enough to compensate for its much slower execution time relative to the TMS320C6701, resulting in its higher energy consumption. These results are illustrated in Figure 8.4-13A.

Memory Usage

The focus in the memory usage analysis is on Control benchmark memory usage. Unlike other benchmarks, the Control benchmark is optimized for minimum memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP code. Finally we discuss constant and non-constant data memory usage.

Since memory usage is independent of the state of the cache, we do not distinguish between the ADSP-2106x and the ADSP-2106x-C when discussing memory usage.

Memory Usage

- **Control benchmark memory usage**: Like the ADSP-2116x, the ADSP-2106x's 48-bit instruction word is the widest of all benchmarked processors, resulting in quite high program memory usage. Although the ADSP-2106x's wide instructions allow flexibility for performing several operations within each instruction, on the Control benchmark this feature is not used often enough to make up for the width of the instruction words. The processor shares last place with the ADSP-2116x on the Control benchmark in terms of total memory usage, as shown in Figure 8.5-9A.
- **Program memory usage**: As can be seen in Figure 8.5-13, the ADSP-2106x's total normalized program memory usage is higher than most of the conventional DSP processors benchmarked, but lower than the processors with SIMD and VLIW architectures (such as the SC140, TMS320C6xxx families, and

ADSP-2116x). The main reason for the ADSP-2106x's high program memory usage relative to other conventional DSP processors is its wide, 48-bit instruction word.

- **Data memory usage**: As shown in Figure 8.5-14 and Figure 8.5-15, the ADSP-2106x constant and non-constant benchmark data memory usage is generally what is expected for a 32-bit DSP processor. In some benchmarks, typically one or two non-constant data words are used to facilitate speed optimizations.

On average across the benchmarks, the ADSP-2106x has higher cycle counts and slower execution times than the other benchmarked floating-point DSP processors. Of the three floating-point processors analyzed in this report, the ADSP-2106x is the only conventional architecture, and it lacks the parallelism and higher clock speeds of the ADSP-2116x and TMS320C67xx.

The ADSP-21065L-C cost-execution time result is better than that of the ADSP-21160 and TMS320C6701; however, as mentioned earlier, lower-cost TMS320C67xx family members may have better cost-execution time performance.

On average across the benchmarks, the ADSP-21065L has lower energy consumption than the ADSP-21160, but higher energy consumption than the TMS320C6701, and it has higher energy consumption than all of the fixed-point DSPs for which energy consumption has been calculated.

The processor's quite high program memory usage on the Control benchmark suggests that poor code density is a disadvantage of this processor. Its code density on the DSP algorithm benchmarks, however, is somewhat better than that on the Control benchmark.

Cost

Price and packaging options for ADSP-2106x processors are shown in Table 7.3-5.

Fabrication Details

According to Analog Devices, the ADSP-21060, ADSP-21061, and ADSP-21062 are fabricated in a 0.45 μm two-metal-layer CMOS process. The ADSP-21065L is fabricated in a 0.35 μm three-metal-layer CMOS process.

Development Tools

Analog Devices provides "VisualDSP," an integrated development environment for the ADSP-2106x that includes an assembler, C compiler, linker, ROM splitter, and cycle-accurate instruction-set simulator. VisualDSP runs under Windows 95, Windows 98,

Part	Speed (MIPS)	Voltage (V)	Package	Price (Qty. 10,000)
ADSP-21060	33.3	5.0	240 MQFP	$199.58
ADSP-21060	44.0	5.0	240 MQFP	$254.37
ADSP-21060	33.3	5.0	240 CQFP	$332.64
ADSP-21060	40.0	5.0	240 CQFP	$423.95
ADSP-21060L	33.3	3.3	240 MQFP	$213.46
ADSP-21060L	40.0	3.3	240 MQFP	$266.82
ADSP-21060L	33.3	3.3	240 CQFP	$355.95
ADSP-21060L	40.0	3.3	240 CQFP	$445.20
ADSP-21061	50.0	5.0	240 MQFP	$46.31
ADSP-21061	40.0	5.0	240 MQFP	$43.05
ADSP-21061	33.3	5.0	240 MQFP	$42.00
ADSP-21061L	44.0	3.3	240 MQFP	$31.19
ADSP-21061L	40.0	3.3	225 PBGA	$31.19
ADSP-21061L	40.0	3.3	240 MQFP	$27.41
ADSP-21062	40.0	5.0	225 PBGA	$101.64
ADSP-21062	40.0	5.0	240 MQFP	$92.40
ADSP-21062	33.3	5.0	240 MQFP	$82.22
ADSP-21062L	40.0	3.3	225 PBGA	$114.35
ADSP-21062L	40.0	3.3	240 MQFP	$103.95
ADSP-21062L	33.3	3.3	240 MQFP	$92.40
ADSP-21065L	66.7	3.3	240 MQFP	$25.00
ADSP-21065l	66.7	3.3	225 PBGA	$27.00
ADSP-21065L	60.0	3.3	240 MQFP	$20.00
ADSP-21065L	60.0	3.3	225 PBGA	$26.00

TABLE 7.3-5. ADSP-2106x price and package summary as of June 1, 2000.

and Windows NT on IBM PC-compatible computers. According to Analog Devices, support for the Windows 2000 operating system will be added during 2000.

Analog Devices also offers an "EZ-ICE" emulator (available only for IBM PC-compatibles) and an "EZ-KIT" development starter kit. The development starter kit consists of a development board and software development tools.

> *Analog Devices' basic software development tools for the ADSP-2106x are fairly good. The ADSP-2106x tools provide a reasonably consistent interface with those for the ADSP-21xx fixed-point processors.*

Third-party support for the ADSP-2106x is available in the form of development boards, function and application software libraries, and real-time operating systems. White Mountain DSP, a DSP development tools company which was acquired by Analog Devices in 1999, offers emulators for use with IBM PC-compatibles and Sun workstations.

Applications Support

Analog Devices supports the ADSP-2106x with user's manuals, an applications handbook, and a quarterly newsletter, *DSPatch*. The main documentation for ADSP-2106x processors is the *ADSP-2106x SHARC User's Manual.* Analog Devices' DSP technical support group can be reached over the Internet. A large amount of information is available from Analog Devices' website. Data sheets are available for the individual ADSP-2106x processors.

> *Analog Devices' user's manuals and application handbooks maintain their position as the best in the industry. They are thorough, clear, and well organized.*

Advantages

- Rich, orthogonal instruction set
- Algebraic assembly language
- Special instructions for FFT, DCT kernel computations
- Good operand-unrelated parallel move support
- Conditional execution of most instructions
- 32-bit or 40-bit floating-point arithmetic with partial support for IEEE-754; 32-bit fixed-point arithmetic
- Barrel shifter
- Multiprecision arithmetic support
- Exponent detect instruction
- Bit manipulation and bit-field manipulation instructions

- Pipeline is typically invisible to the programmer
- Many address registers (16 including shadow registers) with individual modifier and modulo registers
- Shadow registers provided for arithmetic and address registers; single-cycle switching of register set under program control
- Instruction cache for accelerating data accesses to program memory
- Flexible external memory interface with support for page-mode DRAM; ADSP-21065L supports SDRAM
- Powerful DMA controller
- Large, unified address space
- Powerful hardware looping; nestable multi-instruction looping
- Short interrupt latency
- Flexible interrupt handling; fast interrupt capability; shadow registers
- Integrated multiprocessor communications ports (except on the ADSP-21061 and ADSP-21065L)
- Two TDM serial ports with companding support
- 1X external clock (all except ADSP-21065L)
- JTAG debug port with boundary scan
- Good documentation
- Good cost-execution time results on the BDTI Benchmarks (for a floating-point processor)

Disadvantages

- Dissimilar program and data word sizes complicates system design involving external memory
- Large instruction word can increase system cost

7.4 Analog Devices ADSP-2116x Family

**BDTImark2000 Score:
410 at 80 MHz**

Introduction

The ADSP-2116x is the second generation of the "SHARC" family of 32-bit floating-point DSP processors from Analog Devices. The ADSP-2116x (also called "Hammerhead") is assembly source-code compatible with the ADSP-2106x and retains the 48-bit wide ADSP-2106x instruction set, with a few additions. The ADSP-2116x processor targets many of the same applications as the ADSP-2106x, including military, audio, voice recognition, imaging, and telephony applications, especially those requiring multiprocessor systems. The first ADSP-2116x family member, the ADSP-21160, was announced in June 1998. It is currently sampling at 80 MHz (80 MIPS) using a core voltage of 2.5 volts with 3.3-volt I/O. Analog Devices expects the ADSP-21160 to become available in production quantities by October 2000. In September of 2000, Analog Devices introduced the next member of the ADSP-2116x family, the ADSP-21161. This family member is not currently available, but is expected to begin sampling in the fourth quarter of 2000, according to Analog Devices. The ADSP-21161 is projected to operate at 100 MHz and 1.8 volts.

The ADSP-2116x is very similar to the ADSP-2106x; however, the ADSP-21160 is not pin-compatible with any ADSP-2106x family member. It differs from the ADSP-2106x primarily in its wider data buses, duplicated computational units supporting SIMD-style (Single-Instruction, Multiple-Data) operation, and higher clock speeds. Because the two families are so similar, this analysis highlights only the differences between the ADSP-2116x and the ADSP-2106x. Readers should refer to the ADSP-2106x analysis, Section 7.3, for a full discussion of ADSP-2106x architecture, and for details of features common to both families. A summary of current and forthcoming ADSP-2116x family members is provided in Table 7.4-1.

Part	Max. Speed (MHz)	On-Chip Memory	Comments
ADSP-21160	80	512 Kbytes	Six communication ports, host interface, timer, 14 DMA channels, two serial ports, six link ports, external parallel bus.
ADSP-21161[a]	100	125 Kbytes	Two communication ports, host interface, 14 DMA channels, four serial ports, two link ports, external parallel bus, SPI-compatible port, SDRAM controller.

TABLE 7.4-1. ADSP-2116x family summary. On-chip memory size is listed in bytes since on-chip memory can be divided into 48-bit-wide program, 32-bit-wide data, and 16-bit-wide data memory in many different configurations.

[a] Not yet available

Architecture

The ADSP-2116x is based on two identical 32-bit floating-point data paths. These data paths are identical to the data path on the ADSP-2106x. The architecture of the ADSP-21160 is shown in Figure 7.4-1.

Data Path

The ADSP-2116x data paths, denoted "X" and "Y" by Analog Devices, each contain one multiplier, one shifter, one ALU, and one 16 × 40-bit register file with shadow registers.

The programmer can switch between SISD (Single-Instruction, Single-Data) mode and SIMD mode by toggling a mode control bit. When in SISD mode the processor exe-

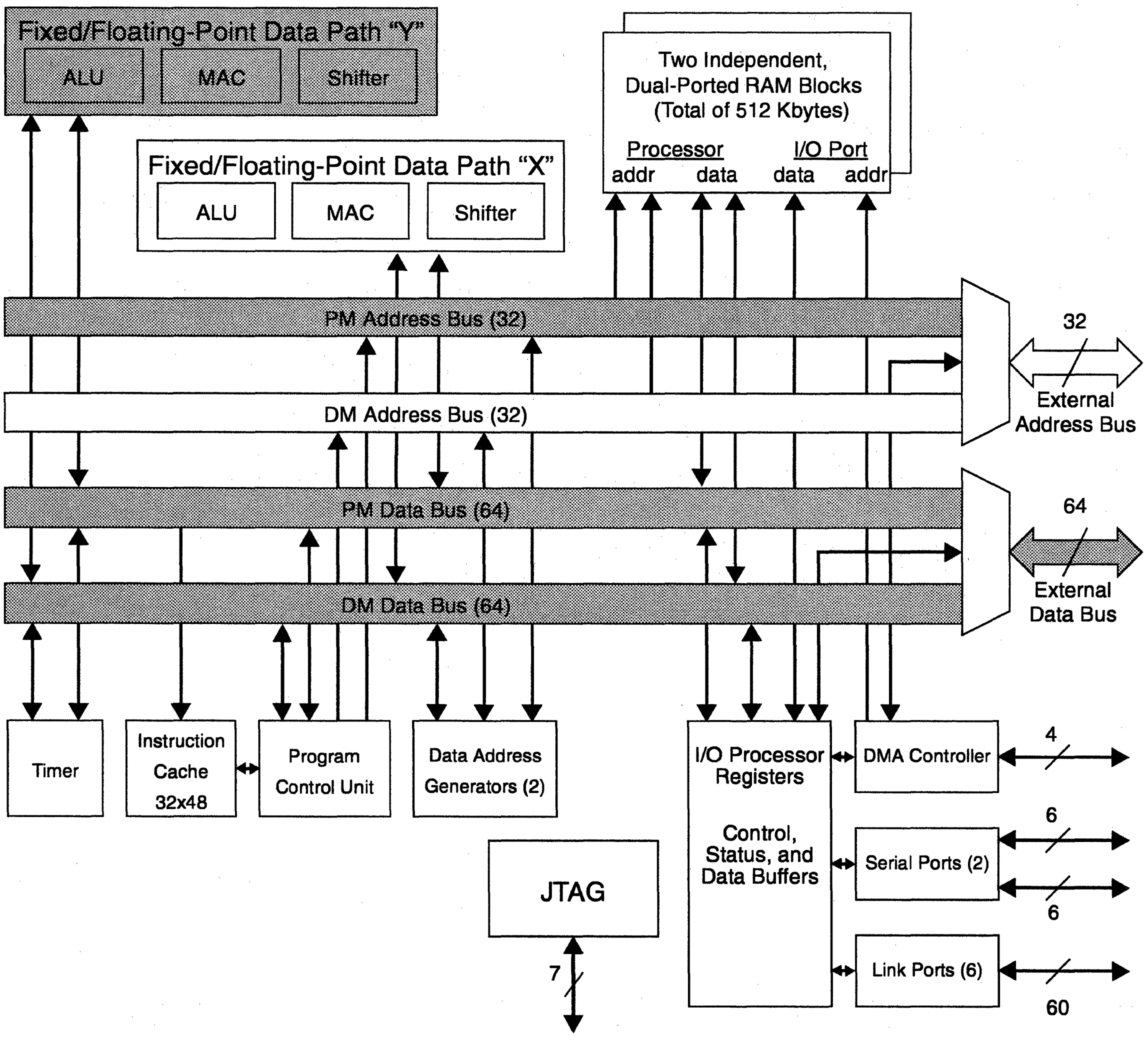


FIGURE 7.4-1. ADSP-21160 architecture. Shaded items indicate differences from the ADSP-2106x.

cutes instructions in a fashion similar to an ADSP-2106x processor. The SIMD mode enables data path "Y," which then executes the same instruction as data path "X," but on the register file associated with data path "Y".

> *The SIMD-style architecture is very powerful for certain algorithms and applications that lend themselves to data-parallel execution. It is an architectural enhancement which is found in many high-performance general-purpose processors, but few DSPs have serious SIMD capabilities. The implementation as two separate, duplicated computational entities is different from, e.g., the Lucent DSP16xxx, where a single computational unit performs parallel operations on a single register file. The two separate register files in the ADSP-2116x do not provide the same flexibility for interchanging results between execution units in the two data paths as found on the Lucent DSP16xxx, but do facilitate an orthogonal instruction set and clean programming paradigm.*

Data can be moved between the two register files and between the register files and memory. The behavior of data-move instructions is dependent upon the state of the SIMD mode bit, as explained in detail under the sections *Memory System* and *Address Generation Units*, below. For more detailed information on the computational units of the ADSP-2116x, please refer to the *Data Path* section of the description of the ADSP-2106x in Section 7.3.

Memory System

The memory system of the ADSP-2116x is similar to the memory system of the ADSP-2106x, with dual-ported on-chip SRAM organized in two blocks and running at full processor speed. Like the 2106x, the 2116x includes a 32-word on-chip instruction cache. A key difference between the ADSP-2116x and the ADSP-2106x is the on-chip data bus width. The ADSP-2116x data buses have been widened to 64 bits to accommodate the extra bandwidth needed by the additional data path. When executing instructions from cache, the processor can load or store two pairs of 32-bit words per instruction cycle, resulting in a bandwidth of 320 million 32-bit words/second for an 80 MIPS ADSP-21160. When the processor is executing instructions from internal memory, instead of from cache, the bandwidth is reduced to 160 million 32-bit words/second. In addition to transfers between registers and on-chip memory performed by software, the DMA controller can transfer 160 million 32-bit words/second between internal memory and peripherals or external memory.

> *The size of the on-chip memory is large compared to other DSPs; e.g., the TMS320C6701 has only a quarter as much memory. The large on-chip memory helps compensate for the 48-bit wide instructions.*

External Memory Interface

The external memory interface of the ADSP-2116x is the same as that of the ADSP-2106x, except that the data bus width is increased to 64 bits. The off-chip data bus runs at half the core speed; hence, an 80 MIPS ADSP-21160 has an external memory bandwidth of eighty million 32-bit words per second.

Address Generation Units

The two ADSP-2116x address generators, DAG1 and DAG2, operate in the same fashion as on the ADSP-2106x, with a few exceptions: DAG2 can now access the full 32-bit address space; a 64-bit long word is supported for data moves; and a number of new SIMD-oriented data moves to and from the two register files have been defined. This section describes only those cases where the data address generators of the ADSP-2116x extend the functionality of the ADSP-2106x. For more general information, refer to the *Address Generation Units* section of the ADSP-2106x analysis in Section 7.3.

The address generators can specify four word sizes: 64-bit long word, 40-bit extended precision word, 32-bit normal word, and 16-bit short word. This allows the transfer of one word of any size with each instruction using a single address generator. Data transfers can be specified with different sizes of operands; e.g., DAG2 can address one 40-bit extended precision word while DAG1 simultaneously addresses two adjacent 32-bit words.

The ADSP-2116x provides several modes for specifying the source or target register(s) for a data move. Three major modes of operation are:

- SISD mode, where only data path "X" is enabled and therefore no data is moved to or from the register file in data path "Y"
- SIMD mode, where both data paths are enabled, and different data is moved to or from both register files
- Broadcast mode, where both data paths are enabled, and the same data is sent to both data paths when a load is performed

In SISD mode, addressing a 32-bit word results in a transfer that is essentially the same as in the ADSP-2106x. The word is transferred to or from the register specified. When addressing a 64-bit long word, the long word is transferred to or from two adjacent registers. The mapping between the 32-bit halves of the long word and a register pair is determined by the register addressed by the DAG. If an even register is specified as the source or destination (e.g., R0, R2, etc.), the lower 32 bits of the 64-bit long word will be mapped to the even register and the upper 32 bits will be mapped to the next (odd) register. If an odd register is specified in the instruction, the lower 32 bits of the 64-bit long word will be mapped to the odd register and the upper 32 bits of the 64-bit long word will be mapped to the prior (even) register. Hence, the order of words in memory can be reversed when moving pairs of words into registers or when moving from registers to memory. In

SISD mode, both DAGs can be used to transfer a 64-bit long word to the same register file in the same instruction.

In SIMD mode, addressing a 32-bit word results in 64 bits of data being transferred: the 32-bit word that was addressed is moved to or from the register specified, and the 32-bit word following the addressed word in memory is moved to or from the corresponding register in the other register file. For example, explicitly loading the 32-bit word at address 0x40000 into register "R0" in register file "X" will also cause the 32-bit word at address 0x40001 to be loaded into register "R0" in register file "Y." The 32-bit word addressed during the move does not have to lie on a 64-bit aligned address. Explicitly loading or storing a 32-bit word at a non-64-bit-aligned address (e.g., 0x40001) will also load or store the 32-bit word at the next address (e.g., 0x40002).

Addressing a 64-bit long word in SIMD mode results in data being transferred to or from only one register file. The other DAG may be used to simultaneously transfer another 64-bit long word to or from the other register file. However, it is not possible to make two simultaneous 64-bit data moves to or from the same register file in this mode.

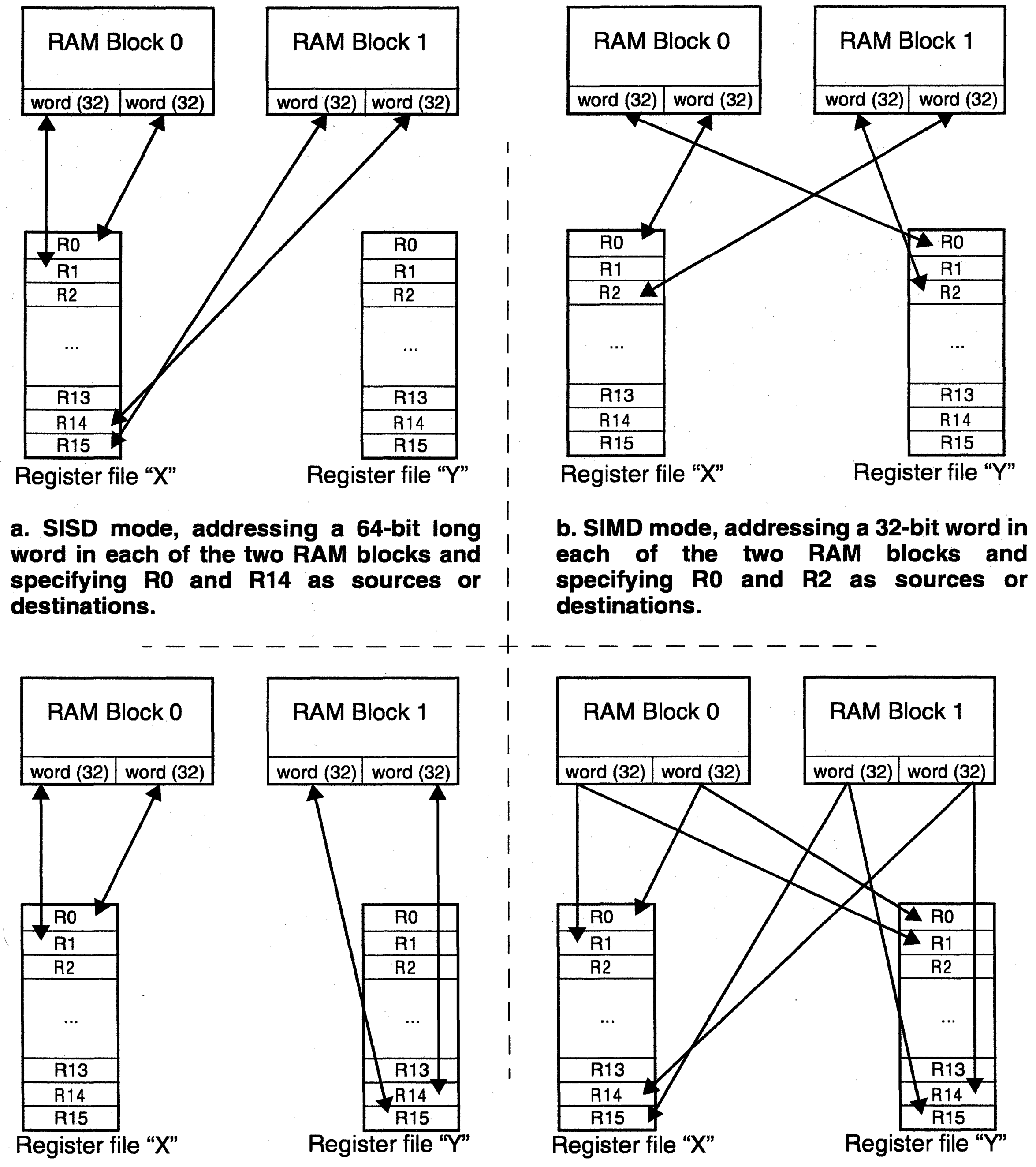
Broadcast mode is only available for loads that use the second address register in each DAG, i.e., I1 and I9. Two software-controlled flags, one for I1 and one for I9, enable broadcast mode. When a broadcast flag is set, loads using the corresponding address register will load a single value into both register files. The broadcast flags do not change the operation of stores from registers to memory.

Figure 7.4-2 illustrates some of the types of loads and stores that can be performed with four 32-bit words on the ADSP-2116x.

> *The use of mode bits to alter the operation of loads and stores ensures very flexible addressing, and helps maintain the orthogonality of the instruction set without increasing the instruction word width and without increasing the number of data address generators. The flexibility in addressing comes at the cost of reduced code readability, however, since an instruction that explicitly loads a register from memory may implicitly load other registers in either data path, depending on the status of the mode bits. These implicit operations put higher demands on the programmer to keep track of data movement, but also make programming easier by adding flexibility to data moves.*

Pipeline

The pipeline of the ADSP-2116x is very similar to the pipeline of the ADSP-2106x, with one improvement: the ADSP-2116x uses a finer-grained test to detect the possibility of a read-after-write sequence which would cause an interlock. In the ADSP-2106x the use of *any* address register in the data address generator would result in a one-cycle pipeline stall. In the ADSP-2116x, read-after-write dependencies are instead



a. SISD mode, addressing a 64-bit long word in each of the two RAM blocks and specifying R0 and R14 as sources or destinations.

b. SIMD mode, addressing a 32-bit word in each of the two RAM blocks and specifying R0 and R2 as sources or destinations.

c. SIMD mode, addressing a 64-bit long word in each of the two RAM blocks and specifying R0 in register file "X" and R14 in register file "Y" as sources or destinations.

d. Broadcast mode, performing 64-bit long word loads to R0 and R14.

FIGURE 7.4-2. Examples of ADSP-2116x data transfers.

resolved over pairs of registers in the data address generators; e.g., if an address is written to I2, then the use of I2 or I3 in the next instruction would incur a one cycle pipeline stall.

> *The improved testing for data dependencies in the data address registers is an advantage which may modestly improve the performance of the ADSP-2116x in some applications.*

Instruction Set

The ADSP-2116x instruction set is the same as that of the ADSP-2106x, with the addition of an instruction which swaps data between the two register files and a 32-bit unsigned fixed-point compare. The ADSP-2116x is able to maintain the same instruction set as the ADSP-2106x without major modifications, even though an extra data path has been added, by the use of a few new mode bits that modify the operation of the existing instructions.

> *The 32-bit unsigned compare complements the 32-bit signed compare in the ADSP-2106x instruction set, and adds to the orthogonality of the ADSP-2116x instruction set.*

When in SISD mode, the ADSP-2116x is assembly source-code compatible with the ADSP-2106x family of processors.

> *The assembly source-code compatibility between the ADSP-2106x and the ADSP-2116x is an unusual level of compatibility between two generations of DSP processor families, and will make it easier for programmers to migrate from the ADSP-2106x to the ADSP-2116x family of processors. ADSP-2106x code should be re-optimized, however, to take advantage of the new capabilities.*

Assembly Language Format

For a description of the assembly language format, please refer to *Assembly Language Format* for the ADSP-2106x, in Section 7.3.

Parallel Move Support

In addition to the parallel moves described for the ADSP-2106x in Section 7.3, the ADSP-2116x supports double 32-bit-word moves, described above in the *Address Generation Units* section. These instructions allow the ADSP-2116x to load four 32-bit words, store four 32-bits words, or load two 32-bit words and store two 32-bit words in parallel with an arithmetic operation. However, this requires that 32-bit data be arranged as pairs in memory.

> *The ADSP-2116x provides unusual flexibility in the use of parallel data moves.*

Orthogonality

The ADSP-2116x instruction set is very orthogonal by DSP processor standards. This is possible due to the processor's 48-bit wide instruction words and its use of mode bits.

Execution Times

The ADSP-2116x instruction execution times (in terms of cycles) are the same as those of the ADSP-2106x.

Instruction Set Highlights

Noteworthy features, where the ADSP-2116x instruction set differs from the ADSP-2106x instruction set, include:

- Parallel, independent, conditional execution of nearly all instructions in the two data paths. In SIMD mode, conditionally executed instructions whose execution is predicated on register comparisons will execute differently in the two data paths, since the two data paths have separate status flags.
- Any combination of two parallel double-word moves (two loads, one load and one store, or two stores) with a computation.

For further instruction set highlights, refer to the *Instruction Set Highlights* section of the ADSP-2106x analysis in Section 7.3.

Execution Control

Clocking

The ADSP-2116x receives its clock from an external oscillator, and is equipped with a PLL that enables the processor to run at different multiples of the input clock, where the input:core clock rate ratio can be: 1:2, 1:3, or 1:4. The processor must be reset to change the clock ratio.

Hardware Looping

The ADSP-2116x provides the same hardware looping support as the ADSP-2106x.

Interrupts

ADSP-2116x interrupt sources are the same and interrupt handling behavior is the same as on the ADSP-2106x, with an additional six interrupt sources: a non-maskable emulator interrupt, four additional DMA interrupts, and an interrupt for conditions that arise due to programming errors (e.g., inadvertent IOP register access).

For more information on interrupts, please refer to the *Interrupts* section of the ADSP-2106x analysis Section 7.3.

Stack

The ADSP-2116x provides the same stack support as the ADSP-2106x.

Bootstrap Loading

ADSP-2116x bootstrap loading is the same as on the ADSP-2106x.

Peripherals

The peripherals of the ADSP-2116x are similar to those of the ADSP-2106x; for more information, please refer to the *Peripherals* section of the ADSP-2106x analysis in Section 7.3.

Notable additions present on the ADSP-21160 and ADSP-21161 are four extra DMA channels (for a total of 14), increased bus width of the DMA channels from 32 bits to 64 bits, and the capability of the DMA controller to interleave 32-bit words from two channels in memory, by packing a 32-bit word from each channel into a 64-bit double word and storing it in memory. The ADSP-21160 provides a dedicated DMA channel for each peripheral, instead of sharing some DMA channels as in the ADSP-2106x.

> *The DMA controller's ability to interleave data from two channels in memory is a good match with the two parallel data paths, and will be very useful in many DSP applications where the same type of data processing is applied to two data streams. This is often the case in sound-processing applications such as AC-3 and ProLogic.*

The ADSP-21160 has two serial ports, which run at a maximum of half the core clock rate. This gives the serial ports on a 80 MIPS ADSP-21160 a maximum transfer rate of 40 Mbps. The four serial ports on the ADSP-21161 support 128-channel TDM and I^2S.

The six link ports on the ADSP-21160 (and the two on the ADSP-21161) are slightly different from the link ports on the ADSP-2106x. They are 8 bits wide instead of 4 bits and can only be clocked once each processor instruction cycle, instead of twice each instruction cycle. A link port on the ADSP-21160 can be configured to be compatible with the 4-bit link ports on the ADSP-2106x.

The ADSP-21160 external memory interface supports a burst mode, for more efficient transfer of blocks of data to or from another processor or off-chip device. As with the ADSP-2106x, the ADSP-21160 external memory interface also serves as a host port and as a means to access the on-chip memory of other ADSP-21160 processors in a multiprocessor system.

On-Chip Debugging Support

ADSP-2116x debugging support is the same as that of the ADSP-2106x.

Power Consumption and Management

The power management facilities of the ADSP-2116x are the same as those of the ADSP-2106x. Typical power consumption of the ADSP-21160 running at 80 MHz and 2.5 volts is 2.35 watts, according to data sheets from Analog Devices.

Benchmark Performance

The ADSP-2116x has been benchmarked with the BDTI Benchmarks™. Overall benchmark results for all benchmarked processors are presented in Chapter 8, *BDTI Benchmark™ Results*. We summarize and analyze ADSP-2116x benchmark performance in the paragraphs below. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not reflect the processor's instruction cycle rate; therefore, lower instruction cycle counts imply a more powerful architecture, but do not imply faster speed. Next we discuss benchmark execution times and cost-execution time products, indicating processor speed and cost-performance, respectively. We then discuss the processor's energy consumption, which reflects the energy consumed by the processor in order to perform a task. Finally, we discuss the processor's memory usage. We divide the memory usage discussion into three parts: Control benchmark memory usage, overall benchmark program memory usage, and benchmark data memory usage.

Because the ADSP-2116x uses an instruction cache, the benchmark analysis presented in Chapter 8, *BDTI Benchmark™ Results*, treats the ADSP-2116x as two distinct processor families. One family (referred to as the ADSP-2116x) assumes that the cache is empty before each benchmark is executed. The other family (referred to as the ADSP-2116x-C) assumes that the processor's instruction cache has been preloaded by executing the benchmark code once prior to measuring the instruction cycle counts for that benchmark. The total normalized cycle count is less than 10% lower for the ADSP-2116x-C than for the ADSP-2116x. For this reason, we have focused on the ADSP-2106x-C in the analysis below. However, the effect of cache preloading is more significant on single-sample benchmarks, where the average difference between total normalized results for the ADSP-2106x and ADSP-2106x-C is more than 20%. When using these benchmark results to gauge the performance of the ADSP-2116x family in a potential application, we urge readers to carefully consider how the processor's cache will perform in that application.

Execution Performance

- **Instruction cycle counts:** Total normalized instruction cycle counts are found in Figure 8.1-13. The ADSP-2116x-C's total normalized instruction cycle count is the fourth-lowest among the benchmarked processors. The ADSP-2116x-C's low total normalized cycle count is due to the powerful instruction set that it shares with the ADSP-2106x-C and to its two-way SIMD capability. In theory, the SIMD capability of the ADSP-2116x-C gives it double the computational throughput of the ADSP-2106x-C for many operations. However, for some benchmarks, it is impossible or impractical to process the input data in a SIMD fashion. For others, making use of SIMD requires additional overhead, such as combining partial results from each data path to form the final result of the computation. Also, since each data path operates on its own register file, it is sometimes necessary to copy data from one register file to another. These factors prevent the ADSP-2116x-C from achieving a total normalized cycle count that is half that of the ADSP-2106x-C; the total normalized cycle count result for the ADSP-2116x-C is only about 25% lower than that of the ADSP-2106x-C.

On the **Single-Sample FIR** filter, the ADSP-2116x-C performs the convolution at a rate of two taps per instruction cycle, but the additional overhead outside the inner loop that is required to enable the use of both data paths prevents the cycle count from being significantly lower than that of the ADSP-2106x-C. Nevertheless, the ADSP-2116x-C has a low cycle count on this benchmark—only the DSP164xx and SC140 have lower results on the Single-Sample FIR filter.

The **Vector Dot Product**, **Vector Add**, and **Vector Maximum** benchmarks lend themselves well to SIMD implementation, and the ADSP-2116x-C has low cycle counts on all three.

On the **Control** and **Bit Unpack** benchmarks, which do not lend themselves to SIMD implementation, the ADSP-2116x uses the same implementation as the ADSP-2106x. Nevertheless, like the ADSP-2106x-C, the ADSP-2116x-C has relatively low cycle counts on these benchmarks. On the Control benchmark, the ADSP-2116x-C takes advantage of conditional instruction execution and optionally delayed branches to keep cycle counts down. On the Bit Unpack benchmark, the ADSP-2116x-C has lower cycle counts than many benchmarked DSPs, due in particular to its powerful bit-field manipulation capabilities and support for delayed branches. However, the benchmarked DSPs with VLIW features—the SC140, the TMS320C55xx, and the TMS320C6xxx families—are able to make use of their parallelism on this benchmark, while the ADSP-2116x is not able to take advantage of its SIMD capability. Since the Bit Unpack benchmark involves unpacking words in sequence from an array, there is a dependency between the computations in one iteration and the next that makes SIMD-style parallelism difficult to apply.

On the **FFT** benchmark, the ADSP-2116x-C cycle count is considerably lower than those of most other processors. The implementation is a radix-4 FFT that makes good use of the processor's SIMD capability, its large register set, and a specialized instruction that allows simultaneous multiply and dual add/subtract operations (taking somewhat restricted operands) with up to two operand-unrelated parallel data moves. Only the SC140, the TMS320C62xx, and the TMS320C64xx have lower cycle counts on this benchmark.

The ADSP-2116x-C cycle count on the **Viterbi** benchmark is relatively low. Its ability to perform the add-compare-select phase of Viterbi decoding efficiently is aided by its flexible parallel move support and its dual add/subtract operation, which is useful for calculating path metrics. Its SIMD capability allows it to parallelize the add-compare-select operations. However, the ADSP-2116x-C has a significantly higher cycle count on this benchmark than that of the Lucent DSP164xx, which has special support for Viterbi decoding, and also higher than those of the benchmarked DSPs with VLIW architectures.

- **Execution times:** In terms of speed, the ADSP-21160-C places between the two other floating-point processors benchmarked—its total normalized execution time is 40% faster than that of the ADSP-21065L-C but 85% slower than that of the TMS320C6701. (The ADSP-21160-C has lower cycle counts than the TMS320C6701 for many benchmarks, but this is outweighed by the TMS320C6701's much higher clock speed.) The 80 MHz ADSP-21160-C has a BDTImark2000 score of 410.
- **Cost-execution time:** The combination of the ADSP-21160's roughly average execution times and its relatively high price tag of $99.00 (quantity 10,000) gives it a poor total normalized cost-execution time result, the worst among the processors for which cost-execution time results were calculated.
- **Energy consumption:** The ADSP-21160's power consumption of 2.35 watts is high, even for a floating-point DSP processor. As a result, the ADSP-21160's total normalized energy consumption is the highest among the benchmarked processors for which energy consumption has been calculated. The total normalized energy consumption results are shown in Figure 8.4-13A.

Memory Usage

The focus in the memory usage analysis is on Control benchmark memory usage. Unlike other benchmarks, the Control benchmark is optimized for minimum memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP code. Finally we discuss constant and non-constant data memory usage.

Since memory usage is independent of the state of the cache, we do not distinguish between the ADSP-2116x and the ADSP-2116x-C when discussing memory usage.

- **Control benchmark memory usage:** The ADSP-2116x uses the same implementation as the ADSP-2106x for this benchmark. Like the ADSP-2106x, the ADSP-2116x has an extremely wide (48-bit) instruction word. Although the ADSP-2116x's wide instructions allow flexibility for performing several operations within each instruction, on the Control benchmark this feature is not used often enough to make up for the width of the instruction words. The processor shares last place with the ADSP-2106x on the Control benchmark in terms of memory usage. The ADSP-2116x's total memory use on this benchmark is about 55% above average, as shown in Figure 8.5-9A.

- **Program memory usage:** The ADSP-2116x's total normalized program memory usage is illustrated in Figure 8.5-13. Only the VLIW TMS320C62xx, TMS320C64xx, and TMS320C67xx DSPs have higher program memory usage. Like the ADSP-2106x, the ADSP-2116x uses wide, 48-bit instructions, increasing program memory usage. In addition, the ADSP-2116x has somewhat higher memory usage than the ADSP-2106x on most benchmarks, since the ADSP-2116x requires additional instructions to re-arrange data for SIMD processing or to combine results from its two data paths.
- **Data memory usage:** The ADSP-2116x constant and non-constant benchmark data memory usage is generally what is expected for a 32-bit DSP processor. In some benchmarks, a few additional non-constant data words are used to facilitate speed optimizations.

> *The ADSP-2116x provides a significant speed-up and easy migration path for customers currently using the ADSP-2106x, but is not nearly as fast as the TMS320C6701.*
>
> *On average across the benchmarks, the ADSP-2116x has moderately fast execution times for a floating-point processor, but its cost-execution time performance is poor, and its energy consumption is high.*
>
> *The processor's high memory usage on the Control benchmark and high overall program memory usage suggest that poor code density is a disadvantage of this processor.*

Cost

The projected production prices for the ADSP-21160 and ADSP-21161 are shown in Table 7.4-2.

Part	Speed (MHz)	Package	Voltage (V)	Price (Qty. 10,000)
ADSP-21160	80	400 PBGA	2.5/3.3	$99.00
ADSP-21161N[a]	100	225 PBGA	1.8/3.3	$30.00

TABLE 7.4-2. ADSP-2116x projected price and package summary.

[a] Not yet available.

Fabrication Details

According to Analog Devices, the ADSP-21160 is fabricated in a 0.25 µm five-metal-layer CMOS process.

Development Tools

The ADSP-2116x is supported by the same "VisualDSP" tool set as the ADSP-2106x. For more information, refer to the *Development Tools* section of the ADSP-2106x analysis in Section 7.3. According to Analog Devices, the C compiler for the ADSP-2116x offers access to SIMD mode through a language extension, and is supported by a library of hand-optimized DSP functions developed by Ixthos, Inc.

According to Analog Devices, the "EZ-ICE" emulator described in Section 7.3 may be used with the ADSP-21160. Analog Devices also offers an "EZ-KIT Lite" ADSP-2116x development starter kit, which consists of a development board for IBM PC-compatibles and software development tools. JTAG-based in-circuit emulators that provide similar functionality to the EZ-ICE and connect to the host debugger through either USB, ethernet, or PCI-card interfaces are available from third-party vendors.

Third party support appears limited, but development boards are available. White Mountain DSP, a DSP development tools company which was acquired by Analog Devices in 1999, offers emulators.

Applications Support

The documentation for the ADSP-2116x currently available on Analog Devices' Web site takes the form of two reference manuals and a guide to efficient ADSP-2116x programming. A datasheet is available for the ADSP-21160. Since the ADSP-2116x in SISD mode is assembly source-code compatible with the ADSP-2106x, ADSP-2106x documentation such as the applications handbook should also be useful for ADSP-2116x programmers. Analog Devices' DSP technical support group can be reached via the Internet.

Advantages

- Rich, orthogonal instruction set
- Good operand-unrelated parallel move support
- Conditional execution of most instructions
- 32-bit or 40-bit floating-point arithmetic with partial support for IEEE-754; 32-bit fixed-point arithmetic
- Algebraic assembly language
- Special instructions for FFT, DCT kernel computations
- Barrel shifter

- Bit manipulation and bit-field manipulation instructions
- Multiprecision arithmetic support
- Exponent detect instruction
- SIMD data path, with two multipliers, two ALU units, two barrel shifters, and two register files
- Pipeline is typically invisible to the programmer
- Many address registers (16 including shadow registers) with individual modifier and modulo registers
- Shadow registers provided for arithmetic and address registers; single-cycle switching of register set under program control
- Flexible external memory interface with support for page-mode DRAM
- Powerful DMA controller
- Large, unified address space (4 Gwords)
- Powerful hardware looping; nestable multi-instruction looping
- Short interrupt latency
- Flexible interrupt handling; fast interrupt capability; shadow registers
- Integrated multiprocessor communications ports
- Two TDM serial ports with companding support
- Flexible PLL
- JTAG debug port with boundary scan
- Compatibility with predecessor (ADSP-2106x)

Disadvantages

- Large instruction word can increase system cost
- Dissimilar program and data word sizes complicates system design involving external memory
- Poor BDTI Benchmark cost-execution time performance
- High BDTI Benchmark energy consumption
- Poor BDTI Benchmark program memory usage

7.5 Analog Devices ADSP-TS0xx Family

BDTImark2000 Score: Not Available

Introduction

The ADSP-TS0xx is a family of processors from Analog Devices based on the VLIW "TigerSHARC" core, introduced in October of 1998. The ADSP-TS0xx operates on a variety of data widths, and supports fixed- and floating-point formats. The ADSP-TS0xx family is targeted at high-performance applications, such as those found in the telecommunications industry (wireless base stations, VoN/VoIP concentrators, cable modems, xDSL, encryption, and third-generation ("3G") wireless), and in high-performance signal processing markets (graphics, CAT scan, MRI, ultrasound, radar, sonar), especially those applications requiring multiprocessing. However, Analog Devices states that through the second quarter of 2001, it is only promoting the ADSP-TS0xx for cellular base station applications.

At its introduction in 1998, the TigerSHARC core was projected to operate at 250 MHz and to be available in sample quantities in the first half of 1999; this clock rate has since been derated to the current projection of 120 MHz, and as of this writing (October 2000) the chip is not yet sampling at this speed. The first ADSP-TS0xx family member, the ADSP-TS001, is expected to begin sampling at 120 MHz with a core voltage of 1.8 volts in late 2000. Analog Devices states that an early pin-compatible version of the chip operating at 2.5 volts and a reduced speed (unspecified) were distributed to a few lead customers in early 2000. Because the ADSP-TS0xx simulator was not cycle-accurate at the time of this writing and development boards were not available for hardware timing, BDTI Benchmark™ results are not available for this processor, and there is no BDTImark2000 score currently available. BDTI expects to complete benchmark implementations for the ADSP-TS0xx in the coming months; check BDTI's website (*www.BDTI.com*) for updated BDTImark2000 scores.

Although the ADSP-TS0xx fixed/floating-point data path has many elements in common with that of the ADSP-2116x, the ADSP-TS0xx is neither source- nor object-code compatible with previous Analog Devices processors, and its overall architecture is significantly different from those of earlier Analog Devices processors.

The ADSP-TS0xx can execute up to four 32-bit instructions in a single clock cycle. These instructions are scheduled for execution by the compiler or assembly language programmer. Its VLIW architecture (called "static superscalar" by Analog Devices) allows the ADSP-TS0xx to achieve a high level of parallelism. Instructions operate on 8-, 16-, 32-, or 64-bit integer data, and 32-bit or 40-bit floating-point data. The first (and, at present, only) member of the ADSP-TS0xx family is the ADSP-TS001 (see Table 7.5-1). This device has a relatively large on-chip memory and a complement of peripherals similar to those found on Analog Devices' floating-point DSPs.

The ADSP-TS0xx is noteworthy for its powerful combination of VLIW and extensive SIMD parallelism, its generous on-chip memory, its built-in multiprocessor capabilities, its powerful instruction *set, and its support for a wide range of data types, including 8-bit fixed-point and 32-bit floating-point data.*

Because the ADSP-TS0xx can issue up to four instructions per clock cycle, the term "instruction cycle" is potentially ambiguous when discussing this processor. As used here, "instruction cycle" or "cycle" means the time required to execute a single group of one to four parallel instructions, called an "instruction packet" by Analog Devices. On the ADSP-TS0xx, one instruction cycle is equal in length to one master clock ("CCLK") cycle. Details regarding ADSP-TS0xx clocks are discussed below in the *Clocking* section.

Architecture

As discussed above, the ADSP-TS0xx is a VLIW architecture, and can execute up to four instructions per instruction cycle. ADSP-TS0xx processors are based on two identical fixed/floating-point data paths, two identical address generation units, three 128-bit internal buses, three internal memory banks, and extensive I/O capabilities including support for multiprocessing. The ADSP-TS0xx uses a 32-bit address space and 32-bit instruction words. Figure 7.5-1 illustrates the ADSP-TS0xx architecture..

Data Path

Like Analog Devices' ADSP-2116x floating-point DSP processor (discussed in detail in Section 7.4), the ADSP-TS0xx provides two identical fixed/floating-point data paths (called X and Y), each with three distinct arithmetic units: a multiplier-accumulator, a shifter, and an ALU. Each data path may execute one or two instructions per instruction packet. If one data path executes two instructions in an instruction packet, these instructions may employ any two of the ALU, multiplier, or shifter. Furthermore, an instruction (or two instructions) can be simultaneously executed in both X and Y data paths. Instructions can also be executed in the address generation units, as described later in the *Address Generation Units (AGUs)* section.

The ADSP-TS0xx's use of two separate, identical computational entities is different from enhanced conventional DSPs, such as the Lucent DSP16xxx, where a single computational unit performs par-

Part	Max. Speed (MHz)	On-Chip Memory	Comments
ADSP-TS001[a]	120	192Kx32	32-bit external address bus, 64-bit external data bus, 4 link ports, DMA controller, multiprocessor-oriented external memory interface

TABLE 7.5-1. ADSP-TS0xx family summary.

a. Not yet available.

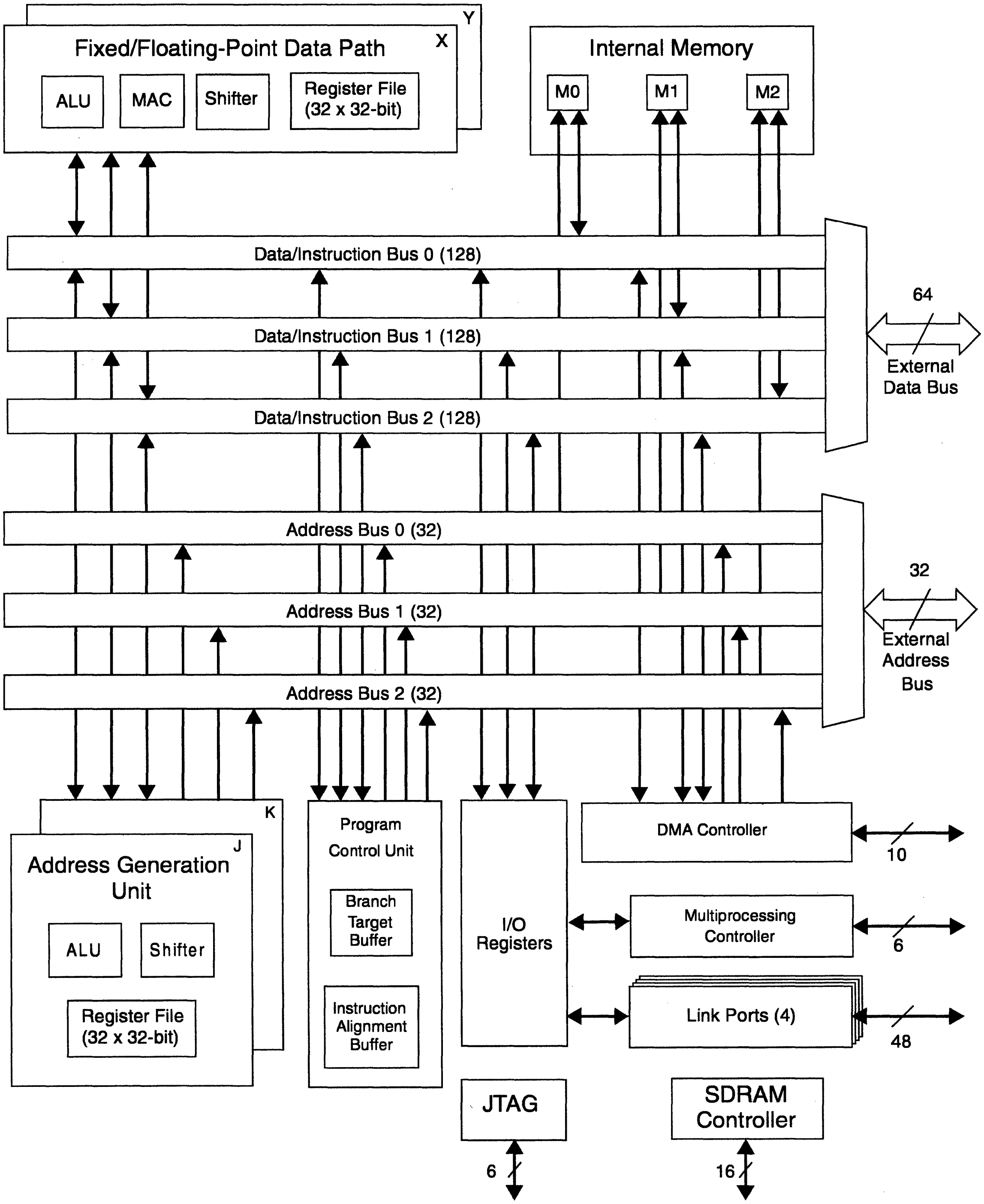


FIGURE 7.5-1. ADSP-TS0xx architecture.

allel operations on a single register file. It is also different from the Analog Devices ADSP-2116x, where use of both data paths requires that they operate in lock-step and execute the same operation.

The ADSP-TS0xx supports six data types: 8-bit fixed-point, 16-bit fixed-point, 32-bit fixed-point, 64-bit fixed-point, 32-bit IEEE-754 floating-point, and 40-bit extended-precision floating-point. The ADSP-TS0xx includes instructions that convert between the various floating-point and fixed-point formats.

The ADSP-TS0xx supports an unusually wide range of data types. For example, it is the only processor in this report that supports 8-bit fixed-point and 32-bit floating-point data types. This makes the ADSP-TS0xx particularly interesting for applications that process many different kinds of signals; for example, video, speech, and audio.

On the ADSP-TS0xx, the multiplier-accumulator, the shifter, and the ALU of each data path access a register file (one file per data path) containing thirty-two 32-bit general-purpose registers. Inputs to most arithmetic operations come from the register file, and results of most arithmetic operations are delivered to the register file. The ALU, multiplier-accumulator, and shifter each have additional dedicated registers, discussed below. Unlike earlier Analog Devices processors, the ADSP-TS0xx does not include shadow registers. 40-bit floating-point data is stored in pairs of adjacent general-purpose registers.

The ADSP-TS0xx supports two types of SIMD operation. As on the ADSP-2116x, a single instruction can control both data paths, so that the same operation is executed in both data paths using different operands. In addition, a single instruction can cause an execution unit in a single data path to perform SIMD operations on multiple sets of input operands. Using SIMD operations, up to eight 8-bit, four 16-bit, or two 32-bit fixed-point operands may be processed by each data path, producing individual results as wide as 32 bits. A 64-bit double-precision integer result can also result from two 64-bit operands, again in each data path. Two or four sequentially numbered registers in the register file may be addressed as a group in one instruction by each data path (discussed further in the *Assembly Language Format* section). By doing so, one data path may produce SIMD result groups of up to 128 bits via one instruction; for example, multiplying four 16-bit operands by four 16-bit operands to produce four 32-bit products.

The two types of SIMD can be combined so that a single instruction causes the two data paths to perform identical SIMD operations. We call this "hierarchical SIMD."

SIMD is an architectural enhancement seen in many high-performance general-purpose processors, but few DSPs have serious SIMD capabilities. The ADSP-TS0xx's "hierarchical SIMD" is particularly unusual and very powerful for certain algorithms and applications.

In addition to the usual fixed- and floating-point arithmetic operations, the data path ALUs support average ([A+B]/2, [A-B]/2), increment, decrement, minimum, maximum, compare (signed and unsigned), clipping (saturation where the limit is specified as an operand of the instruction), and simultaneous add/subtract ([A+B], [A-B]). As detailed in the *Instruction Set* section below, ALU operations can produce up to eight 8-bit, four 16-bit, two 32-bit, or one 64-bit fixed-point results per data path; many of these operations also work with 32- and 40-bit floating-point operands.

> *Simultaneous add/subtract is useful for FFT and Viterbi implementations.*

There are special instructions to combine a group of 8- or 16-bit SIMD operands. For example, the SIMD SUM instruction adds four 8- or 16-bit quantities, accumulating one 32-bit sum. The ADSP-TS0xx supports scaled conversions between fixed-point and IEEE floating-point formats.

The ALU provides the following status flags dependent on the result of the last operation: fixed-point zero/floating-point underflow, negative, overflow, floating-point invalid, and carry. There are additional “sticky” bits for fixed-point overflow, floating-point underflow, floating-point overflow, and floating-point invalid. (A sticky bit, once set by the processor, remains set until cleared under program control.) For SIMD instructions in general, a status bit is determined by the logical *or* of the corresponding status bits of the individual results.

> *As is sometimes seen on SIMD architectures, the ADSP-TS0xx only includes one set of ALU status bits for multiple SIMD operands. In a SIMD operation, the programmer cannot readily detect which of the SIMD operands set which status bit.*

The data path shifter units perform single-bit manipulation, bit-field manipulation, and rotation, logical, or arithmetic shifting operations of up to 64 bits left or right. Single data words of up to 64 bits may be shifted in each data path. As in the ALU, up to eight 8-bit, four 16-bit, or two 32-bit fixed-point operands may also be handled as SIMD operands in each data path, producing results as wide as 32 bits each. The shifter does not make use of the carry bit. The shifter in each data path has its own status flags for indicating zero and negative results. Using dedicated registers, the shifter supports a bit FIFO, which is useful for depositing or extracting a field of varying length into or from a contiguous bit stream. To use this facility, the programmer assigns contiguous registers to hold part of a bit stream. Two other contiguous registers hold control information, pointing to the next bit in the bit stream to be processed and specifying the number of bits.

> *The bit-stream processing capability of the ADSP-TS0xx shifter is extremely useful in implementing compression and decompression of multimedia data.*

The shifter supports exponent detection and can extract, in one instruction cycle, the exponent for eight 16-bit block-floating-point operands.

> *The capability of the ADSP-TS0xx shifter to perform SIMD-style exponent detection is very useful in implementing 16-bit block-floating-point operations, such as may be used in FFTs.*

The multiplier-accumulator units support floating-point and fixed-point integer or fractional multiplication and accumulation. Each multiplier-accumulator provides four 32-bit dedicated accumulation registers (MR0-MR3) and a 32-bit guard bit register (MR4) for fixed-point multiply-accumulates. MR4 provides four guard bits each for eight 16-bit SIMD results, 8 guard bits each for four 32-bit results, or 16 guard bits each for two 64-bit results. Combined accumulator results of up to 80 bits (including guard bits) can be moved to and from 32-bit registers in each data path register file. The multiplier-accumulator units support 20-, 40-, and 80-bit saturation for fixed-point arithmetic. Fixed-point operands are integer or fractional, and may be signed/signed, unsigned/unsigned or signed/unsigned. When operating on fixed-point data, each multiplier-accumulator unit can perform the operations shown in Table 7.5-2. When operating on floating-point data, each multiplier unit performs 32 × 32 → 40-bit or 40 × 40 → 40-bit floating-point multiplications. There is no floating-point multiply-accumulate instruction; floating-point multiplication products can be accumulated using separate ALU instructions. Fixed- and floating-point results may be truncated or rounded to the nearest even number. Floating-point results may also be rounded toward zero.

There is a special instruction to multiply-accumulate two pairs of 16-bit complex numbers (each with 16 bits real, 16 bits imaginary) to produce dual complex results (again each with 16 bits real, 16 bits imaginary). The complex conjugate of one of the operands may optionally be used as an input.

> *This complex multiplication, especially useful in FFTs, is a welcome addition to the instruction set. But for many multimedia appli-*

Number of MACs Per Data Path	Operand Size (Fixed-Point)	Result Size
1	32 × 32	32
1	32 × 32	64
1	32 × 32	80
4	16 × 16	32
4	16 × 16	16
1	16 × 16	40
4	16 × 16	40

TABLE 7.5-2. ADSP-TS0xx fixed-point multiply-accumulate operations. Number of MACs means (possibly SIMD) MACs in one instruction in one data path.

cations, a complex multiplication with precision greater than 16 bits is required. There is no support for mixed-precision operations using this complex multiplication instruction.

The multiplier-accumulator provides the following status flags: fixed-point zero/floating-point underflow, negative, overflow, and floating-point invalid. There are additional "sticky" bits for fixed-point overflow, floating-point underflow, floating-point overflow, and floating-point invalid. As in the ALU, for SIMD instructions, in general a status bit is computed as the logical *or* of the status bits for individual results. For example, if one of four 16-bit SIMD multiplication products is zero, then the "multiplier zero" flag will be set. Similarly, if one of the four products is negative, then the "multiplier negative" status flag is set.

Memory System

The ADSP-TS0xx memory system consists of on-chip 32-bit memory, up to approximately 4 Gwords of general-purpose off-chip 32-bit memory, and up to 3.5 Mwords of off-chip multiprocessor memory (i.e., the 32-bit internal memory of up to seven other ADSP-TS0xx chips). Memory is arranged in a unified, 32-bit word-addressable space that contains both instructions and data.

The ADSP-TS0xx architecture supports three blocks of on-chip memory of up to 512K 32-bit words each; the ADSP-TS001 contains 3 blocks of 64K 32-bit words each. On-chip memory always resides at the lowest addresses in the address space.

The ADSP-TS001 contains significantly more on-chip memory than most commercial DSP processors.

The 32-bit data path register files for both data paths, as well as the address generator register files, discussed below, are mapped into the internal memory space.

Each of the three 128-bit internal data and instruction buses is connected to one of the on-chip memories. Up to 128 bits may be fetched in each instruction cycle from each of the three on-chip memories. Thus, up to four 32-bit instructions may be fetched from one memory while two 128-bit operands are being fetched from the other two on-chip memories. The program control unit drives one bus while two separate address generator units drive the other two buses. When executing instructions from on-chip memory, the maximum data bandwidth for the ADSP-TS0xx operating at 120 MHz is 1.92 billion 16-bit words/second (sixteen 16-bit words/cycle).

The PASS instruction can be used to transfer data between registers within the same execution unit. Unless the PASS instruction is used, register-to-register moves (even between registers in the same register file) must use one of the three internal data bus accesses available for each instruction packet.

In tasks with extensive memory accesses and extensive operations on address registers, register-to-register transfers will compete *with loads and stores for memory bus accesses; this may reduce throughput.*

The ADSP-TS0xx has an unusually high per-cycle data memory bandwidth. In one cycle, up to 256 bits of data can be fetched; for example, sixteen 16-bit data words or eight 32-bit data words to access this full bandwidth. However, data words must be arranged in memory so that each memory access reads or writes a contiguous block of 128 bits.

As described earlier, each instruction packet can include up to four 32-bit instructions. The instruction packets do not have to be aligned on 128-bit memory boundaries. Instead, there is a 5-entry, 128-bit-wide FIFO (called an "instruction alignment buffer" by Analog Devices) which contains up to twenty 32-bit instructions; up to four entries can be dispatched from the FIFO in each cycle. Unlike earlier SHARC chips, the ADSP-TS0xx does not contain an on-chip instruction cache; however it does contain a "Branch Target Buffer" for mitigating branch penalties. Branches (e.g., jumps and calls) generally incur penalties of three or six stall cycles (see the *Pipeline* section for details). To avoid branch stalls, the programmer can specify whether or not a branch should be predicted, using the assembly-language "NP" construct. The branch target buffer (BTB), a 128-entry 4-way set-associative cache, stores the most recently used predicted destination addresses. When a jump address is currently stored in the BTB and correctly predicted, the ADSP-TS0xx inserts the corresponding destination address as the fetch address for the following instruction packet, reducing the branch to a single-cycle operation. The branch target buffer only supports addresses in internal memory. Only one branch may be predicted per aligned quad word of program memory.

If complicated if...then...else *code is repeatedly executed, the branch target buffer can significantly improve execution times. The limitation that the branch target buffer only works with on-chip memory addresses is a disadvantage, however, and is one of several good reasons to execute from on-chip memory whenever possible.*

External Memory Interface

The external 32-bit memory space is divided into six regions:

- Multiprocessor memory
- Three external memory banks of up to 67 million 32-bit words each. One of these memory banks supports a glueless interface to SDRAM
- "Host memory" (approximately 4 Gwords)
- Boot EPROM

The host memory space is nominally intended for memory that is shared with a host processor, but can be used for other kinds of memory.

The processor has six memory select pins and asserts one of the pins during external memory accesses to indicate which part of external memory is being accessed. These pins can be used to drive the chip select inputs of external devices, eliminating the need for external address decoding circuitry in many applications. In addition, wait state configuration is independently specified for three memory banks (external host memory and two external memory banks). The processor supports externally requested wait states, and zero to three programmed wait states.

The external address bus is 32 bits wide, which allows the processor to access approximately 4 Gwords of multiprocessor or external memory. The external data bus width is configured to 32 bit or 64 bits separately for host memory, multiprocessor transactions, and other external memory. The ADSP-TS0xx can access 128-bit data and/or instruction packets stored in external memory, but requires at least two memory accesses to do so.

The external memory interface uses the system clock, SCLK, as discussed below in the *Clocking* section. The maximum system clock rate is one half of the master clock (CCLK) rate. The external memory interface can perform accesses at a maximum rate of one 64-bit word per system clock cycle, compared with one 128-bit word per master clock cycle for on-chip memory. Thus, when executing instructions from external memory, the peak instruction fetch rate is at most one-fourth of the peak instruction fetch rate from on-chip memory, and may be slower depending on bus contention or wait states.

> *Like many high-performance DSP processors, the ADSP-TS0xx must execute instructions from on-chip memory to achieve peak performance.*

Up to eight ADSP-TS0xx processors, a host processor, and external memory can be connected (in a glueless fashion) using the shared bus scheme shown in Figure 7.5-2. In

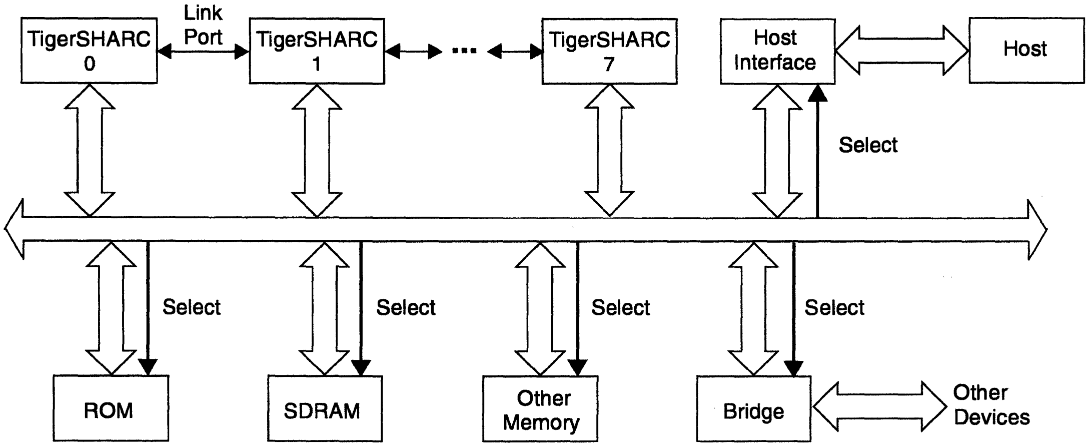


FIGURE 7.5-2. ADSP-TS0xx multiprocessing configuration. The link ports may connect TigerSHARCs 0 through 7 in any configuration, not just the simple configuration shown here.

such a configuration, one ADSP-TS0xx can access the on-chip memory of any other ADSP-TS0xx. Such multiprocessor accesses use one of the three internal memory buses of the accessed ADSP-TS0xx. If no internal bus becomes available for four cycles, one of the three internal buses is forced to service the multiprocessor access. In such a case the instruction expecting to use the commandeered internal bus is delayed. Multiprocessor accesses do not disrupt operations that use only the data path register files and/or address generation unit register files. A processor may also perform a broadcast write to all other processors in one instruction by writing to a reserved write-only section of the internal memory space.

Multiple ADSP-TS0xx processors can share an external bus without additional arbitration circuitry, because an arbitration mechanism is provided in the ADSP-TS0xx that allows a host processor or another ADSP-TS0xx in a multiprocessor system to gain control of the external memory bus. A fixed-priority arbitration scheme is provided, whereby the hardware designer assigns each ADSP-TS0xx a different priority. When a processor requests control of the bus, it waits its turn until all other ADSP-TS0xx processors with a higher priority have finished their bus accesses. A state machine running in lock step in each processor keeps track of the round-robin priority level. The programmer may specify the maximum number of cycles for which one processor may keep the bus. There is a mechanism for a processor with a lower priority to temporarily gain access to the bus; for example, for DMA transfers. The ADSP-TS0xx has conditional instructions that depend upon whether an ADSP-TS0xx is the current bus master in a multiprocessor system. These instructions can be used to perform an atomic read-modify-write operation, for example.

The ADSP-TS0xx has special support for SDRAM, including refresh generation. The ADSP-TS001 offers glueless connection to 16 Mbyte, 64 Mbyte, 128 Mbyte, and 256 Mbyte SDRAMs. SDRAM is divided into two banks. The maximum data rate is one 32-bit word per SCLK cycle. On a 120 MHz ADSP-TS001, assuming that SCLK is set to 1/2 the rate of CCLK, the maximum data rate is thus 60 million 32-bit words per second.

An optional external EPROM, primarily intended for booting, is not part of the main external memory space even though the ROM is attached to the external memory bus. The ROM is limited to a maximum of 24 address bits (16 Mbytes) and 8 data bits. The boot EPROM space cannot be directly accessed by programs, but rather is used for booting and can be accessed via DMA. In addition to EPROM devices, the EPROM address space may also be used for flash memory. The ROM select pins of several processors may be tied together, and more than one ADSP-TS0xx may access or boot from the same ROM.

The ADSP-TS0xx has an on-chip DMA controller that allows transfers between internal memory and external memory, other ADSP-TS0xx processors' internal memory, one of the link ports, and/or between external memory and external peripherals (see the *Peripherals* section). The on-chip DMA controller conducts DMA transfers involving the internal buses by using idle internal bus cycles when available. As happens with multiprocessor transactions discussed previously, if no internal bus becomes available for four cycles, one of the three internal buses is forced to service DMA.

> *The ADSP-TS0xx provides unusually flexible support for external memory.*

Address Generation Units (AGUs)

There are two address generation units (J and K). Each address generation unit contains a memory access unit along with a fixed-point data path.

> *The concepts of DAG, PM, and DM found in earlier Analog Devices processors are not present in the ADSP-TS0xx. This difference represents a significant departure from earlier Analog Devices architectures.*

Each AGU data path contains an ALU, a shifter, and a memory-mapped register file of thirty-two 32-bit registers. Each data path in the address generation unit supports 32-bit fixed-point integer operations, and may execute one instruction per instruction cycle. Recalling that the ADSP-TS0xx executes as many as four instructions per instruction packet, as many as two of those instructions may use the address generation units.

The shifter and ALU of a given address generation unit data path both access the AGU's register file of thirty-two 32-bit registers. Using a generalized register move instruction, the contents of a register from one AGU may be copied into a register from the other AGU. One of the registers in each file is dedicated as a status register for its address generation unit data path. Inputs to all arithmetic operations come from the register file, and results of all arithmetic operations are delivered to the register file. There are no shadow registers in the address generation units.

> *The lack of shadow registers in the ADSP-TS0xx is a disadvantage compared to other processors from Analog Devices. This is offset somewhat by the processor's support for saving or restoring the address generator status register (register 31) and three other registers (28-30) to/from internal memory as a quad word memory operation in one instruction.*

The AGU ALUs support typical address-generation computations, including add, subtract, increment, and decrement. Many operations involve a second input register or an immediate value. The AGU data paths also support a variety of general-purpose arithmetic operations, including add and subtract with carry, absolute value, average ([A+B]/2, [A-B]/2), minimum, maximum, and compare (signed and unsigned). Furthermore, the ALUs support logical *and, or, xor, not,* and *and not*. The AGU shifters perform rotation by one bit left or right and arithmetic or logical right shifts by one bit. One-bit arithmetic left shifts are supported indirectly via adding a register's content to itself. The shifters do not make use of the carry bit.

Each AGU data path provides the following status flags, set by either the ALU or shifter: zero, negative, overflow, and carry.

More information on the address generation unit data path is given in the *Parallel Move Support* section.

> *The ADSP-TS0xx address generation units have impressive capabilities (arithmetic, shifting, logic) far beyond the address generation units of most older DSP processors. In many situations, each address generation unit data path can be used as an additional fixed-point data path in its own right.*

The ADSP-TS0xx supports register-direct and several register-indirect addressing modes, discussed below. Most arithmetic operations use register-direct addressing exclusively. Register-indirect addressing is used for moving data between registers and memory. Memory-direct addressing is not supported. A 5-bit signed immediate value may be loaded into any of the registers in the address generator units, into the fixed/floating-point data paths, and into many other registers as well (such as the DMA or link control registers). The ADSP-TS0xx supports long immediate data via a 32-bit "instruction extension word"; only one 32-bit instruction extension may be included in one instruction packet.

Each address generator contains thirty-one registers that may be used as base address registers or modifier registers. Values stored in the modifier registers are used to post-increment or index from base address register contents. Within each address generator, any modifier register can be used to modify the address held in any other address register. Immediate post-increment values from -128 to 127 are available; with an instruction extension word, an immediate post-increment value of up to 32 bits may be specified.

Indexed addressing is also supported (confusingly, Analog Devices calls it "pre-modify no update"). With indexed addressing, the effective address is formed by adding the value in a modifier register to the value in a base address register; the value in the base address register is not changed.

> *The fact that any of the AGU registers can be used to pre- or post-increment any of the other registers in an address generator provides flexibility for the programmer and compiler.*

The ADSP-TS0xx supports circular addressing with no alignment constraints. Address Registers 0-3 in each address generator are each associated with a companion base address register and buffer length register. Thus, up to eight simultaneous circular buffers can be maintained.

> *Many other Analog Devices processors support an interrupt when an address register reaches the end of a programmer-defined circular buffer. Such an interrupt is not available on the ADSP-TS0xx.*

Bit-reversed addressing is supported in address registers 0-3 in each of the address generators.

The program may load a jump address into a special CJMP register. An address in CJMP may be used for indirect jumps and indirect subroutine calls. This is useful for reconfigurable interrupt service routines and jump tables.

SIMD-style data transfers allow a single instruction to perform two data transfers: one between memory and the X fixed/floating-point data path, and one between memory and the Y data path. As illustrated in Figure 7.5-3, data is transferred between the X data path and one of the three on-chip memory banks, and between the Y data path and a different on-chip memory bank. Each transfer can be 32, 64, or 128 bits wide. The programmer specifies a SIMD-style data transfer by accessing special regions in the on-chip memory space. The special memory regions are listed in Table 7.5-3. For example, if the type of SIMD transfer illustrated by the left-most diagram in Figure 7.5-3 is desired, an address in the region 0x00200000-0x0020FFF is used. Only one of the four instruction slots in an instruction packet is needed for such a SIMD transfer, and either the J or K address generator may provide the address.

A 32-, 64-, or 128-bit data word may be transferred from memory simultaneously into both of the fixed/floating-point data path register files in one instruction (Analog

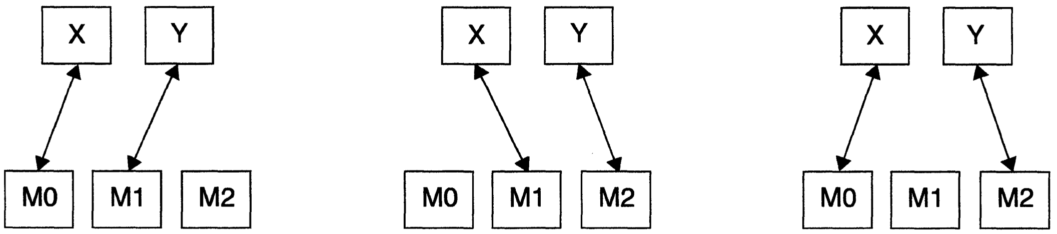


FIGURE 7.5-3. SIMD transfers between fixed/floating-point data path registers and on-chip memory. M0, M1, and M2 are the three internal memory banks shown in Figure 7.5-1.

Address Range in Internal Memory Space	Function
0x00300000 - 0x0030FFFF	SIMD Internal Memory 0 & 2
0x00280000 - 0x0028FFFF	SIMD Internal Memory 1 & 2
0x00020000 - 0x0020FFFF	SIMD Internal Memory 0 & 1
0x00100000 - 0x0010FFFF	Internal Memory 2
0x00080000 - 0x0008FFFF	Internal Memory 1
0x00000000 - 0x0000FFFF	Internal Memory 0

TABLE 7.5-3. Part of ADSP-TS0xx memory map. Combinations of internal memory banks 0, 1, and 2 are memory-mapped in internal memory space for SIMD access (see Figure 7.5-3).

Devices calls this a "broadcast"), filling one, two, or four registers respectively. In addition, a single 64- or 128-bit data word may be split during a transfer between memory and the fixed/floating-point data path registers in the manner shown in Figure 7.5-4 (Analog Devices calls this "merged").

64- and 128-bit data is transferred to and from memory in an aligned fashion. The ADSP-TS0xx, however, indirectly supports unaligned memory reads (not writes) via a special buffer, called the "Data Alignment Buffer," or DAB. The DAB is used for unaligned 64- and 128-bit reads from memory into the register files of the fixed/floating-point data paths. The DAB has two entries of 128 bits, enabling it to hold as many as eight 32-bit words. Data loaded from memory as aligned 64- and 128-bit chunks is temporarily stored in this buffer. The desired subset of unaligned 32-bit words can then be read from the DAB, up to four at a time. (There is also an option for reading unaligned 16-bit words, up to eight at a time). Two read operations are required to fill the DAB. Thus, there is one cycle of delay before unaligned data are available. Thereafter, successive reads can be completed at the rate of one every instruction cycle if the successive reads maintain the same alignment as the initial read and access contiguous 64- or 128-bit chunks of memory.

> *The ADSP-TS0xx's support for unaligned reads is superior to that of some processors, but the limitations of the DAB (no support for writes, cycle penalties) complicate programming and will impact performance in some cases.*

Pipeline

The ADSP-TS0xx uses two sequential instruction pipelines: a three-stage "fetch" pipeline and a five-stage "instruction" pipeline. There is a buffer between the two pipelines.

The fetch unit can transfer four 32-bit instruction words in one cycle from internal memory. As mentioned above, for external memory, the peak instruction fetch rate is 1/4 of the peak internal memory fetch rate and may be slower depending on bus contention or wait states.

The inter-pipeline buffer, called the "Instruction Alignment Buffer" (IAB), is a five-deep, 128-bit-wide FIFO. Whenever a complete instruction packet (consisting of one to four 32-bit words) is available, it may be read from this buffer for execution. Since the decode/execute pipeline may process fewer than four 32-bit instructions per cycle, and since the fetch unit fills the buffer as quickly as possible, the buffer may remain nearly full much of the time.

> *In some processors, a full four-word instruction packet would need to be aligned on a quad word boundary. In the ADSP-TS0xx, the IAB allows one to four instructions to be aligned on any 32-bit boundary. This helps reduce program memory requirements.*

The stages of the instruction pipeline are shown in Table 7.5-4. The pipeline is interlocked and is invisible to the user except for a few cases, the most important of which are:

- A bus conflict. For example, two instructions in the same packet may attempt to access the same data bus. In this case the hardware inserts a stall in the Integer stage until the bus is available.
- A fixed/floating-point data path instruction uses an operand written by an instruction in the previous packet, in which case the hardware inserts a single-cycle stall. For multiply-accumulate instructions, this stall is avoided through the use of special bypassing hardware.
- The address generation unit uses an operand fetched from memory by an instruction in the previous packet, in which case the hardware stalls for four instruction cycles.
- Branch, call, and return instructions take up to six instruction cycles to execute. If a branch is correctly predicted and already resident in the branch target buffer, the branch requires only one cycle.

During a stall in instruction execution, the fetch pipeline proceeds uninterrupted.

Although programmers and compiler users must be aware of the ADSP-TS0xx pipeline to create optimal code, the impact of the pipeline is minor.

All pipeline conflicts are handled automatically in the ADSP-TS0xx hardware. Compared with some other processors with non-interlocked pipelines, this hardware simplifies compiler design.

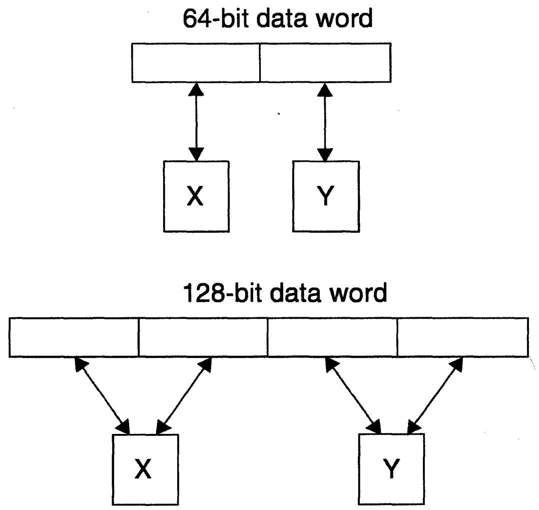


FIGURE 7.5-4. Single data transfers split between X and Y data paths.

Instruction Set

The ADSP-TS0xx uses a 32-bit instruction word. The registers and instruction set for the ADSP-TS0xx fixed/floating-point data paths are summarized in Table 7.5-5, Table 7.5-6, and Table 7.5-7.

Note that most of the instructions that operate on 8-bit ("Byte") and 16-bit ("Short") data are SIMD operations. In a typical case, two inputs each containing eight 8-bit or four 16-bit operations are handled in a single data path, producing either eight 8-bit or four 16-bit results. Many 32-bit operations are also available in SIMD versions, where two pairs of 32-operands are handled simultaneously in one data path to produce two 32-bit results. In the sideways sum operation, four 8-bit quantities in one 32-bit register (or four 16-bit quantities in two adjacent 32-bit registers) are added (or accumulated) by one data path to produce a 32-bit result.

All fixed-point multiply operations can operate on fractional or integer, signed or unsigned values, and can optionally saturate, truncate, and/or round the result. There is a floating-point multiply, but there is no floating-point multiply-accumulate instruction.

Stage Name	Also Known As	Address Generation Unit (J, K) Activity	Fixed/Floating-Point Data Path (X, Y) Activity
Decode	D	Distribute one or more instructions to address generation units and/or fixed/floating-point data paths. Identify branch instruction if present.	
		Address generation unit fetches operands (addresses) from register file. Execute any operations on operands (calculate address). Write flags.	Send instruction to ALU, MAC, or shifter.
Integer	I	If memory access, decide which bus to use, and request bus.	ALU, shifter, and/or MAC operation decode.
Access	A	If memory access and bus available, bus is granted to address generator unit.	Fetch operands from register file.
Execute 1	EX1	Memory access in progress.	Execute.
Execute 2	EX2	Updated address written to register file. Data from memory access available in register.	Write results to registers. Write flags.

TABLE 7.5-4. Instruction pipeline stages of the ADSP-TS0xx. Instruction fetch stages are not shown.

There is no divide iteration instruction; however, the ADSP-TS0xx provides a reciprocal seed instruction. Shift and rotate instructions shift the input operand from 0 to 64 bits, without involving the carry bit.

The ADSP-TS0xx provides a rich instruction set.

For the Viterbi algorithm, the ADSP-TS0xx is capable of two add-compare-select operations on average per instruction cycle. This is a significant improvement over previous processors from Analog Devices.

For the address generation units, the registers and instruction set are summarized in Table 7.5-8 and Table 7.5-9.

Assembly Language Format

The assembly language of the ADSP-TS0xx bears little resemblance to the assembly languages of other Analog Devices processors. The ADSP-TS0xx uses an algebraic assembly language format in which instructions typically contain one field: a memory move with optional address generation register update, a computation, or a branch. The computation instructions specify the operation(s) to be carried out by the ALU, multiplier-accumulator, or shifter in one or both of the fixed/floating-point data paths or one or both of the address generators. The data move instructions specify a register-to-register data move or a register-memory data move. For example, consider the instruction:

```
YR15 = R15 + R6;;
```

(where ";;" indicates the end of an instruction packet). The 32-bit integer quantity in YR6 is added to the previous value of YR15, and the result is written back to YR15. This calculation occurs in the Y fixed/floating-point data path, specified by the "Y" in "YR15".

As mentioned earlier, an execution packet on the ADSP-TS0xx consists of up to four separate instructions. All instructions inside the execution packet are executed in the same instruction cycle. For example, the execution packet:

```
R29:28=CB l[J1+=J10]; XFR4=R3*R30; YFR4=R1*R30; FR6=R6+R4;;
```

loads the 64-bit quantity addressed by address generation unit register J1 into registers 28 and 29 in both the X and Y fixed/floating-point data paths. (Both data paths are used since

Register	Width	Purpose
XR0-XR31	32 bits	ALU/Shifter/Multiply-accumulate input and output
XSTAT	32 bits	Status
XPR0, XPR1	32 bits	Sideways sum accumulation
XMR0-XMR4	32 bits	Accumulators for multiply-accumulate, including guard bits

TABLE 7.5-5. ADSP-TS0xx fixed/floating-point data path register summary. The registers shown here for the X data path are duplicated in the Y data path.

Description	Byte	Short	Normal	Long	Float
ALU					
Add, subtract; add and subtract	4, 8	2, 4	1, 2	1	1
Add with carry or subtract with borrow			1	1	
Absolute value of sum or difference	4, 8	2, 4	1, 2	1	1
Accumulate absolute value of sum or difference	8	4			
Average	4, 8	2, 4	1, 2	1	1
Sideways sum, sideways accumulate	4, 8	2, 4			
Increment or decrement	4, 8	2, 4	1, 2	1	
Absolute value	4, 8	2, 4	1, 2	1	1
Negate	4, 8	2, 4	1, 2	1	1
Compare	4, 8	2, 4	1, 2	1	1
Minimum or maximum	4, 8	2, 4	1, 2	1	1
Clip (saturate)	4, 8	2, 4	1, 2	1	1
Logical (*and, or, xor, not, not-and*)			1	1	
Expand (convert to higher precision)	4, 8	2, 4			1
Expand sum or difference	4, 8	2, 4			
Compact (convert to lower precision)		4	2		1
Compact sum or difference		4	2		
Merge	4, 8	2, 4			
Count ones			1	1	
Viterbi maximum, minimum (VMAX)	8	4			
Set status (PASS)			1	1	1
Copy sign					1
Scale					1

TABLE 7.5-6. Summary of ADSP-TS0xx instructions for the fixed-/floating-point data paths. Byte = 8-bit, Short = 16-bit, Normal = 32-bit, Long = 64-bit, Float = 32-bit or 40-bit. The numbers indicate the number of results produced by the instruction when executed on a single data path using the indicated data type. Parallelism can be further increased by executing the same instruction on both data paths in a SIMD fashion or by executing different instructions in parallel on the two data paths in a VLIW fashion. All instructions can be conditionally executed. (Continued)

there is neither "X" nor "Y" before "R29.") The value in address generation unit register J10 is added to J1 and the result stored in J1 after any circular buffer ("CB") corrections. In the fixed/floating-point data path X, the 32-bit floating-point quantities in registers XR3 and XR30 are multiplied, with the result written into XR4. In the Y fixed/floating-point data path, register YR4 is loaded with the floating-point product of YR1 × YR30. Finally, in both X and Y fixed/floating-point data paths, the floating-point quantity in register R6 is incremented by the register R4, and the results are written to register R6.

The size of data to be moved or manipulated is indicated by the number of registers named and the data type designator. A single register (such as XR3) means a 32-bit

Description	Byte	Short	Normal	Long	Float
Conversion					
Fixed-point to floating-point with optional scaling			1	1	
Floating-point to fixed-point with optional scaling					1
Extract mantissa or exponent					1
Shifting and Bit Manipulation					
Logical or arithmetic shift left/right by 1-64 bits	4, 8	2, 4	1, 2	1	
Rotate left/right by 1-64 bits			1, 2	1	
Field deposit or extract			1	1	
Apply mask			1	1	
Bit set, clear, toggle, or test			1	1	
Exponent detection (count leading zeros or ones)			1	1	
Multiply-Accumulate					
Multiply		4	1		1
Multiply-accumulate		4	1		
Complex multiply-accumulate		1			
Reciprocal seed, reciprocal square-root seed					1

TABLE 7.5-7. Summary of ADSP-TS0xx instructions for the fixed-/floating-point data paths. Byte = 8-bit, Short = 16-bit, Normal = 32-bit, Long = 64-bit, Float = 32-bit or 40-bit. The numbers indicate the number of results produced by the instruction when executed on a single data path using the indicated data type. Parallelism can be further increased by executing the same instruction on both data paths in a SIMD fashion, or by executing different instructions in parallel on the two data paths in a VLIW fashion. All instructions can be conditionally executed. (Continued from previous table)

quantity; a register pair like XR29:28 means a 64-bit quantity; and four registers (such as XR3:0) means 128 bits. The data type is indicated by a prefix: L for 64-bit fixed-point; S for 16-bit fixed-point; B for 8-bit fixed-point; F for 32-bit floating-point. The combination of F and a register pair, such as FR29:28, indicates 40-bit floating-point. The lack of any of these designators indicates a 32-bit fixed-point data type. The data type and operand size may occur in combination, so that the instruction

```
XSR1:0 = R31:30 + R25:24;;
```

specifies that the four 16-bit quantities ("S") in registers XR31:XR30 in the X data path are added to the four 16-bit quantities in registers 25-24 in SIMD fashion, with the results going to registers 1-0.

Parallel Move Support

Parallel moves in the ADSP-TS0xx are independent of computation operations and are allowed with all computation operations. The ADSP-TS0xx can perform up to two reads, two writes, or one read and one write in parallel within an instruction packet. Each move requires one of the two address generation units, and may transfer up to 128 bits.

There is one special case for register-to-register parallel moves. Recall that in the multiplier-accumulator in each fixed/floating-point data path, there are special registers MR0-MR3 for accumulation. In a single instruction such as

```
R7:6 = MR1:0, MR1:0 += R3 * R9 (C);;
```

the current contents of the accumulator registers MR1-MR0 are written to the registers R7-R6 in both fixed/floating-point data paths as a parallel move with the MAC operation. Registers MR1-MR0 are cleared in both data paths, as specified by "(C)". Then the 64-bit product of the contents of R3 and R9 is summed into MR1:0, again in both fixed/floating-point data paths.

The ADSP-TS0xx provides good flexibility in the use of parallel data moves.

Register	Width	Purpose
J0-J30	32 bits	General address/modifier registers
JSTAT	32 bits	Status register (also referred to as J31)
JB0-JB3	32 bits	Modulo base registers
JL0-JL3	32 bits	Modulo length registers

TABLE 7.5-8. ADSP-TS0xx address generation unit register summary. The registers shown here for the J address generator are duplicated in the K address generator.

Orthogonality

The ADSP-TS0xx instruction set is moderately orthogonal by DSP processor standards. This is made possible by its 32-bit instruction word size. For example, all instructions can be conditionally executed, moves are usually handled as separate instructions in an execution packet, and most arithmetic instructions operate on the same set of registers. Also, most arithmetic instructions in the fixed/floating-point data paths support both fixed-point and floating-point formats. As illustrated in Tables 7.5-6 and 7.5-7, however, many instructions only support a subset of the data types supported by the processor, detracting from orthogonality.

> *In spite of its orthogonality, the ADSP-TS0xx is difficult to program because of the complexity of the architecture, the large number of computational units, and the multiple levels of SIMD operation discussed above.*

Execution Times

Nearly all ADSP-TS0xx instructions have single-cycle throughput in the absence of wait states and external memory access conflicts (which may occur, for example, when both instructions and data are located off-chip). Most fixed/floating-point data path instructions have a latency of two instruction cycles, most address generation unit instructions have a latency of one instruction cycle.

Branches, calls, and returns to destinations in internal program memory that are correctly predicted and already in the branch target buffer execute with single-cycle

Class	Instructions
Fixed-Point Arithmetic	Absolute value, add, add with carry, increment, decrement, subtract, subtract with borrow, average two values, minimum, maximum
Logic	*and, or, exclusive-or, not, and-not*
Shifting	Arithmetic/logical shift right by 1 bit
Rotation	Rotate left/right by 1 bit
Comparison	Fixed-point compare
Looping	No direct hardware support except for two loop counter registers; see *Hardware Looping*
Branching	Conditional and unconditional relative or absolute branch
Subroutine Call	Conditional and unconditional call

TABLE 7.5-9. ADSP-TS0xx address generation unit instruction summary. All instructions can be executed conditionally.

latency. Other branches, calls, and returns have latencies from two to six cycles. There are no delayed branches.

Instruction Set Highlights

Noteworthy features of the ADSP-TS0xx instruction set include:

- 16-bit complex multiply
- Multiply with parallel dual add/subtract, useful for computing FFT and DCT kernels
- Bit-field insertion and extraction, with facilities for variable field lengths in a bit stream
- Any combination of two parallel register-memory moves (two loads, one load and/or store, or two stores), up to 128 bits each, in one execution packet
- Conditional execution of all instructions
- SIMD instructions for 8-, 16-, and 32-bit data
- Extensive arithmetic, logic, and shifting features in the address generation units

Execution Control

Clocking

No on-chip oscillator is provided for clock signal generation. The ADSP-TS0xx requires an externally generated clock (LCLK); the master clock (CCLK) is generated from LCLK by an on-chip multiplier that multiplies the frequency of LCLK by a selectable factor between 2 and 5. Thus, a 120 MHz ADSP-TS001 can operate from an LCLK input ranging from 24 to 60 MHz. A separate, externally generated "system" clock (SCLK), which is typically the same as LCLK, drives the external memory interface. SCLK is limited to a maximum of one-half the frequency of the master clock; e.g., 60 MHz for a 120 MHz ADSP-TS001. Hence, the maximum cycle rate of the external bus is one-half the maximum instruction cycle rate.

Condition Codes

Conditional execution of an instruction is based on an arithmetic condition generated by one of the fixed/floating-point data paths or one of the address generation units, the value of a bit I/O pin, the value of either of the special-purpose loop counters discussed under *Hardware Looping*, or loss of mastership of the external memory interface. In addition there are two special-purpose status registers in which the programmer may store one or more condition codes. New values may be loaded into either of these registers, or combined with the contents of the registers via a logical *and*, *or*, or *exclusive-or* operation. Conditional execution may be based on the bits in either of these special-purpose registers.

Hardware Looping

There is almost no hardware support for looping on the ADSP-TS0xx. There is no DO instruction, and no stack on which to store loop variables.

There are two special-purpose loop counters that allow for nesting two loops. These counters can be decremented and tested in one instruction, allowing for low-overhead looping. Consider the instruction sequence:

```
            LC0 = 16;;
    _loop:
            ...
            if nLC0E, jump _loop;
```

in which loop counter 0 is initialized to 16. The "if" statement decrements LC0, tests the result, and jumps to "_loop" as long as LC0 > 0. The jump is predicted to be true unless the programmer indicates otherwise, so the loop overhead is limited to executing one jump instruction on each pass through the loop except the last, plus the overhead to initialize LC0. (In some cases, other instructions can be executed in parallel with the jump, reducing the loop overhead). The loop exit condition can be any condition mentioned under *Condition Codes.*

The maximum number of repetitions is 2^{32} when using the special-purpose loop counters. A sequence of instructions of any length can be contained in a loop. Loops can be interrupted.

> *The ADSP-TS0xx provides unusually meager hardware looping features. The programmer must carefully trap for the possibility that an initial loop count may be 0, since setting a special-purpose loop counter to zero in a code sequence like the one above results in 2^{32} iterations. Many programmers will implement a stack to save and restore the loop counter values.*

Interrupts

The ADSP-TS001 has 29 interrupt sources. Interrupts are individually prioritized (each has a separate but fixed priority), individually maskable, and optionally nestable.

External interrupts include four interrupt pins and an interrupt vector pin. When the vector interrupt is asserted, the processor executes a routine whose address is given in a dedicated memory-mapped register that may be written by a host processor and/or another DSP. Among other uses, this mechanism can be used by a host processor to force reset.

Internal hardware interrupt sources include timer expiration, link port service request, DMA activity, arithmetic exceptions (fixed-point overflow; floating-point overflow, underflow, and invalid operand), illegal operations (such as attempting to read the broadcast memory space) and emulation. There is also a special bus lock interrupt which can be triggered when the processor acquires the bus for an atomic read-modify-write operation.

A user software interrupt loads a special-purpose register with a 5-bit trap code.

Each timer has two interrupt locations, both corresponding to timer expiration, but having different priorities. By masking the high-priority timer interrupt, the programmer can lower the priority of the timer interrupt.

The ADSP-TS0xx responds to an interrupt by branching to the first instruction at the corresponding interrupt vector location, which may be in internal or external memory. The return program counter is stored into a special-purpose register, which must be saved by the programmer if interrupts are nested (there is no hardware stack for interrupt return addresses). Latency from the time when an interrupt is asserted as pending to completion of execution of the first interrupt vector instruction is eight or nine instruction cycles, depending on whether the first instruction in the interrupt service routine is quad-aligned and assuming the processor is in an interruptible state. To return from an interrupt, the programmer places the return address into a special-purpose register and uses the RTI instruction.

The ADSP-TS0xx has the ability to clear, force, and test for the presence of interrupts.

> *The ADSP-TS0xx has flexible interrupt handling mechanisms. The multiprocessor vector interrupt is useful for interprocessor commands in a system with multiple ADSP-TS0xx processors.*

Stack

The ADSP-TS0xx does not provide a hardware stack. A software stack can be implemented using any address generator register as a stack pointer. Push and pop operations can be implemented with load and store instructions with post-increment/decrement. Since there is no pre-increment-store instruction for the address generators, the stack pointer must be decremented before starting to pop items from the stack. Recall that four registers may be stored to internal memory in one instruction by either fixed/floating-point data path or either AGU. In particular, this applies to the address generator status register (register 31) and three other registers (28-30), which may be saved to or restored from internal memory as a quad memory operation in one instruction by either AGU. Storing the AGU status register with three other registers simplifies context store for interrupt handling, for example.

Bootstrap Loading

On reset, the ADSP-TS0xx can begin execution at a fixed location in memory. Alternatively, it can bootstrap load over a link port, or through a byte-wide ROM attached to the external memory interface (see *External Memory Interface*).

Peripherals

The ADSP-TS0xx on-chip peripherals include two timers (with a 64-bit count for each), four "link ports" (each 8 bits wide instead of 4 bits as on the ADSP-2106x), a 14-channel DMA controller, and four bit-I/O pins. The external memory interface also functions as a host port.

- **Timers**

  The ADSP-TS0xx features two 64-bit interval timers. The timers generate interrupts when their counters reach zero. Timer 0 also pulses the TIMEXP output pin when its counter reaches zero. Each timer is clocked with the master chip clock with no prescaler. A timer interrupt can be assigned as a low- or high-priority interrupt.

  Because the ADSP-TS0xx timers use 64-bit counters, the lack of a prescaler is not a drawback.

- **Link Ports**

  The link ports are 8-bit bidirectional parallel communications ports that can be connected to peripheral devices or to the link ports on other ADSP-TS0xx processors. Each link port consists of eight bidirectional data lines, a bidirectional clock line that doubles as a bidirectional acknowledge line, a frame synchronization line, and a pin to indicate whether the link is transmitting or receiving. The direction of data transfer for each link port is specified by the link port control register. Data is automatically packed or unpacked into 128-bit quad words. Single transfers can be interrupt-serviced by the processor core, and block transfers can be serviced by the DMA controller without intervention by the processor core. Each link port is internally double-buffered in both the transmit and receive directions. Each link port can sustain a data rate of up to 120 Mbytes/second.

- **DMA**

  The DMA controller on the ADSP-TS0xx supports 14 channels of DMA. Subsets of these channels can be used to perform data transfers (individual or block) between on-chip memory and the link ports or the external memory space. Additionally, subsets of these channels can be used to transfer data between external memory and external peripherals (referred to as "extern" on the ADSP-21xxx and "fly-by" on the ADSP-TS0xx), from one link port to another, or between internal memory on two ADSP-TS0xx processors. During boot, the DMA controller may load a program from external ROM or through a link port.

  The DMA controller allows the processor or external peripherals to specify block-data transfer operations and then return to normal operation while the DMA controller carries out those transfers independently. DMA transfers involving internal memory are executed using one of the three internal data buses of the ADSP-TS0xx. DMA transfers between external peripheral devices do not use on-chip resources except for the DMA controller.

For DMA transfers involving external devices other than memory, there are four DMA request pins and a special-purpose buffer for accumulating up to 15 DMA requests. An interrupt may be generated when a transfer is finished.

A single transfer may use data widths of 32, 64, or 128 bits. Each ADSP-TS0xx DMA channel utilizes a data transfer buffer in on-chip memory, the size and location of which are specified by the programmer. The parameters for DMA transfers are called a "transfer control block" (TCB) and are read from 32-bit memory-mapped registers, a set of four for each DMA channel. Since the registers are memory-mapped, they may be programmed by external processors.

Four DMA channels are dedicated to external devices, eight to link ports, and two to "Auto-DMA," as shown in Table 7.5-10. With Auto-DMA, an external device may program the DMA channel to transfer data to internal memory. The DMA channels have a priority schedule which the DMA controller uses to determine which channel can drive the bus on each cycle.

DMA Channel Number and Priority (13 = Highest)	Associated Resources
0	External port ("fly-by"), DMA request pin 0
1	External port, DMA request pin 1
2	External port, DMA request pin 2
3	External port, DMA request pin 3
4	Link port 0 transmit
5	Link port 1 transmit
6	Link port 2 transmit
7	Link port 3 transmit
8	Link port 0 receive
9	Link port 1 receive
10	Link port 2 receive
11	Link port 3 receive
12	Auto DMA
13	Auto DMA

TABLE 7.5-10. The 14 DMA channels on the ADSP-TS0xx.

DMA is able to address and transfer two-dimensional memory arrays, in which the X and Y directions on both transmit and receive sides have separate counts and address increments. In addition, the ADSP-TS0xx DMA controller allows DMA chaining. In this scenario, the ADSP-TS0xx automatically configures a new DMA transfer (in the same or a different DMA channel) when an entire transfer is finished.

The ADSP-TS0xx has an extremely flexible and elaborate DMA controller. However DMA transfers may conflict with the processor core for on-chip RAM access.

- **Bit I/O**

  The ADSP-TS0xx has four bit-I/O pins which can be individually configured as inputs or outputs.

- **Host Port**

  The ADSP-TS0xx external memory interface doubles as a host port, although the processor manual does not use that term. Via the host port, a host processor can directly access the on-chip memory of the ADSP-TS0xx. The host processor can initiate DMA transfers and issue commands to the ADSP-TS0xx by triggering a vector interrupt. The host can also initiate burst transfers.

  The lack of serial ports on the ADSP-TS001 is surprising.

On-Chip Debugging Support

ADSP-TS0xx DSPs have an IEEE-1149.1 JTAG serial debugging interface with boundary scan and on-chip debugging support. Any of the models of the Analog Devices SHARC in-circuit emulators also work with the ADSP-TS0xx and make use of this serial interface. Through the JTAG interface, memory and registers can be inspected. Also, watchpoints and breakpoints can be set based on individual addresses or address ranges in program or data memory. Instruction and bus-tracing capabilities are not provided.

Power Consumption and Management

As of this writing, Analog Devices has not disclosed information regarding power consumption for the ADSP-TS001.

Benchmark Performance

As of this writing, the ADSP-TS0xx has not been benchmarked with the BDTI Benchmarks. BDTI expects to benchmark the ADSP-TS0xx within the next year, assuming that Analog Devices provides a cycle-accurate simulator.

Based on BDTI's knowledge of the ADSP-TS0xx architecture. We expect that its cycle counts will be similar to those of the TMS320C64xx in many cases if 16-bit fixed-point data is used.

> *Based on processor's projected clock speed of 120 MHz, we expect that ADSP-TS001 execution times will be significantly slower than those of the TMS320C64xx, however, because the projected instruction cycle rate of the TMS320C64xx is five times faster than that of the ADSP-TS001. We also expect that ADSP-TS001 energy consumption and program memory usage will be high.*

Cost

Projected price and packaging options for ADSP-TS0xx processors are shown in Table 7.5-11.

Fabrication Details

The ADSP-TS001 is fabricated in a 0.18 µm process with a 1.8 volt core supply and 3.3 volt I/O, according to Analog Devices.

Development Tools

Analog Devices provides "VisualDSP," an integrated development environment for the ADSP-TS0xx that includes an assembler, C compiler, C run-time library, linker, loader, and instruction-set simulator. VisualDSP runs under Windows 9x, Windows 2000, and Windows NT on IBM PC-compatible computers, and on Solaris 2.5.1. Analog Devices also offers PCI-, USB-, and Ethernet-based emulators.

> *Analog Devices' basic software development tools for the ADSP-TS0xx are generally full-featured and quite sophisticated; however, the instruction-set simulator is not currently cycle accurate, complicating software development and debugging. The ADSP-TS0xx tools provide a fairly consistent interface with those of other Analog Devices processors.*

Little third-party support exists today for the ADSP-TS0xx.

Part	Speed (MHz)	Voltage (V)	Package	Price (Qty. 10,000)
ADSP-TS001[a]	120	1.8/3.3	360- SBGA, PBGA	$150

TABLE 7.5-11. ADSP-TS0xx projected price and package summary as of June 2000.

a. Not yet available.

Applications Support

Analog Devices supports the ADSP-TS0xx with user's manuals, applications handbooks, and a quarterly newsletter, *DSPatch*. The main documentation for ADSP-TS0xx processors is the *TigerSHARC Hardware Specification* and *TigerSHARC Instruction Set Specification*. Analog Devices' DSP technical support group can be reached via telephone and electronic mail. A large amount of information is available from Analog Devices' website. Data sheets are available for the individual ADSP-TS0xx processors.

> *In general, Analog Devices' user's manuals and application handbooks are some of the best in the industry. They are thorough, clear, and well organized. As might be expected, the ADSP-TS0xx manuals are not as polished as those for Analog Devices' more mature products (such as the ADSP-21xx), but are still quite usable.*

Advantages

- High levels of parallelism
- Rich, orthogonal instruction set
- Algebraic assembly language
- Special instructions for FFT, DCT kernel computations
- Very wide internal and external buses
- Large on-chip memory
- Two independent fixed/floating-point data paths
- Good operand-unrelated parallel move support
- Conditional execution of all instructions
- 8-, 16-, 32-, and 64-bit fixed-point data types, plus 32-bit and 40-bit floating-point types
- Barrel shifter
- Multiprecision arithmetic support
- Exponent detect instruction
- Bit manipulation and bit-field manipulation instructions
- Shifter efficiently handles varying-length fields in a bit stream
- Mostly invisible pipeline
- SIMD operations on 8-, 16-, and 32-bit data
- Large number of registers for operands and addressing (64 32-bit general-purpose registers, 62 32-bit address/modify registers); eight individual modulo register sets
- Address generators have extensive arithmetic/logic capabilities
- Branch Target Buffer for minimizing branch penalties

- Flexible external memory interface with support for SDRAM
- Powerful DMA controller
- Large, unified address space (approximately 4 Gwords)
- Good support for interprocessor communications
- Short interrupt latency
- Flexible interrupt handling
- Integrated multiprocessor communications ports
- 2X - 5X external clock multiplier
- Two timers
- JTAG debug port with boundary scan

Disadvantages

- Complicated programming model
- Limited hardware loop support
- No serial ports
- Two-cycle arithmetic operation latency
- Requires execution from on-chip memory for best performance
- DMA and register-to-register transfers occupy an internal bus
- No stack
- Large instruction word increases system cost
- No cycle-accurate simulator as of this writing

7.6 Lucent Technologies DSP16xxx Family

BDTImark2000 Score: 810 at 170 MHz

Introduction

The DSP16xxx is a family of enhanced conventional 16-bit fixed-point DSP processors from Lucent Technologies. DSP16xxx processors target digital cellular telephony, modem, and wireless communication applications.

At the heart of the DSP16xxx processor family is Lucent's enhanced conventional DSP16000 core, which is the successor to the DSP1600. (Lucent considers the DSP1600 a legacy architecture and is no longer promoting it for new designs.) Two different members of the DSP16xxx family are currently available: the DSP16210, and the DSP16410. The DSP16210 contains one DSP16000 core; the DSP16410 contains two DSP16000 cores.

The most recent member of the DSP16xxx family, the DSP16410B, was introduced in May 2000. This device is currently sampling with each core operating at up to 170 MHz with a core voltage of 1.8 volts. The DSP16210 is available in several packages (see Table 7.6-5), one of which (144 TQFP) is pin-compatible with the DSP1620. Because of differences in the I/O and the presence of an additional DSP core on the DSP16410, the DSP16210 and DSP16410 are not pin compatible.

> *Lucent Technologies' DSP16xxx processor family is true to its DSP heritage, achieving strong performance with a specialized architecture that sacrifices generality and ease of programming. While the DSP16000 core bears a strong resemblance to Lucent's DSP1600 core (introduced in 1990), it adds significant capabilities in the data path and instruction set and boosts on-chip memory bandwidth.*

Architecture

The DSP16xxx includes a 16-bit fixed-point data path, two address generation units, a program control unit, a 31-double-word instruction cache, two separate on-chip bus sets, and a variety of peripherals. The DSP16xxx is designed to operate primarily on 16-bit data but uses 32-bit buses internally and a mixture of 16- and 32-bit instructions, as shown in Figure 7.6-1.

Data Path

The DSP16000 data path, called the data arithmetic unit (DAU), is comprised of two $16 \times 16 \rightarrow 32$-bit multipliers, a 40-bit arithmetic logic unit (ALU), a 40-bit three-input adder, a 40-bit bit manipulation unit (BMU), and eight 40-bit accumulators. Of the 40 bits in each accumulator, 8 are used as guard bits. The DAU also includes a trace-back encoder to accelerate Viterbi decoding. The Viterbi trace-back encoder is considerably improved on the DSP16410. The two multipliers, ALU, adder, and BMU are all capable of single-cycle execution, and many combinations of operations can be executed in parallel. For example, the DAU is capable of executing two multiply-accumulate operations per instruction cycle, assuming that instructions are executed from cache. The DAU is illustrated in Figure 7.6-2.

The ALU, adder, and BMU set several status flags based on their computed results. These flags are used to indicate arithmetic overflow, logical overflow, a zero result, or a negative result. (Logical overflow occurs when the ALU or adder result does not fit in the 40-bit destination, or a BMU control word is out of range.) When multiple operations affecting the same DAU flags are executed in parallel as part of a single instruction, the operation that appears leftmost in the instruction line controls the status of the flags. Restrictions on the ordering of parallel operations within instructions dictate that ALU operations (when present) appear in the leftmost position, followed by either adder or BMU operations. Hence, in an instruction containing both ALU and adder operations, the DAU flags are set by the ALU.

The DSP16000 can process data and instructions as single or double words. Single words are 16 bits long; double words are comprised of two consecutive single words and are 32 bits long. Single and double words can be freely intermixed. To accommodate the mixed-width word format, accumulators, multiplier input registers, and product registers can all be accessed in either full-length or half-length form. For example, the high half (bits 31:16) of multiplier input register X can be accessed as XH, while XL denotes the

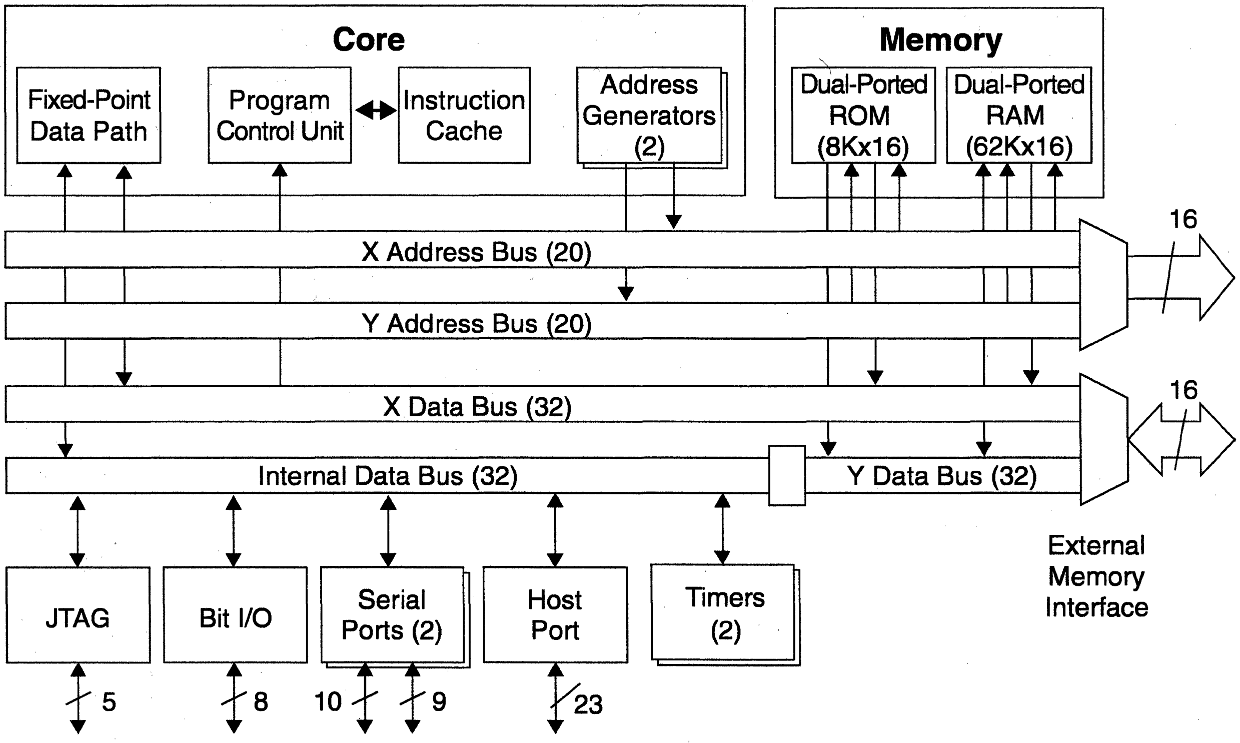


FIGURE 7.6-1. DSP16210 architecture. The internal data bus becomes the Y data bus before connection to memory.

low half (bits 15:0). The full-length register can be accessed simply as X. Similarly, A1H denotes the high half (bits 31:16) of accumulator A1, A1L denotes the low half (bits 15:0) of A1, and the guard bits of A1 (bits 39:32) are designated as A1G. In addition, when transferring accumulator contents to and from memory, the high halves of two accumulators can be concatenated to form a 32-bit accumulator "vector." This is designated by, for example, A0_1H (for the concatenated high halves of accumulators zero and one).

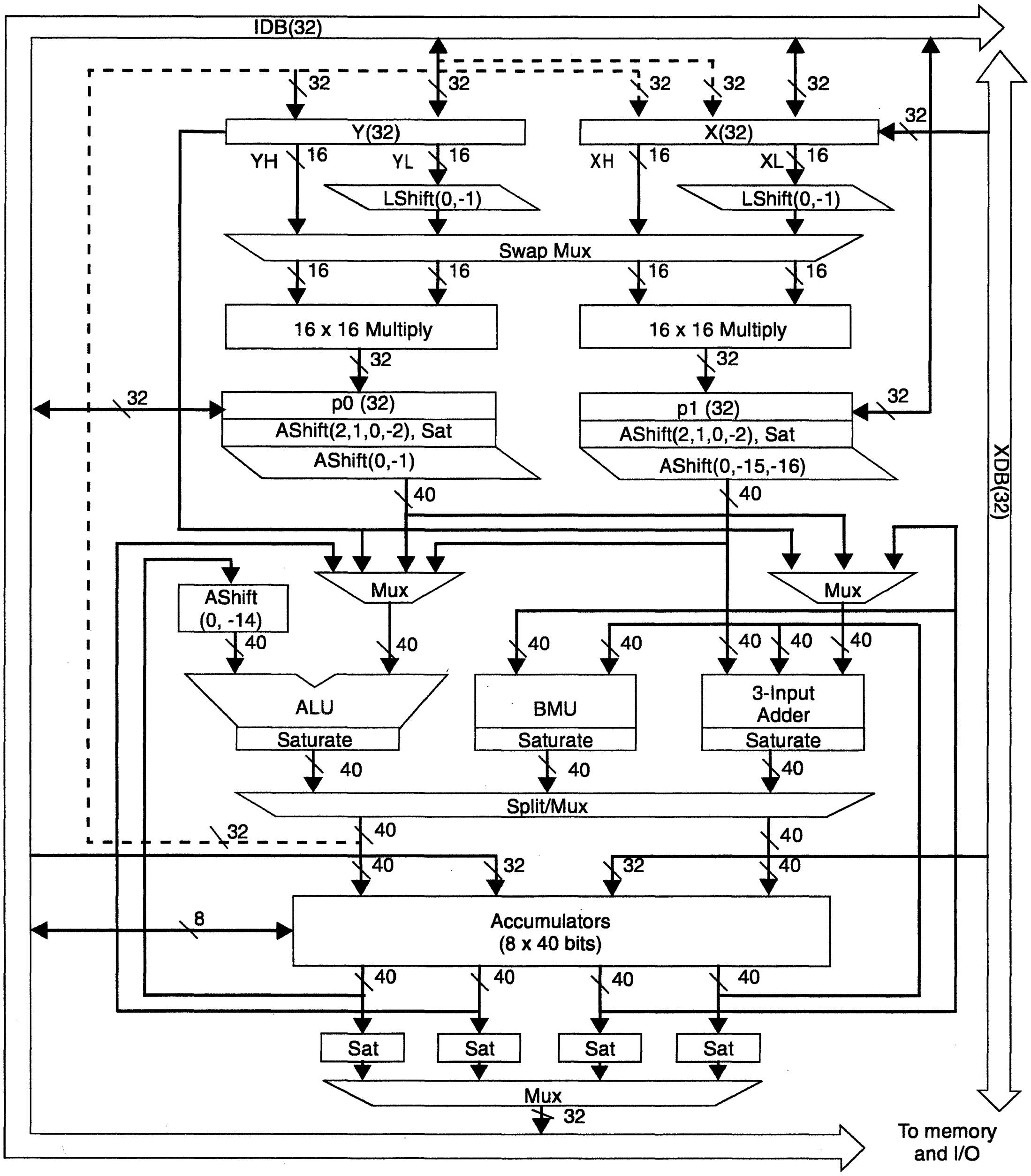


FIGURE 7.6-2. DSP16210 data path. Dashed lines indicate mode-controlled paths. AShift denotes an arithmetic shift; LShift denotes a logical shift. Negative shift values correspond to right shifts.

The DSP16000 provides a large number of accumulators, and allows unusual flexibility in how they are accessed.

The DSP16000 includes two multipliers, each of which can take input data from the 16-bit high or low halves of registers X and Y, or can square the contents of XH or XL. In addition, a mode-controlled option allows the contents of X and Y to be simultaneously loaded with the same data. There are restrictions on the combinations of multiplier inputs that can be used; for example, it is not possible to use YH and YL as inputs to the same multiplier. The 32-bit multiplier results are stored in product registers P0 and P1.

The multipliers can perform integer and fractional multiplications on signed operands. Unsigned multiplication operands are not supported, but the DSP16000 includes several limited-capability shifters in the data path to facilitate extended-precision arithmetic, as shown in Figure 7.6-2. A mode-controlled option allows the contents of the product registers to be shifted left by one or two bits or right by two bits (on an individual basis) before accumulation. In addition, the contents of register P1 can be arithmetically shifted right by 0, 15, or 16 bits before accumulation, as specified by the instruction.

The DSP16000 has good support for multi-precision arithmetic. This is useful for implementing bit-exact telecommunications algorithms such as GSM speech coding. For example, the ability to shift P1 by 16 bits to the right, then one bit to the left [i.e., ((P1>>16)<<1)] before accumulation is useful for implementing the ETSI enhanced full-rate GSM speech compression standard.

The DAU includes a mode-controlled option that allows results deposited in accumulators A6 and A7 to be simultaneously routed to multiplier input registers X and Y, respectively. This feature can only be used with the results of arithmetic or logic functions; data transfers to the accumulators do not affect the contents of X or Y.

The feedback path from the accumulators to the multiplier input registers is unusual. This feature allows programmers to avoid an explicit move instruction in cases where the result of an ALU or adder operation is to be used as input to a subsequent multiply. Hence, use of the feedback path may eliminate a cycle or two in some critical inner loops.

The ALU supports 16-bit, 32-bit, and 40-bit operands. Inputs to the ALU may come from the 32-bit product registers, the 32-bit Y register, any of the eight accumulators (A0-A7), or as immediate data from an instruction. The ALU automatically sign-extends 32-bit operands to 40 bits, and generates a 40-bit output that can be stored in any of the eight accumulators. In addition, the ALU can perform two parallel 16-bit additions or subtractions by using a single-instruction-multiple-data (SIMD)-style operation.

The three-input adder can draw its 32-bit inputs from the product registers, the Y register, or any accumulator. Operands are sign-extended to 40 bits prior to addition; the 40-bit result can be stored in any accumulator.

The presence of a three-input adder in addition to the ALU is an advantage, as it allows both product registers to be added to an accumulator using a single instruction. Among other things, it can be used (in conjunction with the two multipliers) to efficiently implement FIR filters that use complex data.

The ALU supports the add-compare-select function that is used in Viterbi decoding. To implement the add-compare-select function, the ALU first performs two 32-bit additions on the DSP16210. The DSP16410 core has been enhanced so that one DSP16410 core can perform four 16-bit additions. The add-compare-select function then stores the results in two 32-bit accumulators. These accumulators then serve as inputs to one compare function (or two parallel compare functions with one DSP16410 core), which selects the maximum or minimum value(s) and stores it in one (or two with one DSP16410 core) of the source accumulators. The ALU passes a flag (CFLAG), based on the results of the comparison, to the trace-back encoder. The trace-back encoder can use CFLAG to record the history of a series of comparisons, or to record the location of the maximum or minimum value within a vector. This implementation of the add-compare-select function requires two instructions, and hence consumes two instruction cycles. The DSP16210 can achieve one add-compare-select operation on two 32-bit input data and one DSP16410 core can achieve two add-compare-select operations on four 16-bit input data.

Unlike its predecessor, the DSP1600, the DSP16000 includes a BMU on-core. The BMU can perform bit-field insertion and extraction, and arithmetic or logical barrel shifts of up to 31 bits to the right or left. If a left shift results in the loss of significant bits, a flag is set to indicate logical overflow. The data input to the shifter comes from one of the accumulators, and the operand specifying the shift amount can be taken from the high half of an accumulator, from one of four auxiliary registers (AR0-AR3), or as immediate data from an instruction. The BMU sign-extends 32-bit operands to 40 bits, and produces a 40-bit result which can be stored in any accumulator. The BMU can also perform single-cycle exponent detection (with or without simultaneous normalization), which can be used for block floating-point implementation. The DSP16xxx does not support rotate operations.

The DSP16000 bit manipulation capabilities are quite sophisticated.

Saturation hardware is provided at several points in the data path. If an arithmetic overflow occurs and the saturation mode bit (FSAT) is selected, the result is saturated to the largest-magnitude 32-bit number that has the same sign as the result. Saturation on 32-bit overflow can be individually disabled for any accumulator.

The DSP16000 data path supports round-to-nearest rounding via a special instruction.

Two 16-bit signed counter registers, C0 and C1, are provided by the DSP16000 data path. These registers can be used to count events, such as the number of times a sequence of instructions has been executed. The sign bits of C0 and C1 set two corresponding status flags that can be tested by conditional instructions. The conditional test and branch instructions can test the flags and automatically post-increment the counters to support low-overhead software loops. In addition, several instructions that are executed conditionally (on some condition other than the status of the counter flags) can automatically increment C1 each time the condition is satisfied. These instructions include conditional round, shift, and data transfer.

A third counter register, C2, serves as a holding register for C1. The contents of C1 are copied into C2 whenever a conditional instruction increments the contents of C1 and the condition was true. Examples of this type of conditional instruction include a comparison operation for Viterbi decoding support, and register left- and right-shifts by a limited set of powers of two.

> *Overall, the DSP16000 data path is relatively powerful. With two independent multipliers and support for parallel additions, the DSP16000 can produce two independent MAC results per instruction cycle—a strong advantage in comparison to conventional DSP processors. In comparison to recent multi-issue architectures, such as the Texas Instruments TMS320C6xxx and StarCore SC140, the DSP16000 does not offer as much parallelism.*

Memory System

The DSP16xxx on-chip memory system uses a modified Harvard architecture with two separate bus sets, X and Y. Each bus set is comprised of a 32-bit data bus and a 20-bit address bus. The X buses are used primarily for reading instructions and constant data, and the Y buses are used primarily for accessing non-constant data. Memory is physically organized in 16-bit words, but it can be accessed as either 16-bit single words or as 32-bit double words composed of two consecutive single words. Read or write accesses of single words and aligned double words complete in one instruction cycle. (A double word is considered to be aligned if it has an even address.) Memory reads of misaligned double words generate a one-cycle penalty wait-state. For the second and subsequent data memory reads from sequential misaligned memory locations (e.g., using post-incrementing pointer accesses), special hardware enables accesses to be performed with no penalty wait-states.

> *The ability to perform 32-bit accesses without requiring 32-bit alignment is an advantage.*

Except for one block of 512 words on the DSP16210, the internal memory space of the DSP16xxx is unified, meaning that both X and Y buses can access the same addresses. The DSP16000 core can support on-chip memory of up to 128 Kwords of dual-ported RAM and 384 Kwords of dual-ported ROM (words are 16 bits wide).

The DSP16210 chip features a 60 Kword dual-port RAM divided into 1Kx16 blocks, and an 8 Kword boot ROM. Each core in the DSP16410 has a private block of 96 Kword triple-ported RAM: one port accesses the X-memory space, one port the Y-memory space, and one port the DMA space (called "Z memory space" by Lucent). In addition, the DSP16410 has two 1Kx16 banks of shared local memory (SLM) that can be accessed by both cores. However, an access to the SLM takes multiple cycles to execute. Each core in the DSP16410 also has its own boot ROM containing a simple boot routine (identical on both cores). For more details on the boot routine, refer to *Bootstrap Loading* section.

The memory system on the DSP16410 has been considerably improved over that of the DSP16210 so that any host can access the entire memory space of both cores through the DMA space.

Simultaneous X and Y memory accesses to ROM memory or to the same 1Kx32 *module* of RAM incur a penalty of one instruction cycle. (Lucent defines a memory module as an even and odd pair of 1Kx16 banks.) The DSP16xxx incurs a conflict wait cycle when a write is immediately followed by a read to the same memory module.

Conflict wait states often occur during the most computationally intensive loops of DSP algorithms, where successive reads and writes in the same memory module are required; e.g., in DSP algorithms requiring in-place computations.

The two on-core bus sets enable the DSP16000 to perform two single- or double-word memory accesses per instruction cycle per core. The DSP16000 can simultaneously execute two reads or one read and one write per core, provided that they do not access the same memory module, but it cannot perform simultaneous writes because the X data bus can only be used for memory reads. By reading 32-bit memory locations, each memory access can retrieve two 16-bit words. Therefore, a total of four 16-bit words can be fetched in a single cycle if no wait-state occurs.

An on-chip data access rate of two memory accesses per instruction cycle is comparable to most conventional DSP processors. However, if two 16-bit operands are packed into a 32-bit word, four 16-bit operands can be fetched by the DSP16000 in one instruction cycle.

The memory accesses required to fetch the four 16-bit data inputs and instructions needed for dual MAC operations would normally consume two instruction cycles. To improve memory bandwidth and achieve single-cycle dual MAC operations, the DSP16000 includes a 32-bit wide cache memory that can be loaded with up to 31 single- or double-word instructions. The contents of the cache are loaded under program control using the DO instruction, and are executed as part of a zero-overhead loop. During the first iteration of the loop, instructions are executed from program memory and simultaneously loaded into cache. In subsequent repetitions, instructions are fetched from the cache, freeing the X bus to be used for data transfers. Hence, use of the cache decreases the number of cycles required to execute instructions that load two non-adjacent data words. For example, four 16-bit data fetches, two multiplications, and two accumulations can be completed in a single cycle if the instructions are executed from cache and the data is accessed as pairs of 16-bit words. Hence, use of the cache allows the DSP16000 to produce two MAC results per cycle, if the instructions are executed as part of a loop and operands are accessed as pairs. The cache does not support nested loops, and most control-flow instructions cannot be cached.

> *This cache design allows Lucent to achieve three-bus memory bandwidth in small inner loops (which account for a sizable fraction of the computation load in typical DSP applications) without the expense of a third bus or a more complex cache. However, limitations inherent in the cache design (lack of support for nested loops and control-flow instructions) detract from the cache's utility and complicate programming.*

Assuming that instructions are executed from cache and that only aligned double words are accessed, the maximum sustainable on-chip data memory bandwidth for a 150 MIPS DSP16000 core is six hundred 16-bit Mwords/second for reads and three hundred 16-bit Mwords/second per core for writes.

> *The DSP16xxx has an effective on-chip data access rate that is quite high in comparison to most conventional DSP processors.*

External Memory Interface

The DSP16210 processor has one external memory interface (EMI), which provides a 16-bit data bus and a 16-bit address bus. The DSP16410 includes one external memory interface referred as SEMI (System and External Memory Interface), which provides a 32-bit data bus.

The external buses are multiplexed between on-chip X and Y memory buses, with the four most significant bits of the incoming X or Y address bus (bits 19:16) used for generating chip-select signals. One 16-bit external memory read or write can be made per instruction cycle on the DSP16210 (assuming zero wait states). Note however that if consecutive accesses are made to the external memory on the DSP16210, one wait-state is required between two consecutive loads or writes. Although the external memory interface of the DSP16210 only provides a 16-bit data bus, it is capable of processing both 16-bit and 32-bit words. The DSP16210 interface builds a double word by performing two single-word fetches, and then transfers the entire 32-bit word over the appropriate internal data bus. The DSP16410 can achieve one 32-bit external memory read or write per instruction cycle, assuming no wait states. However, in asynchronous mode, one wait-state is required between consecutive reads from the external memory and two wait-states are required between consecutive writes from the SEMI external memory.

The DSP16xxx handles simultaneous X and Y fetch requests to unified external memory by reading the X data first and the Y data second, and then transferring X and Y data from the external memory interface to the core simultaneously. Simultaneous X read and Y write requests are handled similarly; the X data is read and transferred and then the Y write is performed. In both cases, a penalty of one cycle is incurred. Assuming zero wait states and one stall between two consecutive external memory accesses, the maximum sustainable off-chip memory bandwidth for a 150 MHz DSP16210 is 75 16-bit Mwords/second for reads or writes. Assuming one stall between two consecutive reads from the external memory and two stalls between consecutive writes to the external memory, the maximum sustainable off-chip memory bandwidth for a 150 MHz DSP16410 is 150 16-bit Mwords/second for reads and 100 16-bit Mwords/second for writes.

The external memory bandwidth of the DSP16xxx is comparable to that of most other DSPs. However, accessing one 32-bit instruction in external memory requires two instruction cycles for the DSP16210, which may hinder the processor's performance when 32-bit instructions are stored off-chip. This drawback also affects the DSP16410 when 32-bit instructions need to be loaded for one of the two cores.

As is common among newer DSPs, the ratio of on-chip memory bandwidth to off-chip memory bandwidth on the DSP16xxx (4:1 for reads on the DSP16210) is higher than in older DSPs. This makes it increasingly important that programmers make good use of on-chip memory to tap the processor's performance potential.

External memory is divided into four regions—high and low RAM segments, ROM, and I/O. One chip-select line is provided per region, with an additional select line available for enabling the entire external RAM. The processor supports programmable wait states, allowing 0 to 15 single-cycle wait states to be assigned to the various regions of memory. The ROM and I/O regions have individually programmable wait states, while the high and low RAM segments share the same wait-state programming. The external memory interface of the DSP16xxx supports externally requested wait states by providing a READY signal, which can be asserted by an external memory device to extend the access cycle. Any memory segment that uses the READY signal must be programmed with a minimum of four wait states because of the signal's timing requirements. The READY signal can be selectively disregarded for specified memory regions by setting a flag in a configuration register.

The external memory interface of the DSP16210 does not allow another device to obtain bus mastership of its on- or off-chip buses. The external memory interface of the DSP16410 supports shared memory accesses assuming an external bus arbiter.

Address Generation Units

The DSP16000 core provides two address generation units: the X address arithmetic unit (XAAU) and the Y address arithmetic unit (YAAU). The XAAU and YAAU serve different purposes; the XAAU generates addresses for instructions and constant data, and the YAAU generates addresses for non-constant data.

The XAAU contains the program counter (and shadow registers for interrupts and subroutine calls) and two 20-bit pointers (PT0 and PT1) that can be used to address data in

X memory space. Also included in the XAAU are two modifier registers (H and I), used for post-modification of PT0 and PT1.

The YAAU includes eight 20-bit pointer registers (R0-R7), two modifier registers (J and K) used for post-modification of R0-R7, and a pair of modulo arithmetic units with "base" and "end" registers (RB0 and RE0; RB1 and RE1) for modulo addressing. A 20-bit stack pointer (SP) in the YAAU is used for generating indexed addresses, and replaces the YBASE register found on the DSP16xx.

The DSP16000 supports immediate data, register-direct, register-indirect, and indexed addressing modes. Bit-reversed addressing is not supported.

> *The fact that bit-reversed addressing is not supported is not common among DSPs. This forces the programmer to use bit-reversion tables for algorithms such as the FFT.*

In X memory space, modifications to the PT registers for single-word accesses include post-increment by one, post-decrement by one, post-increment by the contents of modifier register I or H, and no update. Register modifications for double-word accesses in X memory space are limited to post-increment by two and post-increment by the contents of modifier register I or H. If a double-word access loads an accumulator vector however, the list of register modifications expands to include post-decrement by two and no update.

Similar register modifications are available in Y memory space, with the addition of post-decrement by two and no update for double-word accesses. Y memory space uses modifier registers J and K instead of H and I.

Modulo addressing is supported via the RB and RE registers, which specify the beginning and ending addresses of a circular buffer. Writing a non-zero value to RE0 enables modulo addressing for YAAU registers R0-R3 (all four are enabled simultaneously); similarly, writing a non-zero value to RE1 enables YAAU registers R4-R7. Modulo addressing is not available in the XAAU. Registers R0-R3 (or R4-R7) must be updated with a simple post-increment (e.g., Y=*R0++) for the modulo addressing to function properly. Other register modifications (e.g., Y=*R0++J) do not update the registers using modulo addressing. When a pointer register becomes equal to the ending address and a post-increment is specified, the register is reloaded with the value of the beginning address. Single words, double words, or a combination of the two can be used in a circular buffer.

> *The support for circular buffering is somewhat inflexible. The fact that address registers must be updated using a simple post-increment may limit the use of circular buffers in some applications.*

Compound addressing (an addressing mode used on the DSP16xx to compensate for its inability to perform single-cycle writes by allowing a simultaneous read and write in two instruction cycles) is not available on the DSP16xxx.

The DSP16xxx's lack of compound addressing is not a drawback in itself, but does reduce the level of compatibility with existing DSP16xx assembly language source code.

Pipeline

The DSP16xxx processor uses a three-stage pipeline that is generally not visible to the user. The DSP16xxx assembler can detect instruction sequences that would cause a pipeline hazard, issue a warning, and automatically insert NOPs to prevent the hazard from occurring. This feature can be disabled using assembler directives embedded in the assembly source code. The assembler cannot fix pipeline hazards detected in cache loops, but a warning is still issued. One of the few instances of pipeline visibility occurs when a value is written to an address register modifier, then used immediately afterwards. In this case, the assembler issues a warning and inserts a NOP to delay access to the modifier by an instruction cycle.

Instruction Set

The DSP16xxx uses both 16-bit and 32-bit instructions, and the two can be freely mixed. Many of the 16-bit instructions are similar to DSP16xx instructions. Table 7.6-1 lists the main DSP16xxx registers and their purposes. The instruction set is summarized in Table 7.6-2. The DSP16xxx relies on mode bits to configure many aspects of the processor's data path, including shifting and saturation; for example, mode bits select one of four scaling options at the output of each multiplier.

Assembly Language Format

The DSP16xxx assembly language uses an algebraic syntax with separate fields for control of different execution units. Multiple operations can be combined in a single instruction, and are executed from left to right. For example, the instruction

```
a0=a4+p0 a1=a5+p1 p0=xh*yl p1=xl*yh x=*pt0++ y=*r1++
```

directs the DSP16xxx to perform two additions, two multiplications, and two data transfers (with register post-increment) in parallel, allowing single-cycle dual-MAC operations. Combinations of operations that can be performed in parallel as part of a single instruction are shown in Table 7.6-3.

Instructions can be categorized as either "regular" or "extended." Regular instructions are generally comparable to DSP16xx instructions. They are 16 bits wide and support a limited subset of operations on a limited subset of the available registers. For example, a multiply operation may be performed in parallel with an ALU operation and up to two 16-bit data transfers. Extended instructions are 32 bits wide, and are able to use the full complement of registers, though not without some restrictions. Extended instructions can use either 16- or 32-bit operands. In general, extended instructions allow a greater number of operations to be executed in parallel than regular instructions, and provide better support for parallel moves. For example, up to two multiply operations may be performed in parallel with an ALU and adder operation and up to two 32-bit data transfers.

Lucent Technologies' goal in using mixed-width instruction words combined with a large number of mode bits is to facilitate compact code while improving per-cycle efficiency. The benchmark results indicate that this strategy is effective; however, the multitude of mode bits has the potential to significantly complicate programming.

Although the DSP16000 assembly language is not compatible with that of the DSP1600, DSP1600 programmers should be comfortable programming the DSP16000. Many DSP16000 instructions are very similar (or even identical) to those of the DSP1600.

Registers	Width	Purpose
A0-A7	40 bits	Accumulators
X	32 bits	Multiplier input (16 MSBs or LSBs)
Y	32 bits	Multiplier input (16 MSBs or LSBs), ALU input, adder input
P0, P1	32 bits	Product registers
AR0-AR3	16 bits	Auxiliary registers
PT0, PT1	20 bits	X memory address registers
H, I	20 bits	Modifier registers for PT0, PT1
R0-R7	20 bits	Y memory address registers
J, K	20 bits	Modifier registers for R0-R7
RB0, RE0 RB1, RE1	20 bits	Base and end registers for circular addressing
PI	20 bits	PC interrupt shadow register
PR	20 bits	Return address register (subroutine calls)
C0-C2	16 bits	Counters

TABLE 7.6-1. DSP16000 register summary.

Parallel Move Support

The DSP16xxx supports operand-unrelated parallel moves with MAC operations and some ALU operations. Like most DSPs, the DSP16xxx places restrictions on the use of parallel moves, the most significant of which are:

- Only certain instructions allow parallel data moves.
- Y data addresses are generated by the Y address generation unit, which can use registers R0-R7. X data addresses are generated by the X address generation unit, and can only use registers PT0 or PT1.

Instructions with parallel reads from Y memory usually execute in one instruction cycle. Instructions with parallel X memory reads or parallel X and Y memory reads execute in two instruction cycles unless the instructions are fetched from cache, which allows single-cycle execution. Many extended instructions allow two 32-bit parallel data moves; regular instructions can only transfer two 16-bit words at a time. Thus, the DSP16xxx can

Class	Instructions
Arithmetic	Add, subtract, increment, decrement, round, negate, absolute value, dual add/subtract, quad add/subtract, three-input add
Multiplication	Multiply, multiply-accumulate, multiply-subtract, square
Logic	*And*, *or*, *not*, *exclusive-or*
Shifting	Arithmetic or logical shift left or right by 0-31 bits
Rotation	*<None>*
Conditional Execution	Negate, round, absolute value, fixed arithmetic shift, branch, subroutine call, return, and most extended instructions can be executed conditionally
Comparison	Compare, select maximum or minimum; dual compare, select maxima or minima
Looping	Single- and multi-instruction hardware loop, conditional counter increment and branch based on value
Branching	Unconditional branch, conditional branch
Subroutine Call	Unconditional subroutine call and return, conditional subroutine call and return
Bit Manipulation	Bit field insert or extract, test for pattern
Special Function	Swap accumulators, exponent computation, normalize, divide-step

TABLE 7.6-2. DSP16000 instruction set summary. Instructions that are supported only on the DSP164xx are underlined.

perform up to four parallel 16-bit memory reads, two 16-bit memory writes, or two 16-bit reads and two 16-bit writes, if words are arranged as pairs in memory. The DSP16xxx provides support for reading two 16-bit values stored as a single 32-bit value, with each 16-bit word stored to a separate accumulator. The DSP16xxx can also write two 16-bit quantities from two accumulators as a 32-bit vector to memory.

The DSP16xxx's support for parallel moves is comparable to that of most older DSP processors in terms of the number of moves that can be performed per instruction cycle; like many newer DSP processors, however, the DSP16xxx can transfer two adjacent 16-bit words per parallel move. Thus, the effective number of parallel moves that can be performed per instruction cycle (four 16-bit words) is high, if words are arranged as pairs in memory.

Orthogonality

The instruction set of the DSP16xxx is not particularly orthogonal, but restrictions on instruction format and register usage are not as severe as those seen on the DSP16xx. The new, 32-bit instruction words used on the DSP16xxx could have been used to implement a highly orthogonal instruction set. Instead, the instruction set was designed to allow a higher degree of parallelism than is available on many other DSP processors. The result of this trade-off is that the DSP16xxx can specify up to six parallel operations in a single instruction, but the operations can generally use only a subset of the processor's registers,

Operation (execution unit)	Combinations Supported in a Single Instruction				
Add/Subtract (ALU)	X				
Round (ALU)		X			
Absolute Value (ALU)			X		
Minimum (ALU)					X
Compare (ALU)				X	
Add/Subtract (adder)	X			X	
Two Multiplications (MPY)	X				
Shift (BMU)		X	X		
Exponent (BMU)					X
Two Data Transfers	X	X	X	X	X

TABLE 7.6-3. Combinations of operations supported in a single instruction. Each column represents a combination of operations that can be executed in parallel as part of a single instruction.

and only certain combinations of operations can be executed in parallel. If just one operation is specified in an instruction, however, the restrictions on register usage are greatly eased.

> *For single-operation instructions, a high degree of orthogonality is attained, resulting in relatively easy programming. For the multi-operation instructions required for high-performance code, however, the instruction set is less orthogonal and harder to use. Programming is also complicated somewhat by the large number of mode bits. Overall, the DSP16xxx is challenging to program.*
>
> *Lucent Technologies asserts that the level of programming difficulty present on the DSP16xxx is a result of design decisions that allow the processor to achieve high execution speed at a reasonable price. They view the resulting programming difficulty as an acceptable trade-off in their target applications.*

Execution Times

MAC, ALU, and BMU operations complete in a single cycle, and can often be executed in parallel. Instructions affecting program control-flow, (e.g., branches and subroutine calls) generally consume three instruction cycles, but conditional branch instructions that do not take the branch execute in two cycles. Branches may require four cycles if the instruction at the target address is a misaligned double word. Delayed branches are not supported.

Instruction Set Highlights

The DSP16xxx provides a number of noteworthy instructions:

- Conditional execution of arithmetic functions and many accumulator modification instructions (including shifts, increment, maximum/minimum, and round)
- Notable conditions that can be tested during conditional instruction execution include the output of a single-bit pseudo-random number generator (the "heads or tails" condition), the sign of one of two post-incrementing counters, the parity of the last BMU operation, and the status of bits in the bit I/O port
- The normalization, exponent, extract, and insert instructions use the on-core bit manipulation unit for single-cycle exponent detection, exponent detection with simultaneous normalization, bit-field extraction, and bit-field insertion
- Special instructions support Viterbi decoding and multi-precision arithmetic. Table 7.6-4 summarizes the differences between the Viterbi instructions available on the DSP16210 and on the DSP16410.

Execution Control

Clocking

The DSP16xxx processor uses a 1X master clock, whose source is specified under software control.

Clock source options include an external oscillator, an internal oscillator, or the JTAG test clock. The DSP16210 has a user-programmable phase-locked loop (PLL) frequency synthesizer. The PLL can generate a master clock at a frequency that is *M*/*N* times the frequency of the input clock source, where *M* can range from 2 to 20, and *N* can range from 1 to 8. The PLL can be disabled (for power reduction purposes) under software control. Both cores of the DSP16410 execute from a common clock source (either the external clock, the PLL, or the JTAG clock). The clock source is controlled by one of the cores, which selects the source for both cores and sets M and N of the PLL.

The DSP16xxx uses a two-phase clock system; it produces versions of the clock signal that are in-phase and out-of-phase with the master clock. The processor supports integration of DSP16xx-style peripherals and modules by generating both inverted and non-inverted versions of the two clock phases.

Hardware Looping

The DSP16xxx family supports hardware looping with the DO instruction. The DO instruction loads the cache with up to 31 instruction words, executing them from program memory as they are loaded. Once loaded, the instructions execute from cache for the specified number of repetitions, freeing the X bus to be used for data fetches. The cache

Special instructions available on the DSP16210/16410 for Viterbi decoding	Special instructions available on the DSP16410 for Viterbi decoding
Dual add/subtract: Example: a0=a4-y a1=a5+y	**Quad add/subtract**: Example: a0h=a4h+yh a0l=a4l+yl a1h=a4h-yh a1l=a4l-yl

Compare, select max or min:
Example:
a0=cmp1(a1,a0)
The binary result of the comparison may simultaneously be stored in register ar0 as follows:
ar0=(ar0<<1)|(?a1>a0)
	Dual compare, select max or min:
	Example:
	a0=cmp1(a0h,a0l) a1=cmp1(a1h,a1l)
	The binary result of the two comparisons may simultaneously be stored in registers ar0 and ar2 as follows:
	ar0=(ar0<<1)|(?a0h>a0l)
	ar2=(ar2<<1)|(?a1h>a1l)

TABLE 7.6-4. Examples of Viterbi instructions available on the DSP16210 and on the DSP16410.

can accommodate both 16-bit and 32-bit instructions. Control-flow instructions cannot be included in a DO loop.

The instructions in cache can be executed from 1 to 65,535 times if the repetition count is taken from the 16-bit CLOOP register, or from 1 to 127 times if the count comes from immediate data. A REDO instruction can later be used to re-execute the instructions that were most recently loaded in the cache. Cache loops are interruptible, and several registers are provided which contain information about the state of the cache. These registers can be saved and restored, along with the contents of the cache, to allow re-entry after an interrupt.

> *The inability to specify a repetition count of 0 in the DO instruction (i.e., to specify that instructions are not to be executed) is inconvenient in some applications. For example, the programmer may want to load the instruction cache without executing the instructions as a means of setting up the cache for a subsequent REDO instruction; this is not supported. In addition, if the repetition count is specified as a variable whose value may be 0, the programmer must explicitly check for this case.*

The DSP16xxx does not allow multi-instruction hardware loops to be automatically nested. However, the two 16-bit counter registers, C0 and C1, can be automatically post-incremented when tested as part of a conditional branch instruction. The three cycles consumed by this instruction are one less than would be required to implement an increment and conditional branch using separate instructions.

> *The inability to nest hardware loops is a drawback. The support for software loops alleviates the problem to some extent, but does not fully compensate for this shortcoming.*

Interrupts

The DSP16000 core supports 20 hardware interrupts. The DSP16210 includes 15 internal hardware interrupts and 64 software interrupts via the ICALL instruction. Each core in the DSP16410 allows 26 hardware interrupts; however, since the DSP16000 core supports a maximum of only 20 hardware interrupts, each core has a programmable interrupt multiplexer.

Internal hardware interrupt sources include the serial ports, parallel port, JTAG interface, and timers. Also included are an external reset pin, a non-maskable TRAP pin, and four external interrupt request pins. The DSP16410 also supports a core-to-core TRAP, which allows one core to interrupt the other core.

> *The core-to-core TRAP is useful for enabling core-to-core communication or synchronization.*

Interrupts are individually prioritized to one of four levels, allowing the processor to resolve simultaneous interrupt requests. Prioritization also provides a mechanism for interrupt nesting by allowing an interrupt service routine to be interrupted by a higher-priority interrupt request. Three user-assigned interrupt priority levels and an "interrupt disabled" setting are available. Each interrupt source can be independently enabled or disabled, and each has its own four-word interrupt vector which can contain either a short service routine or a branch to a longer routine. The return-from-interrupt address is stored in the PI register. Interrupts are not automatically nestable; the service routine must explicitly save and restore the state of the interrupt handler, return register, and cache to allow interrupt nesting. The DSP16xxx provides four vectored interrupt ID pins, which encode the interrupt vector the core is currently servicing. These pins can be used as general-purpose I/O pins by modifying a control register. At reset, the pins are configured as interrupt pins. Processor status is not automatically saved during an interrupt.

Assuming the processor is in an interruptible state, and that the interrupted instruction is a single-cycle instruction, the minimum interrupt latency before execution of the instructions in the appropriate interrupt vector is seven instruction cycles. Also, the core will allow a two- or three-cycle instruction to complete before executing the interrupt vector service handler, which will extend the interrupt latency to eight or nine cycles.

Stack

The DSP16xxx uses a software stack to support interrupt nesting based on the interrupt priority level. Subroutine calls do not use the stack, and instead automatically store the subroutine return address in the PR register. Thus, subroutines cannot be nested without taking explicit steps to save and restore the PR register.

Bootstrap Loading

The DSP16210 can boot from either external memory or from an internal 8-Kword bootstrap ROM. When booting from internal memory, the DSP16210 places the parallel host interface port in 8-bit Intel-compatible mode and awaits a command. The host places a value on the port that selects a routine from one of the 77 available in the bootstrap ROM. Of the 77 bootstrap routines available, 4 enable and set the PLL and core clock, 4 set external memory wait-states, 2 branch to a location in external ROM or internal RAM and begin program execution, 13 perform memory tests, and 54 provide download functionality using either Intel- or Motorola-style transfers. Upon completion of the selected routine, the DSP16210 places the host port into Intel-compatible mode, ready for the next bootstrap routine to be selected. This sequence ends with selection of one of the two branch-and-execute functions.

The DSP16410 can also boot from either external memory or from an internal bootstrap ROM. However, the bootstrap loading process is significantly simplified on the DSP16410 by its enhanced DMA controllers. If the two cores boot from their internal boot ROMs, they execute a boot routine that simply waits for an external host to download code and data via the parallel port (PIU), which has a direct memory access to all DSP memory. (For more information on the parallel port and the direct memory controller, refer to next section.) The host generates a PIU interrupt when the download process has completed.

> *The DSP16410 DMA controllers have been considerably modified so that they can access the entire DSP memory space. This simplifies the bootstrap loading process on the DSP16410 (no need for the host to select a succession of different bootstrap routines) without compromising its flexibility.*

Peripherals

The on-chip peripherals of the DSP16210 include two serial ports with different features, two timers, a parallel host port, and a bit I/O port. The on-chip peripherals of the DSP16410 include two identical serial ports, four timers, a parallel host port and two bit I/O ports.

- **Serial ports**

  The DSP16210 provides a simple serial port and an enhanced serial port. The simple serial port supports 8- or 16-bit data in either MSB-first or LSB-first format. The receive and transmit sections of the serial ports are independent, and each can be controlled by either an external or internal clock. Input and output frame synchronization signals are provided. The DSP16210 has a single internal serial clock generator that divides the master clock by 2, 6, 8, or 10. A programmable DMA controller, called a modular I/O unit (MIOU), is assigned to the simple serial port. The MIOU transfers data between a 1Kx16 block in the on-chip dual-port RAM and the serial port without requiring core intervention. In general, the core and the MIOU operate independently, but if they attempt simultaneous accesses to the dual-port RAM the core access is delayed by one instruction cycle.

  The enhanced serial port processes only LSB-first format, and can operate in simple mode or frame mode. In either mode, the input and output signals are generated externally. In simple mode, 8- or 16-bit serial data can be transferred at up to 25 Mbits/second. In frame mode, up to 16 logical channels of 64, 32, 16, or 8 Kbits/second can be multiplexed or demultiplexed on a 96, 128, 192, or 256 bits/frame TDM data channel. Each logical channel can be set to a bit rate independently of the others. An output or input frame interrupt can be generated every two, four, eight, or sixteen frames. The TDM frame size and interrupt frequency can be selected independently for the physical input and output channels. External input and output frame synchronization signals determine the start of input and output frames.

  The DSP16410 has a pair of identical serial ports that are enhanced versions of the DSP16210 enhanced serial port. Both of the DSP16410 serial ports can process LSB- or MSB-first format and support connection to TDM buses. Using both serial ports, the DSP16410 can process up to 32 TDM logical channels. The serial ports of the DSP16410 also provide optional hardware μ-law or A-law companding.

The DSP16410 serial ports also include a CLK pin, which can be used as a master clock to derive the serial port clock and frame synchronization signals.

The serial interface units are directly connected to the DMA channels. This allows data to be transferred between any memory space of the DSP16410 and the serial port without requiring core intervention.

- **Timers**

  The timers on the DSP16210 and DSP16410 use a 4-bit exponential pre-scaler to divide the master clock by one of 16 factors (2, 4, 8,... , 65,536). The pre-scaled clock is then fed into the 16-bit down-counters, each of which can generate an interrupt upon reaching zero. The timers can then be reloaded with new values and begin counting again.

- **Parallel Host Port**

  The parallel host port on the DSP16210 is called the PHIF16 ("parallel host interface") and is used to transfer data between the host and the DSP16210. The PHIF16 is 16 bits wide, and can only respond to data strobes generated by external devices. It can use either separate read and write strobes, or a single read/write line in conjunction with a chip enable line, making it compatible with both Intel and Motorola signaling conventions. Like the simple serial port, the parallel host port has a dedicated MIOU for exchanging data with the processor via a 1Kx16 block in the dual-port RAM. A data value can be written to the PHIF16 by the host at device reset to select a bootstrap routine from those provided in the internal ROM.

  Like the DSP16210, the DSP16410 provides a 16-bit parallel host port. However, the parallel interface unit (PIU) of the DSP16410 has been enhanced to support interaction with the modified DSP16410 DMA controller. The DMA controller provides the host with access to the entire internal and external memory space of the DSP16410.

  > *The DMA controllers on the DSP16210 are limited in that they can only access 1Kword of IORAM each (1Kword of IORAM for the serial ports and 1Kword of IORAM for the parallel ports). The DMA controllers of the DSP16410 have been significantly enhanced so that the host can access the entire DSP memory space through either the serial ports or the parallel ports.*

- **Bit I/O**

  The DSP16210 provides eight bits of bit I/O. Each bit can be independently configured as input or output, and can be individually set, cleared, or toggled. Additionally, the bit I/O port has circuitry to mask input bits with a pattern and compare the result to a test pattern. The result of the comparison is made available as flags that can be used in conditional instructions. The flags indicate if the pattern matched entirely, if some bits matched, if some bits did not match, or if none of the bits matched. The state of these flags can be recorded in the ALF register for later use. Four of the I/O bits (bits 7:4) share pins with vectored interrupt indication signals. When the device is reset, the vectored interrupt indication signals are connected to the pins. The four I/O bits can be connected to the pins by setting a bit in the IOC register. The DSP16410 provides two groups of seven bits of bit I/O, each controlled by one of the cores. The functionality of the bit I/O pins is the same as that of the DSP16210 but the pins are no longer shared with vectored interrupt ID indicator signals.

The DSP16xxx bit I/O port pattern-matching capability is an unusual feature.

On-Chip Debugging Support

The DSP16210 processor includes a JTAG-based debug and emulation port. The DSP16410 processor includes two complete JTAG-based debug and emulation ports, one connected to each of the DSP16000 cores. This allows the programmer to read and write internal registers and memory, set and clear hardware breakpoints, and start and stop execution on a per-core basis. The JTAG interface to the DSP16210 core and to one of the DSP16410 cores is IEEE 1149.1 compliant and supports boundary scan.

The DSP16xxx also includes an on-chip debugging module. This module contains eight address watchpoint units, two data watchpoint units, four complex condition units, a cycle counter, and a trace unit.

The watchpoint units allow the programmer to specify either address-based or data-based breakpoints. Address-based watchpoints are based on whether the X or Y address is equal to, or greater-than-or-equal-to a pre-selected value. Data-based watchpoints are based on whether the data being transferred to or from memory, or between registers (except between two accumulators) matches a pre-selected pattern. An optional 32-bit mask register allows bits to be masked out of the comparison. When the comparison is valid, a debug event is generated. The watchpoint units can be configured to generate a debug trap, or to pass the event on to the complex condition units.

The complex condition units can logically combine any of the ten events received from the data and address watchpoint units. Each complex condition output can be configured to generate a debug trap, or can be used as the input to another complex condition unit, allowing the programmer to set relatively complex logic-based breakpoints. By chaining the outputs of the condition units, program flow sequences can be trapped.

The processor can be configured to respond to a debug trap in one of two ways: it can execute a service routine that uploads the state of the core to the host debugging computer, or it can execute one of four user-defined service routines that are assigned to specific breakpoint conditions. The latter breakpoint response method is useful for real-time debugging.

The debugging module also records program branch information in the trace unit, allowing the programmer to trace the execution sequence of programs that are complicated by, for example, nested subroutines. Up to eight branches can be recorded in the trace buffer. Single-level loops are recorded as a single entry in the trace buffer; cache loops are not recorded at all. However, discontinuities to and returns from interrupt service routines during execution of a cache loop are recorded. The trace unit can be set to ignore all program flow discontinuities except subroutine and interrupt service routine calls.

The cycle counter unit contains a 32-bit counter which can be set to count freely, or to trap the core after a set number of cycles have elapsed.

> *The debugging unit on the DSP16xxx is unusually flexible and powerful, and reflects the fact that target applications for the DSP16xxx often require extensive real-time testing and debugging.*

Power Consumption and Management

The DSP16210 features a low-power sleep mode, triggered by setting the AWAIT bit in the processor's ALF control register. In this mode only the PLL (if enabled) and the minimum circuitry required to process an interrupt remain active. Any unmasked interrupt awakens the processor from sleep mode.

The DSP16210 features the following power-down capabilities via a power control register (POWERC) and stop pin:

- The power control register (POWERC) features a bit to switch the processor's clock from an external clock to a slower internal clock generated by the on-chip oscillator. Selecting the slower clock before setting the AWAIT bit further reduces power consumption, but increases wake-up latency because latency is a function of clock speed.
- If the user is willing to restrict the wake-up signal to the INT0, INT1, or RESET lines, then the NOCK bit in the POWERC register can be used to shut off the clock to the entire processor. An external device may achieve the same effect as setting the NOCK bit by driving the STOP pin low. The processor's clock is stopped until the STOP pin is released.
- The POWERC register allows unused peripherals and DMA controllers to be disabled (i.e., to have their clocks gated off) on an individual basis.

According to Lucent, the DSP16210 consumes 577 mW during typical operation at 100 MHz and 3.3 volts with the on-chip PLL enabled. In low-power standby mode with the PLL not selected and an input clock rate of 10 MHz, power consumption is 16 mW at 3.3 volts. The DSP16410 contains two cores; power consumption for one of the two on-chip cores and half of the on-chip memory is roughly 280 mW at 1.8 volts and 170 MHz, according to Lucent.

Benchmark Performance

The DSP164xx has been benchmarked with the BDTI Benchmarks™. Overall benchmark results for all benchmarked processors are presented in Chapter 8, *BDTI Benchmark™ Results*. We summarize and analyze the DSP164xx benchmark performance in the paragraphs below. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not reflect the processor's instruction cycle rate; therefore, lower instruction cycle counts imply a more powerful architecture, but do not imply faster speed. Next we discuss benchmark execution times and cost-execution time products, indicating processor speed and cost-performance, respectively. We then discuss the processor's energy consumption, which reflects the energy consumed by the processor in order to perform a task. Finally, we discuss the processor's memory usage. We divide the memory usage discussion into three parts: Control benchmark memory usage, overall benchmark program memory usage, and benchmark data memory usage.

> *Note that instead of the entire DSP16xxx family, the DSP164xx is used in this analysis. As discussed earlier, the cores used in the DSP164xx family members support several instructions that are not supported by the core used in the DSP162xx family members. Two of the twelve benchmarks (the Viterbi and Vector Maximum) use these instructions, and hence results for these benchmarks do not apply to DSP162xx family members.*
>
> *The DSP164xx family members contain two DSP16000 cores. All benchmark results presented below are for one of the two on-chip cores.*

Execution Performance

- **Instruction cycle counts:** As illustrated in Figure 8.1-13 in Chapter 8, *BDTI Benchmark™ Results*, the DSP164xx has a total normalized cycle count that is approximately 15% lower than the average for all processors benchmarked in this report. The DSP164xx has low cycle counts primarily because of its powerful data path with two multipliers; in some cases the DSP164xx's data path essentially allows it to perform twice as many operations simultaneously as can be performed on the conventional DSP processors. On most benchmarks the DSP164xx has lower than average cycle counts.

  On the **Single-Sample FIR** and **Vector Dot Product** benchmarks, the DSP164xx performs two MACs per instruction cycle and requires fewer cycles for initialization compared to most processors. As a result, on the Single-Sample FIR benchmark, the DSP164xx instruction cycle count is roughly 30% below average, and is the second-lowest of the processors benchmarked. Only the SC140 has a lower cycle count on this benchmark. Similarly, on the Vector Dot Product benchmark, the DSP164xx performs two multiply-accumulate operations per instruction cycle, and has the third-lowest cycle count among benchmarked processors, about 40% below the average; only the StarCore SC140 and the TMS320C64xx-C have lower cycle counts.

On the **Vector Add** benchmark, the DSP164xx cycle count is roughly 25% below the average of the processors benchmarked. The DSP164xx, with its support for SIMD-style additions, performs four operand loads, two additions, and two stores in two instruction cycles, which enables the calculation of one output element per cycle. Also, the DSP164xx requires relatively few instructions to perform pointer initialization outside of the inner loop on this benchmark.

On the **Vector Maximum** benchmark, the DSP164xx uses its trace-back encoder to update one counter, make dual comparisons between two operands (using specialized dual compare instructions), and record which operands were largest—all in a single instruction cycle. In addition, two new operands can be loaded in parallel. Since these steps are exactly what are required in the Vector Maximum benchmark, the DSP164xx can process elements at the rate of two elements per instruction cycle. This results in the second lowest cycle count for the Vector Maximum benchmark with a cycle count approximately 60% lower than the average for all benchmarked processors.

On the **FFT** benchmark, the DSP164xx can implement the FFT butterfly using only four instructions. This is an efficient implementation of the butterfly in comparison to most other DSP processors. However, the DSP164xx cycle count result is about 10% above the average, due to the many cycles consumed by control code (data moves, counter, and pointer updates), the lack of support for nestable hardware loops, and because one stall occurs in the butterfly.

- **Execution times:** The DSP16410 core instruction cycle rate of 170 MHz is relatively high. This cycle rate, combined with the processor's low cycle counts, results in a total normalized execution time result that is approximately twice as fast as the average of the fixed-point processors, but nearly six times slower than that of the processor with the fastest (projected) execution time, the Texas Instruments TMS320C64xx-C. At 170 MHz, one DSP16410 core has a BDTImark2000 score of 810.
- **Cost-execution time product:** Since the DSP16410 is a dual-core device, the cost-execution time product information for this processor is not comparable with those of the other processors benchmarked in this report. Hence, this metric is excluded from our analysis.
- **Energy consumption:** Energy consumption reported here is for one of the two cores in the DSP16410 and half of the on-chip memory. Although the power consumption of each DSP16410 core is relatively high compared to other fixed-point devices, its fast execution times result in a total normalized energy consumption figure that is only slightly higher than the average of all fixed-point processors benchmarked, as shown in Figure 8.4-13. However, compared to the Motorola

MSC8101, which has the best result for this metric, the DSP16410 has a normalized energy consumption that is roughly five times higher.

Memory Usage

The focus in the memory usage analysis is on Control benchmark memory usage. Unlike other benchmarks, the Control benchmark is optimized for minimum memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP code. Finally we discuss constant and non-constant data memory usage.

- **Control benchmark memory usage:** On the Control benchmark, the DSP164xx has a total memory usage that is roughly average for the fixed-point processors and 15% lower than the average for all processors that have been benchmarked. The DSP164xx uses a mixture of 16-bit and 32-bit instructions. The processor's total memory usage result on the Control benchmark suggests that the DSP164xx can achieve reasonable code density by using mostly 16-bit instructions in control-oriented tasks while reserving its more powerful 32-bit instructions for speed-critical tasks. In the Control benchmark, the DSP164xx uses 16-bit instructions for roughly 70% of the program.
- **Program memory usage:** The DSP164xx has a total normalized program memory usage that is roughly 45% below the average of all processors benchmarked, as illustrated in Figure 8.5-13. However, compared to all fixed-point processors benchmarked except the Texas Instruments TMS320C62xx and TMS320C64xx (which have exceptionally high program memory usage), the DSP164xx has a total normalized program memory usage figure that is slightly above the average. This is due to the DSP164xx using mostly 32-bit instructions for DSP algorithm code (as opposed to control-oriented code, discussed above). Although the 32-bit instructions tend to increase program memory usage, the high per-cycle efficiency of the DSP164xx allows it to use relatively few instructions to perform a given task.
- **Data memory usage:** With the exception of memory usage on the FFT, the DSP164xx's constant and non-constant data memory usage is generally as expected for a 16-bit processor, as illustrated in Figure 8.5-14 and Figure 8.5-15. For the FFT benchmark, the DSP164xx uses an in-place implementation, which requires less non-constant data memory than is required by many of the other processors, but more constant data memory (for look-up tables).

The DSP16410 has a relatively high instruction cycle rate, and this, combined with its high per-cycle efficiency (a result of its powerful data path) gives it good execution-time performance.

The energy consumption of one DSP16410 core is reasonably good, but not comparable to that of the most energy-efficient processors.

Although the DSP164xx instruction set includes both 16- and 32-bit instructions, in control-oriented software, the DSP164xx will mainly use 16-bit instructions and thus achieve code density that is comparable to that of other DSPs that use exclusively 16-bit instruction words.

Cost

DSP16xxx price and packaging options are shown in Table 7.6-5.

Fabrication Details

According to Lucent Technologies, the DSP16210 and the DSP16410A are fabricated in a 0.25 µm CMOS process and use a 2.5-volt core supply voltage. The DSP16410B is fabricated in a 0.18 µm CMOS process and uses a 1.8-volt supply voltage.

Development Tools

Lucent Technologies' DSP16xxx tools include an ANSI-C optimizing compiler based on the GNU GCC compiler, an assembler, a linker, and an instruction-set simulator and debugger. These tools are currently available on Sun SPARC workstations under SunOS and Solaris, on Hewlett-Packard workstations under HP-UX, and on IBM PC-compatible computers under Microsoft Windows 9x and Windows NT.

The software generation tools are easy to use and support a useful range of options for optimization and debugging support.

Part	Core Voltage (V)	Speed per Core (MHz)	Package	Price (Qty. 10,000)
DSP16210	2.5	100	144 TQFP	$40.00
DSP16210	2.5	120	144 TQFP	$45.00
DSP16210	2.5	150	144 TQFP	$55.00
DSP16210	2.5	100	169 BGA	$42.00
DSP16210	2.5	120	169 BGA	$47.00
DSP16210	2.5	150	169 BGA	$57.00
DSP16410B	1.8	170	208 PBGA	$90.00

TABLE 7.6-5. DSP16xxx price and package summary. Prices as of August, 2000.

The instruction-set simulator and debugger supports both graphical and command-line interfaces. The DSP16210 simulator is not cycle-accurate but the DSP16410 simulator is. The debugger has two modes: software simulation mode and hardware mode. In hardware mode, the debugger is connected to a physical DSP16xxx using the JTAG Communications System (JCS). The JCS consists of an ISA plug-in card for IBM PC-compatible computers and a target interface box. The JCS allows JTAG-based debugging of the target system, as well as programming support for flash-based devices. Special software allows multiple UNIX or Windows NT workstations to access one or more remotely located DSP16xxx devices connected to a JCS over a TCP/IP network.

The simulator and debugger is powerful and in addition to typical debugging features supports:

- Automation of debugging tasks using predefined debugger command scripts
- A wide range of data display formats, including signed/unsigned decimal, fractional integer, floating-point, octal, signed/unsigned binary, and hexadecimal
- Source debugging with mixed disassembly, which displays disassembled instructions beneath each original C or assembly language source file line
- User-defined variables and functions, which can be used for constructing macros or functions for performing repetitive tasks using simple debugger command-line constructs
- Standard C language compatible input/output operations
- File input and output, allowing data streams to be connected to simulated peripherals

In addition to a powerful command language, the debugger provides support for commands using the Tcl/Tk interpreted language. Commands entered at the debugger command line can be passed directly to the Tcl/Tk interpreter, allowing the user to define complex and powerful customized debugging tools.

> *The debugger is very powerful and executes simulated DSP16xxx instructions rapidly. The debugger is unusual in providing comprehensive support for Tcl/Tk, which is a complete interpreted language. However, the debugger may present a significant learning curve for users migrating from other vendors' debugging tools, in part because of its support for Tcl/Tk.*
>
> *While the DSP16xxx debugger is very powerful, it lacks some useful debugging features compared to other vendors' tools. For example, both Analog Devices and Texas Instruments include useful time- and frequency-domain data graphing capabilities in their debugging products.*

The graphical debugger interface supports a feature called "Architectural View," a pictorial representation of the device, which highlights dataflow through the device registers updated on a cycle-by-cycle basis. The view can be zoomed, panned, and printed.

The instruction-set simulator is also available in a C-callable format that Lucent calls the "Linkable Functional Simulator" (LFS). According to Lucent Technologies, the LFS is integrated as a function block in two high-level simulation environments: Synopsys' COSSAP and Cadence Design Systems' SPW. This integration allows DSP16xxx simulations to be incorporated into larger, system-level simulations.

Applications Support

The primary documents for the DSP16210 are the *DSP16000 Digital Signal Processor Core*, the *DSP16000 Digital Signal Processor Core Instruction Set* and the *DSP16210 Digital Signal Processor Data Sheet.* The documents available for the DSP16410 are the data addendum *Enhancement to the DSP16000 Core* and the *DSP16410 Digital Signal Processor Data Sheet.* These cover most aspects of the devices, including timing, electrical, and mechanical characteristics.

Due to the complexity of the instruction set, the instruction set manual uses more than 40 different symbols to represent register subsets. These symbols are used to allow more compact representations of instruction syntax.

> *The multitude of symbols used in the instruction set manual makes it difficult to decipher for a new user. However, it allows the many instructions supported on the DSP16xxx to be described in a very compact way, which can be useful when programming the DSP16xxx.*

Lucent provides applications support for the DSP16xxx through telephone and electronic mail contact with their application engineers. In addition, Lucent provides documentation, application notes, and product information on their *Digital Signal Processors* World Wide Web pages at *http://www.lucent.com/micro/dsp.*

Advantages

- Two independent multipliers, allows two MACs per cycle
- SIMD ALU can add or subtract two sets of operands
- Separate three-input adder
- Good support for operand-unrelated parallel moves
- Conditional execution of many instructions
- Algebraic assembly language
- Many accumulators
- Barrel shifter

- Powerful bit manipulation unit; includes bit-field extract and replace
- Good support for multi-precision arithmetic
- Single-cycle exponent detection, and normalization instruction
- Many address registers
- Large, unified address space
- Good internal memory bandwidth (if data arranged as pairs in memory) when instructions are executed from cache.
- Short interrupt latency
- Two serial ports, one of which supports TDM on the DSP16210, and both of which support TDM on the 16410
- Flexible power management
- Flexible bootstrap modes
- Software stack with frame pointer and indexed addressing supports HLL compilers
- Can use 1X clock, has flexible PLL
- JTAG emulation port with boundary scan, and good internal debugging hardware
- Powerful DMA controller on the DSP16410

Disadvantages

- High cost (range between $40.00 and $90.00 for the different chips available, quantity 10,000)
- Non-orthogonal instruction set
- Multiply operands restricted
- No bit-reversed addressing
- Many restrictions on circular buffering
- Hardware loops limited to 31 instruction words, not nestable
- Requires execution from cache for maximum performance
- Relatively high typical power consumption

7.7 Motorola DSP563xx Family

BDTImark2000 Score:
450 at 150 MHz

Introduction

The DSP563xx family is Motorola's second generation of 24-bit fixed-point DSP processors. The DSP563xx architecture is similar to (and object-code compatible with) its predecessor, the DSP560xx family, but includes significant modifications designed to increase performance. For example, the introduction of a deeper, seven-stage pipeline allows the DSP563xx to complete an instruction cycle with each master clock cycle, compared with the two master clock cycles required by the DSP560xx family. In addition, an enhanced instruction set and new features, such as a barrel shifter, instruction cache, and DMA controller, boost DSP563xx family efficiency. The DSP563xx also allows 24-bit program and data memory addressing for large applications, a significant improvement over the 16-bit addressing found in the DSP560xx. The fastest DSP563xx family members run at 150 MHz at 1.8 volts for the core and 3.3 volts for I/O. Motorola is targeting the DSP563xx at applications with demanding computational requirements, such as cellular base stations, advanced audio products, and videoconferencing.

The DSP56301, the first member of the family, was introduced in 1995 with 66 MHz, 80 MHz, and 100 MHz versions running at 3.3 volts. The DSP56301 features a PCI bus interface and is intended for PC applications that require PCI support. For applications that don't need a PCI bus interface, Motorola offers the lower-priced DSP56303. For prototyping and applications requiring large amounts of on-chip RAM, Motorola offers the DSP56309, which features large RAM memory banks. In 1998, Motorola introduced the 100 MHz DSP56307, running at a core voltage of 2.5 volts, to support wireless infrastructure applications. The DSP56307 includes a filter co-processor to off-load general filtering operations from the DSP core. Simultaneously, Motorola also extended the family with processors aimed at consumer audio applications. In 1998, Motorola introduced the 3.3 volt, 100 MHz DSP56362 followed by the lower-cost DSP56364 in 1999, and a 120 MHz version of the DSP56366 in 2000. The DSP56362 and DSP56366 are available either as general-purpose audio processors or factory programmed with digital audio decoders for DTS, Dolby AC-3, and MPEG2. Contact Motorola for more information. The fastest current family member is the DSP56311, which combines a 150 MHz core running at 1.8 volts with a filter co-processor and large RAM banks. The DSP56311 is intended for use in multi-channel voice and data applications, and was introduced in 1999. Motorola DSP563xx processors are summarized in Table 7.7-1.

In 1996, Motorola announced the DSP566xx family. DSP566xx processors feature nearly the same instruction set as DSP563xx processors, but with a 16-bit data path. Motorola is no longer actively marketing off-the-shelf DSP566xx chips; contact BDTI directly for analysis of the DSP566xx.

Architecture

The DSP563xx architecture consists of a 24-bit fixed-point data path, two address generators, and a program control unit. The architecture of the DSP56301 is shown in Figure 7.7-1.

Data Path

The DSP563xx features an integrated MAC/ALU with a 24 × 24 → 48-bit multiplier, a 56-bit ALU, and two 56-bit accumulators, providing eight guard bits. Multiplier inputs come from two of four input registers (X0, X1,Y0,Y1) or one input register and an immediate operand. MAC/ALU outputs are placed in one of two 56-bit accumulators. The multiplier can perform both signed and unsigned multiplies, and supports multi-cycle dou-

Part	Maximum Speed (MHz)	Voltage (V)	Peripherals
DSP56301	100	3.3	DMA, GPIO, two synchronous serial ports, asynchronous serial port, three timers, 32-bit PCI host port
DSP56303	100	3.3	DMA, GPIO, two synchronous serial ports, asynchronous serial port, three timers, 8-bit host port
DSP56307	100	2.5/3.3	DMA, GPIO, two synchronous serial ports, asynchronous serial port, three timers, 8-bit host port, filter co-processor
DSP56309	100	3.3	DMA, GPIO, two synchronous serial ports, asynchronous serial port, three timers, 8-bit host port
DSP56311	150	1.8/3.3	DMA, GPIO, two synchronous serial ports, asynchronous serial port, three timers, 8-bit host port, filter co-processor
DSP56362	120	3.3	DMA, GPIO, enhanced serial audio interface, serial host interface, digital audio transmitter, three timers, 8-bit host port
DSP56364	100	3.3	DMA, GPIO, enhanced serial audio interface, serial host interface, general-purpose I/O interface
DSP56366	120	3.3	DMA, GPIO, two enhanced serial audio interfaces, serial host interface, digital audio transmitter, three timers, 8-bit host port

TABLE 7.7-1. DSP563xx processor feature summary.

ble-precision multiplies. All arithmetic operations, including multiply-accumulate, execute in two instruction cycles and are pipelined so that a new instruction can be started during every instruction cycle. Depending on the instruction, ALU inputs can come from either the four input registers, the two accumulators, or immediate operands. Status bits, such as a zero bit, overflow bit, and negative bit, provide information about the results of all arithmetic operations.

The DSP563xx data path uses fractional arithmetic in all operations. Because the DSP563xx does not have an integer multiply instruction, programmers must convert the result of a fractional multiply to integer format by shifting a sign bit into the accumulator

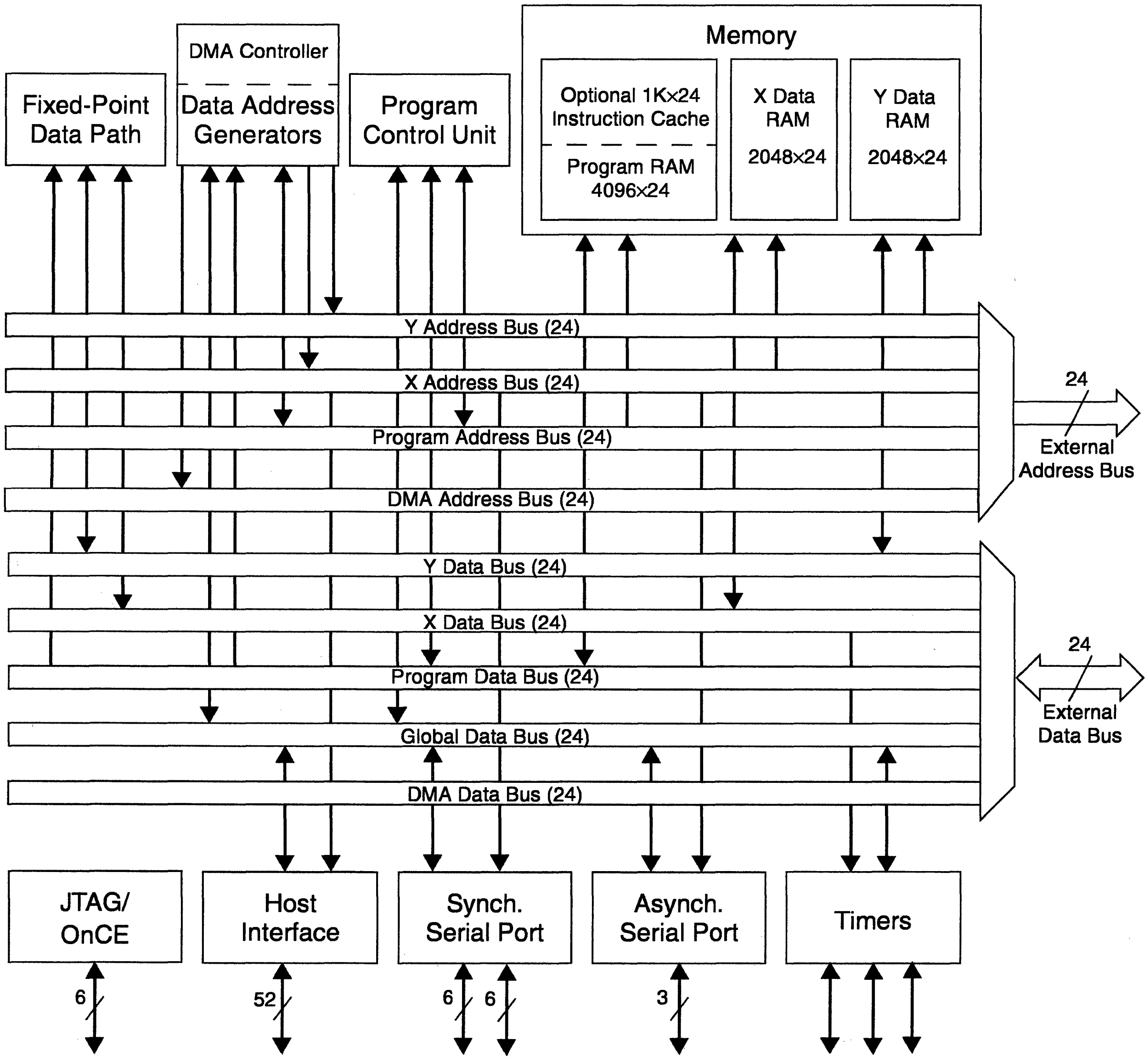


FIGURE 7.7-1. Motorola DSP56301 architecture.

MSB (by executing an arithmetic shift right instruction after the multiply) to perform an integer multiply.

Logic operations, as well as some other operations, are performed only on bits 24 through 47 of a 56-bit accumulator word. In contrast, some processors perform logic operations on the entire accumulator.

> *The fact that some operations are limited to 24-bit operands can limit performance in some applications that require performing logic operations on the entire accumulator for maximum efficiency.*

Conventional or convergent rounding can be optionally applied to output values of the MAC/ALU unit before they are stored in an accumulator. Convergent rounding, selected by a bit in a mode register, reduces bias caused by conventional rounding techniques.

Values stored in the 56-bit accumulators can be transferred to a 24-bit location (e.g., registers or memory) through the shifter/limiter. The shifter allows the value in the accumulator to be optionally shifted one bit left or right for scaling purposes as specified by bits in a mode register. The limiter optionally checks to see if the resulting value fits in 24 bits without overflow and, if not, substitutes a limited data value of the same sign and maximum allowable magnitude. Thus, values less than -1.0 are limited to -1.0 and values greater than or equal to 1.0 are limited to $1.0\text{-}2^{-23}$, or approximately 1.0.

The data path provides support for multi-precision arithmetic with specialized double-precision multiply instructions. Other multi-precision arithmetic operations can be performed using the add-with-carry and subtract-with-borrow instructions.

A barrel shifter and bit manipulation unit allow multi-bit shifting and bit-field replace (Motorola calls this insert) and extract operations. Multi-bit shifting may be either logical or arithmetic. The DSP563xx also features an exponent detect operation, which can be used with the barrel shifter to perform normalization using two instructions. The bit-field-replace operation allows a contiguous field of up to 24 bits from a source accumulator or input register to replace the selected bits of the destination accumulator. Similarly, bit-field-extract copies a bit field up to 24-bits long from a given position in the source accumulator to the destination accumulator. The extracted bit-field may be optionally sign extended. Additionally, the bit manipulation unit supports a merge instruction, a special case of the insert instruction that concatenates 12-bit half-words. DSP563xx shifting and ALU operations make use of the carry bit where appropriate.

> *The DSP563xx barrel shifter and bit manipulation unit is especially flexible and is useful for the protocol processing found in telecommunications standards and other algorithms.*

The DSP563xx features a 16-bit arithmetic mode invoked by a bit in the status register. In this mode, 16-bit data is stored in 24-bit words and moved across 24-bit data buses, but the ALU disregards the most-significant byte. Each accumulator is treated as if it were 40 bits wide and composed of a 16-bit least-significant portion (A0), a 16-bit most-significant portion (A1), and an 8-bit extension register. Instructions that depend on the width of the data path, such as the half-word merge instruction, are adjusted appropriately for 16-bit arithmetic mode (i.e., the merge instruction uses 8-bit half-words instead of 12-bit half-words).

> *The 16-bit arithmetic mode is useful in applications that require bit-exact implementations of 16-bit standard algorithms, such as ITU-T G.728. The DSP563xx is an especially good choice for applications that require 16-bit support for some functionality, but need the extra resolution of 24 bits for other functions.*
>
> *DSP563xx code written in 16-bit arithmetic mode runs without modification on Motorola's DSP566xx processor. The DSP566xx includes the same 24-bit instruction set as the DSP563xx, but has a true 16-bit data path.*
>
> *The 16-bit arithmetic mode should not be confused with the DSP563xx 16-bit compatibility mode. The former limits the DSP563xx data path resolution to 16 bits. The latter limits the width of the DSP563xx address generation registers and address buses to 16 bits to preserve backward compatibility with DSP560xx code, which assumes a 16-bit address space. The two modes may be selected independently. The 16-bit compatibility mode is discussed in the* Instruction Set *section.*

Memory System

The DSP563xx architecture divides memory into three spaces: program space (P), X data space, and Y data space. Each memory space has a separate 24-bit on-chip address bus and 24-bit on-chip data bus, as shown in Figure 7.7-1. Additionally, each memory space is associated with one or more dedicated banks of on-chip ROM or RAM.

The three memory spaces with their dedicated data and address buses allow DSP563xx processors to make three memory accesses per instruction cycle without the use of a cache. The on-chip program memory permits one instruction fetch per instruction cycle, while the on-chip data memories permit up to two reads, two writes, or one write and one read per instruction cycle. Thus, when the processor is executing at 100 MHz, the maximum sustainable on-chip data memory bandwidth is 200 24-bit Mwords/second. This memory bandwidth, coupled with the processor's instruction set, also allows single-cycle double-precision (i.e., 48-bit) transfers between memory and accumulators, facilitating extended-precision arithmetic.

As mentioned previously, the on-chip data memory is accessible from two independent buses. One of these buses is owned by the processor and the other by the DMA controller. The memory is divided into partitions of 1024 words on the DSP56311 and 256 words on the other DSP563xx processors. Although in general the data memory can support two operations simultaneously, each partition can only support one operation. Hence, the data memory can be accessed by DMA in parallel with accesses by the processor core as long as the locations addressed do not lie in the same partition.

All DSP563xx processors feature the ability to configure a portion of on-chip program RAM as a 1Kx24 eight-sector least-recently-used instruction cache (a 2Kx24 cache is also available as a factory mask-programmable option). When the cache is enabled by setting a bit in the DSP563xx operating mode register, the processor allocates a 1 Kword contiguous block of program memory for the exclusive use of the cache. Each of the eight 128-word cache sectors corresponds to a contiguous region of external memory 128 words long. Software instructions can load and lock each sector to obtain deterministic execution for real-time applications. When the cache's burst mode is selected, the cache automatically reads up to four neighboring memory locations each time an in-cache memory location is selected. The additional memory reads add wait states.

> *The DSP563xx cache burst mode is useful for applications that use DRAM. When using an external DRAM for both program and data memory, an instruction may generate a cache miss (i.e., program fetch) from one page in external DRAM, followed by a data move from a different DRAM page. Since programmers often store program instructions and data values in different DRAM pages, this scenario is common. Because the access time to DRAM for an in-page access is typically much shorter than out-of-page access, the above sequence of out-of-page accesses reduces memory bandwidth. Programmers can overcome this problem by using the DSP563xx cache burst mode. In burst mode, the cache fetches up to four consecutive program instructions while postponing the data transfers. Therefore, one out-of-page access is followed by up to three in-page accesses. This scheme minimizes out-of-page DRAM accesses in many situations.*
>
> *Motorola claims cache hit rates over 90% when running a full-rate GSM vocoder from external memory, even when up to four of the cache sectors are disabled. However, as explained in the external memory section, a minimum one-cycle delay is incurred when code stored in external memory is loaded into the cache on the first execution. Programmers should be cautious when relying on cache performance for critical real-time routines. In some applications, programmers may choose to disable the instruction cache and use DMA (discussed in the peripheral section) to pre-load the program memory used by the cache. Because DMA operates without the core's intervention, pre-loading program memory requires little overheard, but programmers must know which sections of program code to load.*

The instruction cache may be useful in reducing system cost. Programmers can select a DSP563xx family member with just enough on-chip RAM to hold critical real-time code. The cache can then be used to speed execution of non-real-time code, such as control functions, which can be placed in relatively inexpensive external DRAM. However, relying on the cache for execution speed can make it difficult to use some of the less expensive family members, since they have little on-chip program RAM from which to subtract a cache. In this case it is a significant disadvantage that the instruction cache is created by sacrificing a significant portion of the available on-chip program RAM.

Effectively using the instruction cache requires careful application profiling to determine when and how frequently code is executed. Motorola provides application profiling capabilities in its development tools. The cache is also modeled in the DSP563xx simulator.

DSP563xx memory configurations are shown in Table 7.7-2.

External Memory Interface

The DSP563xx external memory interface, called an "expansion port" by Motorola, provides one 24-bit data bus and one 24-bit address bus. The internal buses are multiplexed onto these external buses. Read, write, and four independent program/data strobes are provided, enabling glueless interface with up to four types of external memory and peripherals at a time. Synchronous static memory (SSRAM), static memory (SRAM), and dynamic memory (DRAM) are supported. The external memory interface supports page-mode DRAM and includes an internal refresh generator.

The DSP563xx external memory interface is especially flexible. The glueless interface to many memory types and DRAM refresh generator may reduce system cost and lower component count in some applications.

A memory space (X, Y, or program) may be split across two or more of the program/data strobes to allow mapping the space to multiple memory types. Using zero-wait-state synchronous static memory, the processor can perform one external memory data move per instruction cycle yielding a maximum off-chip data memory bandwidth of 100 24-bit Mwords/sec for a 100 MHz DSP563xx. Pipeline effects delay instruction fetches from external memory by one cycle, unless the instruction is already loaded into the on-chip instruction cache. Thus, the processor can fetch one instruction from zero-wait-state external synchronous static memory every two instruction cycles.

The pipeline stall incurred when accessing external program memory is a disadvantage. In some applications, this problem can be mitigated by effective use of the instruction cache or of overlays using DMA.

The DSP563xx supports programmed and externally requested wait states. In the programmed case, each memory space strobe can be independently configured for 0 to 31 wait states; each wait state is one master clock cycle long. In the externally requested case, the processor delays external memory accesses if an external bus wait pin is asserted.

The external memory interface provides handshake lines useful in creating shared-memory multiprocessor systems. Bus request, bus grant, bus busy, bus lock, and memory select pins allow external devices to share the processor's external memory.

Processor	X Memory		Y Memory		Program Memory	
	ROM	RAM	ROM	RAM	ROM	RAM
DSP56301[1]	—	2.0 K	—	2.0 K	—	4.0 K
		3.0 K		3.0 K		2.0 K
DSP56303[1]	—	2.0 K	—	2.0 K	—	4.0 K
		3.0 K		3.0 K		2.0 K
DSP56307[1]	—	8.0 K	—	8.0 K	—	48.0 K
		24.0 K		24.0 K		16.0 K
DSP56309[1]	—	5.0 K	—	5.0 K	—	24.0 K
		7.0 K		7.0 K		20.0 K
DSP56311[1]	—	16.0 K	—	16.0 K	—	96.0 K
		48.0 K		48.0 K		32.0 K
DSP56362[1]	6.0 K	5.5 K	6.0 K	5.5 K	30.0 K	3.0 K
		5.5 K		3.5 K		5.0 K
DSP56364	—	1.0 K	—	1.5 K	8.0 K	0.5 K
		1.0 K		0.5 K		1.5 K
DSP56366	32.0K	13.0 K	8.0K	7.0 K	40.0 K	3.0 K
		8.0 K		5.0 K		10.0 K

TABLE 7.7-2. DSP563xx processor memory summary. All sizes are in 24-bit words and are rounded to the nearest 0.5 K. All family members support multiple memory configurations, selectable by writing to mode registers in software. Instruction caches are assumed to be disabled; enabled caches consume 1 Kword of program RAM.

[1] Other intermediate configurations also possible

The DSP563xx provides atomic bit test-and-set, test-and-clear, test, and test-and-toggle operations. In addition to bit manipulation functions, these instructions can be used to provide resource locking in multiprocessor systems.

Address Generation Units

The DSP563xx supports immediate data and register-direct, memory-direct, and register-indirect addressing modes. The processor also supports short-address versions of some modes for certain instructions.

For register-indirect addressing, DSP563xx processors provide two address generation units. Addresses generated by these units can be used to access X, Y, and P memory. Base addresses come from any of eight 24-bit address registers, R0-R7, although instructions requiring simultaneous access to both X and Y memory must use one address register from R0-R3 and the other from R4-R7.

Each address register has an associated 24-bit modifier register (N0-N7) that holds an offset for indexed addressing or an increment value for post-modification of an address register. Some move instructions support indexed register-indirect addressing with immediate short and immediate long indexing. Additionally, each address register has an associated 24-bit modulo buffer length register (M0-M7) that can be used for modulo and bit-reversed addressing (Motorola calls these registers modifier registers). Modulo buffers may be between 2 and 32768 words long, but the base address must be a multiple of 2^k, where k is a positive integer, and the length of the buffer, M, is less than or equal to 2^k.

Supported address register update modes for indirect addressing are post-increment and post-decrement by one, post-increment or post-decrement by the associated modifier register, and pre-decrement by one. The pre-decrement-by-one mode and the indexed addressing mode require an additional instruction cycle.

Pipeline

DSP563xx processors use a seven-stage pipeline divided into fetch 1, fetch 2, decode, address generation 1, address generation 2, execute 1, and execute 2 stages. The processor uses interlocking to resolve pipeline conflicts. Because interlocks can slow code execution, programmers should avoid them whenever possible. Several conditions can cause interlocks. The most significant are:

- If an accumulator is the destination operand of an ALU operation, then a conflict occurs if the accumulator is the source operand of a move operation in the following instruction cycle. The processor will delay the following instruction by one instruction cycle to resolve this conflict. For example, the second instruction in the example below will be delayed for one instruction cycle.

```
MAC   X0,X0,A
MOVE  A,X:(R0)+
```

When writing compact code, programmers will have difficulty avoiding all instances of the above interlock condition. Program-

> *mers should budget for some pipeline stalls when estimating cycle count requirements. The assembler flags pipeline stalls as an aid to programmers. The profiler also provides statistics on which routines have the most stalls so that programmers can select code sections to optimize.*

- Updates to address generation register sets (the address register, modifier register, and modulo buffer length register) take effect on the fourth instruction cycle following the update. If the updated register set is accessed before the fourth cycle, the processor delays the offending instruction one to three instruction cycles to resolve the conflict. For example, the third instruction in the example below will be delayed for two instruction cycles.

```
MOVE  #$ADDR,R0
CLR   A
MOVE X:(R0)+,Y1
```

Most DSP563xx ALU/MAC instructions are pipelined to allow a new instruction to begin on each instruction cycle. Instructions that use a second 24-bit instruction word to hold an address or immediate data and instructions that change program flow require multiple instruction cycles. In particular:

- Single and multi-instruction repeat loops (REP and DO) require five instruction cycles.
- Branches and other change-of-flow instructions require three to five instruction cycles.
- All two-word instructions require an additional instruction cycle for the program controller to fetch the second instruction word.

Instruction Set

DSP563xx registers and instruction set are summarized in Tables 7.7-3 and 7.7-4, respectively. The DSP563xx instruction set is a superset of the DSP560xx instruction set. With the exception of processor-specific I/O or peripheral control code, the DSP563xx is object-code compatible with DSP560xx code when the 16-bit compatibility mode is enabled by setting a bit in a mode register. In 16-bit compatibility mode, move operations to and from the address generation registers or the program control registers clear the eight most-significant bits of the destination. Since the DSP560xx family only supports 16-bit address generation and program control registers, clearing the eight most-significant bits ensures that DSP560xx object code will run correctly.

> *Sixteen-bit compatibility mode should not be confused with 16-bit arithmetic mode. See the* Data Path *section for an explanation of 16-bit arithmetic mode.*
>
> *Although DSP560xx code must be re-optimized to take full advantage of the DSP563xx's performance, the ability to run unaltered*

DSP560xx object code on the DSP563xx provides an easy software migration path.

With the exception of a few differences in instruction sets and instruction execution restrictions, DSP563xx object code also runs on the DSP566xx, the 16-bit version of the DSP563xx.

Typically, unmodified DSP560xx code requires more instruction cycles when running on the DSP563xx than when running on the DSP560xx because of pipeline stalls and the higher instruction cycle counts of change-of-flow instructions on the DSP563xx. Optimizing DSP560xx code by adding new DSP563xx instructions and minimizing pipeline interlocks can usually lower the instruction cycle count. Of course, the instruction execution rate of the DSP563xx is twice that of the DSP560xx operating at the same master clock rate.

Motorola asserts that the DSP563xx offers more than twice the performance of the DSP560xx at the same clock speed. This claim is based on the doubled instruction execution rate, the addition of new instructions, and the addition of DMA. While the DSP563xx well outperforms the DSP560xx, the claim of more than 100% improvement in execution time (when comparing processors rated for the same clock speed) ignores the inevitable complications of a deep pipeline, such as increased cycle counts on change-of-flow instructions, and is overstated. Only code that makes extensive use of new DSP563xx instructions, such as multi-bit shifting instructions or DMA, gains more than 2X improvement. Typical code gains less than a 2X improvement in execution time.

Registers	Width	Purpose
A, B	56 bits[1]	Accumulators
X0, X1, Y0, Y1	24 bits[2]	General-purpose/Input registers
R0-R7	24 bits[3]	Address registers
N0-N7	24 bits[3]	Modifier registers
M0-M7	24 bits[3][4]	Modulo buffer length registers

TABLE 7.7-3. DSP563xx register summary.

[1] In 16-bit arithmetic mode, these registers are 40 bits wide.

[2] In 16-bit arithmetic mode, these registers are 16 bits wide.

[3] In 16-bit compatibility mode, these registers are 16 bits wide.

[4] Although the registers are 24 bits, only the first 16 bits are used for modulo addressing.

Assembly Language Format

DSP563xx arithmetic instructions are divided into three parts: a MAC/ALU field, an X data bus move field, and a Y data bus move field. The MAC/ALU field describes the arithmetic operation to be performed (e.g., multiply-accumulate, shift, logic operation,

Class	Instructions
Arithmetic	Add and subtract with ALU register or immediate operands, add with carry, shift left/right one bit and add, subtract with borrow, shift left/right one bit and subtract, negate, clear, increment, decrement, round, absolute value
Multiplication	Multiply, multiply-accumulate, and multiply-subtract, with or without rounding and with ALU register or immediate operands; support for double-precision and signed or unsigned operands
Logic	*And*, *or*, and *exclusive-or* with ALU register or immediate operands, *not*
Shifting	Arithmetic/logical shift left/right 0-56 bits
Rotation	Rotate left/right one bit
Conditional Execution	Conditionally transfer data between selected registers, conditionally execute ALU operation
Comparison	Compare, compare magnitude, test accumulator for zero value, signed compare, unsigned compare, transfer maximum, transfer maximum magnitude, compare with immediate value
Looping	Single- and multiple-instruction hardware loop, do forever loop, conditionally exit do loop
Branching	Conditional and unconditional branch with absolute or PC-relative addressing, branch on bit set/clear with absolute or PC-relative addressing
Subroutine Call	Conditional and unconditional call with absolute or PC-relative addressing, call on bit set/clear with absolute or PC-relative addressing
Bit Manipulation	Bit set, clear, test, toggle; bit field insert, replace, and merge
Special Function	Iterative normalization, fast normalization, division iteration, exponent detect, Viterbi shift left

TABLE 7.7-4. DSP563xx instruction set summary. New instructions (relative to the DSP560xx) are underlined.

etc.) using an opcode-operand format, while the data bus move fields describe up to two memory accesses that occur in parallel with the instruction. For example, the instruction:

```
MAC    X0,Y0,A     X:(R0)+,X0        Y:(R4)+N4,Y0
```

multiplies X0 and Y0 together and accumulates the result in the A accumulator, loads register X0 from the X memory location pointed to by register R0, loads register Y0 from the Y memory location pointed to by R4, post-increments R0 by one, and post-increments R4 by the contents of N4.

Parallel Move Support

Most DSP563xx ALU instructions support operand-unrelated parallel data moves. That is, parallel data moves may be unrelated to the arithmetic/MAC instruction being executed. The allowed parallel moves are immediate short data move, register-to-register data move, address register update, X memory data move, X memory and register data move, Y memory data move, register and Y data move, long memory data move, and X and Y data move. The X and Y data move, which may be two reads, two writes, or a read and a write, allows a new data sample and coefficient to be loaded while a filter MAC instruction is executing.

DSP563xx parallel move support is very flexible.

Orthogonality

The DSP563xx family uses a 24-bit instruction word, which provides good orthogonality. For example, in the arguments to the MAC instruction above, the programmer can choose any two registers from X0, X1, Y0, and Y1 as the source operands for the multiply, and either A or B as the destination accumulator. Similarly, in each of the parallel moves, R0 through R7 (R0 through R3 for one move, R4 through R7 for the other move) could have been used with any addressing mode to load the source operand, and either X0, X1, Y0, or Y1 could have been the destination of either move. While the processor's instruction set is not totally orthogonal, the above example illustrates the relatively high level of orthogonality.

Execution Times

Most DSP563xx ALU/MAC instructions execute in a single instruction cycle if pipeline interlocks are avoided. Most two-word instructions execute in two instruction cycles. Branch and other change-of-flow instructions execute in three to five instruction cycles.

Instruction Set Highlights

The DSP563xx provides special instructions for the following operations:

- The Viterbi shift left instruction saves the most-significant 16 bits of an accumulator to a specified address in X memory, shifts the least-significant 16 bits one bit to the left, inserts a zero or one into the least-significant bit according to an operand,

and saves the shifted lower 16 bits to the same specified address in Y memory. This instruction greatly increases the speed of Viterbi decoding.

- Atomic bit test-and-set, test-and-clear, test, and test-and-toggle. In addition to bit-manipulation functions, these instructions can be used to provide resource locking in multiprocessor systems.
- Variable width (up to 24 bits) bit-field extract and replace; merge two 12-bit half-words.
- Conditional arithmetic/logic instruction execution.
- Magnitude compare.
- Maximum and maximum magnitude transfer. These instructions compare the values in the accumulators and transfer the value in one accumulator to the other accumulator if it is greater (or has a greater magnitude in the case of the maximum magnitude transfer).
- Division iteration.
- Two-instruction normalization, achieved by following an exponent detect instruction with a multi-bit normalization function.

Execution Control

Clocking

The internal master clock on all DSP563xx processors runs at the same speed as the processor's instruction execution rate. All DSP563xx processors feature an on-chip phase-locked loop (PLL) frequency synthesizer for clock generation. The PLL allows the processor's master clock to be generated from an external source. The PLL includes a programmable pre-divider, clock multiplier, and divider. The divider may be modified without losing PLL lock. Thus, to reduce power consumption, programmers may lower the clock speed via software when the processor is not being used intensively.

Hardware Looping

The DSP563xx provides two instructions for hardware loops: REP and DO. The former repeats a single one-word instruction, while the latter repeats a block of instructions. In both cases, loops can be repeated from 0 to 65,535 times. Additionally, the DO FOREVER instruction repeats code fragments indefinitely. The DO loop can repeat a code fragment as large as program memory.

DO loops are nestable. Each DO loop uses two words of stack space, so if the stack is otherwise empty, seven DO loops can be active simultaneously. More than seven DO loops may be nested by enabling the optional stack extension discussed below. REP loops may be nested within a DO loop, but REP loops cannot be nested within another REP loop.

REP loops are not interruptible, but DO loops are.

REP and DO instructions require five instruction cycles for setup. In applications with many nested blocks of code, this setup time becomes part of the inner loops and can slow down the application.

Interrupts

All DSP563xx processors have external reset and non-maskable interrupt lines. The DSP563xx provides five general-purpose external interrupt lines. The programmer can configure each general-purpose interrupt pin as either edge- or level-sensitive.

Interrupts generated by DSP563xx on-chip peripherals include interrupts associated with the host interface, enhanced asynchronous serial interface, serial communications interface, DMA controller, and timers.

The four internally generated interrupts are stack error, debug event (discussed in the *On-Chip Debugging Support* section), software interrupt, and illegal instruction trap.

Each interrupt source has its own interrupt vector. A vector base address register can be used by the programmer to place the interrupt vectors anywhere in program memory. Interrupts are prioritized and automatically nestable. The processor supports four interrupt priority levels, 0 through 3. The processor's interrupt priority level can be set to mask out interrupts at or below levels 0, 1, or 2. Level 3 interrupts, which include all of the internal interrupt sources as well as RESET and NMI, are non-maskable. The programmer can assign priorities to each class of interrupts (e.g., all host interface interrupts) on a class-by-class basis via an interrupt priority register.

DSP563xx processors respond to interrupts by fetching two instructions at the appropriate interrupt vector (a vector base address register maps the interrupt vector table to a specified location in program memory). If neither of these instructions is a jump-to-subroutine (JSR) instruction, the interrupt is classified as "fast." When a fast interrupt is processed, the instructions at the interrupt vector location are inserted into the processor's execution pipeline without any saving of context. Fast interrupt routines typically are used to move one or two data words between a peripheral and memory.

Fast interrupts can significantly speed execution time by eliminating the overhead needed for normal interrupt processing. However, use of fast interrupts to move data between peripherals and memory requires dedicating one of the processor's eight address registers to hold the source or destination memory address. This can complicate programming by reducing the number of registers available to the rest of the application. Thus, programmers typically choose fast interrupts for processing frequent interrupts, such as serial I/O, in which the overhead saved justifies dedicating a register. DMA may also be used to process frequent interrupts.

If one of the two instructions in the interrupt vector contains a JSR instruction, a "long" interrupt service routine ensues. The processor saves the PC and status register on the stack, resets some status bits, and begins execution at the address specified by the JSR.

Assuming the processor is in an interruptible state, fast-interrupt latency is 11 instruction cycles from when the interrupt line is asserted to execution of the first word of the two-word interrupt service routine. Because long interrupts must perform a subroutine call, they require 14 cycles before the first word of the service routine is executed.

In both the fast- and long-interrupt cases, all other interrupts, regardless of their priority, are disabled for 11 instruction cycles after an interrupt is recognized as pending.

Motorola DSP563xx interrupt support (with multiple interrupt priorities and automatically nestable interrupts) is extremely flexible.

Stack

DSP563xx family processors feature a 15-level hardware stack for subroutine calls, long interrupts, and hardware DO loops. Programmers may expand the hardware stack to any depth by allocating a block of data memory for use as a stack extension. After a stack push that fills the last word of the hardware stack, the least-recently used stack word is moved to the stack extension area of memory. Assuming on-chip memory is used for the stack extension, the move adds two instruction cycles to the push instruction. Likewise, after a stack pop retrieves a word from the top of the hardware stack, the next-most-recently used stack word is automatically copied from the stack extension to the hardware stack.

Each stack level is composed of two 24-bit words and can store both the program counter and the status register. DO loops use two stack levels per invocation, while subroutine calls and long interrupts use only one. A stack exception trap occurs on stack overflow or underflow.

Bootstrap Loading

DSP563xx processors support a wide variety of bootstrap modes. The state of mode select pins following reset determines the bootstrap mode. The DSP563xx can boot through the external memory interface, the serial control interface, or the host interface.

Peripherals

All current DSP563xx processors provide a six-channel DMA controller, a synchronous serial port, and on-chip debugging support. Most also provide a host interface, an asynchronous serial port, and three timers. Unused host interface pins, timer pins, or serial interface pins can be configured as general-purpose I/O pins. In addition, newer DSP563xx family members provide specialized peripherals to enhance their use for certain applications. Examples are the DSP56307, which includes a filter co-processor to allow general filtering operations to be off-loaded from the DSP core for wireless infrastructure applications, and the DSP56362, which has specialized audio interfaces to support its use in consumer audio applications.

- **DMA controller**

  The six-channel DMA controller allows transfers of data between any combination of on-chip or external memory locations without the processor core's intervention. Since the DMA controller has dedicated address and data buses, and internal memory supports two accesses per cycle, internal DMA transfers do not interfere with the core as long as the accessed memory addresses do not lie in the same partition of memory. Bus allocation conflicts between two or more DMA channels, or between a DMA channel and the core during an external transfer, are resolved using programmable priority levels. This priority level can be independently set at one of four levels for each DMA channel. Internal transfers require two instruction cycles per word, and external transfers require two instruction cycles per word plus any needed wait states. A DMA channel on the DSP563xx can target only one memory space (X, Y, or P) at a time. It is not possible to configure a double-word transfer to two different memory spaces using only one DMA channel.

  Since a single DMA channel on the DSP563xx can target only one memory space at a time, and an efficient FFT implementation makes use of address-aligned data in both the X and Y memory, it is usually necessary to use two DMA channels to efficiently transfer FFT data.

  Each DMA channel has its own control register set. The control registers may be configured to interrupt the processor when a transfer is complete. The control registers can also be configured to allow a variety of addressing options. Any peripheral event (data buffer full or empty) can trigger a DMA transfer. DMA channels support several trigger conditions, including trigger per block transfer and trigger per word transfer.

  The DSP563xx DMA controller has especially flexible addressing options. For example, the DMA control registers can be configured so that a DMA channel will access consecutive memory locations a given number of times, and then jump forward by a given offset value. This is useful for processing two- and three-dimensional arrays.

  Because the DSP563xx has memory-mapped peripherals, DMA can be used to transfer data to and from peripherals. When peripherals are selected as DMA request sources, DMA may be used to service routine I/O transfers.

- **Host port**

  The DSP56301 host port (called the HI32) is a 32-bit parallel host port. The host port can be configured for compatibility with and glueless interface to the Peripheral Component Interconnect (PCI, revision 2.1) bus. Alternatively, it can be configured as a "universal" bus interface, or as general-purpose bit I/O pins. In the universal bus mode, the host port supports 8-, 16-, or 24-bit data transfers. Unused host port pins may be programmed as bidirectional bit I/O pins.

When configured in PCI mode, the host interface can act either as a PCI bus master or as a target (slave) device. Separate control registers and data buffers for the master and slave modes allow the programmer to switch modes with minimal reconfiguration. The DSP56301 supports data bursts of unlimited length. Longer data bursts minimize the proportion of data transfer time dedicated to address generation and bus arbitration, allowing the bus data transfer rate to approach the theoretical maximum of 132 Mbytes per second. Because the DSP56301 native word length is 24 bits, the host interface can extend 24-bit data with sign extension or zero fill to complete a 32-bit word. Alternatively, two 16-bit words may be transferred as one 32-bit word on the PCI bus. The DSP56301 PCI mode supports both 3.3-volt and 5.0-volt signaling environments.

> *Because of the fast data transfer rate and the "plug and play" capability of the PCI bus, the PCI standard has become common on PC motherboards. Thus, the DSP563xx's glueless interface with the PCI bus is an advantage in many PC motherboard and expansion card applications.*

All other DSP563xx family devices have an 8-bit host interface (called the HI08). This interface is a less flexible derivative of the HI32, and does not support PCI mode. The HI08 is designed to provide glueless interface to a variety of multiplexed and non-multiplexed buses. The DSP56362/366 processors come with DMA support for the HI08 (called the HDI08) that allows an external DMA controller device to be connected to the HDI08 on the host side which allows 8-, 16, and 24-bit DMA mode bidirectional data transfers. In addition, the DSP56362/364/366 each have a serial host interface with support for 8-, 16, and 24-bit data transfers.

- **Synchronous serial port**

  The synchronous serial port (called the ESSI, or enhanced synchronous serial interface port) supports 8-, 12-, 16-, or 24-bit words at a maximum bit rate of one-third the processor core's master clock frequency. One receiver and three transmitters may operate simultaneously with either an internally generated or externally supplied clock, the maximum frequency of the internally generated clock being one-fourth the core frequency. This clock can be divided by eight and further divided by a programmable 8-bit linear clock divider. The ESSI transmission can be divided into up to 32 time slots for use in time-division-multiplexed (TDM) networks. The ESSI also supports an asynchronous on-demand burst mode. All DSP563xx family members include two independently controlled ESSI ports, each with its own clock generator.

The ESSI port differs from the SSI port found on the DSP560xx and DSP566xx in several ways. The ESSI port supports DMA, allows faster external clocking for higher throughput, contains slot mask registers to simplify use in time-division-multiplexed networks, supports external memory frame buffers, and contains three transmitters.

The three ESSI transmitters are useful in multi-channel audio and other applications that require multiple output channels.

The ESAI (Enhanced Serial Audio Interface) port found on the DSP5636x processors is a superset of the ESSI port. The ESAI port has six I/O pins, all of which can be configured for transmit. This allows the DSP56362 and DSP56364 to support six-channel audio standards such as AC-3 (Dolby Digital), MPEG-2, and Digital Surround (DTS) while having only a single ESAI port. The DSP56366 has two independent ESAI ports.

The DSP5636x processors also have a separate synchronous serial port called the serial-host interface (or SHI). This port is compatible with both the SPI (serial-peripheral interface) and I^2C bus protocols. It can be programmed either as a master or a slave on the bus.

- **Asynchronous serial port**

  The SCI asynchronous serial port on some family members is intended primarily for asynchronous serial communications, but can also be used as an 8-bit synchronous serial port using a gated clock for frame sync. The maximum clock speed for both the internally generated and externally supplied clock is one-fourth the core frequency. In asynchronous mode, the SCI port supports 10- or 11-bit words (one start bit, eight data bits, optionally one parity or data/address bit, and one stop bit). If an external 16X clock is available for connection to the SCI port's clock pin, then the SCI port can generate a transmit clock that is of a different speed from the receive clock. Otherwise, the transmit and receive clocks must be the same and can be internally generated or externally supplied.

- **Timers**

  All DSP563xx family members other than the DSP56364, feature three timer/counters, each with its own input/output pin. In the standard timer mode, the timers share a common clock, but are otherwise independent. The timer clock can be configured as the processor's master clock divided by two, or as the master clock prescaled by a programmed 21-bit linear value. This clock then increments each timer's 24-bit count register. When a timer's register reaches the programmed comparison value, the timer interrupts the DSP and reloads with the value in the corresponding load register. Other modes use the timer in conjunction with its I/O pin as an event counter, a watchdog timer, a period- or width-measuring device, or a pulse or square wave generator.

Multiple DSP563xx peripherals frequently share the same set of processor I/O pins. For example, serial port pins can be used for either serial I/O or general-purpose I/O. This is both an advantage and a disadvantage: it increases the device's flexibility, but if all peripherals are in use in configurations that use all its pins, the device essentially has no bit I/O.

- **Filter co-processor**

  The DSP56307 and DSP56311 contain a filter co-processor designed to accelerate performance in wireless infrastructure applications. The filter co-processor implements a variety of general filtering and convolution algorithms to off-load that processing from the DSP core.

  The filter co-processor is interfaced as a memory-mapped peripheral. Mode registers define the function and parameters of the co-processor. To initiate co-processor operation, the programmer writes operands to a memory-mapped input register. The input is buffered by a 24-bit x 4-word buffering unit to allow multiple operands to be input before the previous operand has been processed. The co-processor outputs the result in another memory-mapped register, buffered by a single 24-bit register that must be retrieved by the processor core. The DMA controller can be used to move data to and from the co-processor, allowing the co-processor to operate with no intervention from the processor core. Alternatively, interrupts or co-processor mode register polling may be used to monitor the co-processor status, submit operands to the co-processor, and retrieve results.

On-Chip Debugging Support

A six-pin IEEE-1149.1 JTAG port allows access to the OnCE (On-Chip Emulation) scan-based emulation/debugging interface. Through the OnCE interface, the user may examine or modify DSP registers or memory, set hardware breakpoints in both program and data memories (on read, write, or instruction fetch), examine the instruction pipeline, and examine the last twelve change-of-flow instructions executed. Two breakpoints can be placed on individual memory locations, memory intervals, or memory addresses equal, not equal, less than or greater than a given address. These breakpoints may be modified in software, allowing the programmer to set different breakpoints for different sections of code.

In normal operation, the processor enters debug mode when a breakpoint is encountered. Debug mode halts the processor and allows the user to examine and modify the processor's internal register and memory values and to execute instructions by sending commands to the OnCE port from a debugger. Alternatively, setting an interrupt mode enable bit in the OnCE control register causes the processor to execute the vectored interrupt instead of entering debug mode. Thus, the breakpoint can be recognized and appropriately processed in software without halting the processor's operation.

The JTAG port can also be used for boundary scan.

In addition, the DSP563xx features address tracing. By setting a bit in the OnCE Status and Control Register (OSCR), address of internal program fetch, decode and execute can be traced by accessing the external memory address bus (when this bus is not being used for external memory access). Thus, users can trace DSP563xx program execution in real time by monitoring the external memory address pins.

Power Consumption and Management

Motorola states that typical power consumption for the 3.3 V DSP56362 is 466 mW at 100 MHz. For the 2.5 V DSP56307 running at 100 MHz, Motorola states that typical core power consumption is 333 mW. The DSP56311 at 150 MHz and 1.8 volts consumes 189 mW.

The DSP563xx includes two instructions, WAIT and STOP, that reduce processor power consumption. In WAIT mode, the clocks to the internal processor core are gated off, and activity in the processor is suspended until an unmasked interrupt, an external reset interrupt, or a DMA request occurs. Peripherals remain active in WAIT mode. The STOP mode gates off clocks to the processor core and disables peripherals and the on-chip oscillator until the processor is reset, interrupt request A pin is asserted, or a JTAG debug request occurs. Wake-up from STOP mode may take either 24 or 131,070 clock cycles, selectable by the programmer before executing STOP. If the on-chip oscillator is used to drive an external crystal, the longer wake-up period must be used to allow the oscillator to stabilize. STOP mode wake-up time can be further reduced to 8.5 clock cycles if the PLL and on-chip oscillator are not disabled, but power consumption rises.

Motorola states that typical DSP56362 WAIT and STOP mode power consumption when running at 100 MHz and 3.3 volts is 25 mW and less than 1 mW respectively. Motorola states that WAIT and STOP mode power consumption for the DSP56307 is 13 mW and less than 1mW respectively.

All DSP563xx processors have a 3-bit clock divider register that allows the processor to decrease its clock rate by a factor from 2^0 to 2^7 under software control, lowering power consumption proportionally.

Benchmark Performance

The DSP563xx has been benchmarked with the BDTI Benchmarks™. Overall benchmark results for all benchmarked processors are presented in Chapter 8, *BDTI Benchmark™ Results*. We summarize and analyze DSP563xx's benchmark performance in the paragraphs below. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not reflect the processor's instruction cycle rate; therefore, lower instruction cycle counts imply a more powerful architecture, but do not imply faster speed. Next we discuss benchmark execution times and cost-execution time products, indicating processor speed and cost-performance, respectively. We then discuss the processor's energy consumption, which reflects the energy consumed by the processor in order to perform a task. Finally, we discuss the processor's memory usage. We divide the memory usage discussion into three parts: Control benchmark memory usage, overall benchmark program memory usage, and benchmark data memory usage.

Execution Performance

- **Instruction cycle counts**: Total normalized instruction cycle counts are shown in Figure 8.1-13. The DSP563xx total normalized instruction cycle count is somewhat above the average for fixed-point processors benchmarked.

  On the **Real Block FIR** benchmark, the DSP563xx cycle count is roughly 1.5 times the average. The DSP563xx is a single-MAC machine, able to load two operands and perform one multiply-accumulate in one instruction cycle. Hence its cycle count is necessarily higher on this MAC-intensive benchmark, compared with dual-MAC processors such as the TMS320C62xx. Also for this reason, the cycle counts of the DSP563xx on the **Complex Block FIR**, **LMS adaptive FIR**, **Vector Dot Product**, **Vector Add**, and **256-point FFT** benchmarks are above average.

  The DSP563xx achieves an average cycle count on the **Two-Biquad IIR** because the code to implement this benchmark is so short that cycle counts for almost all processors benchmarked lie within a fairly narrow range.

  The DSP563xx's maximum instruction and conditional move instruction give the processor an average instruction cycle count on the **Vector Maximum** benchmark, despite the fact that many of the other processors benchmarked can perform two comparisons in parallel.

  The DSP563xx has a fairly high instruction count on the **Control** benchmark because the processor requires three instruction cycles for each of the many branch instructions required by the benchmark. However, the Control benchmark is optimized for memory usage, not instruction cycle count.

  DSP563xx instruction cycle counts for other benchmarks are unremarkable.

- **Execution times**: Despite the DSP56311's relatively high 150 MIPS instruction execution rate, its high instruction cycle counts contribute to relatively slow executing times, ranking the DSP56311 ninth among the twelve fixed-point processors benchmarked in terms of total normalized execution time. This is illustrated in Figure 8.2-13. Still, given the wide range of total normalized execution times, the DSP56311 is close to average for fixed-point DSPs and is comparable to the other conventional DSP architectures benchmarked, such as the TMS320C5416. The 150 MIPS DSP56311 has a BDTImark2000 score of 450.

- **Cost-execution time**: The DSP56311 ranks worst among the fixed-point processors in terms of total normalized cost-execution time. The DSP56311's total normalized cost-execution time product is approximately 75% above the average for fixed-point DSPs benchmarked, due to the DSP56311's relatively high price of $48.

- **Energy consumption**: The total normalized energy consumption of the DSP56311 is about equal to the average for the 13 fixed-point processors benchmarked on this metric.

Memory Usage

The focus in the memory usage analysis is on Control benchmark memory usage. Unlike other benchmarks, the Control benchmark is optimized for minimum memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP code. Finally we discuss constant and non-constant data memory usage.

- **Control benchmark memory usage:** The DSP563xx's total memory usage on the Control benchmark falls slightly below the average of all processors, as presented in Figure 8.5-9A.
- **Program memory usage**: The total normalized program memory usage for the DSP563xx is about 30% below the average for all benchmarked fixed-point DSP processors. Program memory usage results are presented in Figure 8.5-13.

  The Vector Maximum benchmark on the DSP563xx has the lowest program memory usage for all processors benchmarked, thanks to special instructions for finding the maximum of two numbers and for conditionally executing a transfer between registers.

  The FFT benchmark on the DSP563xx has fairly low program memory even though the DSP563xx implementation unrolls the first two FFT passes into separate code.

  The DSP563xx has fairly high program memory usage for the Viterbi benchmark. This is due to the small number of data registers. For the add-compare-select operation on the DSP563xx, operands must be re-fetched to perform successive adds and subtracts. Each re-fetch uses three bytes of program memory. On other processors with more data registers, the data can be fetched once and used several times. The relatively small number of data registers further means that a series of add and subtract operations in the Viterbi benchmark must be implemented as a set of macro calls instead of implementing these operations inside a loop. Other processors allow one loop to do such operations, resulting in lower program memory usage.
- **Data memory usage**: The DSP563xx is the only processor analyzed here with 24-bit data memory. The DSP563xx's constant and non-constant benchmark data memory usage is generally what is expected for a 24-bit DSP. Because it uses a 24-bit data word instead of the 16-bit word used by other fixed-point processors,

the DSP563xx has the highest non-constant data memory usage among fixed-point processors. Similarly, in terms of constant data memory usage, the DSP563xx generally uses more memory than benchmarked 16-bit, fixed-point DSP processors because of its 24-bit data word. Constant and non-constant benchmark data memory usage is shown in Figure 8.5-14 and Figure 8.5-15.

Cost

Price and packaging options for the DSP563xx family of processors are summarized in Table 7.7-5.

Fabrication Details

DSP563xx family members are all fabricated in 0.32 µm CMOS with three metal layers, except for the DSP56311, which is fabricated in 0.18 µm CMOS with four metal layers.

Part	Speed (MHz)	Voltage	Package	Price (Qty. 10,000)
DSP56301	100	3.3	208 TQFP or 252 PBGA	$43.50
	80	3.3		$34.80
DSP56303	100	3.3	144 TQFP/ 196 PBGA	$18.80/ $18.40
DSP56307	100	2.5/3.3	196 PBGA	$51.90
DSP56309	100	3.3	144 TQFP/ 196 PBGA	$27.80/ $32.20
DSP56311	150	1.8/3.3	196 PBGA	$47.70
DSP56362	100	3.3	144 TQFP	$11.88
DSP56364	100	3.3	100 TQFP/ 100 PBGA	$7.70
DSP56366 (general purpose)	120	3.3	144 TQFP	$16.35
DSP56366 (with ROM decoders)	120	3.3	144 TQFP	$17.15

TABLE 7.7-5. DSP563xx price and package summary for June 1, 2000. Data provided by Motorola.

Development Tools

Motorola's development tools for the DSP563xx include an assembler, linker, instruction-set simulator, a GNU-based C compiler, and a GNU-based debugger. The UNIX versions of the tools are available for SunOS 4, Solaris, and HP-UX. PC-based versions of the tools are available for use with DOS, Windows 95, and Windows NT.

Text-based and graphical interfaces are available for the simulator and the GNU-based debugger. The text-based interface is the same interface used on the DSP560xx tools since 1987. The graphical interface, released in late 1995, provides an ACSII command window for full backward compatibility with the text-based tools.

Motorola's DSP563xx instruction-set simulator is a cycle-accurate, full-functional model that models all of the I/O pins of the devices. To facilitate code optimization, the simulator has application profiling capabilities. The simulator has a C-language procedural interface that allows users to integrate the processor model into simulations of larger systems by writing C code to simulate the rest of the system or by linking the simulator to a general-purpose simulation environment. Third-party interfaces for the DSP563xx simulator may be obtained for the Cadence SPW and Synopsys COSSAP high-level design environments.

> *This simulator provides the user with unusual accuracy and flexibility for creating system simulations.*

Motorola provides two forms of development hardware for the DSP563xx family. The first are low-cost "evaluation modules" (EVM) containing either a DSP56303, DSP56307, DSP56309, DSP56362, DSP56364, or DSP56366 processor, memory, and digital audio codec. The second is a DSP56301 "application development system" (ADS), which can be used as an emulator through the addition of an emulator board (called a command converter) and computer add-in card. The EVM includes a debugger developed by Domain Technologies, while the ADS provides the same user interface as that of the instruction-set simulator.

Motorola now provides a DSP563xx family function library through its website. This library contains a variety of math and signal processing routines, and like other Motorola tools, may be freely downloaded.

Motorola provides documentation, application notes, and development software via its Internet site.

> *Motorola distributes the latest version of its CLAS package on the Internet free of charge. The package contains an instruction-set simulator, assembler, linker, and librarian, and a C compiler. The ADS software may also be obtained free of charge through Motorola's website. Development tool availability on the Internet simplifies software upgrades and is an advantage.*

Applications Support

Motorola provides DSP563xx documentation through the *DSP56300 Digital Signal Processor Family Manual*. This volume discusses the DSP563xx core architecture, but does not discuss memory configurations or peripherals. These details are covered in smaller, variant-specific manuals such as the *DSP56301 User's Manual.*

> *The DSP56300 Family Manual is well written, but suffers from the absence of an index and some needed detail.*

Although the DSP56311 is generally compatible with the other DSP563xx processors, it uses a completely different manufacturing process. This has resulted in small differences between this processor and the other family members. These differences are outlined in the Motorola technical bulletin *Changes in Process Technologies: Hardware and Software Design Implications for the DSP563xx Family Derivatives.*

Motorola provides applications support through the processor user's manuals, training classes, a website, and a newsletter. A telephone and Internet electronic mail helpline for Motorola DSPs is also available.

> *Few application notes are available for the DSP563xx. Motorola does, however, offer a good selection of application notes for the DSP563xx's predecessor, the DSP560xx. The DSP560xx notes are a good starting point for DSP563xx applications.*

A number of companies provide DSP563xx third-party support through boards and software. For example, Domain Technologies provides a OnCE port-based debugging system for the DSP563xx family. Other third-party vendors provide application and function software libraries, emulators, real-time operating systems, and development boards for the DSP563xx. Tasking offers a C compiler for the DSP563xx.

Advantages

- 24-bit data provides greater precision than 16-bit data
- Orthogonal 24-bit instruction set
- Instruction cache
- Eight address registers with individual modifier and modulo buffer length registers
- Virtually unlimited nested zero-overhead loops when stack extender used
- Automatically nestable interrupts
- Conditional execution of ALU operations
- Exponent detect instruction
- Barrel shifter
- Bit manipulation instructions
- Three internal clock/timers

- Good operand-unrelated parallel move support
- On-chip PLL for clock generation
- Clock divider
- Three serial ports on most DSP563xx family members (except DSP56362)
- Synchronous serial port supports TDM (except DSP56362)
- Host port supports PCI (DSP56301)
- Simulator models all I/O pins and can be interfaced to C programs
- "Fast" interrupts
- On-chip scan-based debugging/emulation and boundary scan via JTAG/OnCE port
- Flexible external memory interface with page-mode DRAM support and internal refresh generation
- Some program memory may be designated as an instruction cache to accelerate external memory accesses
- Six-channel DMA controller with separate on-chip address and data buses
- Good power management features
- Flexible bootstrap modes

Disadvantages

- Pipeline effects complicate code optimization and increase cycle count
- Active instruction cache reduces available on-chip program memory
- External memory program fetches cause a one-cycle pipeline interlock

7.8 Motorola DSP568xx Family

**BDTImark2000 Score:
110 at 80 MHz**

Introduction

The Motorola DSP568xx family is based on Motorola's DSP56800 16-bit fixed-point conventional DSP core. Members of the DSP568xx family operate at a maximum of 40 MIPS at 80 MHz using a supply voltage of 3.0-3.6 volts (± 10%). The Motorola DSP568xx processors are targeted at low-cost, low-power applications in wireless and wireline communications, and motion control, where both signal processing and control features are useful. Examples of these applications include digital messaging systems, RF modems, digital answering machines, and servo and motor control.

The DSP568xx family was introduced in 1996. Its architecture and programming model are quite similar to those of the other conventional DSP processor families from Motorola, such as the DSP563xx. However, the instruction set of the DSP568xx has been enhanced and the core architecture has been simplified to increase its general-purpose microcontroller capabilities while maintaining many basic DSP features. In addition, the I/O capabilities of the processor have been tailored to meet the requirements of both DSP and microcontroller applications.

Currently, there are five processors in the DSP568xx family: the DSP56824 and four new closely related members, the DSP56F801, DSP56F803, DSP56F805, and DSP56F807. The "F" in the chip names indicates the presence of on-chip Flash memory. DSP568xx family members are summarized in Table 7.8-1

In October 2000, Motorola introduced the next generation of the DSP56800 architecture, the DSP56800E, which will be used as the basis for the DSP5685x family. This processor family is discussed in Section 7.9 of this report.

At the time of its introduction, the DSP568xx was the first DSP processor family that provides architectural as well as I/O features to address the needs of microcontroller applications. It is an interest-

Part	Max. Speed (MIPS)	Data RAM	Data ROM (*Flash)	Program RAM	Program ROM (*Flash)	Program Boot (*Flash)
DSP56824	35	3.5Kx16	2Kx16	128x16	32Kx16	N/A
DSP56F801	40	1Kx16	2Kx16*	1Kx16	8Kx16*	2Kx16*
DSP56F803	40	2Kx16	4Kx16*	512x16	32Kx16*	2Kx16*
DSP56F805	40	2Kx16	4Kx16*	512x16	32Kx16*	2Kx16*
DSP56F807	40	4Kx16	8Kx16*	2Kx16	60Kx16*	2Kx16*

TABLE 7.8-1. DSP568xx family members.

ing counterpoint to microcontrollers with DSP enhancements, such as Hitachi's SH-DSP.

Architecture

Figure 7.8-1 illustrates the DSP568xx family architecture as typified by the DSP56824.

Data Path

The DSP568xx family's data path is based on a single-cycle 16 × 16 → 32-bit multiplier integrated with a 36-bit ALU. Two 36-bit accumulators, A and B, are provided, each with four guard bits for overflow protection. Both accumulators can be accessed as 36-bit registers or as segments, where the segments include a 4-bit extension register (A2

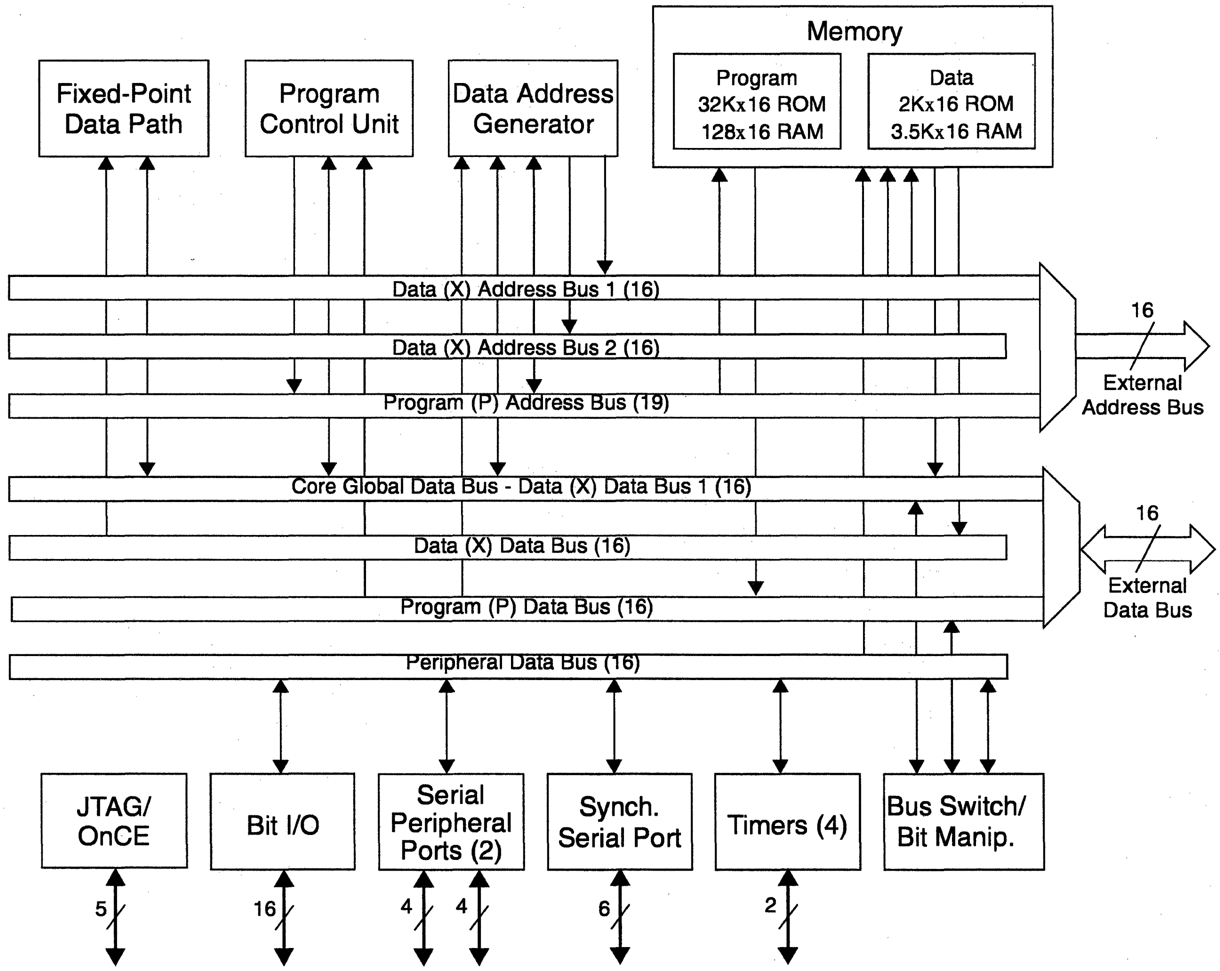


FIGURE 7.8-1. Motorola DSP56824 architecture.

or B2), a 16-bit most-significant register (A1 or B1), and a 16-bit least-significant register (A0 or B0). Additionally, the data path provides three 16-bit data registers: X0, Y0, and Y1. The data registers can be accessed individually, or the Y0 and Y1 registers can be accessed as a single 32-bit register, Y. The inputs for the multiplier can come from registers X0, Y0, or Y1, or from the 16-bit accumulator segments. The inputs for the ALU can come from the X0, Y0, or Y1 registers, from the 36-bit accumulators, from immediate operands, or from memory. ALU operations execute in a single cycle, assuming all operands are registers. Outputs from the MAC/ALU unit may be written to either of the two accumulators or to registers X0, Y0, or Y1. In addition, ALU outputs may be written directly to memory.

The MAC/ALU unit together with the logic unit performs all multiplication, addition, subtraction, logical, and other arithmetic operations except shifting. Besides integer multiplication, the multiply-accumulate unit supports signed/signed and signed/unsigned fractional multiplication; unsigned/unsigned multiplication is not supported. The MAC/ALU unit features a carry bit supporting extended-precision arithmetic and rotate-through-carry operations.

> *The support for signed/unsigned multiplication is an advantage. However, the lack of unsigned/unsigned multiplication complicates implementation of multi-precision arithmetic.*

The data path also includes an accumulator shifter and a 16-bit barrel shifter. The accumulator shifter is capable of performing arithmetic and logical single-bit left/right shift and rotate operations either on a 36-bit accumulator or on 16-bit register operands. The barrel shifter is capable of performing 1- to 16-bit arithmetic and logical left/right shifts in a single instruction cycle. The barrel shifter operates on 16-bit data, and its result may be written to the X0, Y0, or Y1 registers, or to the A or B accumulators.

The data path supports both convergent and round-to-nearest rounding via dedicated rounding instructions. The type of rounding is selected by a bit in a mode register.

The DSP568xx features saturation circuitry that is used to detect 36-bit accumulator values whose magnitude will not fit in a 16-bit destination and to saturate the higher 16-bit portion of the accumulator to a 2's complement representation of ±1.0 when the value is transferred out of the accumulator. The saturation logic cannot be explicitly disabled; however, it only takes effect when the whole 36-bit accumulator is used as a source operand in a MOVE instruction. In addition, the DSP568xx provides a 32-bit saturation mode that can be enabled via a mode bit. When the 32-bit saturation mode is enabled, ALU results will be saturated to the largest or smallest value representable using 32-bit 2's complement arithmetic. Note that this saturation mode is automatically disabled during shift and rotate operations, signed/unsigned multiplications, and logical operations.

Bit manipulations on data memory words, peripheral registers, and all registers within the core are performed by a dedicated bit-manipulation unit that is capable of testing, setting, clearing, or inverting any bits specified in a 16-bit mask.

> *Compared to most other low-cost DSP processors, the bit-manipulation unit of the DSP568xx is powerful, improving the suitability of the processor for control applications.*

Memory System

The DSP568xx family employs a Harvard architecture which allows up to three simultaneous memory accesses: one instruction fetch per instruction cycle, and two data reads or one write per instruction cycle. Program (P) and data (X) memory have separately addressable memory spaces and separate on-chip program and data memories, each of which includes RAM and ROM. Although the program counter (PC) register is only 16-bits, the program (P) memory address space is 19-bits; the upper three "extension bits" of the PC are stored in the Status Register. The data (X) memory address space is 16-bits. The first 128 locations of on-chip program memory are reserved for interrupt vectors, while the last 256 locations of on-chip data memory are reserved for memory-mapped peripheral registers. The DSP56F80x processors implement their program and data ROM in Flash memory, and include an additional 2Kx16 program boot Flash. On-chip memory configurations for the DSP568xx family are shown in Table 7.8-1.

Data memory is accessed via two separate 16-bit buses: the Core Global Data Bus (CGDB) and the Data (X) Data Bus 2 (XDB2) (refer to Figure 7.8-1). Only the Core Global Data Bus can be used to access external data memory. When the processor is executing at 40 MIPS, the maximum sustainable on-chip data memory bandwidth is 80 million 16-bit words/second for read operations and 40 million 16-bit words/second for write operations.

The DSP56824 can execute most instructions from on-chip data memory as well as on-chip program memory, with the exception of instructions that read two operands from data memory. An extra instruction cycle is required for instructions that write to data memory and are executed from data memory.

> *The ability to execute instructions from on-chip data memory can be convenient for patching a program "on-the-fly," but executing from data memory will seriously impair the performance of the DSP568xx in most DSP applications due to the lack of dual parallel reads.*

External Memory Interface

All of the DSP568xx processors except the DSP56F801 have an external memory interface (EMI) which consists of a 16-bit address bus and a 16-bit data bus. The DSP56F801 does not include an EMI. These buses are multiplexed between program and data memory accesses. Program/data and read/write strobes are used to specify the bus contents. One off-chip access can be made without penalty in a single instruction cycle, assuming zero wait states. Because only one of the two data (X) address buses is connected to the external memory interface (see Figure 7.8-1), only one data memory access to the external memory can be made by a single instruction. Thus, the second read in a dual parallel read must be made from on-chip memory. Assuming zero wait states, the maximum peak and sustainable external memory bandwidth is 40 million 16-bit words/second when the processor is executing at 40 MIPS.

Although the program counter (PC), with its extension bits, can address 19-bits of program memory space, the external address bus of the EMI is only 16-bits. According to Motorola, a memory expansion interface would make it possible to take full advantage of the larger 19-bit internal program memory space and increase the size of the external memory from 64Kx16 to 512Kx16. This feature, however, does not appear on any of the current DSP568xx family members.

To adapt the speed of the external memory interface to slower external memory elements, the processor supports programmable wait states. Program and data memory wait states can be independently configured, and specified as a value between 0 and 15 clock cycles. Externally requested wait states are not available.

> *The lack of externally requested wait states is a disadvantage and may complicate system design in some applications. The low-cost DSP56F801 may be unsuitable for some applications due to its lack of an external memory interface.*

Address Generation Unit

The DSP568xx family supports immediate data, memory-direct, register-direct, and register-indirect addressing modes. The processor also supports 6-bit short immediate data and short memory-direct addresses for some instructions.

For register-indirect addressing, the DSP568xx provides a single address generation unit containing a modulo arithmetic unit and an increment/decrement unit. In addition, the address generation unit contains four address registers, R0-R3, a modifier register, N, a modulo register, M01, and a stack pointer register, SP. Addresses generated in the address generation unit are used to access both X and P memory.

The modulo arithmetic unit is used for both linear and circular addressing. Supported register-indirect addressing modes include post-increment or post-decrement by one, by the contents of the modifier register, or no update; and indexed by the contents of the modifier register, by a 6-bit short immediate value, or by a 16-bit long immediate value. The software stack can be accessed using stack-pointer-relative indexing with immediate short (6-bit) or long (16-bit) index values. Pre-modifications of the address registers are not supported.

Register R0 or registers R0 and R1 together can be assigned to operate under circular addressing. Hence, two circular buffers (both having the same size) can be simultaneously active. Circular addressing is enabled by loading the modulo register, M01, with the size of the circular buffer minus one for modulo addressing on only R0, or the size of the circular buffer minus one plus 0x8000 for modulo addressing on R0 and R1. In either case the circular buffer size cannot exceed 16Kx16. The buffer must be aligned on an address that is evenly divisible by *k*, where *k* is the smallest power of two that is greater than or equal to the size of the circular buffer. When circular addressing is disabled (i.e., when linear addressing is used), register M01 must be loaded with 0xFFFF.

The increment/decrement unit performs only linear arithmetic and operates only on the R3 register when it is the second data address register in a dual parallel move instruction. The register-indirect addressing modes for this unit include post-increment and post-decrement by one.

The address generation unit does not support bit-reversed addressing.

> *The lack of bit-reversed addressing in the DSP568xx is a disadvantage for some applications that use radix-2 FFTs.*
>
> *The DSP568xx supports a relatively rich set of addressing modes for a processor with a 16-bit instruction word. However, there are many restrictions regarding the use of these modes.*
>
> *Even though two circular buffers can be active at a time, only one circular pointer (R0 or R1) can be updated within a single instruction. This may complicate implementing some algorithm kernels with single-instruction hardware loops.*
>
> *The orthogonality of the instruction set is reduced by the fact that the second data address generated for a dual parallel read instruction must always come from register R3. This complicates programming.*
>
> *Register-indirect addressing indexed by an immediate value can be effectively utilized by high-level language compilers for accessing local variables in the software stack. For example, add, subtract, increment, decrement, and move instructions accept stack operands.*

Pipeline

The DSP568xx family of processors uses a three-stage pipeline, broken into fetch, decode, and execute stages. The pipeline is visible to the user in the following cases:

- If an address register loaded by the immediately preceding instruction is used in the current instruction, the old value of the register is used.
- Modifications to certain mode registers usually take two instruction cycles to take effect.
- In the case of nested hardware DO loops, after the execution of the inner loop, and after retrieving the loop-counter (LC) and loop address (LA) registers from the software stack, there must be at least two instructions preceding the last instruction of the outer loop, otherwise the old values of the registers are used.

The pipeline effects of the DSP568xx are minor compared to most other DSP processors.

Instruction Set

Tables 7.8-2 and 7.8-3 summarize DSP568xx registers and instructions, respectively.

Assembly Language Format

The DSP568xx assembler supports an opcode-operand style instruction format. For arithmetic instructions, the DSP568xx instruction set specifies an ALU/MAC operation and up to two parallel move instructions. For example:

```
MACR  Y1,X0,A     X:(R0)+N,Y1       X:(R3)+,X0
```

multiplies Y1 with X0, adds their product to the A accumulator, and rounds the result using the rounding mode specified by a bit in the operating mode register (OMR). In parallel, it moves the data memory value pointed to by R0 to register Y1, post-increments R0 by the contents of the N register, moves the contents of the data memory location pointed to by R3 to X0, and post-increments R3 by one. Note that the "R" suffix with the MAC instruction indicates rounding. If rounding is not desired, the MAC instruction should be used instead.

Parallel Move Support

As shown in the above instruction, DSP568xx instructions use operand-unrelated parallel moves. However, because of the limitations of the 16-bit instruction word size, the processor places a large number of restrictions on combinations of instructions, operands, and parallel moves. Most of these restrictions relate to dual parallel moves. These are discussed below in the section on orthogonality.

Registers	Width	Purpose
A, B	36 bits	Accumulators
X0, Y0, Y1	16 bits	ALU/MAC inputs/outputs, general-purpose
R0-R3	16 bits	Address registers
SP	16 bits	Stack pointer register
N	16 bits	Address modifier register
M01	16 bits	Modulo register

TABLE 7.8-2. DSP568xx register summary.

Orthogonality

The programming model of the DSP568xx is in some ways more flexible than that of other conventional DSP processors from Motorola. For example, the X0, Y0, and Y1 registers can be used much like the accumulator registers A and B; e.g., as source and destination operands for arithmetic operations. This improves the orthogonality of the DSP568xx instruction set. Unfortunately, many restrictions are imposed upon the pro-

Class	Instructions
Arithmetic	Absolute value, add, add long word (32 bits) with carry, clear, decrement, increment, negate, subtract, subtract long word with carry, round
Multiplication	Signed multiply (with or without rounding and/or negation), signed multiply-accumulate (with or without rounding and/or negation), signed/ unsigned multiply, signed/unsigned multiply-accumulate, integer multiply
Logic	*And, or, exclusive-or, not* with register or with immediate data
Shifting	Arithmetic and logical one-bit shift left/right, arithmetic and logical multi-bit shift left/right, arithmetic and logical multi-bit shift right and accumulate
Rotation	Rotate left/right one bit
Conditional Execution	Conditionally transfer from a selected register to an accumulator, with an optional transfer of the R0 register to R1
Comparison	Compare accumulators, registers, immediate data, or memory; test accumulator, 16-bit register, or memory location
Looping	Single- and multi-instruction loop, break out of loop unconditionally
Branching	Relative and absolute branch (both conditional and unconditional), branch if selected bits are set/clear
Subroutine Call	Jump to subroutine, return from subroutine, return from interrupt
Bit Manipulation	Bit field test and set, test and clear, or test and change, test high (all of the selected bits are set), test low (all of the selected bits are low)
Special Function	Division iteration, normalization iteration, WAIT and STOP modes, software interrupt, DEBUG mode

TABLE 7.8-3. DSP568xx instruction set summary.

grammer, mainly because of the small (16-bit) instruction width. Most of these restrictions apply to parallel moves. The most significant restrictions are:

- Two reads using register-indirect addressing can be performed in parallel, but only one parallel write is supported.
- When performing two parallel reads, only one address register (R3) can be used for the second parallel read, while two address registers (R0 and R1) are available for the first parallel read.
- In dual parallel read instructions, only registers Y0 and Y1 can be used as destinations for the first parallel read. Similarly, only register X0 can be a destination for the second parallel read.
- In dual parallel read instructions, the first address register (R0) can only be modified by post-incrementing it by one or by the contents of the N register, and the second address register (R3) can only be post-incremented/decremented by one.
- While virtually all ALU/MAC instructions allow one parallel move (read or write), only the following instructions allow two parallel moves (reads only): ADD, MAC(R), MPY(R), and SUB.

Although very efficient parallel code can be written for the processor, we expect that even experienced DSP568xx programmers will be forced to return to the user's manual on a regular basis to remind themselves of the register usage restrictions.

Execution Times

Most DSP568xx MAC/ALU instructions execute in one instruction cycle. Branches, subroutine calls, and return instructions take two to four instruction cycles to execute. Delayed branches are not available. Single- and multi-instruction hardware loops take three instruction cycles to start.

Instruction Set Highlights

The DSP568xx allows most ALU instructions to operate directly on memory without affecting the core registers. For example, the instruction

```
INCW  X:$100
```

increments the word stored in X memory location 0x100 by one. The operands for these instructions can be a 6-bit short immediate address, a 16-bit long immediate address, or the stack pointer indexed with a 6-bit immediate index. These instructions usually require three or four instruction cycles to execute if the memory access is performed to internal memory or to external memory with zero wait states.

This feature is very useful for manipulating temporary variables stored in data memory and memory-mapped registers without disturbing the contents of internal registers. This approach supports both microcontroller functionality and high-level language compil-

ers, and is particularly useful given the processor's small register set.

Other noteworthy DSP568xx instructions include:

- Bit-field test, set, clear, and change instructions (which allow groups of up to 16 bits to be manipulated at once)
- Branch if selected bits are set or clear
- Normalization iteration instruction
- Division iteration instruction
- Conditional transfer instruction for conditional movement of data between registers

The DSP568xx has a rich set of bit manipulation instructions that simplify programming and increase performance in decision-intensive applications, such as state machines.

Execution Control

Clocking

As in the DSP560xx family, DSP568xx processors use a 2X master clock. Thus, an 80 MHz master clock corresponds to an instruction execution rate of 40 MIPS.

The DSP568xx processors include an on-chip clock synthesis module for generating three clock signals for the core and the peripherals. The module is composed of an oscillator, a phase-locked loop (PLL), and a prescaler. The oscillator is usually attached to an external crystal, but can also be driven by an external clock source. The PLL is used to multiply the oscillator output frequency by any 10-bit value (from 1 to 1,024). This multiplied clock signal is delivered to the core and to the peripherals. Some peripherals, such as serial ports, use dividers (separate from the prescaler) to scale the multiplied clock frequency down to an appropriate frequency.

The prescaler is used to scale down the oscillator clock frequency by a factor of 1, 16, 64, or 256. The input for the prescaler comes directly from the oscillator clock (bypassing the PLL). The prescaler output can be used as the clock for the general-purpose timers and the watchdog/real-time interrupt timer (COP/RTI). The serial ports and timers are described in detail in the peripherals section below.

The on-chip clock synthesis module of the DSP568xx is quite flexible and should simplify designs in many applications.

Hardware Looping

Hardware looping is supported in DSP568xx family processors via the REP and DO instructions. The REP instruction repeats a single instruction and is not interruptible.

The DO instruction repeats multiple instructions and is interruptible. In both cases, the repetition count is stored in a 13-bit loop-count (LC) register.

There are two ways to enter a single-instruction hardware loop, depending on the repetition count of the loop. If the repetition count is smaller than 64, it can be encoded as short immediate data in the instruction word. As a result, the loop can be entered by using the following instruction:

```
REP         #N          ; N < 64
```

However, if the repetition count is larger than 63, the following method must be used:

```
MOVE        #N,LC       ; N < 8192
REP         LC
```

The first instruction loads a 13-bit immediate value into the loop-count register and the second instruction starts the repeat loop. Similarly, a multi-instruction loop requires an extra instruction to load the loop counter if short immediate data cannot be used (i.e., if the number of repetitions is greater than 63).

The maximum repetition count which can be stored to the LC register is 8,192. If the LC register is loaded with zero, the loop is repeated 0 times with REP and 8,192 times with DO. The DO and REP instructions each require three instruction cycles to execute.

Any data path register or address register can be used to load the LC register for REP and DO instructions. In this case, the 13 LSBs of the operand register are used to load the loop-count register.

A single-instruction (REP) loop can be nested inside a multi-instruction hardware loop. Nesting of multi-instruction (DO) loops requires saving the loop count and the end address of the outer loop onto the software stack. The start address of the outer loop is automatically saved onto the hardware stack by the DO instruction. The hardware stack is two levels deep and must therefore be saved onto the software stack for each additional nesting level beyond two.

> *Nesting of hardware loops is not well supported on the DSP568xx; for example, two nesting levels incur four overhead instruction cycles per iteration of the outer loop. More than two levels typically incur eight instruction cycles of overhead per loop iteration of each outer loop.*

Interrupts

DSP568xx interrupt requests can be generated from the reset pin, from two external interrupt pins, from on-chip peripherals, or from within the core. Two external, general-purpose interrupt lines are provided on the DSP568xx. The external interrupts are individually maskable and can be programmed as level-sensitive or negative-edge-triggered for DSP56F80x processors or as edge-sensitive or rising- or falling-edge-triggered on the DSP56824. DSP568xx processors also contain eight general-purpose I/O (GPIO) lines that can be programmed to function as external interrupt lines; interrupts generated through these pins have higher interrupt latency than the standard external interrupt pins.

A maximum of ten external interrupt lines are provided by the DSP568xx. This is more than in most other DSP processors.

Interrupts that are generated by on-chip peripherals include: SSI receive and transmit data, SCI receiver full and transmitter empty, ADC conversion complete and zero-crossing, timer/counter overflow, Quadrature Decoder index pulse, PWM reload flag, GPIO interrupt, and the "computer operating properly and real-time interrupt" (COP/RTI). The peripheral modules are described in detail below in the peripherals section. The DSP56824's peripherals use the interrupt channels found on the processor's core. Each peripheral has its own interrupt vector and can be selectively enabled or disabled via the interrupt priority level found on the processor core. The peripherals on the DSP56F80x processors communicate their interrupt requests (IRQs) through the IP Bus Interrupt Controller (ITCN). The ITCN accepts an IRQ from an IP bus-based peripheral, assigns it a user-defined priority level, and then presents the processor core with the highest-level pending IRQ. The ITCN augments the DSP56800 core's interrupt controller, expanding the total number of available peripheral interrupts to 54. The interrupts generated within the core include hardware and software interrupts. The software interrupts include stack error, illegal instruction, and software interrupt traps.

With the exception of the hardware reset and COP timer reset interrupt, all external interrupts and interrupts from the on-chip peripherals are maskable. The interrupts generated within the core cannot be masked. Non-maskable interrupts, such as reset and software trap, have the highest priority level. If more than one interrupt with the same priority level is pending simultaneously, they are arbitrated depending on the interrupt vector location in the interrupt vector table.

Each interrupt has a two-word interrupt vector that resides in low memory. During interrupt handling, only the second word of the vector is fetched in order to determine the starting address of the interrupt service routine. The first word of the vector is never read, but in order to maintain compatibility with future family members, the first word of the vector should always be an unconditional jump to subroutine (JSR). At the start of interrupt processing, the program counter and the status register, which together contain the 19-bit return address, are saved, and the program flow is changed to the start of the interrupt service routine. Interrupts on the DSP568xx are nestable, but the appropriate context save and restore and re-enabling of interrupts must be performed in software.

Assuming that the processor is in an interruptible state, interrupt latency is nine instruction cycles from the assertion of the interrupt signal to the execution of the first instruction of the interrupt service routine.

Stack

DSP568xx processors feature a two-level hardware stack. A software stack is also supported via a dedicated stack pointer register. The hardware stack is used only for hardware looping. The software stack is used for passing parameters, storing return addresses, holding local variables for subroutines, etc. In addition, interrupts use the software stack for storing the return address and the status register when entering an interrupt service routine.

> *Although a software stack can be implemented on any DSP processor, only a few DSPs feature a dedicated stack pointer which is automatically updated after stack operations as found on the DSP568xx. In addition, many DSP568xx instructions support stack-pointer-relative addressing, facilitating development of efficient compilers.*

A stack exception interrupt detects hardware stack overflow. Underflow is not detected.

Bootstrap Loading

Upon reset, the ROM-memory based DSP56824 begins execution at program memory address 0x0000. A boot Flash is provided on the DSP56F80x processors to handle initialization in the event that program memory is corrupt or not yet loaded. The bootstrap routine stored in the 2Kx16 boot Flash should check that the contents of program memory are correct, load or reload program memory if necessary, and setup the memory map by writing to the System Control Register.

Peripherals

The DSP568xx family comprises two distinct groups, the newer DSPF80x and the older DSP568xx family members now represented exclusively by the DSP56824. Several new peripherals are available on the DSP56F80x in addition to most of the peripherals on the DSP56824. In total, the peripherals that are now available include two general-purpose bit I/O ports, one “synchronous serial interface” (SSI), up to two “serial peripheral interfaces” (SPI), up to four general-purpose “Quad Timer Modules”, up to two Pulse Width Modulation (PWM) modules, up to four multi-channel A/D converters (ADCs), up to two quadrature decoder modules, up to two “serial communication interfaces” (SCI), an MSCAN module that implements CAN 2.0 A/B, and a “computer operating properly/real-time interrupt” module (COP/RTI).

> *Motorola DSP56F80x family members target embedded control applications with peripherals like multi-channel PWMs, ADCs, quadrature decoders, and a CAN 2.0 A/B bus.*

- **I/O port B**

  The general-purpose I/O port (port B) provides 16 programmable I/O pins. The pins can be individually configured as inputs or outputs as determined by a control bit in a register. The lower eight pins of this port can be used as edge-sensitive external interrupt pins and can be programmed to detect either a rising or falling edge. Once the desired transition occurs, the interrupt is generated, and all of the enabled pins are latched into a register. The interrupt service routine can then inspect the contents of this register to determine the source of the interrupt.

- **I/O port C**

  The "peripheral communications port" (port C) provides 16 multiplexed programmable I/O pins. These pins can be allocated to the on-chip peripherals (the timer modules, the SSI, SPI, etc.), or can be programmed to function as general-purpose I/O pins in a similar manner to the pins of port B.

  Ports B and C together provide 32 I/O pins, useful for controller applications.

- **Synchronous Serial Interface (SSI)**

  The SSI appears only on the DSP56824. The SSI is a full-duplex serial port, and consists of independent transmitter and receiver sections with independent, internal or external clock generation and frame synchronization. This interface can be accessed using six pins of I/O port C and programmed for word lengths of 8, 10, 12, or 16 bits. It can also operate in a network mode allowing TDM capability with up to 32 time slots per frame. The maximum bit rate of the SSI is the master clock frequency divided by 4, e.g., 17.5 MHz in the case of a 70 MHz master clock frequency.

- **Serial Peripheral Interface (SPI)**

  The DSP568xx provides one or two serial interfaces, called "serial peripheral interfaces" (SPIs), which are used for simple synchronous communication between devices. The SPI data transfer requires a master and a slave device. During data transfer, which is always initiated by the master, one byte of data is exchanged between these devices. Each SPI has four pins including input and output data pins, a slave select pin, and a serial clock pin. The serial clock is generated by the master device. Selection of clock polarity and a choice of two different clocking protocols are allowed by the clock control logic. The maximum frequency of the serial clock is 10 MHz. Thus, for a 70 MHz DSP56824 the frequency of the serial bit clock must be programmed to be no more than 10 MHz. The SPI module also includes error detection logic. Like the SSI, this interface can be accessed via the pins of I/O port C.

- **Timers**

  The general-purpose timer module includes three identical 16-bit timer/counters for the DSP56824, referred to as a tri-timer module, or as four identical 16-bit timer/counters for the DSP56F80x, which Motorola calls a "Quad Timer" module. Up to four Quad Timer modules are available on the DSP56F80x, while the DSP56824 comes with one tri-timer module. Both timer modules provide several common features. Each timer module provides independently programmable timer/counters and can be clocked by internal or external signals; the maximum count rate for the quad-timer is one-half the master clock rate when using internal clocks and one-fourth the master clock rate when counting external events, while the maximum count rate for the tri-timer is one-fourth the master clock rate for both internal and external events. The timer/counters can be preloaded, cascaded, and used to generate interrupts or to signal external devices at periodic intervals. The number of I/O pins available to a timer module varies from two to four and in the case of timer/quadrature decoder pairs, these pins are multiplexed.

- **Pulse Width Modulation (PWM)**

  Pulse Width Modulation is only available on DSP56F80x processors. The DSP56F801/803 have one PWM module, while the DSP56F805/807 have two PWM modules. Features vary slightly between PWM modules but all have six output pins which can be configured as three complementary pairs, or six independent outputs, or any other combination such as two complementary pairs and two independent outputs, etc. Complementary pair operation allows for deadtime insertion, separate top and bottom polarity control, and separate top and bottom pulse width correction via current status inputs or software. All of the PWMs also feature 15-bit resolution, programmable edge-aligned or center-aligned pulse signals, and an unsigned 15-bit PWM modulus which determines the period of the PWM output. A full featured PWM module has three current-sense inputs and four fault inputs as well. The PWM clock is produced by a prescaler with selectable rates equal to the master clock rate divided by two, four, eight, or sixteen. The period and pulse width are alignment dependent; that is, with edge-aligned output, the modulus is the period of the PWM output in PWM clock cycles, but with center-aligned output, the modulus is one-half of the period of the PWM output in PWM clock cycles. In a similar fashion, the pulse width with center-aligned output is twice the value written to the PWM value register.

- **A/D Converters (ADCs)**

  ADC modules are only available on DSP56F80x processors. The DSP56F801/803/805 have one ADC module, while the DSP56F807 has two ADC modules. Each ADC module features two, four-input ADCs with 12-bit resolution. The eight inputs can be configured as up to 8 single-ended channels, up to four differential channel pairs, or a combination of these. The ADC module performs a ratiometric conversion and can be configured for single scan, scan when triggered, or repeat programmed scan sequence. The number of conversions per scan is programmable from 1 to 8 for a single-ended input configuration, or from 1 to 4 for a differential input configuration. The ADC can be configured for simultaneous or sequential conversions. Configured for sequential conversions with multiple conversions per scan, a new conversion is immediately initiated upon completion of the previous conversion. When configured for simultaneous conversion, two different channels are converted in parallel. Running the ADC module at its maximum clock rate of 5 MHz while configured for simultaneous conversions, the

ADC module can produce a maximum of two samples in 1.2 μs, or 1.66 million samples per second. The ADC can also be synchronized to the PWM.

- **Quadrature Decoders**

  The quadrature decoder module is designed to decode signals from an external quadrature encoder, typically installed on a motor shaft or drive shaft. Quadrature decoder modules are only available on DSP56F803/805/807 processors. The DSP56F803 comes with one quadrature decoder module, while the DSP56F805/807 come with two quadrature decoder modules. The quadrature decoder modules are paired with Quad Timer modules and share their I/O pins. Thus, the use of a quadrature decoder precludes the use of its paired timer module; however, at least one dedicated Quad Timer module will still be available. The quadrature decoder module features a configurable digital filter for inputs, a 32-bit position counter, a 16-bit position difference register, and a preloadable 16-bit revolution counter. The maximum count rate is one-half the master clock rate, assuming a master clock rate of 80 MHz, resolutions from 25 ns to 1.6 μs, and count periods from 1.62 msec to 102 msec can be obtained using the prescaler.

- **Serial Communication Interface (SCI)**

  SCI modules are only available on DSP56F80x processors. The DSP56F801/803 have a single SCI module, while the DSP56F805/807 have two SCI modules. The SCI allows full duplex, asynchronous, non-return-to-zero (NRZ) format serial communication between the DSP56F80x processor and remote devices, including other processors. The SCI features 13-bit baud rate selection, programmable 8-bit or 9-bit data formats, separately enabled transmitter and receiver, separate transmitter and receiver interrupt requests, and hardware parity checking. Although the transmitter and receiver sections of the SCI operate independently, they share the same baud rate generator. The maximum baud rate is the SCI module clock rate divided by 16, or equivalently, the master clock rate divided by 32.

- **Motorola Scalable Area Network (MSCAN module)**

  The MSCAN module is only available on the DSP56F803/805/807 processors. The module is a communication controller implementing the CAN 2.0 A/B protocol developed by Robert Bosch GmbH. The CAN protocol, optimized for operation in interrupt-driven, real-time environments, is finding widespread use in industrial control, in addition to its primary use as a vehicle serial data bus. The MSCAN features standard and extended data frames, 0-8 byte data length, programmable bit rates up to 1 Mbps, double buffered receive storage, triple buffered transmit storage, and two full-size extended identifier filters.

- **COP/RTI module**

  The COP/RTI module, which is available on all members of the DSP568xx family, provides two separate functions: a watchdog-like timer and a periodic interrupt generator. The COP timer protects against system failures by providing a means to escape from unexpected situations, such as an endless loop due to a programming error. Once started, the COP timer must be reset frequently so that it never reaches its time-out value. If a time-out value is reached, an internal reset is generated, and the COP reset interrupt handler is executed. The periodic interrupt timer provides a real-time interrupt capability on the DSP568xx.

On-Chip Debugging Support

A six-pin IEEE-1149.1 JTAG/OnCE port allows access to the OnCE (On-Chip Emulation) scan-based emulation/debugging interface. Through this interface the user may examine or modify DSP registers or memory, set hardware breakpoints in both program and data memories (on read, write, or instruction fetch), examine the instruction pipeline, and examine the last five instructions executed. Additionally, any instruction may be executed through this interface.

The JTAG/OnCE port can also be used for boundary scan.

Power Consumption and Management

The DSP568xx family operates from a nominal supply voltage of 3.3 volts ± 10%; i.e., the devices can operate from 3.0 to 3.6 volts. Additionally, most I/O pins are 5 volt tolerant. Motorola states that typical power consumption for DSP568xx family members supplied at 3.3 volts is as follows: 53.5 mW for the DSP56824 when executing at 35 MIPS, and 91.1 mW for the DSP56F80x when executing at 80 MHz.

DSP568xx processors provide WAIT and STOP modes to reduce power consumption when no processing is required. In WAIT mode, the internal clocks to the processor core and memories are gated off, and all activity in the processor is suspended until an interrupt occurs. The clock oscillator, the PLL, and the internal I/O peripheral clocks remain active during WAIT mode. STOP mode achieves the lowest power consumption by gating off the clocks to many of the on-chip peripherals in addition to gating off the clocks to the core and memories. The PLL may remain active in STOP mode or can be disabled before entering this mode to minimize power consumption. However, disabling the PLL increases the processor's wake-up latency. Similarly, the COP/RTI module and the timers may continue operating or be individually disabled when entering STOP mode. The DSP56F80x processors have an optional disable for the STOP and WAIT modes. The STOP and WAIT instructions may be disabled permanently, (that is, until the next reset), or temporarily by writing to the System Control Register (SYS_CNTL). Motorola states that the power consumption when in STOP mode, with all peripherals (including the PLL) disabled at 3.3 volts is 16.5 µW for the DSP568xx family.

Benchmark Performance

The DSP568xx has been benchmarked with the BDTI Benchmarks™. Overall benchmark results for all benchmarked processors are presented in Chapter 8, *BDTI Benchmark™ Results*. We summarize and analyze the DSP568xx's benchmark performance in the paragraphs below. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not reflect the processor's instruction cycle rate; therefore, lower instruction cycle counts imply a more powerful architecture, but do not imply faster speed. Next we discuss benchmark execution times and cost-execution time products, indicating processor speed and cost-performance, respectively. We then discuss the processor's energy consumption, which reflects the energy consumed by the processor in order to perform a task. Finally, we discuss the processor's memory usage. We divide the memory usage discussion into three parts: Control benchmark memory usage, overall benchmark program memory usage, and benchmark data memory usage.

Execution Performance

- **Instruction cycle counts:** As illustrated in Figure 8.1-13 in Chapter 8, *BDTI Benchmark™ Results*, the DSP568xx total normalized instruction cycle count is the third-highest result for all of the benchmarked processors, at about 40% higher than the average.

  The DSP568xx can fetch two operands and perform a multiply-accumulate (or add) operation in one instruction cycle, which is the maximum throughput for a single-MAC processor. Several other processors among those surveyed have architectures with similar MAC throughput; for example, the DSP563xx, Analog Devices ADSP-218x,and Texas Instruments TMS320C54xx. The total normalized instruction cycle count for the DSP568xx is higher than that of these three processors largely because of its very high cycle counts for the FFT and Viterbi benchmarks, discussed below.

  For the **Real Block FIR**, **Complex Block FIR**, and **Vector Add** benchmarks, the DSP568xx instruction cycle counts are relatively high compared with those of other processors benchmarked. The Vector Dot Product benchmark is also above average. For at least these four benchmarks, multi-MAC processors such as the Texas Instruments TMS320C62xx achieve lower cycle counts by performing calculations on more than one sample in one instruction cycle. The advantage of multi-MAC processors is less noticeable on the single-sample benchmarks, such as the **Single-Sample FIR** and the **Two-Biquad IIR** benchmarks.

  The DSP568xx has by far the highest instruction cycle count on the **FFT** benchmark, at more than twice the average. This is mainly a result of several limitations regarding dual parallel memory moves on this processor: A dual parallel move is limited to a dual read. A dual write, or a parallel read and write, is not possible. Further, only address register R3 can be used as an address register for the second parallel read in an instruction, and in such a read, address register R3 can be incremented or decremented only by one. The processor's cycle counts are further increased because it has no hardware support for bit-reversed addressing. Approximately 10% of the total cycles for the FFT benchmark on the DSP568xx are attributable to the processor implementing bit-reversal in software.

The DSP568xx also has the highest instruction cycle count on the Viterbi benchmark, at more than twice the average. This is due primarily to restrictions on double-word read operations, and to its limited number of data registers. For the add-compare-select operation, the DSP568xx must reload operands to perform successive adds and subtracts since it does not have enough data registers to avoid reloading data words. On other processors with more flexible double-word read capabilities, it takes fewer instructions to do the reload. On other processors with more data registers, the data can be loaded once and used several times.

- **Execution times:** The DSP56F801 has the slowest instruction execution rate of the benchmarked DSP processors (40 MIPS at 80 MHz). This, combined with its relatively high cycle counts, makes it the slowest fixed-point processor benchmarked. The total normalized execution time for the DSP56F801 is about four times slower than the average of all benchmarked fixed-point processors, as presented in Figure 8.2-13. The 80 MHz DSP56F801 has a BDTImark2000 score of 110.
- **Cost-execution time:** In terms of total normalized cost-execution time product (presented in Figure 8.3-13), the DSP56F801's fairly low $8.15 price tag (quantity 10,000) cannot wholly offset its slow execution time results. The total normalized cost-execution time of the DSP56F801 is about 30% worse than the average for all benchmarked fixed-point processors.
- **Energy consumption:** The DSP56824 has the lowest power consumption of the processors benchmarked here, but its low power consumption cannot compensate for its very slow execution times. As shown in Figure 8.4-13A and B, the DSP568xx has the second-worst total normalized energy consumption among the fixed-point processors—only the Texas Instruments VLIW-based TMS320C6204 is worse. The DSP56824 total normalized energy consumption is roughly 75% higher than the average for all benchmarked fixed-point DSP processors.

Memory Usage

The focus in the memory usage analysis is on Control benchmark memory usage. Unlike other benchmarks, the Control benchmark is optimized for minimum memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP code. Finally we discuss constant and non-constant data memory usage.

- **Control benchmark memory usage:** As illustrated in Figure 8.5-9A, the DSP568xx achieves the second-lowest total memory usage of all benchmarked DSP processors on the **Control** benchmark, bettered only by its successor, the

DSP5685x. The key features which enable the DSP568xx to achieve good code density on control-oriented code include its short, 16-bit instruction word width, PC-relative branches, and support for short immediate data.

- **Program memory usage:** As Figure 8.5-13 illustrates, except for the StarCore SC140, the normalized program memory use for the processors with 16-bit or 24-bit instructions falls within a fairly consistent range. This includes the DSP568xx. Despite its 16-bit instruction word width, the instruction set of the DSP568xx is extensive (but non-orthogonal) and allows more operations to be performed per instruction than most other 16-bit fixed-point DSP processors. Such efficiency enables the DSP568xx to achieve the lowest program memory usage on the LMS Adaptive Filter, Vector Dot Product, and (tied with four other processors) the Vector Add benchmarks. The processor also achieves the second-lowest program memory usage result (tying in some cases with other processors) on the Single-Sample FIR, Two-Biquad IIR, Vector Maximum, and Bit Unpack benchmarks.

  The DSP568xx has by far the largest program memory usage for the Viterbi benchmark, particularly when compared to the processors with 16- or 24-bit instructions. As with the instruction cycle counts for the Viterbi benchmark, this is due primarily to the restricted double-word read capabilities. For the add-compare-select operation on the DSP568xx, operands must be reloaded to perform successive adds and subtracts. Each reload of a double word uses four bytes of program memory. On other processors with a more flexible double-word data load capability, the amount of program memory required is smaller. The DSP568xx also has high program memory usage because of its limited number of data registers. On other processors with more data registers, the operands can be fetched once and used several times, which is not possible on the DSP568xx. The relatively small number of data registers on the DSP568xx further means that a series of add and subtract operations on the data must be implemented as a series of macro calls, since there are not enough registers to implement the code within a single loop. Other processors with 24-bit instructions, such as the ADSP-21xx family, allow one loop to do such operations.

- **Data memory usage:** The DSP568xx's constant and non-constant data memory usage is generally as expected for a 16-bit fixed-point DSP processor. Total normalized constant data memory usage is shown in Figure 8.5-14 and Figure 8.5-15.

  The DSP568xx has a fairly high non-constant data memory usage on the Control benchmark compared with other benchmarked fixed-point processors. In a subroutine call, the DSP568xx stores the return address and the processor's status word onto the software stack. Most fixed-point processors either do not store the status word or provide hardware stacks or shadow registers for saving registers and return addresses.

The DSP568xx's instruction execution rate of 40 MIPS is the slowest of all benchmarked fixed-point DSP processors. Combined with *the processor's relatively high cycle counts, it results in benchmark execution times that are the slowest of all benchmarked DSP processors. The processor's cost-execution time and energy consumption results are also adversely affected.*

The DSP568xx achieves the second-lowest total normalized program memory usage, and the second-lowest total memory usage on the Control benchmark. As indicated by these metrics, the DSP568xx achieves good code density.

Cost

DSP568xx price and package options are shown in Table 7.8-4.

Fabrication Details

The new DSP568xx family members, the DSP568F801/803/805/807, are all fabricated in 0.25 µm three-metal-layer CMOS technology, while the DSP56824 is fabricated in 0.35 µm three-metal-layer CMOS technology.

Development Tools

Motorola's development tools for the DSP568xx include a text-based assembler, linker, and simulator (CLAS software development package) and a simulator/debugger with a window-based graphical user interface. The tools use either standard COFF object format or Motorola's own ASCII-based object format. This package is available for MS-DOS/Windows, Windows 95/98, Windows NT4.0, SunOS, Solaris, and HP-UX.

The tool set is not a part of an IDE, and other tools, such as "make," are necessary to conveniently handle software projects. However, the debugging facilities of the simulator are good, and the ability to set data breakpoints is noteworthy.

Part	Speed (MIPS)	Voltage	Package	Price (Qty. 10,000)
DSP56824	35	2.7-3.6	100 TQFP	$6.30
DSP56F801	40	3.0-3.6	48 LQFP	$8.15
DSP56F803	40	3.0-3.6	100 LQFP	$11.00
DSP56F805	40	3.0-3.6	144 LQFP	$13.60
DSP56F807	40	3.0-3.6	160 LQFP 160 MBGA	$16.44

TABLE 7.8-4. Pricing for DSP568xx versions. Prices as of July 2000.

Motorola's DSP568xx instruction-set simulator accurately models all of the I/O pins of the device, provides cycle-accurate simulation of the processor and enables modeling of external data streams and interrupts. As is typical for Motorola's simulators, this simulator also has a C-language procedural interface that allows the integration of the processor model with other simulation environments.

Besides the CLAS package, Motorola offers a C compiler for the DSP568xx.

> *Motorola distributes the latest version of its CLAS package, which contains an instruction-set simulator, assembler, linker, and librarian on the Internet free of charge. Development tool availability on the Internet simplifies software upgrades and is an advantage.*

Motorola offers evaluation modules (EVMs) based on each member of the DSP568xx family: DSP56F801/803/805/807EVM and DSP56824EVM. These EVMs run on PC compatible systems running Windows 95/98/NT4.0, and communicate with the host through a UART RS-232 interface or an on-board parallel port "command converter interface." In addition to the processor, the EVMs feature 64Kx16 of external program SRAM and 64Kx16 of external data SRAM, a notable exception being the DSP56F801 EVM, which has no external memory. Additional features that are available on some evaluation modules include: a 13-bit linear voice audio codec, a CAN interface for high speed communications, a 4-channel 10-bit serial D/A converter, and expansion connectors for application-specific components. An alternative to the EVM is an "Application Development System" (ADS). The DSP56824 ADS is a full-featured platform for real-time software development and debugging. The ADS extends the features and functionality of the EVMs to include, among other things, Metrowerks CodeWarrior IDE and an external Command Converter Interface to assist in system-level debugging. The Application Development Systems run on IBM PC-compatible and Sun hosts with a host interface card for ISA or Sun S-bus interfaces.

Besides Motorola's own tools, a number of third-party vendors have development tools and real-time operating systems for the DSP568xx family of processors. Domain Technology's SB-56K emulator supports not only the DSP568xx family, but all other fixed-point Motorola DSP56xxx family processors as well. The SB-56K emulator supersedes the LINK-56K emulator and works at the source level with a Microsoft Windows-based debugger and supports the development of both C and assembly programming. Metrowerks CodeWarrior for Motorola DSP Embedded Systems is an IDE featuring a simulator, debugger, C compiler, and assembler. The IDE is targeted at PC compatible systems running Windows 95/98/NT4.0.

Applications Support

The base document for the DSP568xx family is the *DSP56800 Family Manual.* This volume describes the processor core and the instruction set, but not the peripherals or memory configuration. These variant-specific details are covered in the *DSP56824 User's*

Manual and the *DSP56F801/803/805/807 User's Manual.* Additionally, Motorola produces a data sheet for each processor which contains electrical specifications, timing, pinout, and packaging descriptions.

Overall, the DSP568xx documentation is very good.

Motorola provides applications support via the processor user's manual, application notes, training classes, a telephone hot-line, and a newsletter. User's manuals, technical documentation, product information, and press releases can be found at Motorola's website.

Advantages

- 16-bit barrel shifter
- Bit manipulation unit; good bit-field manipulation instructions
- Software stack with frame pointer supports HLL compilers
- Serial port TDM mode
- Up to sixteen timer/counters
- Good power management features
- Ability to use 1X or slower external clock
- Flexible PLL
- Good documentation
- Low cost (e.g., $8.15 for the 40 MIPS DSP56F801, quantity 10,000)
- JTAG emulation port with boundary scan
- Flexible I/O interfaces
- PWMs
- Dual Multi-Channel ADCs
- Hardware Quadrature Decoder
- Advanced serial communications
- Flash Memory
- Good code density on BDTI's Control benchmark

Disadvantages

- Limited parallel move support: two reads or one write per instruction cycle, second data operand must come from on-chip memory and can only be accessed with address register R3
- No bit-reversed addressing
- Limited circular addressing (two circular buffers supported; however, the size of the buffers must be the same and the circular pointers cannot be simultaneously modified)

- No externally requested wait states
- Slow execution times on the BDTI Benchmarks
- Poor energy consumption results on the BDTI Benchmarks

7.9 Motorola DSP5685x Family

BDTImark2000 Score: Not Available

Introduction

The Motorola DSP5685x family is based on Motorola's new 16-bit fixed-point conventional DSP core, the DSP56800E. The DSP56800E, announced in October 2000, is an enhanced version of Motorola's DSP56800 core, which is used in its DSP568xx family. Because the two families are so similar, this analysis highlights only the differences between the DSP5685x and the DSP568xx. Readers should refer to the DSP568xx analysis, Section 7.8, for a full discussion of the DSP568xx architecture, and for details of features identical between the two families.

Initial members of the DSP5685x family are expected to operate at a maximum of 120 MHz using a dual voltage supply of 1.8 volts for the core and 3.3 volts for I/O. Motorola has announced two members of the DSP5685x family: the DSP56854 and DSP56853, summarized in Table 7.9-1. These chips are expected to sample in the first quarter of 2001, according to Motorola. Motorola states that it has fabricated a DSP56800E-based in-house test chip, which executes at 120 MHz. Because BDTI has not yet verified the clock speed of DSP5685x silicon, there is no BDTImark2000 score currently available for this processor. Check BDTI's website (*www.BDTI.com*) for updated BDTImark2000 scores.

Unlike the 2X master clock on the DSP568xx, the DSP5685x uses a 1X master clock (see *Clocking* in the *Execution Control* section of this analysis for further details). Thus, unlike the DSP568xx, clock cycle and instruction cycle length are equivalent on the DSP5685x. (Each instruction cycle on the DSP568xx consumes two clock cycles.)

The DSP56800E

is a superset of the DSP56800 instruction set; the DSP5685x can execute assembly source code written for the DSP568xx. The hybrid DSP/MCU functionality of the DSP568xx is retained in the DSP5685x, while several of the shortcomings of the DSP568xx, such as a lack of hardware support for nested DO loops, have been eliminated. Like the DSP568xx, the DSP5685x targets automotive and motor control applications. In addition, the DSP5685x targets what Motorola calls "teledatacom" applications, which include low-end cellular phones, PDAs, Internet audio, Web/screen phones, and other Internet appliances.

Part	Max. Speed (MHz)	Data RAM	Data ROM	Program RAM	Program Boot ROM	External Memory Interface
DSP56853	120	4Kx16	4Kx16	12Kx16	1Kx16	No
DSP56854	120	4Kx16	4Kx16	12Kx16	1Kx16	Yes

TABLE 7.9-1. DSP5685x family members.

Unlike the DSP56800, the DSP56800E is available as a licensable core for system-on-chip designers. The core is available in Verilog HDL format.

The availability of the DSP56800E as a licensable core is an advantage.

Architecture

Figure 7.9-1 illustrates the DSP5685x architecture as typified by the DSP56854.

Data Path

The data path of the DSP5685x is similar to the data path of the DSP568xx, but with a few enhancements. The most significant differences are the addition of two 36-bit accumulator registers (for a total of four), the two-stage pipelining of the data path, the

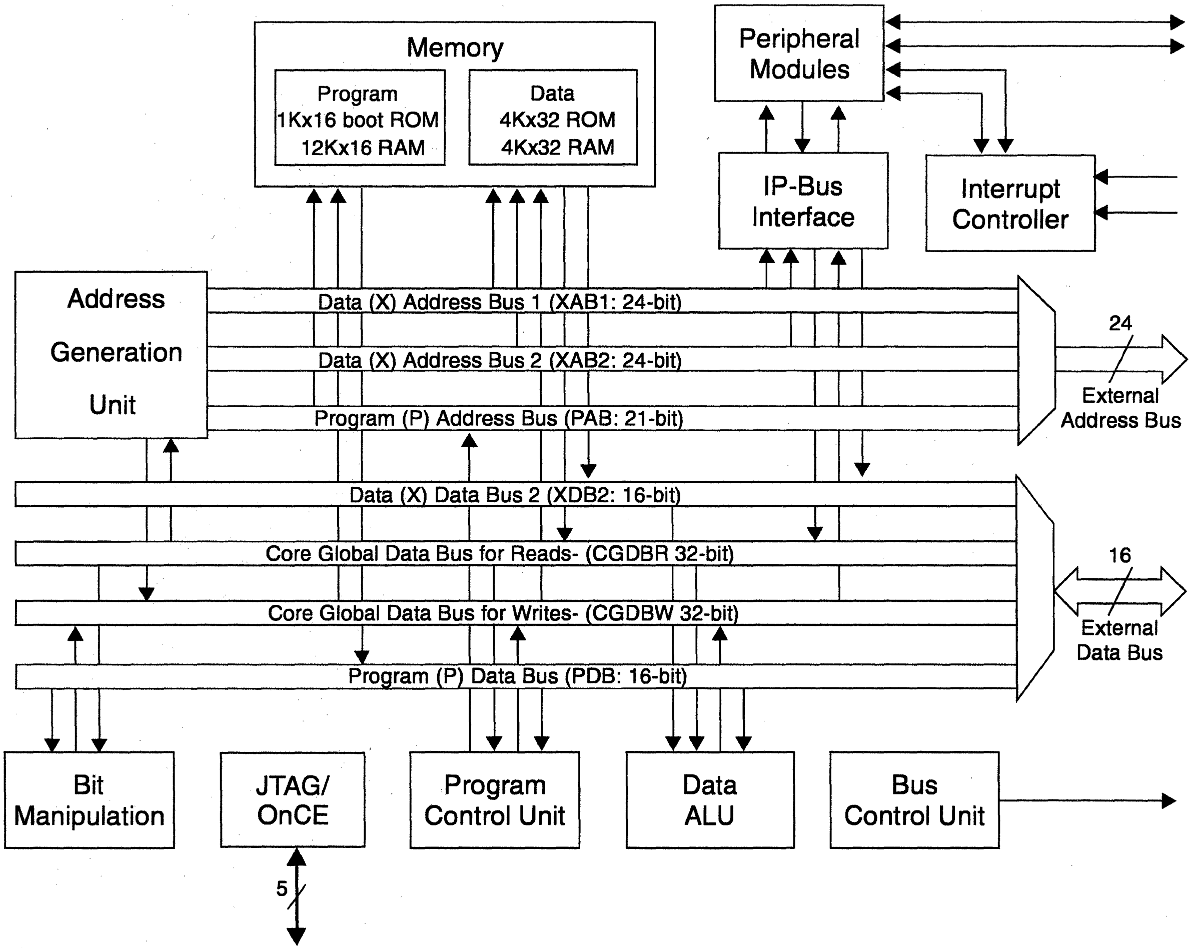

Memory
Program
1Kx16 boot ROM
12Kx16 RAM
Data
4Kx32 ROM
4Kx32 RAM
Peripheral Modules
IP-Bus Interface
Interrupt Controller
Address Generation Unit
Data (X) Address Bus 1 (XAB1: 24-bit)
Data (X) Address Bus 2 (XAB2: 24-bit)
Program (P) Address Bus (PAB: 21-bit)
24
External Address Bus
Data (X) Data Bus 2 (XDB2: 16-bit)
Core Global Data Bus for Reads- (CGDBR 32-bit)
Core Global Data Bus for Writes- (CGDBW 32-bit)
Program (P) Data Bus (PDB: 16-bit)
16
External Data Bus
Bit Manipulation
JTAG/ OnCE
5
Program Control Unit
Data ALU
Bus Control Unit

FIGURE 7.9-1. Motorola DSP56854 architecture.

extension of logical/arithmetic shifting and the logic unit to 32 bits, the addition of an exponent detector which allows single-cycle detection of leading 0s or 1s, support for unsigned/unsigned integer multiplication, and explicit support for three data types: long data (32 bits), word data (16 bits), and byte data (8 bits). (The DSP568xx supports only word data.) Because the data path of the DSP5685x is divided across two pipeline stages, some data ALU operations have a latency of two clock cycles. This is different from the DSP568xx's single-stage, single-cycle data path. Although latency is in some cases increased, instruction throughput (the rate at which instructions complete execution) is in most cases still one instruction per clock cycle.

The DSP5685x has increased the number of 36-bit accumulator registers from two to four (A, B, C, and D), each with four guard bits for overflow protection. All four accumulators can be accessed as 36-bit registers or in segments, where the segments include: a 4-bit extension register (A2, B2, C2, or D2), a 16-bit most-significant segment (A1, B1, C1, or D1), and a 16-bit least-significant segment (A0, B0, C0, or D0). Seven 16-bit data registers are available on the DSP5685x: the same 16-bit X0, Y0, and Y1 registers that the DSP568xx provides plus the A1, B1, C1, and D1 segments of the four accumulators. Data registers can be used as ALU source and destination registers for most ALU operations. Note that the 16-bit least-significant segments of the accumulators (A0, B0, C0, D0) are not considered data registers. The instruction set offers limited access to these segments; integer multiply (IMPY) and integer multiply-and-accumulate (IMAC) instructions access A0, B0, C0, and D0 as source registers, and move (MOVE.x) instructions can access all of the individual segments. Five 32-bit registers are available; the concatenation of the Y1 and Y0 registers forms the 32-bit Y register, while the concatenation of the most-significant and least-significant segments of each of the accumulators forms the A10, B10, C10, and D10 32-bit registers.

The data path of the DSP5685x is based on a 16 × 16 → 32-bit multiplier integrated with a 36-bit ALU and logic unit; this is the same configuration as found on the DSP568xx. The multiplier-ALU combination (MAC unit), together with the logic unit accepts up to three input operands, performs multiplications, additions, subtractions, logical operations, and other arithmetic operations, and produces one 36-bit result. The MAC unit supports integer and fractional multiplications; operands may be signed/signed or signed/unsigned for fractional and integer multiplication; unsigned/unsigned multiplication, however, is only supported in integer format. (The DSP568xx did not support unsigned/unsigned multiplication.)

> *As with the DSP568xx, the DSP5685x's support of signed/unsigned multiplication is an advantage. However, the lack of unsigned/unsigned fractional multiplication complicates implementation of multi-precision arithmetic when full-precision results are required.*

Unlike DSP568xx processors, DSP5685x multiply and multiply-accumulate operations are divided across two pipeline stages. Thus, multiply and multiply-accumulate operations have a latency of two instruction cycles. Arithmetic and logical operations, however, can complete in one instruction cycle. Throughput is one instruction per clock cycle for most instructions; shifting of 32-bit values is an exception, and is discussed below. Multiplier inputs are taken from the seven data registers. Outputs are written back to the same registers for 16-bit results, to the Y, A10, B10, C10, or D10 registers for 32-bit results, or to the A, B, C, or D accumulators for 36-bit results. The ALU accepts inputs from the seven data registers, from X memory, or as immediate data; outputs may be written to any of the seven data registers or directly to X memory.

Like the DSP568xx, the DSP5685x data path includes two shifters: a 36-bit accumulator shifter and a 32-bit barrel shifter. The accumulator shifter performs arithmetic and logical single-bit left/right shifts and rotate operations either on a 36-bit accumulator or on a 16-bit data register. The barrel shifter performs 1- to 32-bit arithmetic and logical left/right shifting on a 16- or 32-bit register. Shifts of 16-bit values execute with single-cycle throughput; shifts on 32-bit values execute with two-cycle throughput. Bidirectional 32-bit shift instructions are supported; the sign of the shift amount operand determines the shift direction. Results from the barrel shifter may be written to data registers or accumulators if operating on 16-bit values, or to accumulators if operating on 32-bit values. Shift amounts can be specified as 4- or 5-bit immediate data or via a value stored in a data register.

The data path supports both round-to-nearest and convergent rounding via the RND instruction, which rounds any accumulator or 32-bit register. The type of rounding is selected by a bit in the Operating Mode Register (OMR). Additionally, the 'R' suffix on ALU instructions, such as MACR, indicates rounding.

The DSP5685x features saturation circuitry similar to that found on the DSP568xx. The saturation circuitry consists of a "data limiter" and a "MAC output limiter." The data limiter protects against overflow when reading data from an accumulator by detecting signed 36-bit accumulator values whose magnitude will not fit in the selected destination. The data limiter substitutes a value of the same sign as the source accumulator with the maximum magnitude that the destination can represent. Data limiting is controlled by the syntax of the MOVE instruction reading the accumulator. For example:

```
MOVE.W  A,X:(R0)+ ;reading entire 36-bit accumulator,
                  ;limiting enabled
MOVE.W A1,X:(R0)+ ;reading 16-bit A1 portion of accumulator,
                  ;limiting disabled
```

The instruction MOVE.W specifies a 16-bit "word" data move. The implied portion of the 36-bit A accumulator used when "word" data is accessed is the 16-bit A1 segment. By specifying the entire A accumulator, as shown in the first instruction, the programmer is enabling limiting. By specifying the 16-bit A1 portion of the A accumulator, as shown in the second instruction, the programmer is disabling limiting.

The MAC output limiter is enabled by setting a bit in the Operating Mode Register (OMR). When enabled, MAC/ALU results that overflow 32 bits are saturated to the largest positive or negative value representable using 32-bit two's complement arithmetic. Note that this saturation mode is automatically disabled during shift and rotate operations, signed/unsigned multiplications, and logical operations. Additionally, the DSP5685x offers the SAT instruction, which can saturate any accumulator value and transfer the saturated value to the A, B, C, or D accumulators, or to the Y1, Y0, X0 registers.

Like that on the DSP568xx, the DSP5685x data path supports iterative division via the DIV instruction and iterative normalization via the NORM instruction. The DIV instruction calculates one quotient bit each time the instruction is executed. To produce an N-bit quotient, therefore, the DIV instruction must be executed N times. Motorola provides examples of integer and fractional division algorithms using the DIV instruction in the *DSP56854/53 User's Manual.* The NORM instruction performs one normalization iteration (a one bit left/right shift) on a value in a 36-bit accumulator each time the instruction is executed. The maximum number of iterations required to normalize a 32-bit value is 31; the actual number of shifts performed to normalize the value is written to an operand register. The DSP5685x's exponent detector can be used to compute the number of redundant sign bits in any of the seven data registers via the CLB (count leading bits) instruction; the exponent value is stored to any of the seven data registers. The CLB instruction executes in a single instruction cycle. Used in combination with the shift instructions ASLL.W (for word data) or ASLL.L (for long data), the DSP5685x can normalize 16-bit values in two instruction cycles and 32-bit values in three instruction cycles.

The DSP5685x shares the DSP568xx's dedicated bit-manipulation unit. The bit-manipulation unit is capable of testing, setting, clearing, or inverting any bits specified in a 16-bit mask. The bit-manipulation unit performs these read-modify-write operations on any core register, on a peripheral location, or directly on data memory with single-cycle throughput.

> *The bit-manipulation unit's ability to operate directly on data memory is a useful feature; however, the unit lacks explicit support for bit-field insertion or extraction.*

Memory System

Like the DSP568xx, the DSP5685x employs a Harvard memory architecture which allows up to three simultaneous memory accesses: one instruction fetch plus two data reads or one data write per instruction. Both program (P) and data (X) memory address spaces have been expanded compared to the DSP5685x: the total addressable program memory space is now 4 Mbytes, while the total addressable data memory address space is now 32 Mbytes. Separate on-chip program and data memories, each of which include RAM and ROM, occupy part of the total addressable memory space on the DSP5685x (see Table 7.9-1. for details about on-chip memory). The remaining addressable memory must be accessed off-chip via the external memory interface, described below.

Data memory is addressed via the X Address Bus 1 (XAB1) and the X Address Bus 2 (XAB2); the XAB1 bus is capable of addressing byte, word, and long data types, while the XAB2 can only address word data types. Data movement occurs along the Core Data Bus for Reads (CDBR), the Core Data Bus for Writes (CDBW), and the X Data Bus 2 (XDB2). XDB2 is a 16-bit bus, while CDBW and CDBR are 32-bit buses. In the case of a single data memory read or write, data is transferred on the CDBR or CDBW bus, respectively; the single data memory access can be a byte, word, or long data type. In the case of two simultaneous data memory reads, one data word is transferred on the CDBR bus and the other data word is transferred on the XDB2 bus; only word data types are allowed in parallel read operations. Two simultaneous data memory writes are not allowed. When the processor is executing at 120 MHz, the maximum sustainable on-chip data memory bandwidth is 240 million 16-bit words/second or 120 million 32-bit words/second for read operations and 120 million 32-bit words/second for write operations.

> *The maximum data memory bandwidth of two 16-bit words/cycle is less than might be expected given the width of the data buses available for the dual parallel read (32 bits for the CDBR and 16 bits for the XDB2).*

Program memory is addressed via the Program Address Bus (PAB) and instruction movement occurs along the Program Data Bus (PDB). The PAB is a 21-bit bus while the PDB is a 16-bit bus; the only data type for program memory is 16-bit words.

External Memory Interface

DSP5685x processors access external memory through the external memory interface (EMI), which multiplexes the internal address and data buses onto an external 24-bit address and 16-bit data bus. Unlike the DSP568xx, all of the DSP5685x internal address and data buses can access external memory. The external buses are multiplexed between program and data memory accesses. The Bus Control Unit in the EMI orchestrates external program/data, read/write accesses. Because there is only one external data bus per EMI, only one access to external memory can be made without penalty in a single instruction on current DSP5685x devices. If a dual parallel read instruction accesses both on-chip (internal) and external memory, the DSP5685x allows either the first or second read to access internal memory, while the other read accesses external memory; in contrast, the DSP568xx requires that the first read accesses external memory and second read accesses internal memory. System-on-a-chip (SoC) designers can specify an additional EMI interface; Motorola states that adding another EMI would allow two simultaneous external memory accesses per instruction. With a single EMI, such as is present on the DSP56854, the maximum peak and sustainable external memory bandwidth, assuming zero wait states, is 120 million 16-bit words/second when the processor is executing at 120 MHz.

To adapt the speed of the EMI to slower external memory elements, programmed wait states are supported. Program and data memory wait states can be independently configured between 0 and 15 clock cycles. Externally requested wait states are not available.

> *As with the DSP568xx, the DSP5685x lacks support for externally requested wait states; this may complicate system design in some applications.*

Address Generation Unit

The address generation unit (AGU) of the DSP5685x is a considerably expanded version of the AGU found in the DSP568xx. The DSP5685x AGU adds two 24-bit address registers, R4 and R5, and increases the width of the R0-R3 address registers from 16 to 24 bits. The stack pointer register (SP) and offset register (N) have also been expanded from 16 to 24 bits, and a 16-bit secondary offset register (N3) has been added. The 24-bit N register can also be used as an extra pointer register, and is supported by additional instructions for fast address calculation. Additionally, modulo register M01 and registers R0, R1, and N are shadowed on the DSP5685x.

Address arithmetic is performed by a 24-bit primary arithmetic unit and a 24-bit secondary arithmetic unit. The primary arithmetic unit is similar to the DSP568xx's modulo arithmetic unit; the secondary arithmetic unit is similar to but more powerful than the DSP568xx increment/decrement unit. The primary arithmetic unit performs calculations for indexed addresses, post-updating of addresses, and AGU arithmetic instructions. The primary arithmetic unit also performs linear or modulo arithmetic, while the secondary arithmetic unit performs only linear arithmetic incrementing, decrementing, or post-updating the R3 register with the contents of the N3 register. The AGU provides addresses for all three of the DSP5685x address buses and provides up to two 24-bit addresses per instruction cycle.

The DSP5685x supports memory-direct, register-direct, and register-indirect addressing modes. Immediate data is also supported. Immediate data formats include: long immediate data (32 bits), immediate data (16 bits), and short immediate data (5-7 bits). Instructions using long immediate data require two additional program words and two additional instruction cycles to complete, while those using immediate data require one additional program word and one additional instruction cycle to complete.

Memory-direct addressing modes include: absolute long (24-bit), absolute (16-bit), absolute short (6-bit), and a special peripheral-short (6-bit) addressing mode designed for fast access to memory-mapped peripheral registers. The memory-mapped peripheral registers (the number of which depends on the number and type of peripherals installed) can be located anywhere in the data memory space, and the peripheral short addressing mode provides efficient access to 64 of these data memory locations. Use of the peripheral short addressing mode eliminates one program word and one clock cycle for move and bit-manipulation instructions.

Register-direct addressing is supported in both the AGU and in the data ALU; each unit has specific arithmetic and logical instructions that make use of this addressing mode.

Register-indirect addressing modes include: no update; post-increment, post-decrement, or update by the contents of the offset registers, N and N3; indexed by 3-, 6-, 16-, or 24-bit immediate offsets; or indexed by the contents of the offset register, N. The software stack can be conveniently accessed using stack pointer (SP)-relative indexing with 3-, 6-, 16-, or 24-bit immediate offsets. Pre-modifications of the address registers are not supported.

Register R0 or registers R0 and R1 together can be assigned to operate under modulo addressing where the size of the circular buffer is specified via the modulo register M01. Hence, two circular buffers (both having the same size) can be simultaneously active, or R0 and R1 can act on the same buffer (e.g., for implementing a FIFO queue). Modulo addressing is enabled by loading M01 with the size of the circular buffer minus one for modulo addressing on only R0, or the size of the circular buffer minus one plus 0x8000 for modulo addressing on R0 and R1. In either case the size of the circular buffer cannot exceed 16Kx16. The buffer must be aligned on an address that is evenly divisible by *k*, where *k* is the smallest power of two that is greater than or equal to the size of the circular buffer. Modulo addressing is disabled (linear addressing is enabled) when 0xFFFF is loaded in the modulo (M01) register.

The AGU shadows registers R0, R1, N, and M01 to facilitate fast interrupt service. These four registers are automatically swapped with their corresponding shadow registers upon entering fast interrupt processing. Details on the fast interrupt service routine can be found under *Interrupts* in the *Execution Control* section of this analysis. The shadow registers are also available during general interrupt processing; the SWAP instruction swaps the values in the R0, R1, N, and M01 registers with their shadow counterparts.

Like the DSP568xx, the address generation unit of the DSP5685x is not capable of bit-reversed addressing.

The lack of bit-reversed addressing in the DSP5685x complicates FFT implementations that require natural order results.

The DSP5685x supports a relatively rich set of addressing modes for a processor with a 16-bit instruction word. However, there are restrictions regarding the use of these modes for instructions with parallel moves (see the Parallel Move Support section of this analysis for further details).

Even though two circular buffers can be active at a time only one size circular buffer is supported; this is a significant limitation. Additionally, only one circular pointer (R0 or R1) can be updated within a single instruction. This may complicate implementing some algorithm kernels with single-instruction hardware loops.

> *Register-indirect addressing indexed by an immediate value can be effectively utilized by high-level language compilers for accessing local variables in the software stack. For example, add, subtract, increment, decrement, and move instructions accept stack operands.*
>
> *Many new address calculation instructions, such as simple shifting, add with shift, and test and compare, have been added to the AGU unit for more efficient pointer calculations and manipulations.*

Pipeline

The DSP5685x's eight-stage pipeline is significantly deeper than the three-stage pipeline of the DSP568xx. The DSP5685x's pipeline is divided into the following stages: prefetch 1, prefetch 2, instruction fetch, instruction decode, address generation, operand prefetch 2, execute and operand fetch, and execute 2. Write-back occurs in the last two stages. Memory read operations are spread across three clock cycles, allowing slower memory to be used while maintaining high instruction throughput.

To ease the additional programming complexity introduced by the deeper pipeline, the DSP5685x pipeline is interlocked. The interlocking hardware resolves only a subset of the possible pipeline dependencies that may arise, however. Dependencies that are not resolved by the interlocking hardware are handled by the assembler, which issues warnings and inserts NOP instructions, or, in a limited number of cases, issues an error message. Thus, pipeline hazards on the DSP5685x can be classified into three categories: those that the interlocking hardware resolves by stalling the pipeline, those that the assembler resolves by inserting NOPs, and those that will not be resolved/assembled, which Motorola calls "restricted instruction sequences."

The addition of NOPs as a means to resolve pipeline dependencies has a potentially negative affect on both code size and execution time. To minimize the negative effect of NOPs on code size, the interlocking hardware resolves pipeline dependencies in instruction sequences that are potentially code-size critical. For example, data dependencies in a hardware DO loop are resolved by the interlocking hardware. In the case of an instruction that modifies a control register, the assembler inserts NOPs after the register operation to ensure mode changes are completed before subsequent instructions are executed. The assumption is that mode changes are relatively infrequent and the NOPs associated with them will have little effect on code size. Since the interlocking hardware resolves pipeline dependencies by stalling the pipeline one or more instruction cycles, there is no difference in execution time between inserting a single NOP instruction and stalling the pipeline one cycle.

Pipeline dependencies arise in the following cases:

- A data register that is used as the destination in a multiplication, multiply-accumulate, 16-bit shift, 32-bit shift, or shift-with-accumulate instruction is used as a source for a move or parallel move in the next instruction. This dependency is resolved by the interlocking hardware stalling the pipeline.

- A data register that is used as the destination in a multiplication, multiply-accumulate, 16-bit shift, 32-bit shift, or shift-with-accumulate instruction is used as a source for a multiplication, multiply-accumulate, 16-bit shift, 32-bit shift, or shift with accumulate instruction in the next instruction. This dependency is resolved by the interlocking hardware stalling the pipeline.
- An address register in the AGU is used as a pointer, an offset, an AGU operand, or in a transfer-address instruction (the TFRA instruction transfers one AGU register to another), and one of the two preceding instructions modified the address register using a move or bit-manipulation instruction. An exception to this rule is the case where the preceding instructions write immediate data to the register, in this case there is no dependency. This dependency is resolved by the interlocking hardware stalling the pipeline.
- Modifications to certain mode registers usually take two instruction cycles to take effect. This dependency is resolved by the assembler inserting NOPs.
- The LC register must be loaded a minimum of two instruction cycles before executing a DOSLC instruction. This dependency is resolved by the assembler inserting NOPs.

> *While the interlocking hardware and assembler together ensure correct program execution, they may do so at the expense of pipeline stalls and NOPs. To produce optimal assembly code, programmers must become familiar with all pipeline dependencies. The complete list of dependencies and restricted instructions are not supplied with the DSP56800E Family Manual, but are available in a separate document. Additionally, code that was optimized for the DSP568xx's non-interlocked pipeline may need significant revisions to achieve optimal performance on the DSP5685x.*

Instruction Set

Table 7.9-2 and Table 7.9-3 summarize DSP5685x registers and instructions, respectively.

Assembly Language Format

The DSP5685x implements a superset of the DSP568xx instruction set and is backward assembly source code compatible with the DSP568xx. The opcode-operand style instruction format is retained, with several new instruction variants to support the DSP5685x's three data types and added capabilities. For example, the DSP5685x instruction set supports an instruction that specifies an ALU/MAC operation with up to two parallel moves; this instruction is very similar to the corresponding DSP568xx instruction, but has added flexibility, as illustrated by the following:

```
MACR  Y1,X0,A X:(R0)+N,Y1 X:(R3)+,X0      ;DSP568xx/5685x
MACR  Y1,X0,A X:(R0)+N,Y1 X:(R3)+N3,X0    ;DSP5685x only
```

The "R" suffix with the MAC instruction indicates MAC with rounding. The rounding mode (convergent or round-to-nearest) is specified by a bit in the Operating Mode Register (OMR). If rounding were not desired (i.e., truncation is desired), the MAC instruction without the "R" suffix would be used instead. The first instruction multiplies Y1 with X0, adds their product to the A accumulator, and rounds the result. In parallel, it moves the data memory value pointed to by R0 to register Y1, post-updates R0 by the contents of the N register, moves the contents of the data memory location pointed to by R3 to X0, and post-increments R3 by one. This instruction format is valid on both the DSP568xx and the DSP5685x. A variation on this instruction is shown in the second instruction. The second instruction performs all of the same operations on the same registers as the first instruction, except that in this case, the R3 register is post-updated by the contents of the N3 register. The ability to post-update the R3 register by the contents of the N3 register is only supported on the DSP5685x and adds considerable flexibility to the limited dual parallel move instructions of the DSP568xx.

Registers	Width	Purpose
A, B, C, D	36 bits	Accumulators
X0, Y0, Y1	16 bits	ALU/MAC inputs/outputs, general-purpose
R0-R5	24 bits	Address registers
SP	24 bits	Stack pointer register
N	24 bits	Primary address offset register and address register
N3	16 bits	Secondary address offset register
M01	16 bits	Modulo register
LA, LA2	24 bits	Loop address registers
LC, LC2	16 bits	Loop counter registers
FIRA	24 bits	Fast interrupt return address
FISR	13 bits	Fast interrupt status register
OMR, SR	16 bits	Operating Mode and Status registers

TABLE 7.9-2. DSP5685x register summary.

Class	Instructions
Arithmetic	Absolute value; clear; increment; decrement; negate; round; saturate; transfer; add byte, word, long with carry; subtract byte, word, long with carry; test byte, word, long, accumulator.
Multiplication	Signed multiply/multiply-accumulate (with or without rounding and/or negation); signed/unsigned multiply/multiply-accumulate (with or without rounding and/or negation); signed, signed/unsigned, unsigned/unsigned integer multiply
Logic	*And, or, exclusive-or, not* with register or with immediate data; count leading 0s or 1s
Shifting	Arithmetic and logical shift left/right: one-bit, multi-bit, multi-bit bidirectional; arithmetic and logical multi-bit shift right and accumulate
Rotation	Rotate left/right one bit: word, long
Conditional Execution	Conditionally transfer from a selected register to an accumulator, with an optional transfer of the R0 register to R1
Comparison	Compare accumulators, registers, immediate data, or memory; test accumulator, 16-bit register, or memory location
Looping	Single- and multi-instruction nested or single loops, break out of loop unconditionally
Branching	Relative and absolute branch (both conditional and unconditional), branch if selected bits are set/clear, delayed branch, delayed jump
Move	Move signed or unsigned byte, word, long; swap shadow registers
Subroutine Call	Jump to subroutine, return from subroutine, return from interrupt, delayed return from interrupt, delayed return from fast interrupt, delayed return from subroutine
Bit Manipulation	Bit field test and set, test and clear, or test and change, test high (all selected bits are set), test low (all selected bits are low)
Special Function	Division iteration, normalization iteration, WAIT and STOP modes, software interrupt, enter DEBUG mode, generate DEBUG event, align SP for long memory access

TABLE 7.9-3. DSP5685x instruction set summary. New instructions (relative to the DSP568xx) are underlined.

Parallel Move Support

As shown in the above examples, DSP5685x arithmetic instructions support operand-unrelated parallel moves. Single parallel move instructions allow an arithmetic operation and one memory access to be completed within one instruction. Dual parallel move instructions allow an arithmetic operation and two memory accesses to be completed within one instruction. In both cases the parallel data moves are unrelated to the arithmetic operation being executed; however, because of the limitations of the 16-bit instruction word size, the processor places restrictions on which arithmetic instructions support single and dual parallel moves, where the operands for the arithmetic operation can come from, which address registers can be used to reference memory, as well as the destinations for the parallel move(s) and the arithmetic result. Most of these restrictions relate to dual parallel moves. The most significant of these restrictions are:

- When performing two parallel reads, only address register R3 can be used for the second parallel read; R0, R1, or R4 may be used for the first parallel read.
- In dual parallel read instructions, only registers Y0 and Y1 can be used as destinations for the first parallel read, and only register X0 and C can be a destination for the second parallel read.
- In dual parallel read instructions, the first address register (R0, R1 or R4) can only be modified by post-incrementing it by one or by the contents of the N register, while the second address register (R3) can only be post-incremented/decremented by one or updated by the contents of the N3 register.
- While virtually all ALU/MAC instructions allow one parallel move (read or write), only the following instructions allow two parallel moves (reads only): add, subtract, multiply-accumulate with or without rounding, multiply with or without rounding, transfer register to register, clear accumulator, arithmetic shift left/right one bit, and move word.

Orthogonality

The orthogonality of the DSP5685x instruction set is slightly improved compared to the DSP568xx. For example, dual parallel move instructions, while still restrictive, are more flexible on the DSP5685x as demonstrated by the example shown in *Assembly Language Format* and as discussed in *External Memory Interface* above. In general, for non-parallel move instructions, any register that can contain the data type being operated on can be used as a source or destination. This is similar to the DSP568xx, except that the DSP5685x has more registers to choose from. As with the DSP568xx, two reads using register-indirect addressing can be performed in parallel on the DSP5685x, but only one parallel write is supported. The least orthogonal aspects of the DSP5685x relate to dual parallel move instructions, discussed in more detail above, in *Parallel Move Support.*

Although very efficient parallel code can be written for the processor, we expect that even experienced DSP5685x programmers will

> *be forced to return to the user's manual on a regular basis to remind themselves of the register usage restrictions.*

Execution Times

Most DSP5685x MAC/ALU instructions execute with single-cycle throughput. Branches, subroutine calls, and return instructions take two to four cycles to execute. Shift instructions take two cycles to execute if the shift amount is greater than 16 bits. Delayed branches have a two-instruction delay slot. Single- and multi-instruction hardware loops take three instruction cycles to start.

Instruction Set Highlights

The DSP5685x provides data ALU and bit-manipulation instructions that operate directly on memory without affecting the core registers. For example, the instruction

```
INC.W X:$100
```

increments the word stored in X memory location 0x100 by one. Byte and long data types can also be operated on in a similar fashion. The operands for these instructions can be addressed via a 16- or 24-bit absolute address, the stack pointer or primary address offset/pointer register indexed with a 6- or 16-bit displacement, an AGU address register with or without a 16-bit displacement, or can be contained in one of the seven data ALU registers. Data ALU instructions require three instruction cycles to execute if the operand is addressed via a 16-bit absolute address or an AGU address register, or four instruction cycles to execute if the operand is addressed via a 24-bit absolute address, the stack pointer indexed by a 6-bit displacement, or an address register indexed by a 16-bit displacement. Bit-manipulation instructions require three instruction cycles to execute if the operand is addressed via a 6- or 16-bit absolute address, an AGU address register, the stack pointer indexed by 6- or 16-bit displacement, or an address register indexed by a 16-bit displacement. The bit-manipulation instructions require four instruction cycles to execute if the operand is addressed via a 24-bit absolute address. These execution times assume the memory access is performed to internal memory or to external memory with zero wait states.

> *The ability to operate on operands directly in memory is very useful for manipulating temporary variables stored in data memory and memory-mapped registers without disturbing the contents of internal registers. This approach supports both microcontroller functionality and high-level language compilers.*

Other noteworthy DSP5685x instructions include:

- Bit-field test, set, clear, and change instructions (which allow groups of up to 16 bits to be manipulated at once)
- Branch if selected bits are set or clear
- Normalization iteration instruction

- Division iteration instruction
- Conditional transfer instruction for conditional movement of data between registers
- Delayed and non-delayed branches, jumps, and return instructions
- Software interrupts at any of five interrupt levels
- Single-cycle exponent detect

Like the DSP568xx, the DSP5685x's bit manipulation instructions can simplify programming and increase performance in decision-intensive applications, such as state machines.

Execution Control

Clocking

Unlike DSP568xx processors which use a 2X master clock, DSP5685x processors use a 1X master clock. Thus, a 120 MHz master clock for the DSP5685x corresponds to an instruction execution rate of 120 MIPS.

The relationship of the clock rate to the instruction execution rate has changed between the DSP568xx and the DSP5685x. Instruction throughput on the DSP568xx is one instruction per two clock cycles (2X master clock rate) compared to the DSP5685x's throughput of one instruction per clock cycle (1X master clock rate). Thus, the DSP5685x is significantly faster (relative to the DSP568xx) than a comparison of the two processor's clock rates would indicate.

DSP5685x processors include an on-chip clock synthesis module similar to that found on DSP568xx processors. For details on the clock synthesis module, please refer to the *Execution Control* section of the DSP568xx analysis in Section 7.8.

Hardware Looping

Compared to the DSP568xx, hardware looping support has been enhanced in the DSP5685x via changes affecting the loop address and loop counter registers. The DSP5685x includes two 24-bit loop address registers (LA and LA2) compared to the single 16-bit loop address register (LA) on the DSP568xx. Additionally, the DSP5685x includes two 16-bit loop counter registers (LC and LC2) compared to a single 13-bit loop counter register (LC) on the DSP568xx. As with the DSP568xx, the REP instruction repeats a single instruction and is not interruptible, while the DO instruction repeats multiple instructions and is interruptible. Unlike the DSP568xx, however, the DSP5685x provides hardware support for nested DO loops, and supports 16-bit repetition count values compared with 13-bit repetition values on the DSP568xx. Additionally, the DSP5685x adds the DOSLC instruction; the DOSLC instruction assumes the LC register is preloaded and begins executing the hardware loop immediately, whereas the DO instruction loads the LC register with the loop count then begins executing the hardware loop.

There are two ways to enter a single-instruction hardware loop, depending on the repetition count of the loop. If the repetition count is smaller than 64, then it can be encoded as short immediate data in the instruction word. As a result, the loop can be entered by using the following instruction:

```
REP         #N          ; N < 64
```

However, if the repetition count is larger than 63, the following method must be used:

```
MOVE        #N,LC       ; N < 65,536
REP         LC
```

The first instruction loads a 16-bit immediate value into the loop-count (LC) register and the second instruction starts the repeat loop. Similarly, a multi-instruction loop requires an extra instruction to load the loop counter if short immediate data cannot be used (i.e., if the number of repetitions is greater than 63).

The maximum repetition count which can be stored in the LC and LC2 registers is 65,536. If the LC register is loaded with zero, the loop is skipped with the REP and DO instructions. If the LC register is loaded with zero or a negative value, the loop is skipped with the DOSLC instruction. The REP and DO instructions have a latency of two and three instruction cycles, respectively, when the LC register is loaded with an immediate value and the last address in the DO loop can be specified with 16 bits (i.e., the loop is entirely contained in the first 64 Kbytes of program memory). Alternatively, any data path register or address register can be used to load the LC register for REP and DO instructions; doing so however, increases latency to five and seven instruction cycles respectively. Additionally, if the last address in a DO loop is specified with more than 16 bits (i.e., the loop extends beyond the first 64 Kbytes of program memory), latency is increased by an additional instruction cycle.

A single-instruction REP loop can be nested inside a multi-instruction DO loop; however, a REP loop cannot be used to repeat an instruction that accesses program memory, an interrupt instruction, or a multi-word instruction.

With the addition of hardware support for two-level nesting of DO loops, nesting additional levels has become more complicated. DO loops nested beyond two levels require not only saving the loop count and the end address of the outer loop onto the software stack (as is the case when nesting two DO loops on the DSP568xx), but also the start address of the outer loop as well (the start address of the outer loop is normally automatically saved onto the hardware stack by the DO instruction, but the hardware stack is only two deep). Additionally, each DO instruction beyond two levels causes a non-maskable hardware stack overflow interrupt.

> *The additional loop nesting depth is an improvement over the DSP568xx; it would have been a further advantage, however, if Motorola had also increased the depth of the hardware stack.*

Interrupts

As with the DSP568xx, interrupt requests on the DSP5685x can be generated from the reset pin, from two external interrupt pins, from on-chip peripherals, or from within the core. Compared with the DSP568xx, the DSP5685x provides two additional programmable interrupt priority levels, for a total of five, and allows the interrupt vector table to be located anywhere within the program memory space. Software interrupts can be generated at any of the five interrupt levels. Additionally, the DSP5685x provides hardware support for fast interrupt processing via the FIRA (fast interrupt return address) register, the FISR (fast interrupt status register) register, and the shadow registers in the AGU (see *Address Generation Unit* in the *Architecture* section of this analysis for more details on the shadow registers). Fast interrupt processing automatically swaps the shadow registers and pushes the Y register onto the software stack, leaving the service routine free to immediately use the R0, R1, N, M01, Y0, and Y1 registers. The return from fast interrupt (FRTID) instruction reverses the swap and has a two-instruction-word delay slot, allowing service routines to initiate return while continuing to execute instructions in the delay slot. Fast interrupt processing reduces the latency from the time the interrupt is acknowledged until the first instruction of the service routine begins execution; general interrupts have a latency of nine instruction cycles while fast interrupts have a latency of five instruction cycles.

Several interrupts can be generated by the JTAG/EOnCE on-chip debug logic including Step Counter Interrupt, Transmit Interrupt, Receive Interrupt, Trace Buffer Full Interrupt, and Breakpoint Unit 0 Interrupt.

Other aspects of interrupts and interrupt handling, such as peripheral interrupts, are similar to those of the DSP56F80x processors with the exception that memory addresses on the DSP5685x are 24 bits. See *Interrupts* in the *Execution Control* section of the DSP568xx analysis in Section 7.8 for further details.

Stack

Like DSP568xx processors, DSP5685x processors feature a two-level hardware stack; however, the DSP5685x stack is expanded to 24 bits per level from the DSP568xx's 16 bits. The hardware stack is used for hardware DO loops; it stores the address of the first instruction of a DO or DOSLC loop. The REP instruction does not affect the hardware stack. Although a stack exception interrupt detects hardware stack overflow, underflow is not detected.

The DSP5685x's software stack is supported via a dedicated 24-bit stack pointer register, compared to 16 bits on the DSP568xx. The software stack is used for passing parameters, storing return addresses, holding local variables for subroutines, etc. In addition, interrupts use the software stack for storing the return address and the status register when entering an interrupt service routine. Transfers to the stack have been made more efficient, both in terms of latency and code density, with the DSP5685x's 32-bit move instructions. For example, the following DSP568xx sequence,

```
LEA   (SP)+       ;increment SP: one word, one cycle
MOVE  Y0,X:(SP)+  ;save lower 16 bits: one word, one cycle
MOVE  Y1,X:(SP)   ;save upper 16 bits: one word, one cycle
```

which saves the 32-bit value in the Y register onto the stack, can be reduced to the following sequence on the DSP5685x:

```
ADDA    #2,SP     ;increment SP: one word, one cycle
MOVE.L Y,X:(SP)   ;save 32 bits: one word, one cycle
```

The above DSP5685x sequence assumes that the stack pointer is long-memory-access aligned; Motorola recommends that new applications developed for the DSP5685x initialize the stack pointer for long memory accesses (using the ALIGNSP instruction), and that only 32-bit values be pushed or popped when adding or removing data from the stack, using the MOVE.L instruction. Note that 8-bit, 16-bit, and 32-bit values can still be accessed from the stack, but only 32-bit values should be pushed or popped from the stack as these instructions modify the stack pointer, which should remain long-memory-access aligned.

Although a software stack can be implemented on any DSP processor, only a few DSPs feature a dedicated stack pointer which is automatically updated upon stack operations, as found on the DSP5685x. In addition, many DSP5685x instructions support stack-pointer-relative addressing, easing compiler development.

Bootstrap Loading

Upon reset, the MA and MB bits of the Operating Mode Register (OMR) are loaded from the external mode select lines MODA and MODB. The DSP5685x then begins execution at the program memory address selected by the state of the MA and MB bits in the OMR and the address of the reset vector. Different vector addresses can be provided for different reset sources; e.g., a Computer Operating Properly (COP) reset and a reset from the $\overline{\text{RESET}}$ pin can have different reset vectors. After the first instruction is fetched and propagates through the pipeline, the processor enters, or returns to, the normal processing state.

Peripherals

DSP5685x processors inherit their peripherals directly from the DSP568xx family. DSP56854/53 peripherals include a Quad Timer module, a Serial Communication Interface (SCI), two Synchronous Serial Interfaces (SSI), a Serial Peripheral Interface (SPI), and a Computer Operating Properly/Real-Time Interrupt (COP/RTI) module. Only the DSP56854 has an External Memory Interface (EMI). For details on the peripheral devices, please refer to the *Peripherals* section of the DSP568xx analysis in Section 7.8.

On-Chip Debugging Support

Compared with the DSP568xx, the DSP5685x family features expanded debugging support. The DSP568xx IEEE-1149.1 JTAG/OnCE port has been replaced by the JTAG/EOnCE port on the DSP5685x. Like the OnCE module on the DSP568xx, the Enhanced On-Chip Emulation (EOnCE) module of the DSP5685x allows interaction between the debugging environment and the processor core: examination of core registers, on-chip peripheral registers, and memory; setting breakpoints on program or data memory; and stepping or tracing instructions. Stepping is supported on the DSP5685x on up to 1,048,576 (2^{20}) instructions, up from 256 on the DSP568xx. Breakpoint logic is more powerful on the DSP5685x, which features two breakpoint compare units, Breakpoint Unit 0 (BP0) and Breakpoint Unit 1 (BP1), compared to the DSP568xx's one breakpoint unit. The two breakpoint units allow the user to program breakpoint actions based on complex combinations or sequences of events. Some of the possibilities include break on BP0 and/or BP1 condition(s) met, break on the n^{th} (up to 2^{16}) occurrence of BP0 condition after BP1 condition met, and break after counting n (up to 2^{16}) instruction cycles after BP0 and/or BP1 condition(s) met. Breakpoint actions can range from generating interrupts, to halting the core and entering the debug mode, to starting or halting the trace buffer. Another addition to the EOnCE features is the Core Events terminal, which communicates EOnCE state information to the user in real time. Some examples of viewable events are: the step counter is started, the step counter reaches zero, the trace buffer capture is started, the trace buffer is full, the core enters a debug session, and any breakpoint unit action.

> *The debugging features of the DSP5685x's EOnCE are more powerful than those found on the DSP568xx; the breakpoint capability is noteworthy when compared to the DSP568xx as well as to other DSPs like the Texas Instruments TMS320C2xxx.*

Power Consumption and Management

The DSP5685x family operates from a dual voltage supply of 1.8 volts for the core and 3.3 volts for I/O. Motorola states that typical power consumption for a DSP56854 core plus on-chip memory is projected to be 65.7 mW at 1.8 volts when executing at 120 MHz.

> *The DSP56854's projected power consumption of 65.7 mW at 120 MIPS is significantly lower than that of the DSP56824 (60 mW when executing 35 MIPS). This improvement is largely due to the migration to a more aggressive fabrication process.*

DSP5685x processors provide WAIT and STOP modes to reduce power consumption when no processing is required. The operation of the STOP and WAIT modes are identical to those of the DSP568xx; please refer to the *Power Consumption and Management* section of the DSP568xx analysis in Section 7.8 for further details. Motorola states that the power consumption for the DSP56854 when in STOP mode, with all of the peripherals (including the PLL) disabled, is projected to be 9.0 µW at 1.8 volts.

Benchmark Performance

The DSP5685x has been benchmarked with the BDTI Benchmarks™. Overall benchmark results for all benchmarked processors are presented in Chapter 8, *BDTI Benchmark™ Results*. We summarize and analyze the DSP5685x's benchmark performance in the paragraphs below. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not reflect the processor's instruction cycle rate; therefore, lower instruction cycle counts imply a more powerful architecture, but do not imply faster speed. Next we discuss benchmark execution times and cost-execution time products, indicating processor speed and cost-performance, respectively. We then discuss the processor's energy consumption, which reflects the energy consumed by the processor in order to perform a task. Finally, we discuss the processor's memory usage. We divide the memory usage discussion into three parts: Control benchmark memory usage, overall benchmark program memory usage, and benchmark data memory usage

Execution Performance

- **Instruction cycle counts:** As illustrated in Figure 8.1-13 in Chapter 8, *BDTI Benchmark™ Results*, the DSP5685x total normalized instruction cycle count is the fourth-highest, at about 35% above the average for all of the processors. Only the DSP568xx, ADSP-219x, and ADSP-219x-C (the ADSP-219x with cache preloaded) have higher total normalized cycle count results.

  The DSP5685x has a total normalized cycle count result that is roughly 5% below that of its predecessor, the DSP568xx. A major reduction in the cycle counts is not be expected for benchmarks such as the **Single-Sample FIR** or the **Vector Dot Product**; however; the code for implementing these benchmarks is very similar on the two processors. The improvements in the architecture of the DSP5685x compared with the DSP568xx become apparent in more complicated benchmarks such as the Viterbi and FFT benchmarks. The DSP5685x has cycle counts on both of these benchmarks that are roughly 25% lower than those of the DSP568xx.

  For the **Real Block FIR**, **Complex Block FIR**, and **Vector Add** benchmarks, the DSP5685x instruction cycle counts are relatively high compared with those of the other processors. In particular, dual-MAC processors such as the TMS320C62xx achieve lower cycle counts by performing calculations on more than one sample in one instruction cycle. In contrast, the DSP5685x can fetch two operands and perform a single multiply-accumulate operation in one instruction cycle, which is the maximum throughput for a single-MAC processor.

  Compared with the DSP568xx, the cycle count of the DSP5685x is somewhat lower for the Real Block FIR benchmark, due to the extra index register on the

DSP5685x. On the DSP568xx, it is necessary to reload the filter coefficient base address from memory after computing each output sample. In the DSP5685x, on the other hand, the extra index register can be used to reset the address register and avoid explicitly reloading the address.

For the **Complex Block FIR** benchmark, the DSP5685x has a lower cycle count in comparison with that of the DSP568xx because of the added index register, and because of the DSP5685x's ability to store long (32-bit) data. In the benchmark code, real and imaginary data are packed into adjacent 16-bit words. Although the DSP568xx can read two 16-bit quantities packed into a 32-bit word, it does not have a similar ability to store two 16-bit quantities. The DSP5685x, in contrast, can store a 32-bit quantity with one write operation, thus lowering its cycle count on this benchmark.

The DSP5685x has noticeably higher cycle counts than the DSP568xx for the **Vector Maximum** benchmark; the DSP5685x cycle count result is roughly 30% higher than that of the DSP568xx. The pipeline on the DSP5685x is deeper than that of the DSP568xx, so copying address registers (to record the location of the maximum) on the DSP5685x requires more cycles than on the DSP568xx.

The DSP5685x has the second-highest instruction cycle count on the **FFT** benchmark, at about 85% above the average. Like the DSP568xx, the DSP5685x has no hardware support for bit-reversed addressing. Approximately 10% of the total cycles for the FFT benchmark on the DSP5685x come from implementing bit-reversal in software. The cycle count for the DSP5685x for the FFT benchmark is significantly lower, however, than that of the processor with the highest cycle count, the DSP568xx. Compared to the DSP568xx, the DSP5685x has two additional data accumulators, two additional address registers, and one additional index register that contribute to its lower cycle counts. The core of the FFT algorithm is the butterfly; in each butterfly, four values are computed using a multiply and an accumulate. The four computed values require four data operands and two coefficients. The availability of four accumulators (one for each value) and the additional address registers available for accessing operands reduces the DSP5685x's cycle count result by eliminating the overhead required by the DSP568xx for swapping values in and out of accumulators and address registers.

For the **Viterbi** benchmark, the DSP5685x has the fifth-highest instruction cycle count, at roughly 90% above the average. The processors with higher cycle counts on this benchmark are the DSP568xx, ADSP-218x, ADSP-219x, and ADSP-219x-C. As mentioned above, long moves have been introduced on the DSP5685x, which improves the Viterbi cycle count relative to that of the DSP568xx by allowing loads or stores of two 16-bit data operands in one instruction cycle. In addition, the DSP5685x has more data registers compared with the DSP568xx; unlike the DSP568xx, the DSP5685x has enough data registers to compute the add-compare-select operation without reloading operands from memory.

On the **Control** benchmark, like the Vector Maximum benchmark, the DSP5685x's cycle count result is generally unremarkable but is somewhat higher (roughly 10% higher) than that of the DSP568xx. Note, however, that the Control benchmark is optimized for minimum memory usage, not minimum cycle counts. In this benchmark, there are many branches, subroutine calls, and returns from subroutines. The deeper pipeline of the DSP5685x causes the cycle count for the Control benchmark to increase compared with that of the DSP568xx.

- **Execution times:** The DSP56853 has a moderately high instruction cycle rate (120 MHz) in comparison to the other conventional DSP processors benchmarked here. Compared to some of the VLIW-based processors, such as the Texas Instruments TMS320C6203, TMS320C64xx, and MSC8101, its instruction cycle rate is significantly slower. The DSP56853 moderate instruction cycle rate combines with its relatively high cycle counts to result in a total normalized execution time that is about 25% slower than the average of all benchmarked fixed-point processors, as presented in Figure 8.2-13.
- **Cost-execution time**: In terms of total normalized cost-execution time product (presented in Figure 8.3-13), the DSP56853's $3.75 price tag (quantity 10,000), the lowest of all processors benchmarked, offsets its relatively slow execution time results. The total normalized cost-execution time product for the DSP56853 is exceptionally low, and is by far the best result among all of the processors benchmarked in this report. The total normalized cost-execution time of the DSP56853 is only about one-fifth the average for all benchmarked fixed-point processors.
- **Energy consumption:** The DSP56854 has the second-lowest power consumption of any of the processors benchmarked in this report; only the DSP56824 has lower power consumption. As presented in Figure 8.4-13B, the DSP56854's low power consumption combines with the processor's moderate execution times to give the DSP5685x better than average total normalized energy consumption, at roughly 45% below the average for all benchmarked fixed-point DSP processors.

Memory Usage

The focus in the memory usage analysis is on the Control benchmark that, unlike other benchmarks, is optimized for minimum memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP code. Finally we discuss constant and non-constant data memory usage.

- **Control benchmark memory usage:** As illustrated in Figure 8.5-9A, the DSP5685x achieves the lowest total memory usage of all of the benchmarked DSP processors on the Control benchmark. In comparison to the fixed-point processors in this report, the DSP5685x has a total normalized result that is roughly 35% below the average. The key features which enable the DSP5685x to achieve good code density on control-oriented code include its 16-bit instruction word width, PC-relative branches, and support for short immediate data. The second-lowest total memory usage for the Control benchmark is achieved by the DSP568xx, which requires slightly more program memory. The DSP5685x has the advantage due to its ability to load and store long (32-bit) data in one instruction.

- **Program memory usage:** As Figure 8.5-13 illustrates, the total normalized program memory usage of the DSP5685x is the third lowest of all benchmarked DSP processors, bettered only by the Analog Devices ADSP-218x and the Texas Instruments TMS320C54xx. Despite its short, 16-bit instruction word width, the instruction set of the DSP5685x is relatively powerful (though non-orthogonal) and allows more operations to be performed per instruction than most other conventional DSP processors. There are, for example, many arithmetic operations which allow parallel fetch or store of two 16-bit data words. Such features enable the processor to have the second-lowest cycle counts on the Real Block FIR and Complex Block FIR benchmarks, and the lowest on the Control, Single-Sample FIR, Vector Add, Vector Dot Product, and Bit Unpack benchmarks. In contrast, the DSP5685x has the third-largest program memory usage (after the DSP568xx and Texas Instruments TMS320C67xx) for the Viterbi benchmark. Compared with the DSP568xx, the DSP5685x uses less program memory on the Viterbi benchmark because the DSP5685x can do long (32-bit) loads and stores. On the DSP568xx, many of these loads require an additional instruction, which increases program memory usage. Also, on the DSP5685x there are enough data registers so that data can be remained for successive additions and subtractions without refetching as required on the DSP568xx.
- **Data memory usage:** Like the DSP568xx, the DSP5685x's constant data memory usage is as expected for a 16-bit fixed-point DSP processor. Total normalized constant data memory usage is shown in Figure 8.5-14.

  As with the DSP568xx, the DSP5685x's non-constant data memory usage is generally as expected for a 16-bit fixed-point DSP processor. The DSP5685x has a higher non-constant data memory usage on the Control benchmark than other benchmarked fixed-point processors. The reason is that, in the case of a subroutine call, the processor stores the return address and the processor's status word onto the software stack. Most fixed-point processors either do not store the status word or provide hardware stacks or shadow registers for saving registers and return addresses. Total normalized non-constant data memory usage is shown in Figure 8.5-15.

The execution times for the DSP5685x are slower than average for a fixed-point processor, but provide a significant speed-up over those of its predecessor, the DSP568xx. This speed-up is mostly a *result of its faster instruction cycle rate (rather than because of lower cycle counts).*

The DSP56853 has the best cost-execution time result for all of the processors benchmarked here. In addition, the processor's energy consumption is extremely good. These results indicate that it is well suited for the power-sensitive applications it targets, and will provide a cost-effective solution in applications that do not require top performance.

As indicated by the processor's low program memory use on the Control benchmark, the DSP5685x should achieve very good code density. In addition, the DSP5685x achieves fairly low total normalized total program memory usage.

Cost

DSP5685x projected price and package options are shown in Table 7.9-4. The DSP56800E core is available for license; contact Motorola for information regarding licensing fees.

Fabrication Details

DSP5685x family members will be fabricated in a 0.18 µm five-metal-layer CMOS technology, according to Motorola.

Development Tools

The DSP5685x is supported by Motorola's Suite56 Software Development Tools, which include a text-based assembler, linker, and simulator (CLAS software development package) and a simulator/debugger with a window-based graphical user interface. The tools use either standard COFF object format or Motorola's own ASCII-based object format. This package is available for MS-DOS, Windows 95/98, Windows NT 4.0, SunOS, Solaris, and HP-UX.

Part	Speed (MHz)	I/O Voltage	Core Voltage	Package	Price (Qty. 10,000)
DSP56854	120	3.3	1.8	144 LQFP/ MBGA	$4.50
DSP56853	120	3.3	1.8	81 MBGA	$3.75

TABLE 7.9-4. Pricing for DSP5685x versions. Prices as of October 2000.

> *The tool set is not a part of an IDE, and other tools, such as "make," are necessary to conveniently handle software projects. However, the debugging facilities of the simulator are good, and.*

Motorola's DSP5685x instruction-set simulator accurately models all of the I/O pins of the devices, provides cycle-accurate simulation of the processor and enables modeling of external data streams and interrupts. As is typical for Motorola's simulators, this simulator also has a C-language procedural interface that allows the integration of the processor model with other simulation environments.

Besides the CLAS package, Motorola offers a C compiler for the DSP5685x.

> *Motorola distributes the latest version of its CLAS package, which contains an instruction-set simulator, assembler, linker, and librarian on the Internet free of charge. Development tool availability on the Internet simplifies software upgrades and is an advantage.*

Motorola's comprehensive "Development Environment" consists of the Software Development Kit (SDK), Metrowerks CodeWarrior for Motorola DSP Embedded Systems (an IDE), and a hardware evaluation module (EVM). The SDK provides peripheral drivers, libraries, and interfaces (for interfacing simulated peripherals), while the IDE features a simulator, debugger, C compiler, assembler, and linker (these are separate from those found in Suite56).

The DSP56854-based EVM features a DSP56854 processor, power supply, 64Kx16 external program SRAM and 64Kx16 external data SRAM, a JTAG/EOnCE port, an on-board command converter interface (emulation interface), an RS-232 serial port, an audio codec, a copy of the SDK, and a 30-day trial license for the CodeWarrior IDE. The EVM runs on PC-compatible systems running Windows 95/98/NT 4.0.

Applications Support

The base document for the DSP5685x family is the *DSP56800E Family Manual.* This volume describes the processor core and the instruction set, but not the peripherals or memory configuration. These chip-specific details are covered in the *DSP56854/53 User's Manual.* Additionally, Motorola provides a data sheet for each processor which contains electrical and timing specifications, pinout, and packaging descriptions.

Motorola provides applications support via application notes, training classes, a telephone hot-line, and a newsletter. User's manuals, technical documentation, product information, and press releases can be found at Motorola's website.

Advantages

- 32-bit barrel shifter
- Large address spaces
- Rich instruction set

- Fast interrupt capability
- Long, word, and byte data types supported
- Bit manipulation unit; good bit-field manipulation instructions
- Software stack with stack pointer supports HLL compilers
- Four 16-bit timer/counters
- Good power management features
- Ability to use 1X or slower external clock
- Flexible PLL
- Low cost (e.g., $3.75 for the 120 MHz DSP56853, quantity 10,000)
- JTAG/EOnCE enhanced debugging
- Available as a licensable core
- Compatible with predecessor (DSP568xx)
- Very good cost-execution time results on the BDTI Benchmarks
- Good energy consumption results on the BDTI Benchmarks

Disadvantages

- Limited parallel move support: two reads or one write per instruction cycle, second data operand can only be accessed with address register R3
- No bit-reversed addressing
- Limited circular addressing (two circular buffers supported; however, the size of the buffers must be the same and the two circular pointers cannot be simultaneously modified)
- No externally requested wait states

7.10 StarCore SC140 Core, Motorola MSC8101

BDTImark2000 Score: 3430 at 300 MHz

Introduction

The SC140 core is a 16-bit fixed-point VLIW-based DSP processor core, announced in April of 1999. It is the first core to emerge from the joint Motorola/Lucent Technologies design center, StarCore. The SC140 core is targeted at a wide range of telecommunication applications, including wireless telecom, IP telephony, and modems. Notably, it is the first VLIW-based DSP architecture to target applications that require low power consumption, such as cellular telephones.

The SC140 will be used by Lucent and Motorola in off-the-shelf chip products; the first chip product based on the SC140, Motorola's MSC8101, is currently sampling at 300 MHz, according to Motorola. The MSC8101 is optimized for networking infrastructure applications such as third-generation wideband wireless infrastructure systems, IP telephony systems, multi-channel modem banks, and multi-channel xDSL.

Lucent's first product based on the SC140, the StarPro 2000, was unveiled in June 2000, and is expected to begin sampling at 300 MHz in April 2001. The StarPro 2000 integrates three StarCore SC140 cores and is optimized for infrastructure applications such as wireline VoIP gateways, remote access servers, wireless mobile switching centers, and radio network controllers. The StarPro 2000 is not covered in this analysis, but will be covered in future reports from BDTI.

Much of the analysis in this chapter pertains to the SC140 core and will generally be relevant to any SC140-based chip. In a few sections, however, we discuss features that are specific to Motorola's MSC8101 chip. Discussions of chip-specific features may not be applicable to other SC140-based chips offered by Motorola or Lucent.

SC140-based development chips have been fabricated by StarCore and execute at 300 MHz at 1.5 volts. In 2000, StarCore announced a scaled-down version of the SC140, the SC110. This core is a reduced-cost, reduced-performance core relative to the SC140; for example, the SC110 provides one MAC unit instead of the SC140's four. The SC110 is not included in this analysis, but will be covered in future reports from BDTI. StarCore states that the SC110 is binary compatible with the SC140 (i.e., binary code for the SC110 can be run on the SC140), providing an easy upward migration path for customers.

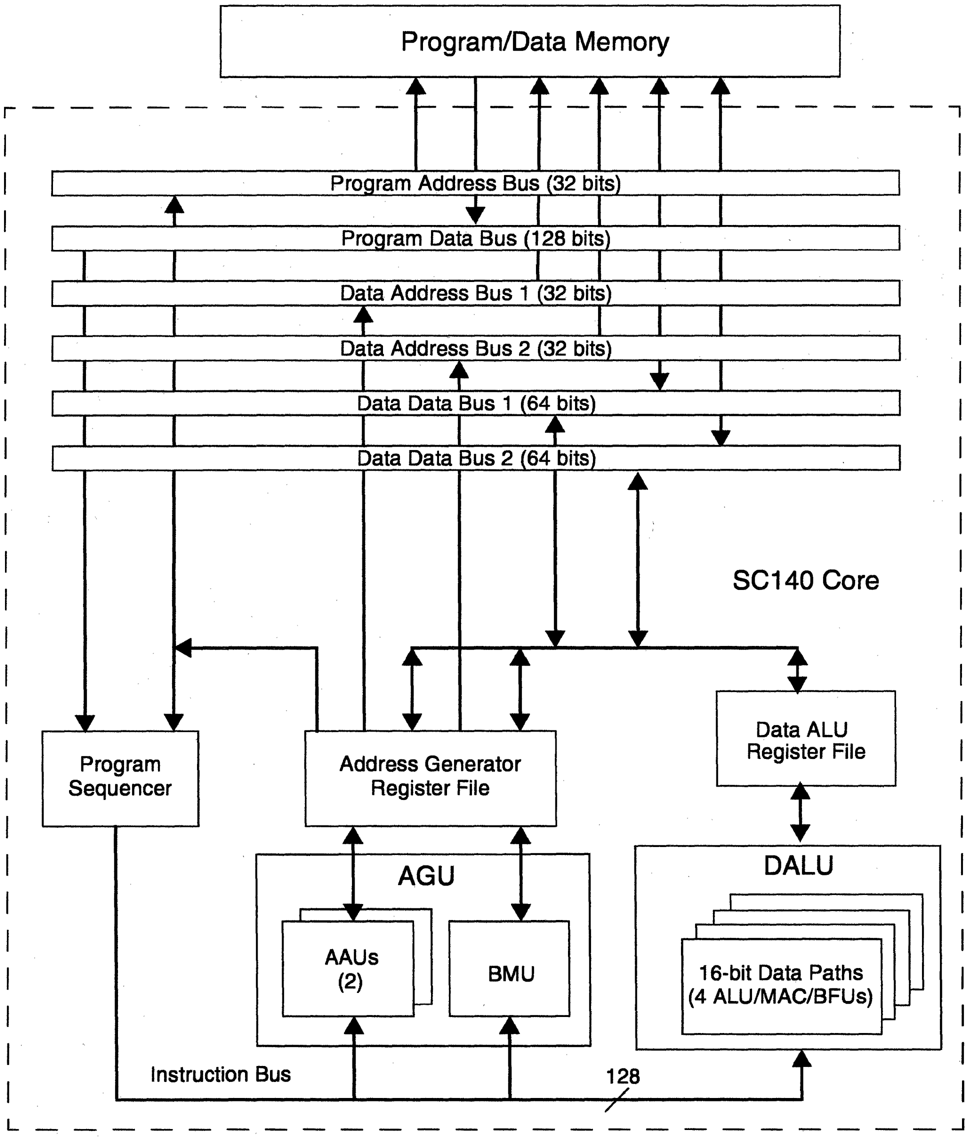
The SC140 is notable for its extremely high level of parallelism, even in comparison to other VLIW-based processors. The core provides this parallelism while achieving very low power consumption, and is the first VLIW-based DSP processor to attempt to combine low power consumption with very high performance. With its high levels of parallelism and at its 300 MHz clock rate, the SC140 is currently the fastest DSP processor to be demonstrated in silicon.

Section 7.10 - SC140

Architecture

The SC140 architecture, shown in Figure 7.10-1, consists of four 16-bit fixed-point data paths, an address generation unit that includes two address arithmetic units and one bit mask unit, and a program control unit.

The SC140 is a VLIW architecture and can execute up to six instructions at a time. Instructions that are grouped for parallel execution are referred to as an "execution set" by StarCore. Instructions are scheduled for parallel execution at compile time by code-generation tools or by the assembly-language programmer.



Program/Data Memory
Program Address Bus (32 bits)
Program Data Bus (128 bits)
Data Address Bus 1 (32 bits)
Data Address Bus 2 (32 bits)
Data Data Bus 1 (64 bits)
Data Data Bus 2 (64 bits)
SC140 Core
Program Sequencer
Address Generator Register File
Data ALU Register File
AGU
AAUs (2)
BMU
DALU
16-bit Data Paths (4 ALU/MAC/BFUs)
Instruction Bus
128

FIGURE 7.10-1. SC140 DSP core architecture.

Data Path

The SC140 contains four 16-bit data paths, each of which contains a combined ALU/MAC/bit-field unit (BFU). The BFU contains a 40-bit barrel shifter. All of the data paths are identical, and share a common set of 16 source and destination registers. The four data paths and the register set together are designated as the "Data ALU" section. The SC140 also includes an address generation unit (AGU) that contains two address arithmetic units (AAUs) and one "bit mask unit" (BMU). The address generation unit has its own set of registers; the BMU can take its operands both from the addressing registers and from the general-purpose registers, and its operation is therefore described in this section.

The MAC units, ALUs, and BFUs that comprise each of the four data paths in the DALU section are not independent (in contrast to other VLIW processors, which typically have independent MAC, ALU, and shifter units). Hence, it is not possible, for example, to issue a set of instructions that uses all four MAC units and also one of the BFUs. For this reason, in each group of six instructions executed in parallel, only four can use the DALU. The remaining two instructions in an execution set can use the address generation unit to perform data moves, pointer arithmetic, or bit mask operations; or they can specify program flow-control instructions. Only one of these two instructions can specify a bit mask operation, since there is only one bit mask unit. StarCore refers to each combined ALU/MAC/BFU as an ALU.

The SC140 provides a total of sixteen 40-bit general-purpose registers (each providing eight guard bits) for the DALU section. Each register has three portions: the high and low halves of the lower 32 bits of the register (referred to as simply the registers' high and low halves) and eight guard bits (MSBs). The high and low halves of each register can serve as inputs or outputs for the arithmetic operations.

The SC140 provides a similar number of execution units as the TMS320C62xx. However, each of the four combined ALU/MAC/BFUs on the SC140 can perform a wider range of arithmetic operations than can the execution units on the TMS320C62xx, providing more flexible parallelism. For example, the SC140 can perform four parallel multiply-accumulates in one cycle; the TMS320C62xx can perform only two. Texas Instruments' newer TMS320C6xxx architecture, the TMS320C64xx, can compute four 16-bit multiplications per cycle, and provides stronger architectural competition for the SC140.

In comparison to conventional DSP processors, such as the TMS320C54xx, the SC140 provides a much higher level of parallelism.

Having a common register set for all data paths eases programming. Since instruction latencies are low and a register can be used as both source and destination in the same instruction cycle, sixteen registers should be sufficient to support the processor's execution

> *units. The register width, 40 bits, is sufficient to provide the necessary dynamic range for applications that use 16-bit data.*

All data path operations are performed in one clock cycle; that is, the result of every arithmetic operation can be used as an input by operations in the next cycle. The SC140 is a load/store architecture, meaning that operands for DALU instructions are taken from and stored to data registers rather than directly from or to memory.

The data paths support 40-bit data for some arithmetic operations; operands smaller than 40 bits are extended to 40 bits using sign extension or zero padding, depending on the instruction being executed. The data registers can be read from or written to memory as signed or unsigned 8-bit (byte), 16-bit (word), or 32-bit (long word) operands. When a 16-bit value is read into a register as a fractional value, it is sign extended and moved into the high half of the 40-bit destination register, with the lower half of the register filled with zeros. The SC140 supports three types of two's complement data formats: signed fractional, signed integer, and unsigned integer.

The data paths include bus shifter/limiters that provide saturation and sign or zero extension. These operations are specified by the move instructions (i.e., move instructions include data format and saturation options), and affect the data before it is placed on the bus en route to memory, but do not affect the data in registers. The SC140 supports saturation of data in registers via explicit saturation instructions.

> *The SC140 data register read/write instructions are powerful. They support 8-bit, 16-bit, and 32-bit data widths, signed and unsigned modes, and integer and fractional formats.*

The SC140's data paths perform single-cycle 16 × 16-bit multiplications. The multipliers support all combinations of signed and unsigned operands, and support fractional and integer formats (both operands must be integer or fractional). Inputs can be taken from the high or low halves of any of the DALU registers, or one input can be specified as 16-bit long immediate data. In addition to multiplications, the data paths support a range of arithmetic operations, including add, subtract, negate, absolute value, and clear. Addition and subtraction operations use 40-bit operands. The data paths also support division iteration, comparison, maximum/minimum operations, transfers between registers, arithmetic shift operations, and rounding, each of which is described below.

Fractional and integer division of both unsigned and signed values is supported by the SC140 via the DIV instruction. The dividend is a 32-bit fractional or 31-bit integer value and the divisor is a 16-bit fractional or 16-bit integer value (the dividend and divisor must both be fractions or integers). The DIV instruction calculates one quotient bit based on the divisor and the previous partial remainder. To produce an N-bit quotient, the DIV instruction is executed N times, where $0<N<17$. Thus, for a full-precision (16 bit) quotient, 16 DIV iterations (and 16 instruction cycles) are required. In general, executing the DIV instruction N times produces an N-bit quotient and a 32-bit remainder that has 32 - N bits of precision and whose N most significant bits are zeros.

The SC140 provides a variety of instructions for implementing minimum and maximum operations. These instructions select the 32-bit maximum or minimum value from two data registers and store it in the second of the two registers. A SIMD version of the maximum/minimum instruction performs the same operation on the lower and upper 16-bit halves of the two registers. The instruction selects the maximum or minimum value in the high halves of the data register pair and stores it in the high half of the second register. The same operation is executed on the registers' low halves. In this case, a total of eight 16-bit maximum/minimum operations can be performed in parallel using all four ALUs. The SC140 data paths also execute a special version of the maximum/minimum operation that is intended to support efficient implementation of Viterbi decoding algorithms. The instruction MAX2VIT compares the 16-bit contents of the high and low halves of a data register pair to find the larger value. The high halves of the two registers are compared to each other, as are the low halves, and the Viterbi flags are set or cleared separately. It copies the larger value to the corresponding portion in the second data register and sets or clears the processor's Viterbi flags (VF0-VF3 in status register) to indicate which portions are larger. The MAX2VIT instruction is used with conjunction with the VSL (Viterbi shift left) instruction, which stores registers with or without shifting based on the VF0-3 bits in the status register.

The SC140's MAX2VIT and VSL instructions should prove quite effective in Viterbi decoding, an observation supported by the processor's results on BDTI's Viterbi benchmark.

Each data path supports SIMD-style addition and subtraction (using the ADD2 and SUB2 instructions) by treating values in registers as packed pairs of 16-bit data operands. For example, using SIMD operations, the SC140 can perform eight 16-bit additions per instruction cycle.

Logical operations are performed by the bit-field units. Each data path contains a 40-bit barrel shifter with a 40-bit input and a 40-bit output, a mask generation unit, and a logical unit. The barrel shifter supports arithmetic and logical shifts of up to 31 bits to the left or right. Logical operations include *and*, *or*, *exclusive-or*, and *not*. The data paths also execute bit extraction and insertion and sign and zero extension. The data paths support block floating-point operations via normalization and exponent-detect instructions.

The SC140 supports both round-to-nearest and convergent rounding. Rounding is accomplished using the ROUND instruction; the type of rounding is specified via a mode bit. The SC140 also supports subtract-and-round, add-and-round, multiply-and-round, and multiply-accumulate-and-round operations. The SC140 provides mode bits to specify that a result should be left or right shifted by one bit before storing into memory. (These mode bits control the bus shifter/limiters.) Results are automatically limited as needed following shifting.

The SC140's status bits are shared by all four of its data paths. Status bits include carry, overflow, and result-of-test (which is set based on the result of a comparison instruction, test instruction, or conditional-branch instruction). If the overflow exception bit in the status register is set, the overflow exception service routine is executed whenever there is an overflow. The overflow bit is referred to as a "sticky bit" by StarCore; once set, it remains set until explicitly cleared by the programmer. If more than one instruction in an execution set attempts to update the carry bit, the bit is updated by the last instruction in the set (as it appears in the program). Only one instruction that affects the result-of-test bit can be a part of an execution set.

> *The lack of separate status bits for each data path and for SIMD operations may cost cycles in some applications. In some cases (particularly in control-oriented software), it may result in the programmer or compiler only being able to use one of the data paths. StarCore states that this lack was a design decision meant to help maintain architectural scalability.*

The bit mask unit (BMU) performs bit test, set, invert, and clear operations on 16-bit data. Source operands can come from the DALU registers, address registers, control registers, or directly from memory.

Memory System

The SC140 has two 32-bit address buses and two 64-bit data buses for transferring data. Instructions are fetched via a 32-bit address bus and 128-bit data bus. Program and data memory are unified; any address can contain either instructions or data. Memory is byte-addressable and can be accessed either as little-endian or big-endian data, controlled by a mode bit.

The 128-bit program data bus allows retrieval of up to eight 16-bit instruction words per cycle. (Although the processor can only execute six instructions per cycle, the processor fetches eight words because some instructions require two or three words, and one or two of the 16-bit words may be used as prefixes, described in the *Instruction Set* section.)

The SC140 can perform two data reads, two data writes, or one read and one write per instruction cycle. Each read or write can access contiguous groups of data up to 64 bits wide. This allows retrieval of up to sixteen 8-bit, eight 16-bit, or four 32-bit words per cycle (assuming that groups of words retrieved by each data bus are contiguous in memory). Word accesses of 16 bits require addresses to be aligned on multiples of two; 32-bit word accesses must be aligned on multiples of four; and 64-bit accesses (four 16-bit or two 32-bit words) must be aligned on multiples of eight. On a 300 MHz SC140, the maximum on-core data memory bandwidth is 2,400 million 16-bit words/second.

> *The SC140 has much higher on-chip data memory bandwidth than most other DSP processors. Its memory bandwidth should be sufficient to keep the execution units supplied with data and avoid data memory bottlenecks when the processor uses data in on-chip memory. The SC140 can only achieve the maximum data bandwidth when accessing groups of contiguous words in memory, however.*

> *The 32-bit address space should be sufficient for virtually all applications.*
>
> *The unified memory architecture provides greater flexibility for users, compared to architectures that segregate program and data memory.*

MSC8101 On-Chip Memory

The MSC8101 contains 512 Kbytes of on-chip SRAM (256 K 16-bit words). As described above, the memory is unified; it is used to store both instructions and data.

External Memory Interface

The SC140 is a core; the external memory interface will be chip-specific.

MSC8101 External Memory Interface

The MSC8101 external memory interface provides a glueless interface to the PowerPC bus. It includes a 32-bit address bus and 32-bit or 64-bit data bus (the width is selectable via software). The interface supports multiple masters, and also supports programmable wait states. The bus operates at 100 MHz, and the maximum external memory bandwidth is 400 million 16-bit words/second.

> *The MSC8101's off-chip memory bandwidth is significantly lower than its on-chip bandwidth. Realizing the chip's performance potential will require careful use of on-chip memory, as is the case with the TMS320C62xx.*

Address Generation

The SC140 provides one address generation unit (AGU) that contains two address arithmetic units (AAU), a bit mask unit (BMU), and a set of addressing registers. The AGU is capable of generating two addresses per instruction cycle. The AGU provides 16 primary registers (R0-R15). The last eight of the primary registers (R8-R15) are also used as base registers for modulo addressing (where they are referred to as B0-B7). The AGU also provides two stack pointers (NSP, ESP), only one of which is active at a time. The active stack pointer is accessed as SP. The SC140 provides four 32-bit post-modifier/offset registers (N0-N3) each of which can be used with any of the primary address registers. These four registers can also be used as general-purpose registers.

The two AAUs are identical; each contains a 32-bit adder which can perform arithmetic operations on the 16 AGU registers (R0-R15), the program counter (PC), the stack pointer (SP), and the four offset registers (N0-N3). The AAUs can add or subtract the contents of two AGU registers, add an immediate value to the contents of a register, increment or decrement registers, and add with reverse carry to the contents of an AGU register. The AAUs also contain a second adder which is used in modulo addressing in conjunction with the modulo comparator. The operations supported by the BMU are described in the *Data Path* section, above.

Like many DSP processors, the SC140's maximum memory bandwidth can only be achieved when data is arranged in groups of contiguous words in memory. This is in contrast to, e.g., Infineon's VLIW DSP processor core, Carmel, which generates four addresses per clock cycle, and can achieve its maximum bandwidth even when accessing non-contiguous data words.

The SC140 supports register-direct, register-indirect, indexed, PC-relative, bit-reversed, and modulo addressing modes. Immediate data is also supported. Bit-reversed addressing is supported via reverse-carry address modification. When an address is post-modified in this mode, the carry bit is propagated in the opposite direction to generate bit-reversed addresses.

Address modifiers include post-increment by one, post-decrement by one, and post-increment by a signed offset register value. The absolute value of the 32-bit offset register must be less than or equal to the modulo value. Indexed addressing can be accomplished using short-immediate displacement (0 to 7 bytes, 16- or 32-bit words), or word displacement. In word displacement, the displacement is specified as a signed 15-bit word, requiring a second instruction word. It is sign-extended to 32 bits and then added to Rn to obtain the operand address. Thus, the displacement in this mode can range from -16,384 to +16,383 for 8-bit moves, -8,192 to +8,191 for 16-bit moves, -4,096 to +4,095 for 32-bit moves, or -2,048 to +2,047 for 64-bit moves. The stack pointers also support indexed addressing with short displacement and word displacement.

Indexed addressing is useful for compiler-generated code.

The SC140 can implement four modulo buffers at a time. The MCTL register contains four bits which, when set by the programmer, activate modulo addressing. The base registers (B0-B7) are used to hold the lower boundary value of the modulo buffer; the upper boundary is calculated as Bn+My-1, where My is one of the modulo registers (M0-M3). Which My register is associated with which Bn register is defined in the MCTL register, which can be modified by the programmer during program execution. Modulo buffer sizes can range from 1 to 2^{32}-1.

The SC140 also supports a special modulo addressing mode, "multiple-wrap-around" modulo addressing. In this mode, the base registers are not used as lower boundary values, and hence can be used for other purposes. The Mn register, which can have a value between 2 and 2^{31} and must be a power of 2 in this mode, is used to derive both the lower and upper boundaries of a modulo buffer. The lower boundary value has zeros in the k LSBs, where Mn = 2^k, and therefore must be a multiple of 2^k. The upper boundary is the lower boundary plus the modulo size minus one. Multiple-wrap-around modulo addressing is functionally identical to regular modulo addressing when simple post-increment and post-decrement operations are used to update addresses, since at each increment, the address will only wrap a maximum of one time. The main difference arises when an offset is used to update the address, and the processor is making long word accesses with very small modulo values. In this case, there is potential for multiple wrapping if the offset value is not less than the buffer size. In multiple-wrap-around mode, multiple wrapping is supported, and the offset value need not be less than the buffer size.

The SC140 addressing modes are typical for modern DSP processors.

Pipeline

The SC140 processor uses a five-stage pipeline consisting of pre-fetch, fetch, dispatch, address generation, and execute stages. The first three stages are implemented in the program sequencer unit and the last two stages are implemented in the address generation unit (AGU) and DALU.

In the pre-fetch stage, the processor generates the address for the program memory fetch and increments the fetch counter (FC). In the fetch stage, the processor fetches eight 16-bit instruction words from program memory. The SC140 has an internal buffer that is used to hold at least one execution set. Each execution set can contain fewer than eight instruction words. If the internal buffer contains more than eight instruction words, the processor does not fetch additional instructions during that instruction cycle.

In the dispatch stage, the program sequencer uses information in the prefix words or the serial grouping encoding (described in the *Instruction Set* section, below) to determine whether each instruction will execute in the DALU or AGU, and which instructions will be executed in parallel. At this point, the instructions are issued to the execution units. The number of cycles required by the entire execution set is determined by the instruction within the set that consumes the most cycles. The dispatch stage also includes decoding for AGU instructions; DALU instruction decoding is performed in the address generation stage, as is address generation for load/store operations.

During the execute stage the processor reads data operands, performs arithmetic operations, and writes the results to destination registers. During the execute stage the processor also performs load/store accesses to memory.

The SC140 pipeline is not fully hidden from the user, as illustrated by the following examples:

- A pipeline hazard occurs if an address register is loaded from memory and used as a pointer in the next cycle. This restriction arises because data is loaded from memory in the execute stage, but registers are used as pointers in the address generation stage (which leads the execute stage).
- Two cycles must elapse between modification of MCTL and use of an address register as a pointer or arithmetic operand.
- Instructions executed in the AGU that use the T register (result-of-test register) cannot immediately follow instructions that update the T register.

- A stall will occur when a change of flow instruction branches to an execution set that is spread over two fetch sets; i.e., an execution set that is not aligned on a 16-byte boundary (a fetch set is a group of instruction words fetched from memory in a single cycle).

The S140 pipeline is not interlocked; however, the assembler detects pipeline hazards and issues warnings.

The first two stages of the five-stage pipeline are dedicated to program memory accesses. This results in a penalty of two cycles for unconditional change-of flow instructions which use immediate destination values. Unconditional PC-relative change-of-flow instructions carry a penalty of three cycles. Conditional change-of-flow instructions (including PC-relative) always have a penalty of three cycles if the change-of-flow occurs; there is no penalty if the change-of-flow is not executed.

The SC140 instruction set also includes delayed versions of the change-of-flow instructions. The number of cycles saved by using the delay slots varies depending on the instructions used in the delay slot.

> *In comparison to other VLIW-based DSP processors, such as Carmel and the TMS320C62xx, the SC140's pipeline is quite short. The SC140's five-stage pipeline is benign, and does not seriously complicate programming. Pipeline hazards can be avoided with very little programming effort.*

Instruction Set

The SC140 registers and instruction set are summarized in Table 7.10-1 and Table 7.10-2, respectively. The SC140 is a VLIW-based architecture; the processor fetches eight instruction words and can execute up to six instructions in parallel (the remaining two words can be used for prefixes or immediate values). Each instruction in the execution set uses one execution unit. The available execution units on the SC140 are:

- Four data paths. These four are referred to collectively as the Data ALU section (DALU).
- Two address arithmetic units
- One BMU
- One program controller

The processor uses two different methods for specifying which instructions will be included in an execution set: serial grouping and prefix grouping (see Figure 7.10-2).

- Serial grouping uses the two most significant bits in the instruction to determine the end of an execution set. If these two bits are not 00, the instruction is considered to be the last instruction in the execution set.
- Prefix grouping uses a one-word or two-word prefix for an execution set. The prefix defines how many instructions are included in the execution set, and also con-

tains information used for conditional execution and looping. Prefix grouping must be used if instructions are to be executed conditionally.

The choice of prefix or serial grouping is not specified by the assembly programmer; the assembler determines the grouping method. Serial grouping is used whenever possible to minimize instruction word usage.

There are a number of restrictions governing which instructions can be executed in parallel. As explained earlier, only four instructions can use the DALU section at a time; hence, although it may appear in some descriptions of the SC140 that there are 12 execution units in the DALU section (an ALU, MAC unit, and bit field unit in each of the four

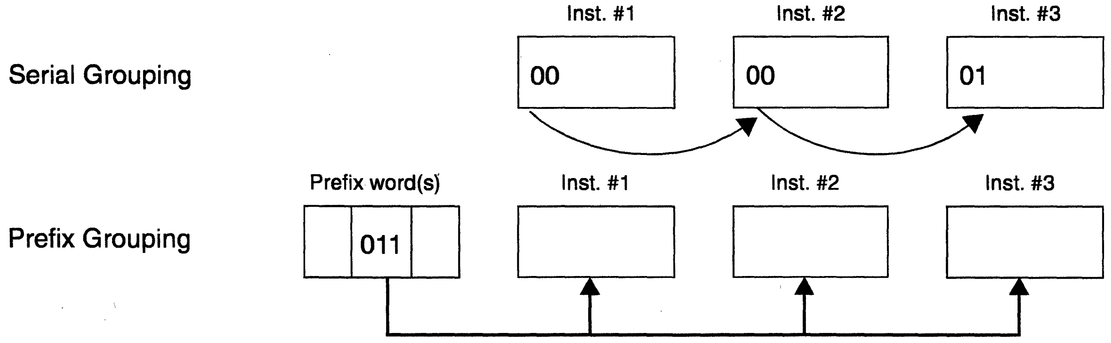


FIGURE 7.10-2. SC140 DSP core instruction grouping methods.

Registers	Width	Purpose
D0-D15	40	General-purpose registers
R0-R7	32	Primary address registers
R8(B0)-R15(B7)	32	Address registers (circular buffer base registers)
N0-N3	32	Address offset registers
M0-M3	32	Address modifier registers (for modulo, multiple wrap-around modulo, and reverse-carry modes)
MCTL	32	Address modifier control register (selects linear, reverse-carry, modulo, multiple wrap-around modulo modes)
NSP, ESP	32	Normal and exception stack pointers
SA0-SA3	32	Loop start address registers
LC0-LC3	32	Loop counter registers
T	1	Result-of-test bit (1 if true, 0 if false)

TABLE 7.10-1. SC140 register summary.

data paths), only four of these can be used in parallel, because the DALU section only has four instruction decoders. Similarly, only one bit mask unit instruction can be executed at

Class	Instructions
Arithmetic	Absolute value, add or subtract, add or subtract with carry, SIMD add or subtract two 16-bit values, add or subtract without changing carry bit, add or subtract and round, clear, divide iterate, decrement and set based on result of test, increment integer or fractional data register, negate, round, saturate fractional or long, shift left and subtract, transfer
Multiplication	Integer or fractional multiplication or multiply-accumulate (all possible combinations of signed and unsigned operands); fractional multiply or multiply-accumulate with rounding
Logic	*And, or, not, exclusive-or*
Shifting	Arithmetic left or right shift by one bit, arithmetic left or right shift by 0-31 bits, logical left or right shift by 0-31 bits
Rotation	Rotate one bit left or right through carry bit; sign extend byte, 16-bit word, 32-bit word; zero extend byte, 16-bit word, 32-bit word
Conditional Execution	Transfer data register based on result-of-test bit in status register; almost all instructions can be executed conditionally using prefixes
Comparison	Compare for equality, greater than, or higher (unsigned) and update T register; choose maximum or maximum of two 16-bit values, absolute maximum or minimum; test for equal to zero, greater than or equal to zero
Looping	Set up loop, set up counter, skip, continue, break
Branching	Unconditional or conditional branch, jump with delay slot, jump without delay slot
Subroutine Call	Unconditional or conditional subroutine call, return with delay slot, return without delay slot
Bit Manipulation	Count leading bits, extract signed or unsigned bit field, insert, bit mask change, clear, set, test if set, test if clear, test and set
Move	Move signed or unsigned byte, word, or long (32-bit) word; move signed two words, two long words, four words; move (with optional saturation) fractional, two-word fractional, four-word fractional
Special Function	Viterbi decoding-related moves and comparisons

TABLE 7.10-2. SC140 instruction set summary.

a time, since there is only one BMU. In addition, a register cannot be the destination of more than one instruction per execution set. Two instructions that update the program counter cannot be executed in parallel.

The SC140 has a program bus width of 128 bits, which constrains the number of words that can be grouped in an execution set. The total instruction word count in an execution set cannot exceed eight words. Some SC140 instructions require a second and even a third instruction word (for example, for holding an immediate value). The second and third words of an instruction are referred to as "extension words" by StarCore. An execution set cannot have more than two extension words.

In contrast to extension words, which affect only a single instruction, prefixes are used to convey information about an entire execution set, including:

- The number of instructions in the execution set.
- Conditional execution information for the whole set or a subset based on the result-of-test bit, T, in the status register. (Conditional execution can only be performed using prefix grouping.)
- Information used for hardware looping.

If the execution set contains instructions which use registers in the higher-numbered halves of the register banks (D8-D15, R8-R15), another prefix word is added to the execution set. A single execution set can contain both an extension word and a prefix.

When the prefix words are not needed, instructions are grouped in execution sets using the serial grouping method. The use of serial grouping imposes further restrictions on which instructions can be executed in parallel. For serial grouping, StarCore groups instructions in four broad categories:

- Type 1. Basic DALU and move instructions. These are single-word instructions that are used very frequently.
- Type 2. Additional DALU, move, and AGU arithmetic instructions. These are also single-word instructions, but they are not used as frequently as Type 1 instructions.
- Type 3. Two-word or three-word DALU, move, and AGU arithmetic instructions.
- Type 4. All other instructions. These instructions may be one or two words long.

The serial grouping options for an execution set are:

- One to six Type 1 instructions
- A single Type 2 instruction grouped with up to five Type 1 instructions
- A single Type 3 instruction grouped with up to five Type 1 instructions
- A single Type 4 instruction

These groupings are subject to the more general restrictions described earlier; e.g., no more than four DALU instructions can be grouped in an execution set.

Assembly Language Format

The SC140 assembly language uses the traditional opcode-operand format. For example, the following instructions are excerpted from an FIR filter implementation:

```
loopstart1
   [
   mac d0,d8,d12 mac d1,d9,d13 mac d2,d10,d14 mac d3,d11,d15
   move.4f (r0)+,d4:d5:d6:d7 move.f (r3)+n0,d8
   ]
   [
   tfr d8,d0 tfr d4,d1 tfr d5,d2 tfr d6,d3
   moves.4f d0:d1:d2:d3,(r1)+ move.4f (r2)+,d8:d9:d10:d11
   ]
loopend1
```

The instructions grouped inside the brackets form an execution set. The "loopstart1" label defines the starting address for hardware looping; "loopend1" defines the ending address. The starting and ending addresses are embedded into the prefix word of the second execution set.

The first execution set performs four multiply-accumulates. D0 is multiplied by D8 and the result is added to D12. D1 is multiplied by D9 and the result is added to D13, and so on. In parallel with those operations, the contents of the data memory location pointed to by the R0 register (four 16-bit words, each represented in 32-bit fractional format; i.e., left shifted 16 bits and sign extended to 40 bits) are moved into the D4, D5, D6, and D7 registers. After issuing the memory move, the R0 register is incremented by eight. The last instruction in the first execution set moves the contents of the data memory location pointed to by R3 to register D8. After issuing the memory move, the N0 register is added to the contents of the R3 register, and the result is stored back into the R3 register.

In the second execution set the contents of the registers D8, D4, D5, and D6 are copied to D0, D1, D2, and D3 respectively. Before the copy operation takes place, the contents of the registers D0, D1, D2, and D3 are moved to the data memory location pointed to by the R1 register. The contents of the registers are assumed to be in fractional format, and if necessary, they are saturated before the move operation. The address register R1 is incremented after the move operation to point to the next four 16-bit words in data memory. The last instruction in the second execution set moves the contents of the data memory location pointed to by R2 to registers D8, D9, D10, and D11. After the move operation the address register R2 is incremented to point the next four 16-bit words in the data memory.

The SC140 allows conditional execution of all or part of an execution set. Conditional execution options are specified in the prefix. If there is no prefix, the whole set is executed. An execution set can have two instruction subsets that can each be executed conditionally. Each subset can include up to two DALU instructions and one AGU instruction. To distinguish between the two subsets, the first subset is placed by the assembler at an even address; the second subset is placed at an odd address. Thus, we refer to these sets as "even subset" and "odd subset." Table 7.10-3 lists the options for conditional execution of subsets.

Parallel Move Support

The SC140 supports operand-unrelated parallel moves by allowing up to two data move instructions to be executed in parallel with other instructions. These data moves can be either one load and one store, two loads, or two stores. The access width of each parallel move can be a byte, a word (16 bits), a long (32-bit) word, two words, or two long words, or four words. If the access width is two or four words, the words must be located in consecutive memory locations.

Orthogonality

The SC140 instruction set is quite orthogonal, because most of the instructions are simple and specify a single operation. Unlike some VLIW processors, the SC140's instruction set is composed of relatively short (16-bit) instructions whose functionality can be extended using prefixes or extensions. Short instructions often require processor architects to place restrictions on, e.g., register usage; the SC140 avoids the need for register restrictions by using prefix words where needed. With the exception of special instructions for Viterbi decoding, all SC140 instructions can use all registers without restriction. There are a number of restrictions on grouping instructions in execution sets, however, which complicate assembly language programming.

The high level of orthogonality and relatively simple pipeline make the SC140 simpler to program in assembly language than many DSP processors, and the processor lends itself well to compilers. Programming is somewhat complicated, however, by the myriad rules governing instruction grouping.

Options for Conditional Execution
Unconditional execution of the entire execution set
Execution of even subset if T=1; execution of odd subset if T=0
Execution of the whole set if T=1
Execution of the whole set if T=0
Execution of even subset if T=1; always execute odd subset
Execution of even subset if T=0; always execute odd subset

TABLE 7.10-3. Conditional execution of subsets of execution sets ("T" is the result-of-test bit in the status register).

Execution Times

One to six instructions can be grouped into an execution set and execute in parallel. Instructions within the same execution set always start execution at the same time. A new execution set begins execution only after all of the instructions belonging to previous execution sets are completed. Therefore, the time required to complete an execution set is determined by the instruction in the set that requires the most time.

All DALU instructions execute in a single cycle. All move instructions to internal memory take one clock cycle to execute, unless the addressing mode performs a pre-calculation (such as a stack pointer indirect addressing with an offset). In this case, the move requires two cycles. All bit mask unit instructions execute in two cycles, unless pre-calculation of a target memory address is needed (such as adding an offset to the SP register); in this case, a third cycle is added.

The cycle count penalty required by change-of-flow instructions varies from one (for a delayed jump with instructions in all delay slots) to six (for a return from subroutine instruction, when the return address register and shadow stack pointer are not "valid," as described in "Stack" within the *Execution Control* section.)

> *Although the pipeline of the SC140 is fairly simple, there are a number of multi-cycle instructions—including all bit mask unit instructions. These instructions slow the execution of the entire execution set in which they are grouped. It should be noted, however, that these instructions were rarely used in the BDTI Benchmarks and had a minimal effect on the processor's benchmark results. Hence, it is likely that they will not significantly detract from the processor's performance in most applications.*

Execution Control

Motorola MSC8101 Clocking

The MSC8101's SC140 core derives its master clock from an external clock through the use of a PLL and clock generation circuitry. (Separate clocks are used to control the CPM and PowerPC bus described in the *Peripherals* section.)

The PLL can divide the input frequency by any integer between 1 and 16 and can multiply the input frequency by factors between 10.0 and 31.988.

The PLL multiplier and divider can be used simultaneously, for a multiplication range of 0.625 to 31.988. Division and multiplication ratios are determined at reset and may be modified during operation.

The clock generator has a divider connected to the output of the PLL unit. The output frequency of the PLL can be further divided by a factor of 2^n (where $-1<n<8$, and n is an integer). The division factor can be modified during operation. Hence, using the PLL and clock generator multiplier and dividers, the input frequency can by multiplied by factors ranging from 0.00488 to 31.988 to derive the core frequency.

It is possible to disable the PLL and use only the clock generation circuitry. This is particularly useful in low power consumption modes. When the chip is required to exit a low power mode, it can immediately do so with no time needed for clock recovery or PLL lock if the PLL is disabled.

The SC140 core is static; the processor state is maintained when the clock is stopped.

Hardware Looping

The SC140 core supports up to four levels of nested hardware loops. Loops are implemented using a prefix word which contains information about the beginning and ending addresses of the loop (specified in an assembly program using the assembly directives “loopstart” and “loopend”). SC140 hardware loop support includes four pairs of registers, each of which holds the start address of a loop and the number of times the loop is to be repeated. A loop can only be nested inside a loop that has a lower number (the zeroth loop uses registers SA0 and LC0; the fourth loop uses registers SA3 and LC3, etc.).

Loop counter registers are 32 bits wide and hold signed values. Loops can be repeated up to 2^{31}-1 times. The SKIPLS instruction causes the loop to be skipped when the loop counter is zero or negative. The SKIPLS instruction is executed before the first execution set of the loop. The SKIPLS instruction eliminates the need to explicitly check the loop counter for zero or a negative number in cases where the loop count value is computed.

The SC140 supports “short loops” and “long loops.” Short loops consists of one or two execution sets. In short loops, the execution sets are stored in internal buffers and are executed the number of times specified by the loop counter. Short loops do not use the loop starting address register; instead, the loop starting and ending addresses are embedded in the prefix words. For long loops, the starting address must be explicitly written to the associated SCy register by executing a DOSETUPy instruction. Both short and long loops are interruptible.

For long loops that are unaligned, meaning that the first execution set is not aligned on a 16-byte boundary, a one-cycle penalty is incurred at each branch to the top of the loop. This penalty can be avoided by aligning the loop through the addition of NOP instruction, or by using assembler directives to ensure alignment.

The CONT instruction causes the current loop iteration to terminate before reaching the last execution set of the loop. If the value of the loop counter is greater than one, then the CONT instruction causes the program to branch to the address stored in the starting address register, and the loop counter is decremented by one. If the value of the loop counter is less than or equal to one, then the CONT instruction causes the program to branch to the address specified by the CONT instruction. The loop counter is cleared and the loop terminates. The CONT instruction can only be used in long loops.

The BREAK instruction causes the loop to terminate, regardless of the value of the loop counter. The program address bus is loaded with the address specified by the BREAK instruction; the value of the loop counter is not changed.

Interrupts

The SC140 core does not have pre-defined external interrupt source definitions; external interrupt sources are chip specific and require an interrupt controller circuit to be added to the SC140 core.

The SC140 supports three hardware interrupts: one for illegal instruction and execution set exceptions, one for overflow exceptions (which can be activated or deactivated by the programmer), and one for debug exceptions which is initiated by the on-chip emulation unit. The SC140 has one software interrupt which is activated by the TRAP instruction.

The SC140 allows the user to determine the base address of the Interrupt Vector Table by writing it to the Vector Base Address (VBA) register. The reset value of VBA is zero. There are 64 possible interrupt vector locations. The distance between adjacent interrupt vectors is 32 words.

The programmer may enable or disable each interrupt by changing the bits in the Interrupt Priority Register (IPR) that are associated with that interrupt. Clearing all of the appropriate bits will disable the interrupt. All maskable interrupts can be simultaneously enabled/disabled by clearing/setting the master Disable Interrupt (DI) bit in the Status Register.

The SC140 provides seven Interrupt Priority Levels (IPLs). The core itself has a user programmable priority (set in bits 23-21 of SR); only interrupts with a higher priority than the core are serviced. Non-maskable interrupts and internal exceptions (on-chip hardware interrupts and TRAP instruction) are serviced without regard to the current IPL.

Motorola MSC8101 Interrupts

The MSC8101 includes an interrupt controller to handle all on-chip peripheral and I/O interrupt requests. It supports 8 non-maskable interrupts and 24 maskable interrupts.

Stack

The SC140 has two stack pointers, one that is used by applications, and one that is used by interrupts and operating systems (called the "exception stack"). The core maintains shadow registers containing the values of each of the stack pointers decremented by one, to save pre-calculation time when the stack pointer is used for a "pop." If the stack pointer register is explicitly changed, however, the shadow stack pointer becomes invalid, and the next "pop" requires an extra cycle for pre-calculation.

The exception stack is used when there is a software or hardware interrupt. Since operating system calls are performed by executing the TRAP instruction (which generates a software interrupt), the exception stack is used for servicing operating system calls.

Both stack pointers support PUSH and POP instructions. In addition, the stacks can be accessed by move or bit-mask instructions that use the stack pointer as a base pointer for short- and word-displacement addressing (see *Address Generation* section for discussion of short- and word-displacement addressing).

Bootstrap Loading

The SC140 includes boot-address lines that are intended to be hard-wired (in an SC140-based chip) to specify any address in its internal or external memory space. Upon reset, the device will begin execution from this address.

Motorola MSC8101 Bootstrap Loading

The MSC8101 provides boot address pins that can be hard-wired or connected to an external device. The MSC8101 can download boot code via its host port or PowerPC bus interface.

Peripherals

The SC140 core does not include any peripherals; Lucent and Motorola will include various peripherals in their SC140-based products.

Motorola MSC8101 Peripherals

- **Communications Processor Module (CPM)**

  The CPM is a programmable protocol machine that uses a 32-bit RISC engine. The CPM supports a variety of interface protocols, including 155 Mbps ATM interface (including AAL 0/1/2/5), 10/100 Mbps Ethernet interface, up to four E1/T1 interfaces (or one E3/T3 interface and one E1/T1 interface), and up to 256 channels of HDLC. The CPM operates at up to 150 MHz and is driven by its own clock. This module is an enhanced version of the CPM used in Motorola's MPC82xx PowerPC family.
- **PowerPC bus interface**

  The PowerPC bus interface provides a glueless interface to the PowerPC bus. It operates at up to 100 MHz with a 64- or 32-bit data width. It provides support for multiple bus masters.
- **Programmable memory controller**

  The programmable memory controller controls up to eight banks of external memory. The memory controller supports programmable wait states and extends this concept via what Motorola calls "user programmable machines" (UPMs). The chip's three UPMs allow the user to define, in software, timings for the MSC8101 external memory interface and for on-chip memory and peripheral buses. This enables glueless interface to various memory types (SRAM, DRAM, EPROM, flash) and to other user-definable peripherals. The memory controller also includes a dedicated pipelined SDRAM memory interface.

- **DMA controller**

  The DMA controller provides 16 DMA channels. The controller uses its own bus and thus does not require cycle stealing. The DMA controller and the core can access memory without contention, as long as they do not attempt to access the same 32 Kbyte block of memory in the same cycle. The DMA is FIFO based; data in external memory is loaded into a FIFO buffer, and then transferred by the DMA controller to local memory in burst mode. This is intended to reduce contention on the DMA bus, which is used by all 16 channels.

- **Enhanced filter co-processor (EFCOP)**

  The enhanced filter co-processor independently and concurrently executes long filters (such as those used for echo cancellation) and runs at 300 MHz. This co-processor is similar to the filter co-processors found on the DSP563xx family members, but has been enhanced to support 32-bit data.

- **16-bit parallel host port**

  This port provides a parallel host interface to other DSP processors or microcontrollers, or can be used for DMA transfers. It can be directly connected to the data bus of a host processor.

On-Chip Debugging Support

The SC140 provides board and chip-level debugging capability through two on-chip modules:

- Enhanced on-chip emulation (EOnCE) module
- JTAG access module

These modules can be accessed via the five-pin JTAG port. In addition, the EOnCE module can support six additional "event" pins, which can be used as inputs to, or outputs from, the EOnCE module. The EOnCE module allows access to the SC140 core and its peripherals, enabling users to examine registers, memory and on-chip peripherals. EOnCE supports:

- Address breakpoints
- Breakpoints on data bus values
- Detection of events, including:
  - Program and data memory address bus range or value
  - Data memory or data bus range or value
  - Data written or read to/from a certain data memory address

- Access to the core and its peripherals
- Profiling (including cycle counts and number of calls executed)
- Program trace buffer, which can record the addresses of change-of-flow instructions, interrupts, hardware loops, and execution sets.

Motorola MSC8101 On-Chip Debugging

The MSC8101's on-chip debugging capabilities are as described above, with the following specific attributes:

- The program trace buffer on the MSC8101 is 8 Kbytes deep, supporting trace of up to 2K addresses
- The MSC8101 includes five EOnCE pins

Power Consumption and Management

The SC140 features two different idle modes. "Wait" mode is triggered by the WAIT instruction. It takes 10 cycles to enter "wait" mode after the execution of the WAIT instruction. In "wait" mode, the global clock to entire core is halted, but peripherals continue to operate using the clock generated from the on-chip PLL. The second idle mode is the "stop" mode which is triggered by the STOP instruction. It takes 10 cycles to enter "stop" mode after the execution of the STOP instruction. In "stop" mode, both the core and the peripherals, including the PLL and clock generator, are halted.

Upon occurrence of one of the following actions the processor exits from "wait" mode and resumes execution:

- An enabled interrupt is received.
- A non-maskable interrupt (NMI) request is received.
- RESET is asserted.
- The JTAG module issues a debug request.
- The EE0 pin from the EOnCE module (if included in the chip and programmed as a debug request input) is asserted.

When one of the following actions occurs, the processor exists from the "stop" mode, turns on the clock generator, and after a clock stabilization delay resumes execution:

- A low level is applied to an external dedicated pin.
- RESET is asserted.
- The JTAG module issues a debug request.
- The EE0 pin from the EOnCE module (if included in the chip and programmed as a debug request input) is asserted.

Motorola MSC8101 Power Consumption

Typical power consumption of the core and on-chip memory of the MSC8101 running at 300 MHz is 250 mW in a 0.13 μm process at 1.5 volts, according to Motorola. Power consumption for the core in wait mode is 15 mW. Power consumption for the core in stop mode is 1.50 μW. However, the stop mode current is highly dependent on the leakage current of the transistors thus can vary depending on the process technology used to manufacture the processor. BDTI has not verified these power consumption figures.

Benchmark Performance

The SC140 has been benchmarked with the BDTI Benchmarks™. Overall benchmark results for all benchmarked processors are presented in Chapter 8, *BDTI Benchmark™ Results*. We summarize and analyze the SC140 benchmark performance in the paragraphs below. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not reflect the processor's instruction cycle rate; therefore, lower instruction cycle counts imply a more powerful architecture, but do not imply faster speed. Next we discuss benchmark execution times and cost-execution time products, indicating processor speed and cost-performance, respectively. We then discuss the processor's energy consumption, which reflects the energy consumed by the processor in order to perform a task. Finally, we discuss the processor's memory usage. We divide the memory usage discussion into three parts: Control benchmark memory usage, overall benchmark program memory usage, and benchmark data memory usage.

For cycle counts and memory usage, we report results for the SC140 core. These results apply to the SC140 core under the assumption that the benchmarks execute from full-speed on-chip statically loaded memory (as opposed to, for example, dynamically loaded cache memory), as is the case for the MSC8101. These results my not apply to SC140-based chips with different memory configurations. For execution times, cost-execution times, and energy consumption, we report results that combine the SC140 cycle counts with the MSC8101's speed, price, and power consumption, respectively. Results for other SC140-based chips, such as the StarPro 2000, will likely differ from the results presented here for the MSC8101.

Execution Performance

- **Instruction cycle counts:** The SC140 has the lowest cycle counts of all of the processors benchmarked here on all but the FFT, Viterbi, and Bit Unpack benchmarks. Cycle counts are illustrated graphically in Figure 8.1-13. The SC140 has the lowest total normalized cycle count—approximately 65% below the average of the processors benchmarked in this report. The total normalized cycle count of the

SC140 is more than 25% lower than that of the TMS320C64xx-C, which has the second-lowest result.

The SC140 is able to obtain these low cycle counts by virtue of parallel use of multiple execution units and wide buses (two 64-bit data buses); for example, these features enable the processor to perform four multiply-accumulates on 16-bit operands together with two data load/store operations in a single cycle. The processor with results closest to those of the SC140, the TMS320C64xx-C, has a similar processing bandwidth. The remainder of this analysis will focus on analyzing their respective strengths and weaknesses.

The SC140's ability to execute four 16-bit multiply-accumulates per cycle is supported by wide data buses and a powerful address generation mechanism; this allows the execution units to be utilized fully in the inner loops of the MAC-intensive benchmarks. However, on the SC140 fetching four operands at once makes it impossible to use a conventional circular buffer as a delay line in some benchmarks. When the processor attempts to read multiple data words that are split between the end and start of the buffer, a wrap-around read is necessary. The SC140, unlike the TMS320C64xx-C, is not capable of wrap-around multiple word data reads from a circular buffer, which is problematic in the single-sample benchmarks which must step through the delay line a word at a time. In such cases, the use of a circular buffer is efficient only when a delay line with a size of four times the number of taps is used. This results in a significant amount of wasted memory, so instead, the delay lines are updated without using the circular buffer addressing support of the processor in these benchmarks. Hence, the SC140 incurs additional overhead on the Single-Sample FIR and LMS benchmarks for updating the delay line using load-store operations instead of using circular buffers. The Two-Biquad IIR filter benchmark is not similarly affected; this benchmark is completely unrolled and uses short, two-word delay lines, so it would not benefit from circular buffering of the delay lines.

Both the SC140 and the TMS320C64xx-C have the ability to perform four multiply operations per cycle (the SC140 using its four MAC units and the TMS320C64xx-C using the two SIMD multiply units). The TMS320C64xx-C lacks a full-fledged MAC unit, but includes support for dot-product (DOTP2 and DOTPN2) instructions that perform operations of the form ($ax \pm by$); so successive DOTP2 or DOTPN2 instructions need to be followed by explicit ADD instructions that may often need to be carefully unrolled and pipelined. On the other hand, the TMS320C64xx-C has the capacity to schedule a wider class of operations in parallel with the four multiplies using the remaining six execution units, while the SC140 is restricted to performing address arithmetic operations and memory moves on the remaining two address generation units.

Unlike the 11-stage pipeline of the TMS320C64xx-C, the SC140 uses a simple five-stage pipeline. The simpler pipeline results in fewer delay slots and simplifies programming considerably. On the TMS320C64xx-C, unrolling of loops is extensively employed in benchmarks to maximize the use of the architectural features of the processor. The setup and cleanup code for these loops results in inflated cycle counts, especially for those benchmarks involving a small number of loop iterations. For example, while the Single-Sample FIR benchmark inner loop on the TMS320C64xx-C requires (3*T/8) cycles, where T is the number of taps in the filter, housekeeping and setup require up a considerable 19 cycles. On the SC140, while the inner loop requires (T/2) cycles (a nearly 35% increase over the TMS320C64xx-C inner loop), the housekeeping overhead is only 6 cycles. This implies that the TMS320C64xx-C would begin to achieve a lower cycle count than the SC140 only for T > 104.

> *An advantage of the SC140 is its single-cycle latency on all ALU operations, including multiply-accumulate operations. Due to its deeper pipeline, the TMS320C64xx-C has one delay slot on multiply operations and three on the DOTP2 and DOTPN2 instructions. Another cycle is required for the accumulate. To achieve cycle counts comparable to the SC140, loops have to be unrolled and software pipelining used extensively, which leads to increased programming complexity.*

On the **Real Block FIR** filter benchmark, the SC140 processes four input samples at a time in the inner loop. The inner loop performs 16 multiply-accumulates using four single-cycle execution sets, resulting in a throughput of four taps per instruction cycle. The execution sets include pointer updates and load-store operations. Single-cycle execution of four multiply-accumulates in parallel with the load-store operations and zero-overhead looping give the SC140 the lowest cycle count of the benchmarked processors. While the TMS320C64xx-C also achieves a throughput of four taps per instruction, it expends cycles in setting up the software pipeline, resulting in a slightly higher cycle count. The SC140 has a cycle count that is roughly half that of the TMS320C62xx on this benchmark. This is mainly due to the fact that the SC140 has twice as many multipliers as the TMS320C62xx. The SC140 has a cycle count on the Real Block FIR benchmark that is approximately 70% below the average.

On the **Single-Sample FIR** benchmark, the SC140 inner loop processes the filter with a throughput of two taps per instruction cycle rather than the four taps per cycle achieved in Real Block FIR benchmark. The single-sample nature of the benchmark means that the SC140 cannot use circular buffering for the delay line (except with the use of significant additional memory, as described above); instead, samples are copied to update the delay line. However, the SC140 still has the lowest cycle count; its cycle count is about 55% below the average, and more than 35% below the processor with the second lowest cycle count, the Lucent Technologies DSP164xx. The DSP164xx also processes two taps per cycle, but has a higher cycle count on this benchmark because of overhead cycles consumed for filling its instruction cache. The TMS320C64xx-C uses loop unrolling and software pipelining to achieve a better throughput of eight taps per three instruction cycles, but its cycle count on the benchmark is inflated by loop setup overhead.

On the **Complex Block FIR** filter benchmark, the SC140 inner loop processes two taps per iteration and the outer loop processes two input samples per iteration. Filter throughput is largely determined by arithmetic operations in the inner loop. The SC140 performs memory accesses and pointer updates in parallel with arithmetic operations, and implements the loop with almost no overhead. In the inner loop, the SC140 is capable of performing the complex filtering at a rate of two taps in two instruction cycles (calculation of each tap spans two cycles) by executing four multiply-accumulates per instruction cycle. In the outer loop, either four multiplies or four multiply-accumulates are performed in each cycle—with the exception of one cycle spent scaling the two complex output samples for overflow protection. The TMS320C64xx-C cycle count in this benchmark is roughly equivalent to that of the SC140. Pipeline overhead on the TMS320C64xx-C is alleviated by unrolling the inner loop four times and the outer loop once and collapsing the two loops into a single loop. The SC140 has a cycle count on the Complex Block FIR benchmark that is approximately 70% below the average of the processors benchmarked, approximately equal to that of the TMS320C64xx-C and roughly half that of the TMS320C62xx.

The **LMS Adaptive** benchmark is implemented in two stages on the SC140. The output of the filter is calculated in the first stage, and the filter coefficients are updated in the second stage. The filter output is computed at a rate of four taps per instruction cycle. Two extra cycles are consumed for summation of partial results, and one extra cycle is consumed for computing the difference between the desired and actual output. In the second stage, two coefficients of the filter are updated per instruction cycle, and the delay line is updated. On the TMS320C64xx-C and TMS320C62xx, the adaptation of coefficients is pipelined and done in parallel with the execution of the FIR filter. This is mainly due to the long latency of the multipliers; if the execution of the filter was followed by the adaptation of coefficients, many cycles would be spent computing the difference between actual and desired output, and multiplying this value by the adaptation rate. However, the loop setup overhead is still considerable. The SC140 has a cycle count on this benchmark that is roughly 70% below the average, and roughly 55% below that of the processor with the second lowest cycle count, the TMS320C64xx-C.

The SC140 has a cycle count on the **Two-Biquad IIR** filter benchmark that is approximately 60% below the average. The processors with the closest cycle counts on this benchmark, the Lucent DSP164xx and Texas Instruments' TMS320C54xx, have significantly higher results. The SC140 implementation, which is completely loop-unrolled, consumes only six cycles for the two biquads.

The TMS320C64xx-C, despite also executing a completely unrolled IIR filter, has a higher cycle count than the SC140 due to instructions with multi-cycle latencies.

In the loop of the **Vector Dot Product** benchmark, the SC140 performs four multiply-accumulates per instruction cycle in parallel with data fetches. These operations are performed with almost no overhead except for two cycles for adding the partial results. Although the TMS320C64xx-C has an equivalent throughput using two parallel DOTP2 instructions, it has a significantly higher loop setup overhead. Thus, the SC140 cycle count is roughly 65% below the average and about 40% below the TMS320C64xx-C on the Vector Dot Product benchmark.

On the **Vector Add** benchmark, the SC140 performs eight additions in three instruction cycles in the inner loop. The requirement of two 16-bit reads and one 16-bit write for each output sample makes it impossible for the SC140 to fully utilize its eight 16-bit adders (using SIMD in each ALU). Full utilization of its eight adders would require eight 16-bit reads and four 16-bit writes per cycle (i.e., 192 bits per cycle), but the processor provides only 128 bits of data bandwidth. The TMS320C64xx-C is similarly bandwidth-limited. In addition, the TMS320C64xx-C has higher load latencies that warrant loop unrolling, resulting in higher overhead. In spite of its bandwidth limitations, the SC140 has a cycle count approximately 70% below the average of the processors benchmarked on the Vector Add, and about 25% below that of the next-lowest processor, the TMS320C64xx-C.

In order to utilize the SC140's multiple execution units on the **Vector Maximum** benchmark, the search is performed in three stages. In the first stage, the vector is partitioned into smaller sets and the local maximum is found for each set. In the second stage, the global maximum and the set to which it belongs are found. The last stage finds the index of the global maximum. The SC140 is capable of making eight 16-bit comparisons and storing the results into registers in one instruction cycle. This capability is utilized in the first stage. The second stage makes use of the SC140's ability to conditionally execute portions of an execution set. Part of the execution set can be executed when a condition is met, and the other part can be executed either unconditionally or if the condition is not met. This capability is used not only to find the global maximum but also to determine which set contains it. The same method is used in the last stage for finding the index of the global maximum. The SC140 has a cycle count on the Vector Maximum benchmark that is about 70% lower than the average of the processors benchmarked and approximately 25% lower than that of the Lucent DSP164xx and the TMS320C64xx-C.

The **Control** benchmark is designed to indicate a processor's memory usage on control-oriented software, and is optimized for memory usage rather than speed. The SC140 has a cycle count on the Control benchmark that is roughly 25% lower than the average of the processors benchmarked. This is largely due to the SC140's ability to conditionally execute execution sets using prefix words. While some of the other processors must make conditional jumps and branches, the SC140 can avoid these in many cases by executing or omitting an execution set based on the condition coded into the prefix word. It consumes only one cycle for such an instruction, which is significantly less than the three or four cycles typically required for a branch. The TMS320C62xx and the TMS320C64xx-C provide a similar capability; they can conditionally execute instructions within execution sets and avoid branching in many cases. In addition to its conditional execution capabilities, the SC140 makes efficient use of delayed branches and calls, resulting in a lower cycle count compared to the other benchmarked processors.

The **FFT** benchmark is implemented on the SC140 using a radix-4 algorithm and totally loop-unrolled stages. Radix-4 FFTs are addition-intensive algorithms. Throughout the radix-4 computations, the SC140 performs an average of four arithmetic operations (a combination of multiply, multiply-accumulate, addition, and subtraction) and two move operations (a combination of stores and loads) per instruction cycle yielding a cycle count that's about 75% lower than the average cycle count. The TMS320C64xx-C also uses a radix-4 FFT, but has a cycle count that is approximately 25% lower than that of the SC140. The DOTP2 and DOTPN2 instructions on the TMS320C64xx-C are particularly well tailored for the FFT and the two multiply units can achieve an effective throughput of one complete butterfly per cycle. However, since the DOTP2 and DOTPN2 instructions have a three cycle delay slot, efficient pipelining is needed to achieve this single cycle throughput. This leaves the remaining L and S units free for other arithmetic and bitshift operations, while still allowing a load/store operation on the D units. In contrast the MAC operations on the SC140 preclude any other arithmetic operations allowing only address computations on the address generation units.

The SC140 has a significantly lower cycle count on the **Viterbi** benchmark than most other processors benchmarked. Its cycle count is roughly 85% lower than the average of the benchmarked processors, and it consumes less than one-third of the cycles required by the next lowest processor, the TMS320C67xx. The cycle count of the TMS320C64xx-C, however, is slightly lower than that of the SC140. The Viterbi decoding benchmark is composed of two stages. In the first stage, soft decision values are used to generate a state transition table. In the second stage, the transition information is traced back to determine the received bit stream. The SC140 has instructions dedicated to Viterbi decoding, such as MAX2VIT and VSL. The first stage of the Viterbi decoding is intensive in additions, subtractions, comparisons, and bit manipulations. The SC140 can process eight trellis butterflies in ten cycles by pipelining the MAX2VIT and VSL instructions. In contrast, the TMS320C64xx-C can process eight trellis butterflies in just six cycles using 8-bit data. This is facilitated by packing four bytes into a single 32-bit word and using the CMPGTU4 instruction, which performs comparisons on packed 8-bit data. Such an optimization is only useful for processors with 8-bit SIMD capability, and cannot be applied to the SC140. The cycle efficiency of the loops on the TMS320C64xx-C is somewhat offset by the overhead required to prime the deep pipeline. The TMS320C64xx-C has a cycle count result that is roughly 10% less than that of the SC140.

On the **Bit Unpack** benchmark, the SC140, the TMS320C62xx, and the TMS320C64xx-C execute a single loop without conditional branches. This is possible because of their ability to conditionally execute the whole set or a subset of the instructions within an execution set. In this benchmark implementation, either one part of an execution set or the other part of the same execution set is executed depending on the value of a condition code, in a single cycle. All other processors benchmarked here must use branches for the update of the temporary buffer. Unlike the other benchmarked processors, the SC140, the TMS320C62xx and the TMS320C64xx-C have cycle counts on the Bit Unpack benchmark that are independent of the input data values. The TMS320C64xx-C has a cycle count that is about 25% lower than the SC140. This is primarily because of the ability of the TMS320C64xx-C to perform 64-bit unaligned data accesses, which are not supported on the SC140. The SC140's result is approximately 60% below the average of the benchmarked processors.

- **Execution times:** The execution-time results are shown in Figure 8.2-13. The MSC8101 executes at 300 MHz, which combines with its very low cycle counts to yield the second-lowest total normalized execution time, roughly 85% faster than the average for the fixed-point processors. The TMS320C64xx-C, projected to execute at 600 MHz, has the highest instruction cycle rate among the processors benchmarked in this report. Although the cycle count for most benchmarks are lower on the MSC8101 than on the TMS320C64xx-C, the latter is projected to operate at twice the clock speed, resulting in a net execution time that is about 30% faster than the MSC8101. However, this result assumes all data and instructions are preloaded into the TMS320C64xx L1 cache, and is therefore a best-case result.
- **Cost-execution time:** The cost execution-time results are shown in Figure 8.3-13. Motorola has provided a price of $96 in quantities of 10,000 for the MSC8101. The MSC8101's price tag is aggressively low compared to the $201 TMS320C6203 but significantly higher than that of other conventional DSPs. Because of its very fast execution times, the total normalized cost-execution time result of the MSC8101 is better than that of all of the other benchmarked processors, except the Motorola DSP56853. It has a total normalized cost-execution time result that is less than one-third of that of the TMS320C6203 and roughly half the average of fixed-point processors.
- **Energy Consumption**: The energy consumption figures are shown in Figure 8.4-13A and B. The Motorola MSC8101 has the lowest total normalized energy consumption of all benchmarked processors, with a result that is about 45%

lower than the average. The processor with the next-lowest result, the TMS320C5510, consumes more than twice as much energy as the MSC8101.

Memory Usage

The focus in our memory usage analysis is on Control benchmark memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. This is because control-oriented instructions often make up the bulk of the software in an application, but take up only a fraction of the application's instruction cycles. Hence, unlike the other benchmarks, the Control benchmark is optimized for minimum memory usage rather than maximum speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP software. Finally, we discuss constant and non-constant data memory usage.

Since memory usage is independent of the state of the cache, we do not distinguish between the TMS320C64xx and the TMS320C64xx-C when discussing memory usage.

- **Control benchmark memory usage:** Figure 8.5-9 presents the Control benchmark total memory usage graphically. The SC140 has a total memory usage on the Control benchmark that is roughly 35% below the average of the benchmarked processors. There are several reasons for this lower memory usage. First, the SC140 groups 16-bit instructions into execution sets using two methods: prefix grouping and serial grouping. In prefix grouping the execution set is preceded by a 16- or 32-bit prefix; the prefix requires two 16-bit words only when registers D8-D15 or R8-R15 are used in the execution set. The prefix provides information for the instruction decode unit about the number of instructions that are in the group, short looping, and conditional execution for the whole set or a subset. Embedding this information into the prefix results in more compact code. For most processors, the loop start and end addresses for hardware loops are either written into a register or embedded into an instruction (in which case the processor writes the values into registers that are invisible to the programmer). For a processor with 16-bit program memory addresses this results in usage of at least two 16-bit words. These two 16-bit words are also used in the SC140, but they are better utilized because they also carry conditional execution information. In some cases, this conditional execution information can be used to reduce the number of branches—and hence, the number of instructions.

  Although the SC140 has significantly lower total memory usage on the Control benchmark compared to the TMS320C62xx and the TMS320C64xx, which both use 32-bit instructions, its non-constant data memory usage (stack usage) is three times that of the Texas Instruments processors. The SC140 can group one or two PUSH instructions in an execution set, and in either case the stack pointer is incremented only once by 8 bytes. However, executing multiple instructions in parallel takes up more memory than issuing each instruction by itself. Thus, there is a trade-off between program memory and non-constant data memory. If the PUSH instructions are issued separately (as is done in the Control benchmark), the size of the non-constant data memory is significantly increased.

> *The SC140's total memory usage result on the Control benchmark indicates that the SC140 will have better control code density than most other processors benchmarked here—and much better than that of the Texas Instruments TMS320C62xx and the TMS320C64xx. The SC140 has a control-code density that is even better than that of most of the conventional DSP processors benchmarked here, which is unusual for a VLIW-based processor.*

- **Program memory usage:** The program memory usage results of the processors benchmarked are shown in Figure 8.5-13. The SC140 has a total normalized program memory usage on DSP algorithm code that is about twice the average of that of the TMS320C54xx and the DSP16xxx, but much lower than that of the TMS320C62xx and the TMS320C64xx-C. The SC140 often must issue multiple instances of the same instruction with different operands at once to make effective use of its multiple execution units. For example, for every MAC instruction issued by the TMS320C54xx, there are typically four MAC instructions issued on the SC140, resulting in higher program memory usage. The TMS320C62xx and the TMS320C64xx have multiple execution units, like the SC140, and use instructions that are twice as wide. Since instruction latencies are short, the SC140 does not require significant software pipelining. In contrast, the TMS320C62xx and the TMS320C64xx have many instructions with multi-cycle latencies, forcing the programmer on compiles to use a significant level of software pipelining to achieve maximum speed—increasing program memory usage. Overall, the SC140 has roughly half the total normalized program memory usage of the TMS320C62xx and the TMS320C64xx.

> *VLIW-based processors often require significantly more program memory to implement DSP algorithms than conventional DSP processors. The SC140 is not an exception, but it does provide significantly better code density than the other fixed-point VLIW-based processors benchmarked here, the TMS320C62xx and the TMS320C64xx.*
>
> *Because control code density is typically more important than DSP algorithm code density, high DSP algorithm program memory usage is generally not a serious disadvantage. The SC140's excellent code density on the Control benchmark indicates that its high DSP algorithm program memory usage will be offset in many applications by its low control-code memory use.*

- **Data memory usage:** Data memory usage results are shown in Figure 8.5-14 and Figure 8.5-15. The SC140 constant and non-constant data memory usage are generally as expected for a processor that uses 16-bit data. It should be noted that, on several benchmarks, one processor uses a small amount of constant data while all of the others use none. For example, on the Bit Unpack benchmark the TMS320C54xx uses two bytes of constant data while the other processors use none. In these cases, the normalized results are not particularly meaningful, and the reader should examine the actual memory usage rather than the normalized result.

> *To summarize our results, the SC140 has the lowest cycle counts of the processors benchmarked here on most of the benchmarks. Only the TMS320C64xx-C has lower cycle counts on some of the benchmarks. With an instruction cycle rate of 300 MHz, the MSC8101's execution-time performance is significantly faster than those of all other processors benchmarked here, with the exception of the TMS320C64xx-C.*
>
> *On DSP algorithm software, the SC140's total memory usage is higher than that of the other benchmarked DSP processors except for the TMS320C62xx and the TMS320C64xx-C. Based on the memory usage results on BDTI's Control benchmark, however, control-oriented software memory efficiency on the SC140 can be expected to be very good. The SC140 achieves much better code density than the Texas Instruments TMS320C62xx and the TMS320C64xx on the Control benchmark.*

Cost

Motorola MSC8101 Cost

According to Motorola, the MSC8101 is priced at $96 in quantities of 10,000.

Fabrication Details

Motorola MSC8101 Fabrication

According to Motorola, the MSC8101 is fabricated in a 0.13 µm effective gate length CMOS process using Motorola's copper interconnect process technology. Other SC140-based devices may be fabricated in other processes.

Development Tools

Currently, StarCore provides a variety of baseline application development tools for the SC140 core, including the following:

- Assembler
- Linker

- Optimizing C/C++ compiler
- Instruction set simulator

These common baseline tools are integrated through an application program interface (API). Third parties and customers can integrate their own development tools using the API. In addition, Lucent has extended its LUxWORKS development tools environment to support the StarPro chip.

A number of third parties have announced plans to develop tools for the SC140. Green Hills Software, Inc. will develop an Integrated Development Environment (MULTI IDE), including a C/C++ compiler and debugger, for the SC140. Enea OSE Systems will port its RTOS to the SC140. According to StarCore, another RTOS, EmPower-RTXC from Embedded Power Corporation will also be available in the future.

Applications Support

StarCore will offer libraries of C-callable hand-optimized assembly routines on their website, *www.starcore-dsp.com*. According to StarCore, the following libraries are available in alpha versions:

- Signal Generation Function Library
- Control Function Library
- Filtering Function Library
- Frequency Domain Function Library
- Image Processing Function Library
- Math Utility Function Library
- Modulation and Communication Function Library
- Complex Vector and Matrix Function Library

Signals and Software, Ltd., a UK-based DSP software vendor, has announced that it will develop telecommunications software including speech coders, modems, and echo cancellers for the SC140.

Advantages

- VLIW-based execution is highly parallel
- Conditional instruction execution of nearly all instructions
- Specialized support for Viterbi decoding
- All DALU registers can be used as accumulators
- Good on-chip data memory bandwidth
- SIMD ALU operations
- Nestable multi-instruction hardware looping
- Very good execution times on the BDTI Benchmarks (MSC8101)

- Very good energy efficiency on the BDTI Benchmarks (MSC8101)
- Very good cost-execution times on the BDTI Benchmarks (MSC8101)
- Very good memory usage result on Control benchmark

Disadvantages

- More complicated programming model than conventional, single-issue DSP processors
- No separate status bits for each data path, or for SIMD operations
- Performance penalties for using off-chip memory are severe (MSC8101)

7.11 Texas Instruments TMS320C2xxx Family and T320C2xLP Core

BDTImark2000 Score: Not Available

Introduction

The TMS320C2xxx is a family of 16-bit fixed-point conventional DSP processors based on the T320C2xLP DSP core. Texas Instruments targets TMS320C2xxx processors at high-volume, cost-sensitive applications such as motor control, telephones, hard disk drives, and modems. The TMS320C24x and TMS320F24x devices, along with the newer TMS320LC24xx and TMS320LF24xx processors, are aimed at motor control applications. For sufficiently high volumes, Texas Instruments can build customer-designed SoCs based on the T320C2xLP core. The fastest members of the family run at 40 MIPS and 3.3 volts. Some low-power members run at 20 MIPS and 3.3 volts. The TMS320C24x and TMS320F24x are only available at 20 MIPS and 5.0 volts.

The TMS320C2xxx family was introduced in 1995 and represents a cross between Texas Instruments' earlier TMS320C2x and TMS320C5x families. The TMS320C209 processor was the first member of the family and has some features that are different from the later family members. The TMS320C2xxx's data path is essentially identical to that of the 16-bit fixed-point TMS320C2x.

The TMS320C2xxx is mostly compatible at the assembly-language source code level with the TMS320C2x; some instructions have been changed, and others have been added. The core uses the same pipeline structure as the TMS320C5x and is fully static, enabling the system designer to slow or stop the processor's clock to reduce power consumption. A summary of TMS320C2xxx family members is shown in Table 7.11-1. In Texas Instruments' TMS320C2xxx part numbers, "L" indicates a 3.3 volt device; devices without an "L" in their part numbers are 5.0 volt devices. Devices with "C" in their part numbers use mask ROM; those with "F" in their part numbers use flash EEPROM.

> *TMS320C2xxx processors are well suited for low-cost applications that are not computationally demanding. The TMS320C2xxx is aggressively priced in quantities of 10,000.*
>
> *However, the maximum instruction cycle rate of 40 MIPS is quite low compared to that of many competing fixed-point DSP processors. Unlike most of the other processors in this report, the speed of the TMS320C2xxx has not increased since the last edition of this report nearly two years ago.*

This description covers both the T320C2xLP core and the TMS320C2xxx processors. Differences between TMS320C2xxx packaged processors and the T320C2xLP core are noted in the text. We use the term TMS320C2xxx to indicate features that are common to the TMS320C2xx, TMS320F2xx, TMS320LC24xx, and TMS320LF24xx family members. We use the term TMS320C24xx to indicate features that are common only to the TMS320LC24xx and TMS320LF24xx family members.

Part	Max. Speed (MIPS)	On-Chip Memory			Comments
		Dual-Access Data RAM	Single-Access Program/ Data RAM	Program ROM	
C203	40	544x16	—	—	Synchronous and asynchronous serial ports
C206/ F206	40	544x16	4Kx16	32Kx16	ROM and flash EEPROM versions, synchronous and asynchronous serial ports
C209	28	544x16	4Kx16	4Kx16	Limited features compared to other TMS320C2xxx processors
C240/ F240	20	544x16	—	16Kx16	ROM and flash EEPROM versions, synchronous and asynchronous serial ports, dual 10-bit A/D converters, 28 bidirectional I/O pins, 12 pulse-width modulation outputs, 5 timers
F241	20	544x16	—	8Kx16	Flash EEPROM, synchronous and asynchronous serial ports, control area network unit, 10-bit A/D converter, 26 bidirectional I/O pins, 8 pulse-width modulation outputs, 3 timers
C242	20	544x16	—	4Kx16	ROM, asynchronous serial port, 10-bit A/D converter, 26 bidirectional I/O pins, 8 pulse-width modulation outputs, 3 timers
F243	20	544x16	—	8Kx16	Flash EEPROM, synchronous and asynchronous serial ports, control area network unit, 10-bit A/D converter, 26 bidirectional I/O pins, 8 pulse-width modulation outputs, 3 timers

TABLE 7.11-1. TMS320C2xxx processor summary (continued on next page).

The T320C2xLP core can be used in SoCs designed by customers and fabricated by Texas Instruments. The T320C2xLP includes 544x16 bits of dual-access RAM (DARAM) on-core. According to Texas Instruments, the T320C2xLP core can operate at speeds of up to 40 MIPS at 3.3 volts. The core can be used as part of an ASIC implemented using a standard cell or gate array methodology. Texas Instruments provides a

Part	Max. Speed (MIPS)	On-Chip Memory			Comments
		Dual-Access Data RAM	Single-Access Program/ Data RAM	Program ROM	
LC2402/ LF2402	40	544x16	—	4Kx16/ 8Kx16	ROM and flash EEPROM+boot ROM versions, asynchronous serial port, 10-bit A/D converter, 21 bidirectional I/O pins, 8 pulse-width modulation outputs, 3 timers
LC2404	40	544x16	1Kx16	16Kx16	ROM, synchronous and asynchronous serial ports, 10-bit A/D converter, 41 bidirectional I/O pins, 16 pulse-width modulation outputs, 5 timers
LC2406/ LF2406	40	544x16	2Kx16	32Kx16	ROM and flash EEPROM+boot ROM versions, synchronous and asynchronous serial port, control area network unit, 10-bit A/D converter, 41 bidirectional I/O pins, 16 pulse-width modulation outputs, 5 timers
LF2407	40	544x16	2Kx16	32Kx16	Flash EEPROM+boot ROM, synchronous and asynchronous serial ports, control area network unit, 10-bit A/D converter, 41 bidirectional I/O pins, 16 pulse-width modulation outputs, 5 timers, external memory interface

TABLE 7.11-1. TMS320C2xxx processor summary (continued from previous page).

variety of memory, peripheral and support blocks that can be integrated with the core on an SoC. Contact Texas Instruments for further information on their core-based SoC offerings.

The availability of the T320C2xLP core in packaged processors and as a core for use in SoC designs provides a natural roadmap for designers to migrate from TMS320C2xxx-based board-level designs to T320C2xLP core-based SoCs when volumes become sufficiently high.

Architecture

TMS320C2xxx processors are based on the T320C2xLP core's 16-bit fixed-point data path, program control unit, and bus structure, but feature different peripherals and amounts of on-chip memory. The architecture of the T320C2xLP core is shown in Figure 7.11-1.

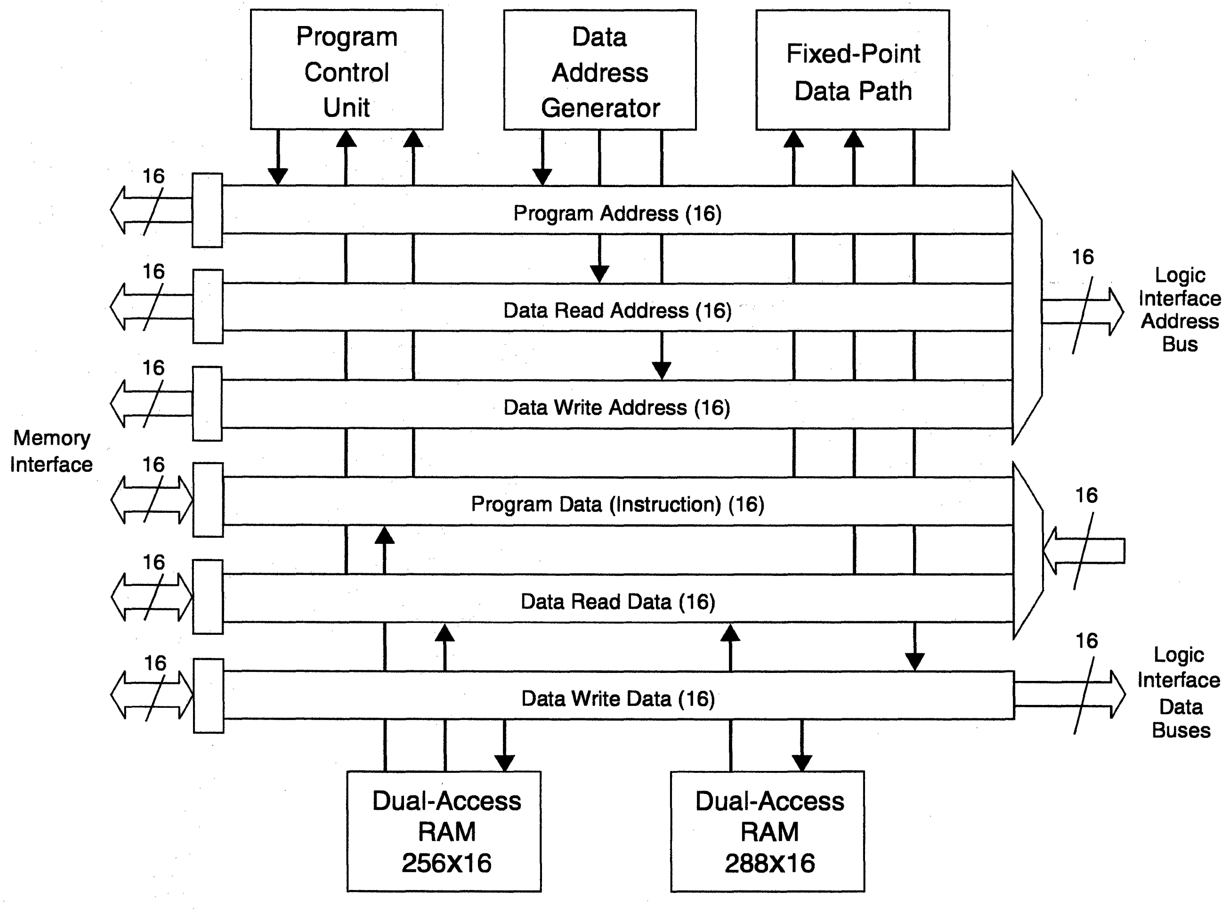


FIGURE 7.11-1. T320C2xLP core architecture.

Data Path

The T320C2xLP core and the TMS320C2xxx processors use a 16-bit data path. The data path centers on a 16 × 16 → 32-bit single-cycle multiplier, a 32-bit ALU, a limited-capability input barrel shifter, a limited-capability output barrel shifter, and a single 32-bit accumulator.

Arithmetic operations include add and subtract, and logical operations include *and*, *or*, *exclusive-or*, and *not*. One ALU input always comes from the accumulator, while the other can come from either the product register (unshifted, shifted right by 6 bits, or shifted left by 1 or 4 bits) or the data bus (shifted left by 0 to 16 bits); these shifts are performed by the input shifter. Both the 16-bit upper half of the accumulator and the 16-bit lower half of the accumulator can be left-shifted by 0 to 7 bits before being written to the data bus; these shifts are performed by the output shifter.

Signed/signed or unsigned/unsigned inputs to the multiplier come from the 16-bit T register and either the program or data bus; multiplier outputs are placed in the product (P) register. The multiplier performs integer multiplication by default, but the processor can be configured to shift the product one bit left, thus allowing fractional multiplication. A single multiply-accumulate (MAC) operation takes three instruction cycles. When repeated in a single-instruction repeat loop, multiply-accumulate operations have single-cycle throughput.

> *Multi-cycle MAC operation is a disadvantage of the TMS320C2xxx. Good MAC throughput can only be achieved when MACs are executed in a single-instruction repeat loop.*

No guard bits are available in the accumulator. To prevent overflow, the output of the product register may be right-shifted by six bits before being presented to the ALU for accumulation. This allows up to 128 multiply-accumulate operations before overflow becomes a possibility.

> *The lack of accumulator guard bits reduces flexibility. However, the option to shift the contents of the product register six bits to the right before it is added or transferred to the accumulator creates an effect that is similar to guard bits and is sufficient in most applications.*

Accumulator saturation on overflow can be enabled via a mode bit, which is set or reset by special instructions. The ALU features a carry bit, permitting extended-precision arithmetic and rotate-through-carry instructions. In addition to the carry bit, a status bit indicates whether the last operation produced overflow. 16-bit round-to-nearest rounding is supported in hardware.

The TMS320C2xxx supports one-bit logical left shift and one-bit logical or arithmetic (depending on a mode bit) right shift of the accumulator.

> *The fact that the ALU does not support a single-cycle multi-bit left shift operation is a limitation which may complicate programming*

and hinder performance in some applications. However, the input and output shifters often allow an operand or result to be shifted without requiring explicit shift instructions.

The TMS320C2xxx does not have hardware support for fast exponent detection and normalization, but an instruction that performs iterative exponent detection and normalization on the accumulator is provided.

ALU operations are performed with single-cycle throughput. The TMS320C2xxx cannot perform an instruction fetch and two data reads simultaneously; thus, although a multiply-accumulate instruction is provided, a multiply-accumulate operation has a throughput of two instruction cycles unless it is executed from the single-instruction repeat buffer (described below).

A drawback of the TMS320C2xxx data path is that it includes only one accumulator, and all ALU operations use the accumulator as one of their input operands. As a result, the accumulator often becomes a bottleneck in the architecture.

The single accumulator means that the processor is not well suited for arithmetic using complex numbers.

Memory System

This section describes the memory system of the T320C2xLP core. Unless otherwise noted, this also applies to the memory system of the TMS320C2xxx processor.

The T320C2xLP core and TMS320C2xxx processors use separate program and data memory spaces. Each memory space can contain up to 64 Kwords of word-addressable 16-bit memory. In memory-direct addressing modes, memory is divided into pages of 128 words each. The T320C2xLP core contains three separate sets of on-core 16-bit address and data buses: the program bus set, the data read bus set, and the data write bus set.

The T320C2xLP includes 544x16 bits of dual-access RAM (DARAM) on-core. DARAM supports one read and one write operation (but not two read or two write operations) per instruction cycle. The on-core DARAM is broken up into two blocks. The first block, which is 256 words in length, can be mapped into program space or data space (but not both) under software control. The second block, which is 288 words long, is always mapped into data space. DARAM is included as part of the core to meet timing constraints, and cannot be expanded off-core.

The inability to add additional dual-access RAM may be a problem in some applications, as the dual-access RAM is used to efficiently implement delay lines (e.g., for FIR filters) on TMS320C2xxx processors.

Texas Instruments provides four types of memory that can surround the T320C2xLP core: mask ROM, EPROM, flash EEPROM, and single-access RAM

(SARAM). These memory blocks are specially designed for use with the T320C2xLP core. Unlike DARAM, they support only one access per instruction cycle. They connect to the core through a special memory interface, described in the *External Memory Interface* section below.

Because the DARAM memory is part of the T320C2xLP core, all TMS320C2xxx processors include 544x16 of on-chip dual-access RAM. In addition, the TMS320C206, TMS320F206, and TMS320C209 include two 2 Kword blocks of single-access RAM that is mapped into both program and data space. The TMS320LC2404 has one 1 Kword block of single-access RAM, while the TMS320LC2406, TMS320LF2406, and TMS320LF2407 each have one 2 Kword block of single-access RAM that is mapped into both program and data space. Each of the single-access RAM blocks allows one access per instruction cycle.

The flash EEPROM on the TMS320F240, TMS320F241 and TMS320F243 is comprised of a single bank. The flash EEPROM on the TMS320F206 and TMS320LF2402 is divided into two banks; the processor can execute instructions from one bank while reprogramming the other. For more flexibility, the 32 Kwords of flash EEPROM on the TMS320LF2406 and TMS320LF2407 is divided into four banks instead of two. In addition to flash EEPROM, the TMS320LF2402, TMS320LF2406, and TMS320LF2407 also contain 256 words of factory programmed boot ROM to facilitate reprogramming. For more information about bootstrap capabilities, see the *Bootstrap Loading* section below. Please refer to Table 7.11-1 for the configuration of on-chip memory in each TMS320C2xxx family member.

On-chip flash EEPROM memory greatly eases development and prototyping and facilitates field upgrades or last-minute programming of devices. Thus, the support for flash EEPROM in the TMS320F206, TMS320F24x, and TMS320LF24xx is an advantage.

Coupled with appropriate types of memory, the three sets of buses allow the T320C2xLP to achieve one program fetch, one data read, and one data write per instruction cycle; i.e., a data memory bandwidth of forty million 16-bit words/second for reads and forty million 16-bit words/second for writes on a 40 MIPS TMS320C203. (The ability to perform a data read and a data write in the same instruction cycle is important for compatibility with previous Texas Instruments fixed-point processors.)

In normal operation one instruction is fetched from program memory and one operand can be fetched from data memory in one instruction cycle. This implies that instructions requiring two data fetches (such as multiply-accumulate) execute in two instruction cycles. However, as on the TMS320C2x, the T320C2xLP core and TMS320C2xxx processors feature a single-instruction repeat buffer that frees the program buses for data fetches when an instruction is being repeated. In this case, instructions requiring two memory reads can execute in one instruction cycle.

To achieve maximum performance, TMS320C2xxx instructions that require two operand fetches must be executed from a single-instruc-

tion hardware loop. This complicates device programming and lowers performance when blocks of instructions must be repeated as a group (e.g., in an FFT butterfly).

About half of the TMS320C2xxx family members have only 544 words of on-chip RAM. This is an unusually small amount of memory.

External Memory Interface

The T320C2xLP core provides two separate external interfaces. The first, called the "memory interface," is specifically intended for connection to the on-chip memory blocks described above. The second, the "logic interface," presents a multiplexed external interface intended for connection to off-core peripherals and custom circuitry. To simplify development, the logic interface is similar to the external memory interfaces on the predecessor TMS320C2x and TMS320C5x processors.

The T320C2xLP memory interface brings out the three sets of on-core buses described above (three 16-bit address buses and three 16-bit instruction and data buses) plus associated control lines, such as read (or write) strobes and "ready" inputs. A unique aspect of the interface is the presence of three sets of five input pins: PRAMEND0-4, PROMEND0-4, and DRAMEND0-4. These inputs indicate to the core the ending addresses (which must be on 2 Kword boundaries) of program RAM, program ROM, and single-access data RAM, respectively. This allows SoCs created using the T320C2xLP core to have varying amounts of memory on-chip without forcing the designer to create timing-critical address decoding logic. Memory accesses past the boundaries specified by the PRAMEND0-4, PROMEND0-4, and DRAMEND0-4 input pins occur over the logic interface. As discussed above, the memory interface is designed solely for connecting to on-chip memory blocks, and is not intended for connection to peripherals.

The T320C2xLP logic interface provides one 16-bit address bus (which multiplexes the three on-core address buses), one 16-bit data input bus, and one 16-bit data output bus. The on-core instruction, data read, and data write buses are multiplexed onto the two data buses. This interface differs from the TMS320C2x and TMS320C5x external interface in that the T320C2xLP logic interface data bus is not bidirectional, but rather uses one data input bus and one data output bus. Texas Instruments states that this was done to simplify interfacing and to give users as much freedom as possible in designing T320C2xLP-based SoCs.

The logic interface can achieve one external read per instruction cycle, assuming zero wait states. Writes over the logic interface take at least two instruction cycles, thus maximum memory bandwidth is, e.g., forty million 16-bit words/second to the processor core or twenty million16-bit words/second from the processor core on a 40 MIPS TMS320C203.

Like the TMS320C5x, the TMS320C2xxx requires multiple instruction cycles for writes to external memory. The additional cycles *required to access external data memory are a significant disadvantage, since the on-chip data memories are fairly small.*

The T320C2xLP logic interface includes a program strobe, a data strobe, and an I/O strobe (discussed below) to identify the type of external access the processor is making. Read and write strobes are also provided. Externally requested wait states are supported via the READY input.

Like the earlier TMS320C2x and TMS320C5x, the T320C2xLP core supports shared memory via its bus request (BR) pin and the READY pin. Under software control, up to 32 Kwords of the processor's upper memory space can be defined to be "global memory." Accessing this memory causes the T320C2xLP to assert the bus request signal and wait for the READY signal to be asserted in acknowledgment. In a shared bus design, an external arbitration device treats BR assertion as a request for access to a shared resource (such as a bus or memory) and responds with READY when the shared resource is available. Global memory can alternatively be used to access up to 32 Kwords of additional unshared external memory by using the BR pin as an additional memory decode signal.

Unlike the TMS320C2x and TMS320C5x, the T320C2xLP core does not include a HOLD input to stop the processor from accessing its external bus. Texas Instruments notes that an SoC designer using the T320C2xLP core can create this functionality via tri-state buses and appropriate logic connected to the core's READY pin. On all TMS320C2xxx processors other than the TMS320C209 the HOLD operation is supported, allowing other devices to take control of the TMS320C2xxx external bus.

The T320C2xLP also provides a 64 Kbyte memory space for I/O, accessible via IN and OUT instructions. I/O space operations function like program or data space external bus cycles, but assert a special I/O strobe signal to indicate to external devices that an I/O cycle is occurring.

The TMS320C2xxx external memory interface pins match the logic interface of the T320C2xLP's core, with a few exceptions. First, the TMS320C2xxx single external data bus is bidirectional, as on the TMS320C2x and TMS320C5x. Second, the TMS320C2xxx adds a programmed wait-state generator that supports from zero to seven wait states which can be separately selected for data, I/O, or upper or lower program space accesses. On the TMS320C209 the programmed wait-state generator only allows zero or one wait state and does not distinguish between upper and lower program space. Lastly, the external memory interface is not available at all on the TMS320C24x, TMS320F24x or TMS320C24xx, except for the TMS320C240, TMS320F240, TMS320F243, and TMS320LF2407.

The similarity between the T320C2xLP core's external interfaces and the TMS320C2x/TMS320C5x/TMS320C2xxx processors' external interfaces simplifies migration from older off-the-shelf parts to a core-based SoC design.

Address Generation Unit

The T320C2xLP core and TMS320C2xxx processors support immediate data, and register-direct, paged memory-direct, and register-indirect addressing. Paged memory-direct addressing uses a single 9-bit data page register that, combined with a 7-bit direct memory address specified in the instruction word, addresses the entire 64 Kword address space. Pages are thus 128 words long. Register-indirect addressing uses one of eight address registers (AR0-AR7); the register used is selected by an address register pointer register. Register-indirect modification modes include post-increment by one, post-decrement by one, post-increment or decrement by the contents of address register AR0, and no update. Additionally, most instructions that support register-indirect addressing also allow the user to select the next address register to be used by modifying the address register pointer register.

The TMS320C2xxx supports bit-reversed addressing through reverse carry propagation. Using this mode, the currently selected address register pointer is used for address generation, but the register is post-modified by adding or subtracting address register AR0 using reverse-carry arithmetic.

Modulo addressing is not supported.

> *The restriction that only address register AR0 can be used for post-increment and post-decrement by a programmed offset is limiting in some applications.*
>
> *The lack of modulo addressing is an inconvenience. However, the MACD instruction (discussed below) can be used to implement a delay line without overhead when performing a vector dot product.*

Pipeline

The T320C2xLP core and TMS320C2xxx processors use a four-stage partially interlocked pipeline identical to that used in the earlier TMS320C5x family. The pipeline is made up of fetch, decode, operand read, and execute/write stages. This represents a significant difference from the TMS320C2x family, which used a three-stage pipeline, and results in different instruction cycle counts for many instructions. The deeper TMS320C2xxx pipeline is visible to the programmer. For example, branches take four instruction cycles. Delayed branches are not available.

Although the TMS320C2xxx pipeline is the same as that of the TMS320C5x family, the TMS320C2xxx avoids the most serious pipeline hazard present in the TMS320C5x. On the TMS320C5x, certain processor registers can be accessed by writing to certain memory locations. Due to pipeline effects, values written to address registers via memory writes are not available for two instruction cycles after the values are written. The T320C2xLP core and TMS320C2xxx processors avoid this problem by not providing memory-mapped access to processor registers. Some pipeline hazards remain, however:

- If an instruction changes the global memory map (by writing to a designated register), the next instruction will use the previous memory map.
- If any of the two instructions immediately following the NORM instruction modify the address register used by the NORM instruction or modify the address register pointer register, the address register or address register pointer register will be modified *before* the NORM instruction is executed.

Instruction Set

Tables 7.11-2 and 7.11-3 summarize the instruction set and registers of the T320C2xLP core and TMS320C2xxx processors. The T320C2xLP core and the TMS320C2xxx processors use mostly 16-bit instructions, but some instructions occupy two words of program memory.

Assembly Language Format

The T320C2xLP core and the TMS320C2xxx processors are mostly compatible at the assembly language source code level with the TMS320C2x, and are compatible with a subset of the TMS320C5x instruction set. Texas Instruments states that T320C2xLP/TMS320C2xxx object code runs on TMS320C5x processors (although the reverse is not true). TMS320C2x object code does not run on the T320C2xLP core or TMS320C2xxx processors.

The TMS320C2xxx instruction set differs in several ways from that of the TMS320C2x. The most significant of these differences are:

- With its four-stage pipeline, many instructions take a different number of instruction cycles to execute on the T320C2xLP than on the TMS320C2x.
- The T320C2xLP includes conditional call and return instructions that do not exist on the TMS320C2x.
- The T320C2xLP includes INTR and NMI software interrupt instructions that do not exist on the TMS320C2x.
- T320C2xLP conditional branch instructions can test multiple conditions. TMS320C2x instructions can only test a single condition.

Registers	Width	Purpose
ACC	32 bits	Accumulator
T	16 bits	Multiplier input register
AR0-AR7	16 bits	Address registers
ARP	3 bits	Address register pointer register

TABLE 7.11-3. T320C2xLP/TMS320C2xxx registers.

- T320C2xLP IN and OUT instructions can address up to 65,536 I/O ports, as opposed to 16 on the TMS320C2x.
- The TMS320C2x serial port configuration instructions are not available on the T320C2xLP and TMS320C2xxx processors. Instead, serial port configuration on these devices is handled via registers mapped into I/O space.

Class	Instructions
Arithmetic	Absolute value; add with 0- to 15-bit left shift; add with carry; add to high accumulator; add unsigned; accumulate; accumulate and load multiplier input; accumulate, load multiplier input, and move data in memory; negate; subtract with 0- to 15-bit left shift; subtract with borrow; subtract from high accumulator; subtract unsigned; subtract product and load multiplier input
Multiplication	Multiply, multiply-accumulate, multiply-accumulate with data load and store, multiply-subtract, unsigned multiply, square-accumulate, square-subtract
Logic	*And*, *or*, *exclusive-or*, *not*
Shifting	Arithmetic/logical shift left/right by one bit (input and output shifters before and after the accumulator provide alternative shift counts)
Rotation	Rotate left/right one bit
Conditional Execution	Conditional subtract
Comparison	Compare address register with AR0
Looping	Repeat single instruction
Branching	Conditional and unconditional branch
Subroutine Call	<u>Conditional</u> and unconditional call, <u>conditional</u> and unconditional return
Bit Manipulation	Bit test
Special Function	Block data-to-data memory move, block program-to-data memory move, read/write data to/from I/O port, move data word in memory to next higher memory location, normalize iteration, pop, push, wait for interrupt

TABLE 7.11-2. T320C2xLP and TMS320C2xxx instruction set summary. New instructions not provided by the TMS320C2x are underlined.

The T320C2xLP core and TMS320C2xxx processors use the traditional opcode-operand assembly language format. For example, the instructions:

```
RPTK  15
MACD  0FF00H,*-
```

implement the kernel of a 16-tap FIR filter. In this example, the MACD (multiply-accumulate with data move) instruction is repeated 16 times. Each execution of MACD accumulates the previous product in the accumulator; fetches two operands, one from program memory (starting at address 0xFF00 and increasing) and one from data memory via register-indirect addressing (starting wherever the currently selected address register points and decreasing); multiplies these operands, depositing the result in the product register; and writes the value read from data memory back to data memory at one location higher than the location from which it was read, implementing a delay line. The MACD instruction requires that the operand located in data memory be located in *on-chip* data memory. If the operand is located in external data memory, the delay line update operation will not be performed.

> *A drawback of the TMS320C2xxx is that its MACD instruction uses memory-direct addressing for one of its operands (the 0xFF00 in the above example). This makes it difficult, for example, to write an FIR filter routine where the locations of both the coefficients and data are specified by addresses stored in registers.*

As on the TMS320C1x, TMS320C2x, and TMS320C5x, the T320C2xLP core and TMS320C2xxx processors handle indirect addressing somewhat differently from other DSP processors. In particular, most processors allow the programmer to directly specify which address register should be used in an instruction. On the T320C2xLP and TMS320C2xxx, however, the current instruction specifies the address register to be used by the *next* instruction by specifying a value to be loaded into the "address register pointer" (ARP); for example:

```
LARP  AR3
ADD   *,AR7
ADD   *
```

The LARP instruction makes the address register pointer point to AR3. The first ADD instruction adds the value pointed to by AR3 (denoted by "*") to the accumulator and sets the ARP for the *next* instruction to AR7. The second ADD instruction adds the value pointed to by AR7 to the accumulator and does not update the ARP.

> *Having to specify the address register to be used in the* next *instruction requires programmers to think "one step ahead" compared to most processors. Programmers seem to rapidly adapt to this style of coding, however.*

Parallel Move Support

The TMS320C2xxx uses operand-related parallel moves. The few instructions that access two operands in parallel (such as multiply-accumulate and block move) must be executed within a single-instruction repeat loop to achieve single-cycle operation.

Some instructions allow one operand-unrelated move; e.g., the LTA instruction that loads a multiplier input and accumulates the previous product in the accumulator.

Orthogonality

The TMS320C2xxx instruction set is not particularly orthogonal, but is fairly consistent. This is largely because the number of registers to choose from is extremely limited (essentially consisting of the accumulator, the T register, and eight address registers), which simplifies the instruction set encoding. On the positive side, instructions that use memory-direct or register-indirect addressing all have the same set of allowable address register and update combinations. Unfortunately, the number of update modes is small. Other addressing modes (notably short and long immediate data) are available only for some instructions.

The following aspects detract from the orthogonality of the TMS320C2xxx instruction set:

- Extensive use of mode registers. The mode register approach reduces the number of actions that must be encoded within an instruction word, allowing a better representation of those that are encoded. Unfortunately, while this approach increases the number of available instructions and enables more combinations of operands, it complicates programming.

  The reliance on mode registers complicates programming of the TMS320C2xxx for several reasons. First, some instructions use mode registers while others do not. Second, understanding the behavior of many instructions requires knowledge of the contents of various mode registers. Additionally, considerable overhead may be required to configure various registers before executing instructions that depend on their contents.

  For example, in a multiply-accumulate operation, the multiplier output is shifted depending on the pre-scale mode bits. In addition, accumulator operations are affected by the sign-extension and saturation mode bits.

- Limited set of dual parallel move instructions. Multiply-accumulate, block move, and table read/write instructions are the only instructions on TMS320C2xxx processors that allow multiple data accesses in a single instruction cycle.

Execution Times

Most single-word T320C2xLP/TMS320C2xxx instructions executed from on-chip program memory take one instruction cycle to execute. However, instructions that access multiple data words, such as the multiply-accumulate instruction, may take two or three instruction cycles unless executed from a single-instruction repeat buffer. All taken branches take four instruction cycles; branches that are not taken take two instruction cycles. Two-word instructions take at least two instruction cycles.

T320C2xLP/TMS320C2xxx instruction execution times differ from those of the TMS320C2x for some instructions due to the deeper pipeline.

Instruction Set Highlights

Noteworthy instructions on the T320C2xLP core and TMS320C2xxx processors include:

- The multiply-accumulate with data move instruction fetches two data words from memory, multiplies and accumulates them, and moves one of the words up one location in memory, implementing a delay line.
- Two block data move instructions move data from program to data memory or vice versa.
- A normalization iteration instruction is available.

One of the chief drawbacks of the instruction set is that the multiply-accumulate instruction must be executed from within a single-instruction hardware loop to achieve single-cycle throughput. The overhead associated with setting up such a loop makes the processor poorly suited for tasks that require multiplies on small blocks of data, since the smaller the block, the greater the percentage of time that is consumed by overhead.

Execution Control

Clocking

The T320C2xLP core uses an externally generated clock signal to generate a master clock that runs at twice the instruction rate of the processor. However, the core contains an on-chip phase-locked loop frequency synthesizer that can be used to increase or decrease the frequency of the supplied clock signal. Five pins can be asserted or deasserted in various combinations to achieve input clock to master clock rate ratios of 1:2, 1:1, 2:1, 3:2, 3:1, and 4:1.

The TMS320C2xxx processor internally uses a master clock that runs at twice the instruction rate of the processor. The clock can be externally generated or generated via an external crystal used in conjunction with the TMS320C2xxx on-chip oscillator. The processor can also use its on-chip phase-locked loop to generate a master clock from a lower-frequency clock operating at 1, 1/2, or 1/4 times the desired master clock rate. The TMS320C209 phase-locked loop only supports an external clock at half the desired master clock rate.

On the TMS320F240, if the phase-locked loop and on-chip oscillator are bypassed, the processor's master clock can be generated from a 1X or 2X external clock. In addition to the three ratios supported by the other TMS320C2xxx processors, the TMS320F240 phase-locked loop can also generate an input clock to master clock rate ratio of 2:3, 2:5, 1:3, 1:4, 2:9, 1:5, and 1:9, configurable by software.

The TMS320F241, TMS320C242, TMS320C243, and TMS320F243 use an on-chip phase-locked loop to multiply an external clock signal by four. The phase-locked loop on these devices cannot be bypassed.

Clock generation on the TMS320C24xx is similar to that of the TMS320C206, except that the phase-locked loop can be programmed to accept an external clock operating at eight different frequencies ranging from 1/4 the desired master clock to twice the desired master clock.

Hardware Looping

Like the TMS320C2x, the T320C2xLP core and TMS320C2xxx processors support single-instruction hardware loops. The RPT instruction repeats a single instruction from 1 to 256 times. Single-instruction repeats cannot be nested. Interrupts are disabled for the duration of the loop.

As explained in the section on memory above, the RPT instruction frees the program bus for parallel data moves and thus reduces the execution time of some instructions, most notably the multiply-accumulate instruction.

Multi-instruction hardware loops are not supported.

The lack of multi-instruction hardware looping is a serious drawback and limits performance in some applications.

Interrupts

The TMS320C2xLP core has six external maskable interrupt lines, an external non-maskable interrupt line, and an external reset input. No internal interrupt sources are provided, as the core does not include any peripherals. Each interrupt has its own interrupt vector in low memory and can be enabled and disabled individually. Interrupts are prioritized but not automatically nestable. Software interrupts are initiated using the TRAP or the INTR instructions. The INTR instruction can be used to invoke any of the processor's interrupts.

When the T320C2xLP receives an interrupt, the processor acknowledges it, providing that interrupts are enabled and the interrupt is not masked. All maskable interrupts are then disabled, and the program counter is pushed onto the stack. The processor then executes from the appropriate interrupt vector.

The TMS320C203 and TMS320F206 processors provide two external maskable interrupt lines, an external reset pin, a non-maskable interrupt line, and four peripheral interrupts.

The TMS320C209 processor has three external maskable interrupt lines, an external non-maskable interrupt line, and an external reset pin. It also has one on-chip peripheral interrupt from the timer.

The TMS320C24x, TMS320F24x, and TMS320C24xx processors include a unit that serves as arbiter for up to 31 peripheral interrupts. These interrupts are mapped to the core's six external maskable interrupt lines.

In other respects the TMS320C2xxx processors provide the same interrupt support as the T320C2xLP core.

Interrupt latency is 12 instruction cycles from the assertion of an external interrupt line to execution of the first word of the interrupt vector, assuming the processor is in an interruptible state.

Stack

The T320C2xLP and TMS320C2xxx hardware stack is eight levels deep and 16 bits wide. It is used to store the program counter for subroutine calls and interrupts. PUSH and POP instructions are also available.

Bootstrap Loading

Neither the T320C2xLP core nor the TMS320C209 processor provide special bootstrap loading circuitry. The TMS320C209 does not provide bootstrap loading features since designers will typically have their own code in the on-chip mask-programmable ROM. On reset, the processor begins executing at program address zero. It is the responsibility of the system or SoC designer to make sure that appropriate initialization code is available at that address on reset.

TMS320C2xxx devices other than the TMS320C209 allow bootstrap loading from a byte-wide ROM connected to the external bus. The EEPROM in the TMS320F206 and TMS320F24x is factory-programmed with a bootstrap loader that allows bootstrap loading over the serial port. The system designer can choose to replace the factory-programmed bootstrap loader with a program that is loaded over the serial port; this way, bootstrap loading only needs to be performed once. By including bootstrap loading capabilities in the replacement code, the system designer can provide the capability of reprogramming the flash EEPROM later via bootstrap loading. The TMS320C24xx has a separate 256-word boot ROM that is factory programmed with a generic bootloader. The boot ROM can be disabled under software control after reprogramming is completed. The presence of the boot ROM can improve reliability, since the generic bootloader is always available, even if reprogramming is interrupted.

The ability to bootstrap load and reprogram the on-chip flash EEPROM provides a convenient scheme for product upgrades after the product has been shipped.

Peripherals

The T320C2xLP core does not contain any on-core peripherals. Instead, the designer of an SoC containing the T320C2xLP core must add peripherals appropriate to the design at hand. Texas Instruments provides a number of T320C2xLP core-compatible peripheral macrocells for their TGC1000, TGC2000, and TGC4000 fabrication processes. Among others, these include a number of 8-bit A/D and D/A converters, a 16-bit timer, a wait-state generator, an interrupt synchronizer, a synchronous serial port, and an asynchronous serial port.

The T320C2xLP core and TMS320C2xxx processors provide one general-purpose input and one general-purpose output pin. The output line, called the XF (external flag) pin, can be set or cleared under software control by the SETC instruction. The state of the input line, called the BIO (branch I/O) pin, can be used as a condition for a conditional branch instruction.

TMS320C2xxx processors feature various peripherals, ranging from only a timer on the TMS320C209 to a variety of peripherals on the TMS320LF2407, including D/A and A/D converters, pulse-width modulation outputs, etc. The peripherals featured by the TMS320C24x, TMS320F24x, and TMS320C24xx are primarily intended for motor control applications. These peripherals, grouped into one or two "event managers," include timers, timer comparators, pulse width modulation generators, and quadrature-encoder pulse units.

- **Serial Ports**

  All TMS320C2xxx processors except the TMS320C209 provide an asynchronous serial port, and most provide a synchronous serial port as well. However, the serial ports are not identical on all family members.

  The TMS320C20x and TMS320F206 synchronous serial port supports 16-bit data at bit rates up to the master clock rate divided by two (e.g., 20 Mbits/second on a 40 MIPS TMS320C203 processor). Data can be transmitted continuously or in burst mode. The receive and transmit sections of the port are independent and can be controlled independently. The transmitter clock can be generated internally or be taken from an external clock. The receive clock is supplied by an external source. Frame synchronization signals can be generated internally or can be taken from an external source.

  The TMS320C24x, TMS320F24x, and TMS320C24xx processors, except the TMS320C242, TMS320LC2402, and TMS320LF2402, include a synchronous serial port designated as the "serial-peripheral interface." The TMS320C240 and TMS320F240 serial-peripheral interface supports 1- to 8-bit data at a maximum bit rate equal to the master clock divided by 8. The remaining TMS320C24x, TMS320F24x, and TMS320C24xx processors allow a data rate of twice that of the

TMS320C240 and TMS320F240 (e.g., 5 Mbits/second on a 20 MIPS TMS320F241).

The TMS320C20x and TMS320F206 asynchronous serial port supports 8-bit data with one start bit and one or two stop bits. The maximum transmission rate is the processor clock speed divided by 16 (e.g., 2.5 Mbits/second on a 40 MIPS TMS320C203 processor). The bit rate is determined either by writing a clock divisor value to a register or by using the asynchronous serial port's baud-detection logic to determine the bit rate of received data automatically.

The TMS320C24x, TMS320F24x, and TMS320C24xx include an asynchronous serial port designated as the "serial-communications interface." The TMS320C24x and TMS320F24x serial-communications interface supports 1- to 8-bit data with 1 start bit, an optional parity bit, and 1 or 2 stop bits. The serial-communications interface supports bit rates up to the master clock rate divided by 32 (e.g., 625 Kbits/second on a 20 MIPS TMS320C24x).

The receiver and the transmitter are configured independently, although they require that the same data format be used. The serial-communications interface on the TMS320C24xx processors is essentially the same, except that bit rates up to the master clock rate divided by 16 are supported (e.g., 2.5 Mbits/second on a 40 MIPS TMS320LC24xx processor).

- **Timers**

The TMS320C20x and TMS320F206 processors include a 16-bit timer which uses the master clock as its clock source. A 4-bit prescaler divides the master clock by up to 16 times, and a subsequent 4-bit "divide-down" scaler divides the prescaled signal further by up to 16 times. A 16-bit counter register uses the resulting signal as its input clock. The timer generates an output pulse on the TOUT pin and interrupts the processor when the counter reaches 0, at which point the counter is reloaded with a user-specified value.

The TMS320C24x, TMS320F24x, and TMS320C24xx include a watchdog timer that generates a reset signal when it reaches zero. Also, the event manager(s) on these processors each provide two prescalable 16-bit timers (three on the TMS320C240 and the TMS320F240) that can operate independently or can be synchronized with each other; i.e., the timers can be started and stopped simultaneously. These timers can generate interrupts when they underflow, overflow, or reach a pre-set value; the interrupts are configurable as one-time interrupts or periodic interrupts.

- **Event Manager**

The TMS320C24x, TMS320F24x, TMS320LC2402, and TMS320LF2402 include one event manager. The other TMS320C24xx processors provide two independent event managers. In addition to the timers mentioned above, each event manager also contains several compare units that can be used to generate up to eight (twelve on the TMS320C240 and TMS320F240) timer compare and pulse-width modulation waveform outputs. Each event manager also provides three (four on the TMS320C240 and TMS320F240) capture units that can each log transitions on a different input pin; the timers are used to count the transitions. A quadrature-encoder pulse unit can be used to receive a quadrature-encoder pulse (used to measure the rotation speed and position of a rotating motor) using two of the capture units.

- **Analog-to-Digital Converters**

  The TMS320C240 and TMS320F240 provide 16 analog inputs that are multiplexed to two 10-bit analog-to-digital converters with conversion times of 6.6 μs. The remaining TMS320C24x, TMS320F24x, and TMS320C24xx processors also provide 16 analog inputs (there are 8 input channels on the TMS320F241, TMS320F243, TMS320C242 and TMS320LC2402), but provide a single 10-bit analog-to-digital converter with a conversion time of 0.85 μs (0.5 μs for the TMS320C24xx).

- **Control Area Network Unit**

  The TMS320F241, TMS320F243, TMS320LC2406, TMS320LF2406, and TMS320LF2407 feature a control area network (CAN) unit. The CAN protocol is a serial communications protocol that passes messages via mailboxes. The physical interface consists of one receive and one transmit pin. Messages are received in mailboxes contained in devices that contain a CAN unit; these messages can subsequently be read by the processor. Similarly, outgoing messages are placed in local mailboxes, later to be transmitted on the CAN by the CAN unit. The CAN unit implements six mailboxes each with four 16-bit words of data. Two mailboxes are for receive, two can be configured for receive or transmit, and two are for transmit.

- **Bit I/O**

  The TMS320C203 and TMS320F206 provide four bit I/O pins and the TMS320C209 provides two. The TMS320C240 and TMS320F240 provide 28 bit I/O pins, the TMS320F241 and TMS320F243 provide 26 bit I/O pins, and the TMS320C242 provides 32 bit I/O pins. The TMS320LC2402 and TMS320LF2402 provide 20 bit I/O pins, while the other TMS320C24xx parts provide 40 bit I/O pins. Most of the bit I/O pins found on the TMS320C24x, TMS320F24x, and TMS320C24xx are multiplexed with other functions.

> *The TMS320C2xxx family members differ mostly in their configuration of on-chip peripherals and memory. Whereas the TMS320C209 has fewer peripherals than many DSP processors, the number of peripherals on the TMS320C24x, TMS320F24x, and TMS320C24xx is impressive. The peripherals on the TMS320C24x, TMS320F24x, and TMS320C24xx are mostly targeted at motor control applications.*

On-Chip Debugging Support

The T320C2xLP core and TMS320C2xxx processors have a JTAG-based interface to on-core/on-chip debugging circuitry for in-circuit emulation. Through the debugging interface an external device can read and write processor memory and registers, set and clear hardware and software breakpoints, and single-step the processor.

Power Consumption and Management

The T320C2xLP corè operates at nominal voltages of both 3.3 and 5.0 volts. According to Texas Instruments, typical core power consumption is 270 mW at 5.0 volts when executing at 28 MIPS, and 66 mW at 3.3 volts when executing at 20 MIPS. These figures correspond to approximately 385 mW at 5.0 volts and 40 MIPS, and 160 mW at 3.3 volts and 40 MIPS.

The TMS320C2xxx chips operate at 5.0 and 3.3 volts. Texas Instruments states that typical power consumption is 380 mW when executing at 40 MIPS and 5.0 volts, and 72 mW when executing at 20 MIPS and 3.3 volts, corresponding to approximately 144 mW at 3.3 volts and 40 MIPS. The power consumption figures for the T320C2xLP core and the TMS320C2xxx assume that the processor is executing a FIR filter from on-core memory.

The T320C2xLP core and all TMS320C2xxx processors except the TMS320F240 provide a power-down mode via the IDLE instruction, which turns off the clock to the core. For example, the typical power consumption of the TMS320C203 processor in IDLE mode is 50 mW at 5.0 volts, and 15 mW at 3.3 volts. Any unmasked external interrupt wakes the processor from IDLE mode.

The TMS320C240 and TMS320F240 provide four modes of low-power operation: idle 1, idle 2, standby, and halt. The TMS320F241 and TMS320F243 eliminate the standby mode. The four modes stop the clocks to various parts of the processor. The processor can be configured by software to enter one of the four modes when it executes the IDLE instruction. Interrupts or reset bring the processor back to normal operation. According to Texas Instruments, the TMS320F240 power consumption in each of the four low-power modes is 250 mW for idle 1, 35 mW for idle 2, 5 mW for standby, and less than 2 mW for halt. In Idle 1 mode, the main processor clock is stopped, halting the core, memory interface and event manager peripherals. In Idle 2 mode, the clock to the remaining peripherals is disabled as well. In standby mode the phase-locked loop is turned off, but the oscillator and watchdog timer are still active. In halt mode the oscillator and all clocks are disabled.

The TMS320C24xx provides the same three low-power modes as the TMS320C24x. In addition, the peripherals can be individually powered down independent of the other low power modes. The clocks for each peripheral are disabled after a reset to allow low startup power, and can be turned on or off during execution by setting the clock enable bits in a control register.

Benchmark Performance

The TMS320C2xxx has not been benchmarked with the current version of the BDTI Benchmarks so this report does not include benchmark results for the TMS320C2xxx. Based on its performance on previous versions of the BDTI Benchmarks, we provide a general, qualitative description of the TMS320C2xxx's performance relative to that of the other processors in this report. In particular, we will compare the TMS320C2xxx with the DSP568xx, as both are conventional fixed-point architectures and have similar target applications.

Execution Performance

- **Instruction cycle counts:** Based on previous BDTI Benchmark analyses, the TMS320C2xxx typically has very high cycle counts in comparison to other fixed-point processors, for several reasons. First, the processor lacks support for multi-instruction hardware loops. While the TMS320C2xxx does provide a decrement-and-branch-if-not-zero instruction, this adds four instruction cycles to each loop iteration of block-based benchmarks—a significant increase. Second, many important DSP instructions (such as multiplies) execute in a single instruction cycle only if they are executed within a single-instruction hardware loop. This increases cycle counts on algorithms that have multi-instruction kernels, such as LMS Adaptive FIR filters, IIR filters, and FFTs. Third, the TMS320C2xx has only a single accumulator that must be used for all arithmetic and logic operations. This increases cycle counts on some algorithms, such as complex-valued block filters and FFTs. Fourth, an inability to move values from register to register without storing them temporarily in memory consumes additional cycles. Fifth, the processor provides only limited support for operand-unrelated parallel moves, which means that additional instructions are required to load or store data in many benchmarks.

  In comparison to the DSP568xx, cycle counts for the TMS320C2xxx can be expected to be nearly 1.5 times higher on average.
- **Execution times:** The fastest member of the TMS320C2xx family has an instruction cycle rate of 40 MIPS. This is quite slow in comparison with most of the DSP processors presented in this report; however, it is similar to that of the fastest member of the DSP568xx family, the 40 MIPS DSP56F801. Because the TMS320C2xx has significantly higher cycle counts than the DSP568xx and a comparable instruction cycle rate, its execution times will be significantly slower than those of the DSP568xx.
- **Cost-execution time:** The 40 MIPS TMS320LC2404 is priced at $6.00 (quantity 10K). In comparison, the 40 MIPS DSP56F801 is priced at $8.15. The TMS320LC2404 is likely to be roughly 40% slower than the DSP56F801 overall, and costs only 25% less. Hence, the TMS320LC2404 is likely to have worse cost-execution times than the DSP56F801.

- **Energy consumption:** According to Texas Instruments, the TMS320C2xxx consumes 144 mW at 40 MIPS and 3.3 volts. In comparison, the DSP56824 consumes 54 mW at 35 MIPS and 3.3 volts. Because the DSP56824 is faster than the TMS320C2xxx and has much lower power consumption, the DSP56824 will have much better energy consumption results than the TMS320C2xxx; TMS320C2xxx energy consumption is likely to be about three times higher than that of the DSP56824.

Memory Usage

The focus of our memory usage analysis is on Control benchmark memory usage. Unlike other benchmarks, the Control benchmark is optimized for minimum memory usage. This benchmark is designed to indicate a processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks, reflecting the processor's program memory usage in general DSP code. Finally we discuss constant and non-constant data memory usage.

Because the TMS320C2xxx has not been benchmarked with the current suite of BDTI Benchmarks, we base our analysis on the processor's results on earlier versions of the BDTI Benchmarks.

- **Control code memory usage:** Based on previous results for BDTI's control-oriented benchmark, the TMS320C2xxx achieves fair code density on control code. Although it might be expected that TMS320C2xxx memory usage would be relatively low because of its short, 16-bit instructions, limitations of the TMS320C2xxx instruction set increase its memory usage on control code. For example, most arithmetic operations must use the accumulator, resulting in the need for transfer instructions to temporarily save and restore the accumulator contents. In addition, the TMS320C2xxx instruction set lacks a PC-relative branch instruction. In comparison to the DSP568xx, which also uses 16-bit instructions, the TMS320C2xxx can be expected to have somewhat higher memory usage on control code (though not dramatically higher).
- **Program memory usage:** Based on previous benchmark results, the TMS320C2xxx requires more program memory than the DSP568xx; on previous versions of the BDTI Benchmarks, the TMS320C2xxx's total normalized program memory result was roughly 30% higher than that of the DSP568xx. In order to achieve low instruction cycle counts on the TMS320C2xxx, it is often necessary to perform loop unrolling to a larger extent than on other DSPs, thus increasing program size. In addition, because the TMS320C2xxx has only limited support for operand-unrelated parallel moves, several instructions are often required where other DSPs can perform the same operations in a single instruction (albeit often at the cost of larger instruction word widths).

- **Data memory usage:** Based on previous benchmark results, the TMS320C2xxx constant and non-constant data memory usage is about as expected for a 16-bit processor.

Cost

Price and packaging options for the TMS320C2xxx are shown in Table 7.11-4. Pricing of an SoC containing the T320C2xLP depends on many variables, including size, packaging, and production volume. Contact Texas Instruments for more information.

The lower priced members of the TMS320C2xxx family are among the least expensive DSPs available.

Fabrication Details

The TMS320C203, TMS320LC203, TMS320C206, TMS320LC206, and TMS320C209 are fabricated in 0.35 µm technology. The TMS320F206, TMS320C240, and TMS320F240 processors are fabricated in 0.72 µm CMOS technology. The TMS320F241, TMS320C242 and TMS320F243 are fabricated in 0.65 µm CMOS. All of the TMS320C24xx processors are fabricated in 0.25 µm CMOS. As mentioned above, the T320C2xLP is available as a core for use in SoCs fabricated by Texas Instruments.

Development Tools

Texas Instruments provides a comprehensive set of code generation, debugging, and system integration tools all of which are well described in their publication, *The TMS320 DSP Development Support Reference Guide.* Texas Instruments provides a macro assembler, linker, instruction-set simulator, and C compiler for the T320C2xLP core and TMS320C2xxx processors. The tools run on both IBM PC-compatible computers under DOS and Windows and on Sun SPARC workstations under SunOS. With the exception of the simulator, which is specific to the TMS320C2xxx, the same tools support the TMS320C1x, TMS320C2x, and TMS320C5x families as well.

Texas Instruments offers the XDS510 scan-based emulator, which can be used with TMS320C2xxx processors and SoCs built around the T320C2xLP core. The XDS510 emulator is hosted on an IBM PC with a plug-in card. The emulator connects to the TMS320C2xxx in the target system via a five-wire JTAG interface. Texas Instruments also provides development boards (called "evaluation modules" or EVMs).

The simulator, emulator, and EVM share a common character-based, windowed user interface. On the IBM PC, the simulator runs in Microsoft Windows 3.1 and 95, but is not a true Windows application. The simulator and emulator support both assembly language and C source-level debugging.

The Texas Instruments-supplied software development tools for the TMS320C2xxx are quite capable, but not as easy to use as one

Part	Speed (MIPS)	Voltage (V)	Package	Price (Qty. 10,000)
TMS320C203	40.0	5.0	100 TQFP	$5.45
	28.5	5.0	100 TQFP	$5.20
	20.0	5.0	100 TQFP	$5.20
TMS320LC203	20.0	3.3	100 TQFP	$5.75
TMS320C206	40.0	5.0	100 TQFP	$8.30
TMS320LC206	40.0	3.3	100 TQFP	$9.15
TMS320F206	40.0	5.0	100 TQFP	$13.50
TMS320C209	40.0	5.0	80 TQFP	$9.60
	28.5	5.0	80 TQFP	$10.15
TMS320C240	20.0	5.0	132 PQFP	$10.20
TMS320F240	20.0	5.0	132 PQFP	$14.74
TMS320F241	20.0	5.0	64 PQFP/ 68 PLCC	$11.36/ $12.11
TMS320C242	20.0	5.0	64 PQFP/ 68 PLCC	$3.96/ $4.54
TMS320F243	20.0	5.0	144 LQFP	$12.85
TMS320LC2402	30.0/ 40.0	3.3	100 LQFP	$2.95/ $3.25
TMS320LF2402	30.0/ 40.0	3.3	64 PQFP	$8.75/ $9.63
TMS320LC2404	30.0/ 40.0	3.3	100 LQFP	$5.45/ $6.00
TMS320LC2406	30.0/ 40.0	3.3	100 LQFP	$5.95/ $6.55
TMS320LF2406	30.0/ 40.0	3.3	100 LQFP	$9.95/ $10.95
TMS320LF2407	30.0/ 40.0	3.3	144 LQFP	$10.45/ $11.50

TABLE 7.11-4. TMS320C2xxx price and package summary as of June 2000.

> *might hope. For example, the simulator does not provide the ability to load values into memory from an ASCII file, and there is no on-line help. The simulator does provide good debugging and profiling features.*

Texas Instruments also offers "Code Composer," a suite of software development tools. Code Composer consists of an IDE shell/project manager that uses Texas Instruments' tools for assembling, compiling, linking, and simulating. Code Composer also includes a debugger which supports graphical data displays (e.g., an FFT "waterfall"), and can be used for debugging individual devices or multiple devices in parallel. Source-level debugging capabilities include breakpoints, watches, and function-profiling, all of which are fully Windows-based. Code Composer also supports a scripting language which allows automation of simple test tasks. Code Composer is available for Windows 95, 98, and NT.

Applications Support

TMS320C2xxx documentation consists of the *TMS320C2xxx User's Guide*, the *TMS320F/C24X DSP Controllers CPU and Instruction Set Reference Guide,* the *'F243/F241/C242 DSP Controllers System and Peripherals Reference Guide,* and the *TMS320LF/LC240x DSP Controllers System and Peripherals User's Guide*. These documents describe the TMS320C2xxx architecture, instruction set, and peripherals. Qualified customers can obtain proprietary documentation on the T320C2xLP core from Texas Instruments. Separate data sheets for the TMS320C2xxx processors and the T320C2xLP core are available.

Additionally, the articles contained in the TMS320C2x application handbooks are useful for TMS320C2xxx users.

> *In general, TMS320C2xxx family documentation is well organized and clear.*

Applications support for all TMS320 family processors is provided by staff who are available via telephone hotline, fax, and e-mail.

Texas Instruments has extensive World Wide Web pages (at *www.ti.com*) providing numerous applications notes, data sheets, and other information regarding the TMS320C2xxx family processors.

Advantages

- Eight address registers
- Flash EEPROM versions available (TMS320F206, TMS320F24x, TMS320LF24xx)
- Two or more serial ports (except on TMS320C209, TMS320C242, TMS320LC2402, TMS320LF2402)

- Large number of peripherals on TMS320C24x, TMS320F24x, and TMS320C24xx, including digital-to-analog converter, three timers, up to 40 bit I/O pins, one or two "event manager," and Control Area Network unit
- Good third-party development tools
- Good applications support in the form of software function and application libraries
- Low cost (e.g., $3.25 for the 40 MIPS TMS320LC2402, quantity 10,000)
- Available as a DSP processor and as a foundry-captive core for application specific designs

Disadvantages

- Non-orthogonal instruction set
- Only one accumulator, and few additional arithmetic registers
- Limited parallel move support
- No guard bits
- Multiply and multiply-accumulate are multi-cycle instructions unless executed from within a hardware loop
- Multiplication operands restricted
- No circular addressing
- Only one modifier register (AR0)
- No multi-instruction hardware loop
- Small on-chip data memory (some family members)
- Off-chip data accesses require multiple cycles unless executed within a single-instruction hardware loop
- Few peripherals (only a timer) on TMS320C209
- Slow execution times, based on previous BDTI Benchmark analysis
- Poor energy efficiency, based on previous BDTI Benchmark analysis

7.12 Texas Instruments TMS320C3x Family

BDTImark2000 Score: Not Available

Introduction

The TMS320C3x is a family of 32-bit floating-point conventional DSP processors from Texas Instruments. The TMS320C3x targets digital audio, data communications, and industrial automation and control applications. The fastest member of the TMS320C3x family (the TMS320VC33) executes at 75 MIPS with a core voltage of 1.8 volts. BDTI has not implemented its current suite of benchmarks on the TMS320C3x; hence, no BDTImark2000 score is currently available for this processor. Previous BDTI Benchmark results for this processor are available directly from BDTI.

The TMS320C3x family is Texas Instruments' first generation of floating-point DSPs. The first family member, the TMS320C30, was introduced in 1988. The TMS320C31, introduced in 1990, is a lower-cost version of the TMS320C30 and lacks the on-chip ROM, the second external expansion bus, and the second serial port of the TMS320C30. The TMS320C32, introduced in 1995, is a reduced-cost version of the TMS320C31 that includes several enhancements: an external memory interface designed for connection to 8- and 16-bit external memories, an extra channel in the DMA controller, and other features. The TMS320VC33, introduced in 1999, is a higher-performance, reduced-cost, low-voltage version of the TMS320C31. Additional features include 34 Kwords of on-chip RAM, an IEEE-1149.1 (JTAG) emulation port, internally decoded page strobe signals, and a phase-locked loop for clock generation. TMS320C3x processors are summarized in Table 7.12-1.

The TMS320C3x has long been stagnant, with Texas Instruments focusing on its higher-performance TMS320C67xx floating-point family. The TMS320VC33 is the family's first new member in sev-

Part	Maximum Speed (MIPS)	On-Chip Memory		Notes
		RAM	ROM	
TMS320C30	25	2Kx32	4Kx32	Two external buses, two serial ports, one DMA channel
TMS320C31	40	2Kx32	—	One external bus, one serial port, one DMA channel
TMS320C32	30	512x32	—	Enhanced external bus interface, one serial port, two DMA channels
TMS320VC33	75	34Kx32	—	One external bus, one serial port, one DMA channel, JTAG interface, PLL

TABLE 7.12-1. TMS320C3x processor summary.

eral years, and offers nearly twice the performance of the TMS320C31 for about half the price.

The TMS320C31 and TMS320C32 are two of the lowest-cost floating-point DSPs on the market. With a unit price of less than $10 for the 20 MIPS TMS320C32 (any quantity), the processor is competitive in terms of cost with many fixed-point processors.

Architecture

Figure 7.12-1 illustrates the TMS320C3x family architecture as typified by the TMS320C30. Noteworthy features include the multiple address and data buses, the DMA controller and its dedicated buses, and the two external memory interfaces.

Data Path

The TMS320C3x data path consists of a multiplier, a barrel shifter and ALU, and a register file containing 28 registers, eight of which are 40-bit "extended-precision" registers.

The ALU operates on 32-bit signed integer and 40-bit floating-point data, providing arithmetic and logical operations as well as integer/floating-point conversions. There is no explicit support for rounding when converting from floating-point to integer. Instead, floating-point values are always floored to the nearest integer when converted. When 40-bit floating-point data is stored to memory, the lower eight bits are discarded. However, a full precision store or load can be achieved with four instructions: two loads from the source register and two stores. Note that the stored data would occupy two words in memory.

The barrel shifter is coupled to the ALU and can perform shifts of up to 32 bits left or right. The shifter supports rotate through carry. All multiplies, ALU operations, and shifts are performed in a single cycle.

The multiplier can multiply 32-bit floating-point input data, producing a 40-bit floating-point result, or 24-bit signed integer input data, producing a result of which the least-significant 32 bits are retained.

Floating-point multiplier and ALU inputs must come from the eight extended-precision registers; integer inputs can come from any register. Multiplier and ALU outputs can be stored in subset of the register file. Guard bits are not provided when operating on integer values; only the lower 32 bits of a register are used for integer operations. Integer saturation can be enabled via a mode control bit. Carry and borrow are supported for integer addition and subtraction respectively.

Carry, overflow, underflow, zero, or negative status bits are set according to the outcome of an integer, logical, or floating-point operation.

IEEE floating-point formats are not supported. The TMS320C3x uses an internal 40-bit extended-precision floating-point representation and a 32-bit single-precision floating-point representation for storing floating-point data in memory. Both of these formats use 8-bit exponents and the extended-precision format has eight extra mantissa bits. The family also supports a 16-bit "short floating-point" data type for immediate floating-point operands which has a 12-bit mantissa and a 4-bit exponent.

All arithmetic and logical operations are performed with single-cycle throughput. Some operations may be performed in parallel; for example, a floating-point multiply and a floating-point add can be performed in the same instruction cycle, and may be used for single-cycle multiply-accumulate or multiply-add operations.

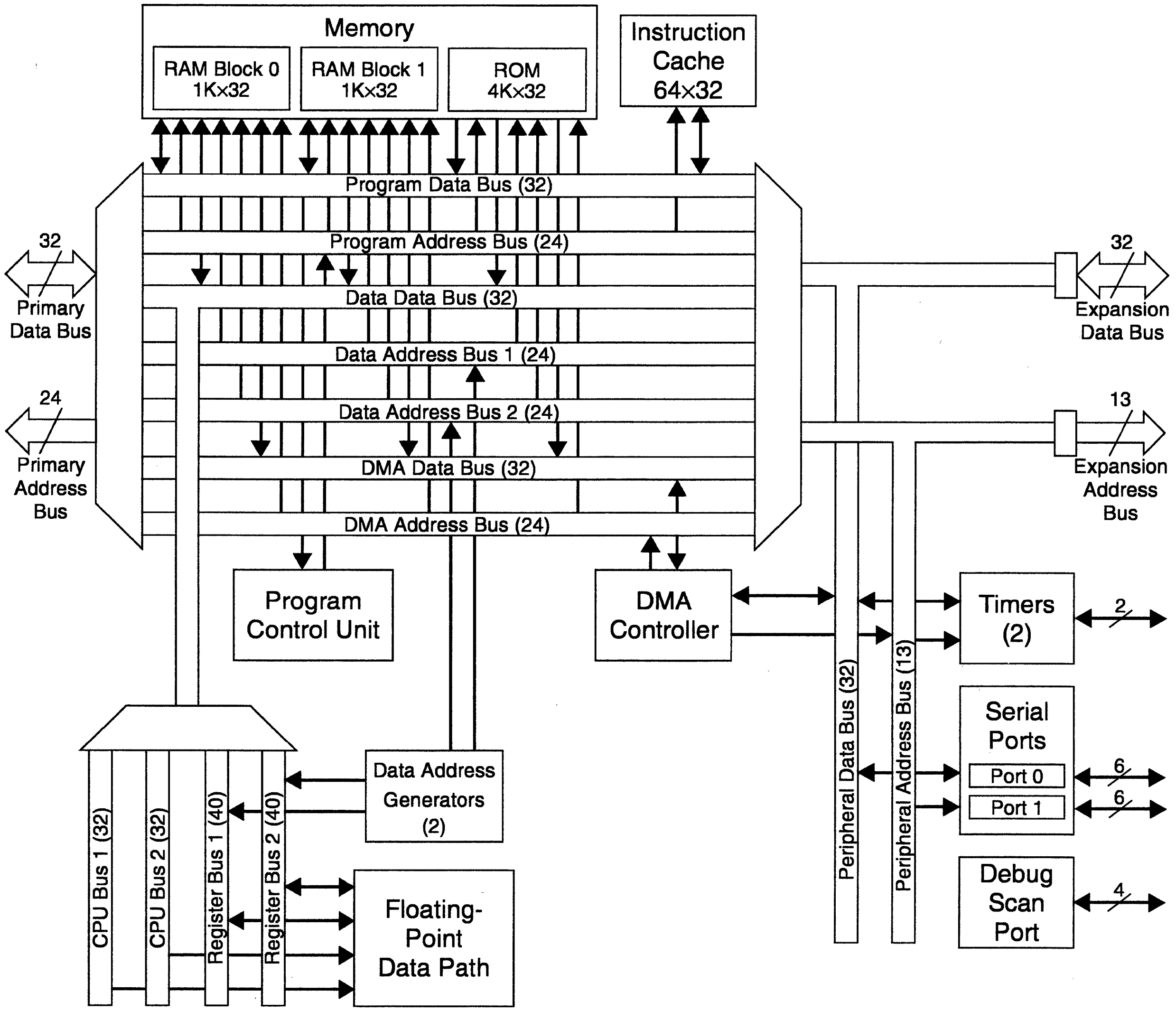


FIGURE 7.12-1. TMS320C30 processor architecture.

Memory System

The TMS320C3x family provides a unified address space that allows the programmer to use any portion of memory for program or data as appropriate. The address space is 32-bit word addressable; however, on the TMS320C32, external memory is byte, 16-bit word, and 32-bit word addressable.

On-chip buses include a 24-bit program memory address bus, a 32-bit program memory data bus, two 24-bit data memory address buses, and a 32-bit data memory data bus. The data memory data bus allows two sequential accesses per instruction cycle. These buses allow the processor to address up to 16 Mwords of program and 16 Mwords of data memory. Additionally, a separate on-chip 24-bit DMA address bus and a 32-bit DMA data bus attached to an on-chip DMA controller allow I/O to and from memory without conflicting with normal bus activity.

The TMS320C30 and TMS320C31 provide two 1Kx32 blocks of on-chip RAM, the TMS320C32 provides two 256x32 blocks of on-chip RAM, and the TMS320VC33 provides two 16Kx32 and two 1Kx32 blocks of on-chip RAM. The TMS320C30 also includes one 4Kx32 block of on-chip ROM. All of these memories support two accesses during every instruction cycle. Combined with the processor's bus configuration, the memory system permits an instruction word fetch and two data accesses per instruction cycle, assuming that no more than two accesses (instructions and data) are attempted to a single memory block. Additionally, the DMA controller can be used in parallel with these accesses to achieve an additional memory access per instruction cycle. Thus, the maximum sustainable on-chip data memory bandwidth is 300 million 32-bit words/second on a 75 MIPS TMS320VC33.

The dedicated DMA address and data buses are an advantage for I/O-intensive applications.

The TMS320C3x architecture contains a 64-word instruction cache to speed accesses from slower external memories. The instruction cache is divided into two 32-word segments, each of which is associated with a contiguous 32-word region of memory aligned on a 32-word boundary. When the processor attempts to fetch an instruction word from external memory, the cache circuitry checks to see if the word is in cache. If so, the word is used from cache, and the external fetch does not occur. If the word is not in cache, the instruction is fetched externally and the cache is updated in one of two ways. If the fetch was from a region of memory associated with one of the two cache segments, then the word is stored in the appropriate location in the cache. Otherwise, the contents of the least recently used 32-word cache segment are marked as invalid, and that segment becomes associated with the 32-word region of memory containing the instruction address. The word is then stored in the cache. The cache can be disabled or locked (i.e., no cache updates allowed) under software control.

The cache is not necessary for full-speed operation (indeed, it is not used when the processor executes from internal memory), but can increase performance when instructions are stored in slower external memories.

External Memory Interface

Each member of the TMS320C3x family has a somewhat different external memory interface, as described in the following paragraphs.

The TMS320C30 provides two external bus interfaces: the primary bus interface and the expansion bus interface. The primary bus interface consists of a set of two external buses: one 24-bit address bus and one 32-bit data bus. The expansion bus interface consists of a 13-bit address bus (with two strobe lines) and a 32-bit data bus. The primary and expansion buses are independent, and simultaneous accesses over both buses may be made. Both buses can be used for program, data, or I/O.

> *The two sets of external data buses on the TMS320C30 are an advantage for larger applications requiring multiple simultaneous access to external memories. However, the expansion port supports only 16 Kwords of external memory (two sets of 8 Kwords with one strobe line per set) which limits its utility.*

The TMS320C31 eliminates the TMS320C30 expansion bus. That is, it provides only the primary bus interface.

Like the TMS320C31, the TMS320C32 has only the primary bus interface. On the TMS320C32 the primary bus is enhanced to handle 8-, 16-, and 32-bit data types and 8-, 16-, and 32-bit physical memory. For each of three regions of memory, the programmer can specify the width of the physical memory available in that region and the width of the data stored in that memory. The enhanced memory interface then performs byte or word packing or unpacking as necessary. For an image processing application, for example, 8-bit data might be used with 8-bit external memory. In this case no packing or unpacking needs to be done; the enhanced memory interface essentially provides a byte-addressable interface. In another application, 8-bit data might be stored in a 32-bit memory block made up of four individual 8-bit memories. In this case the interface asserts the appropriate byte select strobe to access the desired 8-bit value. In a third situation a 32-bit value might be stored in 8-bit-wide external memory. In this case the interface performs four 8-bit reads and packs the values together to form a 32-bit word.

> *The TMS320C32 enhanced external memory interface is very flexible and reduces system cost by supporting economical external memory configurations.*

The TMS320VC33's external interface is nearly identical to that of the TMS320C31 but provides four page-select lines that are decoded from the upper two address bits.

The pre-decoded page-select lines of the TMS320C32 reduce system cost by eliminating the need for additional off-chip logic.

The TMS320C30, TMS320C31, and TMS320VC33 provide a $\overline{\mathrm{STRB}}$ signal that is asserted during a primary bus access. The TMS320C30 provides two additional strobe signals, $\overline{\mathrm{MSTRB}}$ and $\overline{\mathrm{IOSTRB}}$, that can be asserted during an expansion bus access. Each of these strobe signals is associated with a fixed region of the processor's address space.

The TMS320C32 has three sets of strobe signals; each set is associated with a fixed region of the processor's address space. The first two sets each contain four strobe lines. These lines serve as byte- or word-select signals when accessing data types or physical memory less than 32 bits wide. The third set contains only a single $\overline{\mathrm{IOSTRB}}$ line.

On all TMS320C3x processors, external bus writes take two instruction cycles. However, the external bus logic can buffer a single word so the CPU can execute the next instruction without delay provided that instruction is not another external bus write. Reads take one instruction cycle unless they follow a write, in which case they also take two cycles. This yields a maximum sustainable memory bandwidth for each external bus of 37.5 million 32-bit words/second for writes and 75 million 32-bit words/second for reads on a 75 MIPS TMS320VC33.

The two-cycle external writes reduce performance in some applications.

The TMS320C3x can use a mixture of programmed and externally requested wait states: an external bus interface can be programmed to use between zero and seven master clock cycle wait states, or to wait until an external $\overline{\mathrm{RDY}}$ pin is asserted, or to wait until either the earlier or later of these two events occurs.

All TMS320C3x family members have programmable memory page boundary detection circuitry that can be used to add a wait state when the processor crosses a memory page boundary. Memory page size can be configured in powers of two, from 256 words to 16 Mwords.

The primary bus on all TMS320C3x family members has a $\overline{\mathrm{HOLD}}$ input that allows an external device to obtain exclusive access to the processor's primary bus. When an external device asserts $\overline{\mathrm{HOLD}}$, the processor completes any external bus cycle in progress, places its external buses in a high-impedance state, and then asserts the $\overline{\mathrm{HOLDA}}$ (hold acknowledge) pin. The external device can then read or write the processor's external memory (e.g., for block-based I/O) without risk of bus conflicts. While $\overline{\mathrm{HOLD}}$ is asserted, the TMS320C3x continues to execute normally as long as it does not attempt an external bus access. The processor is suspended upon attempting such access. When the external device is finished with the external bus, it deasserts $\overline{\mathrm{HOLD}}$, allowing normal execution to resume.

To support multiprocessor applications, TMS320C3x processors include several "interlocked" instructions that can be used for atomic read-modify-write sequences over the external bus. For example, execution of the "load interlocked" instruction asserts the

XF0 pin and then attempts an external bus access. The access is delayed until an external bus arbiter asserts XF1 indicating that the device has exclusive access to the bus. The read then completes; XF0 is left asserted to maintain exclusivity. The processor then modifies the value read and executes a "store interlocked" instruction that writes to the external bus and deasserts XF0. The external bus arbiter then deasserts XF1 and allows other devices to access the bus.

Address Generation Units

The two address generation units (called "auxiliary register arithmetic units") are used for generating up to two addresses for data accesses in each instruction cycle. Memory is only word-addressable; accessing single bytes within a word is not possible. Supported addressing modes include immediate direct, register direct, paged memory direct, indexed, and register indirect. For paged memory-direct addressing, addresses are formed by combining eight bits stored in a data page register with 16 bits stored in the instruction word; pages are thus 64 Kwords long. Immediate data is also supported.

Register-indirect addresses come from one of the eight 32-bit address registers, AR0-AR7, although only the lower 24 bits of these registers are used for address generation. Two modifier registers, IR0 and IR1, provide 24-bit values for pre- and post-increment and decrement as well as indexed addressing. Additionally, pre- and post-increment and decrement and indexed addressing modes can use 8-bit immediate data for offsets in most cases. Three-operand instructions restrict this immediate offset to 1 or 0.

> *Although the processor has only two modifier registers (IR0 and IR1) for pre- and post-modification, the support for 8-bit immediate offset and pre- and post-modification means that the limited number of modifier registers should not limit performance.*
>
> *Indexed addressing is useful for compiler-generated code.*
>
> *Because byte addressing is not supported, applications that require data shorter than 32 bits suffer a performance loss.*

Bit-reversed addressing is available, but only in one addressing mode: register-indirect with post-increment by the contents of IR0.

The TMS320C3x supports circular addressing through a single "blocksize" register. Circular addressing can be used with post-increment or post-decrement register-indirect addressing by placing a "%" suffix (similar to the C programming language's modulo operator) after the register name in the assembly code. Since there is only one block size register, only one size of circular buffer is supported at a time. A circular buffer can span the entire addressable range, but must start at an address that has at least as many least-significant bits equal to zero as the number of bits required to represent the modulo buffer size (e.g., to implement a 200-entry buffer, the base address must be aligned at a 256-word boundary).

The lack of support for simultaneous circular buffers of more than one size is a disadvantage.

Pipeline

The TMS320C3x uses an interlocked, four-stage pipeline divided into instruction fetch, instruction decode, operand read, and instruction execute stages. The pipeline's interlock mechanisms guarantee that unexpected results are not produced due to pipeline effects. However, the pipeline is visible to the programmer because instructions that pose a pipeline hazard are delayed. This means that although the programmer does not have to worry about the pipeline affecting a program's functionality, the programmer must worry about extra execution time introduced by interlocks.

The main sources of pipeline conflicts on the TMS320C3x are:

- Instructions that read from an address register (i.e., AR0-AR7) or that use an address register for address generation following an instruction that directly loads *any* address register are delayed until the load finishes executing. This can result in up to two NOPs being inserted into the pipeline. (Note that pre- or post-modification of an address register during register-indirect addressing does not count as a direct load of the address register.)
- Instructions that directly read the value of an address register (e.g., in moving the value stored in an address register to another register, as opposed to using the address register for address generation) delay any subsequent instruction using *any* address register until the read has completed, which occurs in the operand read stage. This can result in one NOP being inserted into the pipeline.
- Since the CPU is only allowed two data accesses per instruction cycle, sequences of instructions that attempt three data accesses in a single instruction cycle result in a delay. The two situations where this arises are when an instruction that stores a value to memory is followed by an instruction that reads two values from memory, and when an instruction that stores two values to memory is followed by an instruction that performs at least one read from memory. One NOP is inserted in the pipeline in these situations.
- Non-delayed branches flush the pipeline, resulting in three NOP instructions following the branch. Delayed branches are available.

Writing efficient software for the TMS320C3x requires a thorough understanding of the device's pipeline. In particular, instructions that would seem to involve only two data accesses (and thus should execute in a single instruction cycle) may in fact take multiple instruction cycles due to conflicts with adjacent instructions.

Instruction Set

TMS320C3x registers and instructions are summarized in Tables 7.12-2 and 7.12-3. All TMS320C3x instructions are 32 bits wide.

Assembly Language Format

The TMS320C3x assembly language uses the traditional opcode-operand style. Almost all instructions on TMS320C3x processors use one of two basic instruction formats. The first format uses a single register as a source and destination operand with a second source operand that may be a 16-bit immediate value, any register, a directly addressed memory value, or an indirectly addressed memory value. The second instruction format uses three operands: one is a register, and two others are either registers or indirectly addressed memory locations. Fixed-point instructions may use any register as a source or destination operand, but floating-point operations may only use the eight 40-bit registers that support the floating-point data format. Typical two- and three-operand instructions are:

```
ADDI        R0,R1
MPYF3       *AR0++(1),*AR1++(1),R0
```

The first instruction performs an integer add of R0 to R1, leaving the result in R1. The second instruction multiplies the floating-point values pointed to by AR0 and AR1 and stores the result in register R0. Registers AR0 and AR1 are post-incremented by one (indicated by "++(1)").

The TMS320C3x allows parallel execution of certain operations. These operations can be a multiply in parallel with an add or subtract operation; a multiply or ALU operation in parallel with a data move; or two data moves in parallel. A typical multiply-accumulate instruction is:

```
       MPYF3       *AR0++(1),*AR1++(1)%,R0
||     ADDF3       R0,R2,R2
```

Registers	Width	Purpose
R0-R7	40 bits	Extended-precision registers for integer/floating-point
AR0-AR7	32 bits	Address registers
IR0, IR1	32 bits	Modifier registers
BK	32 bits	Block size register (for circular addressing)
SP	32 bits	Stack pointer

TABLE 7.12-2. TMS320C3x registers.

The "||" symbol indicates that the two instructions are to be executed in parallel. The MPYF3 instruction executes a floating-point multiply of the values stored in memory at the addresses specified by address registers AR0 and AR1 and places the product in register R0. AR0 and AR1 are post-incremented by one; the "%" character indicates that AR1 is updated using modulo addressing. The ADDF3 instruction adds the result of the previous multiply instruction (in R0) to R2, which serves as an accumulator in this example. The "3" suffix on the instructions indicates that they take three operands.

Parallel Move Support

Not including DMA, the TMS320C3x allows up to two data memory accesses (two loads, two stores, or a load and/or a store) per instruction cycle (assuming no pipeline conflicts occur, as described above). The processor provides parallel move support through two mechanisms. First, most instructions support two memory accesses via regis-

Class	Instructions
Arithmetic	Absolute value (both), add with carry (integer), add (both), negate (both), subtract with borrow (integer), subtract (both), reverse subtract (both)
Multiplication	Multiply (both), multiply-accumulate (both), multiply-subtract (both)
Logic	*And*, *not-and*, *not*, *or*, *exclusive-or*
Shifting	Left or right arithmetic shift 0-32 bits, left or right logical shift 0-32 bits
Rotation	Rotate left/right one bit through carry or not through carry
Conditional Execution	Conditional load, conditional store, conditional subtract (integer)
Comparison	Compare (both)
Looping	Single- and multi-instruction repeat
Branching	Conditional and unconditional branch (delayed and non-delayed, with or without decrement)
Subroutine Call	Conditional and unconditional call, conditional and unconditional return
Bit Manipulation	Test bit field
Special Function	Convert floating-point to integer, convert integer to floating-point, wait for interrupt, normalize, push, pop, "interlocked" operations for read-modify-write, software interrupt, trap

TABLE 7.12-3. TMS320C3x instructions. "Both" indicates that the instruction supports both fixed- and floating-point data types.

ter-indirect addressing. This gives the processor operand-related parallel moves. As an example, the instruction:

```
                    ASH3        *AR1,*AR0,R0
```

executes an arithmetic shift of the value in memory pointed to by AR0 and places the result in R0. The number of bits to shift by is given by the value in the memory location specified by AR1.

Second, operand-unrelated parallel moves can be performed using the parallel instruction execution format shown in the multiply-accumulate example above. In addition to allowing the combination of certain ALU and multiply operations in a single cycle, the parallel execution format can also be used to combine an ALU or multiply operation with an unrelated load or store operation. For example:

```
                    ASH3        R0,*AR0,R1
             ||     STI         R2,*AR1
```

This executes an arithmetic shift of the value pointed to by AR0 and places the result in register R1. The number of bits to shift by is specified by R0. In parallel, the processor executes an unrelated integer store instruction, writing the value in register R2 to the memory address specified in AR1.

Orthogonality

As described in the section on assembly language format above, the TMS320C3x instruction set is quite orthogonal. Most instructions fall into one of two categories, having either two or three operands chosen from a uniform set of options. This level of orthogonality is in large part made possible by the 32-bit instruction word.

Certain commonly used parallel instructions impose restrictions on allowable registers. For example, the MPY and ADD instructions are commonly executed in parallel to form a multiply-accumulate instruction. The destination registers for the multiply result can only be R0 or R1, and for the accumulation, R2 or R3.

> *The limitations on opcode-operand combinations increase programming difficulty.*

Execution Times

Most TMS320C3x instructions nominally execute in one instruction cycle. However, as explained in the section on pipelining above, the TMS320C3x interlocking pipeline can cause some single-cycle instructions to require multiple cycles.

Non-delayed branch, call, return, and repeat instructions execute in four instruction cycles; delayed branches execute in one instruction cycle and provide three delay slots.

Instruction Set Highlights

The most noteworthy features of the TMS320C3x instruction set are the parallel execution instructions, the preponderance of three-operand instructions, and the interlocked instructions for read-modify-write bus cycles. Other noteworthy instructions include:

- Conditional load
- Delayed and non-delayed branches
- A delayed decrement-and-branch instruction that provides software looping with only one instruction cycle of overhead per loop iteration

Execution Control

Clocking

The TMS320C3x master clock can come from either an externally generated clock signal or an external crystal used in conjunction with an on-chip oscillator. The crystal or clock frequency is twice the instruction execution rate of the processor, e.g., 60 MHz for 30 MIPS execution on a 30 MIPS TMS320C31. The master clock and its complement are available on two output pins. The TMS320VC33 is the only family member to provide a phase-locked loop (PLL) to generate a master clock from a lower-frequency external clock. The TMS320VC33 PLL uses a fixed 5X multiplier and allows input frequencies from 5-20 MHz to the PLL.

Hardware Looping

The TMS320C3x provides both single- and multi-instruction hardware loops through the RPTS ("repeat single") and RPTB ("repeat block") instructions. RPTS repeats a single instruction from 1 to 2^{32} times. Interrupts are disabled during repetition. RPTB repeats a block of any number of instructions from 1 to 2^{32} times and allows interrupts. The RPTB instruction requires the programmer to explicitly load the repeat counter before the RPTB instruction.

Block repeats are not automatically nestable, although the programmer can nest loops by explicitly saving and restoring the state of the count, start address, end address, and status registers before and after the nested loop.

> *The inability to nest hardware loops without explicit save and restore operations costs cycles in some algorithms that contain several nested loops, e.g., the fast Fourier transform. The delayed decrement-and-branch instructions used for software loops incur less overhead than saving and restoring the multi-instruction hardware loop registers.*

Bootstrap Loading

The TMS320C30 does not provide any bootstrap loading support; on reset, the processor jumps to the memory address stored at memory location 0. In contrast, the TMS320C31, TMS320C32, and TMS320VC33 sample the status of the interrupt pins at reset. These pins can be used to cause the processor to bootstrap load over the serial port or from external memory at one of three locations, or to begin execution at the address contained in memory location 0. The TMS320C32 also provides a "handshake" bootstrap mode that uses the XF0 and XF1 pins as data-ready and data-acknowledge signals. The TMS320C31, TMS320C32, and TMS320CV33 all support bootstrap loading from 8-, 16-, and 32-bit-wide memory.

Interrupts

TMS320C3x processors provide a reset pin and four general-purpose external interrupt lines. The general-purpose external interrupt lines are level-sensitive on the TMS320C30 and TMS320C31, but can be configured to be edge- or level-sensitive on the TMS320C32 and TMS320VC33.

Internal interrupt sources include transmit and receive interrupts from the serial port (or ports, on the TMS320C30), one interrupt from each of the two timers, and a DMA controller interrupt. Software interrupts can be generated via the SWI instruction, the TRAP instruction, or by explicitly setting an interrupt flag.

Interrupts are prioritized but not automatically nestable. Each interrupt source can be individually enabled or disabled, and each interrupt source has its own interrupt vector. On the TMS320C32 (but not other TMS320C3x family members) the interrupt vector table can be relocated beginning at any 256-word boundary in memory. However, the reset interrupt vector is always at address 0.

When an interrupt occurs and interrupts are enabled, interrupts are disabled and the processor completes all fetched instructions. The program counter is then pushed onto the stack and then loaded with the address fetched from the appropriate entry in the interrupt vector.

For the TMS320C30 and any other family members that are not bootloaded, reset, interrupt, and trap vectors point directly to the appropriate interrupt service routines (ISRs). In this case interrupt latency is ten instruction cycles from the assertion of an external interrupt signal to the execution of the first word of the ISR, assuming the processor is in an interruptible state. If the TMS320C31 and TMS320VC33 are bootloaded, vectors are hard-coded in an internal ROM. If the TMS320C32 is bootloaded, the reset vector is at address 0 and, as mentioned above, the rest of the interrupt vector table can be located at any 256-word boundary. In either case, these vectors point to contiguous RAM locations that are presumed to contain instructions that branch to the appropriate ISR. Due to the extra branching instruction the pipeline must be purged before execution of the ISR, adding four more cycles to the latency.

Stack

The TMS320C3x uses a software stack with a dedicated stack pointer register. The stack pointer is not initialized at reset.

Peripherals

Peripherals on the TMS320C3x include synchronous serial ports (two on the TMS320C30, one on the TMS320C31, TMS320C32, and TMS320VC33), two 32-bit timers, a DMA controller, and two bit I/O pins.

- **Serial Ports**

  The TMS320C3x synchronous serial ports support 8-, 16-, 24-, or 32-bit data words at bit rates of up to the master clock divided by two (e.g., up to 30 Mbits/second on a 30 MIPS TMS320C31 processor). The receive and transmit sections of the ports are independent and can each be controlled by an external or internal clock. Internally generated clocks are the result of dividing the master clock by a programmable 16-bit divisor. If a serial port is not used, it can be used as two 16-bit timers or its pins can be used for bit I/O.

- **Timers**

  The 32-bit timers can take their clock from the processor's master clock divided by two or from an external clock input. They then count up until reaching values specified by their individual period registers, at which point they produce output pulses and/or generate interrupts. Timer outputs can be either pulses or square waves. There is only one external pin per timer, so a timer cannot both use an external clock source and produce an output pulse. The timer external pins can be used for general-purpose I/O if they are not used by the timers.

- **DMA**

  All TMS320C3x family members include an on-chip DMA controller with dedicated on-chip DMA address and data buses. The DMA controller on the TMS320C30, TMS320C31, and TMS320VC33 supports one channel of DMA, while the TMS320C32 DMA controller provides two channels. The programmer can configure each channel with a source memory address, a destination memory address, and a transfer count. The programmer can also specify whether the source or destination address should be incremented, decremented, or not changed after each transfer. The DMA controller can run free, or it can be synchronized to an external interrupt source (in which case it delays either reads or writes until a specific interrupt occurs). Once started, the DMA controller transfers data from the source address to the destination address independently of the CPU.

  In the event of a conflict between the DMA controller and the CPU (e.g., the CPU attempts two reads from a block of memory at the same time as the DMA controller attempts one read, exhausting memory bandwidth), the CPU is always given priority on the TMS320C30, TMS320C31, and TMS320VC33. On the

TMS320C32, the programmer can select whether the CPU or DMA controller should be given priority, or if they should alternate in priority. The programmer can also select which of the TMS320C32's two DMA channels is the higher-priority channel.

The presence of a DMA controller is an advantage. However, the DMA controller is not as flexible as DMA controllers found on other processors.

- **Bit I/O**

  Two bits of bit I/O are provided in the form of "external flag" pins. These can be configured as inputs or outputs under software control. These pins are also used by the "interlocked" instructions discussed previously.

On-Chip Debugging Support

The TMS320C30, TMS320C31, and TMS320C32 provide a five-pin, scan-based, on-chip debugging/emulation facility called the "Modular Port Scan Device" (MPSD) interface. This port allows the user to read and write memory and registers and execute instructions on the device. The debugging/emulation facility allows the user to insert breakpoints and to perform single-stepping. The port is not JTAG-compatible, although it is quite similar to a JTAG port. It does not support chaining together multiple TMS320C3x devices, nor does it support boundary scan.

The TMS320VC33 provides a JTAG-compatible debug port instead of the MPSD port of the other family members. This alternative allows multiple TMS320VC33s and other JTAG-compatible devices to be emulated with a single header, and supports boundary scan. Otherwise, debugging features of the two different ports are identical and Texas Instruments' emulation software is compatible with both.

Power Consumption and Management

The TMS320C31 is available in both 3.3- and 5.0-volt versions; the TMS320C30 and TMS320C32 are only available at 5.0 volts. The TMS320VC33 uses a dual supply of 1.8 V for the core and 3.3 V for external interfaces.

According to Texas Instruments, typical TMS320VC33 power consumption is 200 mW at 75 MIPS and 1.8 volts. The TMS320C31 has a typical power consumption of 1.2 W at 30 MIPS and 5.0 volts.

The TMS320C31, TMS320LC31, TMS320C32, and TMS320VC33 provide two power-down modes not available on the TMS320C30. The first mode, activated by the LOPOWER instruction, internally divides the processor clock down by a factor of 16. This slows processor execution and reduces power consumption approximately by a factor 16. The processor can be restored to full-speed operation via the MAXSPEED instruction; full-speed operation is attained during the read phase of the MAXSPEED instruction. In LOPOWER mode, the TMS320LC31 has a typical power consumption of 28 mW at 20

MIPS and 3.3 volts, the TMS320C31 has a typical power consumption of 63 mW at 30 MIPS and 5.0 volts, and the TMS320VC33 typically consumes approximately 12.5 mW at 75 MIPS and 1.8 volts. Note that in the LOPOWER mode the processors do not actually execute at a rate of 20, 30, or 75 MIPS respectively, as the clock is divided by a factor of 16. The actual instruction rates are 1.25, 1.88, and 4.69 MIPS.

The second power-down mode, entered via the IDLE2 instruction, disables the clock to the processor core (and turns off the output clock signal) until an unmasked external interrupt occurs. According to Texas Instruments, in IDLE2 low-power mode, the TMS320LC31 has a power consumption of 66 µW at 20 MIPS and 3.3 volts, the TMS320C31 consumes 250 µW at 30 MIPS and 5.0 volts, and the TMS320VC33 has a typical power consumption of 45 µW at 75 MIPS and 1.8 volts.

Benchmark Performance

This report does not include benchmark results for the TMS320C3x. Based on its performance on previous versions of the BDTI Benchmarks, we provide a general, qualitative description of the TMS320C3x's performance relative to that of the other processors in this report. In particular, we will compare the TMS320C3x with the ADSP-2106x as both are conventional floating-point architectures with many of the same target applications.

Execution Performance

- **Instruction cycle counts:** The TMS320C3x and ADSP-2106x have similar data paths. However, based on previous BDTI Benchmark analyses, the ADSP-2106x typically achieves slightly lower cycle counts on the BDTI Benchmarks. For example, the ADSP-2106x supports more versatile bit-field manipulation operations, and has more powerful conditional execution features. A key difference between the two architectures is that the ADSP-2106x supports a simultaneous add and subtract instruction that is not available on the TMS320C3x; hence, in earlier versions of the BDTI Benchmarks, the ADSP-2106x cycle counts for the **FFT** benchmark were approximately half as high as those of the TMS320C3x.
- **Execution times:** Although the TMS320C3x cycle counts are likely to be higher than those of the ADSP-2106x, the TMS320C3x is available with a slightly faster instruction execution rate; 75 MIPS for the TMS320VC33 compared to 66 MIPS for the ADSP-21065L. Thus, the execution times for the TMS320VC33 are likely to be similar to those of the ADSP-21065L, with the exception of the FFT algorithm (on which the ADSP-21065L is expected to be much faster because of its support for simultaneous add and subtract operations).
- **Cost-execution time:** In general, the TMS320VC33 and ADSP-2106x are likely to have similar execution times, but the price of the TMS320VC33 is more than 30% lower than that of the ADSP-21065L. This combination indicates that the

TMS320VC33 will have a better overall cost-execution time result than the ADSP-21065L.

- **Energy consumption:** The TMS320VC33 has much lower power consumption than the ADSP-21065L (approximately 200 mW for the TMS320VC33 compared to approximately 930 mW for the ADSP-21065L). Since their execution times are generally comparable, the TMS320VC33 is likely to have significantly lower energy consumption results than the ADSP-21065L.

> *The TMS320VC33 runs nearly twice as fast as any of the other TMS320C3x family members. Coupled with its decreased power consumption, the TMS320VC33 provides a long-overdue performance upgrade for the TMS320C3x family.*

Memory Usage

The focus in the memory usage analysis is on Control benchmark memory usage. Unlike other benchmarks, the Control benchmark is optimized for minimum memory usage. This benchmark is designed to indicate a processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks, reflecting the processor's program memory usage in general DSP code. Finally we discuss constant and non-constant data memory usage.

Because the TMS320C3x has not been benchmarked with the current suite of BDTI Benchmarks, we base our analysis on the processor's results on earlier versions of the BDTI Benchmarks.

- **Control code memory usage:** Based on previous results on BDTI's control-oriented benchmark, the TMS320C3x achieves significantly better code density on control code than the ADSP-2106x, in large part because of the difference in the two processors' instruction word widths: The TMS320C3x uses 32-bit instructions, compared to 48-bit instructions on the ADSP-2106x.
- **Program memory usage:** Based on previous benchmark results, the TMS320C3x requires much less program memory than the ADSP-2106x. Although the ADSP-2106x has a slightly more powerful instruction set than the TMS320C3x (and thus may require fewer instructions to implement some DSP algorithms), the ADSP-2106x still requires significantly more program memory because its instructions are so much wider than those of the TMS320C3x.
- **Data memory usage:** Based on previous benchmark results, the TMS320C3x data memory usage is typically somewhat higher than that of other processors that use 32-bit data words (such as the ADSP-2106x), in part because the TMS320C3x restricts immediate data to 16 bits. Unfortunately, the TMS320C3x uses 24-bit addresses that do not fit in 16 bits. Therefore, in order to load an address register with an address, look-up tables with the necessary addresses are used. On previous

versions of the BDTI Benchmarks, this consumed a small amount of constant data (typically, a few bytes per benchmark).

Cost

Price and packaging options for some members of the TMS320C3x family are shown in Table 7.12-4.

Fabrication Details

The TMS320C30, TMS320C31, and TMS320C32 are fabricated in a 0.62 µm, three-metal-layer CMOS process. The TMS320VC33 is fabricated in a 0.18 µm, four-metal-layer CMOS process.

The TMS320C3x is available as a core for use in ASICs designed by customers and fabricated by Texas Instruments.

Part	Speed (MIPS)	Voltage (V)	Package	Price (Qty. 10,000)
TMS320C30	25.00	5.0	181 PGA	$178.50
	20.00			$155.20
	16.5			$135.00
TMS320C31	40.00	5.0	132 PQFP	$34.51
	30.00			$31.36
	25.00			$28.52
	20.00			$25.93
TMS320LC31	20.00	3.3	132 PQFP	$25.93
TMS320C32	30.00	5.0	144 PQFP	$16.40
	25.00			$14.91
	20.00			$9.95
TMS320VC33	75.00	3.3 I/O 1.8 core	144 PQFP	$16.37
	60.00			$12.32

TABLE 7.12-4. TMS320C3x price and package summary as supplied by Texas Instruments, June 2000.

Development Tools

Texas Instruments provides a comprehensive set of code generation, debugging, and system integration tools all of which are well described in their publication, *The TMS320 DSP Development Support Reference Guide.* The basic code generation tools include an optimizing C and Ada-compiler, assembler, linker, and archiver. These tools are available for DOS, Windows 9x, or UNIX (HP-UX or Solaris). Also for Windows 9x, Texas Instruments offers "Code Composer," an integrated development environment that integrates all of the above code generation functionality with simulation, emulation, and profiling tools.

> *The development and debugging facilities of Code Composer are much more sophisticated than those of the Texas Instruments tools.*

Debugging and emulation software tools include a simulator and emulator. These tools employ one of two interfaces. One runs from a DOS command prompt in Windows and supports basic mouse functionality. The more sophisticated Code Composer interface is a true Windows application. The emulator supports both JTAG and MPSD and can interface to a variety of different hardware. Hardware offered by Texas Instruments includes the Extended Development System (XDS) and the TMS320C30 Evaluation Module (EVM). The XDS is a card that plugs into either a PC or a SPARC workstation and connects to any TMS320 hardware that implements the JTAG or MPSD interface. The TMS320C30 EVM is available as a PC card that employs a 16.5 MIPS TMS320C30 and some peripheral devices for real-time verification of TMS320C3x code. Another hardware tool from Texas Instruments, the DSP Starter Kit (DSK), is a low-cost board with parallel-port PC interface, a reduced-feature debugger, self-linking assembler, and some peripherals. This evaluation tool is targeted towards first-time users. A variety of other hardware is also available through Texas Instruments third-party vendors. Code Composer supports many of these third-party emulation boards with pre-installed drivers.

> *The basic DOS-based interface for the TMS320C3x tools is quite capable, but not as easy to use as one might hope. For example, values can not be loaded into memory from an ASCII file. The interface does provide good debugging and profiling features, and it has strong support for C source-level debugging.*
>
> *The Code Composer interface adds many useful project management and debugging features that make it more flexible then the basic DOS-based interface.*
>
> *The simulator (used in both the DOS-based tools and Code Composer) is not cycle accurate. This greatly decreases its utility as a measuring tool.*
>
> *The number of third-party vendors providing tools and software for Texas Instruments' DSPs is the largest of any DSP processor vendor. This is due partly to Texas Instruments early entry into the DSP*

processor market and to its strong efforts to cultivate third-party support.

Applications Support

The main source of documentation for the TMS320C3x is the *TMS320C3x User's Guide*, which covers the architecture, instruction set, and programming of the device. At the time of writing, however, the manual has not been updated with the details of the TMS320VC33. Instead, Texas Instruments has provided a supplement entitled *How to Begin Development Today with the TMS320VC33 DSP.* Also, volume III of Texas Instruments' *Digital Signal Processing with the TMS320 Family* contains a number of application notes and article reprints relevant to the processor. A number of shorter application notes are also available as part of Texas Instruments' "Designer's Notebook Pages" and "DSP Application Reports." Both of these resources can be downloaded from Texas Instruments' website. All of these resources are adequately summarized in the *TMS320 DSP Development Support Reference Guide.*

Advantages

- Fairly orthogonal instruction set
- Barrel shifter
- Good parallel move support
- 32-bit floating-point arithmetic
- Eight 40-bit extended-precision registers
- Many address registers
- Instruction cache for accelerating external memory access
- Large, unified address space
- Two independent address generation units
- Flexible external memory interface with byte packing/unpacking (TMS320C32 only)
- Programmable and externally generated wait states
- Two external buses (TMS320C30 only)
- Good on-chip memory bandwidth
- Software stack plus indexed addressing mode facilitate compiler code generation
- On-chip DMA controller with separate DMA address and data buses
- Two timers
- Two serial ports (TMS320C30 only)
- Clock divider (TMS320C31, TMS320C32, and TMS320VC33)
- Special instructions for inter-processor communication

- Flexible bootstrap modes (TMS320C31, TMS320C32, and TMS320VC33)
- Core version available
- Low-cost family variants available (e.g., less than $10 for a 20 MIPS TMS320C32, any quantity; less than $13 for a 60 MIPS TMS320VC33, quantity 10K)
- JTAG-compatible emulation port on TMS320VC33
- Extensive third-party support

Disadvantages

- No guard bits on integer operations
- Multiply-accumulate operands are restricted
- Pipeline effects complicate code optimization
- Only one circular buffer size at a time
- Limited indexing features for bit-reversed addressing mode
- Large instruction word increases system cost
- External memory writes take two instruction cycles
- Second external bus on TMS320C30 supports only 16 Kwords of memory
- Hardware loops not nestable without explicit state saving
- Scan-based emulation port on TMS320C30, TMS320C31, and TMS320C32 not JTAG compatible
- User's Guide not updated for TMS320VC33
- Simulator not cycle-accurate

7.13 Texas Instruments TMS320C54xx Family

BDTImark2000 Score: 500 at 160 MHz

Introduction

The Texas Instruments TMS320C54xx is a family of 16-bit fixed-point DSPs. TMS320C54xx processors are targeted at high-volume wireless applications (such as digital cellular telephones) and at networking, wireline voice communications, and computer telephony applications. The first members of the TMS320C54xx family were introduced in Japan in 1994 and in the U.S. in 1995.

The fastest processors in the family run at 160 MHz with a 1.6-volt core voltage supply in a 0.15 μm process (TMS320VC5416). Other family members execute at 120 MHz with a 2.5-volt supply fabricated in 0.25 μm (TMS320VC5410); at 80 MHz with a 1.8-volt supply fabricated in 0.18 μm (TMS320UC5402); and at 30 MHz with a 1.2-volt supply fabricated in 0.18 μm (TMS320UVC5402). Multiple-core TMS320C54xx family members are available with each core running at 100-133 MHz.

Despite their confusingly similar names, the TMS320C54xx and TMS320C5x are distinct processor families with different architectures, and the TMS320C54xx is not compatible with predecessor Texas Instruments processor families. The TMS320C54xx architecture is similar to that of the earlier TMS320C5x family, but adds a number of significant architectural enhancements such as accumulator guard bits, improved parallel moves, increased memory bandwidth, and hardware to accelerate Viterbi decoding. Characteristics of the various family members are shown in Table 7.13-1.

The TMS320C54xx is available as a core for use in SoCs designed by customers and fabricated by Texas Instruments. The core is designated the "CLEAD." Contact Texas Instruments for further information on their core-based SoC offerings.

In February of 2000, Texas Instruments announced the successor to the TMS320C54xx architecture, the TMS320C55xx. This architecture is discussed in detail in Section 7.14.

Architecture

TMS320C54xx processors contain a 16-bit fixed-point data path used for integer or fractional arithmetic, a data address generator, a program control unit, RAM and ROM, four sets of on-chip buses, and several peripheral interfaces. Figure 7.13-1 illustrates the TMS320C54xx family architecture as typified by the TMS320C541.

Data Path

The TMS320C54xx data path is based on five function units: a 16×16→32-bit multiply-accumulate unit which contains a 40-bit dedicated adder, a 40-bit ALU, a barrel shifter, an exponent detector, and a compare-select-store unit. Two 40-bit accumulators, designated A and B, are available.

Device	# of Cores	Core MHz	Core Volts	On-Chip Memory		Serial Ports (Std/ TDM/ BSP/ McBSP)	Host Port	DMA Ch.
				Program /Data RAM	Program /Data ROM			
’VC5441	4	133	1.5v	640Kx16	—	0/0/0/12	HPI 16	24
’VC5421	2	100	1.8v	256Kx16	4Kx16	0/0/0/6	HPI 16	12
’VC5420	2	100	1.8v	200Kx16	—	0/0/0/6	HPI 16	12
’VC5416	1	160	1.5v	128Kx16	16Kx16	0/0/0/3	HPI 8/16	6
’VC5410	1	120	2.5v	64Kx16	16Kx16	0/0/0/3	HPI 8	6
’VC5409	1	100	1.8v	32Kx16	16Kx16	0/0/0/3	HPI 8/16	6
‘VC549	1	120	2.5v	32Kx16	16Kx16	0/1/0/2	HPI	—
’VC5402	1	100	1.8v	16Kx16	4Kx16	0/0/0/2	HPI 8	6
‘UVC5409	1	30	1.2v	32Kx16	16Kx16	0/0/0/3	HPI 8/16	6
’UVC5402	1	30	1.2v	16Kx16	4Kx16	0/0/0/2	HPI 8	6
‘UC5409	1	80	1.8v	32Kx16	16Kx16	0/0/0/3	HPI 8/16	6
’UC5402	1	80	1.8v	16Kx16	4Kx16	0/0/0/2	HPI 8	6
‘LC549	1	80	3.3v	32Kx16	16Kx16	0/1/2/0	HPI	—
‘LC546A	1	66	3.3v	6Kx16	48Kx16	1/0/1/0	—	—
‘LC545A	1	66	3.3v	6Kx16	48Kx16	1/0/1/0	HPI	—
‘LC543	1	50	3.3v	10Kx16	2Kx16	0/1/1/0	—	—
‘LC542	1	50	3.3v	10Kx16	2Kx16	0/1/1/0	HPI	—
‘LC541	1	66	3.3v	5Kx16	28Kx16	2/0/0/0	HPI	—
‘C542	1	40	5.0v	10Kx16	2Kx16	0/1/1/0	HPI	—
‘C541	1	40	5.0v	5Kx16	28Kx16	2/0/0/0	HPI	—

TABLE 7.13-1. TMS320C54xx family variants. RAM and ROM sizes are approximate because some registers are memory mapped to the lower portion of memory.

The TMS320C54xx provides a 40-bit ALU that is used for arithmetic and logical operations. ALU inputs can come directly from data memory, the output of the barrel shifter, or the accumulators. Data can be fetched from memory, flow through the barrel shifter, and serve as an input to the ALU within a single instruction cycle, thus enabling combined ALU/shifter operations. ALU outputs are always stored in one of the two accumulators.

The ALU supports sign extension and saturation. Sign extension is enabled by setting a sign extension mode bit in a control register. In this mode, 16-bit ALU inputs are sign extended to 40 bits. Similarly, saturation is enabled by setting an overflow mode bit in a control register. In overflow mode, ALU outputs that require more than 32 bits to represent are saturated to the maximum magnitude positive or negative number that fits in 32 bits as they are stored in the destination accumulator. Otherwise, the upper eight bits in the accumulator are used as guard bits. Except for the TMS320LC542 and TMS320LC543, the TMS320C54xx has an additional saturation mode that allows the result of a multiplication to be saturated *before* it is accumulated in multiply-accumulate operations. In this

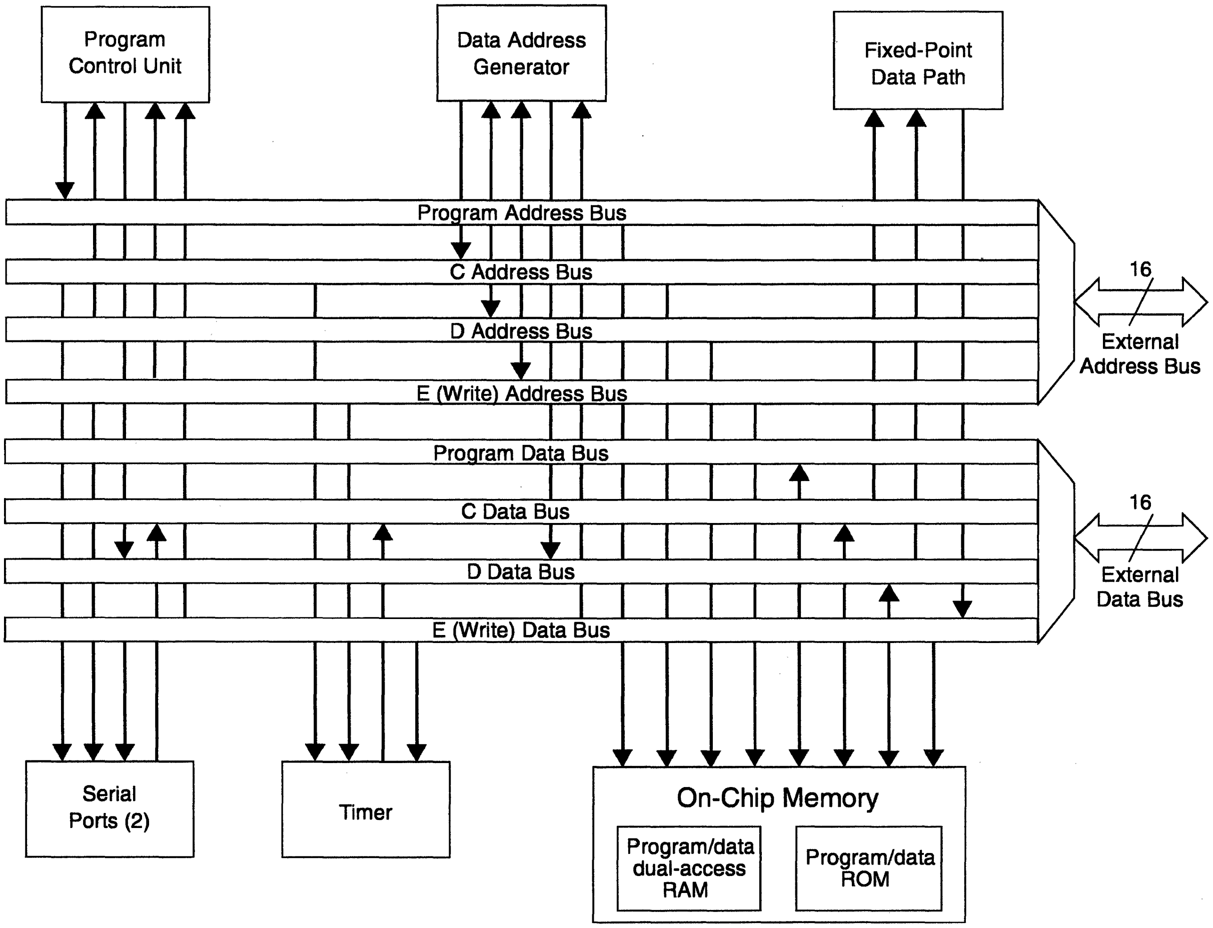


FIGURE 7.13-1. TMS320C541 processor architecture.

mode a multiplication of the two largest negative numbers representable with 16 bits is automatically saturated to the largest positive number representable with 32 bits.

The TMS320C54xx provides status bits that indicate if the last operation produced a carry or overflow. Add-with-carry and subtract-with-borrow operations are supported.

The TMS320C54xx ALU supports dual 16-bit operations. In this mode, the lower 32 bits of the ALU act as two parallel 16-bit ALUs. Among other things, this capability is useful in conjunction with the compare-select-store unit (described below) for use in implementing the Viterbi algorithm. Dual-operation mode is enabled by setting a bit in a control register.

The TMS320C54xx features a 40-bit barrel shifter that can shift values left by up to 31 bits or right by up to 16 bits in a single instruction cycle. Shifter inputs can come directly from data memory or either of the two accumulators. Shifter outputs can be sent to the ALU or stored in memory.

The TMS320C54xx multiply-accumulate (MAC) unit performs a 16×16→32-bit multiply-accumulate operation in a single instruction cycle. A dedicated 40-bit adder allows the multiplication result to be accumulated with the contents of either 40-bit accumulator, or with zero. The output of the 40-bit adder is always stored in accumulator A or B. The multiplier supports signed/signed multiplication, signed/unsigned multiplication, and unsigned/unsigned multiplication. These operations allow efficient extended-precision arithmetic. Also, fractional multiplication may be supported through the use of a fractional mode bit in a control register. Many instructions using the MAC unit can optionally specify automatic biased rounding. Inputs to the MAC unit can come directly from data memory, program memory, a 16-bit temporary data register T, or bits 16 to 32 of the A accumulator.

A compare-select-store unit (CSSU) is part of the processor's data path and can be used to implement the Viterbi algorithm efficiently. The CSSU compares two 16-bit halves of the lower 32 bits of an accumulator, storing the larger of the two values to memory. It also shifts a bit indicating which half was the larger into the 16-bit transition register (TRN). The TRN register is then used to keep track of the comparison history for determining the maximum likelihood path through the Viterbi algorithm's state trellis.

An exponent detector in the data path can be used to compute the number of redundant sign bits in either accumulator in a single instruction cycle via the EXP instruction. The exponent value is stored in the T register. A subsequent NORM instruction can be used to normalize an accumulator by the shift amount stored in the T register in a single instruction cycle.

All arithmetic and logical operations as well as multiplications are performed in a single instruction cycle.

> *The TMS320C54xx data path is both powerful and well thought out. The CSSU and dual 16-bit operations should be extremely useful in many wireless communications applications requiring Viterbi*

decoding, and the MAC unit is well suited for extended-precision arithmetic.

Memory System

The TMS320C54xx divides memory into word-addressable program, data, and I/O spaces. Each space can contain 64 Kwords of instructions or data. The newer TMS320C54xx members support 256K - 8M words of program memory using a page register that is modified in software. When internal RAM is mapped to program memory, the internal program RAM is mapped in the lower 32 Kwords of each page and the upper 32 Kwords are paged external memory. All TMS320C54xx family members access no more than 64 Kwords of memory per page.

The processor accesses on-chip memory over four sets of address and data buses: one set of program address and data buses, two sets of data read address and data buses, and one set of data write address and data buses. The program bus set is connected to program/data ROM and RAM. In addition, the data bus sets are connected to one or more blocks of dual-access program/data RAM (DARAM) that support two read operations or one read and one write operation per instruction cycle. DARAM can be mapped as data space only or program/data space. Most of the TMS320C54xx family members also contain single-access RAM (SARAM) that can be mapped either as data space or as program/data space.

Most TMS320C54xx family members have both on-chip SARAM and DARAM; the TMS320VC5402, TMS320VC5409, and TMS320VC5416 contain on-chip dual-access RAM only. The TMS320VC5410, for example, includes 56 Kwords of on-chip single-access RAM and 8 Kwords of dual-access RAM, mapped into data space or program/data space. The amount of DARAM varies between 4 Kwords and 128 Kwords among family members. The amount of SARAM varies between 0 Kwords and 256 Kwords. DARAM and SARAM is divided into blocks of 1 Kword, 2 Kwords, 4 Kwords, or 8 Kwords (depending on the family member) so that data from one block can be accessed by the CPU while another block is being accessed by the DMA without incurring a stall.

The dual-core TMS320C5420 contains both DARAM and SARAM that can be used as data space only or as program/data space. Each core on the TMS320C5420 has 16 Kwords of DARAM and 84 Kwords of SARAM. None of the memory is shared between the cores. The dual-core TMS320C5421 and four-core TMS320C5441 do allow sharing of program SARAM only. Each core in the TMS320C5421 has 32 Kwords of program/data DARAM and 32 Kwords of data SARAM. The two cores share 120 Kwords of program DARAM. Each core in the TMS320C5441 has 64 Kwords of single-access program RAM, 32 Kwords of program/data DARAM, and 64 Kwords of data SARAM. The four cores are grouped in two pairs where each pair shares (two-way, read-only) the combined 128 Kwords of single-access program RAM.

The combination of dual-access RAM, single-access RAM, program ROM, and the processor's four sets of buses could allow the processor to perform one instruction fetch, two data reads, and one data write per instruction cycle. However, the TMS320C54xx does not provide instructions that perform two data reads and one data write simultaneously. The maximum data bandwidth that is supported by the instruction set is two data reads, or one data read and one data write. Thus, at the maximum master clock speed of 160 MHz (TMS320VC5416), this yields an on-chip peak and maximum sustainable data memory bandwidth of 320 million 16-bit words/second for reads, or 160 million 16-bit words/second for reads and 160 million 16-bit words/second for writes.

Although the TMS320C54xx provides four sets of buses, its per-cycle memory bandwidth is not larger than those of most competitor conventional DSP processors.

DARAM can be mapped into either program space or program and data spaces under software control on all TMS320C54xx family members. Additionally, all family members except the TMS320VC5420 and TMS320VC5441 feature on-chip program ROM. On the processors that have program ROM, the ROM can be disabled allowing mapping of more external memory. In the case of the TMS320VC5402, approximately 4 Kwords of program ROM can optionally be mapped into both program and data memory space. On the TMS320LC545A and the TMS320LC546A, approximately 16 Kwords of program ROM can be mapped into program and data memory space. Depending on the processor, program ROM is divided into blocks of 2 or 4 Kwords each, and each block supports a single read access per instruction cycle. Thus, fetching two values from data ROM per instruction cycle can only occur if the values reside in different blocks. Attempting to fetch two data operands (or one program fetch and one data fetch) simultaneously from the same ROM block results in a one-instruction-cycle penalty.

The TMS320C54xx has special support for handling 32-bit data. 32-bit values can be stored in contiguous memory locations in on-chip RAM or ROM, and special "double" instructions (double load, double store, double add, etc.) can be used to access two sequential 16-bit values as one 32-bit value in one instruction cycle. The address specified for the 32-bit value is the address of the upper 16 bits of the 32-bit word. The lower 16 bits are fetched from the location before or after the address given, depending on the address of the 32-bit value being odd or even.

Few competing 16-bit conventional DSP processors have this level of support for double-precision operations. However, memory accesses on 32-bit quantities execute differently for even and odd addresses. This can cause difficulties for applications that do not align operands in memory correctly.

Many of the processor's registers (for example, accumulators and status registers) are mapped into the lower 32 words of data RAM. This allows context save and restore operations to be accomplished by using the processor's block memory move instructions.

Some TMS320C54xx instructions can access read-only data from program ROM over the program bus. These instructions execute in multiple instruction cycles due to contention for the program data bus, unless they are repeated within a single-instruction repeat loop. Examples of these instructions include various program/data memory move instructions as well as some TMS320C5x-compatible multiply-accumulate instructions.

For the TMS320C541, TMS320LC545A, and TMS320LC546A, the ROM includes an interrupt vector table and built-in self-test code; the remaining ROM is factory-programmed with customer code.

The 2Kx16 ROM on the TMS320LC542, TMS320LC543, TMS320LC549, and TMS320VC5410 is factory programmed with a 256-word μ-law expansion table, a 256-word A-law expansion table, a 256-word sine table, built-in self-test code, an interrupt vector table, and a boot loader. The 16 Kword ROM on the TMS320VC5410 and TMS320VC5416 adds 256/1024-point radix-2 decimation-in-time (DIT) FFT functions and twiddle factors (coefficients), and coefficient tables for the GSM EFR speech coder.

External Memory Interface

The external address and data buses on the TMS320C541, TMS320C542, TMS320LC543, TMS320LC545A, and TMS320LC546A are 16 bits wide, addressing 64 Kwords of external program memory. Most of the newer TMS320C54xx family members have a 16-bit data bus and an 18-bit to 23-bit external address bus, thus addressing 256 Kwords to 8 Mwords of paged program memory. (Paging is discussed further in the *Address Generation Units* subsection, below.) The four-core TMS320VC5441 does not extend address and data buses off-chip; data must be transferred in and out of the device through the host port interface (HPI) or via DMA transfers through the serial ports.

For processors with an external memory interface, external reads take one instruction cycle assuming no wait states; external writes take two instruction cycles in the absence of wait states, unless the write is immediately preceded or followed by an external read, in which case the write takes three instruction cycles. At 160 MHz, this yields a peak and maximum sustainable external memory bandwidth of 160 million 16-bit words/second for reads and 80 million 16-bit words/second for writes.

> *Like the TMS320C5x, the TMS320C54xx requires multiple instruction cycles for writes to external memory. This reduces performance in some applications.*

The four on-chip bus sets are multiplexed onto the single external bus set. If an instruction requires more than one external access, the accesses are sequenced over multiple instruction cycles. Priority is given first to data writes, then to data reads, and then to instruction fetches.

Slow external devices can be accommodated either via programmed or externally requested wait states. Programmed wait-state support allows zero through seven wait states to be generated when accessing external memory. The newer TMS320C54xx family members, such as the TMS320VC5402, TMS320VC5416, and TMS320VC5421, have a control register bit to double the number of wait states. When this bit is set, an area of memory programmed for seven wait-states will wait for 14 cycles. Different wait-state values can be programmed for I/O space, the lower half of data memory, the upper half of data memory, the lower half of program memory, and the upper half of program memory. Externally requested wait states are generated by an external device asserting the READY pin.

The TMS320C54xx $\overline{\text{HOLD}}$ input allows an external device to obtain exclusive access to the processor's external memory. When an external device asserts $\overline{\text{HOLD}}$, the processor completes any external bus cycle in progress, places its external buses in a high-impedance state, and then asserts the $\overline{\text{HOLDA}}$ (hold acknowledge) pin. The external device can then read or write the processor's external memory (e.g., for block-based I/O) without fear of bus conflicts. When finished, the external device deasserts $\overline{\text{HOLD}}$, allowing normal execution to resume. When $\overline{\text{HOLD}}$ is asserted, the TMS320C54xx can proceed in one of two user-selectable modes: if the HM bit in the ST1 control register is set, the processor simply halts when $\overline{\text{HOLD}}$ is asserted. If HM is cleared, the processor continues to execute while $\overline{\text{HOLD}}$ is asserted, as long as no external memory accesses are required.

The TMS320C54xx can automatically insert an extra wait state when it crosses certain memory address boundaries. This feature, called programmable bank switching, provides extra time for a slower memory to release the bus when switching from one memory bank to another. Bank sizes can be powers of two from 4 Kwords to 32 Kwords, inclusive.

Address Generation Units

The TMS320C54xx supports immediate data and register-direct, paged memory-direct, stack pointer (SP) relative, memory-direct, and register-indirect addressing modes.

Paged memory-direct addressing and stack pointer relative addressing use the same instructions; the CPL (compiler mode) bit in status register ST1 determines which addressing mode is used. Paged memory-direct addressing is implemented as on the TMS320C5x family: a 9-bit data-page register is combined with a 7-bit memory-direct address stored in the instruction word to form a complete 16-bit data address. In stack pointer-relative addressing, the 7-bit value stored in the instruction word is interpreted as an unsigned offset that is added to the stack pointer to produce the effective address. Stack pointer-relative addressing is supported by the TMS320C54xx C compiler. Most TMS320C54xx instructions support long memory-direct addressing, meaning that the data address is specified by a 16-bit extension word following the instruction.

The long memory-direct addressing mode costs extra cycles and memory, but is very convenient for the programmer.

Addresses generated in register-indirect addressing come from eight address registers, AR0-AR7. The TMS320C54xx data address generator can generate addresses for two register-indirect accesses every instruction cycle.

Two different sets of register-indirect modification modes are supported. The first set supports post-increment (optionally with circular addressing), post-decrement (optionally with circular addressing), pre-increment, post-increment by the contents of AR0 (optionally with either circular or bit-reversed addressing), post-decrement by the contents of AR0 (optionally with either circular or bit-reversed addressing), indexed by 16-bit immediate data, and pre-increment by 16-bit immediate data. These update modes are only available for instructions that access a single operand (which in some cases can be 32 bits wide) from data memory.

The second set of register-indirect modification modes are used with instructions that use operand-unrelated parallel moves. This set is more limited than the previous set and allows only post-decrement, post-increment, and post-increment by the contents of AR0 with circular addressing. Additionally, only registers AR2 through AR5 can be used in such instructions.

> *The restriction that only address register AR0 can be used for post-incrementing and post-decrementing by an offset other than one is limiting in some applications.*
>
> *A common problem in supporting operand-unrelated parallel moves on processors with 16-bit instructions is that there are not enough bits in an instruction word to encode all possible operation and addressing combinations, resulting in a non-orthogonal instruction set. Texas Instruments' solution of supporting a wide variety of indirect update modes for single-operand instructions but relying on a* standard *subset of addressing modes for instructions with dual accesses is a good compromise. However, the instruction set is still non-orthogonal compared to other DSPs with wider instruction words.*

The newer TMS320C54xx members support 256K - 8M words of program memory using a page register that is modified in software. The page register selects which of up to 128 64-Kword pages (8 Mword versions) of program memory is active. Special instructions are also provided for branching to 23-bit addresses. Some instructions have been modified to allow loads and stores at 23-bit program memory addresses.

The TMS320C54xx differs from previous fixed-point processors from Texas Instruments in that instructions specify the address register to use for the current instruction rather than for the next instruction. That is, to add the values pointed to by AR3 and AR4 to the accumulator on the earlier TMS320C5x, one would write:

```
; Assume ARP points to AR3
ADD    *,0,AR4
ADD    *,0
```

In the above instructions, the ",AR4" indicates that on the next instruction the value of "*" will be the value pointed to by AR4. In contrast, TMS320C54xx code would look like:

```
ADD    *AR3,0,A
ADD    *AR4,0,A
```

The TMS320C54xx has a "compatibility mode" that allows it to function like a TMS320C5x processor in this regard.

> *The ability to specify address registers to be used in the current instruction rather than for the next instruction simplifies assembly language programming compared to the predecessors of the TMS320C54xx, and is standard on competing processors.*

The TMS320C54xx supports bit-reversed addressing through reverse carry propagation. Using this mode, the currently selected address register pointer is used for address generation, but the register is post-modified by adding or subtracting the contents of address register AR0 using reverse carry arithmetic.

The TMS320C54xx supports modulo addressing for use with circular buffering. Any auxiliary register can be used with modulo addressing, but only one circular buffer size, specified by the BK register may be active at one time. The circular buffer can be any size that fits in memory; however, the buffer must be aligned on an address that is evenly divisible by *k*, where *k* is the smallest power of two that is greater than or equal to the size of the circular buffer.

> *Most competitor DSP processors support at least two simultaneous circular buffers of different lengths. The TMS320C54xx restriction to a single size will be a limitation in some applications.*

Pipeline

The TMS320C54xx uses a six-stage partially interlocked pipeline, broken into prefetch, fetch, instruction decode, "access" (data address generation), data read, and execute/write.

There are a number of pipeline effects that are visible to the user. Some of the more important are:

- Unconditional branches take four instruction cycles due to the pipeline being flushed. Conditional branches take five instruction cycles if the condition is true, and three if the condition is false and the branch is not taken. Delayed versions of both kinds of branches are available. Delayed unconditional branches take four instruction cycles to execute, but two of these cycles can be used to execute instructions in the delay slots. Delayed conditional branches take five instruction cycles whether or not the condition is true, but, as with unconditional delayed branches, two of these instruction cycles can be used for instructions in the delay slots.

- The conditions used by the conditional execution instruction (XC) are those established two instruction cycles prior to the execution of the XC instruction.
- Some instructions that modify an address register or the circular buffer size register may require up to two instruction cycles before the new value in the address register is available. The exact number of cycles depends on the instructions following the modification of the address register.
- Similarly, some instructions that modify the data page pointer (DP) or stack pointer (SP) may require up to three instructions cycles before the new value can be used.

The TMS320C54xx six-stage pipeline is deep compared to many other conventional DSPs. The TMS320C54xx pipeline impacts the programmer more than those of previous generations of fixed-point processors from Texas Instruments, due to its greater depth.

Instruction Set

The TMS320C54xx registers and instruction set are summarized in Tables 7.13-2 and 7.13-3, respectively. The processor uses mostly single 16-bit word instructions; instructions that use long immediate data use two 16-bit words. Also, instructions using the long memory-direct addressing mode will use an extra 16-bit word, so that some instructions require a total of three 16-bit words.

The TMS320C54xx instruction set includes the instructions used by its predecessors, the TMS320C2x, TMS320C2xxx, and TMS320C5x, but uses a different operand format and is not assembly language source-code compatible with these earlier processors.

Although the TMS320C54xx is not compatible at the assembly language level with its predecessors in the Texas Instruments

Registers	Width	Purpose
A, B	40 bits	Accumulators
T	16 bits	Multiplier input register
SP	16 bits	Stack pointer register
AR0-AR7	16 bits	Address registers
BK	16 bits	Circular buffer block size register

TABLE 7.13-2. TMS320C54xx register summary.

fixed-point DSP processor families, it uses many of the same instructions; a programmer migrating from a TMS320C54xx predecessor would feel fairly familiar with the TMS320C54xx instruction set.

Class	Instructions
Arithmetic	Absolute value, add, add with carry, add with shift, negate, subtract, subtract with borrow, subtract with shift, sum and difference
Multiplication	Multiply-accumulate (with or without rounding), multiply (signed/signed, signed/unsigned, unsigned/unsigned), multiply-subtract, square-accumulate, square-subtract
Logic	*And*, *or*, *exclusive-or*, *not*
Shifting	Arithmetic/logical shift left 0-31 bits, right 0-16 bits
Rotation	Rotate accumulator left/right through carry, rotate accumulator left through carry and TC bit
Conditional Execution	Execute one or two instructions conditionally, store certain registers conditionally
Comparison	Test for less than, greater than
Looping	Single and multi-instruction repeat, multi-instruction repeat delayed
Branching	Conditional and unconditional branch, branch on value of address register, branch to address in accumulator. (All branches available in delayed and non-delayed versions.)
Subroutine Call	Conditional and unconditional call, indirect call, return, return conditionally. (All call/return instructions available in delayed and non-delayed versions.)
Bit Manipulation	Test bit in memory; *and*, *or*, *exclusive-or* bit pattern direct to memory
Special Function	Absolute difference between two points, compare-select, exponent detect, load register and move data in delay line, LMS filter tap update, maximum value, minimum value, normalize, polynomial evaluation step, saturate, square difference between two points, stack push, stack pop, symmetric FIR filter computation

TABLE 7.13-3. TMS320C54xx instruction set summary.

Assembly Language Format

TMS320C54xx assembly language uses the traditional opcode-operand format. As an example, the following code implements the inner loop of a 128-tap FIR filter:

```
RPT         #127
MAC    )    *AR2+,*AR3+,A
```

The RPT instruction starts a hardware loop for 128 repetitions, causing the MAC instruction to be repeated 128 times. Each execution of the MAC instruction fetches the two operands pointed to by AR2 and AR3 over the data buses, multiplies them, and accumulates the products in the A register. AR2 and AR3 are then post-incremented.

Some instructions allow operand-unrelated parallel moves. The parallel move is specified as a separate field of the instruction; for example:

```
ST    A,*AR2
||    ADD    *AR3,B
```

which stores the contents in accumulator A to the memory location specified by the contents of AR2, and simultaneously adds the contents of accumulator B to the contents of the memory location specified by AR3 and stores the result in accumulator B. The "||" keyword indicates that the two instructions are to be executed in parallel.

Parallel Move Support

In a departure from the previous-generation TMS320C5x architecture, the TMS320C54xx adds limited support for operand-unrelated parallel moves. Adopting syntax from the TMS320C3x/C4x families, the TMS320C54xx includes instructions to load or store a value from or to memory in parallel with ALU and multiply operations such as multiply, multiply-accumulate, add, and subtract. The TMS320C54xx also has an instruction that permits a store in parallel with a load.

> *The TMS320C54xx is the first fixed-point processor from Texas Instruments to support operand-unrelated parallel moves. This simplifies programming and increases performance on algorithms that require frequent, nonlinear access to single values in memory, such as the FFT butterfly.*

The TMS320C54xx can perform up to one load or one store in parallel with another operation.

Orthogonality

The TMS320C54xx instruction set is not particularly orthogonal for a 16-bit DSP processor. The number of classes of instructions that support different addressing modes is limited, and the large number of specialized instructions consume bits in the instruction word that could otherwise have been used to implement a more orthogonal instruction set.

Execution Times

Most TMS320C54xx instructions execute in a single instruction cycle when executed from internal memory. Instructions that use 16-bit immediate data (which is stored in a separate program word from the instruction) require an additional instruction cycle. Some instructions that use the program bus for data movement (e.g., MACD) execute in multiple instruction cycles unless they are repeated within a single-instruction repeat loop. Branches take four or five cycles, but delayed branches allow up to two instructions to be executed within the delay slots.

Instruction Set Highlights

The TMS320C54xx has a large number of instructions optimized for communications applications:

- The compare-select instruction (CMPS) compares two 16-bit values stored in an accumulator, stores the larger of the two to memory, and shifts a bit indicating which one was larger into a history register. CMPS is useful in implementing a Viterbi decoder.
- Six instructions (DADD, DADST, DRSUB, DSADT, DSUB, DSUBT) perform 32-bit arithmetic operations by fetching two 16-bit words from memory. Depending on a mode bit, these instructions can perform either two separate 16-bit operations or one 32-bit operation. In the former case, several of the instructions compute both sum and difference values, which is useful for Viterbi decoding. Separate 32-bit load and store instructions (DLD and DST) are also available.

  The CMPS instruction and the various dual 16-bit arithmetic instructions will be very useful for many communications applications.

- MIN and MAX instructions that store the smallest or largest values of the two accumulators in either accumulator, and set the carry bit to indicate the source.
- A delayed decrement-and-branch instruction that provides software looping with only two instruction cycles of overhead per loop iteration.
- The FIRS instruction can be used in a single-instruction repeat loop to implement a symmetric FIR filter in one-half the execution time and one-half the coefficient memory of a standard implementation. This is accomplished by fetching a coefficient from program memory, multiplying it by the value in the A accumulator, and accumulating the result in the B accumulator. Simultaneously, the instruction fetches two state variables (stored in different banks of dual-access memory) from memory and adds them together, storing the result in the A accumulator for the next iteration of the FIRS instruction.

  The FIRS instruction allows a considerable performance gain for symmetric filters (which are quite common in communications applications). FIRS loses one bit of precision relative to a standard *implementation of an FIR filter because the two state variables that are added together must be scaled by one bit to ensure that their sum does not overflow. This is likely to be an acceptable trade-off in many cases.*

- The LMS instruction performs both a dot product iteration and a coefficient update (including rounding) for an LMS adaptive filter.
- The POLY instruction allows computation of an n^{th}-order polynomial in *n* instruction cycles.
- The ABDST and SQDST instructions compute the absolute difference (i.e., $|x\text{-}y|$) or squared difference (i.e., $(x\text{-}y)^2$) between the values stored in two memory locations. In parallel, the instruction accumulates the distance computed by the last instruction. Thus, in a repeat loop these instructions compute the absolute or squared distance between two vectors.

Other noteworthy instructions include:

- Single-cycle exponent detect
- Single-cycle normalize
- Both delayed and non-delayed versions for branch, call, and return instructions
- A general-purpose conditional execution instruction that conditionally executes the following one or two instructions
- Several instructions (ANDM, ORM, XORM) that manipulate bits directly in memory without interfering with the accumulators

The number and capabilities of TMS320C54xx special function instructions is impressive.

Execution Control

Clocking

Internally, the TMS320C54xx uses a 1X master clock. The TMS320C54xx has a phase-locked loop frequency synthesizer that can be used to derive this master clock from a lower- or higher-frequency source. The external source can come from either an external clock generator or from an external crystal operating in conjunction with the processor's on-chip oscillator.

The TMS320C54xx phase-locked loop is either hardware configurable or software configurable, depending on the processor. The TMS320C541, TMS320LC542, TMS320LC543, TMS320LC545A, and TMS320LC546A use a hardware-configurable phase-locked loop. All newer TMS320C54xx family members, such as the TMS320VC5402, use a software-configurable phase-locked loop.

On the older TMS320C54xx family members that have an on-chip hardware-configurable phase-locked loop, three clock configuration pins define how the input clock source or crystal is treated. Additionally, the processors are available in two mask options that have slightly different clocking features. The input clock frequencies relative to the desired master clock frequency available under the different options are summarized in Table 7.13-4.

The newer TMS320C54xx family members add a software-configurable phase-locked loop that is software programmable in one of two configurations: either in divider mode where the input clock is divided by two or four, or in PLL mode where the input clock is multiplied by one of 31 possible ratios from 1/4 to 15. The processor is initially configured by the three clock configuration pins to either divide an internal or external clock source by two to generate the master clock, or to use the phase-locked loop to generate a master clock from an external clock at the same frequency.

On all TMS320C54xx processors the clock configuration pins also allow an external device to put the processor into a low-power state that is equivalent to executing the IDLE3 instruction.

Hardware Looping

The TMS320C54xx features both single- and multiple-instruction hardware loops via the RPT, RPTB, and RPTBD instructions. RPT repeats a single instruction from one to 65,536 times and is not interruptible.

RPTB and RPTBD repeat an arbitrary-length block of instructions from one to 65,536 times. Two registers are loaded with the start and end addresses of the block, and the RPTB and RPTBD instructions specify the number of iterations. The RPTBD instruction is a delayed version of the RPTB instruction: it executes the two instructions immediately following it only once, and then repeats the block of instructions following them. As a result, the RPTBD instruction executes in only two instruction cycles, as opposed to four for the RPTB instruction. Clearing the "block repeat active" flag in the ST1 status register allows code within a block repeat to conditionally terminate the repeat loop. The multi-instruction hardware loops are interruptible.

Clock Source	Mask Option	
	1	2
Externally Generated Clock	1/3, 1/2, 2/3, 1, 2	1/5, 2/9, 1/4, 1, 2
External Crystal	1/3, 2	1/5, 2

TABLE 7.13-4. TMS320LC542 and TMS320LC543 clocking options. The values shown in the table are the ratio of the input clock frequency to the master clock frequency.

The RPT instruction can be nested within an RPTB or RPTBD loop, but RPTB/RPTBD instructions cannot be nested within an RPTB/RPTBD loop without saving and restoring certain repeat registers.

> *The inability to nest hardware loops without explicit save and restore operations costs cycles in some algorithms, e.g., the fast Fourier transform. The delayed decrement-and-branch instruction used for software loops incurs less overhead than saving and restoring the multi-instruction hardware loop registers.*

Instructions that access data in both program memory and data memory (for example, instructions that move data between the two memory spaces) execute faster within a single-instruction RPT loop. This is because the instruction does not need to be fetched for each iteration through the loop, freeing the program bus to be used for data accesses.

Interrupts

The TMS320C54xx architecture allows for up to 16 maskable interrupts and 16 non-maskable interrupts. Not all of these interrupts are used on all family members. Non-maskable interrupts on current family members include reset, non-maskable interrupt, and software interrupts generated by the TRAP instruction. Maskable interrupts include receive and transmit interrupts for each serial port, a timer interrupt, and interrupts associated with four general-purpose interrupt lines.

Interrupts are prioritized but are not automatically nestable. That is, interrupts are only nestable if the interrupt service routine explicitly reenables interrupts. Each interrupt has its own four-word vector location. The spacing of the vector locations permits using a delayed branch to a main interrupt service routine followed by two instructions, which can be used to fetch data from a serial port, for example. The interrupt vectors can be remapped to the beginning of any 512-word page in memory by writing to the PMST configuration register.

On interrupt, the TMS320C54xx acknowledges the interrupt and disables interrupts. Only the program counter is saved onto the software stack; the interrupt service routine must save any other registers that it wishes to preserve.

Interrupt latency is 13 instruction cycles from the assertion of the external interrupt line to the execution of the first word of the interrupt vector if the processor is in an interruptible state. The first two words of the interrupt vector typically contain a delayed branch to the interrupt service routine. The two words following the delayed branch instruction in the interrupt vector are executed in the delay slots of the delayed branch. Thus, the latency from the assertion of the interrupt line to the execution of the first word following the delayed branch is 15 instruction cycles.

Stack

The TMS320C54xx features a software stack in data memory with a dedicated stack pointer. The stack is used for saving the return address for interrupts and subroutine calls. It can also be accessed via the PUSHD and POPD instructions, which push or pop a word to or from data memory onto or off of the stack. Because TMS320C54xx registers are also accessible as memory locations, the PUSHD and POPD instructions can also be used to move registers to and from the stack.

The TMS320C54xx provides an instruction (FRAME) that adds an offset to the stack pointer, and allows stack pointer-relative addressing. This can be used to ensure stack alignment to an even address, which is required for correct operation of subsequent double word load and store operations to the stack.

> *The TMS320C54xx FRAME instruction and stack pointer-relative addressing mode are particularly useful for C compilers where parameters are passed to subroutines via the stack.*

Bootstrap Loading

TMS320C54xx processors provide several mechanisms for bootstrap loading. Shortly after reset, the processor performs a read from I/O port address OxFFFF and uses the lower eight bits of the data read to determine the bootstrap mode. Most TMS320C54xx processors support bootstrap loading over the serial ports (using 8- or 16-bit words in several formats), over the parallel port (using 8- or 16-bit words), over I/O space (using 8- or 16-bit words), and from 8- or 16-bit-wide EPROM. Many of the processors, such as the TMS320VC5402, TMS320VC5409, TMS320VC5410, and TMS320VC5416, support bootstrap loading over the host port as well. The TMS320VC5409 and TMS320VC5416 also support serial bootloading from an SPI serial EEPROM.

The dual-core TMS320VC5420 allows either a host-controlled boot over the host port interface (HPI) or a stand-alone boot. In stand-alone boot mode, either both cores boot the same program from external memory simultaneously, or in sequential mode, the master core boots a program from external memory and then releases control to the slave core allowing it to boot.

The TMS320VC5421 supports the same host-controlled HPI booting as the TMS320VC5420, but supports different stand-alone boot modes, including 8/16-bit parallel EPROM booting and an 8-bit serial boot from the serial ports.

The four-core TMS320VC5441 can only be boot loaded using the same host-controlled HPI booting available on the TMS320VC5420 and TMS320VC5421. Serial boot loading is not supported, nor is boot loading from external memories since the TMS320VC5441 does not extend the memory address and data buses off chip.

Peripherals

TMS320C54xx on-chip peripherals include a timer and combinations of synchronous serial ports, time-division multiplexed serial ports, buffered serial ports, multi-channel buffered serial ports, and host ports. Parallel I/O is also available on all processors except the TMS320VC5441 through the external memory interface. Refer back to

Table 7.13-1 for combinations of serial ports and host ports on different members of the TMS320C54xx family.

- **Serial ports**

  Synchronous serial ports are like those found on TMS320C5x processors and support 8- or 16-bit words. When using an internally generated bit clock, they can run at bit rates up to the instruction rate divided by four (e.g., 25 Mbits/second for a 100 MHz processor); when using an externally generated bit clock, they can run at bit rates up to the instruction rate divided by three. The transmit and receive portions of the serial port are independent and have their own clock, frame sync, and data pins. The receive clock is always externally supplied, while the transmit clock can be either internally generated, with frequency equal to the instruction rate divided by four, or externally supplied. The clock pins can be used as general-purpose inputs if the serial port is not in use.

  The buffered serial port is divided into two parts: a synchronous serial port interface and an "auto buffering unit":

  - The serial port interface is a synchronous serial interface that transfers 8-, 10-, 12-, or 16-bit words at a maximum bit rate equal to the processor's master clock rate for devices operating at up to 50 MHz. For faster devices, the maximum operating frequency is 50 Mbits/second. The transmit and receive portions of the serial port are independent and have their own clock, frame sync, and data pins. The receive clock is always externally supplied, while the transmit clock can be either internally generated (with frequency equal to the master clock rate divided by a five-bit selectable divisor) or externally supplied. The clock pins can be used as general-purpose inputs if the serial port is not in use. Transmit clock and frame sync polarity can be selected by the user.
  - The auto buffering unit (ABU) is associated with a 2-Kword block of on-chip dual-access RAM. It allows up to 2,048 words to be transferred between main memory and the serial ports via a form of DMA before the processor is interrupted. For both serial port data transmission and reception, the auto buffering unit allows the programmer to specify a buffer size (up to 2,048 words) and an 11-bit starting memory address within the 2 Kword memory block. The auto buffering unit performs transfers between the serial port and memory automatically and independent of the processor, interrupting the processor when the buffer is empty, half full, or entirely full. If the CPU and the ABU attempt to access the 2-Kword buffer memory block simultaneously, the CPU is delayed access by one cycle. For newer processors such as the TMS320VC5410 and TMS320VC5421, the ABU buffer may be placed anywhere in memory. For older TMS320C54xx devices, such as the TMS320LC545A, the ABU buffer is fixed in location and maximum size.

The auto buffering unit will be quite valuable in some applications, significantly reducing the amount of time the processor must spend servicing serial I/O.

The time-division multiplexing (TDM) serial port is identical to the TDM serial port on TMS320C5x processors and supports up to eight transmit slots per frame. Please refer to the section discussing the TMS320C5x for details on the TDM serial port.

The multi-channel buffered serial port (referred to as the "McBSP" by Texas Instruments) offered on the TMS320VC5410 is similar to the buffered serial port available on other TMS320C54xx devices, and supports up to 128 channels per port. The McBSP is intended for connection to T1/E1 framers, as well as other high-speed serial devices. The McBSP on the TMS320VC5410 and other newer TMS320C54xx family members shares the enhanced DMA features of that part, allowing serial data to be spooled to or from any point in the address space. In addition, the McBSP supports transparent A-law and μ-law compression and decompression in hardware.

- **Host Port Interface**

The TMS320LC542, the TMS320LC545A, the TMS320LC549, and TMS320VC5410 provide an 8-bit parallel host port interface (HPI). The HPI allows a host processor to read and write a 1-Kword region of the DSP's on-chip RAM. The host processor specifies (one byte at a time) the starting address, and, if writing, the 16-bit data value to write. On subsequent reads and writes, the addresses can be updated automatically. In normal mode, both the host and the DSP can access this memory. If the DSP agrees to not access the memory for a period of time, the host is given higher-bandwidth access to the memory. Both the host and the DSP can interrupt one another through the HPI.

The host port interface (HPI) supports both non-multiplexed or separate address and data bus interface which is common on most microprocessors. It also supports a multiplexed address and data bus interface where the address is latched first followed by a data read or write. Multiplexed interfaces are common on 8051 family microcontrollers and other 8-bit microcontroller functions.

The newer, single core processors, such as the TMS320VC5409, have a 16-bit parallel host port interface, but also support an 8-bit HPI for interface to lower cost 8-bit microcontrollers. The multiple-core devices, the TMS320VC5420, TMS320VC5421, and TMS320VC5441, only support the 16-bit HPI interface.

- **Timer**

Most members of the TMS320C54xx family include a timer with 20-bit resolution that uses the processor's master clock signal as its clock source. A prescaler divides the master clock by a programmable 4-bit count. A programmable 16-bit counter register uses the resulting frequency as its input clock. The values in both the prescaler and counter are readable under software control. The timer generates an interrupt and pulses the TOUT pin when a counter reaches zero, at which point the counter is reloaded with the user-specified value.

Each core on the TMS320VC5441 has two timers with 32-bit resolution; a general-purpose timer and a watchdog timer. The general-purpose timer has programmable 16-bit prescaler and 16-bit count registers. A maskable interrupt, TINT, is generated when the counter register reaches zero. The timer can be started, stopped, reset, and reloaded via the software-controllable timer control register. The timeout period of the watchdog timer is controlled by a 16-bit count register and a 16-bit period register. Under normal system operation, the watchdog timer is periodically reset before the timeout period expires. When the watchdog timer reaches zero, the watchdog output flag is asserted; this flag can be connected, via a GPIO (general-purpose I/O) pin to the NMI (non-maskable interrupt) input to restart the system in the case of a watchdog timeout event.

Each core on the TMS320VC5441 has two timers; a general-purpose timer and a watchdog timer. Both timers have 32-bit resolution since the prescaler divides the master clock by an amount determined by a programmable 4-bit field. Each 4-bit value yields a prescale count equal to $2^{(n+1)}$-1. The watchdog timer prevents the system from locking up and must be accessed periodically to prevent it from decrementing to zero. In the case when the watchdog decrements to zero, if enabled, a maskable interrupt is triggered. A general-purpose I/O (GPIO) pin can be configured to pulse low, thus notifying the system of the watchdog event.

- **Bit I/O**

  Bit I/O is provided by two pins. One of these (BIO) is an input and the other (XF) is an output. The state of the BIO pin can be used in conditional branches, while the XF output pin can be set or cleared by the SSBX and RXBX instructions.

  The multiple-core devices, the TMS320VC5420, TMS320VC5421, and TMS320VC5441, each have four GPIO pins and an XF output pin for each core. One GPIO pin functions as the ROM enable input on reset, but afterward can be programmed as an input or output. The second pin can be programmed as input or output with no alternate function. A third GPIO pin, when programmed as an input, is compatible with the BIO pin. A fourth GPIO pin can function as the timer output, pulsing when the timer resets.

- **Parallel I/O**

  TMS320C54xx processors also provide a general parallel I/O mechanism via I/O memory space. Essentially, I/O space operations function like normal external bus operation, with the I/O port address on the address pins and data on the data pins, but they also assert a special I/O strobe pin to indicate to external devices that an I/O operation is occurring. Unlike regular external bus cycles, I/O reads always take a minimum of two instruction cycles to complete. The I/O address space is 16 bits, permitting 65,536 logical ports to be accessed via PORTR and PORTW instructions. The four core TMS320VC5441 is the only device that does not support I/O memory expansion because the device does not extend the address and data buses off-chip.

On-Chip Debugging Support

The TMS320C54xx offers a JTAG-based interface to on-chip emulation and debugging circuitry. Through the JTAG interface, an external device can read and write processor memory and registers, set and clear breakpoints, and single-step through a program. The JTAG port can also be used for boundary scan.

Power Consumption and Management

The TMS320C54xx provides three low-power modes via the IDLE1, IDLE2, and IDLE3 instructions. In all idle modes, the clock is turned off to the processor's core, reducing power consumption. In IDLE1 mode, on-chip peripherals (the serial port and timer) and interrupt lines remain active, and any unmasked interrupt, e.g., if the auto-buffered serial port receive buffer is full, wakes the processor. In IDLE2 mode, the on-chip peripherals are turned off, and only an interrupt on an external interrupt line wakes the processor. IDLE3 mode is similar to IDLE2 mode but it also turns off the on-chip crystal oscillator and PLL circuitry. As a result, wake-up from IDLE3 mode can require up to 2,053 clock cycles for the PLL to recover.

As an additional power management feature, the output clock and the internal clock to the external interfaces can be turned off if they are not needed.

The newer TMS320C54xx family members use dual supply voltages as shown in Table 7.13-1. For example, the TMS320VC549 uses a 2.5-volt supply and the external interface uses 3.3 volts. The lower core supply lets the processor operate at 100 MHz with a typical power consumption in normal operation of 125 mW, according to Texas Instruments. Newer, lower-voltage processors such as the TMS320UVC5402 operate from a 1.2-volt internal supply and from a range of external supply voltages of 1.2 volts to 2.75 volts. Typical power consumption for some TMS320C54xx family members are listed in Table 7.13-5.

Benchmark Performance

The TMS320C54xx has been benchmarked with the BDTI Benchmarks™. Overall benchmark results for all benchmarked processors are presented in Chapter 8, *BDTI Benchmark™ Results*. We summarize and analyze TMS320C54xx benchmark performance in the paragraphs below. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not reflect the processor's instruction cycle rate; therefore, lower instruction cycle counts imply a more powerful architecture, but do not imply faster speed. Next we discuss benchmark execution times and cost-execution time products, indicating processor speed and cost-performance, respectively. We then discuss the processor's energy consumption, which reflects the energy consumed by the processor in order to perform a task. Finally, we discuss the processor's memory usage. We divide the memory usage discussion into

Processor	Speed (MHz)	Voltage (V)	Typical Power Consumption
TMS320C541	40	5.0	200 mW
TMS320LC541B	66	3.3	80 mW
TMS320C542	40	5.0	200 mW
TMS320LC542	50	3.3	165 mW
TMS320LC543	50	3.3	165 mW
TMS320LC545A	66	3.3	131 mW
TMS320LC546A	66	3.3	131 mW
TMS320LC549	80	3.3	106 mW
TMS320VC549	120	2.5/3.3	135 mW
TMS320VC5410	120	2.5/33	144 mW
TMS320VC5402	100	1.8/3.3	60 mW
TMS320UC5402	80	1.8/1.8-3.6	50 mW
TMS320UVC5402	30	1.2/1.2-2.75	12 mW
TMS320VC5409	100	1.8/3.3	72 mW
TMS320UC5409	80	1.8/1.8-3.6	50 mW
TMS320UVC5409	30	1.2/1.2-2.75	12 mW
TMS320VC5420	100	1.8/3.3	266 mW
TMS320VC5416[a]	160	1.5/3.3	90 mW
TMS320VC5421	100	1.8/3.3	160 mW
TMS320VC5441[a]	133	1.5/3.3	550 mW

TABLE 7.13-5. TMS320C54xx family typical power consumption, according to Texas Instruments.

[a] Not available as of this writing.

three parts: Control benchmark memory usage, overall benchmark program memory usage, and benchmark data memory usage.

Execution Performance

- **Instruction cycle counts:** As illustrated in Figure 8.1-13 in Chapter 8, *BDTI Benchmark™ Results*, the TMS320C54xx's total normalized instruction cycle count is approximately 25% higher than the average for all benchmarked processors. Although the TMS320C54xx's has a dual-accumulator data path, single-cycle MAC instructions, and parallel memory move instructions, it has higher cycle counts compared to newer DSP architectures that have more execution units and hence higher levels of parallelism.

  The TMS320C54xx has a fairly low instruction cycle count on the **Two-Biquad IIR filter** benchmark—approximately 15% lower than the average for all benchmarked processors. The TMS320C54xx is able to use only five instruction cycles per biquad due to support for multiply-accumulate operations using two data memory operands in a single instruction, parallel memory move support, and zero-overhead hardware looping.

  The TMS320C54xx also has a fairly low instruction cycle count on the **Viterbi** benchmark, at approximately 25% lower than the average for all benchmarked processors. The TMS320C54xx has a specialized single-cycle add-compare-select instruction that reduces the cycles required for the first stage of this benchmark. However, compared to other processors benchmarked that also have specialized Viterbi decoding instructions or have higher levels of parallelism, the TMS320C54xx has a cycle count that is approximately 40% higher than average.

  The TMS320C54xx has a high instruction cycle count on the **Control** benchmark. However, the Control benchmark is optimized for minimum memory usage, not for minimum instruction cycles. The TMS320C54xx instruction cycle count on this benchmark is about 85% above average for all benchmarked processors. The fact that the TMS320C54xx does not support short immediate data or PC-relative branches increases its instruction cycle count (and also program memory usage) on this benchmark.

  The Texas Instruments TMS320C54xx has the highest cycle count for the **Bit Unpack** benchmark, about 30% higher than that of typical conventional DSPs benchmarked here. Compared to the other conventional DSPs benchmarked, the TMS320C54xx lacks flexible bit-field manipulation capabilities and thus must perform a larger number of discrete shift-and-logical-operation steps for each bit-field extraction. The TMS320C54xx also has limited register-to-register move support; hence, loading the shift control register takes more cycles than in competing processors that have more flexible register move support. Limited support for conditional instruction execution and operand-unrelated parallel moves also contribute to the high cycle count.

- **Execution times:** The TMS320VC5416's 160 MHz instruction cycle rate combined with its higher-than-average instruction cycle counts yields a total normalized execution time that is approximately 10% below the average of the fixed-point processors benchmarked, as presented in Figure 8.2-13. However, the TMS320VC5416 is almost four times slower than the fastest processor, the Texas Instruments TMS320C64xx. At 160 MHz, the TMS320C54xx has a BDTImark2000 score of 500.
- **Cost-execution time:** The TMS320VC5416's better-than-average execution time performance coupled with its relatively high price of $33.50 (quantity 10,000) give it fifth place among all benchmarked fixed-point processors in terms of total normalized cost-execution time product. As shown in Figure 8.3-13, the total normalized cost-execution time product for the TMS320C54xx is roughly 15% higher than the average for all benchmarked fixed-point processors.
- **Energy consumption:** As illustrated in Figure 8.4-13B, the 1.5-volt TMS320VC5416 has the third-lowest energy consumption of all benchmarked DSP processors, trailing only the Motorola MSC8101 and the Texas Instruments TMS320C5510. The TMS320VC5416's total normalized energy consumption is about one-half of the average for benchmarked fixed-point processors.

Memory Usage

The focus in the memory usage analysis is on Control benchmark memory usage. Unlike other benchmarks, the Control benchmark is optimized for minimum memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP code. Finally we discuss constant and non-constant data memory usage.

- **Control benchmark memory usage:** The TMS320C54xx's total memory usage on the Control benchmark is the third-highest of the fixed-point processors benchmarked, and is average compared to all of the processors benchmarked, as presented in Figure 8.5-9A. The TMS320C54xx lacks PC-relative branches and support for short immediate data. Branch addresses and short immediate data must be encoded using an extra 16-bit instruction word, increasing memory usage on the Control benchmark.
- **Program memory usage:** The total normalized program memory usage for the TMS320C54xx is about 20% lower than average for benchmarked fixed-point DSP processors with 16-bit instructions. Program memory usage results are presented in Figure 8.5-13.

  The TMS320C54xx's program memory usage is the lowest of the fixed-point processors benchmarked on the real-block FIR filter benchmark. The TMS320C54xx has specialized instruction support for calculating the output of FIR filters, allowing the output samples to be calculated in fewer instructions than other fixed-point processors with 16-bit instructions.

The program memory usage for the TMS320C54xx on the **Vector Dot Product** benchmark is the second-lowest of all processors benchmarked. Flexible multiply-accumulate instructions and parallel store/load and store/multiply instructions help the TMS320C54xx to achieve low program memory usage on this benchmark.

- **Data memory usage:** The TMS320C54xx's constant data memory usage is the lowest of all benchmarked 16-bit fixed-point DSP processors. Total normalized constant data memory usage is shown in Figure 8.5-14.

  The TMS320C54xx's non-constant data memory usage is slightly lower than the average for benchmarked 16-bit fixed-point DSP processors. Total normalized non-constant data memory usage is shown in Figure 8.5-15.

> *The TMS320C54xx combines higher-than-average instruction cycle counts with a relatively high instruction execution rate, resulting in lower than average execution time results on the BDTI Benchmarks. This, combined with the third-lowest power consumption among processors benchmarked, results in the third-lowest total normalized energy consumption of all benchmarked processors.*
>
> *The TMS320C54xx has good overall program and data memory usage, which is the result of a relatively powerful data path and a 16-bit instruction word size.*

Cost

Price and packaging options for TMS320C54xx processors are shown in Table 7.13-6.

Fabrication Details

TMS320C54xx family members use a variety of fabrication processes, ranging from a 0.55 µm three-metal-layer CMOS process for the 5.0-volt TMS320C541, to a 0.15 µm four-layer-metal CMOS process for the single-core TMS320VC5416, dual-core TMS320VC5421, and four-core TMS320VC5441.

The 80 MHz, 1.8 volt TMS320UC5402 and TMS320UC5409 are fabricated in a 0.18 µm four-layer-metal CMOS process. The 30 MHz, 1.2 volt TMS320UVC5402 and TMS320UVC5409 are fabricated in a 0.15 µm four-layer-metal CMOS process.

The TMS320C54xx is available as a core for use in SoCs designed by customers and fabricated by Texas Instruments.

Part	Speed (MHz)	Voltage (V)	Package	Price (Qty. 10,000)
TMS320C541	40	5.0	100 TQFP	$14.98
TMS320LC541	66	3.3	100 TQFP	$8.37
TMS320C542	40	5.0	144 TQFP	$18.47
TMS320LC542	50	3.3	128, 144 TQFP	$20.32
TMS320LC543	50	3.3	100 TQFP	$19.41
TMS320LC545A	66	3.3	128, 144 TQFP	$16.66
TMS320LC546A	66	3.3	100 TQFP	$15.43
TMS320LC549	80	3.3	144 TQFP 144 BGA	$19.43
TMS320VC549	120	2.5/3.3	144 TQFP 144 BGA	$25.65
TMS320VC5402	100	1.8/3.3		$5.42
TMS320UC5402	80	1.8/1.8-3.6		$6.32
TMS320UVC5402[a]	30	1.2/1.2-2.75		$6.85
TMS320VC5409	120	1.8/3.3		$11.93
TMS320UC5409	80	1.8/1.8-3.6		$14.31
TMS320UVC5409[a]	30	1.2/1.2-2.75		$15.50
TMS320VC5410	120	2.5/3.3	144 TQFP 176 BGA	$32.24
TMS320VC5416	160	1.5/3.3	144 BGA 144 TQFP	$33.50
TMS320VC5420	100	1.8/3.3	144 BGA 144 TQFP	$56.41
TMS320VC5421	100	1.8/3.3		$84.21
TMS320VC5441[a]	133	1.5/3.3	179 BGA 176 TQFP	$168.43

TABLE 7.13-6. TMS320C54xx price and package summary. Pricing data is provided by Texas Instruments as of June 2000.

a. Not available as of this writing.

Development Tools

Texas instruments provides a COFF-based assembler, linker, archiver, instruction-set simulator, emulation debugger, and an optimizing ANSI C compiler for the TMS320C54xx. The tools run on IBM PC-compatible computers under Microsoft Windows 95 or higher.

Texas Instruments bundles the tools in a package called "Code Composer Studio" (CCS), a suite of software development tools. Code Composer Studio is an integrated development environment that integrates the assembler, C compiler, linker, and simulator or emulator debugger. CCS's support for JTAG-based emulation includes debugger support of graphical data displays (e.g., an FFT "waterfall"). The debugger can be used for debugging individual devices or multiple devices in parallel. Source-level debugging capabilities include breakpoints, watches, and function-profiling, all of which are fully windows-based. Code Composer Studio also supports a scripting language which allows automation of simple tasks. Code Composer Studio is available for Windows 95/98 and Windows NT.

Texas Instruments provides a "TMS320C54xx DSKplus" development board based on the TMS320VC5402. The accompanying software includes a true Microsoft Windows-based debugger interface. An evaluation module based on the TMS320VC549 and TMS320VC5410 device is also available as a plug-in card for IBM PC compatible systems.

The XDS510 emulator can be used with the TMS320C54xx with appropriate host software. There is a large variety of third-party support for the TMS320C54xx in the form of development boards, emulators, and function and application software libraries.

Texas Instruments and TechOnline, Inc. provide a "VirtuaLab" where registered users may allocate time and compile or upload code to an on-line TMS320C54xx development board at the TechOnline, Inc. World Wide Web home page via the Internet.

A wide variety of third-party vendors offer tools and DSP software libraries for the TMS320C54xx.

> *The number of third-party vendors providing tools and software for Texas Instruments' DSPs is the largest of any DSP processor vendor. This is due partly to Texas Instruments' early entry into the DSP processor market and to its strong efforts to cultivate third-party support.*

Applications Support

The primary documentation for the TMS320C54xx family is the four-volume *TMS320C54xx Reference Set.* This manual covers the architecture, instruction set, and programming of the device. Separate data sheets discuss the hardware aspects of the processors.

Applications support for all TMS320 family processors is provided by staff who are available via telephone hotline, fax, and Internet electronic mail.

Texas Instruments also provides a computer bulletin board system that allows TMS320 users to download code and brief application notes. Internet users can access copies of bulletin board files via anonymous FTP and via the World Wide Web.

Advantages

- Rich instruction set
- Conditional instruction execution
- Specialized instructions for LMS, symmetrical FIR filter, polynomial evaluation, exponent detect, and absolute or squared difference
- Barrel shifter
- Support for dual 16-bit operations; support for multi-precision arithmetic
- Special function unit in data path for Viterbi decoding
- Many address registers
- High per-cycle memory bandwidth
- Flexible bootstrap modes
- Software stack with frame pointer and indexed addressing supports HLL compilers
- Three low-power modes; low-voltage versions have low power consumption
- Two or more serial ports; autobuffering serial port on some family members; TDM serial port on some family members; enhanced multi-channel serial ports on some family members; enhanced DMA on some family members
- Uses 1X or slower external clock; flexible PLL (software configurable on some devices)
- JTAG emulation port with boundary scan
- Large number of family variants including two and four core versions
- Core version available
- Good third-party tool and library support
- Good energy consumption results on the BDTI Benchmarks

Disadvantages

- Exposed pipeline
- Only one circular buffer size can be used at a time
- Only one modifier register (AR0)
- Writes to external memory take at least two instruction cycles

7.14 Texas Instruments TMS320C55xx Family

BDTImark2000 Score: Not Available

Introduction

The TMS320C55xx is a very long instruction word (VLIW), 16-bit fixed-point DSP processor family from Texas Instruments, announced in February 2000. The TMS320C55xx can execute up to two instructions in parallel, with instruction widths varying from 8 to 48 bits. Instruction widths depend on the number of arguments, bytes of immediate data, and number of parallel operations. The TMS320C55xx is based on Texas Instruments' earlier fixed-instruction-length TMS320C54xx family, but adds significant enhancements to the architecture and instruction set. The TMS320C55xx is assembly source-code upward compatible with the TMS320C54xx; the TMS320C55xx can execute assembly code written for the TMS320C54xx, but not object code.

Target applications for the TMS320C55xx include traditional DSP applications, such as cellular telephones and modems; and telecom infrastructure applications, such as voice over IP gateways and multi-channel modem banks. The TMS320C55xx interfaces directly to SDRAM, making it well suited for use in portable consumer products where large memory buffers are required; e.g., digital cameras and portable digital audio players.

The first device based on the TMS320C55xx core, the TMS320C5510 (Table 7.14-1), is currently sampling at 160 MHz using a 1.6-volt supply, according to Texas Instruments. The 160 MHz device is slated for production in the fourth quarter of 2000. BDTI has executed the BDTI Benchmarks on a TMS320C55xx development board, but was unable to verify the projected clock speed of 160 MHz (the development board executed at a much lower clock rate). Hence, the BDTImark2000 score for this processor is not available pending verification of the clock speed. Check BDTI's website (*www.BDTI.com*) for the latest BDTImark2000 scores. A 200 MHz device fabricated using a copper process is planned for the first quarter of 2001, according to Texas Instruments. The TMS320C55xx supports two multiply-accumulate (MAC) operations per instruction cycle; peak MAC throughput is 320 million MACs per second at 160 MHz.

TMS320C55xx Version	Max. Speed (MHz)	On-Core Memory			
		Program and Data ROM	Single Access RAM (SARAM)	Dual Access RAM (DARAM)	Instruction Cache
TMS320C5510	160	16 K x 16	128 K x 16	32 K x 16	24 K x 8
TMS320C5510	200	16 K x 16	128 K x 16	32 K x 16	24 K x 8

TABLE 7.14-1. TMS320C55xx family summary. The memory configuration of each devices includes on-chip ROM, dual-access RAM (DARAM), single-access RAM (SARAM), and an instruction cache that stores the most recent instructions accessed from external memory.

> *Support for variable-length instructions allows the TMS320C55xx to use complex, multi-operation instructions for signal processing tasks while achieving compact code density for control tasks. On many tasks the TMS320C55xx is faster and requires fewer cycles than older architectures such as the Motorola DSP563xx, Texas Instruments TMS320C54xx, and Analog Devices ADSP-218x. It is comparable in many respects to recent dual-MAC architectures such as the Lucent DSP16xxx and Infineon TriCore.*

Architecture

The TMS320C55xx is a VLIW architecture, executing up to two instructions in parallel per instruction cycle. Instructions are scheduled for parallel execution at compile-time by the assembly programmer or code-generation tool. Unlike most VLIW architectures, the TMS320C55xx does not use simple, RISC-like instructions, however; the instruction set supports multiple parallel operations within a single instruction (for example, a single instruction can specify two MAC operations). A more detailed discussion of the processor's instruction execution model is contained in the *Instruction Set* section of this chapter.

The TMS320C55xx architecture consists of a 16-bit fixed-point data path, an address unit, a program flow control unit, and an instruction buffer unit. The TMS320C5510 architecture is illustrated in Figure 7.14-1.

The 16-bit fixed-point data path includes, among other execution units, two 17 × 17-bit multipliers (compared with one on the TMS320C54xx) with four 40-bit accumulators (compared with two on the TMS320C54xx). The data path supports special instructions designed to optimize implementation of FIR filters, echo cancellers, Viterbi decoders, and codebook search algorithms. Each MAC unit can multiply two data operands and generate a product per instruction cycle, but one operand is shared between the two multipliers.

> *Since one operand must be shared between the two multipliers, the processor's dual-multiply feature is limited to use in algorithms where one vector is shared, such when using the same filter on two channels of data. Other dual-MAC processors, such as the Lucent DSP16xxx and Infineon TriCore, can multiply two pairs of unique data operands, but can execute these computations at full speed only if pairs of data operands are fetched from contiguous memory locations.*

The address unit contains stack pointers and eight address registers capable of accessing data using several addressing modes. It is also capable of performing 16-bit ALU operations in parallel with computations performed in the main data path.

The program control unit handles all instruction flow control tasks such as branches, subroutine calls, returns from interrupts and subroutines, conditional instruction execution, and resolving pipeline conflicts.

Instruction lengths vary from 8 to 48 bits. 32 bits of instruction data is fetched from memory or the instruction cache in each instruction cycle. The instruction buffer unit stores up to 64 bytes of recently fetched instructions in the instruction buffer queue, parses the buffered instructions to determine where instructions start, and dispatches parsed instructions. When executing small loops, the instruction buffer queue retains the most recently executed instructions for reuse, thus avoiding the need to reload the instructions from memory.

Data Path

The TMS320C55xx fixed-point data path, called the "Data Computation Unit," contains a 40-bit ALU, a 40-bit barrel shifter, two 17 × 17-bit multiply-accumulate units, and four 40-bit accumulator registers. In addition, the TMS320C55xx address unit, designated the "Address-Data Flow Unit," contains a general-purpose 16-bit ALU that is inde-

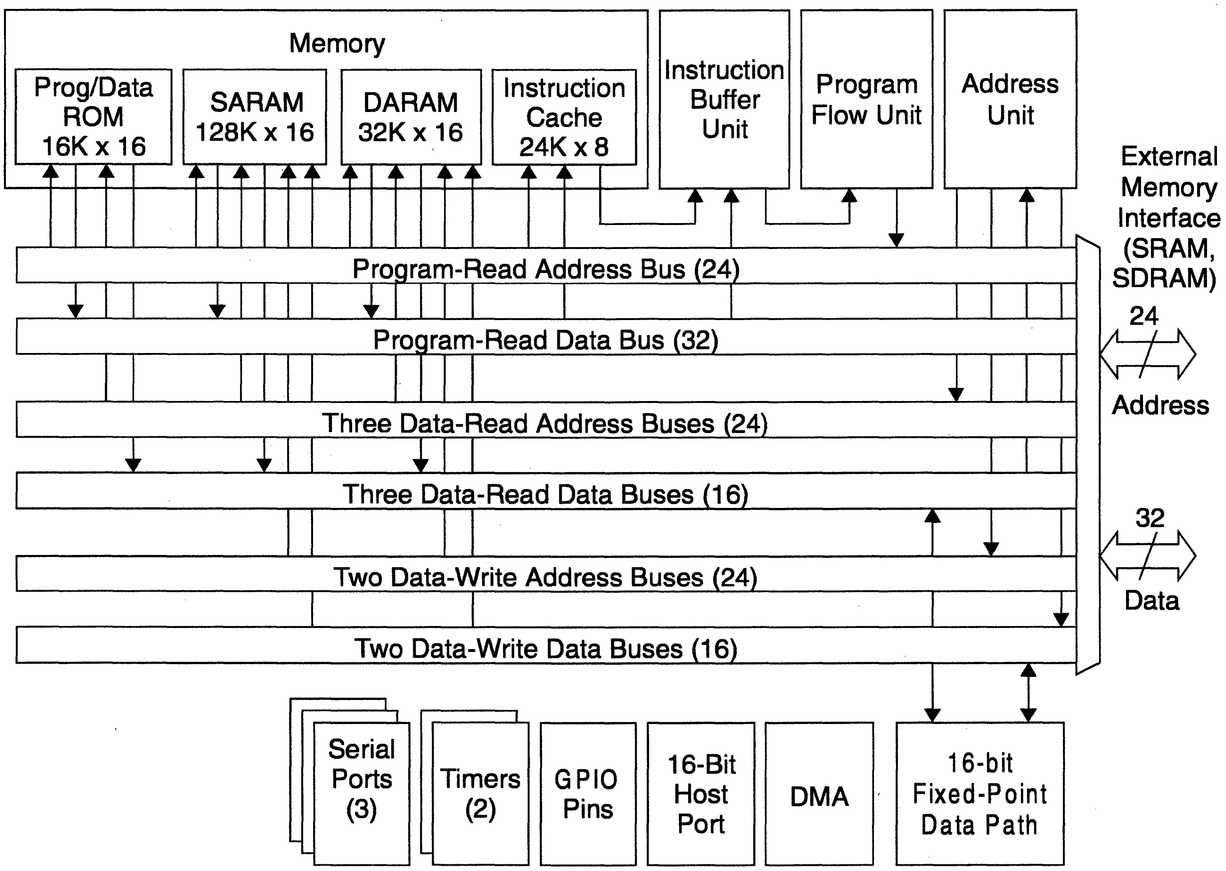

Memory
Prog/Data ROM 16K x 16
SARAM 128K x 16
DARAM 32K x 16
Instruction Cache 24K x 8
Instruction Buffer Unit
Program Flow Unit
Address Unit
External Memory Interface (SRAM, SDRAM)
24
Address
32
Data
Program-Read Address Bus (24)
Program-Read Data Bus (32)
Three Data-Read Address Buses (24)
Three Data-Read Data Buses (16)
Two Data-Write Address Buses (24)
Two Data-Write Data Buses (16)
Serial Ports (3)
Timers (2)
GPIO Pins
16-Bit Host Port
DMA
16-bit Fixed-Point Data Path

FIGURE 7.14-1. The TMS320C55xx architecture, as illustrated by the TMS320C5510.

pendent from the processor's three address generators and can be used in parallel with the main data path. The TMS320C55xx (like the TMS320C54xx) includes eight 16-bit "auxiliary registers," AR0-AR7. These registers are primarily used (with 7-bit extension registers) as address registers but, unlike on the TMS320C54xx, they can also be used as general-purpose registers in some cases.

The data path operates on data directly from memory when performing high throughput operations such as single-cycle, dual multiply-accumulates. This reduces the need for data registers to only those needed for storage of intermediate results. The instruction set allows multiplier, ALU, or shifter inputs to be sourced from the auxiliary registers or from the accumulators instead of from memory when necessary.

TMS320C55xx arithmetic ALU operations include addition, subtraction, absolute value, compare, negation, minimum, maximum, round, and saturate. Results from the 40-bit ALU may overflow into the upper 8 bits of the 40-bit accumulator register, or, if a saturation mode bit is enabled, results can be limited to a 32-bit range. In some cases of addition and subtraction instructions, the second data operand (accumulator, memory, or 16-bit constant) can be pre-shifted arithmetically up to 31 bits left or 32 bits right before computation; the shift amount is specified as an immediate value in the instruction or by the contents of one of the T registers.

TMS320C55xx logical operations include bitwise AND, OR, XOR, and complement. For some logical instructions, the second data operand (accumulator or constant) can be pre-shifted logically up to 31 bits left or 32 bits right before computation.

ALU operations execute in a single cycle unless a memory access incurs pipeline stalls or wait states. Most ALU instructions (ADD, SUB, NEG, MIN, MAX, AND, OR, NOT, XOR, and single-bit shifts) can be performed in either the 40-bit ALU or 16-bit ALU. Unlike the 40-bit ALU, the 16-bit ALU cannot use accumulators as destination registers, but can use the lower 16 bits of an accumulator as an input operand. The 40-bit ALU supports 16-bit and 40-bit input operands, which can come from a register or directly from memory. In addition to the 40-bit accumulator registers, the auxiliary registers and four temporary registers (T0-T3) in the address generation unit can be used as operand sources for the ALUs. Alternatively, one operand can be specified as immediate data if the other operand comes from an accumulator.

> *The TMS320C55xx is not a load/store architecture; the processor primarily operates on data directly from memory. This approach eases assembly programming and makes the processor more C compiler friendly since the compiler does not have to keep track of data registers beyond the four accumulators.*

Like the ALU in the TMS320C54xx, the TMS320C55xx 40-bit ALU supports dual parallel (SIMD) additions and subtractions by treating 32-bit memory operands as pairs of 16-bit words. (The 16-bit ALU does not support SIMD operations.) For SIMD operations, the ALU operates on a 32-bit data operand from memory and/or a 40-bit data operand from one of the accumulators. In this mode, the ALU computes two additions, two subtractions, or one addition and one subtraction in parallel. The ALU supports dual 16-bit maximum and minimum operations; dual 16-bit maximum and minimum instructions operate on pairs of 16-bit data. One of the results is stored in the lower 16 bits of the accumulator, and the other is stored in the upper 24 bits of the accumulator, but is limited to 16 bits if the saturation mode bit is enabled. Two transition registers, TRN0 and TRN1, are updated to reflect the results of the two comparisons.

Dual 16-bit ALU capability is useful in many applications, including computation of path metrics in Viterbi decoding. This is evidenced by the processor's relatively low cycle counts on BDTI's Viterbi benchmark.

The TMS320C55xx also supports a maximum difference instruction that, when combined with a dual memory store, replaces two TMS320C54xx compare-select instructions.

A conditional subtract instruction is used for iterative division to obtain the result of a 32-bit dividend divided by a 16-bit divisor. The result is obtained by repeating the conditional subtract 16 times, yielding a result in 16 cycles.

The TMS320C55xx's two multiply-accumulate (MAC) units perform 17 × 17 → 40-bit multiplication. Multiplier results can be stored in an accumulator, added to an accumulator, or subtracted from an accumulator. Both MAC units can operate in parallel with single-cycle throughput and latency. The multiplier product can by automatically shifted left by one bit to support fractional multiplies via a mode-controlled option. Accumulator results can be limited to 32 bits in saturation mode, or can expand into the upper 8 bits of the 40-bit accumulator, depending on the setting of a mode bit.

The two multipliers share one operand input, taken directly from memory. The other multiplier inputs are unique to each multiplier, and can be accessed from memory using a single address register (AR0-AR7) with two address offsets or using two address registers, or can be sourced from one of the T registers in the address generation unit. Multiply and multiply-accumulate instructions have single-cycle latencies.

Combined with its 160 MHz instruction execution rate, the dual MAC units enable the TMS320C55xx to achieve 320 million multiply-accumulates per second. The TMS320C55xx's single-cycle, dual multiply-accumulate capability is comparable to that of the Lucent DSP16xxx family in tasks such as FIR filters, correlations, and multi-channel operations where two multipliers can readily share an operand. Other algorithms, such as BDTI's Vector Dot Product benchmark, may not be able to take advantage of a shared operand between multipliers, demoting the TMS320C55xx to single-MAC performance.

Multiplication operands can be signed/signed, signed/unsigned, or unsigned/unsigned. Multiplication operations include multiply, multiply-add, and multiply-subtract. The TMS320C55xx can optionally shift the multiplier product 16 bits right before adding to the accumulator, improving the efficiency of multi-precision operations by combining a multiplication and a shift into a single instruction.

> *The zero-overhead 16-bit right shift of multiplier products enables the TMS320C55xx to implement 32-bit multiplications where the least significant 16 bits of the product are dropped in three cycles. In contrast, the Analog Devices ADSP-218x family, for example, takes five cycles because separate shift instructions are required.*

The TMS320C55xx barrel shifter is capable of performing arithmetic and logical shifts of up to 31 bits left or up to 32 bits right in a single instruction cycle. Shift operations can optionally set the carry flag. All shift operations take the shift amount either from an immediate operand encoded in the instruction word or from one of the T registers. A single-bit left or right rotate instruction is also provided.

The TMS320C55xx barrel shifter supports single-cycle exponent detection of a value stored in an accumulator. The processor also supports single-cycle normalization, in which the processor calculates the exponent and normalizes the accumulator contents. The TMS320C55xx lacks support for single-cycle block exponent calculations, in which a block of data is scanned iteratively and the exponent of the largest-magnitude value is returned.

> *Block floating-point capability would make the TMS320C55xx more efficient in algorithms which use block floating-point arithmetic, such as some FFTs, speech coders (e.g., G.728), and state-space control algorithms.*

The TMS320C55xx barrel shifter supports single-bit and bit-field modification, extraction, and insertion, along with bit and bit-field test instructions. Bit-field modification is useful for parsing packed data structures and for graphics functions. Bit-field test operations are useful in network applications where a header must be found a bit stream. The barrel shifter's bit count instruction counts one bits in an accumulator and can be used for parity calculations.

> *The TMS320C55xx barrel shifter instruction set should facilitate efficient code whether programming in assembler or in C, assuming the C compiler takes advantage of these instructions. The TMS320C55xx's advantage over other processors is that the barrel shifter operates on data directly from memory while many other processors, such as the Analog Devices ADSP-218x, require that data operands first be loaded into registers.*

TMS320C55xx arithmetic and logical ALU operations, multiplier operations, and barrel shifter operations update carry and overflow (one for each accumulator) status bits that can be used for conditional branches, conditional returns, or the execute conditional instructions. The same instructions can directly test any accumulator, address, or temporary register for an arithmetic condition relative to zero (equal to zero, less than zero, not equal to zero, or greater than zero.).

> *Most processors have shared arithmetic status bits that contain the status generated by the last arithmetic instruction executed. The TMS320C55xx's approach of testing individual accumulators, auxiliary registers, and temporary (T) registers eliminates some extraneous register comparison instructions when testing a register's status relative to zero.*

Testing a register's status relative to zero is not always useful. TMS320C55xx compare instructions generate status in one of two auxiliary bits, TC1 and TC2, when comparing two registers or when comparing a memory location with an immediate data value. The TC bits are also set by instructions such as register minimum/maximum instructions and bit test instructions, and serve as the carry bit for some shift instructions. One or both TC bits can be used as conditions for conditional branches, conditional returns, and the execute conditional instructions. The execute conditional (XCC and XCCPART) instructions allow single-cycle conditional execution of any TMS320C55xx non-branch instruction such as arithmetic or data move instructions. Simpler instructions can be executed in parallel with XCC, yielding single-cycle conditional execution. More complex instructions must follow XCC, yielding two-cycle conditional execution.

> *Conditional execution of non-branch instructions optimizes search loops and other routines that require frequent decision making by eliminating the need for branches. This improves code execution time and, by improving code density, facilitates re-use of instructions from the 64-byte instruction buffer queue when executing loops.*

The TMS320C55xx accumulator can be set for a 32-bit or 40-bit overflow boundary. If the saturation mode bit is enabled and the accumulator overflow boundary is set to 32 bits, results of operations are limited to the largest positive or negative 32-bit value. To achieve bit-exact results for many of the ITU voice compression algorithms (e.g., G.723, G.729), 32-bit saturation must be enabled. If saturation is enabled and the accumulator overflow boundary is set to 40 bits, results are saturated upon overflow past the 40-bit boundary. If saturation mode is disabled or if 40-bit accumulator overflow boundaries are being used, an explicit instruction can be used to perform saturation on a 32-bit accumulator boundary.

Saturation is performed when data transfers to and from memory include a shift operation. When accumulator data is shifted left before a transfer to memory, if the result overflows and saturation mode is enabled, the data will be saturated to 32 bits or 40 bits depending the setting for the accumulator overflow boundary. If saturation mode is enabled, when data from memory is shifted before being loaded into an accumulator and the shifted data exceeds the accumulator overflow boundary, the accumulator is loaded with a saturated data word. Data operands taken directly from memory and pre-shifted are saturated before input to a computation.

There is an explicit instruction to perform rounding between bits 15 and 16 of an accumulator. Alternatively, rounding can be optionally performed as part of a multiply, multiply-accumulate, or multiply-subtract instruction. Rounding can also be optionally performed on an accumulator when performing a memory read, memory store, or a saturation instruction.

> *Rounding and saturation incur no overhead since they can be performed in combination with arithmetic and load instructions.*

Two special finite impulse response (FIR) filter instructions accomplish FIR filter computation at a rate of two multiply-accumulates per instruction for FIR filters with symmetric coefficients. The instruction adds the delayed input samples that are equidistant from the left and right of the center tap(s), since these two samples share the same filter coefficient. On the next cycle, the summed samples are multiplied by the coefficient while the next two equidistant samples are summed. Another form of the instruction subtracts the equidistant taps before multiplying, implementing a coefficient array that is arithmetically inverted (anti-symmetrical) about the center tap(s).

> *In cases where filter coefficients are not symmetric (such as LMS filters where the coefficients change for each execution of the filter), a shared filter coefficient in a multiply-accumulate is not useful for achieving two multiply-accumulates per instruction cycle. Processors such as the Lucent DSP16xxx can read four data operands in parallel with dual multiply-accumulates, and so can perform dual multiply-accumulates in a wider range of applications.*

The TMS320C55xx includes special instructions that combine use of the data path ALU and multiplier-accumulator for efficient calculation of distances for Viterbi decoders. Also, an LMS instruction allows two-cycle least mean-squared (LMS) filter coefficient update.

> *The LMS instruction optimizes the coefficient update loop to two cycles, as on the TMS320C54xx. Many older DSP architectures, such as the Analog Devices ADSP-21xx, support two-cycle LMS filter coefficient update loops. Considering that many telephony applications developers desire for improved line echo canceller performance, it is surprising that Texas Instruments did not further optimize calculation of LMS filters to improve performance over that of the TMS320C54xx.*

Memory System

The TMS320C55xx memory system implements a modified Harvard architecture, with multiple memory blocks within a 16 Mbyte (24-bit address) unified memory map. Instruction and data accesses occur simultaneously using multiple address and data buses. Memory regions include on-chip dual-access RAM (DARAM), on-chip single-access RAM (SARAM), on-chip ROM, and external memory.

The TMS320C5510 uses the following memory configuration:

- 16K x 16 of on-chip program/data ROM (or external memory if disabled)
- 32K x 16 of on-chip dual access RAM
- 128K x 16 of on-chip single-access RAM
- 16032K x 8 of external memory space, with four chip selects each governing approximately 25% of the external memory.

The TMS320C55xx fetches instructions using a 24-bit program memory address bus and a 32-bit program memory data bus. The TMS320C55xx uses 8-, 16-, 24-, 32-, 40-, and 48-bit instructions. Internal memory can be treated as being 16 or 32 bits wide, and external memory can be up to 32 bits wide. The variable-length instructions stored in memory do not have to be aligned on memory word boundaries. The instruction buffer unit fetches instruction data from memory, stores the instruction data into the 64-byte instruction buffer queue, and parses the variable-length instructions from the instruction buffer queue for decoding.

As shown in Figure 7.14-1, the TMS320C55xx includes five unidirectional, on-chip data memory bus sets: three data read bus sets and two data write bus sets. Each bus set includes a 24-bit address bus and 16-bit data bus. The write buses support storage of two 16-bit words from separate accumulators or one 32-bit word from one accumulator. Therefore, maximum data write bandwidth is 320 million 16-bit words per second on a 160 MHz TMS320C5510. The TMS320C55xx can read up to three data operands per instruction cycle; thus, maximum on-chip data read bandwidth is 480 million 16-bit words per second at 160 MHz.

> *With three data read buses, the TMS320C55xx supports single-cycle dual two multiply-accumulates when the two multipliers share one operand. In contrast, for example, the dual-MAC Lucent DSP16xxx provides two 32-bit buses to read two pairs of contiguously located data operands.*

Care must be taken in mapping instructions and data to internal RAM in order to achieve single-cycle instruction execution. Internal DARAM and SARAM are divided into 4 Kword (8 Kbyte) blocks. To avoid stalls in the case where three data operands are required for dual multiply-accumulates, the programmer must be careful to arrange data in memory to avoid memory stalls; for example, by sourcing two data operands from the same DARAM block, the coefficient data operand from a different ROM, DARAM, or SARAM block, and the instruction from a different SARAM block.

The TMS320C55xx architecture requires a programmer to plan memory usage in order to ensure that vector data is placed in different areas of memory. Otherwise, memory conflict stalls are incurred due to too many accesses to the same memory block. An example of where this maybe a problem is an autocorrelation algorithm that uses both multipliers, where all data operands are from the same vector. To avoid memory conflict stalls in this case, a copy of the vector must be made before computation.

The TMS320C55xx supports DMA transfers in parallel with instruction-related data accesses. The DMA controller is discussed in the *Peripherals* subsection.

External Memory Interface

The TMS320C5510 multiplexes its internal program and data buses onto a single external memory interface consisting of a 24-bit address bus and a 32-bit data bus. The external memory interface provides signaling to access approximately 15.6 Mbytes partitioned into four areas, with the size of each area fixed at approximately 4 Mbytes. Each area is controlled by a separate chip select pin and can be configured for a different type of memory device: SDRAM, synchronous burst SRAM (SBSRAM), or asynchronous memory (SRAM, flash ROM, or parallel peripherals).

The TMS320C5510's large address space and support for low-cost memory devices such as SDRAM provides good flexibility for system designers and accommodates large programs.

Zero wait-state operation is supported if sufficiently fast memory devices are used. Wait states are programmed independently for each of the four external memory areas to support, for example, one area using a slower flash memory device and another area using fast SDRAM. The TMS320C5510 has an ARDY pin that functions as an asynchronous wait input, allowing an external device to halt the processor until requested data is available.

The TMS320C5510 is capable of reading one instruction, transferring up to three data operands between the CPU and memory, and performing two DMA transfers in a single instruction cycle. As long as only one of those memory transfers requires the external memory bus and the transfer does not require wait states, no penalty is incurred. When more than one of the transfers accesses off-chip memory, the TMS320C5510 prioritizes the requests and performs the transfers sequentially.

Data stored in external memory can be 16 or 32 bits wide. If an external memory bank is 32 bits wide, even and odd word addresses correspond to the lower and upper halves of a 32-bit memory location.

The TMS320C55xx supports memory read and write instructions to access 8-bit data located in 16-bit or 32-bit memory, but the instruction must explicitly specify the lower or upper byte of a 16-bit word. Since the TMS320C55xx supports 8-, 24-, and 40-bit instructions (among other sizes), instruction addresses can fall on any byte boundary. The processor does not support 8-bit external memory.

The TMS320C55xx allows other devices to take control of its external buses by asserting a bus hold pin; a corresponding hold acknowledge output pin indicates to the requestor that the bus is available. The TMS320C55xx continues program execution while the bus is held as long as all instructions and data operands are accessed from internal memory. The processor stalls when an instruction or data access to external memory is attempted and remains stalled until the external bus is released by the requestor.

Address Generation Units

The TMS320C55xx Address-Data Flow Unit contains three address generation units and eight 24-bit address registers, XAR0-XAR7. The Address-Data Flow Unit also contains a 16-bit ALU, which can be used for general-purpose ALU computations as described in the *Data Path* section. The XAR0-XAR7 registers are referred to as "extended auxiliary registers" since the lower 16 bits of these registers are actually the auxiliary registers discussed earlier, AR0-AR7. The upper 7 bits of the extended registers contain the data page, and are used to support the wider address space of the TMS320C55xx relative to that of the TMS320C54xx. AR0-AR7 are memory mapped registers; XAR0-XAR7 are not. The address generators also contain a number of other special-purpose addressing registers, such as circular buffer registers, as described below. Like the auxiliary registers, many of these registers concatenate 7-bit extension registers with 16-bit base registers to create 23-bit addresses.

The TMS320C55xx supports memory-direct and register-indirect addressing for accesses of 16-bit and 32-bit data. The upper of lower 8 bits of 16-bit location can be accessed by specifying a "high_byte" or "low_byte" qualifier in a data access instruction. The 8-bit data is read into or written from the lower 8 bits of a register regardless of whether the high or low byte of the memory location is accessed. Immediate data loads to registers are supported. Register-indirect addressing uses one of the eight 23-bit address registers (XAR0 through XAR7) and the 23-bit extended coefficient data pointer (XCDP) to read or write data operands. Register-indirect modification modes include pre- or post-modification by -1, 0, or +1 for 16-bit data accesses, or if performing a 32-bit data access, by -2, 0, or +2. Addresses can be pre- or post-modified by the contents of one of two 16-bit modifier registers, T0 and T1. Another mode allows an address register or the stack pointer to be indexed by an immediate constant. When performing an indexed access of data using a 16-bit immediate offset, the resultant address is not fed back into the address register, so that the address register can be reused as the base address of a structure or for stack data.

To allow more freedom in addressing modes without the use of wider instruction words, the operation of the address generators is controlled by a special mode bit, ARMS. When ARMS is cleared, the processor is in "DSP mode," and the addressing modes available include those commonly used in DSP algorithms, such as bit-reversed addressing.

When ARMS is set, the processor is in "Control mode," and the addressing modes available include those that are particularly useful for general-purpose CPU operations, such as indexed addressing with a short (3-bit) immediate index. Many addressing modes are available in both DSP and Control modes. The TMS320C55xx assembler provides macros to switch between modes in one cycle. The TMS320C55xx also provides an assembly directive to allow tracking the status of the ARMS bit, which generates a warning if an unavailable addressing mode is specified.

> *The ARMS bit increases the number of addressing mode without increasing the number of bits per instruction. The result can be interpreted as either more orthogonality in the instruction set and improves code density for control code. This approach forces the programmer to keep track of one more mode bit while programming, though increasing the chances of bugs during software development; however, the assembly directive provides some assistance in this regard.*

When performing dual multiply-accumulates, three data operands are accessed. The 16-bit coefficient data pointer (CDP) is used to access the operand common to both multipliers, while each multiplier uses one of eight auxiliary registers (AR0-AR7) to access a unique second data operand. When performing dual multiply-accumulates, the coefficient data pointer update options are limited compared to those supported for AR0-AR7. The coefficient data pointer (CDP) does not support pre-increment, pre-decrement, update with the temporary registers, or bit-reversed addressing. The coefficient data pointer can also be used for single multiply-accumulates or for individual data accesses.

> *When compared to similar DSP architectures, the TMS320C55xx has fewer address register update restrictions when performing a dual multiply-accumulate in parallel with access of multiple data operands. One significant limitation is that only the coefficient data pointer (CDP) can be used for the common data operand.*

The TMS320C55xx allows access of 32-bit data operands from memory with address register update to point to the next or previous 32-bit memory location when incrementing or decrementing the address register. 16-bit or 32-bit data can be transferred from memory to memory with single-cycle throughput. Two 16-bit data operands can be accessed using separate address registers and placed in the lower and upper 16-bit halves of an accumulator register for a subsequent dual 16-bit ALU operation. Likewise, results of a dual 16-bit ALU operation can be stored two separate areas in memory.

> *Transfer of 32-bit data words makes use of two of the TMS320C55xx's five internal 16-bit data buses, thus restricting execution of parallel instructions that require data transfers.*

The TMS320C55xx supports circular addressing. Each pair of address registers share a common buffer start address and two groups of four address registers share a buffer size. Each address register has a circular buffer enable bit so that one address register in a pair can be constrained to the circular buffer boundaries while the other accesses memory linearly. The coefficient data pointer has its own circular buffer start address and buffer size register. The maximum circular buffer length is 64K x 16. There are no restrictions on circular buffer start addresses.

> *The TMS320C55xx supports five circular buffers simultaneously, but four of the buffers are shared by two address registers. Another limitation is that the circular buffers pointed to by address registers AR0-AR3 must share the same buffer size. The same is true for address registers AR4-AR7. This is more flexible than the TMS320C54xx family, but still less flexible than the Analog Devices ADSP-218x and Motorola DSP563xx families in which each of the eight address registers have unique circular buffer control. However, the TMS320C55xx allows unrestricted placement of circular buffers. In contrast, many other DSP processors require circular buffers to be aligned on a 2^n-sized block boundary, where n=ceiling[$\log_2$(buffer size)].*

Any of the TMS320C55xx auxiliary registers (AR0-AR7) can be used for bit-reversed addressing. The coefficient data pointer, CDP, cannot be used for bit-reversed addressing. Bit-reversed addressing is specified in the instruction word. In this addressing mode, the lower 16 bits of the address register are output in reverse order and are post-incremented or decremented by the contents of the temporary register, T0. Bit-reversed addressing mode enables the processor to transparently perform the data scrambling required for radix-2 FFTs.

Memory-direct-like addressing is accomplished using a 16-bit immediate value as an address offset. There are two mutually exclusive modes of memory-direct addressing, which are controlled by the CPL mode bit. Accesses can be made relative to the 16-bit data page pointer (DP mode), or relative to the 16-bit stack pointer (SP mode). SP mode is required when executing code in the C run-time environment since it allows access of local function data from a local stack frame based on the stack pointer address.

> *The CPL mode bit is an another example of Texas Instruments packing more flexibility and parallelism into the TMS320C55xx instruction set at the expense of the programming complexity, in the form of mode bits.*

Pipeline

The TMS320C55xx uses a seven-stage execution pipeline consisting of stages for instruction decode, address generation, data access (two stages), reading data into the input registers, execution, and writing of results to registers or register contents to memory. This is deeper than the six-stage pipeline of the TMS320C54xx.

The pipeline is fully interlocked; that is, the processor inserts pipeline stalls to avoid unexpected results due to pipeline conflicts. Pipeline stalls are a common occurrence on the TMS320C55xx, but in many cases can be avoided with a thorough understanding of where in the pipeline instructions update addresses, load immediate data, access data from memory, execute arithmetic operations, and update arithmetic registers.

> *The fast clock speed of the TMS320C55xx is enabled partially by a deep pipeline. The TMS320C55xx pipeline can stall up to two cycles on arithmetic operations, but most conflicts can be avoided by a knowledgeable assembly language programmer or well-crafted C compiler.*

An example of where a pipeline stall can occur is when one instruction loads an address register and the next instruction uses the same address register as an address source. The example below loads AR1 with an immediate constant in the first instruction, an operation that is performed in pipeline stage six (the execute stage). The next instruction uses AR1 as an address source, but stalls for four cycles in pipeline stage two, the address phase, waiting for the first instruction to pass through pipeline stage six.

```
MOV #y16,AR1
; 4 pipeline stalls incurred
MOV *AR1+,AC0
```

One workaround to this common problem is to place non-conflicting instructions between the stall-inducing instructions. The instruction set provides another workaround to the programmer. In the example below, the first instruction instead loads AR1 with an immediate data value using the AMAR (modify address register) instruction. AMAR loads the address register in pipeline phase 2, the address phase. Since the next instruction completes the transfer in the same pipeline phase, it does not wait for the first instruction to complete, thus no pipeline stalls occur.

```
AMAR #y16,AR1
; no pipeline stalls
MOV *AR1+,AC0
```

Another case of where a pipeline stall will occur is when the first instruction transfers an address register to another address register or accumulator register and the next instruction uses the same address register as an address source, with an address register update. The first instruction transfers the address register in phase 5, the read phase. The next instruction uses the address register in phase 2, the address phase, resulting in a two-cycle pipeline stall waiting for the first instruction to start its phase 5. No workaround is available in this situation except for placement of two unrelated instructions between the stall-inducing instructions.

```
MOV AR1,AC1
; 2 pipeline stalls incurred
MOV *AR1+,AC0
```

When vectoring to interrupts, calling subroutines, branching, or returning from interrupts or subroutines, the TMS320C55xx discards instructions that have not completed the decode pipeline stage. Instructions that have been decoded finish execution, and are followed by instructions starting at the new program address. Conditional branches execute in four cycles if the branch is not taken, or in five cycles if the branch is taken. Unconditional branches execute in five cycles for relative addresses or three cycles for an immediate address. Subroutine calls and conditional returns have similar cycle counts. Unconditional returns from subroutines or interrupts execute in three cycles.

Unlike the TMS320C54xx, the TMS320C55xx does not support delayed branches. To maintain assembly source code compatibility, the TMS320C55xx mnemonic assembler relocates instructions that occupy delay slots in software written for the TMS320C54xx, placing these instruction before the branch.

> *In some cases, TMS320C54xx code that is reassembled and executed on the TMS320C55xx may require more cycles to execute on the TMS320C55xx, for several reasons: the TMS320C55xx does not support delayed branches, it has a deeper pipeline, and it has the potential for stalls in the instruction buffer queue. Since the TMS320C5510 executes at the same clock rate as the fastest members of the TMS320C54xx family, such a cycle count increase has the surprising effect of causing software written for the TMS320C54xx to run more slowly on the TMS320C55xx.*
>
> *Unlike some competitor DSP processors, the TMS320C55xx does not support delayed branches. Each instance of a branch results in up to four wasted cycles—cycles that could be occupied by useful instructions if delayed branching were supported. This is particularly problematic in control code (where branches are more common). Three features mitigate this disadvantage; first, the processor allows a compute and data transfer instruction to be combined with a branch, reducing the branch penalty. Second, the TMS320C55xx can avoid branching in some cases by using conditional instructions. And lastly, the* execute conditionally *(XC) instruction replaces the next instruction with a NOP if a condition is not true, thus avoiding the full branch penalty. In cases where only a single instruction needs to be executed conditionally.*

Instruction Set

The TMS320C55xx provides limited VLIW capabilities, executing up to two instructions in parallel in each instruction cycle. The TMS320C55xx instruction set is a superset of that of the TMS320C54xx, in many cases allowing two TMS320C54xx-like instructions to be executed in parallel. As a superset of the TMS320C54xx instruction set, the TMS320C55xx instruction set is not similar to the RISC-like instruction sets commonly seen on other VLIW DSP processors, such as the TMS320C62xx. TMS320C55xx instructions vary in size from 8 to 48 bits depending on the amount of parallelism and whether immediate operands are used. Many TMS320C55xx instructions specify two parallel operations in addition to parallel data moves.

The TMS320C55xx places a number of restrictions on which instructions can be executed in parallel, the most significant of which are:

- The combined width of two instructions executed in parallel cannot exceed 48 bits.
- Parallel instructions cannot load or store more data than is supported by the data buses in one cycle.
- Two instructions accessing the data or system stack cannot execute in parallel.

Only instructions which use a 16-bit immediate address or address register offset require 48 bits. Because of their width, these instructions cannot be executed in parallel with another instruction. Only program control instructions which use absolute 24-bit addresses require 40 bits. Only the NOP, MMAP, and reserved instructions for future co-processors require 8 bits. Therefore, the majority of common instructions require 16-32 bits. The TMS320C55xx instruction set and key registers are summarized in Table 7.14-2 and Table 7.14-3.

> *The TMS320C55xx instruction set thoroughly exercises the processor's hardware, providing single-cycle throughput for nearly every instruction. The processor supports dual 16-bit multiplications, dual 16-bit additions, and parallel use of both the 16-bit address unit and the 32/40-bit data path for arithmetic operations.*

Assembly Language Format

TMS320C55xx programmers have a choice of an algebraic or a mnemonic style assembly instruction syntax. Examples in this report use the mnemonic assembly language syntax.

Some instructions support up three data reads in parallel with arithmetic operation(s), as illustrated by the following instruction:

```
MAC *AR3+, *CDP+, AC0
:: MAC *AR4+, *CDP+, AC1
```

The above instruction performs two multiply-accumulate operations using three data operands: a common coefficient pointed to by address register CDP and a unique operand for each MAC unit pointed to by address registers AR3 and AR4. The two products are summed in accumulators AC0 and AC1. Address registers AR3, AR4, and CDP are each incremented after being used to read data from memory. This instruction has options (not shown) to designate multiplier inputs as unsigned, to specify accumulator saturation, and to specify accumulator rounding between bits 15 and 16.

The two colons (::) shown in this example indicate that the two multiply-accumulates are part of the same instruction rather than being two separate instructions. Texas Instruments refers to this type of parallelism as "implied parallelism." When two instructions are paired for parallel execution, the || notation is used (as TMS320C6xxx architectures). Pairing two instructions is referred to by Texas Instruments as "user parallelism."

Class	Instructions
Arithmetic	Absolute value, add, negate, subtract, add-subtract, minimum and maximum, dual 16-bit add, dual 16-bit subtract
Multiply-Accumulator	Single or dual multiply, multiply-add, or multiply-subtract (support any combination of signed and unsigned multiplier inputs, rounding, and saturation)
Logic	*And, exclusive-or, or, complement*
Shifting	Arithmetic/logical/rotate shift left/right by 0-31 bits, single-bit shift in addressing units
Conditional Execution	Conditional branch, conditionally execute next one instruction, conditionally execute parallel instruction, conditional shift, conditional subtract for iterative division, conditional add/subtract for Viterbi decoder algorithm
Comparison	Compare 16-bit, 32-bit or 40-bit registers with instruction-specified condition (>, >=, <, <=, ==, !=). Comparison of memory to register for == condition. Result is a single true or false status bit
Looping	Nested single- and multi-instruction hardware loops
Branching	Conditional relative branch, absolute branch, absolute branch on counter status, compare and branch
Subroutine Call	Conditional subroutine call, conditional return, return from interrupt, push, pop
Bit Manipulation	Bit-field extract, bit-field expand, bit-field ones count; register bit test, bit complement, bit set, bit clear; memory bit test, bit complement, bit set, bit clear
Special Arithmetic Functions	Division step, absolute distance, FIR filter two taps, LMS filter tap and update, round and saturate, square distance, exponent detect, normalize

TABLE 7.14-2. TMS320C55xx instruction set summary. Underlined instructions are new for the TMS320C55xx and are not supported on the TMS320C54xx.

In some cases, an instruction with implied parallelism can be paired with another instruction, thus combining implied and user parallelism.

Parallel Move Support

The TMS320C55xx supports up to three operand-related parallel data moves within a single instruction, such as in the example shown above. Three data operands are

Unit	Register(s)	Width	Description
Address Unit	AR0-AR7	16 bits	Auxiliary registers
	XAR0-XAR7	24 bits	Extended auxiliary registers
	T0-T3	16 bits	Temporary registers
	CDP	16 bits	Coefficient data page pointer
	DP	16 bits	Data page pointer
	SP	16 bits	Stack pointer
	PDP	7 bits	Peripheral data pointer
	SSP	16 bits	System stack pointer
	BK03, BK47, BKC[a]	16 bits	Circular buffer size registers
	BSA01, BSA23, BSA45, BSA67, BSAC[a]	16 bits	Circular buffer start address registers
Data Path	AC0-AC3[a]	40 bits	Accumulator registers
Control Unit	PC	24 bits	Program counter
	RETA	24 bits	Return address register
	BRC0, BRC1[a]	16 bits	Block repeat counters
	BRS1	16 bits	Block repeat save register
	RSA0, RSA1, REA0, REA1[a]	24 bits	Block repeat start and end address registers
	RPTC	16 bits	Single repeat counter

TABLE 7.14-3. TMS320C55xx programmer-controlled registers. Underlined registers are not included in the TMS320C54xx

[a] The TMS320C54xx has two 40-bit accumulators, one circular buffer size register, one circular buffer start address register, and one block repeat counter.

read directly from memory when performing single-cycle, dual multiply-accumulates. In simpler instructions, only one or two data operands are read directly from memory.

The TMS320C55xx supports operand-unrelated parallel data moves in some instructions with implied parallelism, such as the multiply-and-store instruction shown below:

```
MPYMR *AR0+,T0,AC1
:: MOV HI(AC0<<T2), *AR1+
```

This instruction performs a multiplication and stores the results in AC1; in parallel, the processor shifts and stores the upper 16 bits of AC0 to memory. Depending on the instruction, the move must specify either a16-bit load or a 16-bit store.

In addition to the operand-unrelated parallel move support within a single instruction, a pair of instructions that are executed in parallel may include a move instruction, thus achieving the same effect. Two 16-bit data operands can be stored to separate locations; or one 32-bit data operand can be stored to a 32-bit memory location. Similar operations are supported for loading data. Register-to-register, immediate-data-to-register, and memory-to-register data move instructions can be paired with arithmetic, data transfer, or branch instructions. In addition, the TMS320C55xx supports a swap instruction that exchanges the contents of one or two register pairs (i.e., AR6<=>T2 or both AR4<=>T0 and AR5<=>T1) in a single cycle.

Below are examples of parallel move support on the TMS320C55xx.

```
1) MOV #3, AC1
        || MOV AC2, dbl(*AR4)
2) MOV #3 << #16, AC1
        || MPY T1, AC3
3) MPY *AR3+, *CDP+, AC0
        :: MPY *AR4+, *CDP+, AC1
        || RPT CSR
```

The two operands for an arithmetic operation can both be accessed from memory, one from memory and one from a register, or both from registers. There are several restrictions on addressing modes used for parallel moves. When both operands are from memory, both must be accessed using register-indirect addressing with a non-immediate (-1, 0, +1, T0 or T1 register) address register post-modify, and may use bit-reversal and modulo addressing. When only one of the two operands are from memory, more flexibility is allowed in data addressing modes; a pre-modified address register, stack pointer with 16-bit immediate offset, or a 16-bit paged direct address is allowed. However, when a 16-bit immediate address register offset, stack pointer offset, or paged direct memory address is specified, the instruction requires 40 or 48 bits.

The TMS320C55xx also supports read-modify-write arithmetic and logical operations, as illustrated by the following example:

```
XOR   030h,*AR3(012h)
```

This instruction reads an operand from memory, *xor*s it with an immediate operand, and stores the result back to the same location in memory. This form of data access has the same effect as an operand-related parallel data move, in that the data is accessed by the same instruction that performs the arithmetic operation. In this example, address register AR3 is the base address and 012h is an immediate offset. This instruction executes in one cycle and is 48 bits long.

Orthogonality

As discussed in the *Instruction Set* section, the TMS320C55xx allows two instructions to be paired for parallel execution, but imposes several restrictions on which instructions can be paired. In addition, there are a number restrictions on the use of addressing modes for parallel data moves, depending on the type of data accessed and on whether the instruction is paired with another instruction for parallel execution. These restrictions complicate programming and reduce the orthogonality of the processor.

The TMS320C55xx instruction set and instruction execution model are unusual and fairly complex. "Implied" and "user-defined" parallelism can be confusing, and will likely create a need for programmers to look up instruction syntax regularly even after they are familiar with the device. In addition, the processor uses a number of mode bits (such as the ARMS bit) to control various aspects of its operation, which improve code density but complicate programming and debugging.

> *Overall, the TMS320C55xx is not particularly orthogonal. Its unusual instruction set allows the processor to maintain assembly source code compatibility with its predecessor while extending parallelism, but as a result it suffers somewhat from legacy constraints and does not provide a clean, straightforward programming model.*

Execution Times

Most TMS320C55xx instructions execute in a single cycle, including the read-modify-write instructions discussed earlier. Many program control instructions force pipeline flushes, and thus require four to seven cycles to execute. Program control instructions include all branches and subroutine calls (regardless of whether they use a relative or absolute address) and return instructions (i.e., subroutine and interrupt returns).

The first iteration of block repeat loops suffer a one-cycle stall if the first instruction in the loop requires more than 32 bits (thus requiring two fetches by the instruction buffer) and this instruction is the destination of a program discontinuity.

Pipeline conflicts are discussed in an earlier section. Most notably, a two-cycle pipeline stall occurs when an address register is loaded into a register, and the programmer attempts to use the source address register with address modification in the next instruction cycle.

Instruction Set Highlights

The TMS320C55xx instruction set and instruction execution model are unusual. The processor combines the ability to execute the complex, compound instructions typical of conventional DSP processors with the VLIW-like ability to execute up to two instructions in parallel. Other notable features of the TMS320C55xx instruction set include:

- Variable instruction widths ranging from 8 bits to 48 bits
- Instructions can specify multiple operations, and the processor can execute up to two instructions in parallel.
- Uniform single-cycle execution of instructions and instruction pairs with the exception of program flow-control instructions.
- The execute conditionally instructions (XCC and XCCPART) allow conditional execution of an instruction combined with them in parallel, or allow the following parallel instructions or non-parallel instruction to be executed conditionally.
- Dual 16-bit multiply, multiply-accumulate, add, subtract instructions.
- Read-modify-write instructions.
- Input data to computations, whether from memory or a register, can be pre-shifted before use as an operand.
- Register data can be shifted as it is stored to memory.
- Indexed addressing modes allow stack data or structure members to be operands of compute instructions.
- Exponent detect and normalize instructions.
- Division-step instruction.
- Specialized minimum/maximum, square distance, LMS filter tap and update, FIR dual tap, and absolute distance instructions.

Execution Control

Clocking

The TMS320C55xx allows use of an external crystal with its on-chip oscillator, or an external clock oscillator. The TMS320C55xx supports two clock modes: bypass and lock. In bypass clock mode, the master clock is the external clock frequency divided by 1, 2, 4, or 8.

In lock mode, the PLL multiplies the external clock frequency by an integer value between 2 and 31, or divides the external clock frequency by 1, 2, 3, or 4. The PLL output clock drives the master clock. In lock mode, when a 200 MHz master clock is desired, the external clock can vary between 6.45 MHz and 800 MHz depending on the PLL multiplier and divider settings. A clock output pin represents the master clock, and is phase locked to the external clock when in lock mode.

Hardware Looping

The TMS320C55xx allows three levels of nested hardware loops. Additional levels of loop nesting can be implemented using conditional branches or by storing and restoring loop control registers. Single and block repeat loop status are saved to the stack upon interrupt or subroutine call and restored upon return. This allows an interrupt service routine or subroutine to use loops without having to restore loop status before return. In single or block repeat loops, these routines must save upon entry and restore upon return the loop count, loop start address, and loop end address registers.

When all three hardware looping levels are used, the two outermost loops are block repeat loops; the innermost loop must be a single-instruction repeat loop which can repeat either one instruction or two parallel instructions. A single-instruction repeat can be used more than once inside a block repeat. Unlike on the TMS320C54xx, both single and block hardware loops are interruptible.

Support for interruptible single-instruction hardware loops is an advantage.

The block repeat instruction uses a loop count register that must be preloaded. When two block repeat loops are nested, the inner loop relies on a block repeat save (BRS0) register that can be preloaded before the outer loop; this can save cycles inside the outer loop when the inner loop uses a fixed repeat count. BRS0 can be modified inside the outer loop, but the modification must be done three cycles before execution of the inner loop's block repeat instruction to avoid a pipeline stall. Block repeats are terminated when the loop counter decrements below zero or if the program branches past the loop end address (similar to a C-language *break* inside a *while* loop).

There are two forms of block repeat instructions: repeat-block, and repeat-block-local. Repeat-block-local uses the instruction buffer queue to store the complete loop contents; loop contents must not exceed 56 bytes. This form of hardware loop conserves power and eliminates overhead due to program-memory wait-states or instruction buffer unit stalls. The assembler will generate a warning if the 56-byte limit is exceeded. If the limit is exceeded and the assembler warning is ignored, the loop will behave as a non-local repeat-block and fetch instructions from memory if they are not already contained in the instruction buffer queue. When the repeat block is bigger than the size of the instruction buffer queue, the instruction buffer queue provides a prefetch mechanism which tests the contents of the block repeat counter and starts refetching instructions at the top of the loop if BRC is greater than zero. This prefetch mechanism is not active for block-repeat-local loops.

The single-instruction repeat allows repetition of a single instruction or two parallel instructions. The repeat count can either be specified by an immediate constant or preloaded into a dedicated register. The contents of the dedicated register are copied into a second repeat count register, preserving the original repeat count. Thus, the repeat count can be reused, or (using a special form of the single repeat instruction) it can be incremented or decremented by a 4-bit constant or 16-bit register each time the single-repeat instruction is executed. The single-repeat instruction can be executed conditionally so that the loop is only executed if the result of a previous calculation is true.

> *The ability to re-use the loop count of a single-repeat loop across each iteration of the outer loop saves overhead cycles that would otherwise be required in some applications for setting up the inner loop. For example, this capability will be useful for the correlations used in speech compression algorithms.*

Interrupts

The TMS320C5510 has the following interrupt sources:

- External reset
- Non-maskable interrupt (NMI)
- Six edge-triggered external interrupts
- Two timer interrupts
- 13 interrupts generated by peripheral events (DMA, serial I/O, etc.)
- Bus error
- Data log event (described in the *On-Chip Debugging* section)
- Real-time operating system (RTOS) interrupt
- Six software interrupts

All interrupts are maskable except for the reset, non-maskable, data log event, RTOS, and bus error interrupts. Interrupts that occur simultaneously are serviced in order based on a fixed priority scheme.

TMS320C5510 peripheral interrupts include events from the host port, six serial port interrupts (one transmit and one receive from each of three serial ports), and interrupts from each of six DMA channels. The peripheral, software, and the external interrupts are individually maskable and all maskable interrupts can, as a group, be disabled by a global enable bit. Each maskable interrupt is also individually controlled by a unique debug interrupt enable bit in a control register. This bit, when set, allows the interrupt to be serviced, even when the processor is halted by on-chip emulation activity.

Each interrupt is assigned a unique eight-byte area in the interrupt vector table. When an interrupt event occurs, if the interrupt is unmasked, the processor starts executing code at the address stored at the vector location.

Normally upon interrupt (before executing code at the vector address), the address of the instruction in the decode pipeline phase is stored to the stack. In fast interrupt mode, the instruction's address is instead stored into a return address (RETA) register. Simultaneously, the contents of RETA, which are likely to contain the return address of the previously called subroutine, are pushed onto the stack. The instructions already in the pipeline, between the decode and execute phases, are completed as the instruction pipeline is fed new instructions starting from the address stored in the interrupt vector table. In fast interrupt mode, by not relying on retrieving of the return address from the stack, the interrupt return is completed faster.

Interrupts are executed with eight cycles of latency due to the seven- stage pipeline of the TMS320C55xx. During this latency, the four status registers, the loop control register, and debug status register are saved to the stack.

> *The fast interrupt feature on the TMS320C5510 should not be confused with the fast interrupt feature of Motorola DSPs, which reduces interrupt latency by inserting a small number of interrupt service routine instructions into the processor's pipeline without requiring a branch operation.*
>
> *The TMS320C5510 lacks a watchdog timer, a feature commonly used on processors that are responsible for system control. A watchdog timer forces a software reset if the processor fails to periodically rewrite a register. If this functionality is required on the TMS320C5510, it can be implemented in software via an interrupt service routine.*

Stack

The TMS320C55xx maintains two 16-bit wide software stacks, the system stack and data stack, each with unique stack pointers. Upon interrupt or subroutine call, the 8-bit loop status and 24-bit return address are pushed into the two 16-bit stacks simultaneously, using the two stacks as a single 32-bit stack. (The two stack pointers move together as a 32-bit stack, but the two stack pointers contain different addresses.) Upon interrupt or subroutine return, loop status and return addresses are popped from the two stacks.

The stacks can be configured in three modes. In 32-bit stack mode, the stacks are used in tandem, forming a single 32-bit wide stack; the two 16-bit stack pointers move in lock-step even when values are only pushed into the data stack. In addition, there are two dual 16-bit stack modes in which the two stack pointers are independent, and push and pop operations only affect the data stack. One 16-bit stack mode uses the fast interrupt return mode, in which the loop status and return addresses are stored to dedicated registers instead of to the stack in order to save cycles upon subroutine or interrupt return. The system stack is still used in this mode, because the previous contents of the dedicated registers must be stored order to support interrupt and subroutine nesting.

The second dual 16-bit stack mode saves the return address and loop status directly to the system stack as in 32-bit stack mode, and uses the data stack for local variables.

Bootstrap Loading

The TMS320C55xx, like earlier TMS320 families, supports a microprocessor (MP) mode and microcomputer (MC) mode, selected via the BOOTM pins upon hardware reset. There are three boot mode (BOOTM) input pins.

In microcomputer mode, the TMS320C5510 boot loads RAM from the 16-bit host port, 8- or 16-bit buffered serial port, or from an 16- or 32-bit external parallel device such as a flash EEPROM. In microcomputer mode, the on-chip ROM is memory mapped.

In microprocessor mode, the TMS320C5510 executes from external memory and the on-chip ROM is not accessible. After hardware reset or booting, the processor can be manually switched between microprocessor and microcomputer mode through software control, removing or inserting the on-chip ROM in the memory map.

> *The TMS320C5510 boot modes support 16-bit and 32-bit booting from an external device such as a flash memory, but 8-bit parallel booting is not supported. This is unfortunate, since it is often more cost-effective for system designers to use 8-bit flash memory.*

Peripherals

The TMS320C5510 includes a host port, three serial ports, a DMA controller, two timers, and a bit I/O port. Data may be transferred to and from peripherals directly under program control. In addition, DMA may be used to automate data transfers when using serial ports, making transfers between two memories, or between memory and an asynchronous external peripheral (for example, a FIFO, analog/digital converter, or digital/analog converter).

- **Host port**

  The host port is a 16-bit interface that allows a host processor to access TMS320C5510 internal and external memory. The host port interface consists of separate address and data buses. The host processor drives a 20-bit address (addressing the first 1M x 16 of TMS320C55xx memory), read or write signals, chip select, and data in the case of a write operation.

  Data access is indirect. When the host processor places addresses and data (in case of data write) on the host port bus, the host sees the TMS320C5510 as an asynchronous memory device. Within the TMS320C5510, the latched address and data (in the case of a data write) is handed to the DMA controller, which performs the data transfer. Upon completion, the TMS320C5510 DMA controller releases the host ready (HRDY) signal. HRDY is typically connected to the host's memory acknowledge input pin, stalling the host until the transfer is completed.

The host can interrupt the DSP by setting a bit in a host port control register. The DSP can interrupt the host processor by setting a control register bit, indirectly pulsing a pin that can be connected to one of the host processor's interrupt inputs.

The host port can operate in a multiplexed mode where the host processor's address and data is latched over the host data bus. Optionally, once the address has been latched, subsequent reads or writes can automatically increment the address so that address reloads are not necessary for contiguous data accesses. Upper and lower byte-enable input pins allow host processor access of the upper or lower bytes of TMS320C5510 16-bit memory locations.

- **Serial port**

  The TMS320C5510's three buffered serial ports (BSPs) are synchronous, bidirectional, and buffered. On the transmit side, each serial port is double-buffered and on the receive side, it is triple-buffered, allowing the processor more latency to respond to serial port events. The serial ports can be configured for up to 128 TDM time-slots where each transmit and receive time-slot can be selected by setting a unique bit in a bank of control registers.

  The serial ports can be used in a interrupt-driven mode where each word transmitted or received generates an interrupt, or in a DMA mode where interrupts are generated when a buffer of serial transmit data has been emptied or a buffer of receive data has been filled.

  Serial port transmit and receive clocks, frame syncs, and data lines are separate so that transmit and receive data streams can operate independently if desired. Serial clocks and frame syncs can be derived from the master clock and sent to a serial device. Alternatively, they can be received from a serial device or from another processor's serial port.

- **DMA controller**

  The TMS320C5510 DMA controller allows simultaneous DMA transfers in parallel with the normal memory accesses required by the processor's instruction flow. The DMA controller maintains up to six channels served in a round-robin priority scheme, and is capable of two 16-bit transfers per instruction cycle or 320 million 16-bit words/second on a 160 MHz TMS320C5510.

  Each DMA channel can be configured to transfer data between two memories (each either external or on-chip), between a serial port and memory, or between a memory-mapped device in external memory and memory. Each DMA channel has two eight-word FIFOs, one for reads and one for writes, to protect against overwrites of data in the event a DMA transfer cannot be immediately serviced. Source and destination data can be packed or unpacked by the DMA controller to accommodate differing device widths or data types of 8, 16, or 32 bits. DMA data transferred between memory and memory or memory and peripherals can be synchronized to serial port transfers, external interrupts, or timer interrupts.

The TMS320C5510 DMA controller has five ports. Two ports are dedicated to access of on-chip memory, one for DARAM and one for SARAM. These two ports can handle transfers simultaneously, but if two DMA channels attempt to use DARAM or SARAM in the same cycle, a conflict occurs and one of the DMA transfers is stalled until the other completes.

The TMS320C5510 has 32K x 16 of DARAM and 128K x 16 of SARAM, each divided into 4K x 16 blocks. The 4K x 16 block granularity is significant because DMA transfers cannot simultaneously use the same memory block as instructions or data operands required by processor instruction flow. When such a conflict occurs, DMA accesses are stalled since the CPU has priority.

In addition to the two ports for on-chip memory access, the DMA controller has three additional ports. One port connects to the host port, one port connects to the external memory interface, and one port connects to on-chip peripherals such as the buffered serial ports.

Each DMA port is capable of one transfer per instruction cycle. Stalls are incurred when DMA channels attempt to simultaneously use the same port for either source or destination data. The host port's DMA port is not considered a DMA channel. It resides in the DMA controller because host port transfers access DARAM and SARAM using DMA controller ports.

> *The TMS320C5510's DMA capability is more flexible than that of many older DSP processors because the six DMA channels are not dedicated to particular on-chip peripherals or to types of transfers. The ability to configure each DMA channel's transfers to synchronize to timer events or to an activity on an external interrupt pin adds to this flexibility.*

- **Timers**

  The TMS320C5510 has two timers. One timer can be dedicated to a real-time operating system or simple task switcher, for example, while the other timer is used for more general-purpose tasks such as timing flash memory writes or sector erases. Each timer's 16-bit counter can be decremented every 1 to 16 cycles depending on the value in the 4-bit pre-scaler register, providing up to 20 bits of range. The timer clock can be the master clock or an external timer clock input. A timer output pin can be used to create a clock with pulse widths equal to the timer interval or to generate a single pulse each time the timer reaches zero.

> *A 16-bit timer counter may be inadequate for applications requiring long timer intervals. For example, if the processor is running at 200 MHz, the timer will elapse every 5 milliseconds if the maximum pre-scaler value is used. If longer time intervals are necessary, an external clock can be used instead.*

- **Bit I/O**
  The TMS320C5510 has eight general-purpose input/output pins to, for example, test flag outputs from peripherals or control peripheral inputs. Three of the general-purpose input/output pins double as boot-mode configuration inputs on reset, but can be used as inputs or outputs after reset. A separate dedicated programmable output pin, XF, is a vestige from earlier generations of Texas Instruments' digital signal processors.

On-Chip Debugging Support

The TMS320C5510 has a JTAG port for hardware debugging. Through this port, an emulator can download programs, set breakpoints, start and stop program execution, and view or modify memory and register contents.

Texas Instruments' on-chip debug circuitry includes an analysis module. The analysis module uses address and data hardware breakpoint ranges combined with event counters to halt the processor after a specific number of occurrences of an event. The event can be the reading or writing of data in a specific range of memory addresses, evaluation of a conditional branch, or execution of an instruction at a specific program location. The event counter also tallies cycle counts for benchmarking or profiling purposes, eliminating the need to use one of the general-purpose timers for this purpose.

The TMS320C5510 on-chip debug circuitry allows a programmer to monitor registers or memory during program execution using the real-time data exchange (RTDX) facility. Texas Instruments' Code Composer debugger uses RTDX to provide features such as an oscilloscope-like display of a variable real-time, real-time watchpoint updates, and periodically storing the value of a variable into a file. These features are configured through the debugger and do not require modification of user code.

If debug data is time-critical, the transfer of debug data can be given priority over program execution, halting the processor as needed. RTDX also allows a programmer to send data from a host computer to the processor during run-time over the JTAG port. The data stream, for example, may emulate an input device not yet implemented in hardware or may temporarily replace an input device in order to inject test data.

> *The goal of hardware debugging tools is to approach the configurability of a software simulator while not requiring modification of user code or affecting real-time execution. The TMS320C55xx real-time data exchange features are state-of-the-art. The ability to monitor data in real-time, whether in a register or in memory, while simultaneously injecting test input data makes development more productive. The ability to stream test data to the target instead acquiring data through an actual I/O device also helps in test and debug.*

The TMS320C5510 can service maskable interrupts during debug while halted. This is controlled via a second set of interrupt mask registers that allow a programmer to select which interrupts are serviced when the processor is halted.

The on-chip debug circuitry includes an instruction trace FIFO. The FIFO stores either the last 32 program locations or the last 16 program location discontinuities. By storing the last 32 locations, the debugger can report exact program flow leading up to an event such as a breakpoint. By storing the last 16 program discontinuities, a longer history of program execution can be provided.

Power Consumption and Management

The TMS320C5510 is projected to consume 109 mW at 160 MHz and 1.6 volts, according to Texas Instruments.

The TMS320C55xx's configurable IDLE domains provide a means for a programmer to select which regions of the device will be disabled after an IDLE instruction has been executed. The regions are the processor core, DMA, instruction cache, peripherals, clock generation circuitry, and external memory interface. The internal memory is always powered regardless of which idle domains have been selected.

After execution of an IDLE instruction, the processor remains in an idle state until an interrupt (either non-maskable, or maskable and unmasked) occurs. If the serial ports and DMA controller have not been not configured as idle regions, serial data acquisition may continue as long as the serial port clocks and framing signals are provided externally or if generated internally, the clock generation circuitry is not configured as an idle region.

Benchmark Performance

The TMS320C55xx has been benchmarked with the BDTI Benchmarks™. In this section, we summarize and analyze the benchmark performance of the TMS320C55xx. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not consider the processor's instruction cycle rate; hence, lower instruction cycle counts do not necessarily indicate better performance. Next we discuss benchmark execution times and cost-execution time products, indicating processor speed and cost-performance, respectively. We then discuss the processor's energy consumption, which reflects the energy consumed by the processor in order to perform a task. Finally, we discuss the processor's memory usage, dividing the discussion into three sections: Control benchmark memory usage, overall program memory usage, and overall data memory usage.

The cycle counts serve as the basis for execution time results. The cycle counts presented here were obtained by BDTI using a pre-production TMS320C55xx development board, because a cycle-accurate simulator was not available at the time of this writing. Even with the assistance of Texas Instruments, however, it was not possible to measure cycle count results that matched our expected results to within one cycle. In addition, we were not able to determine the reason for the discrepancies. The results we present here are the measured results (rather than the results estimated via hand calculation). Unlike other benchmark results presented in this report, the TMS320C55xx results should be considered to be accurate only to within a few cycles.

Our execution time results use 160 MHz, the clock rate at which Texas Instruments states the TMS320C5510 is currently sampling; the pre-production device clock speed was measured at 20 MHz.

- **Instruction Cycle Counts:** As shown in Figure 8.1-13, the TMS320C55xx has a total normalized cycle count result that is approximately 10% lower than the average for all of the processors benchmark. The TMS320C55xx requires fewer cycles than the average of the processors benchmarked on eight of the twelve benchmarks. On four benchmarks the TMS320C55xx consumes more cycles than the average. As might be expected, in general the TMS320C55xx requires fewer cycles than processors with a single multiply unit (such as the ADSP-218x and TMS320C54xx), but more cycles than processors with more than two multipliers (the SC140 and TMS320C64xx).

  Several of the processors benchmarked in this report are similar to the TMS320C55xx in that they have two multipliers; these include the ADSP-2116x, DSP164xx, TMS320C62xx, and TMS320C67xx. These processors have total normalized benchmark cycle counts that are roughly 10-30% lower than that of the TMS320C55xx. A key difference between the TMS320C55xx and these processors is that the TMS320C55xx's two MAC units must share an input, a restriction not imposed by the other architectures. The TMS320C55xx's two MAC units are fully utilized on some benchmarks, but are not fully utilized on the Single-Sample FIR, LMS Adaptive FIR, and Vector Dot Product benchmarks because of this limitation.

  On the **Real Block FIR** filter benchmark, the TMS320C55xx is able to fully utilize its dual MAC units despite of the fact that the implementation does not use the processor's special FIR filter instruction (FIRADD) because that instruction requires use of symmetrical filter coefficients (a condition that cannot be guaranteed in the BDTI Benchmarks). The TMS320C55xx requires roughly half of the cycles required by most single-multiplier processors, but more than twice the cycles required by the two four-multiplier processors. Compared to the TMS320C54xx, the TMS320C55xx requires roughly 45% fewer cycles on this benchmark.

  The parallelism in the TMS320C55xx instruction set improves the processor's efficiency in the loop setup and function termination code; the repeat instruction is combined with the first dual multiply-accumulate, and immediate data loads are combined with indirect data transfers.

  For the **Single-Sample FIR** benchmark, the TMS320C55xx requires about 20% fewer cycles than the average. As in the Real Block FIR, the TMS320C55xx does not use the special FIR filter instruction (FIRADD) on this benchmark. The Single-Sample FIR benchmark processes only one sample per invocation. Therefore, each multiplication requires unique sample and coefficient inputs. The TMS320C55xx's dual multiply-accumulate capability cannot be used since it requires the two multipliers to share one data operand. However, the TMS320C55xx still has a lower cycle count than the TMS320C54xx because of its efficient loop setup and function termination code, as described in the Real Block FIR Filter benchmark discussion, above.

On the **Complex Block FIR** filter benchmark, TMS320C55xx requires about 55% fewer cycles than the average. The requirement that both TMS320C55xx multipliers must share a one operand is not an impediment when performing complex multiplications because each real and imaginary operand is used twice within a complex multiplication. On this benchmark, the TMS320C55xx requires fewer cycles than most of the other two-multiplier architectures: the ADSP-2116x, DSP164xx, and TMS320C67xx. However, the TMS320C55xx requires more cycles than the two-multiplier TMS320C62xx, which has several additional execution units in comparison to the TMS320C55xx and has shorter instruction latencies than the TMS320C67xx.

The TMS320C55xx has a relatively low cycle count on this benchmark because of its dual-MAC capabilities, and its ability to perform immediate data loads in parallel with data transfers and a block repeat instruction in parallel with the dual multiplications. In addition, the TMS320C55xx can shift accumulator results as part of a memory write, an operation that often requires two separate instructions on other processors.

TMS320C55xx cycle count is about average on the **LMS Adaptive FIR Filter** benchmark, and roughly 15% lower than that of the TMS320C54xx. The requirement that the two TMS320C55xx multipliers must share a data operand reduces the processor's efficiency in the FIR filter portion of the LMS Adaptive FIR filter. Since coefficients are updated every sample interval, two adjacent sample intervals cannot be calculated in parallel (the technique employed in the Real Block FIR filter). The coefficient-update portion of the LMS Adaptive FIR filter uses dual-multiply-accumulates because all multiplications have a common "error times adaptation rate constant" operand. In the second instruction of the two-cycle, two-coefficient update loop, the TMS320C55xx performs a dual 16-bit read and dual 16-bit write in parallel, each transferring two filter coefficients, in a single cycle.

The **Two-Biquad IIR Filter** on the TMS320C55xx has a cycle count that is about 15% lower than the average and is equal to that of the TMS320C54xx. The TMS320C55xx implementation splits the coefficient buffer into two separate buffers, one for the numerator and one for the denominator. Computations using numerator and denominator coefficients are performed in parallel. However, the results of the two accumulators need to be summed, requiring an extra cycle, which results in a net advantage over the TMS320C54xx of just one cycle. Even that advantage is masked by the extra cycle needed to initialize pointers for the split coefficient buffers. Thus, the advantage of the dual MAC capability is not apparent on this benchmark, but would be seen if the number of biquad sections were increased.

The TMS320C55xx requires more cycles on the **Vector Dot Product** benchmark than other two multiplier-accumulator processors such as the ADSP-2116x and DSP164xx. It requires about 15% more cycles than the average. The Vector Dot Product benchmark multiplies elements of two vectors and accumulates the products. Each vector element is used only once. The TMS320C55xx's relatively high cycle count on this benchmark is obtained because it can only use one multiplier-accumulator. In order for the TMS320C55xx to use both multiplier-accumulators, one multiplier input must be shared, which is not possible with this algorithm. Hence, the TMS320C54xx has a similar cycle count result on this benchmark. Surprisingly, though, the TMS320C54xx result is 5% *lower* than that of the TMS320C55xx. As mentioned earlier, BDTI was not able to measure cycle counts on the TMS320C55xx hardware that were accurate to within a cycle. Hence, it is possible that the actual cycle counts for the TMS320C55xx are not higher than those of the TMS320C54xx. It is also possible that this difference does exist; as discussed earlier in this chapter, the TMS320C55xx may require more cycles than the TMS320C54xx to execute the same code because of differences in the two processors' pipelines, a lack of delayed branches, and because of potential stalls from the instruction buffer queue.

TMS320C55xx is able to take advantage of dual 16-bit arithmetic operations for the **Vector Add** benchmark because it can perform two parallel 16-bit adds, fetch four operands, and store two results in a two-cycle, two-instruction loop. The TMS320C5510's ability to perform a dual 16-bit read and dual 16-bit write in parallel, as done in the LMS filter benchmark coefficient update, is a key enabler of the two-cycle loop, along with its ability to operate on operands in memory. Among the processors benchmarked, the TMS320C55xx requires 25% fewer cycles than the average. This result is comparable to the ADSP-2116x and DSP164xx, which also are capable of performing dual arithmetic operations.

The TMS320C55xx's **Vector Maximum** benchmark implementation requires roughly 20% fewer cycles than the average, more cycles than required by the other dual-MAC processors capable of dual arithmetic operations: the ADSP-2116x, DSP164xx, TMS320C62xx, and TMS320C67xx. Finding the maximum is performed efficiently on the TMS320C55xx, which can process two 16-bit vector elements each cycle. The TMS320C55xx uses the MAXDIFF instruction to calculate the Vector Maximum. This instruction compares two adjacent 16-bit operands in a 32-bit register with a two adjacent 16-bit operands from memory, finding two

16-bit maxima. Afterward, however, determining the maximum's index requires many cycles of post-processing.

The inefficient calculation of the maximum index is a side effect of the way in which MAXDIFF records the occurrence of a maximum. MAXDIFF shifts the 16-bit TRN0 and TRN1 registers right one bit and sets the most significant bit to one when an input is greater than the previously determined maximum, and to zero when the input is not greater than the previous maximum. For 16 inputs, the most-significant bit set in TRN0 and TRN1 represents the index of the most recently determined maximum for the lower and upper 16 bits in the 32-bit register. In order to find the maximum's index, the programmer must determine which bit in either TRN0 or TRN1 represents the most recent maximum and, based on the bit's position, determine the maximum's offset in the input data buffer.

A better TMS320C55xx maximum instruction would have automatically saved the index registers for the lower and upper maxima to two temporary registers. This would have eliminated much of the overhead required to enable use of the MAXDIFF instruction.

The TMS320C55xx has the highest cycle count on the **Control** benchmark, requiring about 60% more cycles than the average. Note, however, that this benchmark is optimized for minimum memory usage rather than minimum cycle counts, and is primarily an indicator of the processor's memory use efficiency. The TMS320C54xx has a slightly lower cycle count, about 45% higher than the average. The TMS320C55xx's ability to perform single-cycle arithmetic operations using immediate constants and to perform unrelated arithmetic and data move operations in parallel are useful features for this benchmark. The TMS320C55xx's high cycle count is due to branch and subroutine call overhead, which consumes four or five cycles per instance depending on whether the instruction is conditional and if conditional, whether the condition is true. Other processors either have shorter instruction pipelines or support delayed branches such that instructions can be placed after a branch to occupy otherwise empty pipeline slots.

Lack of delayed branches on the TMS320C55xx increases cycle counts in control code.

The TMS320C55xx cycle count on the **FFT** benchmark is about 15% higher than the average. The TMS320C54xx, in contrast, required 65% more cycles than the average. Another dual multiplier-accumulator 16-bit processor in this study, the DSP164xx, requires 10% more cycles than the average. The ADSP-2106x, ADSP-2116x, SC140, TMS320C62xx, TMS320C64xx, and TMS320C67xx all have very low cycle counts because they either have special instructions designed to optimize the FFT butterfly, or they have a large number of parallel multiplier and ALU units accessing data from a large bank of data registers. These processors have very low cycle counts which skew the average somewhat. The TMS320C55xx's radix-2 FFT butterfly, the most critical loop, requires five instructions and executes in six cycles due to pipeline stalls. In comparison, the ADSP-2106x's radix-2 butterfly executes in four cycles.

TMS320C55xx requires about 55% fewer cycles than the average on the **Viterbi** decoder benchmark. The TMS320C55xx cycle count is much lower than that of the TMS320C54xx, which is about 25% lower than the average. The TMS320C55xx replaces the TMS320C54xx's CMPS instruction with the MAXDIFF instruction. MAXDIFF simultaneously determines the two maxima from two 16-bit input pairs and shifts a one or zero into two transition registers, recording whether or not each maximum operation yielded a new maximum. In contrast, the TMS320C54xx CMPS instruction only generates one transition bit for one 16-bit input pair. The transition register results from the add-compare-select section of the algorithm reduce the trace-back section to a two-instruction loop in the TMS320C55xx.

Like the Control benchmark, the **Bit Unpack** benchmark demonstrates a processor's ability to handle functions sometimes delegated to a host microprocessor; in this case, extraction of bit fields from a block of 16-bit packed full-length words in memory. TMS320C55xx's cycle count on the Bit Unpack benchmark is about 20% lower than the average. In contrast, the TMS320C54xx requires 85% more cycles than the average. The TMS320C55xx's efficiency on this benchmark is partly due to its ability to use the address generation unit ALU as a 16-bit general-purpose ALU and to combine 16-bit ALU operations with data path operations using the accumulators. Also, the ability to perform arithmetic operations on data from memory rather than having to first load registers, as is the case with the ADSP-218x and DSP563xx processors, reduces the number of instructions required in the main loop.

- **Execution times:** The TMS320C5510's instruction cycle rate of 160 MHz is relatively fast, and gives the TMS320C5510 faster-than-average execution time results, despite its overall middle-of-the-road cycle counts. TMS320C5510's execution times are faster than those of non-VLIW processors, such as the ADSP-219x, ADSP-21065L, and TMS320C5416. The TMS3205510's total normalized benchmark execution time is about 35% below the average for fixed-point processors. Total normalized execution time results are illustrated in Figure 8.2-13.
- **Cost-execution time:** The TMS320C5510's moderately fast execution time performance coupled with its low price of $29.00 (quantity 10,000) give it a strong total normalized cost-execution time product, at about 30% below the average for the fixed-point processors in this report. As shown in Figure 8.3-13, the total normalized cost-execution time product for the $33.50 TMS320C5416 is about 65% higher than that of the TMS320C5510. The low cost of the ADSP-2186M at $8.50 (quantity 10,000) allows it to score similarly to the TMS320C5510 even though its instruction cycle rate is roughly half as fast at 75 MHz and it has a 50% higher benchmark cycle count.

- **Energy consumption:** As illustrated in Figure 8.4-13A and Figure 8.4-13B, the 1.6-volt TMS320C5510 has the second-lowest total normalized energy consumption of all benchmarked DSP processors, trailing only the MSC8101. The TMS320C5510's total normalized energy consumption is about half of the average for fixed-point processors.

Memory Usage

The focus in our memory usage analysis is on Control benchmark memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. This is because control-oriented instructions often make up the bulk of the software in an application, but take up only a fraction of the application's instruction cycles. Hence, unlike the other benchmarks, the Control benchmark is optimized for minimum memory usage rather than maximum speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP software. Finally, we discuss constant and non-constant data memory usage.

- **Control benchmark memory usage:** The TMS320C55xx has better than average total memory usage on the Control benchmark, requiring about 30% less memory than the average of all benchmarked processors, as shown in Figure 8.5-9A. The TMS320C55xx has variable-length instruction words that range from 8 bits to 48 bits. For control-oriented code, the processor relies primarily on 16-bit instructions, with a few exceptions. The TMS320C55xx requires 16 bits for short immediate data loads, stack push/pop operations, and arithmetic operations adding short immediate data to the accumulators. It requires 24 bits for PC-relative subroutine calls and branches. More complex instructions use 48 bits, but these are most commonly used for multiplier-accumulator intensive functions, not for control functions.

  The TMS320C55xx requires roughly 20%-25% less memory than processors with 24-bit instructions, such as the ADSP-218x, ADSP-219x, and DSP563xx. The TMS320C55xx requires roughly 10% more memory than the 16-bit instruction word processors such as the DSP568xx and DSP5685x. It requires 30% less memory than its predecessor, the TMS320C54xx, due to its support for PC-relative branches and short immediate constants for arithmetic and logical operations. The TMS320C54xx requires an absolute address for branches and only supports 16-bit immediate constants with arithmetic and logical operations.

  > *Part of the rationale for the TMS320C55xx's variable-length instruction word is to increase code density in control-oriented tasks. In this respect, the instruction set seems to have fulfilled its promise; the processor requires fewer program bytes than the 24-bit instruction word processors, and fewer than Texas Instruments' previous generation architecture, the TMS320C54xx.*

- **Program memory usage:** As shown in Figure 8.5-13, the TMS320C55xx's total normalized program memory usage is comparable to that of non-VLIW, fixed-point processors with fixed-size, 16-bit or 24-bit instruction words such as the ADSP-218x, ADSP-219x, DSP563xx, DSP568xx, and TMS320C54xx. Overall, the TMS320C55xx requires less program memory than the average, but this statistic is skewed by the fact that some of the VLIW processors, such as the TMS320C62xx, TMS320C64xx, and TMS320C67xx, require three to four times more program memory. This is not surprising since VLIW processors often require loop unrolling when optimizing for cycle counts, while non-VLIW DSPs such as the ADSP-218x and DSP563xx usually do not benefit from loop unrolling.

  The TMS320C55xx's highest program memory usage result occurs on the IIR Filter and Vector Maximum benchmarks. On the IIR Filter, the program size is high because the TMS320C55xx uses a high concentration of 48-bit instructions. The 48-bit instructions are parallel arithmetic instructions and long immediate address register loads. The TMS320C55xx's program memory usage on the Vector Maximum benchmark is expanded by the post-processing required to extract the maximum's index. On the Viterbi benchmark, the TMS320C5510 requires 50% less program memory than the average because of its dedicated instructions designed to optimize the algorithm's cycle count.
- **Constant data memory usage:** As shown in Figure 8.5-14, the TMS320C5510's constant data memory usage is comparable to that of other 16-bit data word processors such as the ADSP-218x, ADSP-219x, DSP568xx, SC140, DSP164xx, TMS320C54xx, TMS320C62xx, and TMS320C64xx on almost all benchmarks. The only significant difference is evident on the FFT benchmark, where the TMS320C55xx is able to perform the benchmark using smaller cosine/sine tables. Addressing mode limitations force most processors to use larger cosine/sine tables in order to optimize for cycle count. The TMS320C55xx has the second-lowest constant data memory usage result on the FFT; its constant data memory requirement is less than those of the ADSP-218x, ADSP-219x, DSP164xx, SC140, TMS320C62xx, and TMS320C64xx, but 50% more than the TMS320C54xx.
- **Non-Constant data memory usage:** As shown in Figure 8.5-15, the TMS320C5510's normalized non-constant data memory usage is similar to those of other 16-bit data word processors such as the ADSP-218x, ADSP-219x, DSP568xx, DSP164xx, TMS320C5416, TMS320C62xx, and TMS320C64xx. As expected, the 32-bit data word processors require roughly twice as much non-constant data memory. The most significant differences among 16-bit processors occur on the Viterbi benchmark, where the TMS320C55xx's specialized instructions eliminate the need for some intermediate data buffers required by other processors.

Cost

The quantity 10K pricing for the TMS320C5510 is $29 for the 160 MHz (320 MMACS) version and $35 for the 200 MHz (400 MMACS) version, each packaged in a 240-pin ball grid array (BGA) package.

Fabrication Details

The 160 MHz TMS320C5510 devices sampling at the time of this writing are fabricated using a 0.15 µm CMOS process with a 1.6 volt core supply. Texas Instruments expects to migrate the 160 MHz and 200 MHz devices to a 0.13 µm copper process in 2001.

Development Tools

The TMS320C55xx development software is based on the Code Composer IDE and tool suite. Code Composer includes an IDE that allows a programmer to configure and manage projects through a graphical user interface. When a build is selected, the IDE calls the Texas Instruments C compiler, assembler, linker, library builder, and hex conversion utilities as needed. Code Composer is available for Microsoft Windows 95/98 and Windows NT operating systems. The code generation tools can be called from the DOS command line, enabling use of another IDE or editor.

> *At the time of this writing, the TMS320C5510 assembler and simulator had several bugs. Most notably, the simulator was not yet cycle-accurate, making code timing difficult.*

The Code Composer debugger allows C and assembly level debug of software using a simulator or using a JTAG-based hardware emulator. The simulator and hardware emulator share the same user interface. The software simulator allows a programmer to simulate input or output pin activity and configure input/output data streams using files. Using JTAG emulation, a host data file can supply target hardware with real-time data. Real-time data streaming is made possible by the TMS320C5510's on-chip debug circuitry and its real-time data exchange (RTDX) feature.

Multiple Code Composer debug sessions can be controlled by the parallel debug manager. In the case of the hardware emulator, the JTAG ports of multiple TMS320C55xx processors in a system can be serially chained together and each processor can be individually controlled. The parallel debug manager allows multiple processors to be stepped, started, and halted together. Debug commands can be broadcast to one processor, a subgroup, or to all of the processors.

For the TMS320C55xx, Texas Instruments offers a tools suite called C5000 Code Composer Studio that includes the IDE, code generation tools, visual linker, software simulator, hardware emulator, DSP function libraries based on Texas Instruments' eXpressDSP algorithm coding standard, and eXpressDSP BIOS, a low-level operating system, for $2,995. The same package minus the hardware emulator costs $1,495. The C5000 Code

Composer Studio is augmented by Texas Instruments' third-party network, which includes a variety of suppliers of hardware and software. Code Composer Studio is available for Windows 9x and Windows NT.

> *In general, Texas Instruments provides an extremely competent software development environment with an advanced IDE, mature C compilers, sophisticated on-chip JTAG emulation capabilities, and cohesive multi-processor development tools. They also enjoy the largest third-party network of providers of algorithms, operating systems, emulators, target hardware, and engineering services. During our evaluation of the TMS320C55xx, however, we encountered a number of problems with the TMS320C55xx tools, including a lack of cycle accuracy in the instruction-set simulator.*

Applications Support

Texas Instruments' documentation includes separate manuals for the processor core, peripherals, instruction set, C compiler, and assembly language tools, plus specific processor datasheets. All documentation is available on-line or on a CD-ROM in PDF format. At the time of this writing, the TMS320C55xx DSP Peripherals User's Guide was only available as an on-line help document.

Web-based support includes answers to frequently asked questions (FAQs) and on-line forms for requesting literature or for asking a technical support question. Email applications support is reliable, usually responding within 24 hours.

Texas Instruments' telephone applications support staff includes front-line support personnel. If the problem is beyond the abilities of the front-line person, users are often connected to a more experienced applications engineer. If all engineers are busy, one will usually call back within a half day.

> *Texas Instruments' has very good applications support for general-purpose digital signal processors. Response is surprisingly prompt and helpful considering the number of processor families and customers that require support.*

Advantages

- Conditional execution of all instructions except program control instructions.
- Zero-overhead shifting of accumulators, inputs from memory, and results stored to memory.
- Zero-overhead saturation of computation results and of overflowed data when stored to memory.
- Eight address registers.
- Five simultaneous circular buffers.

- Good on-chip memory bandwidth.
- SIMD 16-bit adds and subtracts.
- Special instructions to optimize execution of FIR filters, LMS adaptive FIR filters, and Viterbi decoders.
- Exponent detection and normalization.
- Interruptible single-instruction hardware loops.
- Six simultaneous, concurrent DMA channels, each supporting memory-to-memory, peripheral-to-memory, or memory-to-peripheral.
- Three buffered, synchronous serial ports (up to 128 software selectable TDM time slots).
- Large on-chip memory (TMS320C5510).
- On-chip JTAG-based debug circuitry allows event trapping, real-time data exchange, and interrupt servicing while halted.
- Assembly source-code upward compatibility with previous generation (TMS320C54xx).
- Good benchmark energy consumption results on the BDTI Benchmarks.

Disadvantages

- Lack of delayed branch, delayed subroutine call, and delayed return from subroutine.
- Current devices do not support boot load from an 8-bit parallel device. (Only 16-bit and 32-bit devices are supported.)
- Timer range of only 20 bits (5 ms at 200 MHz master clock).
- As of this writing, simulator is not cycle-accurate.

7.15 Texas Instruments TMS320C62xx Family

BDTImark2000 Score: 1920 at 300 MHz

Introduction

The TMS320C62xx is a family of VLIW-based fixed-point DSP processors from Texas Instruments. The TMS320C62xx architecture, introduced in 1997, is completely different from that found in earlier DSP processor families from Texas Instruments. The processor contains eight execution units, including two multipliers and four ALUs. Using these eight execution units, the processor can execute up to eight 32-bit RISC-like instructions in a single clock cycle, enabling it to achieve a high level of parallelism. Instructions operate on 16-, 32-, or 40-bit data. The TMS320C62xx family targets applications with demanding performance requirements, such as wireless base stations, digital subscriber loops, multi-line modems, ISDN modems, imaging, and sonar systems.

Because the TMS320C62xx can execute a group of up to eight parallel instructions per clock cycle, the term “instruction cycle” is potentially ambiguous when discussing this processor. As used here, “instruction cycle” refers to the minimum time required to issue a group of parallel instructions. On the TMS320C62xx, this time is equal in length to one master clock cycle, i.e., one group of parallel instructions can be issued on every cycle of the master processor clock.

The first member of the TMS320C62xx family, the TMS320C6201, was announced in February 1997. This original 0.25 μm member used a 2.5-volt core supply (with 3.3-volt I/O). Texas Instruments has since replaced the 0.25 μm member with a 0.18 μm fabrication process; the 2.5-volt version is obsolete. The 0.18 μm version operates at 200 MHz using a 1.8-volt core supply (with 3.3-volt I/O) and executes up to 400 million MACs per second. (This device is referred to as a “1,600 MIPS” processor by Texas Instruments, since it executes a maximum of eight RISC-like instructions per clock cycle, when running at 200 MHz.)

In 1998, Texas Instruments announced two new variants of the TMS320C62xx: the TMS320C6202 and the TMS320C6211. The TMS320C6202 supports more on-chip memory and a higher clock frequency (250 MHz) compared to the TMS320C6201. The TMS320C6211 is a reduced-cost version providing on-chip caches for data and instructions and a lower operating frequency (150 MHz).

In 1999, Texas Instruments expanded the TMS320C62xx family further, announcing three new processor variants: the TMS320C6203, TMS320C6204, and TMS320C6205. These processors use a 0.15 μm fabrication process. The TMS320C6203 is currently the fastest TMS320C62xx processor: it runs at 300 MHz and executes up to 600 million MACs per second. The TMS320C6202, TMS320C6203 and TMS320C6204 are pin-compatible (the TMS320C6201 is not pin-compatible with the other TMS320C62xx processors).

In 2000, Texas Instruments announced a new revision of the TMS320C6202 using the 0.15 μm fabrication process. This new device is referred to as the TMS320C6202B. It will be equivalent to the TMS320C6202, but will use a lower core supply voltage and will therefore have lower power consumption.

The TMS320C6201, TMS320C6202, and TMS320C6211 are currently being produced in volume. The TMS320C6203, TMS320C6204, TMS320C6205 are sampling now, with volume production scheduled for the fourth quarter of 2000, according to Texas Instruments. Samples of the TMS320C6202B are scheduled for the first quarter of 2001. The TMS320C62xx family members are summarized in Table 7.15-1.

Texas Instruments also offers a floating-point version of the TMS320C6xxx architecture, the TMS320C67xx. The TMS320C67xx is covered in Section 7.17 of this report. The TMS320C67xx instruction set is a superset of that of the TMS320C62xx, adding floating-point support. The TMS320C67xx can execute TMS320C62xx object code unmodified, but the TMS320C62xx cannot execute all TMS320C67xx instructions.

In 2000, Texas Instruments announced the next generation of the fixed-point TMS320C6xxx family, the TMS320C64xx. This processor is covered in Section 7.16 of this report. The TMS320C64xx instruction set is a superset of that of the TMS320C62xx, adding an extensive range of SIMD operations and application-specific instructions (for telecom, audio, and image processing applications). The TMS320C64xx can execute TMS320C62xx object code unmodified, but the TMS320C62xx cannot execute all TMS320C64xx instructions.

The TMS320C62xx was the first commercially successful VLIW DSP processor. Since its introduction in 1997, many other VLIW-based DSP architectures have emerged, including the StarCore SC140.

By using a VLIW architectural approach, the TMS320C62xx achieves a high level of parallelism while avoiding the need for complex instruction scheduling and dispatch hardware in the processor. Instead, the burden of instruction scheduling is shifted to the code generation tools or the assembly language programmer. This results in a simpler and faster processor architecture compared to processors with run-time instruction scheduling.

VLIW architectures typically suffer from several disadvantages, such as high program memory usage and complexity in designing efficient compilers. The TMS320C62xx architecture includes several features designed to reduce program memory requirements and alleviate other disadvantages typically associated with VLIW architectures. These features include instruction packing, conditional execution for all instructions, and variable-length execution packets, all of which are discussed below. Despite these features, the TMS320C62xx consumes more program memory than other *fixed-point DSPs, as detailed in our discussion of benchmark results.*

Texas Instruments newer TMS320C64xx architecture, the next generation of the TMS320C62xx, was developed in an effort to address some of the weaknesses of the TMS320C62xx while providing substantially faster performance.

Device	Operating Voltage (V)	Max. Speed (Millions of MACs per Second)	On-Chip Memory		Notes
			Program RAM	Data RAM	
C6201	1.8 / 3.3[1]	400	16K×32	32K×16	Four-channel DMA, 16-bit host port interface, two buffered serial ports, two timers
C6202	1.8 / 3.3[1]	500	64K×32	64K×16	Four-channel DMA, 32-bit expansion bus, three buffered serial ports, two timers
C6202B[2]	1.5 / 3.3[3]	500	64K×32	64K×16	Same as 'C6202, but with improved DMA
C6203	1.5 / 3.3[3]	600	96K×32	256K×16	Same as 'C6202, but with improved DMA
C6204	1.5 / 3.3[3]	400	16K×32	32K×16	Four-channel improved DMA, 32-bit expansion bus, two buffered serial ports, two timers
C6205	1.5 / 3.3[3]	400	16K×32	32K×16	Four-channel DMA, two buffered serial ports, two timers, 32-bit PCI interface
C6211	1.8 / 3.3[1]	300	128×256 L1 cache	512×64 L1 cache	16-channel enhanced DMA, 16-bit host port interface, two buffered serial ports, two timers
			8K×64 unified L2 cache		

TABLE 7.15-1. TMS320C62xx processor family summary.

[1] The core operates at 1.8 volts while all I/O signals are 3.3-volt compatible.

[2] The TMS320C6202B is not yet available and will be sampled in early 2001, according to Texas Instruments.

[3] The core operates at 1.5 volts while all I/O signals are 3.3-volt compatible.

The TMS320C62xx and TMS320C67xx are the only DSP processor families to offer instruction set compatibility between fixed- and floating-point processors. This can be an advantage in applications where algorithms are initially designed using floating-point arithmetic and are later converted to a fixed-point implementation for volume production, because control code from the floating-point implementation does not need to be re-written for the fixed-point version. However, due to the different latencies of fixed- and floating-point instructions and other restrictions, algorithm kernels will almost always have to be completely re-written when migrating from the TMS320C67xx to the TMS320C62xx.

Architecture

The core architecture of the TMS320C62xx family consists of two fixed-point data paths, a program control unit (including program fetch, instruction dispatch, and instruction decode units), and program and data memory interfaces. Figure 7.15-1 illustrates the TMS320C62xx family architecture as typified by the TMS320C6201.

Data Path

The TMS320C62xx has two nearly identical data paths. As illustrated in Figure 7.15-2, each data path has a set of four execution units, a general-purpose register

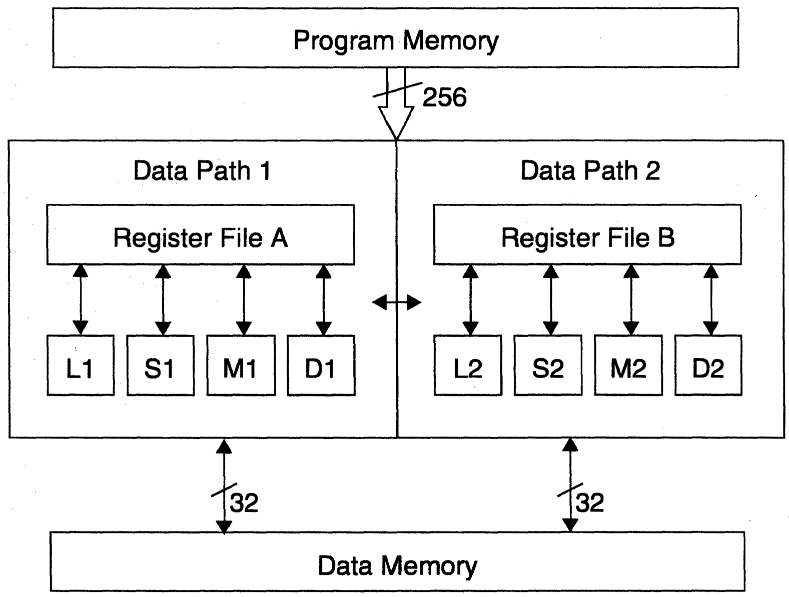


FIGURE 7.15-2. TMS320C62xx data paths. Each data path includes four execution units (L, S, M, and D), described in the text. The arrow between the data paths denotes the cross paths that allow each data path to access the register file of the other data path.

file, and paths for moving data between memory and the data path. The execution units in each data path consist of L, S, M, and D units. Typically the L, S, and D units operate on 32-bit operands, but the L and S units can also operate on 40-bit ("long") operands. The M units operate on 16-bit operands. As described below, each execution unit is capable of performing a specific set of operations. All of the execution units can operate in parallel.

- The **L units** (L1 for data path one, L2 for data path two) each contain a 40-bit integer ALU. They are used for 32/40-bit arithmetic and compare operations, 32-bit logical operations, and 32/40-bit normalization. The L units support saturated arithmetic for 32/40-bit operands. All L-unit operations execute in a single instruction cycle.
- The **S units** (S1 for data path one, S2 for data path two) each contain a 32-bit integer ALU and a 40-bit shifter. The S units are used to perform 32-bit arithmetic, logical and bit field operations, and 32/40-bit shifts. In addition, they are used for branching, constant generation, and register transfers to and from control registers.

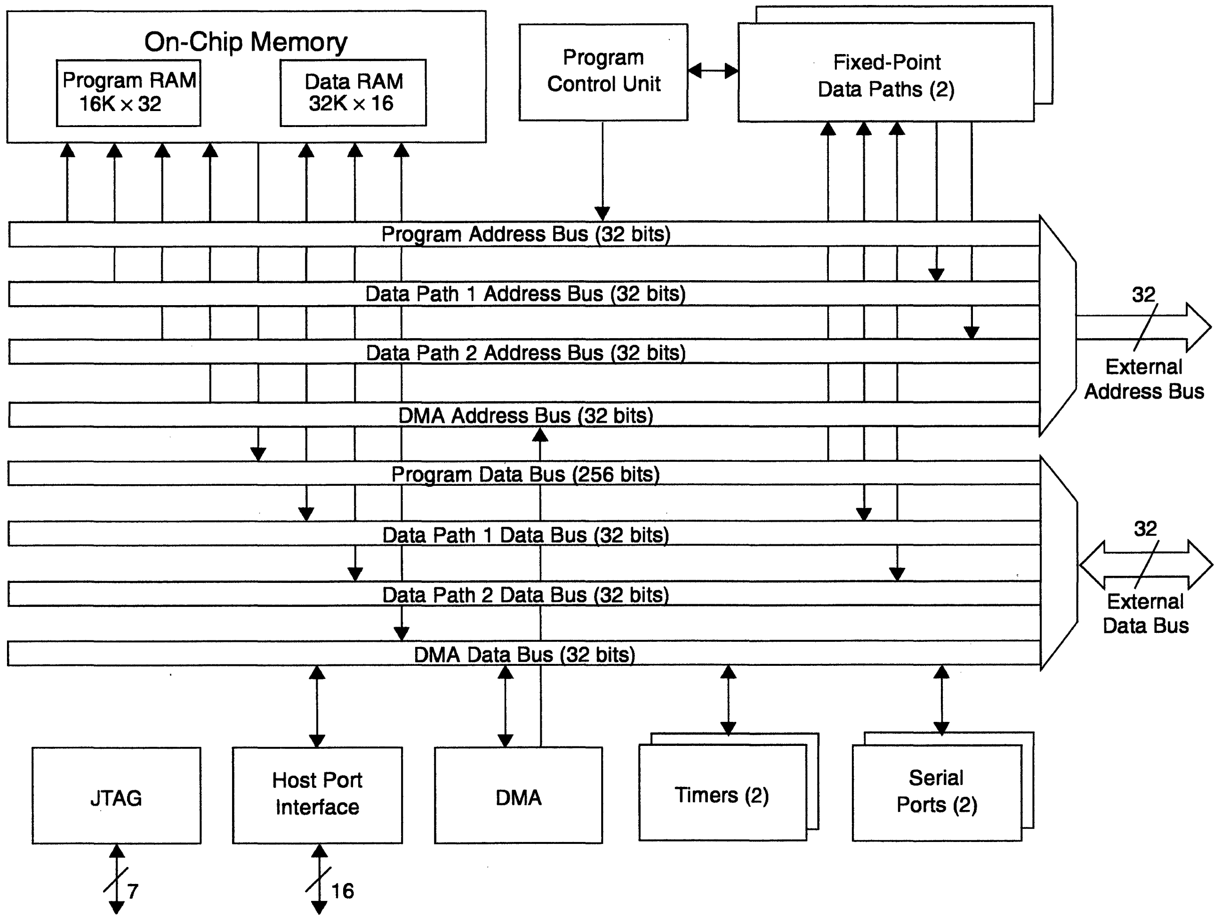

On-Chip Memory
Program RAM
16K x 32
Data RAM
32K x 16
Program
Control Unit
Fixed-Point
Data Paths (2)
Program Address Bus (32 bits)
Data Path 1 Address Bus (32 bits)
Data Path 2 Address Bus (32 bits)
DMA Address Bus (32 bits)
Program Data Bus (256 bits)
Data Path 1 Data Bus (32 bits)
Data Path 2 Data Bus (32 bits)
DMA Data Bus (32 bits)
32
External
Address Bus
32
External
Data Bus
JTAG
Host Port
Interface
DMA
Timers (2)
Serial
Ports (2)
7
16

FIGURE 7.15-1. TMS320C6201 processor architecture.

Most S-unit operations execute in a single instruction cycle. The only exceptions are branch instructions, which have single-cycle throughput but a six-cycle latency.

- Multiplications are performed by the **M units** (M1 for data path one, M2 for data path two), which are capable of performing 16 × 16 → 32-bit multiplications. Multiplier operands may come from the higher or lower 16-bit portions of any of the 32-bit general-purpose registers, enabling the use of pairs of 16-bit operands packed into 32-bit registers. Each multiplier can complete one multiplication per cycle, but multiplications have a latency of two cycles. The multipliers support integer multiplications of signed, signed/unsigned, and unsigned operands. In addition, fractional multiplication is supported for signed operands.
- The **D units** (D1 for data path one, D2 for data path two) each contain a 32-bit adder/subtractor. They are used for address generation including linear and circular address calculations. Because each data path has one D unit, the processor can perform a total of two address calculations in one instruction cycle.

In the best case, all units operate in parallel, and the processor performs four arithmetic operations, two multiplications, and two address calculations in one instruction cycle.

As a result of the processor's high clock speed and its ability to perform a maximum of eight operations in parallel, the TMS320C62xx offers a significant performance improvement over conventional DSP processors. This enables the use of a programmable DSP processor in applications that previously required multiple processors or custom hardware.

However, in many cases the processor's maximum level of parallelism cannot be achieved due to application constraints and resource limitations. In the BDTI Benchmarks™, the typical level of parallelism achieved in algorithm kernels is six to seven instructions per cycle. Additionally, the fact that several instructions require multiple cycles to complete (e.g., the multiplier has a two cycle latency and the branch instruction has a six cycle latency) complicates programming and forces the use of software pipelining in typical DSP algorithm implementations (e.g., convolution).

Note that the TMS320C62xx instruction execution rate of eight RISC-like instructions per instruction cycle, e.g., "1,600 MIPS" at 200 MHz, cannot be directly compared to the instruction execution rates of conventional DSP processors. For example, a MAC operation consisting of one multiplication, one addition, and two parallel moves is implemented as a single instruction on conventional DSP processors, but as four instructions on the TMS320C62xx.

The TMS320C62xx provides two register files, A and B, each containing sixteen 32-bit general-purpose registers. These registers can be used for storing addresses or data. The registers are labeled A0-A15 for data path one and B0-B15 for data path two. To support 40-bit arithmetic, pairs of adjacent registers can be used to hold 40-bit ("long") data. In this case the 32 LSBs (least significant bits) are stored in an even-numbered register and the 8 MSBs (most significant bits) are stored in the 8 LSBs of the next (odd-numbered) register. The remaining bits of the odd-numbered register are zero filled.

The TMS320C62xx implements a load-store architecture: operands must be loaded into the registers before they can be used by the execution units. Generally, the execution units of data path one operate on registers in register file A and the units of data path two operate on registers in register file B. However, the register files are interconnected to the opposite data path's functional units via cross paths. This allows each data path to fetch one 32-bit operand per instruction cycle from the register file of the other data path.

In each data path, each execution unit has its own read and write ports to its register file. Thus, all execution units in each data path can access the local register file simultaneously. This means that in an ideal situation all execution units in both data paths operate independently and eight simultaneous operations can be performed. However, some restrictions apply, the most significant of which are:

- Only one 40-bit (long) result can be written to each register file per instruction cycle.
- A 40-bit (long) register read cannot be issued in the same instruction cycle as a memory write from the same register file.
- Two simultaneous memory accesses cannot use registers of the same register file as address pointers.
- More than four reads of the same register cannot be performed in one instruction cycle.

These limitations are minor considering the complex architecture of the processor and should not cause any major problems in most applications. However, selecting 40-bit arithmetic reduces the number of available registers and restricts parallelism.

Overflow protection is supported on the TMS320C62xx via saturation logic and 40-bit arithmetic. Saturation is supported by the L, S, and M units via special instructions, such as add and subtract with saturation (SADD and SSUB). These instructions perform the indicated arithmetic operation and, in case of overflow, saturate the result to the largest positive or negative value that can be represented using 2's complement arithmetic. The saturated result is either a 32- or 40-bit value, depending on the width of the destination register. The L, S, and M units automatically set the saturation bit in the control status register when saturation occurs; this bit can only be cleared via an explicit instruction. The L and S units can operate on 40-bit operands, which corresponds to having a 32-bit register with 8 guard bits. A dedicated instruction can be used to convert a 40-bit value to 32 bits with saturation.

Besides the bit that indicates the occurrence of saturation, no other status bits (carry, negative, etc.) are provided by the TMS320C62xx data paths. The TMS320C62xx does not provide hardware rounding.

> *The lack of common status bits is not a problem on the TMS320C62xx; due to its parallel architecture and conditional instruction execution, status bits can be implemented (emulated) in software without a significant performance penalty.*

Memory System

The on-chip memory system of the TMS320C62xx implements a modified Harvard architecture providing separate address spaces for program and data memory. Program memory has a 32-bit address bus and a 256-bit data bus. Data memory has two 32-bit address buses and two 32-bit data buses. Both program and data memory spaces are byte-addressable for all of the TMS320C62xx processors.

> *The TMS320C62xx 32-bit address space is among the largest of fixed-point DSP processors. However, the external memory interface of the TMS320C6201, TMS320C6202, TMS320C6203, TMS320C6204, and TMS320C6205 can only address up to 52 Mbytes of memory (combining all of the four sub-memory regions described in the* External Memory Interface *section), which is equivalent to a 26-bit address space. The TMS320C6211 external memory interface can address up to 1,024 Mbytes of external memory.*

All memory accesses on the TMS320C62xx must be aligned. For example, this means that when a 16-bit access is performed, the address of the memory location accessed must be a multiple of 2. For 32-bit accesses, the address must be a multiple of 4. If an address is not aligned, no exception is generated. Instead, the next-lowest aligned address is used.

> *The fact that memory accesses must be aligned complicates programming and sometimes reduces performance. For example, in filters that require a delay line to be maintained to store successive inputs, the delay line is typically implemented using a circular buffer, and the starting location of the buffer is advanced by one sample position for each iteration of the convolution. In order to use the full processing power of the TMS320C62xx, it is necessary to load a pair of samples from the delay line via a single read. Due to the alignment requirement, though, this cannot be done if a circular buffer is used. Hence, filter implementations on the*

TMS320C62xx often use a different approach to implementing delay lines.

The TMS320C6201, TMS320C6204, and TMS320C6205 have 64 Kbytes of 32-bit on-chip program RAM and 64 Kbytes of 16-bit on-chip data RAM. The TMS320C6202 has 256 Kbytes of 32-bit on-chip program RAM and 128 Kbytes of 16-bit on-chip data RAM. According to Texas Instruments, the TMS320C6202B will have the same on-chip memory as the TMS320C6202. The TMS320C6203 has 384 Kbytes of 32-bit on-chip program RAM and 512 Kbytes of 16-bit on-chip data RAM. The TMS320C6211 has 4 Kbytes of on-chip level-1 cache program memory and 4 Kbytes of on-chip level-1 data memory. Additionally, it has 64Kbytes of unified level-2 on-chip program and data memory.

An on-chip program memory size of 64 Kbytes may appear to be large but is quickly consumed by the large code size of typical TMS320C62xx programs. As a result, the 64 Kbyte on-chip program memory of the TMS320C6201, TMS320C6204, and TMS320C6205 can be expected to accommodate roughly the same amount of application code as about 20 Kwords of on-chip memory on conventional 16-bit fixed-point DSP processors.

The on-chip program memory of the TMS320C6201 comprises a single block that consists of eight 8 Kbyte banks of 32-bit RAM. The data bus width of the program memory is 256 bits and thus, eight 32-bit instructions (called a “fetch packet” by Texas Instruments) are always loaded from the on-chip program memory at a time. The on-chip program memory can be accessed on 32-byte boundaries and can be configured to operate as on-chip RAM or as a direct-mapped instruction cache. Direct mapping means that an external program memory address can be mapped to only one cache location. (In contrast, with a set-associative cache an external address might be mappable to two or four possible cache locations, for example.) The processor’s on-chip program memory can operate in the following modes:

- **Program memory.** The on-chip program memory is used as program memory space and cache operation is disabled.
- **Cache enable.** The on-chip program memory acts as a direct-mapped instruction cache. Program memory reads access the cache and cache misses cause the cache to be updated.
- **Cache freeze.** The contents of the entire cache are locked. Program memory reads access the cache, but the cache is not updated on cache misses.
- **Cache bypass.** All program memory accesses are performed from external memory, i.e., the on-chip cache is completely bypassed. Cache misses do not cause the cache to be updated.

The on-chip program memory of the TMS320C6205 follows the same single block organization found on the TMS320C6201, but with an increased bank size. The on-chip program memory of the TMS320C6202, TMS320C6203, and TMS320C6204 is organized as a pair of blocks, one of which can be configured to serve as a program cache. The same four modes (program memory, cache enabled, cache freeze, cache disabled) are associated with the first block of on-chip program memory as on the TMS320C6201. The second block of program memory is always configured as program memory (not cache).

The support for two distinct blocks allows the programmer to implement a program-paging scheme. E.g., this partitioning allows the processor to execute a program from one block while the DMA is simultaneously loading the next segment of the code into the other block without contention.

The on-chip data memory of the 1.8-volt core TMS320C6201, TMS320C6202, TMS320C6203, TMS320C6204, and TMS320C6205 is divided into two equal-sized blocks (the initial 2.5 volt TMS320C6201 processor had only one block of data memory). On the 1.8-volt TMS320C6201, each block contains 32 Kbytes organized as four banks of single-access 16-bit RAM. This memory can be accessed 8 or 16 bits at a time, or 32 bits at a time when two memory banks are combined. Via the two 32-bit address and two 32-bit data buses, it is possible to fetch four 16-bit data operands at a time assuming that each operand comes from a different memory bank. Assuming no memory bank conflicts, the maximum on-chip data access rate is two 32-bit words or four 16-bit words (if located as pairs of adjacent words) per instruction cycle. On a 300 MHz TMS320C6203, this results scale in a maximum on-chip data memory bandwidth of 600 million 32-bit words per second (or 1,200 million 16-bit words per second).

> *Conventional DSP processors provide a single data path and typically support an on-chip data access rate of two data memory accesses per instruction cycle. On the TMS320C62xx, if two 16-bit operands are packed in one 32-bit data word, four 16-bit operands can be fetched in one instruction cycle. Special multiply instructions support this technique by allowing operands to be selected from the lower or upper halves of 32-bit registers. This functionality provides sufficient data memory bandwidth for the processor's two data paths.*

The TMS320C6211 replaces the on-chip program and data RAM found on the other TMS320C62xx processors with a 4 Kbyte on-chip level-1 program cache and a separate 4 Kbyte on-chip level-1 data cache. It also features a unified 64 Kbyte on-chip level-2 cache. The on-chip level-2 cache services requests from the level-1 data and program caches as well as from the DMA controller. The level-1 caches are only accessible by the CPU. The level-1 program cache of the TMS320C6211 is organized as a direct-mapped cache and features a 64-byte line size whereas the level-1 data cache is based on a 32-byte line size. The level-1 data cache is a two-way set-associative cache with a LRU (least-recently-used) replacement policy. The line size of the level-2 cache is 128 bytes. The level-2 cache is a one- to four-way set-associative cache, also with a LRU replacement policy, and each set can be independently and dynamically configured to serve as cache memory or on-chip RAM. The overall organization of caches and associated buses is depicted in Figure 7.15-3.

As mentioned earlier, the processor's core always fetches a group of eight instructions at a time, regardless of whether they are executed sequentially or in parallel. When groups are already present in the level-1 instruction cache (i.e., a cache hit occurs), up to eight instructions can be executed by the core every cycle. In contrast, when a group of instructions isn't found in the level-1 instruction cache but lies in the level-2 cache (i.e., a cache miss occurs in the level-1 instruction cache), five cycles are required to update a cache line. Two groups of eight instructions are copied from the level-2 memory to the level-1 program cache; the core's instruction pipeline stalls until the cache line update is complete.

The level-1 data cache of the TMS320C6211 is based on a dual-ported memory. As with the level-1 program cache, there is no cycle penalty when cache hits occur. This means that two load or two store operations can be issued every cycle (assuming no instruction cache miss occurs). Moreover, the dual-ported architecture of the level-1 data cache allows two concurrent accesses to the same cache line without cycle penalty. As with the level-1 program cache, when a cache miss occurs in the level-1 data cache, data is copied from the level-2 cache to the level-1 data cache. Because the level-1 line size is

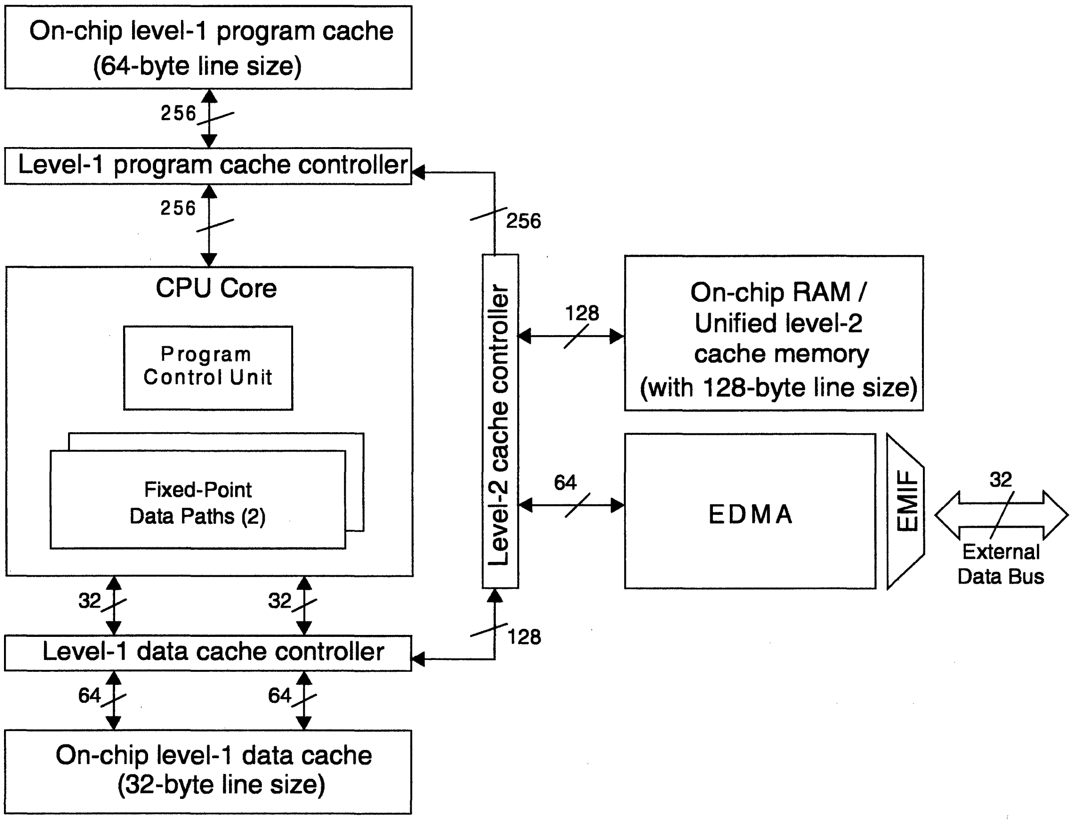

On-chip level-1 program cache (64-byte line size)
256
Level-1 program cache controller
256
256
CPU Core
Program Control Unit
Fixed-Point Data Paths (2)
Level-2 cache controller
128
On-chip RAM / Unified level-2 cache memory (with 128-byte line size)
64
EDMA
EMIF
32
External Data Bus
32
32
128
Level-1 data cache controller
64
64
On-chip level-1 data cache (32-byte line size)

FIGURE 7.15-3. TMS320C6211 cache organization.

twice the width of the bus between the level-1 data cache and the level-2 cache, two copy operations are required between the level-2 and level-1 data cache. Each copy operation takes 2 cycles to complete. Hence, a latency of four cycles is required to update an entire level-1 data cache line.

The level-2 memory of the TMS320C6211 is organized as four 64-bit wide banks. The level-1 program and data caches and the DMA can all access the level-2 memory whether it is configured as cache or as on-chip RAM. Concurrent accesses to the level-2 memory by any of these three requestors occur without contention when they access distinct banks. When concurrent accesses to the same bank are detected, a configurable priority mechanism gives access to the highest priority requestor. The priority level of the requestor can be configured by the application.

When the level-2 memory is configured as cache, the DMA controller is used to update a 128-byte level-2 cache line. A 64-bit data path is used to convey instructions or data between the enhanced DMA controller and the level-2 cache.

The bus between the level-2 controller and the DMA controller of the TMS320C6211 is 64 bits wide. Therefore, multiple requests are necessary to update a level-2 cache line. The number of cycles required to service each 64-bit request depends on the type of external memory being accessed, and whether the cache line being replaced must be written back to external memory before being replaced. For synchronous-burst SRAM (SBSRAM), if no write-back is required, the processor stalls for 47 cycles. During this period of time, the level-2 controller updates a 128-byte line of the level-2 cache. (This data was provided by Texas Instruments, assuming a 2:1 frequency ratio between the processor master clock and the SBSRAM clock. In the case of more conventional synchronous DRAM (SDRAM), the CPU stalls for 48 cycles if the requested SDRAM page is active and for 53 cycles if the SDRAM page is inactive, according to Texas Instruments.)

> *Roughly 2.5 cycles per byte are required to update a level-2 cache line in the scenario detailed above. This penalty is significant, and may cause severe bottlenecks when an application requires the cache to be frequently updated. Configuring part of the level-2 cache to serve as on-chip RAM somewhat mitigates the bandwidth limitation. In typical applications, the level-2 memory is partitioned as cache and on-chip RAM. The cached region usually holds instructions and the stack. One non-cached portion of the level-2 memory can be used to hold the input stream currently being processed, while an EDMA transfer request (issued by the program) uses another section of the non-cached region to import the next block of data. With this approach, the program is not required to wait for input samples to be downloaded into the level-2 memory before processing can occur.*

The TMS320C6211 allows manual invalidation of a specified range of addresses in the level-1 and level-2 cache by writing to a dedicated control register. The level-1 and level-2 caches can be flushed (causing the resident data to be copied back to external memory) and additionally, the level-2 cache can be cleared (causing the requested data to be tagged as not present without copying it back to external memory). The cache architecture of the TMS320C6211 allows a subset of memory ranges to be individually configured so as not to be cached in the level-2 and level-1 caches. This implements a cache bypass mechanism that allows, for example, reading a single word of data without the need to update an entire level-2 line. According to Texas Instruments, 35 cycles are required to read a single word from external memory.

> *The TMS320C6211 is the first commercial DSP processor to feature both level-1 and level-2 caches on chip. The use of small caches instead of large banks of on-chip RAM lowers the cost of this device, but increases the frequency of external memory accesses (each of which takes multiple cycles to complete), thus reducing performance. Additionally, the use of caches may result in unpredictable execution times in many applications. This can be a problem in applications with tight real-time constraints. These drawbacks are alleviated to some extent by the ability to configure portions of the level-2 cache as on-chip RAM and, to a smaller extent, by the manual invalidation mechanisms that are provided.*

The on-chip program and data memory interfaces of the TMS320C62xx are pipelined. Data memory reads have a throughput of one instruction cycle but a latency of five instruction cycles. Data memory writes have a throughput and an apparent latency of one instruction cycle. In fact, data memory writes take three instruction cycles to complete. However, the contents of a memory location written by the immediately preceding instruction can be read by the current instruction.

> *The latency of five instruction cycles for data memory reads complicates programming.*

All TMS320C62xx family members support both little- and big-endian byte ordering via an external pin. The state of this pin is readable in the control status register.

The TMS320C6201, TMS320C6202, TMS320C6203, TMS320C6204, and TMS320C6205 provide a four-channel DMA controller. The TMS320C6211 provides an enhanced DMA controller that supports 16 channels. These DMA controllers are discussed in more detail in the *Peripherals* section.

External Memory Interface

All TMS320C62xx external memory accesses pass through the processor's external memory interface (EMIF). The EMIF provides a 23-bit external address bus and a 32-bit data bus. These buses are multiplexed between program and data memory accesses.

External memory is divided into four spaces: CE0, CE1, CE2, and CE3. Each space has a dedicated chip-enable signal that is asserted during accesses to that space. On all TMS320C62xx family members except the TMS320C6211, spaces CE0, CE2, and CE3 are always 32 bits wide, while space CE1 can be accessed 8, 16, or 32 bits at a time to support low-cost boot ROM devices. In addition, the EMIF provides the necessary set of signals to interface to multiple types of external memory including asynchronous SRAM, synchronous DRAM, and synchronous burst SRAM. For synchronous DRAM, the necessary refresh signals are provided by the EMIF and the refresh period can be programmed in a dedicated period register. With synchronous memory, the speed of the external memory interface on the TMS320C6201 can be either 1X or 1/2-X the master clock rate. It is restricted to 1/2-X the master clock rate on the other TMS320C62xx members, except for the TMS320C6211. The TMS320C6211 employs a separate clock thus supporting a wider range of memory speeds.

The external memory interface of the TMS320C6202, TMS320C6203, TMS320C6204, and TMS320C6205 is similar to that found on the TMS320C6201. One important difference is that the external memory interface of the TMS320C6201 (and TMS320C6211) allows connection of synchronous burst SRAM and synchronous DRAM devices to the same DSP; the TMS320C6202, TMS320C6203, TMS320C6204, and TMS320C6205 do not support this capability.

The external memory interface of the TMS320C6211 is more flexible than those of the other TMS320C62xx members. On the TMS320C6211, accesses to asynchronous memory, SDRAM, and SBSRAM can be 8, 16, or 32 bits wide for each of the four CE memory regions. Additionally, unlike other TMS320C62xx processors, there are no restrictions on the type of memory associated with each CE region.

> *The external memory interface of the TMS320C62xx family is very flexible, providing logic to interface with many different types of memories.*

As with all processors, off-chip memory bandwidth depends heavily on the type and speed of the external memory. Given the high clock speed of the TMS320C62xx, however, obtaining and affording off-chip memory fast enough to keep up with the processor's maximum transfer rates may be a challenge. In the fastest case, when connected to a high-speed SBSRAM, the first memory access in a series requires additional cycles, but subsequent accesses can be performed in a single instruction cycle. In this configuration, the TMS320C6201 achieves a peak access rate of one 32-bit word per instruction cycle. This corresponds to a peak external memory bandwidth of 200 million 32-bit words per second on a 200 MHz TMS320C6201. (Because the external memory interface is limited to one half of the master clock rate on the TMS320C6203, the peak external memory bandwidth on a 300 MHz TMS320C6203 is 150 million 32-bit words per instruction cycle.) The external memory bandwidth for other types of memories is significantly lower; e.g., bandwidth is limited to 100 million 32-bit words per second for asynchronous SRAM and synchronous DRAM on the TMS320C6201. Additionally, the peak memory bandwidth for any particular type of memory can only be achieved via DMA transfers and instruction fetches. When accessing external memory via load or store instructions, latencies are always incurred due to the pipelined memory interface. The latency incurred for load and store instructions varies depending on the type of memory used and whether a load or a store is being performed and ranges from six cycles for a store to SBSRAM or asynchronous SRAM, to as many as 42 cycles for a load from an inactive row of SDRAM. Latencies are incurred for each 32-bit word accessed—performing loads or stores from or to sequential external memory locations does not eliminate the latencies after the first access.

The memory map organization of the TMS320C6204 and TMS320C6205 is identical to that of the TMS320C6201. The TMS320C6204 and TMS320C6205 also support the same variety of memory types and CE memory partitions as the TMS320C6201. The TMS320C6203 memory map is a superset of the TMS320C6202 memory map, which is a superset of the TMS320C6201 memory map. The internal and external memory map organization of the TMS320C6211 processor is not compatible with those of other TMS320C62xx members but is identical to that of the TMS320C6711 floating-point processor.

In typical applications, the peak external memory bandwidth will be significantly lower than 200 Million 32-bit words/second at 200 MHz for the TMS320C6201. The reason for this is that the full-speed SBSRAM interface would require very fast (5 ns) SRAM memory, which is not practical in many applications.

Even with high-speed SBSRAM memory, the TMS320C62xx cannot execute at full speed from off-chip memory. For example, if a program is executed entirely from off-chip memory, the peak instruction execution rate will be 50 million MACs per second at 200 MHz for synchronous burst SRAM on the TMS320C6201 (or 37.5 million MACs per second on a 300 MHz TMS320C6203, because the external memory interface is limited by the master clock rate on this device). Additionally, the external memory interface can become a serious bottleneck for applications that require the use of external data memory due to the latencies incurred by load and store instructions that access external memory.

Considering the bottleneck created by the relatively slow external memory interface of the TMS320C62xx, the size of the TMS320C6201, TMS320C6204, and TMS320C6205 on-chip memory is fairly small. In addition, the use of direct mapping (as opposed to two- or four-way set associativity) will limit the efficiency of the cache and increase traffic between the cache and external memory. The additional on-chip memory and 32-bit external expansion bus provided on the TMS320C6202 and TMS320C6203 alleviate this problem for many applications.

> *The two-level cached memory system of the TMS320C6211 has the advantage that external data memory accesses an entire cache line rather than fetching a single word from external memory. Thus, the latencies incurred for the access can be amortized over 32 data words. However, the small size of on-chip cache memory means that more external memory accesses will be required in most applications compared to other TMS320C62xx processors, thus reducing performance.*

Address Generation Units

As mentioned earlier, the TMS320C62xx has two address generation units, D1 and D2, one for each data path. In each unit, a 32-bit adder/subtractor is used for calculating memory addresses. Any general-purpose register can be used as an address register by these units. The D units can also be used as general-purpose ALUs (for 32-bit fixed-point addition or subtraction) if they are not needed for address generation in a given cycle.

The TMS320C62xx supports register-direct and register-indirect addressing modes and immediate data. In register-indirect addressing mode, the address register modification options include pre-increment/decrement by a short (5-bit) immediate or by the contents of any general-purpose register, and post-increment/decrement by a short immediate or by the contents of any general-purpose register.

Register-indirect addressing with indexing is also supported. In this case, immediate data or any general-purpose register can be used as an index. The immediate index can be either a short (5-bit) or long (15-bit) value. In the case of a long immediate index, only two registers, B14 and B15, can be used to hold the base address.

> *The address register modification options on the TMS320C62xx are flexible. The availability of indexed addressing is useful for compilers.*

A suffix of B, H, or W is used with load and store instructions to indicate the data width: byte, half word (16 bits), or word (32 bits). The modifier values are scaled according to the data width; i.e., depending on the data width—byte, half word, or word—the modifier value is left-shifted by 0, 1, or 2 bits, respectively.

Register-indirect addressing with a long immediate index can be used to implement an addressing mode similar to paged-memory-direct addressing: the registers B14 and B15 can be used as memory page pointers and 32 Kwords within each page can be addressed using 15-bit immediate index values.

The TMS320C62xx supports modulo addressing. Up to eight registers, A4-A7 for data path one and B4-B7 for data path two, can be configured to operate under modulo addressing. Two different circular buffer sizes can be simultaneously active. These buffer sizes must be powers of two and are programmed by specifying two 5-bit block-size fields in the "addressing mode register" (AMR). The circular buffer size is calculated as:

$$\text{Buffer size (in bytes)} = 2^{(N+1)}$$

where N is the 5-bit value in either of the block size fields. (Thus circular buffers can be of size 2, 4, 8, 16, ..., 2^{32} bytes.) In addition, the starting address of the buffer must be a multiple of the buffer size. For each register that supports circular addressing the AMR specifies the addressing mode used, which can be one of the following: linear (default), modulo addressing using block size field one, or modulo addressing using block size field two.

> *The restriction that circular buffer sizes must be powers of two is a slight drawback. While this does not significantly reduce performance in most applications, working around this limitation often requires more data memory in comparison to the data memory used by DSPs that support arbitrary circular buffer sizes.*

The TMS320C62xx does not support bit-reversed addressing.

Pipeline

The pipeline stages of the TMS320C62xx consist of fetch, decode, and execute operations which are further divided into 11 stages as presented in Table 7.15-2. The pipeline of the TMS320C62xx is non-interlocked and is significantly deeper than those found in most currently available DSP processors.

As mentioned earlier, instructions from on-chip program memory are always fetched eight at a time by the program fetch unit regardless of the number of instructions that will be executed in parallel. A group of eight 32-bit instructions comprises a "fetch packet." The instructions within a fetch packet that are to be executed in parallel form an "execution packet." Thus, one execution packet may contain up to eight instructions. Execution packet boundaries are identified by the least significant bit (LSB) of each instruction. If the bit is set for a particular instruction, then the instruction is dispatched in

Operation	Stage	Description
Fetch	PG	Program address generate
	PS	Program address send
	PW	Program address ready wait
	PR	Program fetch packet receive
Decode	DP	Instruction dispatch
	DC	Instruction decode
Execute	E1	Execute stage 1
	E2-E5	Execute stages 2-5 (used by multiply and load instructions)

TABLE 7.15-2. Pipeline stages of the TMS320C62xx.

parallel with the next instruction. This is illustrated in Figure 7.15-4. For example, if one fetch packet contains eight execution packets, each execution packet contains only one instruction (Fetch Packet A in Figure 7.15-4). If one fetch packet contains only one execution packet, the execution packet contains eight instructions (Fetch Packet B in Figure 7.15-4). If a fetch packet contains four execution packets, each execution packet may contain from one to five instructions (Fetch Packet C in Figure 7.15-4). It is up to the code generation tools or the programmer to determine which instructions can be executed in parallel.

The LSB of the last instruction of the fetch packet is always zero because execution packets cannot cross fetch packet boundaries. If an execution packet attempts to cross a fetch packet boundary, the linker places it in the next fetch packet. The remainder of the current fetch packet is filled with NOP instructions.

> *Note that the architecture of the TMS320C62xx differs from traditional VLIW architectures by allowing the processor to fetch multiple execution packets at a time. Texas Instruments implements this by utilizing variable-length execution packets. (In traditional VLIW architectures, execution packets have a fixed length.) This technique is used to reduce the large program memory requirements typically associated with earlier VLIW architectures. The restriction that execution packets cannot cross fetch packet boundaries limits the effectiveness of this technique, however. This restriction is removed in the TMS320C64xx to improve code density.*

Instructions are grouped into execution packets by the assembly language programmer and/or the code generation tools. The processor performs no resource contention checking among instructions in an execution packet, and therefore resource conflicts in manually written assembly language code that are not detectable by the tools may lead to unwanted behavior. This is discussed in more detail in the *Instruction Set* section, below.

After a 256-bit fetch packet is fetched by the processor, the instructions of one execution packet are dispatched to the corresponding execution units in parallel. If a fetch packet contains more than one execution packet, the next fetch packet is not fetched until the last execution packet of the current fetch packet has been dispatched.

> *Note that in contrast to some VLIW architectures, instructions within execution packets on the TMS320C62xx are not positional. This allows the length of execution packets to vary, as described above.*

Most TMS320C62xx instructions require only one execution stage (E1), and thus execute in seven pipeline stages. However, multiplications and data loads require multiple execution stages and thus use more than seven pipeline stages.

The pipeline of the TMS320C62xx is visible to the programmer in the following cases:

- Multiply operations take two execution stages to complete (E1 and E2), thus introducing a latency of two instruction cycles. However, multiplications have single-cycle throughput allowing a new operation to be started in every instruction cycle.
- Data loads have a throughput of one instruction cycle but introduce a latency of five instruction cycles. For a data load instruction, the execution stage E1 is used to calculate the memory address and four additional stages (E2-E5) are used to address the data, read the data, and load it into a register.
- All branches on the TMS320C62xx are delayed branches introducing five delay slots. This is because the branch instruction is executed in the E1 stage, and all previously fetched instructions are dispatched before the branch takes effect.
- The pipeline is stalled by one instruction cycle if two data memory operands are read from the same memory bank in one execution packet. Additionally, interrupts may stall the pipeline.

> *Pipeline effects significantly complicate TMS320C62xx assembly programming. Because the pipeline is both deep and completely exposed, writing assembly code that is both correct and efficient can be extremely tricky.*
>
> *Additionally, the exposed pipeline of the TMS320C62xx can cause problems when servicing interrupts. This issue is discussed further below.*

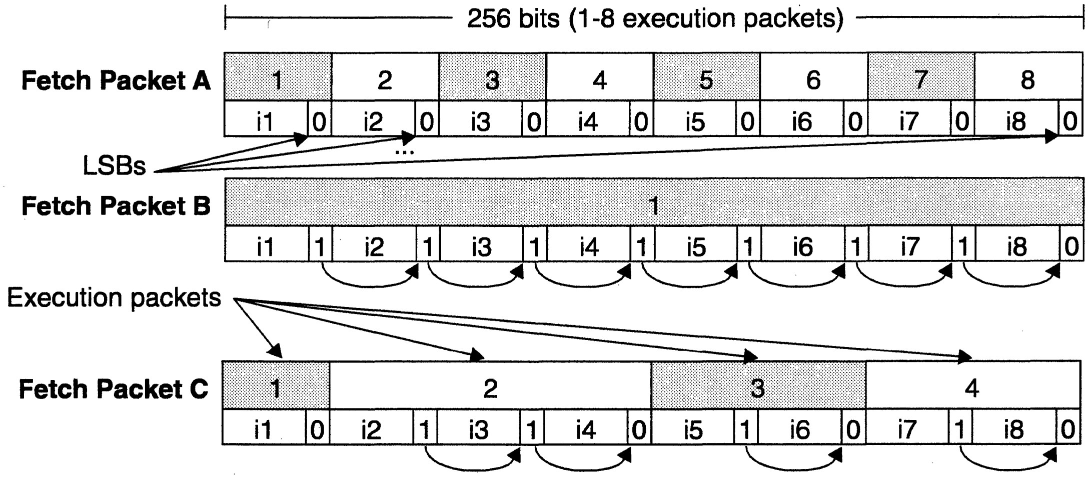


FIGURE 7.15-4. Example fetch packets A, B, and C. Fetch packet A consists of eight instructions (i1-i8), each forming its own execution packet indicated by the LSB of each instruction. Fetch packet B contains one execution packet consisting of eight instructions. Fetch packet C contains four execution packets, each consisting of one to three instructions. Execution times for fetch packets A, B, and C are eight, one, and four cycles, respectively.

The pipeline of the TMS320C62xx is a good target for high-level language compilers in the sense that the pipeline latencies are readily predictable, and each instruction has single-cycle throughput. On the other hand, the TMS320C62xx architecture introduces many challenges for both high-level language compilers and assembly programmers, such as complicated instruction scheduling and parallel dispatching. To alleviate some of the complexity encountered by an assembly programmer, Texas Instruments developed a new tool, the "Assembly Optimizer." This tool is further discussed in the Development Tools *section.*

Instruction Set

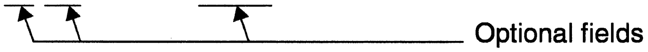
The TMS320C62xx registers and instruction set are summarized in Tables 7.15-3 and 7.15-4, respectively.

Assembly Language Format

The TMS320C62xx uses an opcode-operand assembly language format where each instruction has an opcode field for the operation and an operand field for one to four operands. In addition, three optional fields can be used to indicate parallel execution, conditional execution, and the targeted execution unit.

For example, the instruction:

```
||[A0] SADD  .L1   A1,A2,A1
```



adds the 32-bit value stored in register A2 to register A1 and saturates the result. The "||" symbol indicates that the instruction is to be executed in parallel with the previous instruction. The "[A0]" symbol indicates that this instruction will be executed only if the contents of register A0 are non-zero. The ".L1" directive indicates the execution unit to be used.

If the target execution unit field is omitted from the instruction, the assembler attempts to automatically select an appropriate execution unit.

Registers	Width	Purpose
A0-A15	32 bits	General-purpose registers for data path 1
B0-B15	32 bits	General-purpose registers for data path 2

TABLE 7.15-3. TMS320C62xx register summary.

Class	Instructions
Arithmetic	32/32 or 32/40-bit signed or unsigned addition or subtraction with optional saturation; dual 16-bit SIMD addition or subtraction of two 16-bit integers
Multiplication	Signed, signed/unsigned, and unsigned 16 × 16 → 32 multiply; signed multiply and left shift by one with saturation
Logic	*And, or, exclusive-or, not*
Shifting	Arithmetic/logical left/right shift by 0-40 bits, left shift by 0-40 bits with saturation
Rotation	*none*
Data move	Load/store from/to memory with/without indexing, move data between registers, move a 16-bit signed constant into a register, store to program memory, zero a register
Comparison	Compare for equality, signed/unsigned compare for greater/less than
Looping	*none*
Branching	Branch, branch using interrupt return pointer, branch using non-maskable interrupt (NMI) return pointer
Subroutine Call	Branch using register
Bit Manipulation	Set/clear bit field, extract and sign-extend a bit field, extract unsigned bit field
Special Function	Absolute value with saturation, leftmost significant bit (exponent) detection, normalize, convert a 40-bit value to 32 bits with saturation, conditional subtract and shift (to support division), IDLE (multi-cycle NOP with no termination until interrupt), NOP

TABLE 7.15-4. TMS320C62xx instruction set summary. Note that all instructions can be executed conditionally.

As mentioned earlier, an execution packet on the TMS320C62xx consists of up to eight simple, RISC-like instructions. All instructions inside an execution packet are executed in parallel. For example, the execution packet:

```
LOOP:
         ADD    .L1   A0,A3,A0     ;A0=A0+A3
||       ADD    .L2   B1,B7,B1     ;B1=B7+B1
||       MPYHL  .M1X  A2,B2,A3     ;A3=A2(hi)*B2(lo)
||       MPYLH  .M2X  A2,B2,B7     ;B7=A2(lo)*B2(hi)
||       LDW    .D2   *B4++,B2     ;load h(i) & h(i+1)
||       LDW    .D1   *A7--,A2     ;load x(i) & x(i+1)
||[B0]ADD       .S2   -1,B0,B0     ;decrement counter
||[B0]B         .S1   LOOP         ;branch if B0 non-zero
```

implements the kernel of an FIR filter calculating two taps at a time. ADD instructions accumulate the previous products to the A0 and B1 registers. The MPYHL instruction multiplies the higher portion (16 MSBs) of the A2 register with lower portion (16 LSBs) of the B2 register and stores the 32-bit result to the A3 register. MPYLH multiplies the lower portion of the A2 register with the higher portion of the B2 register and stores the result to the B7 register. LDW instructions load new 32-bit words (containing two 16-bit values each) to the B2 and A2 registers while post-incrementing/decrementing the pointers in registers B4 and A7. If the loop count register (B0 in this case) is non-zero, the ADD instruction decrements the loop counter by one. Also depending on the contents of the B0 register (if it is non-zero), the B instruction branches conditionally back to the beginning of this execution packet. This execution packet can be repeatedly executed once per instruction cycle. For this loop to work properly, preceding execution packets must initiate multiplies, loads, and branches appropriately to account for the multi-cycle latencies of these instructions (refer to *Hardware Looping* below).

> *The fact that multiplies, loads, and branches have multi-cycle latencies significantly complicates writing and understanding code. For example, assuming that the FIR kernel above is executing the k^{th} iteration of the loop, the results of the multiplies issued during this iteration become available in iteration k+2 of the loop. Similarly, the data loaded during the current k^{th} iteration are only available during iteration k+5 and the branch issued in the current iteration takes effect at the end of the $k+6^{th}$ iteration.*

All instructions on the TMS320C62xx can be executed conditionally. Five general-purpose registers, B0, B1, B2, A1, and A2, can be used as condition registers. For example, the instructions:

```
         [B0]  ADD   .L1   A1,A2,A1
||       [!B0] ADD   .L1   A1,A3,A1
```

are mutually exclusive and add A2 to A1 if B0 is non-zero, or A3 to A1 if B0 equals zero.

The conditional execution support provided on the TMS320C62xx is very flexible and powerful.

The assembler and compiler are capable of detecting “simple” resource conflicts such as multiple writes to the same register within one execution packet. More complicated conflicts cannot be detected by the assembler or compiler and may lead to incorrect results. These include:

- Instructions with different latencies writing to the same destination register. For example, issuing an ADD one cycle after a MPY with the same destination register will cause a failure because of their different latencies.
- Instructions that share resources and are executed conditionally depending on the contents of unrelated registers. For example:

```
          [A1]   ADD    .L1    A1,A3,A1
   ||     [B1]   ADD    .L2    A1,A3,A1
```

  If both A1 and B1 are non-zero, multiple writes to register A1 will be performed, causing a failure. In this case, it is the programmer’s responsibility to ensure that A1 and B1 are not simultaneously non-zero.

Parallel Move Support

The TMS320C62xx supports operand-unrelated parallel moves by allowing up to two data move instructions to be executed in parallel with other instructions. These data moves can be either two loads, two stores, or a load and a store.

Orthogonality

Due to its 32-bit instruction width, simple (RISC-like) instructions, and uniform register sets, the instruction set of the TMS320C62xx is extremely orthogonal. Restrictions regarding the use of registers apply to conditional execution, circular addressing, and the use of 15-bit immediate indexes in indirect addressing.

Despite the orthogonal instruction set of the TMS320C62xx, the processor is extremely complex to program due to its high level of parallelism and the need for the programmer to manually schedule instructions for maximum performance. Texas Instruments’ assembly optimizer tool (discussed below) and C compiler simplify code development by automating the scheduling and parallelization processes, but these tools do not always result in optimal code.

Execution Times

Most instructions on the TMS320C62xx have a latency of one single cycle of the master processor clock (assuming no delays due to memory bank conflicts or interrupts). The branch, multiply, and load instructions produce results (or produce an effect in case of a branch) only after multiple cycles. (E.g., once issued, six cycles are required for a branch to take effect.) This multi-cycle latency does not mean that other instructions can not be issued while results or effects are pending. Indeed, assuming all instructions and data reside in on-chip memory and that memory bank conflicts and interrupts are avoided, all of the TMS320C62xx instructions have single-cycle throughput. This means that a new instance of an instruction can be issued in every cycle. This single-cycle throughput makes it possible for the pipeline to dispatch one execution packet (containing up to eight instructions) per instruction cycle, although some of the instructions issued can have a multi-cycle latency.

Instruction Set Highlights

The RISC-like instruction set of the TMS320C62xx is small and consists of fairly simple instructions. Noteworthy features include:

- Conditional execution of all instructions.
- Normalization instruction. This instruction can be used for exponent detection and as a part of normalization operation. It counts the number of redundant sign bits.
- Multicycle NOP. This instruction can be used to fill the branch and load delay slots without significantly increasing code size.

Execution Control

Clocking

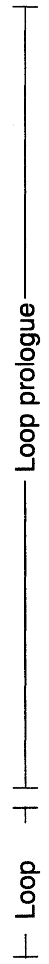
The TMS320C62xx uses a 1X master clock. An on-chip PLL is provided to allow the on-chip master clock to be generated from a slower external clock. The TMS320C62xx requires an external oscillator to drive the PLL. On the TMS320C6201, TMS320C6211, and TMS320C6202, the PLL can be programmed to multiply the input clock by a factor of 1 or 4 via two dedicated clock mode pins. Setting the multiplication ratio to one causes the PLL to be bypassed. The TMS320C6203, TMS320C6204, and TMS320C6205 offer an extended range of frequency multipliers: factors of 1, 4, 6, 7, 8, 9, 10, and 11. The TMS320C6202B will also support the same set of extended factors, according to Texas Instruments.

Hardware Looping

The TMS320C62xx does not support hardware looping; all loops must be implemented in software. However, the parallel architecture of the processor allows the implementation of software loops with virtually no overhead.

Loops are implemented on the TMS320C62xx by conditionally decrementing the loop counter (stored in a general-purpose register) and branching to the top of the loop in parallel with other instructions. The branch instruction latency of six instruction cycles complicates assembly language programming, especially when loops contain fewer than six execution packets. In such short loops, the pipeline must be properly filled with branches to the top of the loop prior to execution of the first loop instruction. The example in Figure 7.15-5 illustrates the programming of single-packet software loops. As presented in this example, the pipeline is filled with branches in the loop-prologue code. After the loop prologue is executed, the loop can be started with a throughput of one execution packet per instruction cycle. In the first five iterations of the loop, the five branches executed in the loop prologue take effect. After the first five loop iterations, in each loop iteration *k>5*, the branch encountered in loop iteration *k-5* takes effect.

> *Due to the deep pipeline and the delayed branch instructions of the processor, implementing loops on the TMS320C62xx in assembly language is very tricky. The long branch latency leads to heavily pipelined loop implementations that result in increased program memory usage for computer-generated and hand-optimized assembly codes, and assembly code that is difficult to write and understand.*



```
; Execution packet 1:
...<0-6 other instructions>...
||[B0]B     LOOP        ; Branch to LOOP
||[B0]ADD   -1,B0,B0    ; Decrement loop cnt

; Execution packet 2:
...<0-6 other instructions>...
||[B0]B     LOOP        ; Branch to LOOP
||[B0]ADD   -1,B0,B0    ; Decrement loop cnt

; Execution packet 3:
...<0-6 other instructions>...
||[B0]B     LOOP        ; Branch to LOOP
||[B0]ADD   -1,B0,B0    ; Decrement loop cnt

; Execution packet 4:
...<0-6 other instructions>...
||[B0]B     LOOP        ; Branch to LOOP
||[B0]ADD   -1,B0,B0    ; Decrement loop cnt

; Execution packet 5:
...<0-6 other instructions>...
||[B0]B     LOOP        ; Branch to LOOP
||[B0]ADD   -1,B0,B0    ; Decrement loop cnt

; Execution packet k:
LOOP:
...<0-6 other instructions>...
||[B0]B     LOOP        ; Branch to LOOP
||[B0]ADD   -1,B0,B0    ; Decrement loop cnt
```

FIGURE 7.15-5. An example “single-packet” loop.

Interrupts

The TMS320C62xx architecture supports up to twelve maskable interrupts and two non-maskable interrupts. The non-maskable interrupts include reset and NMI. Not all of these interrupts are available on all TMS320C62xx family members; for example, the TMS320C6201 has seven maskable and two non-maskable interrupts.

Interrupts are prioritized with non-maskable interrupts having the highest priorities. A low-to-high transition on interrupt pins sets the pending flags of an interrupt flag register. If the interrupt is enabled, control is then transferred to the appropriate interrupt service routine. Maskable interrupts can be enabled and disabled by setting or clearing the corresponding bits in the interrupt enable register.

Each interrupt has a 32-byte space (the size of one fetch packet) in the interrupt vector table where the interrupt service routine is stored. If the service routine is larger than 32 bytes, the routine must branch out of the table.

> *A performance penalty will often be encountered if the service routine branches out of the table. This is because the programmer is forced to fill the five delay slots of the branch within the same fetch packet as the branch. This severely limits the number of instructions that can be executed in parallel in the delay slots of the branch.*

The service packet for the reset interrupt is always located at address zero. The "interrupt service table pointer" (ISTP) determines the starting location of the service table for the other interrupts and can point to any memory location on a 1,024-byte boundary.

Upon interrupt, the return address is stored to the "interrupt return pointer" (IRP), or "NMI return pointer" (NRP) register. In addition, interrupts are disabled and control is transferred to the corresponding service routine. The minimum interrupt latency from the assertion of the interrupt line until the execution of the first instruction of the service routine is 12 instruction cycles. The processor is not interruptible while any execution packet in the pipeline contains a branch or is in the delay slot of a branch. Additionally, interrupts can cause unexpected program behavior due to the exposed pipeline. For example, load instructions have a latency of five cycles. Thus, the result of a load into register A0 executed at cycle N is available in A0 at cycle N+4. An arithmetic instruction can still read the old contents of register A0 until cycle N+3. Suppose, however, that an interrupt occurs at cycle N+1. The delay caused by the interrupt means that once the is serviced and the pipeline returns to executing the arithmetic instruction, the arithmetic instruction reads the new value of A0 instead of the old value of A0. To avoid such problems, interrupts must often be disabled when executing efficient code.

> *The fact that the processor is not interruptible while a branch is pending means that tight loops are not interruptible on the TMS320C62xx. In applications where interrupt latency is a con-*

cern, programmers will often have to make significant sacrifices in performance in order to ensure reasonable interrupt latency. The fact that interrupts can interfere with the TMS320C62xx's exposed pipeline and cause unexpected program behavior further exacerbates this problem.

An interrupt service routine is exited using dedicated branch instructions. These instructions re-enable interrupts and use IRP or NRP as a return address. Interrupts are not automatically nestable on the TMS320C62xx. In order to nest interrupts, the IRP and NRP must be saved and interrupts must be re-enabled by the service routine.

Stack

The TMS320C62xx does not provide a hardware stack.

A software stack can be implemented using any general-purpose register as a stack pointer. Push and pop operations can be implemented using load and store instructions with pre- or post-increment or decrement.

Bootstrap Loading

The TMS320C62xx provides bootstrap loading capability via its DMA controller and host port (HPI) or via the expansion bus that replaces the HPI for some TMS320C62xx members. (The DMA controller, HPI, and expansion bus are discussed in detail in the *Peripherals* section below.) Additionally, the TMS320C6211 processor can boot directly from external SDRAM (this option is referred to as the "no boot process").

For DMA bootstrap loading, the DMA controller may be configured to load the internal program memory from 8-, 16-, or 32-bit wide external memory. Except for the TMS320C6211 processor, when bootstrap loading via the DMA controller, the DMA controller loads 16 Kwords from CE1 memory space to internal program memory starting at address 0, after which the processor starts executing automatically (the enhanced DMA of the TMS320C6211 loads 256 32-bit words from CE1).

When bootstrap loading via the HPI, it is up to the host processor to load the internal program memory and signal the TMS320C62xx when bootstrap loading is complete. After bootstrap loading via either the DMA controller, the HPI, or the expansion bus, the processor begins execution at address 0.

The bootstrap loading options are flexible in terms of supported memory type, speed, and width.

Peripherals

The TMS320C6201 and TMS320C6211 include a host port, a multi-channel DMA controller, a multi-channel buffered serial port, and two 32-bit timers. The TMS320C6202, TMS320C6203, and TMS320C6204 include the same peripherals as the TMS320C6201, except the host port has been replaced with the more sophisticated

"Expansion Bus." The TMS320C6205 features the same peripherals as the TMS320C6201, except the host port has been replaced with a PCI interface.

- **Host Port (HPI)**

  The TMS320C6201 and TMS320C6211 provide a 16-bit parallel host port interface. Through this port, an external device (host) can access the internal memory of the TMS320C62xx.

  The HPI contains a 32-bit control register, a 32-bit data register, and a 32-bit address register. The host can read and write each of these 32-bit registers by performing two 16-bit reads or writes over the 16-bit parallel interface. To perform a read or write from or to the TMS320C62xx's memory space, the host must first write the desired memory address into the HPI address register, then read or write the HPI data register. Data transfers between the HPI data register and memory occur over the DMA controller's auxiliary channel. The HPI address register can be configured to be automatically incremented to the next word address when a read or write of the HPI data register occurs.

  The host can interrupt the TMS320C62xx by setting a bit in the HPI control register. Similarly, a bit in the HPI control register is provided to enable the TMS320C62xx to interrupt the host processor.

- **Expansion Bus**

  Instead of the 16-bit host port interface, the TMS320C602, TMS320C6203, and TMS320C6204 processors include a more sophisticated 32-bit expansion bus. Like the HPI, this 32-bit interface allows external device accesses to the TMS320C62xx internal memory space. The expansion bus can operate in synchronous or asynchronous mode. In asynchronous mode, the expansion bus operates like the 16-bit HPI of the TMS320C6201 processors, although the throughput is improved via a wider 32-bit bus. Compared to the TMS320C6201, the added synchronous mode allows the TMS320C6202, TMS320C6203, and TMS320C6204 to be readily interfaced with a broader range of external devices including other processors, FIFOs, and PCI bridges.

- **PCI Interface**

  The TMS320C6205 replaces the host port with a 32-bit PCI interface. This interface is compliant with Revision 2.2 of the PCI Local Bus specification and supports 33 MHz master and slave targets. The interface features parity generation, error detection and reporting, and power management capabilities. An on-chip EEPROM interface is embedded in the TMS320C6205 to allow loading a PCI configuration from an external serial EEPROM.

- **DMA Controller**

  With the exception of the TMS320C611 which supports 16 channels, all current TMS320C62xx processors provide a four-channel DMA controller. The DMA controller can move data to or from any part of the TMS320C62xx address space, including on- and off-chip memory and peripherals.

Each of the four channels can be independently configured to transfer 8-bit, 16-bit, or 32-bit data elements. DMA transfers are performed on blocks of data, which are subdivided into an arbitrary number of frames, each containing an arbitrary number of data elements. The number of bytes between two blocks and the number of bytes between two frames (both offsets are referred to as a "stride") must be configured as part of the transfer request parameters. The support for strides allows the DMA controller to automatically generate the source or destination addresses that occur during relatively complex data transfers. For example, a two-dimensional array contained in another larger two-dimensional array can be automatically extracted by splitting the information into frames and blocks and setting up the strides appropriately. Because of this capability, the DMA controller can be characterized as supporting multi-dimensional transfers. Additionally, apart from the address calculations that occur during a transfer, the source and destination addresses of a transfer can be automatically updated upon transfer completion.

Each channel can be configured to interrupt the CPU at the end of a frame or block transfer. Additionally, each channel can automatically re-initialize itself at the end of a block transfer for continuous operation.

A split-channel mode allows one of the DMA channels to handle simultaneous input and output streams for the same peripheral (full duplex mode), if the peripheral uses a destination address equal to the source address +4 bytes.

DMA element or frame transfers can be automatically synchronized to a variety of events such as timer, serial port, or external interrupts. The DMA controller also features programmable channel priorities.

A fifth, auxiliary DMA channel is used by the HPI or the expansion bus (described above) to allow an external host to access the TMS320C62xx memory space.

The TMS320C6203, TMS320C6204, and TMS320C6205 DMA controller is improved by the addition of separate internal buffers (FIFOs) used for each DMA channel in place of a single buffer for all of the channels as found on the TMS320C6201 and TMS320C6202. The features are the same as those of the DMA controllers on the TMS320C6201 and TMS320C6202, and programs written for the TMS320C6201 and TMS320C6202 can run unmodified on the TMS320C6203, TMS320C6204, or TMS320C6205 with respect to the use of DMA. This improves the overall throughput of transfers when multiple channels are used in parallel to access the peripherals (e.g., the external memory interface, the serial ports or the expansion bus). The DMA controller processes one channel at a time but a transfer performed through a higher priority channel can interrupt a transfer occurring in a lower priority channel. The overhead from switching channels is reduced when multiple FIFOs are used to convey the atomic read and write requests.

The DMA controller of the TMS320C6211 is referred to as an enhanced DMA controller (EDMA) by Texas Instruments. The EDMA controller differs from the other TMS320C62xx members in several ways. First, it supports sixteen channels instead of four. Second, the EDMA controller provides the ability to link transfers; i.e., once a transfer completes, another one can automatically be started.

> *The support for linked DMA requests can increase performance since the core isn't required to intervene when two consecutive transfer requests are issued. This somewhat mitigates the drawback that the TMS320C62xx is rarely in an interruptible state in many applications.*

Third, instead of the configuration registers that are used to configure DMA requests on the other TMS320C62xx processors, DMA transfer request parameters are gathered in a separate 2 Kbyte on-chip memory section of the TMS320C6211 memory map. Fourth, the EDMA controller of the TMS320C6211 provides a "quick" transfer mode that reduces the set-up latency required to begin a DMA transfer. Five cycles are required to submit a quick DMA request rather than 36 for a conventional DMA request.

> *The quick DMA mode supported by the enhanced DMA controller supports DMA transfers with minimum overhead. This feature is most useful when the location of the data to process is determined at run-time. For example, when compensating block motion in MPEG-like video decompression algorithms, the location of the blocks to be reconstructed is determined during run-time processing and depends on the contents of the compressed data stream being decoded. This type of run-time dependency defines "data-dependent" programs.*

Finally, the EDMA controller features three transfer-request queues that allow the controller to multiplex groups of atomic read and write accesses (or burst accesses) of up to three pending transfers with minimum contention when switching from burst accesses. This feature further improves the sophisticated DMA controller of the TMS320C6203, TMS320C6204, and TMS320C6205 in the sense that transfers in the three channels are performed concurrently. (In comparison, only one channel can operate at a time on the TMS320C6201 and TMS320C6202, although a higher-priority channel can interrupt a transfer occurring in a lower priority channel.) Additionally, although the EDMA allows for concurrent transfers, it also implements three priority levels for each of the concurrent transfers. Each EDMA queue is given a fixed priority level. Transfers initiated by a host or by TMS320C62xx software can be configured to use one of the two lowest-priority queues. The highest-priority queue (the "urgent" priority queue) is reserved for level-2 cache controller requests.

The structure of the EDMA is unique in the industry. The three-level priority queues allow balanced bandwidth usage when accessing the external memory interface, the host port, or the serial ports concurrently rather than allowing a single channel to monopolize the entire bandwidth.

The TMS320C6211 EDMA controller has the same multi-dimensional transfer support as the DMA controller of the other TMS320C62xx members; transferred data is grouped per block, frame, and elements.

In several ways, the EDMA controller of the TMS320C6211 resembles the DMA found on the now-obsolete TMS320C8x multi-processor DSPs (three-dimensional DMA, parameter RAM to hold transfer requests, support for linked transfer requests), although the TMS320C8x supported additional transfer modes intended for image processing applications.

The EDMA controller of the TMS320C6211 supports the same ability to synchronize with a variety of internal and external events as other TMS320C62xx family members. Sixteen external events are supported, each of which is associated with a DMA channel on the TMS320C6211. Like the TMS320C62xx DMA controller, the event synchronization support of the EDMA controller can trigger submission of an entire DMA request or the transfer of an element or block.

The TMS320C62xx DMA and EDMA controllers are very flexible. The ability of the DMA controllers to synchronize to peripheral or external interrupts is extremely useful, and somewhat mitigates the drawback that the TMS320C62xx is rarely in an interruptible state in many applications.

- **Serial Ports**

The serial ports of the non-cache based TMS320C62xx family members (all devices except the TMS320C6211) are identical. The TMS320C6201, TMS320C6204, and TMS320C6205 provide two serial ports. The TMS320C6202 and TMS320C6203 provide three serial ports. The TMS320C6202B will also provide three serial ports, according to Texas Instruments.

Each serial port is capable of transferring 8-, 12-, 16-, 20-, 24-, or 32-bit words. The receive and transmit sections of each serial port can be independently configured to use internally generated or externally supplied bit clock and frame sync signals. The polarity of the clock and frame sync signals can be independently configured for receive and transmit. Only one clock and frame sync generator is provided per serial port, so if the receive and transmit sections of a serial port are both configured to use internally generated clock and/or frame sync, the two sections must share the same clock and frame sync signals. The internally-generated bit clock can be derived from the CPU clock or from an externally supplied clock by dividing it by a programmable 7-bit value.

The serial ports support automatic A-law or μ-law companding for 8-bit transfers. The serial ports support multi-channel time-division-multiplexed operation. Up to 128 logical TDM channels are supported. Up to 32 channels, selected from one of eight groups of 32 channels each, can be selected to be active at one time, or all 128 channels can be active at once.

Using the DMA controller (described above), data received by the serial ports can automatically be stored to a memory buffer without CPU intervention. Similarly, the DMA controller can automatically fetch data for transmission over the serial ports.

The serial port pins can be used as general-purpose bit-I/O pins when the serial ports are not in use.

The two serial ports found on the TMS320C6211 are similar to those found on the other TMS320C62xx processors. One key difference is that the TMS320C6211 serial ports can be configured for LSB-first or MSB-first reception and transmission.

> *The serial ports on the TMS320C62xx are flexible and support various communication protocols (e.g., I^{2}S, AC97, SPI, ST-BUS and IOM2). The serial ports of the TMS320C62xx processors are very similar to those of the TMS320C54xx processors.*

- **Timers**

  All current TMS320C62xx family members include two identical 32-bit timers. The timers can be clocked by the CPU clock divided by four, or can count external events. Each timer has an input pin and an output pin. The input pin is used for counting external events. The output pin can be configured to output a pulse with a width of one or two CPU clock cycles when the timer's count register reaches the timer period. Alternatively, the output pin can be configured to toggle its state when the count register reaches the timer period. When the timer is not in use, the timer input pin can be used as a general-purpose input pin and the timer output pin can be used as a general-purpose output pin.

  When the timer period is reached, the count register is automatically reset to zero and the CPU can be interrupted.

On-Chip Debugging Support

The TMS320C62xx offers a JTAG-based interface to on-chip emulation and debugging circuitry. Through the JTAG interface, an external device can read and write processor memory and registers, set and clear hardware breakpoints, and single-step through a program. The JTAG port can also be used for boundary scan.

The JTAG interface can also be used to convey a small amount of information in real-time (such as text messages or data to display). Texas Instruments refers to this feature as the Real-Time Data Exchange (RTDX). When coupled with the use of Texas Instruments' DSP-BIOS real-time operating kernel (further discussed in the *Development Tools* section), the JTAG interface can be used to provide the programmer with kernel information such a message log or a task-switching trace.

Power Consumption and Management

The typical power consumption for the various TMS320C62xx members is presented in Table 7.15-5.

The TMS320C62xx features three low-power modes that can be entered by setting the power-down control bits in the control status register. In IDLE1 mode, the core is disabled but the PLL and peripherals continue to operate and may wake up the core. In IDLE2 mode, the core and the peripherals are disabled, but the PLL stays active. The interrupt that wakes up the processor in this case must come from an external device. IDLE3 mode completely shuts down the processor. In this mode the core and peripherals are disabled and the PLL is halted. The only way to exit IDLE3 mode is to reset the processor.

Note that the IDLE instruction is not used to enter the low-power modes described above. Instead, the IDLE instruction causes the processor to start executing a multicycle NOP instruction. This instruction can only be terminated by an interrupt.

The TMS320C6202, TMS320C6203, TMS320C6204, and TMS320C6205 provide the ability to disable each peripheral, including the serial ports, the EMIF interface,

Device	Frequency	Typical power consumption
TMS320C6201 (1.8V)	200 MHz	1.3 W
TMS320C6202	200 MHz	1.7 W
TMS320C6202	250 MHz	2.1 W
TMS320C6211	150 MHz	0.9 W
TMS320C6203	250 MHz	1.1 W
TMS320C6203	300 MHz	1.3 W
TMS320C6204	200 MHz	0.8 W
TMS320C6205	200 MHz	0.8 W

TABLE 7.15-5. Typical TMS320C62xx power consumption summary.

and the DMA controller in addition to the three IDLE modes supported on the TMS320C6201 and TMS320C6211.

Benchmark Performance

The TMS320C62xx has been benchmarked with the BDTI Benchmarks™. Overall benchmark results for all benchmarked processors are presented in Chapter 8, *BDTI Benchmark™ Results*. We summarize and analyze TMS320C62xx benchmark performance in the paragraphs below. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not take into account the processor's instruction cycle rate. Therefore, lower instruction cycle counts imply a more powerful architecture, but do not imply better performance. Next we discuss benchmark execution times and cost-execution time products, indicating processor performance and cost-performance, respectively. We then discuss the processor's energy consumption, which reflects the energy consumed by the processor in order to perform a task. Finally, we discuss the processor's memory usage. We divide the memory usage discussion into three parts: Control benchmark memory usage, overall benchmark program memory usage, and benchmark data memory usage.

Except for the TMS320C6211, TMS320C62xx processors have on-chip program RAM that can be configured to operate as a program memory or as direct-mapped instruction cache (the TMS320C6211 program memory is always configured as cache). The benchmark results presented here and in Chapter 8 assume that the on-chip program RAM is configured as program memory and that the benchmarks are pre-loaded and executed from this memory. In addition, all data are assumed to be pre-loaded in the on-chip data memory. Thus, these results neglect penalties associated with cache misses, which can be significant. BDTI has attempted to estimate the effect of cache miss penalties on the TMS320C6211 and has measured several of the BDTI Benchmarks on a TMS320C6211 development board. Although we do not present detailed results for each benchmark that include cache miss penalties, this analysis does include a general assessment of the performance impact for the case where the TMS320C6211 L1 cache is flushed at the inception of the benchmark.

TMS320C62xx performance strongly depends on the use of on-chip memory. For example, when executing instructions exclusively from off-chip memory, even if the fastest type of off-chip memory is used (full-speed synchronous burst SRAM), the TMS320C62xx's maximum instruction execution rate is slowed by a factor of eight. Thus, if there is a need to use external memory for program and/or data in a potential application for the TMS320C62xx, we urge readers to carefully consider how the processor's external memory interface and on-chip instruction cache will perform in that application. Relatedly, the impact of the TMS320C6211 two-level cache architecture on performance varies for each benchmark and can some-

times significantly increase the cycle counts discussed in this report.

Execution Performance

- **Instruction cycle counts:**

On all BDTI Benchmarks, the TMS320C62xx cycle count is among the five lowest results. As illustrated in Figure 8.1-13, the total normalized instruction cycle count for the TMS320C62xx is third-lowest for all benchmarked processors, at about 40% below the average. However, its result is roughly 20% higher than that of the TMS320C64xx-C (the TMS320C64xx with L1 cache preloaded) and about 70% above that of the SC140.

The TMS320C62xx has low cycle counts primarily because it contains a total of eight independent execution units and can execute up to eight instructions per instruction cycle. The TMS320C62xx is particularly effective in convolution-oriented benchmarks that process blocks of data, such as the Real Block FIR filter. The processor's cycle counts on the single-sample benchmarks are not as low, as it cannot make full use of its parallelism and it suffers from pipeline delays.

The only two processors that consistently achieve lower cycle counts on most of the BDTI Benchmarks are the SC140 and the TMS320C64xx-C. The TMS320C64xx is an enhanced version of the TMS320C62xx, adding a number of new instructions and additional hardware. The TMS320C64xx-C's lower instruction cycle counts are largely due to three key extensions: new dot-product instructions, application-specific instructions (such as the bit-reversing instruction used in the FFT benchmark and the bit de-interleaving operation used in the Viterbi benchmark), and a wide range of added arithmetic and logical SIMD instructions. The SIMD extensions include support for four 16-bit multiplies in parallel—twice as many as the TMS320C62xx.

Compared to the TMS320C62xx, the SC140 has a lower cycle count on all benchmarks (much lower on many benchmarks) due to three key factors. First, the SC140 can execute four MACs in parallel (compared with two on the TMS320C62xx). Second, the SC140 has a shallower pipeline and single-cycle latencies for most operations, which is particularly advantageous on single-sample benchmarks. Finally, the SC140 provides a powerful fractional-data store instruction capable of storing the upper 16-bit part of the product of a fractional $16 \times 16 \rightarrow 32$-bit multiplication. In comparison, the TMS320C62xx does not provide this feature and, in many cases, requires that multiplication products be explicitly shifted to select the desired portion.

The TMS320C62xx cycle counts are notably low on the Real and Complex Block FIR filter, Vector Add, Vector Maximum, FFT, Viterbi, and Bit Unpack benchmarks.

The TMS320C62xx achieves instruction cycle counts on the **Real Block FIR** and **Complex Block FIR** benchmarks that are roughly 45% and 40% lower, respectively, than the average for all of the benchmarked DSP processors. The TMS320C62xx's key advantages on these benchmarks are its high on-chip memory bandwidth and high level of parallelism, which enable the processor to compute two filter taps per instruction cycle. In addition, the TMS320C62xx makes effective use of its ability to conditionally execute instructions to eliminate the housekeeping overhead associated with loading the next input value, resetting the convolution loop counter, and shifting and storing the previous result. The inner (convolution) loop is combined with the outer (housekeeping) loop by conditionally executing the housekeeping instructions within the convolution loop; new input samples are loaded during the last iterations of the convolution loop while the previous results are saved during the first iterations of the next convolution loop's execution (the convolution loop is executed one extra time to load the new input samples).

Although it can compute two FIR filter taps in a single cycle, on the **Single-Sample FIR** benchmark the TMS320C62xx takes two cycles to compute two taps. The additional cycle is used for moving data to update the filter delay line FIFO. An alternative implementation would avoid moving data by using circular buffering. Moving data to update the delay line allows the setup time for the main loop to be significantly reduced, however, resulting in an implementation that is slightly more efficient for modest numbers of filter taps and that is therefore preferred for this benchmark.

On single-sample benchmarks (such as the Single-Sample FIR filter or the LMS Adaptive filter), setting up the main loop takes a relatively large number of cycles on the TMS320C62xx compared to the inner loop cycles required. For this reason, the TMS320C62xx cycle counts are relatively high compared to those on block-oriented benchmarks, but are still approximately 20-30% lower than the average of the benchmarked DSP processors. This also applies to the **Two-Biquad IIR Filter**, although the TMS320C62xx implementation of this benchmark doesn't use a loop (the implementation is optimized for two biquads and is completely unrolled). For these three single-sample benchmarks, the TMS320C62xx suffers from its deep pipeline and the multi-cycle latencies of some of its key instructions (such as loads, multiplies, and branches). Unlike block-oriented benchmarks, which can amortize the multi-cycle latencies via optimization techniques such as software pipelining, the single-sample benchmark cycle counts are strongly affected by the processor's pipeline depth and long latencies.

On the **Vector Add** benchmark, the TMS320C62xx's instruction cycle count is about 40% lower than the average of all benchmarked processors. Due to the processor's high on-chip data memory bandwidth, four 16-bit vector elements can be read or written at a time. Together with the two parallel data paths and the use of dual 16-bit SIMD addition instructions, this enables the TMS320C62xx to process four vector elements every three clock cycles.

On the **Vector Maximum** benchmark, the TMS320C62xx's instruction cycle count is roughly 60% lower than the average for all benchmarked processors. Parallelism and conditional execution are the key TMS320C62xx features on this benchmark; two comparisons and two conditional assignments are performed in parallel.

On the **FFT** benchmark, the TMS320C62xx achieves an instruction cycle count that is roughly 60% below the average of all benchmarked processors. The TMS320C62xx executes one radix-4 butterfly in just ten cycles. The dual 16-bit SIMD add and subtract instructions (ADD2 and SUB2) are used to speed up the loop kernel. An average of six parallel instructions per cycle (out of a possible eight) are executed in this inner loop, demonstrating good resource usage.

On the **Viterbi** benchmark, the TMS320C62xx has the fifth lowest cycle count result, at roughly 60% below the average of all benchmarked processors. On this benchmark, the TMS320C62xx performs the traceback loop using only three clock cycles per bit. Conditional execution and the bit-extraction instruction are used to optimize the implementation of the benchmark. Also, a specific addition instruction, the "addressing mode" addition, is used to further reduce the number of cycles. This instruction is primarily intended for incrementing a pointer by an offset that is automatically scaled (multiplied) by a factor of two or four depending on the width of the data referred to by the pointer. This instruction combines a shift and an add significantly reducing the cycle count for the traceback loop.

On the Viterbi benchmark, the TMS320C64xx-C and the SC140 achieve cycle counts more than three times lower than that of the TMS320C62xx, for two key reasons. First, the SC140 and the TMS320C64xx-C feature dedicated instructions for the Viterbi algorithm (bit-interleaving and de-interleaving instructions on the TMS320C64xx-C, special dual 16-bit SIMD maximum instructions associated with a conditional shifting instruction on the SC140). The TMS320C64xx-C also uses an unusual optimization that allows it to take advantage of its quad 8-bit SIMD capabilities; the TMS320C64xx uses quad 8-bit SIMD additions, quad SIMD compares, and quad SIMD maximum instructions in the bit-interleaving loop of the Viterbi benchmark (also known as the add-compare-select loop), resulting in an extremely efficient implementation. The TMS320C62xx does not support 8-bit SIMD operations.

The TMS320C62xx has low cycle counts on the **Bit Unpack** benchmark primarily because of its high level of parallelism and support for conditional execution, discussed earlier. The TMS320C62xx lacks support for unaligned 64-bit data accesses, however, and this is the main reason why its cycle count on this benchmark is higher than that of the TMS320C64xx-C and about equal to that of the SC140. The floating-point TMS320C67xx uses the same implementation of this benchmark as the fixed-point TMS320C62xx, since floating-point operations are not useful in this benchmark. Hence, the cycle counts of the two processors are equal.

The results discussed above do not include cache miss penalties for the TMS320C62xx. To estimate the effect of L1 cache miss penalties, BDTI measured the cycle counts for several benchmarks on a TMS320C6211 development board. For these measurements, we flushed the level-1 instruction and data cache but pre-loaded the unified level-2 cache with data and instructions.

> *Focusing only on the level-1 cache impact is reasonable for many applications, since the level-2 memory can be configured to serve as cache or on-chip RAM. If used as on-chip RAM, data and instructions can be pre-fetched into level-2 memory with a minimum cycle penalty using the DMA controller.*

BDTI measured the effect of L1 cache misses on the Vector Dot Product, Real Block FIR filter, FFT, and Viterbi benchmarks. Note that the level-1 instruction and data cache sizes of the TMS320C6211 are sufficiently large (4 Kbytes each) to contain all of the instructions and data for these benchmarks. Additionally, the benchmarks are optimized to avoid cache conflicts. Therefore, the level-1 cache impacts are proportional to the memory usage and the cycle count of each benchmark (and not to the memory access patterns of the benchmarks).

On the Vector Dot Product benchmark, the combined level-1 data and instruction cache miss penalties of the TMS320C6211 roughly double the cycle count. In the absence of cache effects, this benchmark has a relatively low cycle count of 31 cycles. On the Real Block FIR filter, which has a higher cycle count than the Vector Dot Product (348 cycles without cache impact), the level-1 cache miss penalties is lower; i.e., about 20%. The FFT benchmark has a cycle count of 2,492 cycles when cache misses are neglected; the effect of level-1 cache miss penalties increases this cycle count by roughly 25%. Finally, cache miss penalties increases cycles by less than 5% for the Viterbi benchmark.

> *Our limited analysis of the effect of L1 cache misses indicates that the cache miss penalty can be significant (e.g., a 25% increase in cycles for the FFT, and a 100% increase for the Vector Dot Product). Therefore, we urge readers to carefully consider cache miss penalties when estimating the performance of the TMS320C6211.*

- **Execution times:**

Among the processors benchmarked, the TMS320C6203 shares the second-highest instruction cycle rate, 300 MHz, with the Motorola MSC8101. Only the 600 MHz projected instruction cycle rate of the TMS320C64xx is faster. The low cycle counts of the TMS320C62xx combine with its high instruction cycle rate to give a total normalized execution time that is roughly 4.2 times faster than the average for the fixed-point processors. The execution time results for the TMS320C6203 is

60% slower than that of the MSC8101, however, and nearly 2.5 times slower than the TMS320C64xx-C, as shown in Figure 8.2-13.

- **Cost-execution time:**

  The TMS320C6203 is the most expensive processor here ($201.42 in quantity of 10K for the 300 MHz TMS320C6203). Its price is roughly twice that of the second most expensive fixed-point processor, the $96 MSC8101. The TMS320C6203's fast execution times do not compensate for its high cost. The TMS320C6203's total normalized cost-execution time is roughly 1.7 times worse than the average for the fixed-point processors, and roughly 3.5 times worse than that of the MSC8101.

  The TMS320C6203's speed and price data are used to generate the cost-execution time result for the TMS320C62xx family. It should be noted, however, that the cost-execution time result for the TMS320C6211 would be significantly better, since the TMS320C6211 costs seven times less than the TMS320C6203 while running half as fast (neglecting cache penalties).

- **Energy consumption:**

  The TMS320C6204 uses a low, 1.5-volt core supply, but still has relatively high power consumption (0.8 watts). As illustrated in Figure 8.4-13B, the TMS320C6204 total normalized energy consumption is roughly 1.8 times higher than the average for the fixed-point processors. (Note that the TMS320C64xx and ADSP-219x are excluded from this average since power consumption figures have not been disclosed for these processors.) The TMS320C6204 result is roughly eight times higher than that of the MSC8101, which has exceptionally low energy consumption.

Memory Usage

The focus in the memory usage analysis is on Control benchmark memory usage. Unlike other benchmarks, the Control benchmark is optimized for minimum memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP code, and constant and non-constant data memory usage.

As mentioned earlier, TMS320C62xx instructions are 32 bits wide. Each 32-bit instruction is a simple, RISC-like instruction, and up to eight parallel 32-bit instructions can be executed in each instruction cycle. Since each instruction is quite simple, the TMS320C62xx often requires multiple instructions to perform the same work as one instruction on a conventional DSP processor. In addition, making efficient use of the processor's multiple execution units often requires programmers to issue multiple instances of the same instruction (e.g., to issue two multiply instructions in parallel) and to use optimization techniques such as loop unrolling (in which the loop body is replicated one or more times inside the loop structure). These optimizations increase speed at the cost of increased memory usage. Relatedly, the TMS320C62xx branches, loads, and multiplies introduce five, four, and one delay slots, respectively, encouraging the use of software pipelining to avoid wasting cycles waiting for results. Filling and flushing deep software pipelines often requires many instructions, thus increasing code size.

- **Control benchmark memory usage:** The Control benchmark implementation for the TMS320C62xx is identical to that of its floating-point counterpart, the TMS320C67xx. As illustrated in Figure 8.5-9A, the TMS320C62xx's Control benchmark memory usage is roughly 50% higher than the average for fixed-point processors. The floating-point ADSP-2106x and ADSP-2116x (which have 48-bit instructions and share their benchmark implementation) have the highest total memory usage on this benchmark. The processor's high Control benchmark memory usage is primarily due to the processor's 32-bit instruction width and the fact that TMS320C62xx instructions are fairly simple. The only other processor with 32-bit instructions, the TMS320C64xx, has a total memory usage result that is roughly 15% lower than that of the TMS320C62xx.

  Because the Control benchmark is optimized for memory usage rather than speed, the processor does not use a deep software pipeline in this benchmark. However, without deep software pipelining, multi-cycle NOP instructions must sometimes be used to fill branch delay slots.

- **Program memory usage:** The TMS320C62xx has the highest program memory usage of all of the benchmarked fixed-point DSPs in eight out of twelve of the BDTI Benchmarks. As illustrated in Figure 8.5-13, the total normalized program memory usage for the TMS320C62xx is about twice the average of all of the processors.

  The main reasons for the TMS320C62xx's high program memory usage were discussed earlier, and include the processor's wide, RISC-like instructions and the fact that many of the TMS320C62xx benchmark implementations use speed optimizations (such as loop unrolling) that result in higher memory use. For instance, a MAC operation takes 12 bytes (three32-bit instructions: a multiply, an add, and a shift) on the TMS320C62xx versus 2 bytes on most 16-bit fixed-point processors. In addition, as discussed earlier in this chapter, TMS320C62xx execute packets must be entirely contained within fetch packets. The assembler automatically inserts parallel NOPs as necessary to prevent execute packets from crossing etch packet boundaries. This doesn't affect cycle counts, but it contributes to the processor's high program memory use.

- **Data memory usage:** Constant data memory usage of the TMS320C62xx is as expected for a 16-bit fixed-point processor. Normalized constant data memory usage is shown in Figure 8.5-14.

  The TMS320C62xx non-constant data memory usage is slightly higher than the average for 16-bit fixed-point DSP processors, as presented in Figure 8.5-15. The TMS320C62xx's non-constant data memory use is higher as a result of optimizations made to increase speed on some benchmarks. For example, the LMS Adaptive filter benchmark implementation uses a coefficient buffer that is twice the normal size. The extra buffer size is used to store invalid coefficient values generated on the first iteration of the loop as a result of software pipelining. Some benchmarks, the non-constant data memory usage is slightly above the average of all benchmarked 16-bit fixed-point processors because the TMS320C62xx uses 32-bit memory addresses and requires four bytes each time it stores a pointer in memory.

  As mentioned earlier, the TMS320C62xx provides several features to reduce program memory use. These features include variable size execution packets and conditional execution of all instructions. In spite of these features, the TMS320C62xx program memory use on DSP algorithm code is dramatically higher than that of the other processors in this report. However, its total memory usage on the Control benchmark is only 30% higher than that of the other fixed-point processors, and in general, a processor's code density on control code is more important than its code density on DSP algorithm code.

Cost

Price and packaging options for TMS320C62xx processors are shown in Table 7.15-6.

Fabrication Details

The 1.8 volt TMS320C6201, TMS320C6202, and TMS320C6211 are fabricated using a 0.18 μm five-metal-layer CMOS process. The TMS320C6203, TMS320C6204, and TMS320C6205 are fabricated using a 0.15 μm five-metal-layer CMOS process. The TMS320C6202B will be fabricated using a 0.15 μm five-metal-layer CMOS process, according to Texas Instruments.

Development Tools

Texas Instruments provides a COFF-based C/C++ compiler, assembler, linker, archiver, instruction-set simulator and emulator for the TMS320C62xx. The same tool set is shared by all of the TMS320C6xxx processors: the TMS320C62xx, TMS320C67xx, and TMS320C64xx. The assembler, linker, archiver, C/C++ compiler, assembler and simulator are command-line-driven tools. They are available for PC-compatible computers under Microsoft Windows 9x, Windows NT, and Windows 2000, and for Sun SPARC computers under Solaris. The tools can also be used from within the Code Composer Studio integrated development environment; Code Composer Studio is discussed further below.

Compiler

Version 4.0 of the compiler adds support for C++, covering most of the latest C++ extensions. The support for C++ is not complete, however; for example, the C++ exception-handling mechanism is not currently supported. Texas Instruments states that future versions of the compiler will expand support for C++.

Support for C++ is not common among DSP processors, though several vendors (including StarCore and Analog Devices) have recently announced C++ compilers. The use of C++ often results in increased code size in comparison to code generated using C, without necessarily providing faster speed. This combination has historically made C++ less attractive for DSP processors, which are often used in speed, cost, and memory-sensitive applications.

Device	Speed (Millions of MACs per second)	Voltage (V)	Package	Price (Qty. 10,000)
TMS320C6201-200	400	1.8 / 3.3[1]	352 BGA	$81.64
TMS320C6202-200	400	1.8 / 3.3[1]	352/384 BGA	$96.78
TMS320C6202-250	500	1.8 / 3.3[1]	352/384 BGA	$129.12
TMS320C6203-250	500	1.5 / 3.3[2]	352/384 BGA	$154.94
TMS320C6203-300	600	1.5 / 3.3[2]	352/384 BGA	$201.42
TMS320C6204-200	400	1.5 / 3.3[2]	288 BGA	$35.14
		1.5 / 3.3[2]	340 BGA	$42.17
TMS320C6205-200	400	1.5 / 3.3[2]	288 BGA	$47.14
TMS320C6211-150	300	1.8 / 3.3[1]	384 BGA	$28.74

TABLE 7.15-6. TMS320C6201 price and package summary. Prices as of June 2000.

[1] The core operates at 1.8 volts while all I/O signals are 3.3-volt compatible.

[2] The core operates at 1.5 volts while all I/O signals are 3.3-volt compatible.

The compiler supports a variety of directives to enable the programmer to better control characteristics of the generated code. For example, as discussed earlier, the TMS320C62xx can rarely be interrupted during optimized small loops (see *Interrupts* section). To ensure that software is periodically interruptible, programmers can use a compile-time directive in their C/C++ source code. This directive tells the compiler to generate code that allows interrupts to be serviced after a specified maximum number of cycles. To achieve this goal, the compiler adds an interruptible outer loop surrounding the inner loop. In general, this approach reduces speed and increases code size.

Version 4.0 of the C/C++ compiler is also able to automatically unroll a loop; i.e., to replicate the contents of a loop to expose more operations to be executed in parallel. Usually, the effect of this optimization is to improve speed at the cost of increased memory usage. The number of times a loop kernel is replicated is determined by the compiler or set by the programmer using a specific directive.

Other directives can be included in the C/C++ source code to enable the compiler to apply more aggressive optimizations. For example, the programmer can specify that two pointers in a critical tight loop don't point to an overlapping external memory region, allowing the compiler to make use of the fact that no data dependency exists and implement additional optimizations. Similarly, the programmer can specify that a loop will execute a specific number of times to enable the compiler to use this information to reduce the size of the generated code.

Programmers can also use intrinsic functions in C/C++ programs to aid the compiler in generating optimized code. Intrinsic functions force the compiler to use a given specific assembly instruction without disturbing the C/C++ usage of registers. This partially compensates for the fact that the C/C++ compiler isn't always able to determine the best assembly instruction (or instructions) to use. On the other hand, the use of intrinsic functions requires more expertise from the programmer and renders the C/C++ program more difficult to port to other targets.

Texas Instruments' C/C++ compiler for the TMS320C62xx is easy to use but experiments using simple algorithms (e.g., convolution-oriented functions) show that the compiler generated code that was slower than hand-optimized code by about a factor of two. In an experiment using a relatively complex algorithm, the compiler generated code that was slower than hand-optimized code by about a factor of five. This level of efficiency is good considering the complexity of the processor and is better than that of other DSP processor compilers that we have evaluated. Nevertheless, programmers will often need to hand-optimize their software to meet their performance goals. The performance of compiler-generated code can often be enhanced by providing more information to the compiler via compiler directives.

Assembler Optimizer

To achieve maximum performance, programming in assembly language is still required. Recognizing that the complexity of the TMS320C62xx architecture creates challenges for assembly programmers, Texas Instruments developed a new class of tool, the "assembly optimizer," to assist developers in creating optimized assembly code. The assembly optimizer accepts sequential (i.e., not parallelized) meta-assembly code and produces optimized, parallelized assembly code. The input meta-assembly (called "linear" assembly by Texas Instruments) code is distinguished from regular TMS320C62xx assembly code in three ways. First, it uses symbolic names for register variables; the optimizer assigns these variables to registers. Second, the programmer specifies the operation, but not the execution unit; the optimizer assigns operations to execute units. Third, the programmer is free to assume that all meta-assembly instructions have single-cycle latencies, significantly simplifying programming. The assembly optimizer schedules the instructions based on their latencies, assigns variables to registers, and decides what functional unit to use for each instruction.

The assembly optimizer does not detect opportunities to exploit parallelism that are not explicitly expressed in the meta-assembly code. For example, if the meta-assembly implementation of an FIR filter is written to process one filter tap at a time, the assembly optimizer does not optimize the code to process two taps at a time. Thus, to get the best results from this tool, the programmer must be familiar with the TMS320C62xx architecture and the capabilities of the tool, and must tailor the meta-assembly code accordingly.

The assembly optimizer is available for Microsoft Windows 95/98/NT.

> *Texas Instruments' TMS320C62xx assembly optimizer is an innovative tool and represents a step in the right direction for helping programmers develop optimized DSP software.*
>
> *The assembly optimizer provides significant assistance for creating optimized assembly code on the TMS320C62xx, but it is not a complete substitute for expert manual crafting of code. Programmers desiring maximum performance will need to understand the full details of the TMS320C62xx architecture and will optimize their code—or at least portions of it—manually. However, due to the complexity of programming the TMS320C62xx, the effort required to manually optimize significant portions of an application is likely to be prohibitive. Therefore, designers of large or complex applications should not assume they will be able to tap the maximum performance of the TMS320C62xx.*
>
> *Code generated by the assembly optimizer is often significantly different from the meta-assembly source and is often difficult to understand. This complicates the debugging and further optimization of code generated with the assembly optimizer.*

Code Composer Studio

Code Composer Studio (CCS) is a DSP-oriented integrated development environment common to the TMS320C5xxx and TMS320C6xxx processors. CCS combines an editor, an integrated debugger (based on the simulator or emulator), a code profiler, a project manager ("makefile" generator and parser), and integration with all of the TMS320C6xxx code-generation tools. Code Composer Studio is available for Windows 9x and Windows NT.

The instruction-set simulator and the emulator share a common graphical interface embedded in the Code Composer Studio integrated development environment. Through this interface, the simulator and emulator support C/C++ language source-level debugging. The simulators for the TMS320C6201, TMS320C6202, TMS320C6203, TMS320C6204, and TMS320C6205 are cycle accurate and properly model external memory accesses in most cases. However, the throughput enhancement added to the DMA of the TMS320C6203, TMS320C6204, and TMS320C6205 (as discussed in the *Peripherals* section) is not modeled by the current simulators. The PCI interface of the TMS320C6205 is also not modeled. Additionally, external accesses to synchronous burst SRAM (SBSRAM) or asynchronous memory are only modeled with 85% cycle accuracy on the TMS320C6211 simulator, according to Texas Instruments.

CCS has several key features to support software development for DSP/embedded applications, described below.

First, CCS's debugging ability is enhanced by the use of advanced breakpoints, which Texas Instruments refers to as "probe points." Probe points make it possible to save or load data once a break point instruction is reached, or to display data sets in various formats (e.g., in the frequency domain, using eye diagrams, or as two dimensional images).

Second, CCS provides an integrated tool for configuring DSP BIOS, the real-time operating system bundled with CCS. DSP BIOS includes optional libraries containing various features (e.g., task communication/synchronization mechanisms, software/hardware interrupt management, I/O programming models). These libraries can be included or excluded from the executable in order to optimize the overall memory footprint of the application. CCS provides a graphical means of selection of the various DSP BIOS components and parameters.

CCS supports Texas Instruments' Real-Time Data Exchange technology (RTDX). RTDX is JTAG-interface-based communication protocol which, together with the DSP BIOS infrastructure, enhances debugging support with the ability to trace task-related information and/or to retrieve message logs via a PC host. Most of the time, the very low overhead associated with the debugging functions allows programmers to leave the calls to the debugging functions in the production version of the code. This feature allows inspection of the on-site and real-context behavior of the final product using a JTAG probe and the CCS environment.

Revision 1.2 of CCS introduces a new tool to profile the compilation process. This tool is able to test various compiler options and graph the speed vs. memory usage characteristics of the resulting code. Using this tool, the programmer can select the compiler option that produces code that best meets the desired speed vs. memory trade-off.

Texas Instruments recently introduced software development guidelines to help third-parties provide ready-to-integrate software libraries. This technology is known as the XDAIS (eXpressDSP Algorithm Interoperability Standard). These rules mainly govern how a module must use and/or declare use of the internal memory space and I/O resources in order to ease integration of modules from multiple sources, including modules provided in object code form. (For example, XDAIS-compliant modules must allow several instances of a given algorithm to exist simultaneously.) To be labeled as XDAIS compliant, a module must be reviewed and certified by Texas Instruments.

The high speed of the TMS320C62xx processors allows them to be used to implement more complex applications, such as multi-channel audio encoders and decoders. To realize the maximum benefit of this powerful but complex architecture, software tools have become increasingly important. In recognition of this, Texas Instruments has invested significant resources in tool development. As a result, Texas Instruments' tool suite and support infrastructure are among the most advanced offered for any DSP processor.

Recently, Texas Instruments has made available two royalty-free assembly-optimized digital signal processing libraries for the TMS320C62xx. The first library is a general-purpose digital signal processing library consisting of 32 high-level DSP functions. The second library contains 22 image processing functions.

In addition, Texas Instruments offers a TMS320C6201-based EVM (EValuation Module) board that fits into a standard PCI slot. With the EVM, an on-board program can be debugged using the debugger function of CCS without the need for a Texas Instruments XDS510 emulator interface. The EVM board can also be used without a PCI connection, using a separate power supply.

Texas Instruments also offers a stand-alone TMS320C6211-based board with a separate power supply. This kit is referred to as the "starter kit" and costs roughly $300. The EVM and the starter kit boards both include various synchronous and asynchronous memories and an audio codec. Third-party daughter boards (such as a video-processing board) can be plugged to EVM boards or to the starter kit.

Third-party support for the TMS320C62xx exists in the form of development boards, emulators, application boards, development tools and software libraries. Third-party vendors are numerous and include Pentek, Hunt Engineering, and Blue Wave Systems. Texas Instruments' website includes a list of third-party companies.

The number of third-party vendors providing tools and software for Texas Instruments' DSPs is the largest of any DSP processor ven-

> *dor. This is due partly to Texas Instruments' early entry into the DSP processor market and to its strong efforts to cultivate third-party support.*

Applications Support

The TMS320C62xx family shares most of its documentation with the TMS320C67xx and the TMS320C64xx. This includes the *TMS320C6xxx CPU and Instruction Set Manual* and the *TMS320C6xxx Programmer's Guide*. These manuals cover the architecture, instruction set, and programming model of the device. The peripherals are described in the *TMS320C6xxx Peripherals Reference Manual.*

> *In several sections, the peripherals documentation (currently labeled SPRU190C by Texas Instruments) is incomplete or insufficiently clear. For example, this document does not provide all of the necessary information required to understand the impact that the TMS320C6211 two-level cache architecture has on performance. Also, this document does not cover all peripheral-related variations associated with the various TMS320C62xx members. Several supplementary application reports discussing each of the processor's details are available on Texas Instruments' website, but the fact that the information is spread over multiple documents complicates understanding the differences between various TMS320C62xx family members.*

Separate data sheets and application reports discuss the hardware aspects of the processors.

Applications support for all TMS320 family processors is provided by Texas Instruments staff who are available via telephone, fax, and electronic mail. Documentation and brief application reports are also available via the World Wide Web.

Advantages

- Highly orthogonal instruction set
- Good parallel move support (two 32-bit or four 16-bit words per instruction cycle)
- Conditional instruction execution
- Four ALUs, two multipliers, and two barrel shifters
- Good on-chip memory bandwidth
- Large number of registers for operands and addressing (thirty-two 32-bit general-purpose registers)
- Exponent detect instruction
- Large, unified address space
- Instruction cache for accelerating off-chip memory accesses

- Flexible external memory interface: ROM, asynchronous SRAM, synchronous burst SRAM, and synchronous DRAM support; DRAM refresh generation
- On-chip DMA controller with dedicated address and data buses
- JTAG emulation port with boundary scan
- Two serial ports, two timers
- Compatible floating-point family (TMS320C67xx)
- Good BDTI Benchmark execution times

Disadvantages

- Two-cycle multiplier latency
- Exposed pipeline complicates programming and interferes with ability to service interrupts
- No bit-reversed addressing
- Off-chip memory accesses take multiple cycles
- Requires execution from on-chip memory/cache for good performance
- Poor Control benchmark memory usage
- Poor BDTI Benchmark program memory usage

7.16 Texas Instruments TMS320C64xx Family

BDTImark2000 Score: Not Available

Introduction

The TMS320C64xx is a VLIW-based, 16-bit fixed-point family of DSP processors from Texas Instruments. Announced in February 2000, the TMS320C64xx is an extension to Texas Instruments' earlier TMS320C62xx architecture. Its instruction set is a superset of the TMS320C62xx instruction set, and adds significant SIMD (single-instruction, multiple-data) processing capabilities, among other enhancements. Texas Instruments has not yet announced any products based on the TMS320C64xx core, but states that the first member of the family will begin sampling in the first half of 2001. Because the chip has not yet been demonstrated in silicon, there is no BDTImark2000 score currently available for this processor. Check BDTI's website (*www.BDTI.com*) for updated BDTImark2000 scores.

Like the TMS320C62xx, the TMS320C64xx family targets high-performance applications such as wireless base stations, digital subscriber loops, multi-line modems, ISDN modems, imaging, 3D imaging applications, and radar and sonar systems. Initial family members are projected to execute at 600 MHz with a 1.2-volt core voltage and 3.3-volt I/O. Table 7.16-1 summarizes characteristics of the initial TMS320C64xx family members. With its combination of increased parallelism and increased clock speed, the TMS320C64xx will be significantly more powerful than its predecessor.

> *When it was introduced in 1997, the TMS320C62xx gained widespread attention because of its unusual architecture and extremely high speed relative to other DSP processors of the time. Its success has been hindered, however, by high memory usage and high energy consumption. With the TMS320C64xx, TI has attempted to address the weaknesses of its earlier architecture, and to offer a processor that is competitive with newer VLIW-based processors, such as the StarCore SC140.*
>
> *Noteworthy enhancements include support for 8-bit SIMD operations, which make the TMS320C64xx very well suited for image processing and image compression/decompression algorithms such as MPEG-2 HDTV video decoding. New application-specific instructions also improve the processor's performance in some traditional audio- and telecom-oriented algorithms.*

Like the TMS320C62xx, the TMS320C64xx uses a VLIW-like architecture with eight execution units, including two multipliers and four ALUs. Using its eight execution units, the processor can execute up to eight 32-bit instructions in a single clock cycle, allowing it to achieve a high level of parallelism. Compared to the TMS320C62xx, the TMS320C64xx adds quad 8-bit and dual 16-bit SIMD arithmetic and logical instructions to most of the execution units, and introduces new quad 8-bit and dual 16-bit vector dot-product instructions to the two multiply units. With this enhancement, the TMS320C64xx is able to perform four 16-bit multiplications in parallel.

In addition, the new architecture improves overall code density by removing some of the constraints associated with packing instructions found in the TMS320C62xx, and by enabling algorithms to be implemented using fewer (more powerful) instructions.

As on the TMS320C62xx, one instruction cycle is equal in length to one master clock cycle on the TMS320C64xx. At 600 MHz, the TMS320C64xx will execute up to 2.4 billion 16-bit fixed-point MACs per second. Since the processor can execute up to eight instructions in every cycle, Texas Instruments characterizes the TMS320C64xx as executing at 4,800 MIPS at 600 MHz.

> *As is the case with the TMS320C62xx, it is not always possible to execute eight instructions in parallel on the TMS320C64xx. In addition, many TMS320C64xx instructions are much simpler than those of conventional DSP processors. This combination makes MIPS-based performance comparisons between the TMS320C64xx and conventional DSP processors misleading. Unlike the TMS320C62xx, however, some of the new instructions found on the TMS320C64xx (e.g., SIMD instructions) perform more than one RISC-like atomic operation.*

The TMS320C64xx is object-code upward compatible with the TMS320C62xx. That is, the TMS320C64xx can execute TMS320C62xx object code unmodified, but the TMS320C62xx cannot execute all TMS320C64xx instructions. Because the two processors are similar, this analysis highlights only the key differences between the TMS320C64xx and the TMS320C62xx. Readers should refer to the TMS320C62xx analysis for a full discussion of the TMS320C62xx architecture, including features common to the TMS320C62xx and TMS320C64xx.

Part	Operating Voltage (V)	Millions of 16-bit MACs per Second	On-Chip Memory		Peripherals (Preliminary)
			Program RAM	Data RAM	
C64xx	1.2 / 3.3*	2,400**	512×256 L1 cache	4K×32 L1 cache	32-channel DMA, 32-bit host port interface, three multi-channel buffered serial ports, three timers, one 64-bit external memory interface (EMIF), one 16-bit EMIF
			4K×256 unified L2 cache		

TABLE 7.16-1. Projected characteristics of (as yet unannounced) initial TMS320C64xx family members.

* The core operates at 1.2 volts while all I/O signals are 3.3-volts compatible.

** Projected

Architecture

Like the TMS320C62xx, the core architecture of the TMS320C64xx family consists of two fixed-point data paths, a program control unit (including program fetch, instruction dispatch, and instruction decode units), and program and data memory interfaces. Figure 7.16-1 illustrates the TMS320C64xx family architecture. Like one member of the TMS320C62xx family, the TMS320C6211 (and like the floating-point TMS320C6711), initial TMS320C64xx family members will provide a two-level cache-based memory architecture. Figure 7.16-1 depicts the level-1 and level-2 cache controllers as well as the two data paths of the core.

Data Path

Like the TMS320C62xx, the TMS320C64xx has two identical fixed-point data paths (numbered 1 and 2). Four functional execution units (L, S, M, D) are associated with each data path. The eight functional units are capable of executing up to eight 32-bit instructions in parallel using two register files. As in the TMS320C62xx, each register file is associated with one data path: register file A is associated with the data path 1 and register file B is associated with the data path 2. As in the TMS320C62xx, execution units in one data path can access registers associated with the other data path (with some restrictions) through the use of register cross-paths.

The TMS320C64xx has 64 32-bit general-purpose registers, twice as many as the TMS320C62xx. For each of the A and B register files, the TMS320C64xx adds 16 general-purpose registers (A16-A31 and B16-B31) to those found on the TMS320C62xx. Each register file has eleven 32-bit read ports and eight 32-bit write ports, allowing multiple concurrent accesses from a group of parallel instructions to various registers of each register file.

The TMS320C64xx can operate on 8-, 16-, 32- and 40-bit ("long") data (as can the TMS320C62xx), and can also operate on 64-bit ("double word") data when loading or storing data from or to memory. The TMS320C64xx uses 64-bit data words to hold the results of some multiply operations. The TMS320C64xx handles 64-bit data by using a pair of registers; an even-numbered register and the next consecutive odd-numbered register.

The TMS320C64xx adds support for SIMD instructions that can process four 8-bit data operands packed into 32-bit registers. It also extends the support for SIMD processing of two 16-bit operands packed into 32-bit registers. As in the TMS320C62xx, all execution units in the TMS320C64xx have a throughput of one cycle and latencies from one to several cycles, depending on the instruction.

For each pair of execution units (L1/L2, S1/S2, M1/M2, D1/D2), the key differences between the TMS320C64xx data paths and the TMS320C62xx data paths are described below.

- The **L units** on the TMS320C64xx have added support for 16-bit dual and 8-bit quad integer SIMD additions and subtractions. Dual16-bit and quad 8-bit minimum and maximum SIMD instructions have also been added. These instructions use packed data from two 32-bit operand registers (two 16-bit operands or four 8-bit operands are packed in each 32-bit operand register for dual or quad operations, respectively) and produce packed results in 32-bit registers. The TMS320C64xx, like the TMS320C62xx, does not support 32-bit minimum and maximum instructions.

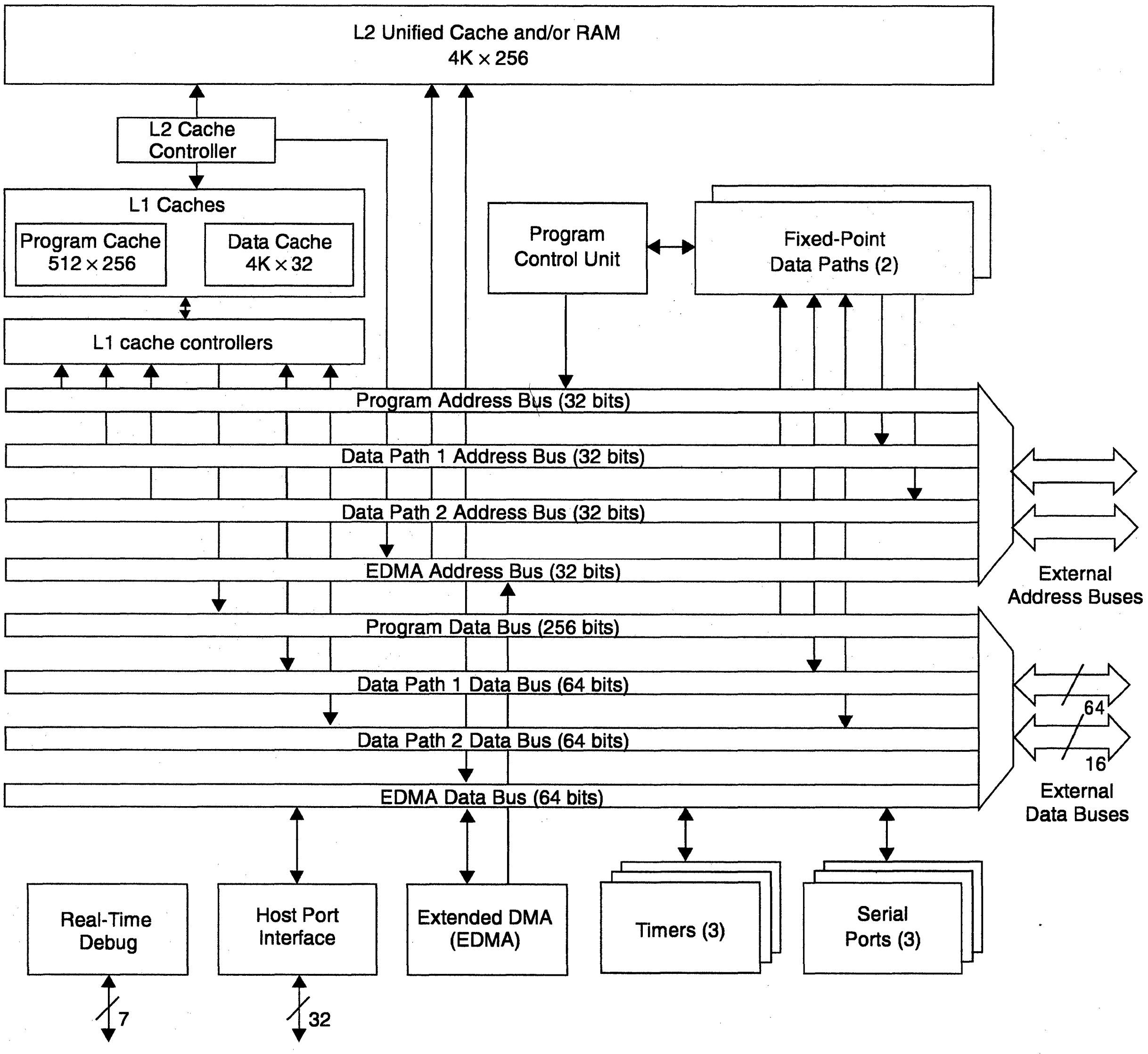


FIGURE 7.16-1. TMS320C64xx processor architecture. Peripherals and cache configurations are based on preliminary information from Texas Instruments; actual family members may include different peripherals and memory configurations.

The L units also add support for dual 16-bit and quad 8-bit packed-data manipulation instructions, such as data swapping, data selecting, and data re-arranging. These instructions are discussed later (refer to Table 7.16-5 for a summary of these instructions). All of the new L-unit instructions have single-cycle latencies.

- The **S units** on the TMS320C64xx support dual 16-bit saturating additions. Furthermore, the TMS320C64xx adds support for quad 8-bit saturating additions. For dual 16-bit operations, packed data in each of the 32-bit operand registers can be all signed or unsigned, or one 32-bit operand register can contain all signed packed data and the other one all unsigned packed data. For quad 8-bit operations, the processor only supports unsigned operands.

  The TMS320C64xx also supports dual 16-bit or quad 8-bit compare instructions. For dual 16-bit compares, the processor supports compare for equal, greater than, or less than with signed operands. For quad 8-bit compares, the processor supports only unsigned operands. The processor supports dual 16-bit arithmetic shift right operations (in which the signed 16-bit lower and upper parts of a 32-bit register are individually shifted right). Like the L units, the S units of the TMS320C64xx have added support for packed-data manipulation. Apart from some new branching instructions, all of the new instructions that execute in the S unit have single-cycle latencies.

- Each **M unit** on the TMS320C64xx can perform SIMD multiplications on 16-bit and 8-bit operands. Each M unit can perform dual 16 × 16 → 32-bit multiplications on signed operands, producing two packed signed results in a 64-bit register pair (each 64-bit register is composed of a 32-bit odd- and even-numbered register pair). Like the TMS320C62xx, the TMS320C64xx does not provide guard bits. Each M unit can also perform four 8 × 8 → 16-bit signed/unsigned multiplies, producing four packed signed results in a 64-bit register pair. Quad 8-bit unsigned/unsigned multiplies are also supported, producing four unsigned results in a 64-bit register pair.

  The TMS320C62xx supports fractional multiplications via a signed 16 × 16 → 32-bit multiply instruction that includes a left-shift-by-one operation. The TMS320C64xx extends this support with a dual signed 16 × 16 → 32-bit multiply producing two left-shifted-by-one and saturated results in a 64-bit register pair. (Note that this operation isn't supported for quad 8-bit multiplications.)

  Additionally, the TMS320C64xx offers support for extended-precision multiplications via signed 16 × 32-bit → 48-bit multiplications. The 48-bit results are written to the lower part of a 64-bit register pair. This form of multiplication can optionally include rounding and right-shifting by 15 to produce a 32-bit result.

  Like the TMS320C62xx, the TMS320C64xx does not explicitly support 32 × 32 → 64-bit multiplications.

  In addition to support for SIMD-style multiplications, the M units of the TMS320C64xx also add support for dual and quad SIMD dot products. The results of dual 16-bit × 16-bit or quad 8-bit × 8-bit SIMD multiplications are summed together (within the M units) and placed in a 32-bit register. For dual 16-bit arithmetic, packed data in each of the 32-bit operand registers can be all signed or unsigned, or one 32-bit operand register can contain all signed packed data and the other can contain all unsigned packed data. For quad 8-bit operations, packed data in each of the 32-bit operands register can be all unsigned, or one 32-bit operand register can contain all signed packed data and the other one can contain all unsigned packed data (all signed data isn't supported). Additionally, the dual 16-bit dot product instruction supports three variations. First, the two intermediate results of dual 16-bit signed/signed SIMD multiplications can be subtracted instead of added. Second, the result of a dual 16-bit signed/unsigned dot product operation can be rounded and right shifted by 16 bits to produce a 16-bit result. Third, the two intermediate results of dual 16-bit signed/signed rounded and shifted multiplications can be subtracted instead of added prior to rounding and shifting.

> *The TMS320C64xx SIMD dot product instructions increase code density for MAC-based DSP functions and increase performance by allowing more parallelism. The multiply, add/subtract, and shift operations are all performed within a single M-unit dot product instruction. Unlike in the TMS320C62xx, these operations don't require the use of the other execution units, freeing the other units for additional parallel operations.*

As on the TMS320C62xx, all TMS320C64xx multiply and dot product instructions have a multi-cycle latency. The TMS320C64xx supports the 16-bit × 16 → 32-bit multiplication (with any combination of signed/unsigned operands) found on the TMS320C62xx with the same two-cycle latency and single-cycle throughput. On the TMS320C64xx, all other SIMD and non-SIMD multiplies (i.e., extended-precision multiplies) and SIMD dot product instructions have four-cycle latencies and single-cycle throughput. Table 7.16-2 summarizes the key attributes of all TMS320C64xx multiply and dot product instructions.

The M units on the TMS320C64xx also add support for 32-bit rotate instructions and 32-bit bidirectional shifts by a variable amount (for which the shift amount and direction depend on the contents of an operand register). These instructions have two-cycle latencies and single-cycle throughput.

> *The TMS320C64xx supports twelve multiply instructions (including dot product instructions). The variety of distinct instructions complicates learning and programming the architecture.*

- The **D units** on the TMS320C64xx have added support for 64-bit loads and stores and related address computations. Compared to the TMS320C62xx, the D units of the TMS320C64xx also add support for non-aligned loads and stores, i.e., memory accesses whose addresses aren't necessarily multiples of the size of the accessed

M Unit Instr.	Description	SIMD	Signed/ Unsigned Operand Support	Latency Cycles
MPYxxzz[a,b]	16-bit × 16-bit multiply → 32-bit result (as on TMS320C62xx)	None	S×S, U×S, U×U	2
MPY2	Packed × packed operands → 64-bit packed results	Dual	S×S	4
MPYzz4[b]		Quad	S×U, U×U	
SMPY	Fractional 16-bit × 16-bit multiply → 32-bit left-shifted and saturated result (as on TMS320C62xx)	None	S×S	2
SMPY2	Fractional packed × packed operands → 64-bit left-shifted and saturating packed results	Dual	S×S	4
MPYIzz[b]	Extended precision 16-bit × 32-bit → 48-bit result	None	S×S	4
MPYIzR[b]	Extended precision with round and right shift 16-bit × 32-bit → 32-bit result	None	S×S	4
DOTP2	Sum of packed × packed operands → 32-bit result	Dual	S×S, U×U, S×U	4
DOTPzz4[b]		Quad	S×U, U×U	
DOTPN2	Difference of packed × packed results → 32-bit result	Dual	S×S	4
DOTPRz2[b]	Rounded sum of packed × packed results → 32-bit result	Dual	S×U	4
DOTPNRzz2[b]	Rounded difference of packed × packed results → 32-bit summed result	Dual	S×S	4

TABLE 7.16-2. TMS320C64xx support for multiply and dot product operations.

[a] Like the TMS320C62xx, the TMS320C64xx supports instructions that access the 16-bit upper or lower part of 32-bit register operands. The "xx" in the instruction name here indicates that the programmer specifies the high or low 16 bits of two 32-bit registers in the instruction mnemonic; e.g., MPYLH uses the low 16-bit part of the first operand and the high part of the second one.

[b] A "z" or "zz" in an instruction name indicates that the programmer specifies one or more options for the instruction in the instruction mnemonic; for example, whether the operands are signed, unsigned, or mixed.

data word. As on the TMS320C62xx, all load instructions on the TMS320C64xx have a five-cycle latency and store instructions have single-cycle latency.

The above latencies (and more generally, all latencies discussed here) must be understood as best-case latencies; i.e., these latencies apply when accesses to memory locations require data that is already in on-chip level cache memory. When a cache miss occurs because the required data isn't in the cache memory, the entire instruction pipeline stalls to allow the cache to be updated. In this case, load and store instructions require more cycles than the number of cycles expressed above. Preliminary information regarding cache miss penalties is discussed in the Memory System *section below.*

As on the TMS320C62xx, all load and store instructions on the TMS320C64xx (including non-aligned loads/stores) allow the address pointer to be pre/post modified as part as the load instruction. The modified address is updated with single-cycle latency (in contrast to the actual load itself, which has a latency of five cycles).

The D units on the TMS320C64xx can execute AND and OR instructions (on the TMS320C62xx only the S and L units can execute these instructions). In addition, the D units support two new instructions that compute bitwise logical *xor* and *and-not* (*and-not* is a bitwise logical *and* between one 32-bit operand and the bitwise logical inverse of a second 32-bit operand). These two instructions can also execute in the S and L units of the TMS320C64xx.

In contrast to the TMS320C62xx, the TMS320C64xx L and D units can be used to load immediate 5-bit constants, in addition to the S unit's ability to load 16-bit constants.

As with the TMS320C62xx, there are no status bits associated with any of the arithmetic instructions (e.g., adds, subtracts, multiplies, absolute values) on the TMS320C64xx. Overflow and underflow protection is supported only by certain instructions (e.g., add and subtract instructions with saturation).

The lack of status bits is not a problem on the TMS320C64xx; due to its parallel architecture and conditional instruction execution, status bits can be implemented (emulated) in software without a significant performance penalty.

Like the TMS320C62xx, the TMS320C64xx allows up to four reads from a given register in a given cycle. In addition, the TMS320C64xx allows up to two 64-bit loads to be written to each register file per instruction cycle (through the parallel use of the D1 and D2 units).

Some register ports are specifically designated for transferring 40-bit "long" values to the L and S execution units. These ports are shared between execution units, potentially causing resource contention when instructions that read or store 40-bit values from or to registers are scheduled to execute in parallel with other instructions. Specific cases of resource contention vary among the TMS320C62xx, TMS320C67xx, and TMS320C64xx processors. Table 7.16-3 summarizes the differences among the TMS320C6xxx architectures in this respect. Unlike the TMS320C62xx, the TMS320C64xx allows the S and L units of the same data path to manipulate two 40-bit values in parallel. In addition, the TMS320C64xx also allows 32-, 16- or 8-bit stores (but not 64-bit stores) to read data from the same register file that is used as a 40-bit long source by the L unit. Like the TMS320C62xx, though, the TMS320C64xx does not allow the use of 32-, 16-, or 8-bit store instructions while the S unit reads a 40-bit long source in the same data path. In addition, the TMS320C64xx introduces two new constraints when using a 64-bit load or store instruction while a 40-bit value is accessed by the S or L unit, detailed in Table 7.16-3.

> *Although the TMS320C64xx reduces the number of restrictions on operations that can be performed in parallel with manipulation of 40-bit operands in comparison to the TMS320C62xx (and TMS320C67xx), the use of 40-bit arithmetic is still subject to such constraints, which complicate programming.*

On the TMS320C62xx, only one execution unit per data path can access an operand register from the register file associated with the opposite data path. (In other words, a total of two cross-path accesses are allowed per instruction cycle.) The TMS320C64xx

40-bit Operation	C62xx	C67xx	C64xx
Both S and L units on the same side (i.e., S1/L1 or S2/L2) read or write a 40-bit long data operand	Invalid	Invalid	**Valid**
A 64-bit load instruction writes data to the same register file used as the destination of a 40-bit long result produced by the S unit	N/A[a]	Invalid	Invalid
A 64-bit load instruction writes data to the same register file used as the destination of a 40-bit long result produced by the L unit	N/A	Invalid	**Valid**
A 32-, 16-, or 8-bit store instruction reads data from the same register file that is used as a 40-bit long source by the S unit	Invalid	Invalid	Invalid
A 32-, 16-, or 8-bit store instruction reads data from the same register file that is used as a 40-bit long source by the L unit	Invalid	Invalid	**Valid**
A 64-bit store instruction reads data from the same register file that is used as a 40-bit long source by the L unit	N/A	N/A	Invalid

TABLE 7.16-3. TMS320C62xx, TMS320C67xx, TMS320C64xx 40-bit (long) resource conflicts. Invalid combinations of operations are rejected by the assembler.

[a] N/A means "Not Applicable", i.e., rejected by the assembler because 64-bit loads or stores are not supported for the target processor.

alleviates this restriction by allowing up to two execution units per side to simultaneously access the same cross-path (i.e., the same register from the opposite data path). Unlike the TMS320C62xx, however, the TMS320C64xx may incur "cross-path stalls." Cross-path stalls occur whenever an instruction attempts to read a cross-path register that was updated in the previous cycle. In this case, a single-cycle stall occurs. No stall occurs when the cross-path register is the destination of a load instruction.

Cross-path stalls complicate software optimization for the TMS320C64xx, and also complicate the process of porting code among the three TMS320C6xxx architectures. For example, software written for the TMS320C62xx may incur cross-path stalls when executed on the TMS320C64xx, thus requiring more cycles to execute on the newer architecture. In the worst case, this increase in cycle counts may entirely offset the performance gains afforded by the TMS320C64xx's higher clock rate.

Cross-path stalls can sometimes be avoided by rescheduling instructions; i.e., by rewriting or recompiling code specifically for the TMS320C64xx. When possible, the C compiler and assembly optimizer of the TMS320C64xx attempt to avoid or reduce cross-path stalls, according to Texas Instruments. To improve the performance of existing C code, TMS320C62xx software can be recompiled for the TMS320C64xx using the same tool suite.

To achieve optimal performance, however, software may have to be re-engineered to limit the impact of cross-path stalls. This detracts from the value of the code compatibility between the TMS320C64xx and the other TMS320C6xxx families.

On the TMS320C62xx, the D units have limited support for cross-path register accesses in comparison with the other execution units; cross-path accesses are only supported for loads or stores. In contrast, the TMS320C64xx can use cross-path accesses for logical and arithmetic instructions in the D unit. For load and store instructions, the TMS320C64xx, like the TMS320C62xx, must use an address register from the same side of the data path as the D1 or D2 unit uses to execute the instruction. Destination/source registers for load/store instructions can use a cross path, however, as on the TMS320C62xx.

Memory System

The on-chip memory system of the TMS320C64xx is similar to that of the TMS320C62xx, but improves data memory bandwidth by supporting 64-bit data transfers in addition to 32-bit and 16-bit transfers. Each of the two data paths on the TMS320C64xx can load or store one 64-bit word per instruction cycle, compared to one 32-bit word per data path on the TMS320C62xx. The TMS320C64xx supports the wider data word via wider data buses in each data path (64 bits, compared to 32 bits on the TMS320C62xx).

Initial TMS320C64xx devices will use a similar two-level cache organization as the TMS320C6211, according to Texas Instruments. The on-chip memory architecture of initial TMS320C64xx devices is depicted in Figure 7.16-2. Numbers in bold-italics in Figure 7.16-2 are key differences between the TMS320C6211 and the projected TMS320C64xx family members. All information regarding cache miss penalties described in this section is based on preliminary data provided by Texas Instruments. Texas Instruments states that this information may change prior to fabrication of TMS320C64xx-based devices.

The level-1 memory is organized as separate program and data caches. The level-2 cache is a unified on-chip cache; i.e., it is used both for instructions and data. The level-2 cache of the initial TMS320C64xx devices comprises two memory banks, each of which can be dynamically continued to serve as second-level cache or as direct-mapped RAM (i.e., reducing the size or eliminating the L2 cache). In comparison, the level-2 cache of the TMS320C62xx comprises four banks. As on the TMS320C6211, the two level-1 caches of the TMS320C64xx are always active.

As on the TMS320C6211, the level-1 instruction cache of the initial TMS320C64xx devices is organized as a read-only direct-mapped cache comprising a sin-

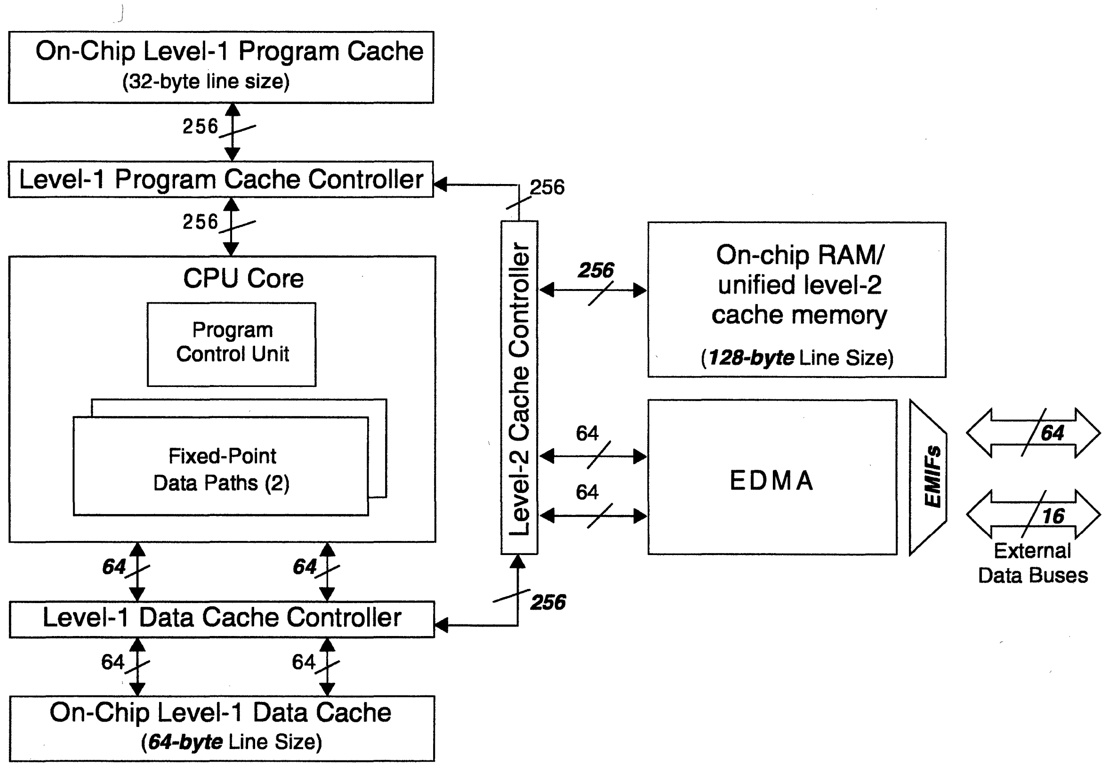

On-Chip Level-1 Program Cache
(32-byte line size)
256
Level-1 Program Cache Controller
256
256
CPU Core
Program Control Unit
Fixed-Point Data Paths (2)
Level-2 Cache Controller
256
On-chip RAM/ unified level-2 cache memory
(128-byte Line Size)
64
64
EDMA
EMIFs
64
16
External Data Buses
64
64
256
Level-1 Data Cache Controller
64
64
On-Chip Level-1 Data Cache
(64-byte Line Size)

FIGURE 7.16-2. TMS320C64xx two-level cache organization as projected for initial family members. Numbers in bold-italics reflect key differences between the TMS320C64xx and the TMS320C6211.

gle-ported 256-bit wide memory bank. The line size of the TMS320C64xx level-1 instruction cache is 32 bytes (it holds an entire fetch packet). The level-1 instruction cache of the initial TMS320C64xx can store up to 4K 32-bit instructions.

As on the TMS320C6211, the level-1 data cache of the initial TMS320C64xx devices is organized as a two-way set associative read/write cache. The line size is 64-bytes, twice that of the TMS320C6211. The cache can contain a maximum of 16 Kbytes. As on the TMS320C6211, the replacement policy of the level-1 data cache on the TMS320C64xx is based on the Least Recently Used (LRU) algorithm. The TMS320C6211 uses a dual-ported memory for the level-1 data cache in order to allow the two D units to perform two concurrent accesses in a single cycle. In comparison, the level-1 data cache of the TMS320C64xx consists of eight 32-bit wide single-ported memory banks. As on the TMS320C620x and TMS320C6701, the memory banks of the TMS320C64xx level-1 data cache are interlaced so that consecutive 16-bit data are spread over the various memory banks rather than grouped into a single bank. Consequently, even though the level-1 data cache of the TMS320C64xx is single-ported, two concurrent accesses from the D units are supported when the requested data words are not located in the same memory bank.

> *The frequency of conflicts can be reduced or even avoided when the set of data being accessed concurrently (i.e., in the same cycle or execute packet) is properly arranged in external memory. Multiple adjacent accesses (e.g., array accesses in which adjacent 16-bit data words are read one after another in a loop) can best benefit from this optimization. In order to reduce conflicts, when possible the programmer should align the starting address of arrays being accessed so that no bank conflicts occur throughout the execution of the loop. Wherever necessary, this technique is used in the implementation of the BDTI Benchmarks™ as discussed below. Although this approach increases efficiency, it complicates programming.*

When level-1 cache misses occur, a request for information is sent to the level-2 cache controller. If the required information is in the level-2 memory (whether it is configured as cache or RAM), several cycles are needed to copy the necessary information from the level-2 memory to the level-1 instruction or data cache.

According to Texas Instruments, the TMS320C64xx implements instruction prefetching in order to reduce the number of cycles associated with instruction cache misses. Once a miss has occurred, the instruction cache controller determines whether a miss will also occur in the next cycle. If this is the case, the cache line updates occur in parallel and the average number of cycles required to update a line is reduced. Depending on the number of instructions executing in parallel, the stall penalty is either zero or four cycles. More precisely, if two consecutive groups of up to four parallel instructions occur in a program (i.e., two consecutive execute packets containing four or fewer parallel instructions), the prefetching mechanism allows the second group to execute without a level-1 cache miss penalty. This is because the second group was prefetched during the cache miss of the previously executed instruction. On the other hand, if an execute packet of more than four parallel instructions follows an execute packet with fewer than four instructions, the second execute packet has a level-1 cache miss penalty of four cycles. Similarly, if an execute packet containing fewer than four instructions follows an execute packet having more than four parallel instructions, the CPU also stalls for four cycles during the execution of the second packet after a level-1 cache miss.

The level-1 data cache also implements prefetching. As with the level-1 instruction cache, the stall penalty varies according to the sequence of data accesses. Due to the TMS320C64xx's ability to perform cache line updates in a concurrent manner, the cache miss penalty of the level-1 data cache varies from 4 to 2.5 cycles per 64-byte line. This penalty varies depending on the number of cache misses occurring during a cycle (one cache miss per data path or D unit can arise), and depending on the total number of cache misses measured during two successive execute packets. With two consecutive execute packets, a total of one to four cache misses can occur. The lowest per-line cache impact (i.e., 2.5 cycles per line) is encountered when a total of four cache line misses occur during two consecutive execute packets (i.e., one miss per data path for both execute packets). A 4-cycle stall is encountered when only one cache miss occurs (either in the D1 or D2 functional unit) during the execution of an execute packet that is followed by an execute packet which doesn't suffer a cache miss.

As on the TMS320C6211, the unified level-2 cache of the TMS320C64xx is organized as a set associative cache with a variable number of supported cache "ways." As on the TMS320C6211, the cache replacement policy of the level-2 cache of the TMS320C64xx is based on the Least Recently Used (LRU) algorithm. One to four ways can be configured on the TMS320C6211, one per level-2 cache memory bank excluding banks configured to serve as direct-mapped RAM. This also applies to the TMS320C64xx: each of the two banks is divided into two regions and each region can be configured to be part of the level-2 unified cache. When all four regions of the TMS320C64xx level-2 memory are configured as cache, the level-2 memory acts as a four-way set associative cache.

When a level-2 cache miss occurs, the enhanced DMA (EDMA) controller updates the appropriate LRU line of the cache. The number of cycles required to update a level-2 cache line depends on the type of external memory used.

Simultaneous accesses to the level-2 memory banks are allowed without penalty only when distinct banks are used. Conflicts may arise when, for example, the level-1 instruction cache accesses a bank of the level-2 memory while, in parallel, the level-1 data cache and/or the EDMA controller attempts to access the same memory bank.

As on the TMS320C6211, a set of external memory regions can be configured so as not to be cached in the level-1 or level-2 caches.

The penalties due to cache misses can have a major effect on performance. For example, functions that operate on a small set of

> *data or functions that execute in a few cycles can easily see their performance drop by a factor of two if their instructions and data are not pre-loaded in the caches. Functions that randomly access an array whose size is greater than the projected level-1 cache size (i.e., 16 Kbytes) can also suffer severely from cache miss penalties.*
>
> *The use of cache memory enables faster clock speeds, but significantly impairs the programmer's ability to predict software execution times. Consequently, ensuring that real-time constraints are met and optimizing software can be very difficult on a cache-based processor such as the TMS320C64xx.*

External Memory Interface

Unlike the TMS320C62xx processors, initial TMS320C64xx devices will provide two distinct external memory interfaces (EMIFs), EMIFA and EMIFB. The two interfaces support a wide range of asynchronous and synchronous external memories: RAM, ROM, flash, synchronous DRAM or SRAM, synchronous-burst SRAM (SBSRAM), and FIFO. EMIFA has a data bus width of 64 bits, whereas EMIFB has a data bus width of 16 bits. As in other TMS320C6xxx processors, the addressable space of each EMIF interface is divided into four areas. As on the TMS320C62xx, asynchronous memory can be used in all areas. When combining all of the areas, the EMIFA interface can address up to 1024 Mbytes of external memory and EMIFB can address up to 512 Mbytes of external memory.

EMIFA and EMIFB are both byte addressable. EMIFA can access memory widths of 8 bits, 16 bits, 32 bits, and 64 bits, while EMIFB is restricted to accessing memory widths of 8 and 16 bits. Similar to the EMIF on the TMS320C6211 and TMS320C6711, EMIFA and EMIFB require that an external clock source be provided by the system to drive the external memory (the same external clock is used for both EMIFs). From this signal, the processor generates another clock signal at half or one quarter of the input clock rate. This second signal can be used to drive peripherals operating at different speeds on the same bus. Preliminary information from Texas Instruments states that the maximum memory cycle rate supported will be 133 MHz for SDRAM and SBSRAM on both TMS320C64xx EMIFs. EMIFA and EMIFB allow for programmable SBSRAM and SDRAM read and write latency, providing support for a wide range of SBSRAM memories (other TMS320C6xxx families support programmable read and write latencies for SDRAM, but not for SBSRAM). As on the other TMS320C6xxx devices, refresh signals for SDRAM are provided by EMIFA and EMIFB, and the refresh period is programmable. For a TMS320C64xx running at 600 MHz, the projected maximum bandwidth is 1,064 Mbytes per second for the 64-bit EMIF and 266 Mbytes per second for the 16-bit EMIF, according to preliminary information from Texas Instruments. Both EMIFs will be able to operate in parallel, thus achieving a total bandwidth of about 1,300 Mbytes per second.

In contrast to the EMIFs found in other TMS320C6xxx families, the TMS320C64xx EMIFs add support for “peripheral data transfers.” This feature allows data to be transferred directly between external devices (e.g., from a FIFO to an external memory) under the control of the EMIF. In peripheral data transfers, the EMIF generates control signals and addresses to effect a direct transfer between external devices, rather than the typical approach of performing a read followed by a write. This increases throughput between external devices.

> *Peripheral data transfers should prove valuable for off-loading the TMS320C64xx from routine data-transfer duties, which should facilitate efficient use of the processor in many applications.*

Address Generation Units

The D1 and D2 address generation units on the TMS320C64xx are similar to those of the TMS320C62xx, but they add support for 64-bit data addressing. 64-bit loads and stores use pairs of 32-bit registers.

Additionally, compared to the TMS320C62xx, the TMS320C64xx adds support for non-aligned loads and stores. Non-aligned memory accesses allow references to data whose addresses aren’t necessarily multiples of the size of the accessed data word. In contrast, all 16-bit and 32-bit load instructions on the TMS320C62xx must use addresses that are divisible by two and four, respectively (no restrictions apply when loading byte values).

Non-aligned loads and stores are supported on the TMS320C64xx through specific instructions that execute in the D1 or D2 units. LDNx and STNx are the non-aligned load and store instructions, where x is the data size suffix: W for word (32-bit) and DW for double word (64-bit). Non-aligned half-word (16-bit) loads aren’t supported. The TMS320C64xx restricts the use of non-aligned memory accesses in each execute packet: when a D unit executes an LDNx or STNx operation, the other D unit cannot perform a memory access in parallel.

> *Despite these restrictions, the support for non-aligned accesses provides flexibility for the programmer and compiler, which is particularly valuable when using SIMD instructions. In addition, since the D units can now execute more non-memory related operations (like logical operations), one D unit can perform other useful work while the other D unit is performing a non-aligned access.*

The TMS320C64xx supports modulo addressing through the same set of instructions and the same set of registers (A4-A7 and B4-B7) as used on the TMS320C62xx. Like the TMS320C62xx, the TMS320C64xx supports eight concurrently active circular buffers. The TMS320C64xx architecture adds support for double word modulo addressing through the aligned and non-aligned double-word load and store instructions.

As on the TMS320C62xx, two simultaneous memory accesses on the TMS320C64xx cannot use registers in the same register file as their address pointers.

Pipeline

The TMS320C64xx pipeline is identical to that of the TMS320C62xx. The number of stages and the description of these stages are unchanged. This information is presented in Section 7.15.

The TMS320C64xx alleviates the restriction found on the TMS320C62xx (and the TMS320C67xx) which requires each execute packet to be contained entirely within a single fetch packet. (A fetch packet is a group of eight instructions loaded from memory; execute packets are groups of instructions that execute in parallel.) As on the TMS320C62xx, the TMS320C64xx always fetches eight instructions from on-chip memory at a time. These instructions can all be scheduled to execute in parallel (in which case the execute packet matches the fetch packet). Alternatively, each instruction in a fetch packet can execute sequentially or some instructions can be scheduled to execute in parallel and others to execute sequentially. This approach increases instruction packing density when compared to conventional VLIW architectures with fixed-length instructions. However, the TMS320C62xx does not allow an execute packet to span two fetch packets. Instead, the TMS320C62xx tools insert NOP instructions to ensure that execute packets are always entirely contained within fetch packet bounds. Note that these inserted NOPs execute in parallel with padded fetch packets and hence don't *directly impact* speed (see comment below), but they do increase memory usage. The TMS320C64xx removes this restriction, and allows execute packets to span fetch packets.

> *NOP padding is handled automatically by the TMS320C62xx (and the TMS320C67xx) assembler, so from a TMS320C64xx programmer's point of view, nothing has changed. With the removal of this constraint, however, TMS320C64xx code density is increased over that of the TMS320C62xx. With the improvement in code density, a larger number of relevant instructions can be cached in internal memory at a time, causing an indirect speed gain due to a reduction in cache misses.*
>
> *Texas Instruments states that on average, TMS320C64xx code size is reduced by 25% compared to TMS320C62xx code size. When reassembling BDTI's Control benchmark optimized for the TMS320C62xx on the TMS320C64xx, however, the code size did not change. On average, reassembling the TMS320C62xx BDTI Benchmarks on the TMS320C64xx (without optimizing them for the newer architecture) reduced the code size by about 12%. Comparing optimized TMS320C64xx BDTI Benchmarks to optimized results for the TMS320C62xx, the code size is reduced by roughly*

15%. Our analysis indicates that the improvement in code density is likely to be less than the 25% projected by Texas Instruments.

Instruction Set

The TMS320C64xx instruction set is a superset of the TMS320C62xx instruction set, which is summarized in Table 7.15-4. Key instructions added to the TMS320C64xx include SIMD and application-specific operations.

The 16-bit fixed-point ADD2 and SUB2 instructions found on the TMS320C62xx allow two 16-bit additions or subtractions to be calculated concurrently in each S unit. The two sets of two 16-bit operands are packed into two 32-bit registers and the packed results are written into the 16-bit upper and lower parts of a 32-bit register. The TMS320C64xx extends the SIMD capability of the TMS320C62xx in two ways: it allows more functional units to execute the ADD2 and SUB2 instructions, and supports more SIMD instructions (multiply, dot product, shift, compare, etc.).

Noteworthy TMS320C64xx SIMD instructions are described in the *Data Path* section above and are summarized in Table 7.16-4. Separate instructions are provided for dual 16-bit and quad 8-bit versions of each operation. For example, the ADD4 instruction is used to perform quad 8-bit SIMD additions whereas the ADD2 instruction performs dual 16-bit SIMD additions. All SIMD instructions add a “2” or “4” suffix to the base name of the instruction.

Associated with its wide range of SIMD instructions, the TMS320C64xx adds a number of packed-data manipulation instructions. These instructions allow packed data in 32-bit operand registers to be accessed, combined, and re-ordered. Table 7.16-5 summarizes these instructions for dual 16-bit and quad 8-bit processing, illustrating how packed data is rearranged in the destination register for the various key instructions.

The packed data manipulation instructions of the TMS320C64xx are flexible and very useful when using SIMD arithmetic. For example, the quad 8-bit unpack and pack instructions allow expanding (unpacking) of two 8-bit values into a pair of 16-bit values packed into a 32-bit operand register. Then, dual 16-bit SIMD arithmetic instructions can be used to improve the precision of intermediate results. The 16-bit results can be converted back to two 8-bit values using the dedicated packing instruction.

In addition, the instruction set has been extended with application-specific instructions, most of which execute in the M units. Noteworthy application-specific instructions include:

- The **L units** add the quad 8-bit SIMD SUBABS4 instruction which produces four absolute differences. This instruction targets motion-estimation algorithms found in video compression standards. Dual 16-bit SIMD absolute-difference operations are not supported, but dual 16-bit SIMD absolute value operations (without the difference calculation) are supported by the L unit.

> *Combined with the quad 8-bit SIMD absolute-difference instruction of the L unit, the quad dot-product instruction allows efficient implementation of the MAD (mean absolute difference) operation used in motion estimation for video compression standards.*

- The **M units** add most of the application-specific instructions. They add support for dual 16-bit and quad 8-bit SIMD mask-generating instructions that execute with a two-cycle latency. These instructions convert the two or four least-significant bits of an operand register to the corresponding two or four packed 16-bit or 8-bit masks in a 32-bit register. For example, 1001 in binary produces an ordered sequence of four masks; 0xFF, 0x00, 0x00 and 0xFF gathered in a 32-bit register (i.e., 0xFF0000FF). In comparison, 01 in binary produces the two masks 0x0000 and 0xFFFF or 0x0000FFFF in a 32-bit result register.

> *The combination of SIMD comparison instructions with separate status bits and mask generation instructions is useful for image*

SIMD Operation	Execution Unit	
	Quad 8-bit Support	**Dual 16-bit Support**
Multiply	M	M
Multiply with Saturation	not supported	M
Dot Product	M	M
Addition/Subtraction	L	L, S, D
Addition with Saturation	S	S
Absolute Value	not supported	L
Subtract with Absolute Value	L	not supported
Comparison	S	S
Shift	not supported	S
Packed-data manipulation	L or S depending on operation and data type	
Minimum/Maximum	L	L
Mask generation	M	M

TABLE 7.16-4. Overview of the main TMS320C64xx SIMD operations. Note that all instructions can be executed conditionally.

thresholding and video motion compensation. For example, to perform a thresholding operation, an SIMD comparison instruction is first executed, comparing multiple data values against a threshold. The status bits indicating the outcomes of each of these comparisons are then used to generate a mask. Finally, the mask is used with a logical AND instruction to pass only those values greater than the threshold, setting the other values to zero.

The M units of the TMS320C64xx introduce support for dual 16-bit or quad 8-bit averaging operations. The AVG2 and AVGU4 instructions compute the mean val-

	Dual 16-bit Operations:			Quad 8-bit Operations:				
Operands Registers		32	0		32			0
	Src1:	a1	a0	Src1:	a3	a2	a1	a0
	Src2:	b1	b0	Src2:	b3	b2	b1	b0
Instructions and Contents of the Packed 32-bit Register Result[a]	PACK2[b]	a0	b0					
	PACKH2	a1	b1	PACKH4[c]	a3	a1	b3	b1
	PACKHL2	a1	b0					
	PACKLH2	a0	b1					
				PACKL4[c]	a2	a0	b2	b0
				SHLMB[d]	b2	b1	b0	a3
				SHRMB[d]	a0	b3	b2	b1
	SWAP2	a0	a1	SWAP4	a2	a3	a0	a1
				UNPACKHU4	0x00	a3	0x00	a2
				UNPACKLU4	0x00	a1	0x00	a0

TABLE 7.16-5. TMS320C64xx packed-data manipulation instructions.

a. Instructions on a black background use only the Src1 packed 32-bit operand register. All other instructions use Src1 and Src2.

b. An additional SPACK2 instruction treats Src1 and Src2 operands as non-packed 32-bit values in order to saturate and merge them into two 16-bit values packed in a 32-bit destination register.

c. An additional SPACKU4 instruction treats Src1 and Src2 operands as two packed 16-bit values in order to saturate and merge the four 16-bit values into four 8-bit results packed in a 32-bit destination register.

d. SHLMB/SLRMB stand for SHift Left/Right and Merge Byte.

ues of two or four pairs of data packed into two 32-bit registers. After adding the data in each pair, division is performed using a signed arithmetic right shift by one operation for dual arithmetic and an unsigned right shift by one for quad arithmetic. Results are written to a 32-bit register. These instructions have a two-cycle latency.

The SIMD averaging instructions can ease implementation of video motion compensation algorithms.

Also targeted at image and video processing applications, the M unit of the TMS320C64xx includes a quad 8-bit SIMD bit-count instruction (not found on the TMS320C62xx) that counts the number of "1" bits in each of the four SIMD operands contained in a 32-bit register, writing the four 8-bit results to a 32-bit register with two-cycle latency.

Compared to the TMS320C62xx, the M units of the TMS320C64xx add support for a bit-reverse instruction, easing implementation of the radix-2 fast Fourier transform.

The M units of the TMS320C64xx also add support for error-correction coding schemes found in many modern digital communication systems. First, each M unit adds a Galois field modulo-multiply operator that targets the implementation of Reed-Solomon encode and decode functions. This dedicated multiplication operator is implemented as a quad 8-bit SIMD multiply instruction. It performs four modulo multiplications in the form of packed 8-bit unsigned/unsigned multiplies, with the four results written to a packed 32-bit register. This instruction has a latency of four cycles (and has one-cycle throughput, like all other instructions on the TMS320C64xx). The number of elements in the Galois field is configured through an added control register and varies between 2^1 and 2^8.

Also, to further ease and optimize the implementation of error correction algorithms, the M units add bit interleaving and de-interleaving instructions. The de-interleaving instruction re-organizes the bits contained in a 32-bit register to gather all even-numbered bits of the operand register in the lower 16-bit part of the 32-bit destination register and all odd-numbered bits in the upper 16-bit part of the destination register. The interleaving (or shuffle) instruction performs the opposite action; i.e., it interleaves each bit found in the lower 16-bit part of an operand register with the bit found in the same location in the upper 16-bit part of the same operand register. These instructions have a two-cycle latency.

The interleaving and de-interleaving instructions greatly ease implementation of convolutional encoders and decoders.

Finally, to optimize the implementation of ITU vocoders, the M units on the TMS320C64xx add support for bidirectional shifts for which the shift amount and direction depend on the contents of an operand register. These instructions have a latency of two cycles.

Table 7.16-6 summarizes application-specific instructions discussed in this section.

Assembly Language Format

The TMS320C64xx uses the same assembly language syntax as the TMS320C62xx, where parallel bars ('||') are used to indicate that instructions are to be executed in parallel. As on the TMS320C62xx, all instructions on the TMS320C64xx can be executed conditionally. Five general-purpose registers, B0, B1, B2, A1, and A2, can be used as condition registers on the TMS320C62xx. The TMS320C64xx adds support for the A0 register as a conditional register.

Compared to the TMS320C62xx, the TMS320C64xx allows more operations to be executed in parallel in a single instruction. For example, Table 7.16-7 shows the assembly language code for a 16-bit dot-product loop implemented on the TMS320C62xx and on the TMS320C64xx. Instructions in bold show the key differences between the two loops. The TMS320C64xx loop loads two 64-bit data blocks (each containing four 16-bit words) using the LDDW instructions and executes two dual, signed 16-bit dot product instructions (DOTP2) allowing the processor to compute four 16-bit multiplication results in each iteration of the loop. Each iteration consumes one cycle since all instructions can be grouped into a single execute packet. In comparison, the TMS320C62xx can only load two 32-bit values per cycle (using the LDW instructions) and perform two non-SIMD signed 16-bit multiplies per cycle (using the MPY and MPYH instructions). Therefore,

Typical Targeted Application Domain	Instruction	Description
Image processing/ Image compression	SUBABS4	Quad 8-bit absolute difference
	AVG2/AVGU4	Dual 16-bit/quad 8-bit average
	XPND2/XPND4	Dual 16-bit/quad 8-bit bit expansion
	BITC4	Quad 8-bit count
Transform: FFT	BITR	32-bit bit reversal operation
Communication	GMPY4	Quad 8-bit Galois Field multiply
	SHFL	Bit interleaving
	DEAL	Bit de-interleaving
ITU vocoders	SSHVL, SSHVR	Signed variable shift

TABLE 7.16-6. TMS320C64xx application-specific instructions.

half as many results are produced during each iteration of the loop compared with the TMS320C64xx.

Parallel Move Support

The TMS320C64xx extends the parallel move support of the TMS320C64xx in the sense that it can load or store aligned 64-bit double words with the use of a 64-bit register pair. The TMS320C64xx has the same support as the TMS320C62xx for byte and aligned 16-bit half-word and aligned 32-bit word load and store instructions. As with the TMS320C62xx, the TMS320C64xx uses the two D units to allow up to two memory accesses during each cycle in parallel with the use of the other functional units. Hence, up to four 32-bit registers can be loaded or stored in every cycle on the TMS320C64xx—twice as many as on the TMS320C62xx. This applies to aligned memory accesses only; only one TMS320C64xx non-aligned 32-bit word or 64-bit double word transfer can be issued per cycle. Also, assuming the level-1 data cache is pre-loaded, two TMS320C64xx aligned loads or stores can execute in a single cycle only if the two concurrent accesses point to distinct level-1 data memory banks. (This is further discussed in the *Memory System* section). This restriction doesn't apply to the TMS320C6211 where the level-1 data memory is dual-ported, allowing two concurrent accesses to occur without cycle penalty (as long as the level-1 cache is loaded).

Orthogonality

In comparison to the TMS320C62xx, the TMS320C64xx eases a number of the restrictions that contribute to the earlier architecture's lack of orthogonality. For example, logical and arithmetic instructions can now be executed by the D unit in addition to the S

<table>
<tr><th colspan="2"><b>Code Comparison: 16-bit Dot-Product Inner Loop Implementation</b>
$(A7 + B7 = \Sigma h(i) \times x(i))$</th></tr>
<tr><th>TMS320C62xx</th><th>TMS320C64xx</th></tr>
<tr><td><b>LDW.D1 *A3++,A5 ;Load h(i)&h(i+1)</b>
||<b>LDW.D2 *B3++,B5 ;Load x(i)&x(i+1)</b>
||[B0]SUB.S2 B0,1,B0;Dec. loop cntr
||[B0]B.S1 LOOP ;Branch if loop !=0
||<b>MPY.M1x A5,B5,A6 ;A6=x(lo)*h(lo)</b>
||<b>MPYH.M2x A5,B5,B6 ;B6=x(hi)*h(hi)</b>
||ADD.L1 A7,A6,A7 ;A7=A7+A6
||ADD.L2 B7,B6,B7 ;B7=B7+B6</td><td><b>LDDW.D1 *A3++,A5:A4 ;Load h(i..i+3)</b>
||<b>LDDW.D2 *B3++,B5:B4 ;Load x(i..i+3)</b>
||[B0]SUB.S2 B0,1,B0 ;Dec. loop cntr
||[A0]BDEC.S1 LOOP,A0 ;Branch if loop !=0
||<b>DOTP2.M1x B5,A5,A6 ;2 16x16 mult+add</b>
||<b>DOTP2.M2x B4,A4,B6 ;2 16x16 mult+add</b>
||[!B0]ADD.L1 A7,A6,A7 ;A7=A7+A6
||[!B0]ADD.L2 B7,B6,B7 ;B7=B7+B6</td></tr>
<tr><td><b>Two</b> products are produced by this single-cycle execution packet.</td><td><b>Four</b> products are produced by this single-cycle execution packet.</td></tr>
</table>

TABLE 7.16-7. Example assembly language comparison between the TMS320C62xx and the TMS320C64xx. An "x" in the execution-unit assignment portion of an instruction (e.g., ".M1x") indicates that a register cross-path is used.

and L units. In addition, most of the restrictions on cross-path register accesses have been removed. For example, the D units of the TMS320C62xx do not support register cross-paths except for loads and stores, whereas cross-path accesses are also supported for logical and arithmetic operations that execute on the D units of the TMS320C64xx.

Although some of the causes of the lack of orthogonality on the TMS320C62xx have been eliminated in the TMS320C64xx, the TMS320C64xx is still not an orthogonal processor, for a number of reasons. Different instructions often must be used for different data types (e.g., there are different SIMD instructions depending on whether quad 8-bit or dual 16-bit values are used). Similarly, different instructions are required to perform aligned versus non-aligned accesses. Not all SIMD instructions have both dual and quad counterparts. For example, the quad 8-bit SIMD absolute-difference instruction is provided but a dual 16-bit SIMD absolute-difference isn't. In addition, support for signed and unsigned multiplication operands varies depending on which dot-product instruction is used.

> *Texas Instruments states that the TMS320C64xx is more orthogonal than the TMS320C62xx, in part because several instructions can execute in a wider variety of execution units. We believe, however, that despite the alleviation of some resource constraints, the instruction set of the TMS320C64xx is generally less orthogonal than that of the TMS320C62xx. Because of its deep pipeline and instructions with multi-cycle latencies, the older TMS320C62xx is a difficult processor to program in assembly language. The TMS320C64xx is even more difficult to program in assembly because of the complexity of its extended instruction set. For this reason, the quality of the C/C++ compiler will be particularly important for the TMS320C64xx.*

Execution Times

All fixed-point arithmetic, logic, load/store, and branch instructions of the TMS320C62xx have the same latency and throughput on the TMS320C64xx as on the TMS320C62xx. The new TMS320C64xx instructions have latencies from one to five cycles and all have a throughput of one cycle. (Even when instructions have multi-cycle latencies, a throughput of one cycle allows the processor to issue a new instance of the instruction in every cycle, though the results of previous instructions are not yet available.)

Instruction Set Highlights

In addition to noteworthy features of the TMS320C62xx instruction set described in Section 7.15, the TMS320C64xx adds the following noteworthy features:

- A wide range of dual/quad arithmetic and logical SIMD instructions with support for saturation

- A wide range of bit and packed-data manipulation instructions
- Compound instructions that perform more than a single basic operation (e.g., dot product, averaging instructions).
- Application-specific instructions to ease development and improve performance of typical image processing, communication and audio applications
- Instructions to improve code density (such as the dot-product instructions)

> *In addition to having a higher clock rate, the TMS320C64xx enhances the TMS320C62xx architecture with a significant number of new and powerful instructions. Although these instructions allow the processor to perform more operations in parallel, they result in a less RISC-like instruction set. The detracts from the processor's compiler-friendliness and programmability relative to that of the TMS320C62xx.*

Execution Control

Clocking

Like on other TMS320C6xxx processors, an on-chip PLL is provided on the TMS320C64xx to allow the on-chip master clock to be generated from a slower external clock. The PLL can be programmed to multiply the input clock by a factor of 1, 5, 6.67, 7.5, 10, 15, 20 or 30 on initial TMS320C64xx family members. These factors are preliminary, according to Texas Instruments.

Hardware Looping

Unlike the TMS320C62xx (which does not support hardware looping or hardware-assisted software looping), the TMS320C64xx provides hardware-assisted software looping. The TMS320C64xx adds the BDEC instruction, which combines a branch instruction with the decrement of any of the 32-bit registers that serves as a loop counter. Only one BDEC may be issued per cycle. Since BDEC can use any of the general-purpose registers as a loop counter, several BDEC instructions can be used to implement nested loops.

A new BPOS instruction can also be used to implement looping: it combines a test-positive and a branch operation. In addition, like all of the TMS320C64xx instructions, the BDEC and BPOS instructions can be predicated using a subset of general-purpose registers.

> *The new BDEC instruction replaces three instructions required to perform the same function on the TMS320C62xx, freeing up processor resources. This enhancement reduces code size and improves performance.*

Like TMS320C62xx branches, the BPEC and BPOS instructions have a latency of five cycles.

> *Because of the processor's deep pipeline and delayed branch instructions, implementing loops on the TMS320C64xx is very tricky. This leads to heavily pipelined loop implementations that result in increased program memory usage and assembly code that is difficult to write and understand.*

Interrupts

At the time of this writing, the number and sources of maskable and non-maskable interrupts for the TMS320C64xx devices have not been disclosed by Texas Instruments.

Note that like the TMS320C62xx, the TMS320C64xx is not interruptible while any execution packet in the pipeline contains a branch or is in the delay slot of a branch. The programmer can avoid this problem by unrolling loops, but this approach significantly increases code size.

Interrupts can cause unpredictable program behavior due to the exposed pipeline. For example, load instructions have a latency of five cycles. Thus, the result of a load into register A0 executed during cycle N is available in A0 during cycle N+4. An arithmetic instruction can still read the old contents of register A0 during cycle N+3. Suppose, however, that an interrupt occurs during cycle N+1. The delay caused by the interrupt means that the arithmetic instruction now executes in cycle N+4 or later, and therefore reads the new value of A0 instead of the old value of A0. Thus, interrupts must often be disabled when executing heavily optimized software.

> *The fact that the processor is not interruptible while a branch is pending means that tight loops are not interruptible on the TMS320C64xx. In applications where interrupt latency is a concern, programmers will often have to make significant sacrifices in performance (or code density) in order to ensure reasonable interrupt latency. The fact that interrupts can interfere with the TMS320C64xx's exposed pipeline further exacerbates this problem.*

Stack

Like the TMS320C62xx, the TMS320C64xx does not provide a hardware stack. A software stack can be implemented using any general-purpose register as a stack pointer. Push and pop operations can be implemented with load and store instructions using pointer pre/post-increment or decrement. Compared to the TMS320C62xx, the TMS320C64xx adds an instruction that helps reduce the number of instructions needed to set up the return address for a function call.

Bootstrap Loading

Texas Instruments had not disclosed information on the TMS320C64xx bootstrap loading features at the time of this writing. Bootstrap loading support will be specific to each TMS320C64xx device, according to Texas Instruments.

Peripherals

According to Texas Instruments, the initial TMS320C64xx family member will include a host port, a 32-channel DMA controller, three multi-channel buffered serial ports and three 32-bit timers.

- **Host port (HPI)**

  The initial TMS320C64xx device will include a 32-bit host port that extends the capabilities of the 16-bit host port found on the TMS320C6211 and TMS320C6711 processors. It can operate in two modes, referred as to HPI-16 and HPI-32 by Texas Instruments (the mode is selected at boot time). The modes support a 16-bit or 32-bit external interface, respectively. Like the TMS320C6211 and TMS320C6711 16-bit host port, the HPI-16 mode of the TMS320C64xx provides a 32-bit data path to the processor's on-chip memory with a 16-bit external interface. In HPI-16 mode the host port combines successive 16-bit transfers to provide 32-bit data to the processor's internal memory. The HPI-32 mode improves throughput by providing wider external accesses that don't require a pair of 16-bit reads or writes. A separate read and write buffer also improves the transfer throughput in autoincrement mode (in autoincrement mode the host port automatically updates the on-chip address for successive reads or writes). Preliminary information from Texas Instruments states that a maximum bandwidth of 133 Mbytes per second can be achieved by the TMS320C64xx HPI. Unlike the 32-bit expansion bus found on the TMS320C6202, TMS320C6203, and TMS320C6204 processors, which extends the TMS320C6201 16-bit host port, the 32-bit host port of the initial TMS320C64xx doesn't support synchronous memories or synchronous external peripherals.

- **DMA Controller**

  Like that of the TMS320C6211, the DMA controller of the TMS320C64xx is a chainable full-duplex multi-channel/multi-dimensional DMA controller that supports auto-initialization and an extensive range of prioritized synchronizing internal and external events. The TMS320C64xx's DMA controller supports all of the key attributes found on the TMS320C6211's EDMA controller and adds various key extensions. (The TMS320C64xx's DMA controller is also referred to as the EDMA controller by Texas Instruments.) First, the TMS320C64xx's DMA controller supports 32 channels instead of 16 channels. Second, all of the TMS320C64xx's channels can be chained whereas only four channels can be chained on the TMS320C6211. Third, four transfer request queues are provided instead of three. (Request queues support interleaved data transfer to improve the bandwidth utilization of the external bus.) Fourth, commensurate with the wider external memory interface, the TMS320C64xx's maximum external DMA bandwidth is twice that of the TMS320C62xx processors (up to 2,400 Mbytes per second for a 600 MHz TMS320C64xx device).

> *The projected TMS320C64xx DMA bandwidth should be sufficient to sustain the combined maximum bandwidth of the two external memory interfaces (EMIFA and EMIFB), the HPI, and the serial ports discussed below.*

- **Serial Ports**

  The initial TMS320C64xx device will include three multi-channel buffered serial ports. The serial port architecture of the initial TMS320C64xx device is similar to that of the TMS320C6211. As on the TMS320C62xx, each serial port supports 128 channels through time-division multiplexed data streams. The key difference between the TMS320C64xx and the TMS320C62xx serial ports is that only up to 32 out of 128 channels can be selected to be active at one time on the TMS320C62xx, compared to 128 channels on the TMS320C64xx; however, all channels can be operated without selection on both processors (all data is transferred or received). As on the TMS320C62xx, each channel on the TMS320C64xx can be configured independently for transmission and reception. The TMS320C64xx serial ports support MSB-first or LSB-first transmission like those of the TMS320C6211 (not supported on other TMS320C62xx devices). Preliminary information from Texas Instruments indicates that the bandwidth per serial port will be 100 Mbits per second.

- **Timers**

  The three timers found on the initial TMS320C64xx device feature the same attributes as those found on the TMS320C62xx processors, according to Texas Instruments.

On-Chip Debugging Support

The initial TMS320C64xx member will offer the same on-chip debugging support as the TMS320C62xx, according to Texas Instruments.

Power Consumption and Management

At the time of this writing, Texas Instruments hasn't disclosed information regarding power-down modes or projected power consumption.

Benchmark Performance

The TMS320C64xx has been benchmarked with the BDTI Benchmarks™. Overall benchmark results for all benchmarked processors are presented in Chapter 8, *BDTI Benchmark™ Results.* We summarize and analyze TMS320C64xx benchmark performance in the paragraphs below. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not reflect the processor's instruction cycle rate; therefore, lower instruction cycle counts imply a more powerful architecture, but do not imply faster speed. Next we discuss benchmark execution times and cost-execution time products, indicating processor speed and cost-performance, respectively. We then discuss the processor's energy consumption, which reflects the energy consumed by the processor in order to perform a task. Finally, we discuss the processor's memory usage. We divide the memory usage discussion into three parts: Control benchmark memory usage, overall benchmark program memory usage, and benchmark data memory usage.

The initial TMS320C64xx family members will use a two-level cache architecture. The results reported for the "TMS320C64xx-C" assume that all needed instructions and data are pre-loaded into the level-1 cache prior to each benchmark's execution. The "TMS320C64xx" results, in contrast, assume that the level-2 cache is pre-loaded with instructions and data but that the level-1 instruction and data caches aren't pre-loaded. This is a reasonable assumption for many applications, since the level-2 memory can be configured to serve as cache or on-chip RAM. If used as on-chip RAM, data and instructions can be pre-fetched into level-2 memory with a minimum cycle penalty using the DMA controller.

As described in Chapter 8, *BDTI Benchmark™ Results*, the "TMS320C64xx" results are *estimated* based on preliminary cache-miss penalty information provided by Texas Instruments. At the time of this writing, the TMS320C64xx simulator was not cycle-accurate in modeling cache miss penalties; hence, we were not able to confirm the results we report here (unlike the "TMS320C64xx-C" results, which were confirmed by simulation). We include the "TMS320C64xx" results in this report to provide an estimate of the effect of cache miss penalties on the processor's performance.

- **Instruction cycle counts:**

  As illustrated in Figure 8.1-13 in Chapter 8, *BDTI Benchmark™ Results*, the TMS320C64xx and TMS320C64xx-C total normalized instruction cycle counts are the second- and fourth-lowest among all benchmarked processors. The TMS320C64xx architecture achieves the lowest normalized instruction cycle counts on the FFT, Viterbi and Bit Unpack benchmarks. The TMS320C64xx architecture takes advantage of its dual data paths and dual and quad SIMD instructions to boost per-cycle throughput. Most benchmarks also take advantage of 64-bit load and store instructions, as well as the new unaligned load/store instructions.

  The SC140 is the only processor with a total normalized instruction cycle count result that is lower than that of the TMS320C64xx-C for most of the benchmarks (all benchmarks except the FFT, Viterbi and Bit Unpack benchmarks). The SC140 and the TMS320C64xx are both VLIW architectures with a similar level of parallelism. For example, both the SC140 and the TMS320C64xx can execute up to eight 16-bit additions per cycle or up to four 16-bit multiplies per cycle. However, the SC140 has two key advantages relative to the TMS320C64xx.

First, the SC140 supports store instructions that save the upper 16 bits of a value in a register. This feature allows the SC140 to save the result of fractional multiplies without requiring an additional shifting step. In contrast, explicit shifting is required in most cases on the TMS320C64xx. Second, the SC140 has a short pipeline, and most arithmetic instructions (such as MAC instructions) have single-cycle latency. In comparison, a single TMS320C64xx MAC operation must be implemented using a multiply instruction followed by an add instruction, with a resulting total latency of three cycles. A dual-MAC operation implemented using a SIMD dot product instruction followed by an add instruction has a total latency of five cycles. Although the effect of these long latencies can be mitigated, in some cases, via the use of optimization techniques such as software pipelining, these latencies still exert a considerable influence on the TMS320C64xx cycle counts, particularly in the single-sample benchmarks.

In most benchmarks, these two major differences are key to the cycle count differences between the two architectures. The overall normalized cycle count of the SC140 is about 30% lower than that of the TMS320C64xx-C and about 50% lower than that of the TMS320C64xx.

It is also of interest to compare the cycle counts of the TMS320C64xx to those of its predecessor, the TMS320C62xx. Although the TMS320C64xx has a much more powerful architecture and can execute twice as many 16-bit multiplies per cycle as the TMS320C62xx, it does not always have significantly lower cycle count results on the benchmarks. As a general rule, the cycle count reduction between the TMS320C62xx and the TMS320C64xx-C isn't significant in benchmarks that execute a modest number of loop iterations, such as benchmarks that process a single sample. For benchmarks in this category, the cycle counts of both the TMS320C64xx and the TMS320C62xx are both dominated by the processors' deep pipelines and instructions with multi-cycle latencies (such as multiply, load, and dot-product instructions). Where possible, software pipelining is used in benchmark implementations to reduce the effect of these latencies. However, pipelined loops require initialization and termination stages to fill and flush the instruction pipeline, adding overhead cycles outside of the loop.

Software pipelining overhead is often more significant on the TMS320C64xx than on the TMS320C62xx, for two reasons. First, in the case of convolution-oriented benchmarks, the dual TMS320C64xx 16-bit SIMD dot-product instruction has a latency of four cycles compared to two for the multiply instruction used on the TMS320C62xx. This longer latency usually results in longer loop termination code. Second, the higher parallelism offered by the TMS320C64xx is most effectively utilized via optimization techniques such as loop unrolling. However, this technique often increases the loop set-up overhead—as the loop performs more parallel instructions, more setup operations (and more cycles) are required before entering the loop (for instance, to allow input data to be loaded into registers prior to running the first iteration of the loop).

For these reasons, the TMS320C64xx architecture is at its most efficient in loops with high repetition counts, where the large initialization and termination penalties can be amortized. To illustrate, consider the Vector Dot Product benchmark. For this benchmark, the cycle count for the TMS320C62xx will be identical to that of the TMS320C64xx-C when the number of samples N is equal to sixteen, and the TMS320C64xx-C cycle count will be 1.5 times lower than that of the TMS320C62xx when N=92. (N=40 is used in the BDTI Benchmarks.)

The loop setup overhead is also larger on the TMS320C64xx than on the TMS320C62xx for the Vector Add and Vector Maximum benchmarks. However, as in the Vector Dot Product benchmark, this increase is offset by a significant reduction in cycle counts in the main loop body. The inner loop of the TMS320C62xx Vector Addition implementation executes in N/2 cycles; the TMS320C64xx implementation executes in N/4 cycles. The TMS320C64xx-C Vector Addition benchmark benefits from support for 64-bit loads. (SIMD dual-addition instructions are used in both the TMS320C64xx and the TMS320C62xx implementations, but because of its support for wide loads, the TMS320C64xx processes twice as much data per cycle in the loop.) The inner loop of the TMS320C62xx Vector Maximum implementation executes in 3*N/5 cycles; the TMS320C64xx-C implementation executes in 3*N/8 cycles. On the Vector Maximum benchmark, the TMS320C64xx-C makes good use of its dual 16-bit maximum instruction. Including the initialization overhead, the cycle counts of the Vector Addition and Vector Maximum benchmarks on the TMS320C64xx-C are roughly 30% and 3% lower, respectively, than those of the TMS320C62xx.

For the **Two-Biquad IIR** benchmark, the cycle count is two cycles *higher* on the TMS320C64xx-C than on the TMS320C62xx. It should be noted, though, that executing the TMS320C62xx implementation of the Two-Biquad IIR benchmark on the TMS320C64xx produces the same number of cycles on both architectures. Nevertheless, a different TMS320C64xx implementation of the IIR benchmark is used in this report, because the TMS320C64xx implementation (which uses the dot-product instruction) results in a 40% decrease in program memory usage at the expense of only two cycles, resulting in a better speed-memory usage tradeoff. A similar decision was made for the **Control** benchmark; although the TMS320C64xx Control benchmark implementation requires 15% more cycles than the TMS320C62xx implementation, program memory usage (the more important figure of merit on the Control benchmark) is reduced by 15%.

The cycle count of the **Bit Unpack** benchmark is about 25% lower on the TMS320C64xx-C than on the TMS320C62xx. For this algorithm, the key to the cycle count decrease is support for 64-bit non-aligned loads.

For the **FFT** and **Viterbi** benchmarks, the TMS320C64xx-C has cycle count results that are reduced by factors of 2 and 3.5, respectively, in comparison to results for the TMS320C62xx. In addition to the extensive use of new SIMD instructions in these benchmarks, the TMS320C64xx cycle count is lowered via the new application-specific instructions. The FFT benchmark requires a digit-reversal step in order to reorder the output data; this operation is efficiently implemented using the bit reversing instruction.

The TMS320C64xx-C has a very low cycle count on the Viterbi benchmark. This is mostly due to the use of 8-bit data with quad SIMD arithmetic and logical (compare) instructions in the first part of the function (for the "add-compare-select" loop). The TMS320C64xx-C can process eight trellis butterflies in just six cycles using this approach. In order to work with 8-bit data, a modified algorithm is employed that takes advantage of bounds on the differences between metrics. This requires a simple control loop to be set up to keep the metrics within the allowed 8-bit range. This is facilitated by packing four bytes into a single 32-bit word and using the CMPGTU4 instruction (which performs comparisons for greater than on packed 8-bit data). Such an optimization is only useful for processors with 8-bit SIMD capability; the TMS320C64xx is the only processor benchmarked in this study that supports quad 8-bit arithmetic and quad 8-bit compare instructions, and thus it is the only processor that can benefit from this optimization. The TMS320C64xx cycle count is also lowered by the use of the application-specific bit de-interleaving instruction (DEAL), which is intended for convolutional error correction algorithms.

Cache Impact

As discussed in Chapter 8, *BDTI Benchmark™ Results*, BDTI used preliminary cache miss penalty information provided by Texas Instruments to generate estimates of the effects of cache misses on BDTI Benchmark cycle counts for the TMS320C64xx.

The estimated level-1 cache miss impact on the TMS320C64xx cycle count results varies across the BDTI Benchmarks. The impact of flushing the level-1 caches prior to benchmark execution ranges from almost no impact (less than 1% on the cycle count for the Control benchmark) to an increase of about 85% (for the Single-Sample FIR). On average, not preloading the level-1 caches causes cycle counts to increase by about 45%.

> *The estimated effect of cache misses on the cycle counts of the TMS320C64xx is quite large compared to the instruction cache impact found on Analog Devices' ASDP-2116x or ASDP-219x processors (about 10% and 5% respectively).*

It is worth noting that despite the significant impact of not preloading the level-1 caches, the cycle counts for the TMS320C64xx on the FFT, Viterbi, and Bit

Unpack benchmarks are still the lowest of all benchmarked processors, even when the level-1 caches are not preloaded. Not preloading the level-1 caches increases the TMS320C64xx cycle count by more than 30% only when benchmarks have fairly small cycle counts (below 200 cycles). For larger benchmarks, the level-1 cache impact is lower than 20%. Keep in mind, however, that the figures estimated here do not take into account penalties associated with L2 cache misses.

- **Execution times:**

  The TMS320C64xx-C's low cycle counts and high projected instruction execution rate (600 MHz) make it the fastest of all benchmarked processors by a significant margin. The projected total normalized execution time presented in Figure 8.2-13 shows that the TMS320C64xx-C is approximately 30% faster than the second-fastest processor, the MCS8101. The impact of not preloading the level-1 caches causes the TMS320C64xx to have an estimated total normalized execution time that is slightly (about 5%) slower than that of the MCS8101.

  Compared to the TMS320C6203, the TMS320C64xx-C is roughly 2.4 times faster. The total normalized execution time of the TMS320C64xx-C is roughly 10 times faster than the average for fixed-point processors. This factor is reduced to about 7 times when the level-1 caches are not pre-loaded.
- **Cost-execution time:** At the time of this writing, Texas Instruments has not disclosed pricing information for TMS320C64xx family members. Thus, the TMS320C64xx is not evaluated on this metric.
- **Energy consumption:** At the time of this writing, Texas Instruments hasn't disclosed power consumption information for the initial TMS320C64xx family members. Thus, the TMS320C64xx is not evaluated on this metric.

Memory Usage

The focus in the memory usage analysis is on Control benchmark memory usage. Unlike other benchmarks, the Control benchmark is optimized for minimum memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP code. Finally we discuss constant and non-constant data memory usage.

Since memory usage is independent of the state of the cache, we do not distinguish between the TMS320C64xx and the TMS320C64xx-C when discussing memory usage.

- **Control benchmark memory usage:** As presented in Figure 8.5-9A, the TMS320C64xx's total memory usage on the Control benchmark is the second-highest of the fixed-point processors benchmarked, after the TMS320C62xx. The TMS320C64xx's total memory usage on the Control benchmark is about 45% higher than the average of fixed-point processors, and equal to the average of all benchmarked processors. Compared to the TMS320C62xx, the program memory use of the TMS320C64xx implementation on the Control benchmark is about 15% lower.

The TMS320C64xx has lower total memory usage than the TMS320C62xx on this benchmark for two reasons. First, the TMS320C64xx implementation takes advantage of the new "BNOP" and "ADDKPC" instructions. The BNOP instruction performs a branch instruction followed by a multi-cycle NOP instruction (i.e., a NOP instruction that executes for a given number of cycles). The multi-cycle NOP instruction is primarily intended to consume a specified number of cycles so that none of the instructions that follow the branch instruction in memory are executed while the branch is still pending. To accomplish this on the TMS320C62xx, a separate branch (B) and a multi-cycle latency NOP instruction are needed, consuming twice as many instructions to achieve the same effect. Similarly, the ADDKPC instruction helps to reduce the number of instructions needed when a return address is set up for a function call. ADDKPC combines the saving of the program counter and a multi-cycle NOP instruction that has the same purpose as the NOP operations of the BNOP instruction.

The second reason for the reduced memory usage comes from a key architectural difference between the TMS320C62xx and TMS320C64xx. Unlike the TMS320C62xx, groups of instructions to be executed in parallel (referred to as an "execute packet") aren't restricted to be entirely contained in a group of instructions read from memory at one time (referred to as a "fetch packet"). This is discussed in more detail in the *Pipeline* section above. On the TMS320C62xx, an execute packet that would otherwise cross a fetch packet boundary is automatically aligned by the compiler or assembler using NOPs. The NOP instructions are executed in parallel with the other instructions in the execute packet, and don't affect performance. They do, however, consume program memory. Even though the TMS320C62xx implementation of the Control benchmark is optimized to avoid such mis-aligned execute packets where possible, the TMS320C64xx still achieves lower program memory usage.

- **Program memory usage:** The total normalized program memory usage for the TMS320C64xx is the second-highest of all benchmarked fixed-point processors, after the TMS320C62xx. Program memory usage results are presented in Figure 8.5-14.

  The program memory usage of the TMS320C64xx is about 15% lower than that of the TMS320C62xx, for two reasons; first, NOPs aren't required to align execution packets on the TMS320C64xx increases code density. Second, in general, the higher parallelism of TMS320C64xx instructions increases code density because the TMS320C64xx can often accomplish more work in a single instruction than can the TMS320C62xx. It should be noted, though, that on some benchmarks the TMS320C64xx uses *more* program memory than does the TMS320C62xx. Both processors' memory usage is increased due to frequent use of loop unrolling to optimize the benchmarks; however, this technique increases program memory usage more on the TMS320C64xx than on the TMS320C62xx. As previously discussed, this is because, compared to the TMS320C62xx, the higher level of parallelism on the TMS320C64xx motivates programmers to unroll loops more than with the TMS320C62xx. For this reason, four benchmarks out of twelve (Real Block FIR, Single-Sample FIR, LMS Adaptive, and Vector Maximum benchmarks) require more program memory on the TMS320C64xx than on the TMS320C62xx.

To better measure the impact of NOP padding in execution packets, we measured the program memory usage of the TMS320C62xx benchmarks after reassembling them for the TMS320C64xx. Results from this experiment show that on average, program memory usage is about 10% lower on the TMS320C64xx than on the TMS320C62xx due to the removal of alignment requirements.

Despite this improvement, the program memory usage of the TMS320C64xx is among the highest of all of the processors benchmarked. Among fixed-point processors, only the TMS320C62xx has a higher program memory usage. In contrast, the SC140, which uses 16-bit instructions, has a program memory usage approximately half that of the TMS320C64xx.

Even despite their 48-bit instruction words, Analog Devices' ADSP-2106x and ADSP-2116x floating-point processors achieve a total normalized program memory usage that is about half that of the TMS320C64xx.

- **Data memory usage:** The TMS320C64xx constant data memory usage is as expected for a 16-bit fixed-point DSP processor. Constant data memory usage is shown in Figure 8.5-14.

  Overall, the non-constant data memory usage of the TMS320C64xx is also as expected for a 16-bit fixed-point DSP processor, except for the Viterbi benchmark. Normalized non-constant data memory usage is shown in Figure 8.5-15.

  On the Viterbi benchmark, the TMS320C64xx has a non-constant data memory usage that is among the lowest of all benchmarked processors. One reason for this is the use of 8-bit words to store the samples of the input soft decision array instead of a larger word as used by other architectures (in most cases, 16-bit words are used on the other fixed-point DSPs). On the TMS320C64xx, 8-bit words are used in conjunction with quad 8-bit SIMD arithmetic and logical instructions. This allows the TMS320C64xx to achieve both a low cycle count and low memory usage for this benchmark. As mentioned earlier, the TMS320C64xx is the only processor benchmarked in this study featuring support for quad 8-bit arithmetic and quad 8-bit compare instructions that are useful for the Viterbi benchmark.

Cost

Texas Instruments has not yet announced pricing for TMS320C64xx products.

Fabrication Details

The first member of the TMS320C64xx family will be fabricated using a 0.12 µm six-metal-layer CMOS process, according to Texas Instruments.

Development Tools

The TMS320C64xx shares the same assembler, linker, archiver, instruction-set simulator, C/C++ compiler, and assembly optimizer as the TMS320C62xx and the TMS320C67xx. The Code Composer Studio environment (CCS) is also common to the TMS320C62xx, TMS320C67xx and TMS320C64xx devices. Please refer to Section 7.15 for discussion of the TMS320C62xx/TMS320C67xx/TMS320C64xx development tools.

Version 4 of the C/C++ compiler used at the time of this writing is not able to *automatically* take advantage of the TMS320C64xx dot-product instruction nor is it able to automatically use the new SIMD instructions. C/C++ programmers can force the compiler to issue these instructions by inserting intrinsics or compiler directives in their source code. At the time of this writing, the TMS320C64xx simulator doesn't model the effect of the level-1 and level-2 caches or of external memory accesses on code execution time. A cycle-accurate simulator is expected to be provided by the first quarter of 2001, according to Texas Instruments.

Note that the TMS320C62xx assembly optimized digital signal processing library and the TMS320C62xx image processing library, which are distributed without charge by Texas Instruments, can be used with the TMS320C64xx, although the instruction set extensions of the TMS320C64xx aren't used in these libraries. Optimized DSP libraries for the TMS320C64xx are expected to be available in the second quarter of 2001, according to Texas Instruments.

A TMS320C64xx-based evaluation board (similar to the TMS320C62xx and TMS320C67xx EVM boards) is scheduled for beta release in the second quarter of 2001, and for production by the second half of 2001, according to Texas Instruments.

Third-party support for the TMS320C64xx exists or is planned in the form of development boards, emulators, application boards, development tools and software libraries from a variety of vendors.

Applications Support

The TMS320C64xx shares most of its documentation with the TMS320C62xx and TMS320C67xx. This includes reference and programmer's guides and a peripherals guide. Separate data sheets will discuss the hardware aspects of specific devices.

Applications support for all TMS320 family processors is provided by Texas Instruments staff via telephone hotline, fax, and Internet electronic mail.

Texas Instruments also provides a website that allows TMS320 users to download code and application notes.

Advantages

- High level of parallelism
- Flexible support for SIMD processing
- Good parallel move support
- Support for non-aligned load and stores
- Conditional instruction execution
- Four ALUs, two multipliers, two barrel shifters
- Powerful and flexible SIMD MAC instructions
- Exponent detect instruction
- Bit manipulation and bit-field manipulation instructions (and support for bit-reversed addressing)
- Dedicated instructions for video and image processing applications, communication applications and audio processing applications
- Single-cycle throughput of all instructions
- Flexible external memory interface: ROM, asynchronous SRAM, synchronous burst SRAM, and synchronous DRAM support; DRAM refresh generation
- On-chip DMA controller with dedicated address and data buses
- Powerful DMA controller with many features and high bandwidth
- JTAG emulation port with boundary scan
- Three serial ports, three timers and two independent external memory interfaces
- Code compatibility with previous generation
- Very good projected execution times on the BDTI Benchmarks

Disadvantages

- Long instruction latencies which are different for different instructions
- Two-cycle multiplier latency
- Exposed pipeline complicates programming and interferes with ability to service interrupts
- Many variants of instructions performing the same task (e.g., more than 12 multiply instructions)
- Difficult to program
- Complex resource constraints when processing 40-bit data
- Cache-based architecture complicates software optimization and creates difficulties in guaranteeing real-time performance
- Processor not yet available at the time of this writing
- High program memory usage on the BDTI Benchmarks

7.17 Texas Instruments TMS320C67xx Family

BDTImark2000 Score: 820 at 167 MHz

Introduction

The TMS320C67xx family is the 32-bit floating-point version of Texas Instruments' TMS320C62xx family of 32-bit fixed-point DSP processors. The TMS320C67xx instruction set is a superset of the TMS320C62xx instruction set, adding floating-point arithmetic to the TMS320C62xx instruction set. The TMS320C67xx family is a high-performance, VLIW-based architecture targeting performance-hungry applications such as wireless base stations, digital subscriber loops, 3D graphics, medical imaging, radar, and speech recognition.

Like its fixed-point counterpart, the TMS320C67xx uses a VLIW-like architecture with eight execution units that include two multipliers and four ALUs. Using these eight execution units, the processor can execute up to eight 32-bit RISC-like instructions in a single clock cycle, enabling it to achieve a high level of parallelism.

Because the TMS320C67xx can execute a group of up to eight parallel instructions per clock cycle, the term "instruction cycle" is potentially ambiguous when discussing this processor. As used here, "instruction cycle" refers to the minimum time required to issue one group of parallel instructions. On the TMS320C67xx, this time is equal in length to one master clock cycle; i.e., one group of parallel instructions can be issued during every cycle of the master processor clock.

The first member of the TMS320C67xx family, the TMS320C6701, was announced in April of 1998. The TMS320C6701 executes at a maximum 167 MHz clock rate with a 1.9-volt core supply and a 3.3-volt I/O supply. At 167 MHz, the TMS320C67xx executes up to 334 million floating-point MACs per second. In March 1999, Texas Instruments announced a new TMS320C67xx member, the TMS320C6711. The TMS320C6711 is a reduced-cost version of the TMS320C6701 (a 100 MHz TMS320C6711 costs $24, compared to $139 for the 167 MHz TMS320C6701). The TMS320C6711 provides on-chip caches for data and instructions. In September 2000, the lowest-cost family member was announced, the TMS320C6712. The TMS320C6712 eliminates the host port interface and operates at 100 MHz. The TMS320C6712 is currently available in sample quantitates, with volume production planned for the first quarter of 2001, according to Texas Instruments. Like the TMS320C6711, the TMS320C6712 is a cache-based architecture, and features the same amount of on on-chip memory as the TMS320C6711. Table 7.17-1 shows the key characteristics of the TMS320C67xx processors.

The TMS320C67xx is backward-compatible with the TMS320C62xx; the TMS320C67xx can execute TMS320C62xx object code unmodified, but the TMS320C62xx cannot execute all TMS320C67xx instructions. The TMS320C67xx is not object-code compatible with the TMS320C64xx, Texas Instruments' next generation of the fixed-point TMS320C6xxx architecture, since the TMS320C64xx extends the

TMS320C62xx instruction set with instructions that aren't supported by the TMS320C67xx.

Because the TMS320C67xx and TMS320C62xx are similar, this analysis highlights only the differences between the two processors. Readers should refer to Section 7.15 for a full discussion of the TMS320C62xx architectures and for details of features identical between the two families.

> *The TMS320C62xx and TMS320C67xx are currently the only DSP processor families to offer instruction set compatibility between fixed- and floating-point processors. This can be an advantage in applications where algorithms are initially designed using floating-point arithmetic and are later converted to a fixed-point implementation for volume production, because control code from the floating-point implementation does not need to be re-written for the fixed-point version. However, due to the different latencies of fixed- and floating-point instructions and other restrictions, and because of differences in numeric fidelity, algorithm kernels will almost always have to be completely re-written when migrating from a floating-point to a fixed-point implementation.*

Device	Operating Voltage (V)	Maximum Speed (Millions of MACs per Second)	On-Chip Memory		Notes
			Program RAM	Data RAM	
C6701	1.9 / 3.3[1]	334	16K×32	16K×32	4-channel DMA, 16-bit host port interface, 32-bit external memory interface, two buffered serial ports, two timers
C6711	1.8 / 3.3[2]	300	128×256 L1 cache	512×64 L1 cache	16-channel enhanced DMA, 16-bit host port interface, 32-bit external memory interface, two buffered serial ports, two timers
			8K×64 unified L2 cache		
C6712	1.8/3.3[2]	200	128×256 L1 cache	512×64 L1 cache	16-channel enhanced DMA, two serial ports, 16-bit external memory interface, two timers
			8K×64 unified L2 cache		

TABLE 7.17-1. TMS320C67xx characteristics.

[1] The core operates at 1.9 volts while all I/O signals are 3.3-volt compatible.

[2] The core operates at 1.8 volts while all I/O signals are 3.3-volt compatible.

The low-cost TMS320C6712 is intended to provide an alternative to Analog Devices' low-cost ADSP-2106x (SHARC) processors; the 100 MHz TMS320C6712 cost is expected to sell for $21.95 in quantity 10K units, according to Texas Instruments, compared to roughly $25.00 for a 60 MHz ADSP-2106x processor.

Architecture

The core architecture of the TMS320C67xx family consists of two floating-point data paths, a program control unit (including program fetch, instruction dispatch, and instruction decode units), and program and data memory interfaces. Figure 7.17-1 illustrates the TMS320C67xx family architecture as typified by the TMS320C6701.

Data Path

The TMS320C67xx's two floating-point data paths provide a superset of the functionality of the fixed-point data paths of the TMS320C62xx, adding support for IEEE-754

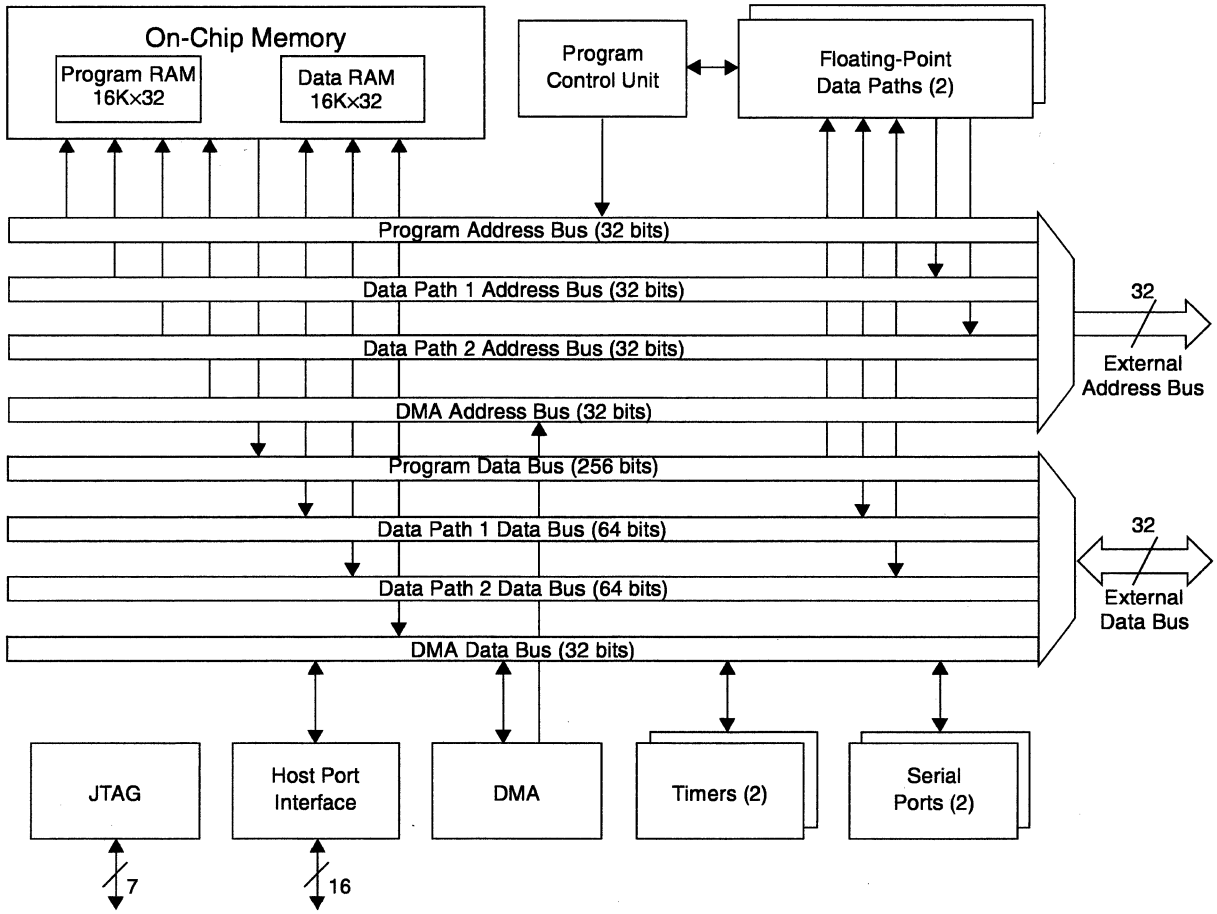

On-Chip Memory
Program RAM 16K×32
Data RAM 16K×32
Program Control Unit
Floating-Point Data Paths (2)
Program Address Bus (32 bits)
Data Path 1 Address Bus (32 bits)
Data Path 2 Address Bus (32 bits)
DMA Address Bus (32 bits)
Program Data Bus (256 bits)
Data Path 1 Data Bus (64 bits)
Data Path 2 Data Bus (64 bits)
DMA Data Bus (32 bits)
32
External Address Bus
32
External Data Bus
JTAG
Host Port Interface
DMA
Timers (2)
Serial Ports (2)
7
16

FIGURE 7.17-1. TMS320C6701 processor architecture.

32-bit single-precision and 64-bit double-precision floating-point arithmetic. Since TMS320C67xx general-purpose registers are 32 bits wide, 64-bit floating-point values are supported by pairing an even-numbered register with the next higher odd-numbered register. For example, registers A2 and A3 can be paired to hold a single 64-bit floating-point value.

> *The TMS320C67xx is the first mainstream DSP to support IEEE-754 double-precision floating-point arithmetic. Because the TMS320C67xx memory bandwidth and execution units are much better suited for single-precision floating-point and fixed-point arithmetic, however, the speed of double-precision floating-point arithmetic will be several times slower than the performance of single-precision floating-point arithmetic on this processor. The single-precision floating-point format will be sufficient for most applications. In the rare case where an application needs to make light use of double-precision floating-point arithmetic, the TMS320C67xx will have an advantage over other floating-point DSPs.*

Specific differences between the TMS320C62xx data paths and the TMS320C67xx data paths are listed below:

- The **L units** on the TMS320C67xx have added support for 32- and 64-bit floating-point arithmetic operations, conversions between fixed- and floating-point formats, and conversion from 64- to 32-bit floating-point format.
- The **S units** on the TMS320C67xx have added support for 32- and 64-bit floating-point absolute value, comparison, reciprocal seed, and reciprocal-square-root seed operations, and conversion from 32- to 64-bit floating-point format.
- The **M units** on the TMS320C67xx have added support for 32- and 64-bit floating-point multiplication, and 32-bit by 32-bit integer multiplication.
- The **D units** on the TMS320C67xx have added support for 64-bit loads and related address computations.
- The latencies of floating-point operations on the TMS320C67xx are typically longer than those of the corresponding fixed-point operations. Additionally, while all fixed-point operations on the TMS320C62xx and TMS320C67xx have single-cycle throughput, double-precision floating-point operations and 32-bit by 32-bit integer multiplications on the TMS320C67xx often have throughput longer than one cycle. When an instruction with throughput longer than one cycle uses a cross path to access the register file of the other data path, the cross path cannot be used by subsequent instructions until the functional unit using it becomes available. That is, the throughput of the instruction applies to both the functional unit and the cross path used by the instruction.

> *The long and varying latencies and throughputs of floating-point instructions complicate programming. Additionally, the fact that latencies of floating-point instructions differ from those of the corresponding fixed-point instructions complicates the conversion of algorithms from floating-point to fixed-point implementation.*
>
> *The multi-cycle throughput of many double-precision floating-point operations and the restrictions on using the cross paths while performing these operations mean that the speed of algorithms that use double-precision floating-point arithmetic will be several times slower than that of the same algorithm using single-precision arithmetic.*

- The TMS320C67xx provides three control registers that contain status and configuration bits for floating-point arithmetic operations. Configuration bits allow one of four rounding modes to be selected for each of the L, S, and M units. Rounding modes include convergent rounding, round toward zero, round toward positive infinity, and round toward negative infinity. Status bits for each of the L, S, and M units indicate the occurrence of overflow or underflow, whether a source operand is denormalized or "Not a Number" (NaN), whether a result is infinite, inexact, or invalid, and whether the source operand for a reciprocal operation was zero.

On the fixed-point TMS320C62xx, only one 40-bit fixed-point result can be written to each register file per instruction cycle. On the TMS320C67xx, this restriction also applies to 64-bit floating-point results.

Memory System

The on-chip memory system of the TMS320C67xx is similar to that of the TMS320C62xx, but improves data memory bandwidth to accommodate 64- and 32-bit floating-point data as opposed to 32- and 16-bit fixed-point data. Each of the two data paths on the TMS320C67xx can load a 64-bit double-word per instruction cycle, compared to a 32-bit word on the TMS320C62xx. This is accomplished via an additional bus in each data path that can be used to load the most significant 32 bits of a 64-bit double-word.

> *Only one 40-bit or 64-bit result can be written to each register file per instruction cycle, as described above. This can limit the processor's ability to make use of 64-bit loads in applications that use double-precision floating-point arithmetic or 40-bit fixed-point arithmetic operations. Since use of double-precision floating-point arithmetic or combined 64-bit loads and 40-bit arithmetic should be rare, however, this restriction will not be a problem in most applications.*

As on the TMS320C62xx, all memory accesses on the TMS320C67xx must be aligned. For example, this means that when a 32-bit access is performed, the address of the memory location accessed must be a multiple of 4. For 64-bit accesses, the address must be a multiple of 8. If an address is not aligned, no exception is generated. Instead, the next-lowest aligned address is used.

> *The fact that memory accesses must be aligned complicates programming and sometimes reduces performance. For example, in filters that require a delay line to be maintained to store successive inputs, the delay line is typically implemented using a circular buffer, and the starting location of the buffer is advanced by one sample for each iteration of the convolution. In order to use the full processing power of the TMS320C67xx, it is necessary to load a pair of samples from the delay line via a single read. Due to the alignment requirement, though, this cannot be done if a circular buffer is used. Hence, filter implementations on the TMS320C67xx often use a different approach to implementing delay lines.*

The on-chip data memory of the TMS320C6701 consists of two blocks of 32 Kbytes each. Each block is further divided into eight banks of single-access 16-bit memory (by comparison, each block of on-chip data memory on the TMS320C62xx has four banks). This memory can be accessed 8 or 16 bits at a time, or 32 or 64 bits at a time when two or four memory banks are combined. Via the two 32-bit address and four 32-bit data buses, it is possible to fetch two 64-bit data operands at a time assuming that each 16-bit suboperand comes from a different memory bank. Assuming no memory bank conflicts, the maximum on-chip data access rate is thus two 64-bit double-words or four 32-bit words (if located as pairs of adjacent words) per instruction cycle. On a 167 MHz TMS320C6701 this results in a maximum sustainable on-chip data memory bandwidth of 334 million 64-bit double words or 668 million 32-bit words per second.

> *Most older DSP processors provide a single data path and support an on-chip data access rate of two data memory accesses per instruction cycle. On the TMS320C67xx, if two 32-bit operands are packed in one 64-bit double data word, four 32-bit operands can be fetched in one instruction cycle when accessed as groups of adjacent memory words. This provides sufficient data memory bandwidth for the processor's two data paths.*

As on the TMS320C6201, the 64 Kbyte on-chip program memory of the TMS320C6701 can be dynamically configured to act as a program cache.

The TMS320C6711 has the same two-level on-chip cache organization as the TMS320C6211. As with the TMS320C6211, the level-2 memory can be configured to serve as cache or as on-chip RAM. The amount of memory in each cache (i.e., the level-1 data/instruction caches and unified level-2 cache), the cache line size, cache replacement policy and cache miss penalties are the same on the TMS320C6711 and the TMS320C6211. The buses between on-chip memories, the DMA controller and the core have the same width as on the TMS320C6211, except for the 64-bit read paths between the TMS320C6711 core and its level-1 data cache. This extension allows for single-cycle 64-bit loads from the level-1 data cache. (Store operations still use a 32-bit write path between the core and the level-1 data cache.) Figure 7.17-2 shows the memory organization of the TMS320C6711.

As with the TMS320C6211, the TMS320C6711 level-1 and level-2 caches can be explicitly flushed by the program during execution. Also as with the TMS320C6211, the external memory is partitioned into sub-regions that can individually configured to be cached or to be directly accessed, bypassing the cache.

The cache architecture of the TMS320C6712 is identical to that of the TMS320C6711, according to Texas Instruments.

External Memory Interface

The TMS320C6701 32-bit external memory interface is identical to that of the TMS320C6201. As with all processors, off-chip memory bandwidth depends heavily on the type and speed of the external memory. In the fastest case, i.e., when connected to high-speed synchronous-burst SRAM, the first memory access in a series requires additional cycles, but subsequent accesses can be performed in a single instruction cycle. In

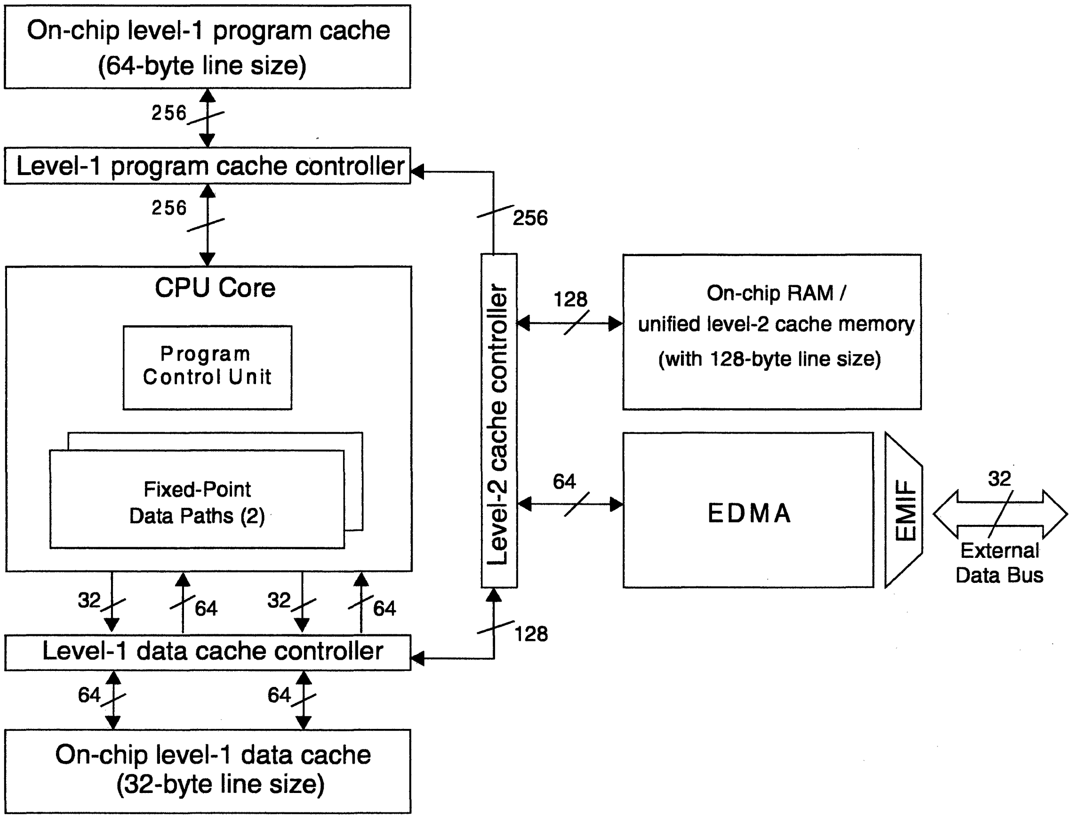

On-chip level-1 program cache
(64-byte line size)
256
Level-1 program cache controller
256
256
CPU Core
Program Control Unit
Fixed-Point Data Paths (2)
Level-2 cache controller
128
On-chip RAM / unified level-2 cache memory (with 128-byte line size)
64
EDMA
EMIF
32
External Data Bus
32
64
32
64
128
Level-1 data cache controller
64
64
On-chip level-1 data cache
(32-byte line size)

FIGURE 7.17-2. TMS320C6711 cache organization.

this configuration, the TMS320C6701 achieves a peak access rate of one 32-bit word per instruction cycle. This corresponds to a peak external memory bandwidth of 167 million 32-bit words per second on a 167 MHz TMS320C6701. The external memory bandwidth for other types of memories is significantly lower; e.g., up to 83 million 32-bit words per second for asynchronous SRAM and synchronous DRAM. Additionally, the peak memory bandwidth for any particular type of memory can only be achieved via DMA transfers and instruction fetches. When accessing external memory via load or store instructions, wait states are always incurred. The number of wait states incurred for load and store instructions varies depending on the type of memory used and whether a load or a store is being performed, and ranges from six cycles for a store to synchronous-burst SRAM or asynchronous SRAM to as many as 42 cycles for a load from an inactive row of SDRAM. Wait states are incurred for each 32-bit word accessed—performing loads or stores from or to sequential external memory locations does not eliminate the wait states after the first access.

> *As with the TMS320C62xx, external memory bandwidth will be a significant bottleneck in many applications using the TMS320C67xx. Please refer to the TMS320C62xx analysis for a more detailed discussion of this issue. Because applications using 32-bit floating-point data may require twice the data memory bandwidth of similar applications using 16-bit fixed-point data, the external memory bandwidth bottleneck will have a greater impact on the TMS320C67xx in comparison to that on the TMS320C62xx.*

The external memory interface of the TMS320C6711 is similar to that of the TMS320C6211 but supports more memory device types. Unlike the TMS320C6701 and TMS320C6711, the external memory interface of the TMS320C6712 is 16 bits wide. At the time of this writing, no other information regarding the TMS320C6712 external memory interface was available.

> *The memory bottleneck discussed in the previous comment will likely be even more severe on the TMS320C6712 due to its 16-bit external memory interface.*

Address Generation Units

The D1 and D2 address generation units on the TMS320C67xx are similar to those of the TMS320C62xx, but add support for 64-bit data addressing as described below.

- A 64-bit load instruction allows a pair of 32-bit registers consisting of an even numbered register and the next-highest odd numbered register to be loaded with a 64-bit double-word.
- A 64-bit address calculation instruction allows an index into an array of 64-bit double-words to be shifted left by three bits and added to the base address of an array, producing the byte address of the desired element. The index and base

address come from general-purpose registers and the value loaded is placed in a general-purpose register.

Pipeline

Compared to the TMS320C62xx pipeline, the TMS320C67xx pipeline adds five new execute stages, resulting in a pipeline depth of 16 stages. The TMS320C67xx pipeline stages are summarized in Table 7.17-2.

The pipeline of the TMS320C67xx is by far the deepest of those found in other currently available mainstream DSP processors.

The fetch and decode stages of the TMS320C67xx pipeline are identical to those of the TMS320C62xx. Additionally, all TMS320C62xx instructions use the same number of execute stages on the TMS320C67xx as they do on the TMS320C62xx. Floating-point instructions on the TMS320C67xx use from one to ten execute stages. Most TMS320C67xx instructions have single-cycle throughput. However, as mentioned above, 32-bit by 32-bit fixed-point multiplies and most double-precision floating-point instructions have throughputs longer than one cycle. For example, double-precision floating-point additions have a latency of six cycles and a throughput of one addition every two cycles. If a program issues an instruction to the same execution unit in the cycle following a double-precision addition, the new instruction will be discarded, unless the double-precision addition instruction is conditional and the condition was false. TMS320C67xx instruction latencies and throughput are shown in Table 7.17-3.

As on the TMS320C62xx, all branches on the TMS320C67xx are delayed branches introducing five delay slots. This is because the branch instruction is executed in the E1 stage, and all previously fetched instructions are dispatched before the branch takes

Operation	Stage	Description
Fetch	PG	Program address generate
	PS	Program address send
	PW	Program address ready wait
	PR	Program fetch packet receive
Decode	DP	Instruction dispatch
	DC	Instruction decode
Execute	E1	Execute stage 1
	E2-E10	Execute stages 2-10 (used by floating-point, fixed-point multiply, and load instructions)

TABLE 7.17-2. Pipeline stages of the TMS320C67xx.

effect. Additionally, memory access conflicts and interrupts can stall the TMS320C67xx pipeline in the same way that they stall the TMS320C62xx pipeline.

> *Pipeline effects significantly complicate TMS320C67xx assembly programming. Because the pipeline is both deep and completely exposed, writing assembly code that is both correct and efficient can be extremely tricky. Due to the long latencies of floating-point instructions, pipeline effects complicate TMS320C67xx programming more severely than on the TMS320C62xx.*

Instruction Set

The TMS320C67xx instruction set is a superset of the TMS320C62xx instruction set. TMS320C67xx registers are identical to those found on the TMS320C62xx.

Instruction	Latency	Throughput
Load	5	1
Branch	6	1
Fixed-point arithmetic and logic	1	1
Fixed-point 16-bit by 16-bit multiplication	2	1
Single-precision floating-point add, subtract, multiply; floating-point to integer conversion, integer to single-precision floating-point conversion; double-precision to single-precision floating-point conversion	4	1
Double-precision floating-point add and subtract	7	2
Double-precision floating-point multiply, 32-bit by 32-bit integer multiply with 64-bit result	10	4
32-bit by 32-bit integer multiply with 32-bit result	9	4
Single-precision floating-point absolute value, compare, reciprocal estimate, reciprocal square root estimate	1	1
Double-precision floating-point absolute value, reciprocal estimate, reciprocal square root estimate, single- to double-precision floating-point conversion	2	1
Double-precision floating-point compare	2	2
Integer to double-precision floating-point conversion	5	1

TABLE 7.17-3. Instruction latencies and throughput on the TMS320C67xx.

TMS320C67xx instructions not supported in the TMS320C62xx instruction set are summarized in Table 7.17-4.

Assembly Language Format

The TMS320C67xx uses the same assembly language syntax as the TMS320C62xx.

Parallel Move Support

The TMS320C67xx has the same parallel move support as the TMS320C62xx. In addition, the TMS320C67xx also supports 64-bit double-word loads but not stores.

Orthogonality

The TMS320C67xx instruction set has about the same level of orthogonality as that of the TMS320C62xx. The TMS320C67xx support of double-word loads but not stores detracts slightly from its orthogonality.

Execution Times

As discussed in the *Pipeline* section above, fixed-point arithmetic, logic, load/store, and branch instructions on the TMS320C67xx have the same latency and throughput as on the TMS320C62xx. Floating-point operations have latencies from one to ten cycles. Double-precision floating-point operations also have reduced throughput as described above in the *Pipeline* section. The latency and throughput of floating-point instructions are summarized in Table 7.17-3 above.

Class	Instructions
Arithmetic	Single- and double-precision floating-point add, subtract, absolute value
Multiplication	Single- and double-precision floating-point multiply, 32 × 32 → 64 integer multiply, 32 × 32 → 32 integer multiply
Data move	Double-word (64-bit) load, double-word address calculation
Comparison	Single- and double-precision floating-point compare
Special Function	Single- and double-precision floating-point reciprocal estimate and reciprocal square-root estimate; fixed-point/floating-point conversions; single-precision/double-precision floating-point conversions

TABLE 7.17-4. TMS320C67xx instructions not found in the TMS320C62xx instruction set. Note that all instructions can be executed conditionally.

Instruction Set Highlights

In addition to noteworthy features of the TMS320C62xx instruction set, the TMS320C67xx instruction set includes the following noteworthy features:

- Double-precision floating-point support
- Floating-point reciprocal estimate and reciprocal square-root estimate instructions
- Instructions to convert between fixed- and floating-point formats

Execution Control

Clocking

The TMS320C6701 clocking features are the same as those of the TMS320C6201. TMS320C6711 clocking is the same that of the TMS320C6211. The TMS320C6712 is expected to operate at a maximum frequency of 100 MHz. No further information regarding TMS320C6712 clocking features was available at the time of this writing.

Hardware Looping

Like the TMS320C62xx, the TMS320C67xx does not support hardware looping. Therefore all loops must be implemented in software. However, the parallel architecture of the processor allows the implementation of software loops with virtually no overhead.

> *Due to the deep pipeline and the delayed branch instructions of the processor, implementing efficient loops on the TMS320C67xx in assembly language is very tricky. The C compiler confronts these same issues when generating code for loops. This leads to heavily pipelined loop implementations that result in increased program memory usage and assembly code that is difficult to write and understand.*

Interrupts

The TMS320C67xx provides the same interrupt support as the TMS320C62xx. Note that the processor is not interruptible while any execution packet in the pipeline contains a branch or is in the delay slot of a branch. Additionally, interrupts can cause unpredictable program behavior due to the exposed pipeline. For example, load instructions have a latency of five cycles. Thus, the result of a load into register A0 executed during cycle N is available in A0 during cycle N+4. An arithmetic instruction can still read the old contents of register A0 in cycle N+3. Suppose, however, that an interrupt occurs during cycle N+1. The delay caused by the interrupt means that the arithmetic instruction now executes during cycle N+4 or later, and therefore reads the new value of A0 instead of the old value of A0. Therefore, interrupts must often be disabled when executing efficient code.

> *The fact that the processor is not interruptible while a branch is pending means that tight loops are not interruptible on the TMS320C67xx. In applications where interrupt latency is a concern, programmers will often have to make significant sacrifices in performance in order to ensure reasonable interrupt latency. The fact that interrupts can interfere with the TMS320C67xx's exposed pipeline further exacerbates this problem.*

Stack

Like the TMS320C62xx, the TMS320C67xx does not provide a hardware stack.

A software stack can be implemented using any general-purpose register as a stack pointer. Push and pop operations can be implemented using load and store instructions with address pre/post-increment/decrement.

Bootstrap Loading

The TMS320C6701 provides the same bootstrap loading capabilities as the TMS320C6201. The TMS320C6711 provides the same bootstrap loading capabilities as the TMS320C6211. No specific information regarding the bootstrap loading capabilities of the TMS320C6712 was available at the time of this writing.

Peripherals

The TMS320C6701 provides the same peripherals as the TMS320C6201. The TMS320C6711 provides the same peripherals as the TMS320C6211.

The TMS320C6712 includes a DMA controller, two timers, and two serial ports, all of which are expected to be identical to those of the TMS320C6711. Unlike the TMS320C6711, the TMS320C6712 doesn't have a host port.

> *Unlike Analog Devices' floating-point ADSP-2106x and ADSP-2116x families, the TMS320C67xx family does not offer extensive I/O features to facilitate the design of multiprocessor systems. This is somewhat surprising, as floating-point DSPs are often used in multiprocessor designs. Texas Instruments' earlier TMS320C4x family did provide multiprocessor-oriented I/O features.*

On-Chip Debugging Support

The TMS320C67xx offers the same on-chip debugging support as the TMS320C62xx.

Power Consumption and Management

Typical power consumption data provided by Texas Instruments is summarized in Table 7.17-5.

The TMS320C6701 provides the same power management features as the TMS320C6201 and TMS320C6211. The power management features of the TMS320C6712 are expected to be identical to those of the TMS320C6711.

> *The fact that the 167 MHz and 150 MHz versions of the TMS320C6701 processors require a different voltage for the core (i.e., 1.9 or 1.8 V) is unusual but is unlikely to pose significant problems.*

Benchmark Performance

The TMS320C67xx has been benchmarked with the BDTI Benchmarks™. Overall benchmark results for all benchmarked processors are presented in Chapter 8, *BDTI Benchmark™ Results*. We summarize and analyze TMS320C67xx benchmark performance in the paragraphs below. We first discuss instruction cycle counts, which indicate the relative power of the processor's architecture. Note that instruction cycle counts do not take into account the processor's instruction cycle rate. Therefore, lower instruction cycle counts imply a more powerful architecture, but do not imply better performance. Next we discuss benchmark execution times and cost-execution time products, indicating processor performance and cost-performance, respectively. We then discuss the processor's energy consumption, which reflects the energy consumed by the processor in order to perform a task. Finally, we discuss the processor's memory usage. We divide the memory usage discussion into three parts: Control benchmark memory usage, overall benchmark program memory usage, and benchmark data memory usage.

Unlike the TMS320C6711 and TMS320C6712, the TMS320C6701 has 64 Kbytes of on-chip program RAM that can be configured to operate as a program memory or direct-mapped instruction cache (the TMS320C6711 and TMS320C6712 program mem-

Device	Frequency	Typical power consumption
TMS320C6701 (1.9V)	167 MHz	1.4 W
TMS320C6701 (1.8V)	150 MHz	1.3 W
TMS320C6711	150 MHz	1.1 W
TMS320C6711	100 MHz	0.8 W
TMS320C6712	100 MHz	0.7 W

TABLE 7.17-5. Typical TMS320C67xx power consumption summary.

ory is always configured as cache). The benchmark results presented here and in Chapter 8 assume that the on-chip program RAM is configured as program memory and that the benchmarks are pre-loaded into and executed from this memory. In addition, all data is assumed to be pre-loaded in the on-chip data memory.

> *TMS320C67xx performance is strongly dependent on the use of on-chip memory. As discussed above, results shown in this report assume that the program and data are preloaded in the on-chip memory. For example, they do not apply to the cache-based TMS320C67xx family members (i.e., the TMS320C6711 and the TMS320C6712) except in a "best-case" scenario where the instruction and data caches are preloaded. (Although the TMS320C6701 has an instruction cache, we classify the TMS320C6701 as non-cache based since, unlike the TMS320C6711 and TMS320C6712, it doesn't have a data cache and the level-1 instruction cache can be de-activated.) The impact that the two-level TMS320C6711 or TMS320C6712 cache architecture can have on performance varies for each benchmark and in some cases significantly increases cycle count numbers discussed in this report.*
>
> *The on-chip memory usage on the non-cache based TMS320C6701 can also significantly decrease the cycle count. When executing instructions exclusively from off-chip memory, even if the fastest type of off-chip memory is used (full-speed synchronous burst SRAM), the TMS320C6701's maximum instruction execution rate is slowed by a factor of eight. Thus, if there is a need to use external memory for program and/or data in a potential application for the TMS320C67xx, we urge readers to carefully consider how the processor's external memory interface and the on-chip instruction cache will perform in that application.*

Execution Performance

- **Instruction cycle counts:**

  The TMS320C67xx has a total normalized instruction cycle count that is second lowest of the three floating-point DSPs benchmarked: about 20% lower than that of the ADSP-2106x with its cache preloaded, and about 10% higher than that of the ADSP-2116x with its cache preloaded. The TMS320C67xx total normalized cycle count is about 15% lower than the average for all benchmarked DSP processors, as illustrated in Figure 8.1-13 in Chapter 8, *BDTI Benchmark™ Results.*

  TMS320C67xx benchmark cycle counts are low primarily due to its two data paths and eight execution units. These execution units include two floating-point multipliers and four floating-point ALUs, enabling the processor to achieve low cycle counts on benchmarks that perform block processing. However, the

TMS320C67xx cycle count are increased by the long latencies of floating-point multiply instructions and add instructions. These latencies necessitate the use of deep software pipelines that take a large number of cycles to set up and to flush. This overhead can outweigh the benefits of the multiple execution units on single-sample benchmarks.

On the **Real Block FIR** benchmark, the TMS320C67xx computes two multiply-accumulate operations per cycle but its throughput is hindered somewhat by the long instruction latencies described above. On this benchmark, the TMS320C67xx has the lowest cycle count of all benchmarked floating-point processors and a cycle count about 20% below the average of all benchmarked processors.

On the **Single-Sample FIR**, the TMS320C67xx cannot take advantage of its ability to perform two MACs in a single cycle, because it cannot load unaligned double words. Thus, the processor cannot always access two samples in the delay line per cycle, and requires additional load instructions in the convolution loop. In addition, the TMS320C67xx requires more cycles outside the inner loop in comparison to other DSPs due to the long instruction latencies described above; NOP instructions must be executed while waiting for the results of floating-point instructions. Thus, the TMS320C67xx has the third highest cycle count of all benchmarked DSPs on the Single-Sample FIR benchmark.

On the **Complex Block FIR** benchmark, the TMS320C67xx computes two multiply-accumulate operations per cycle but its throughput is hindered somewhat by its long instruction latencies (as with the Real Block FIR benchmark). The TMS320C67xx Complex Block FIR benchmark has a cycle count that is about 35% lower than that of the ADSP-2106x with its cache preloaded, and about 20% higher than that of the ADSP-2116x with its cache preloaded. The TMS320C67xx cycle count is about 15% below the average of all benchmarked processors.

Despite its extensive parallelism, the TMS320C67xx has the highest cycle count of the benchmarked processors on the **Two-Biquad IIR** benchmark. The higher cycle count of the TMS320C67xx on this benchmark is largely due to the processor's exceptionally deep pipeline and long instruction latencies (for example, four cycles for single-precision add and multiply operations). These latency cycles inflate the cycle count for this benchmark.

On the **LMS** benchmark, the TMS320C67xx has a cycle count that is about 5% below the average of all benchmarked processors. The TMS320C67xx combines the coefficient update and the filter convolution of the LMS benchmark in a single inner loop. Using its ability to perform two floating-point multiplications per cycle, the TMS320C67xx processes four filter taps in five cycles in the inner loop, compared to at least one tap in two cycles on most DSP processors. Combining the filter convolution and coefficient update in a single loop also reduces the required memory bandwidth and helps limit the impact of the double-word load alignment restriction discussed above for the Single-Sample FIR benchmark. Additionally, because it is a floating-point processor, the TMS320C67xx does not need to perform rounding when updating filter coefficients, simplifying the inner loop. However, due to the long latencies of some floating-point instructions, the TMS320C67xx uses a deep software pipeline on this benchmark. As in the Single-Sample FIR, NOP instructions must be executed outside the inner loop while waiting for the results of some instructions. This detracts somewhat from the benefits of the large number of execution units.

On the **Vector Dot Product** benchmark, the TMS320C67xx has a cycle count that is about 10% higher than the average of all benchmarked DSPs. Although it can compute two MACs per cycle, the TMS320C67xx takes 24 cycles—more than half of its total cycle count—to fill and flush the software pipeline on this benchmark.

On the **Vector Add** benchmark, the TMS320C67xx has a cycle count that is roughly 25% lower than the average of all benchmarked processors. This is due to the TMS320C67xx's ability to load four floating-point values from memory in a single cycle. The TMS320C67xx can load two elements from each of two vectors, perform two additions, and store two results in two cycles in the inner loop of this benchmark. In contrast, most DSPs perform only one addition every two cycles on this benchmark due to memory bandwidth limitations.

On the **Vector Maximum** benchmark, the TMS320C67xx has the lowest cycle count of the benchmarked floating-point DSPs, about 50% lower than the average cycle count of all benchmarked processors. The TMS320C67xx benefits from its ability to perform two floating-point comparisons per cycle and from flexible conditional execution features. Since floating-point comparison instructions do not have long latencies on the TMS320C67xx, the processor does not suffer from long instruction latencies as severely on this benchmark as on other benchmarks.

On the **FFT** benchmark, the TMS320C67xx has a cycle count about 5% lower than that of the ADSP-2106x with its cache preloaded, and about 35% higher than that of the ADSP-2116x with its cache preloaded. The TMS320C67xx cycle count for the FFT benchmark is about 45% below the average for all benchmarked processors. Although the ADSP-2106x has only one multiplier compared to two on the TMS320C67xx, the ADSP-2106x achieves a similar cycle count on the FFT benchmark due to its powerful instruction set, including an instruction specifically designed to accelerate FFTs. The SIMD-based, dual-multiplier ADSP-2116x also benefits from these specialized instructions, and achieves an even lower cycle count due to its SIMD parallel operations. Although the TMS320C67xx takes advantage of its high parallelism in this benchmark, the long latencies of its floating-point arithmetic instruction (four cycles for additions and multiplications using single precision) combined with the lack of floating-point SIMD addition and subtract instructions prevent it from obtaining a lower cycle count.

On the **Viterbi** benchmark, the TMS320C67xx has the lowest cycle count among the floating-point processors, and is among the lowest of all benchmarked processors, after the TMS320C62xx, TMS320C64xx, and the StarCore SC140. The TMS320C67xx achieves a cycle count that is about 65% lower than the average of all benchmarked processors. On this benchmark, the TMS320C67xx traceback loop requires very few cycles—only three clock cycles per bit. Conditional execution and the bit-extraction instruction are used to optimize the implementation of the benchmark. Also, as with the TMS320C62xx Viterbi benchmark implementation, a special addition instruction is used in the TMS320C67xx implementation. This instruction is referred to as the "addressing mode" addition. It is primarily intended to be used to increment a pointer by an offset that is automatically scaled (multiplied) by a factor of two or four depending on the data width referred to by the pointer. This instruction is useful in the traceback loop, and helps to reduce cycle counts on this benchmark.

On the Viterbi benchmark, the TMS320C64xx and the SC140 (both fixed-point processors) achieve cycle counts which are roughly 3 and 2.7 times lower than that of the TMS320C67xx, respectively. Two key reasons explain this difference. First, the SC140 and the TMS320C64xx feature dedicated instructions for the Viterbi algorithm (bit-interleaving and de-interleaving instructions on the TMS320C64xx, and special dual 16-bit SIMD maximum and conditional shifting instructions on the SC140). Second, the TMS320C64xx uses quad 8-bit SIMD addition, quad SIMD compare and quad SIMD maximum instructions in the add-compare-select loop of the Viterbi benchmark, which are not supported by the TMS320C62xx.

The results discussed above do not include cache miss penalties for the TMS320C67xx. To estimate the effect of L1 cache miss penalties, BDTI measured the cycle counts for several benchmarks on a TMS320C6711 development board. For these measurements, we flushed the level-1 instruction and data cache but preloaded the unified level-2 cache with data and instructions.

> *Focusing only on the level-1 cache impact is reasonable for many applications, since the level-2 memory can be configured to serve as cache or on-chip RAM. If used as on-chip RAM, data and instructions can be pre-fetched into level-2 memory with a minimum cycle penalty using the DMA controller.*

BDTI measured the effect of L1 cache misses on the Vector Dot Product, Real Block FIR filter, FFT, and Viterbi benchmarks. Note that except for the FFT benchmark, the level-1 instruction and data cache sizes of the TMS320C6711 are large enough (4 Kbytes each) to contain all of the benchmarks' instructions and data. Additionally, the benchmarks are optimized to avoid cache conflicts (i.e., instructions and data aren't loaded twice in the level-1 caches). Therefore, the level-1 cache impacts are mostly proportional to the data/program memory usage and the cycle count of each benchmark (and do not depend on the order of memory accesses in the benchmark).

On the Vector Dot Product benchmark, the combined L1 data- and instruction-cache misses more than double the cycle count of the TMS320C67xx. On the Real Block FIR, the L1 cache impact is lower, increasing the cycle count by about 20%. The effect of L1 cache misses on the FFT benchmark increases the cycle count by about 25%, which is partially due to the large amount of data required for the function. Finally, the L1 cache impact increases the cycle count by less than 5% for the Viterbi benchmark.

> *Our limited analysis of the effect of L1 cache misses indicates that the cache miss penalty can be significant (e.g., a 25% increase in cycles for the FFT, and a 100% increase for the Vector Dot Product). Therefore, we urge readers to carefully consider cache miss penalties when estimating the performance of the TMS320C6711.*

- **Execution times:** Its moderately low cycle counts and high instruction cycle rate (for a floating-point processor) of 167 MHz give the TMS320C6701 a total normalized execution time on the BDTI Benchmarks that is the fastest of the benchmarked floating-point processors by a significant margin. As illustrated in Figure 8.2-13, the total normalized execution time of the TMS320C6701 is about 2.2 times faster than the average for the floating-point DSPs, and about 1.8 times faster than its closest competitor, the Analog Devices ADSP-21160-C. At 167 MHz, the TMS320C6701 has a BDTImark2000 score of 820.
- **Cost-execution time:** The TMS320C6701's high price of $139 (in quantity 10,000) combined with its fast execution times give it a total normalized cost-execution time product that is worse than the ADSP-21065L-C but better than the ADSP-21160-C (both floating-point processors from Analog Devices). As illustrated in Figure 8.3-13, the TMS320C6701 has a total normalized cost-execution time product that is about twice as high as that of the ADSP-21065L-C but about 25% better than that of the ADSP-21160-C.

> *The TMS320C6701's speed and price data are used to generate the cost-execution time result for the TMS320C67xx family. It should be noted, however, that the cost-execution time for the TMS320C6711 would be significantly better, since the 100 MHz TMS320C6711 costs nearly six times less than the TMS320C6701 while maintaining roughly 60% of its speed.*

- **Energy consumption:** As illustrated in Figure 8.4-13B, the TMS320C6701's fast execution times combine with its moderate power consumption (for a floating-point DSP) to give it a total normalized energy consumption that is the lowest of the benchmarked floating-point DSPs by a wide margin. The TMS320C6701's total normalized energy consumption is about half as high as that of its nearest floating-point competitor, the Analog Devices ADSP-21065L-C.

Memory Usage

The focus in the memory usage analysis is on Control benchmark memory usage. Unlike other benchmarks, the Control benchmark is optimized for minimum memory usage. This benchmark is designed to indicate the processor's memory efficiency in control-oriented tasks, where memory usage is often more important than speed. We also discuss overall program memory usage in the BDTI Benchmarks™, reflecting the processor's program memory usage in general DSP code. Finally we discuss constant and non-constant data memory usage.

As mentioned earlier, TMS320C67xx instructions are 32 bits wide. Each 32-bit instruction is a simple, RISC-like instruction, and one execution packet may consist of up to eight 32-bit instructions. Thus, fewer operations per instruction are usually performed on the TMS320C67xx than on other floating-point DSP processors. This is the one reason why TMS320C67xx program memory usage tends to be higher (by a significant margin) than that of other benchmarked floating-point DSP processors. In addition, the fact that branches, loads, adds, and multiplies introduce five, four, three, and three delay slots, respectively, requires the use of software pipelining in most loop implementations. Filling and flushing deep software pipelines often requires many instructions, increasing code size.

- **Control benchmark memory usage:** On this benchmark, the TMS320C62xx and TMS320C67xx have the third-highest total memory usage of all benchmarked processors. These processors use the same implementation of this benchmark, since floating-point arithmetic is not useful here. As illustrated in Figure 8.5-9A, the total Control benchmark memory usage of the TMS320C67xx is lower than those of the other benchmarked floating-point DSPs (Analog Devices ADSP-2106x and ADSP-2116x, which use 48-bit instruction words), but higher than that of all other benchmarked DSPs.

  As discussed above, the high Control benchmark memory usage is primarily due to the 32-bit instruction width and the fact that TMS320C67xx instructions are fairly simple. Because the Control benchmark is optimized for memory usage rather than speed, the benchmark does not make use of deep software pipelining. However, without deep software pipelining, multi-cycle NOP instructions must sometimes be used to fill branch delay slots.

  On this benchmark, the conditional execution of instructions and a large number registers help the TMS320C67xx reduce program memory usage.

- **Program memory usage:**

  As illustrated in Figure 8.5-13, the total normalized program memory usage for the TMS320C67xx is the highest of all benchmarked floating-point DSPs, and is more than two times higher than that of the Analog Devices ADSP-2116x. The

TMS320C67xx has the highest program memory usage of all of the benchmarked floating-point DSPs in all BDTI Benchmarks except for the Control benchmark.

The main reasons for the TMS320C67xx's large program memory usage were discussed above. For example, a MAC operation takes 12 to 16 bytes (three to four 32-bit instructions) on the TMS320C67xx versus 6 bytes on the ADSP-2116x.

- **Data memory usage:** The constant and non-constant data memory usage of the TMS320C67xx are mostly as expected for a 32-bit floating-point processor. The TMS320C67xx has the lowest overall non-constant data memory usage of the benchmarked floating-point processors, primarily due to its low non-constant data memory usage on the Viterbi benchmark. On this benchmark, the TMS320C67xx uses the minimum amount of non-constant data memory necessary, while the other benchmarked floating-point DSPs use additional non-constant data memory for various speed optimizations. Total normalized constant data memory usage is shown in Figure 8.5-14 and total normalized non-constant data memory usage is shown in Figure 8.5-15.

On the majority of BDTI Benchmarks, the TMS320C67xx achieves the fastest execution times of the benchmarked floating-point processors. This provides the TMS320C6701 with a moderately good cost-execution time for a floating-point processor despite its high price. Additionally, the TMS320C6701 has the lowest energy consumption of the benchmarked floating-point DSPs by a wide margin.

As mentioned earlier, the TMS320C67xx provides several features to reduce memory use. These features include variable-size execution packets, conditional execution of all instructions, and instruction packing. Unfortunately, despite these features, the TMS320C67xx's total normalized program memory usage is the highest of all benchmarked processors. Compared to other benchmarked floating-point processors, the TMS320C67xx program memory usage is the highest in all benchmarks except the Control benchmark. Memory usage is often dominated by control-oriented code, however, making the Control benchmark memory usage result more important than memory usage on the other benchmarks.

Cost

Price and packaging options for TMS320C67xx processors are shown in Table 7.17-6.

Fabrication Details

The TMS320C6701 and TMS320C6711 are fabricated using a 0.18 µm five-metal-layer CMOS process. The TMS320C6712 is also fabricated using a 0.18 µm process, according to Texas Instruments.

Development Tools

The TMS320C67xx shares the same assembler, linker, archiver, instruction-set simulator, C compiler, assembly optimizer, and Code Composer Studio environment as used by the TMS320C62xx and TMS320C64xx. Please refer to Section 7.15 for a discussion of the TMS320C62xx/TMS320C67xx/TMS320C64xx development tools.

As with the TMS320C6211 simulator, the current TMS320C6711 simulator models external memory accesses. Like the TMS320C6211 simulator, the TMS320C6711 simulator is about 85% cycle-accurate when modeling external accesses to synchronous burst DRAM (SBSRAM) or asynchronous memory, according to Texas Instruments.

Note that the TMS320C62xx assembly optimized digital signal processing library and the TMS320C62xx image processing library, freely available from Texas Instruments, can be used with the TMS320C67xx. The floating-point features of TMS320C67xx are not currently used in these libraries, however.

An evaluation board for the TMS320C6701 is available from Texas Instruments and is designed to be used with a PCI expansion slot in a PC. This board is known as the TMS320C6701 EVM (for EValuation Module). Like the TMS320C6201 EVM, the TMS320C6701 EVM supports JTAG-based emulation that allows the user to debug on-chip programs from the CCS environment without the need for an XDS510 JTAG pod. Like the TMS320C6201 EVM, the TMS320C6701 EVM can be used as a stand-alone board (without the need for a host PC). The TMS320C6701 EVM provides the same amount and type of external memory (256 Kbytes of 133 MHz SBSRAM memory and 8 Mbytes of 100 MHz SDRAM). Like the TMS320C6201 EVM, an audio capture device

Device	Frequency (MHz)	Speed (Millions of MACs per second)	Voltage (V)	Package	Price (Qty. 10,000)
TMS320C6701	167	334	1.9 / 3.3[1]	352 BGA	$139.06
TMS320C6701	150	300	1.8 / 3.3[2]	352 BGA	$96.78
TMS320C6711	150	300	1.8 / 3.3[2]	256 BGA	$39.78
TMS320C6711	100	200	1.8 / 3.3[2]	256 BGA	$23.99
TMS320C6712	100	200	1.8/3.3[2]	256 BGA	$16.73

TABLE 7.17-6. TMS320C67xx price and package summary. Prices as of June 2000.

[1] The core operates at 1.9 volts while all I/O signals are 3.3-volt compatible.

[2] The core operates at 1.8 volts while all I/O signals are 3.3-volt compatible.

and external audio jacks are provided to connect a microphone and speakers to the board. As with the TMS320C6211 EVM, another less-sophisticated TMS320C6711-based evaluation board is also provided by Texas Instruments. This board is referred to as the Device Starter Kit (DSK) and is priced $295. The TMS320C6711 DSK board embeds 16 Mbytes of SDRAM and an audio codec.

Third-party support for the TMS320C67xx exists in the form of development boards, emulators, application boards, development tools and software libraries from a variety of vendors.

> *The number of third-party vendors providing tools and software for Texas Instruments' DSPs is the largest of any DSP processor vendor. This is due partly to Texas Instruments' early entry into the DSP processor market and to its strong efforts to cultivate third-party support.*

Applications Support

The TMS320C67xx shares most of its documentation with the TMS320C62xx and TMS320C64xx. This includes reference and programmer's guides and a generic peripherals guide for all of the TMS320C6xxx processors. Separate data sheets discuss the hardware aspects of specific devices.

Applications support for all TMS320 family processors is provided by Texas Instruments staff who are available via telephone, fax, and electronic mail. Documentation and brief application reports are also available via the World Wide Web.

Advantages

- Supports IEEE-754 single- and double-precision floating-point arithmetic
- Supports floating-point arithmetic and also provides instruction set compatibility with a fixed-point variant
- Highly orthogonal instruction set
- Good parallel move support (two 64-bit double-words or four 32-bit words can be loaded per instruction cycle)
- Conditional instruction execution
- Four ALUs, two multipliers, and two barrel shifters
- Good on-chip memory bandwidth
- Large number of registers for operands and addressing (thirty-two 32-bit general-purpose registers)
- Exponent detect instruction
- Large, unified address space
- Instruction cache for accelerating off-chip memory accesses

- Flexible external memory interface: ROM, asynchronous SRAM, synchronous burst SRAM, and synchronous DRAM support; DRAM refresh generation
- On-chip DMA controller with dedicated address and data buses
- Ability to use 1X or slower external clock
- JTAG emulation port with boundary scan
- Two serial ports, two timers
- Good BDTI Benchmark execution times for a floating-point processor
- Good BDTI Benchmark energy consumption for a floating-point processor

Disadvantages

- Multi-cycle instruction latencies complicate programming and increase code size
- Exposed pipeline complicates programming
- Exposed pipeline conflicts with ability to service interrupts
- No bit-reversed addressing
- Off-chip memory accesses take multiple cycles if not performed via DMA
- Requires execution from on-chip memory/cache for good performance
- Poor BDTI Benchmark program memory usage

8. BDTI Benchmark™ Results

This chapter presents the results of processor comparisons using the BDTI Benchmarks™. The BDTI Benchmarks are a set of DSP software functions that Berkeley Design Technology, Inc. has independently designed to provide an objective basis for comparing processor performance characteristics including speed, cost-performance, energy consumption, and memory usage for DSP applications. Benchmark functions have been programmed in assembly language by expert DSP programmers. The resulting code has been verified for functional correctness and adherence to the BDTI Benchmark specification, and its performance has been measured through detailed simulation and/or hardware measurements.

Note that the BDTI Benchmarks were significantly revised in 1999. This report presents benchmark results based on the **1999 version** of the BDTI Benchmarks.

> Note: Benchmark results obtained using the 1999 version of the BDTI Benchmarks should not, in general, be compared to results obtained using the previous version of the benchmarks, such as those presented in several of BDTI's prior reports: *Buyer's Guide to DSP Processors, 1999* and earlier editions; *DSP on General-Purpose Processors*; *Inside the Siemens TriCore*; and *Inside the Lucent DSP16000.*
>
> More specifically, results for the FFT, IIR Filter, and Control (which replaces the earlier FSM) benchmarks cannot be compared to previously released BDTI Benchmark results because the specifications for these benchmarks were changed in the 1999 version of the BDTI Benchmarks. The Viterbi and Bit Unpack benchmarks were added in the 1999 version of the BDTI Benchmarks, so results for these benchmarks do not appear in BDTI reports published prior to 2000. Other benchmarks may be comparable; contact BDTI for details.

As an introduction to the benchmarks, we first discuss the motivation for their development and some considerations that led to the current selection of benchmarks and processors. This is followed by an explanation of how the benchmark comparisons were performed and how results are organized and annotated. Finally, the results are presented in detail, with analysis that reveals why certain processors perform better or worse than others on individual benchmarks and overall.

The Need for Benchmarks

Execution speed is often the primary characteristic that designers use to compare DSP processors. This is a simple concept, but is difficult to measure fairly. Some manufacturers are fond of quoting the MIPS (millions of instructions per second) ratings of their processors. But different DSPs accomplish very different amounts of work in a single instruction, making MIPS-based comparisons misleading. Other possible measures are MOPS (millions of operations per second) or MFLOPS (millions of floating-point operations per second). But these pose a similar problem: there is no standard definition of an "operation." Some processor vendors say their chips can perform six operations per instruction (instruction fetch, multiply, add, two data reads, and a data write), resulting in inflated MOPS figures when compared to vendors who use a more conservative definition of "operation."

Another possibility is to standardize on a single, simple type of operation for comparison purposes. For DSP applications, the natural choice for this is the multiply-accumulate, or MAC, operation, which is at the heart of many DSP algorithms: filtering, correlation, dot products, etc. While virtually all DSPs have MAC instructions, measuring the execution time of this instruction misses a number of important aspects of processor performance. Important though it may be, the MAC instruction is not by itself representative of the computations that take place in applications. Even the most MAC-intensive applications inevitably make heavy use of many other types of instructions. Further, not all MAC instructions are equal. For example, some processors have MAC instructions that allow a variety of independent data moves and address pointer updates to be executed in parallel with the MAC operation, while others place severe restrictions on such parallel operations. In addition, limited memory bandwidth or other restrictions may make it difficult to execute MAC operations in an application at the maximum rate that is theoretically possible. Finally, while some processors have the ability to execute multiple MAC operations in parallel, some algorithms have data dependencies that make it difficult or impossible to use such capabilities.

A standard suite of application-oriented benchmarks, implemented fairly across all processors, avoids the limitations of simplified measures likes MIPS and MOPS. Below we discuss some approaches to DSP processor benchmarking.

Benchmarking Approaches

Ideally, DSP processor benchmarks would be entire DSP applications, or even suites of applications. Examples might include personal digital audio players, xDSL modems, disk drive servo controllers, or voice-over-IP phones. Teams of engineers would implement the applications using assigned DSPs and would report back with completed designs. The designs could then be evaluated on the basis of overall design time, execution time, memory usage, and a host of other factors. This type of application benchmark has long been used for benchmarking the performance of computer systems for business and scientific applications.

This approach works best in cases where there is application software portability; i.e., when applications are implemented in a high-level language like C. Unfortunately, because of the limited efficiency of C compilers for the most cost-effective DSP processors and the demanding performance requirements of DSP applications, developers typically write significant amounts of assembly code to maximize performance of key code sections on a particular architecture. Therefore, benchmarks written in high-level languages have limited relevance when judging processor performance for DSP applications. Benchmark applications written in high-level languages measure both the compiler and the processor, making it difficult to draw conclusions about the performance of the processor alone.

Even if application benchmarks are coded in assembly language, one encounters four practical problems with DSP application benchmarks: First, most applications are not sufficiently well defined to permit fair comparisons. For example, two implementations of a standard-compliant modem can use different equalizers, one much simpler than the other, depending on whether the objective is a high-quality implementation or one that makes minimum demands on the processor. Second, when implementing complex applications, it is virtually impossible to ensure that software is optimal, or even near optimal. This means that a benchmark based on a complete application is likely to be measuring the programmer, as well as the processor. Third, complete-application benchmarks tend to measure a system's performance, not just a processor's. Isolating the performance of the processor from that of other system components like external memory and host processors can be very difficult. Last, optimizing an entire application can take years of engineering time.

Because of the practical difficulties of complete-application benchmarks for DSP, we have chosen an alternative approach: the use of *application kernels* to represent larger applications. These are small fragments of DSP software that form the key pieces of larger applications. Example kernels include FIR and IIR filters, vector processing functions, fast Fourier transforms (FFTs), error correction coders, and decision-making control code. Application kernels have several compelling advantages for use as benchmarks:

- **Relevance**. Application kernels can be selected by examining common DSP applications and focusing on those portions of the applications that account for the largest share of the processing time. This guarantees their relevance.
- **Ease of specification**. By virtue of their modest size, application kernels can be well defined: a specification can state their input and output requirements and indicate which algorithm variants and optimizations are allowable. Test vectors can be provided to ensure conformance to the specification.

  Note, however, that while application kernel benchmarks *can* be well defined, they often are not. When processor manufacturers publish kernel benchmark results, for example, they often don't clearly specify the functionality that is implemented or the optimizations used. This can make processor comparisons based on manufacturer-supplied application kernel benchmark data hazardous.
- **Optimization**. Because application kernels are of a moderate size, a skilled programmer can be fairly certain that their implementation is optimal, or very close to optimal, for a given processor.

- **Ease of implementation**. Due to their moderate size, application kernels can be implemented in assembly language in a reasonable amount of time, even with thorough optimization.

The trickiest part of the application kernel approach lies in specifying the benchmarks. The goal is to allow programmers enough freedom to attack each benchmark in a natural way on each processor, while also constraining them to stay within the bounds of reasonable implementation practices.

For example, an FFT benchmark can be implemented using a number of algorithms: radix-2, radix-4, in-place, not in-place, decimation-in-time, decimation-in-frequency, and permuted algorithms, to name just a few. Not all of these are equivalent in terms of functionality or in terms of the theoretical number of arithmetic operations required. Even if the algorithm is precisely specified, there are a large number of optimizations (e.g., eliminating multiplications in the outer loops, loop unrolling) that can be used to speed up the implementation. It is critical that the benchmark specification dictate which of these optimizations are permissible. The challenge is to do this without overly constraining the implementor.

Our experience has shown that even with good benchmark specifications, questions still arise when benchmarks are implemented by different programmers. Thus, to help ensure consistency, BDTI serves as an impartial arbiter for BDTI Benchmark programmers, making fair and consistent decisions regarding the admissibility of different implementation approaches. Without such an arbiter, fair comparisons of processor performance are impossible.

Note that benchmark results published by processor manufacturers are sometimes very aggressively optimized, using optimizations that might not be permitted under the BDTI Benchmarks. In addition, the functionality implemented by manufacturers' benchmarks is often inconsistent or ill-defined (for example, some FFT implementations include shuffling the input or output data while others do not). These problems motivated us to develop the BDTI Benchmarks. For these same reasons, manufacturer-reported benchmark results for benchmark functions similar to those found in the BDTI Benchmarks may suggest significantly better performance than that shown by the BDTI Benchmarks.

This highlights the fact that the BDTI Benchmarks are not designed to reveal the speed of the fastest-possible implementation of a given function on a particular processor. Rather, our benchmarks are designed to reveal the performance that is attainable under a reasonable and consistent set of assumptions and restrictions, and thus to provide a fair basis for comparison among processors.

Benchmarking Limitations

Application kernel benchmarks implemented in a fair manner can be an extremely useful tool for comparing processor performance in DSP applications. In developing and implementing the BDTI Benchmarks and the analyses of our results, we have striven to provide DSP processor users with an objective basis for comparing processors and for selecting the processor which will best suit the needs of an application. But as with any tool, the user of these benchmark results must respect their inherent limitations. Like all benchmarks, ours suffer from important limitations that should be thoroughly understood by anyone using them. The following paragraphs describe these limitations.

- **Completeness.** One of the key shortcomings of benchmarking is that it is very difficult to capture all of the important aspects of processor performance in a set of benchmarks. For example, our benchmarks do not attempt to measure the input/output performance of processors, even though I/O performance is a significant consideration for many, if not most, applications. Other potentially important operations, such as context switching, are also omitted. We plan to expand the BDTI Benchmarks in future releases to address some of these aspects.

  No matter how comprehensive it is, a set of benchmark functions usually doesn't include all of the functions that are important for a particular application. This means that in using benchmarks to compare processors, users must be prepared to work with incomplete information.

- **Numerical accuracy**. Processors with larger native data word sizes generally produce more accurate results than processors with smaller word sizes. Floating-point processors generally produce more accurate results than fixed-point processors. In this sense, a processor with a larger numeric format or with a floating-point format can be considered to do more work in the course of computing a given function than a processor with a smaller or fixed-point format, if both processors use their native data types. These differences are mostly ignored in our benchmarks. In the BDTI Benchmarks, each processor uses its native numeric format(s) for all benchmarks. In using the benchmark results, the DSP processor user must be aware of the precision and dynamic range requirements of their application, and interpret the benchmark results accordingly. For example, if a user is planning to implement an FIR filter with 24 bits of precision, this can be done on a 16-bit processor, but it will require the use of multi-precision arithmetic, and this means that the FIR filter implementation will be significantly slower than suggested by the benchmark results.

  In the BDTI Benchmarks, each processor uses its native data word size. In cases where processors support multiple data word sizes, BDTI Benchmark implementations generally use the data word size that yields best performance while maintaining a reasonable level of numeric fidelity for the benchmark function. An exception to this is floating-point processors, which use their floating-point data type for most benchmarks, even in cases where use of a fixed-point data type would be faster. A few of the benchmarks, however, are inherently integer in nature, and do not use floating-point data types in any case. This includes the Control and Bit Unpack benchmarks, for example.

- **System design**. Many aspects of processor performance depend very strongly on overall *system* design, not just on processor design. For example, if an application becomes too large for the entire program and data to be stored in on-chip memory, the capabilities of the off-chip memory and memory interfaces significantly impact performance and cost. Our benchmarks mostly ignore issues of system design, and focus exclusively on the processor. Earlier chapters of this report (especially Chapter 7, *Processor Analyses*) highlight various issues impacting overall system performance. We urge readers to use our benchmark results in conjunction with our detailed processor qualitative evaluations found in Chapter 7.
- **Weighting of individual benchmarks**. In the analysis that follows, we first present performance results for individual benchmarks. We then compute aggregate performance results for each processor by combining the results for each benchmark function. When combining the results for each benchmark function, we apply a *uniform* weighting to the results from each individual benchmark function. The uniform weighting ensures that the overall score reflects the contributions of each benchmark equally. This approach gives a balanced picture, but the uniform weighting is almost certainly not appropriate for any specific application. For a particular application, some of the BDTI Benchmark functions will be more important and some will be less important. *The correct weighting for a particular application must be determined by the user.* The profiling data and analysis provided in Chapter 6, *Choosing a Processor* should aid in determining appropriate weightings for many kinds of applications.
- **Changing speeds and prices**. To enable fair comparisons, the analysis presented here is based on processor speeds and prices available as of June 2000. But processor speeds and prices change frequently. Readers should consult processor vendors for the latest speed and price data before making a processor selection.

BDTI offers the Benchmark Analysis Tool™ for users who wish to customize the analysis presented here. The Benchmark Analysis Tool allows the user to specify weightings to be applied to the individual benchmark functions for purposes of computing aggregate results, to update speed and pricing data, and to add data for new or proprietary processors. For more information on the Benchmark Analysis Tool, please contact BDTI.

Processors Benchmarked

Of the processor families reviewed in detail in the previous chapters, 14 have been analyzed using the BDTI Benchmarks. The benchmarked processors are listed in Table 8.0-1. A few processors included in previous chapters are not included here, for various reasons. In some cases, such as the Texas Instruments TMS320C2xxx, we elected not to invest the effort in implementing the new version of the BDTI Benchmarks because earlier benchmarking had shown the processor to be in a different (that is, lower) performance class than most of the other processors benchmarked here. In other cases, we were unable to obtain final BDTI Benchmark results in time for this publication.

Most of the processors listed in Table 8.0-1 were in production, or available in sample quantities at the time of this writing. A few were not yet available in sample quantities, or had not been demonstrated in silicon at their target speeds; results for these pro-

Vendor	Model/Family	Intro-duced	Data/Arithmetic		Comments
			Width	Type	
Analog Devices	ADSP-218x	1986	16	Fixed	24-bit instructions
	ADSP-219x	1999	16	Fixed	
	ADSP-2106x	1994	32	Floating	48-bit instructions
	ADSP-2116x	1998	32	Floating	
Lucent Technologies	DSP164xx	1997	16	Fixed	16- and 32-bit instructions
Motorola	DSP563xx	1995	24	Fixed	24-bit instructions
	DSP568xx	1990	16	Fixed	16-bit instructions
	DSP5685x	2000	16	Fixed	16-bit instructions
	MSC8101	1999	16	Fixed	Based on the SC140
StarCore	SC140	1999	16	Fixed	Lucent/Motorola joint design
Texas Instruments	TMS320C54xx	1994	16	Fixed	16-bit instructions
	TMS320C55xx	2000	16	Fixed	VLIW-based architecture
	TMS320C62xx	1997	16	Fixed	VLIW-based architecture
	TMS320C64xx	2000	16	Fixed	VLIW-based architecture
	TMS320C67xx	1998	32	Floating	VLIW-based architecture

TABLE 8.0-1. DSP processors benchmarked in this report.

cessors are marked "projected" throughout this chapter. These processors include the Analog Devices ADSP-219x, Motorola DSP5685x, and Texas Instruments TMS320C55xx and TMS320C64xx.

Within a single processor family, multiple devices often exist with different memory configurations, peripherals, and low-power features. Moreover, these variants may be offered with different maximum clock frequencies. Benchmark results may vary between these family members. To obtain fair comparisons in our benchmark analysis, in each section of our analysis we selected the processor variant from each family that best matched the nature of the analysis category. For our analysis of execution time and cost/execution-time product, we chose the fastest version of each processor, in its least expensive packaging variant. For our analysis of energy consumption, we chose the lowest-voltage version of each processor, in its fastest speed and lowest-cost package. We restricted ourselves to processor variants that contain sufficient (but not excessive) on-chip memory to run the benchmarks efficiently. The criteria for selecting family members are stated in each section of our benchmark analysis.

Note that for many processor families, selecting family members other than those used here yields significantly different benchmark results. For example, our cost/execution time product analysis of the Texas Instruments TMS320C62xx family uses the TMS320C6203, the fastest family member. If we instead chose a low-cost family member, such as the TMS320C6211, cost-performance results would be improved significantly, while speed would be reduced, and less on-chip memory would be available. Users may wish to adjust the results presented in this chapter, using data provided in Chapter 7, *Processor Analyses*, based on the family members determined to be most suitable for a particular application.

Organization of Benchmarking Results

The benchmark results are organized as follows:

- **Instruction Cycles**
  In Section 8.1 we compare the number of instruction cycles that each processor requires to complete each of the twelve benchmark functions.
- **Execution Time**
  In Section 8.2, we use the instruction cycle rate of the fastest available version of each processor to calculate the actual time required for each processor to compute each benchmark function. This section also includes the BDTImark2000 scores of the processors benchmarked.
- **Cost-Execution Time Product**
  In Section 8.3, we use processor cost data to calculate the *cost-execution time*

product of each processor. The cost-execution time product is a combined figure of merit that reflects both a processor's cost and its speed.

- **Energy Consumption**
  In Section 8.4, we use processor power consumption data to estimate the amount of energy each processor requires to compute each benchmark function. This gives an estimate of the energy efficiency of each processor.
- **Memory Use**
  In Section 8.5, we evaluate processors' memory use efficiency on the BDTI Benchmarks. We report and analyze Control benchmark memory use, overall program memory use, overall constant data memory use, and overall non-constant data memory use.

Note that in all of these analyses except for instruction cycle counts, lower numbers imply better performance. As we discuss below in Section 8.1, *Instruction Cycle Counts*, lower instruction cycle counts are not necessarily an indicator of good performance.

Each section begins with an introduction that explains the meaning and interpretation of that section's analysis. Next, the detailed results for each processor are presented in tabular form. Then the processor families are compared directly in bar graph form for each benchmark. Finally, the overall results obtained from combining results for each processor over all benchmarks are displayed in bar graph form.

The results given in tabular form are intended for those readers who wish to extend or modify the analyses for their own purposes; Section 8.1 discusses this further. The casual reader may choose to turn directly to the graphical representations to gain the essence of the results. But please review each section's introduction to obtain a clear understanding of how the results were obtained and what they mean.

Benchmark Specification and Methodology

This section highlights important aspects of our benchmark specification, implementation, and measurement. It focuses on general issues applicable to the set of benchmarks as a whole. We begin by discussing benchmark specification, touching upon issues of code structure, optimization, and memory use.

Table 8.0-2 lists the benchmarks. Although we refer to the individual benchmarks as *functions*, in fact our benchmarks are not programmed using the formal programming notion of a function. Rather, our model for the organization of the benchmark programs is based on a program *macro,* a parameterized block of code that is expanded in-line in the final program. If multiple references to a macro are made in a program, then the macro code is repeated for each reference. A macro clarifies the calling parameters and return values (the interface) of a reusable block of code, but does not incur the overhead of a subroutine call. As a rule, BDTI Benchmarks™ are not implemented as subroutines, nor do they use subroutines internally. The Control benchmark is an exception to this rule.

Function	Description	Example Applications
Real Block FIR	Finite impulse response filter that operates on a block of real (not complex) data.	Speech processing (e.g., G.728 speech compression).
Single-Sample FIR	FIR filter that operates on a single sample of real data.	Speech processing, general filtering.
Complex Block FIR	FIR filter that operates on a block of complex data.	Modem channel equalization.
LMS Adaptive FIR	Least-mean-square adaptive filter; operates on a single sample of real data.	Channel equalization, servo control, linear predictive coding.
Two-Biquad IIR*	Infinite impulse response filter that operates on a single sample of real data.	Audio processing, general filtering.
Vector Dot Product	Sum of the pointwise multiplication of two vectors.	Convolution, correlation, matrix multiplication, multidimensional signal processing.
Vector Add	Pointwise addition of two vectors, producing a third vector.	Graphics, combining audio signals or images, vector search.
Vector Maximum	Find the value and location of the maximum value in a vector.	Error control coding, algorithms using block floating-point.
Viterbi Decoder	Decodes a convolutionally encoded bitstream.	Wired and wireless communications; e.g., digital cellular phones.
Control*	A contrived series of control (test, branch, push, pop) and bit manipulation operations.	Virtually all DSP applications include some control operations.
256-Point FFT*	Fast Fourier transform converts a normal time-domain signal to the frequency domain.	Radar, sonar, MPEG audio compression, spectral analysis.
Bit Unpack	Unpacks words of varying lengths from a continuous bit stream.	Audio and speech decompression.

TABLE 8.0-2. BDTI Benchmark™ functions.

* These functions are significantly different from the similarly named functions found in the earlier version of the BDTI Benchmarks.

Structurally, each benchmark program is divided into two sections: power-up initialization and the main benchmark body. The power-up initialization includes processor configuration actions that are required only once at power-up. The main benchmark body contains all of the functionality required to execute the benchmark, except for the one-time initialization steps included in the power-up initialization section. The main benchmark body does include initialization and clean-up steps that must be performed for each invocation of the benchmark; for example, loading memory pointer registers and configuring special addressing modes. Since the kind of code found in the power-up initialization section of the benchmarks is usually not relevant to application performance, we exclude power-up initialization code when reporting benchmark results.

A detailed specification of each benchmark function has been created to define the precise functionality required and the allowable implementation approaches for each benchmark. This specification is available under license from BDTI.

In all but one benchmark (the Control benchmark), the primary goal of the programmer is to minimize execution time, and the secondary goal is to minimize memory use. In the Control benchmark, however, conservation of memory is the primary goal, and speed of execution is secondary. This is because the Control benchmark is designed to be representative of decision-making control code, which often makes up the bulk of an application's code space but only takes up a small fraction of the execution time. Thus, speed is generally less important than code density in control-oriented code.

All benchmark programs are written entirely in assembly language and carefully optimized. Some of the benchmark programs were implemented by BDTI, while others were implemented by processor manufacturers. In all cases, the functionality and performance of the benchmark programs were independently verified by BDTI. In addition, we have carefully reviewed each benchmark implementation on each processor to ensure that it follows the coding guidelines set out in the specification and to help ensure optimality.

We have measured the performance of each benchmark on each processor using simulators developed by the processor manufacturers. On the whole, we have found these simulators to be accurate in their reports of program cycle counts. Nevertheless, we have encountered some erroneous reports. Therefore, for the most critical sections of the benchmark programs (primarily the kernels) we have manually checked the results reported by the simulators. Where practical, we have also confirmed out results via measurements on processor hardware.

Benchmark Programming Restrictions and Optimizations

Here, we briefly discuss some of the more important restrictions placed on benchmark programmers by the BDTI Benchmark™ specifications. The complete BDTI Benchmark specification is available under license from BDTI.

The specification includes coding guidelines that outline the kinds of algorithmic and programming optimizations that are permissible for each benchmark function. In assembling the coding guidelines for the BDTI Benchmarks, we have attempted to strike a balance between the conflicting goals of optimization and creating a level playing field, which sometimes precludes improvements in the code which might very well be reasonable in some applications. Our guidelines generally permit the use of optimizations that we consider reasonable (i.e., that an application programmer would be likely to use in production software), but prohibit more extreme optimizations that are not likely to be practical in most applications.

- **Sharing of processor resources.**
  We require each benchmark function to be written as if it were one of many functions in an application. Each benchmark must be written so that it can be executed multiple times to process a continuous stream of data, and so that its execution can be interleaved with the execution of other code. This means that between invocations each benchmark function must store all of its private state information in memory and not in processor registers that may be used by other code. For example, the IIR filter benchmark must store its state variables in memory between invocations.
- **Multiprocessors.**
  For chips containing multiple processors, the BDTI Benchmarks are implemented using only one of the processors. Different types of applications and algorithms differ in how efficiently they can be parallelized using multiple processors. Therefore, in general it is not correct to assume that an *N*-processor chip will have *N* times the performance of one of the processors. The only multiprocessor device in this report is the Lucent DSP164xx. The DSP164xx is used here instead of the single-processor DSP162xx because the multiprocessor device incorporates instruction-set enhancements and achieves a higher clock speed.

Note that the results shown in the report for the Lucent DSP164xx use only one of the chip's two processors.

- **Caches**
  On-chip caches are increasingly common among DSP processors. As detailed in Chapter 7, *Processor Analyses*, the caches found in DSP processors differ significantly from one processor to another. The Analog Devices ADSP-219x, ADSP-2106x, and ADSP-2116x, for example, use very specialized instruction caches. These processors provide two on-chip memory buses. Their caches are used to hold instructions only in the case where the instruction fetch conflicts with one of two data accesses performed by the previous instruction.

  The Lucent DSP164xx uses a similar approach: its specialized instruction cache is used to free up one of its two on-chip memory buses for performing a second data access. In contrast with the Analog Devices processors, though, the Lucent processor uses its cache only when a hardware loop is executed, and stores all of the instructions within the hardware loop body in the cache.

In contrast, the TMS320C64xx and some members of the TMS320C62xx and TMS320C67xx families from Texas Instruments rely on a two-level instruction and data cache architecture similar to those found in high-performance general-purpose processors. Unlike caches typically found in high-performance general-purpose processors, though, the second-level caches found in these Texas Instruments processors can be configured by the programmer to act as conventional RAM.

The state of a processor's caches upon entering the benchmark can have a significant impact on performance for benchmarks that spend a significant amount of time executing code that is not contained in small loops. Thus, caches must be taken into account when implementing benchmarks and when interpreting benchmark results. If all or part of the benchmark code has been previously loaded into the cache, some benchmarks execute much more quickly than if none of the needed code is contained in the cache. In the BDTI Benchmarks, this difference affects single-sample benchmarks (Single-Sample FIR, IIR, and LMS filters) more than block-oriented benchmarks.

In implementing the BDTI Benchmarks, we have addressed caches in various ways, depending upon the details of each processor's cache design:

- If all members of a processor family rely on caches, we present two sets of results: the first set (which appears under the heading of the processor name) is measured with the first-level cache empty when the benchmark begins execution; the second set (which appears under the processor name with a "-C" appended; e.g., "ADSP-2106x-C") is measured with the first-level cache preloaded with the needed instructions and data prior to benchmark execution. In both cases, the second-level cache or on-chip RAM is assumed to be preloaded with the necessary instructions and data.

  Processors in this category include the Analog Devices ADSP-219x, ADSP-2106x, and ADSP-2116x, and the Texas Instruments TMS320C64xx. The Lucent DSP164xx is excluded from this group because the small size of its instruction cache means that the cache-preloaded scenario is unlikely to be encountered in typical applications.

- If some members of a processor family rely on caches and some members do not, we have chosen to ignore caches in our benchmark results. This means that the benchmark results do not apply to family members with caches. This applies to cache-based members of the Texas Instruments TMS320C62xx and TMS320C67xx families.

For processors for which two sets of results are reported, if there was a trade-off to be made in optimizing for the cache-preloaded versus the non-cache-preloaded scenario, the cache-preloaded scenario was given priority.

For processors for which two sets of results are reported, users must determine whether the cache-preloaded or non-cache-preloaded results, or some mixture of the two, are more relevant to the target application.

> Readers should compare cache-preloaded ("-C") results for processors with caches to other cache-preloaded results, and to results for processors without caches. Comparing non-cache-preloaded results to cache-preloaded results or to results for non-cached processors may not yield meaningful comparisons. Determining whether cache-preloaded or non-cache-preloaded results are more relevant requires a detailed understanding of the memory access behavior of the target application.

As explained in Chapter 7, *Processor Analyses*, non-cache-preloaded results for the Texas Instruments TMS320C64xx are estimated, because, at the time of this writing, Texas Instruments had not released final specifications for the TMS320C64xx caches.

- **Memory.**
  In general, each benchmark has been programmed so that the entire benchmark program and all associated data are located in on-chip memory to allow the processor to execute the benchmark as fast as possible. While this is not necessarily a realistic scenario for all applications, it allows benchmark performance to be measured independently of complex considerations such as external memory structure, interfaces, and cost. In most cases, processors will achieve the performance reported here only for relatively small data and program sizes. The threshold beyond which external memory becomes a bottleneck varies from processor to processor and system to system, based on the size and organization of on-chip memory and the capabilities of the off-chip memory system and interface(s). Chapter 7, *Processor Analyses*, presents detailed analyses of the on-chip memory architectures and off-chip memory interfaces of each of the processors covered in this report.
- **Loop unrolling.**
  An example of an optimization that we consider extreme and have therefore prohibited in implementation of BDTI Benchmarks™ is excessive loop unrolling. Loop unrolling is a technique for increasing execution speed by replacing program looping structures (like for-next loops) with repetitive sets of instructions accomplishing the same effect. Loop unrolling can improve performance, for example, on processors where loop constructs consume processor cycles, and on processors with limited address register update capabilities. In some cases, loop unrolling can also allow algorithmic transformations that enable improved performance. However, excessive loop unrolling can result in unreasonably large programs, and

reduces the generality (and therefore the reusability) of code. In some benchmarks previously published by other organizations, some programmers used loop unrolling, while others did not, resulting in unrealistic results and unbalanced comparisons. In implementing the BDTI Benchmarks™, we have limited loop unrolling. In deciding how much loop unrolling to allow in a specific case, we compared the increase in code size to the resulting performance gain for different amounts of unrolling. We chose an amount of unrolling that does not cross the point of diminishing returns in this respect, and that also does not result in unreasonable loss of generality.

- **Look-up tables.**
  Like loop unrolling, the use of look-up tables is an optimization technique that can often be used to improve execution speed at the cost of additional memory use. In our opinion, the use of look-up tables tends to obscure meaningful differences in processor performance. Therefore, in most cases we have prohibited the use of such tables in implementations of the BDTI Benchmarks. An exception is the use of tables to store "twiddle factors" for the FFT benchmark.

The detailed specification for the BDTI Benchmarks outlines many additional restrictions placed on implementation of these benchmarks in an effort to ensure that the benchmark results are fair and accurate.

Benchmark Parameters and Scaling of Results

Many common DSP functions operate on varying quantities of data (and coefficients), depending on the needs of the application. For example, an FIR filter can have any number of taps and can operate on any number of input samples. For some (but not all) of the BDTI Benchmarks, the specification requires that benchmark programmers implement the benchmarks so that they can accommodate a reasonable range of data and/or coefficient set sizes. Other benchmark functions have fixed data set sizes.

The IIR, Control, 256-point FFT, Viterbi, and Bit Unpack benchmarks have fixed data set sizes. The Real Block FIR, Single-Sample FIR, Complex Block FIR, LMS Adaptive FIR, Vector Dot Product, Vector Add, and Vector Maximum benchmarks have variable data-set and/or coefficient-set sizes. In addition, some implementations of the Vector Maximum and Bit Unpack benchmarks have data-dependent execution times.

For the benchmarks with variable data-set and/or coefficient-set sizes, or data-dependent execution times, the following parameters are used to characterize the data and coefficients:

- **Number of points, N**
  The number of input or output data points (real or complex) processed by a benchmark function.
- **Number of filter taps, T**
  The number of filter taps for FIR-type filter benchmarks.

- **Number of new relative maxima, M**

  For the Vector Maximum benchmark, the number of new relative maxima encountered when scanning the input data from beginning to end.

For the benchmarks with variable data-set and/or coefficient-set sizes, the instruction cycle counts and data memory use results presented in this chapter are first presented as algebraic expressions in terms of the size of the parameters listed above. Next, we substitute a set of reasonable but otherwise arbitrary values for these parameters and evaluate the expressions. The resulting values are used in our subsequent analysis. The parameter values chosen for our analysis are listed in Table 8.0-3. The parameter values we have chosen are not appropriate for all applications. Therefore, readers may wish to repeat our analysis, substituting values that are better suited to their applications.

Note, however, that benchmark results cannot be arbitrarily scaled using the expressions provided. A number of factors complicate the scaling of benchmark results. First, benchmark implementors are permitted (and encouraged) to optimize their implementations for the particular parameter values listed in Table 8.0-3 and used in this analysis. Thus, for different data set sizes, a different benchmark implementation may yield better performance. (This also explains why benchmark results for benchmark functions with fixed data-set sizes, such as the 256-point FFT, are not scalable, and thus, why results for these benchmarks are reported as constant values rather than as expressions.) In addition, common optimizations often yield a loss of generality; for example an FIR filter implementation on a dual-MAC processor will typically only accommodate FIR filters with even numbers of coefficients. Finally, while scaling of benchmark results can be effective within a range of parameter values, there are always limits beyond which the extrapolation breaks down; for example, when the size of internal memory is exceeded, and data must be moved to and from slower, off-chip memory.

Note that cycle counts for the Texas Instruments TMS320C64xx with caches not preloaded are expressed as constants for all benchmarks, rather than as formulas. This is because the timing effects of the TMS320C64xx caches make it difficult to characterize benchmark cycle counts as formulas. These constants are valid only for the particular benchmark parameters used in this analysis; that is, they are not scalable. For the Texas Instruments TMS320C55xx, a cycle-accurate instruction-set simulator was not available, so the benchmarks were timed exclusively via hardware measurement, and cycle count formulas could not readily be derived from these measurements.

BDTI offers the Benchmark Analysis Tool™ for users who wish to customize the analysis presented here. The Benchmark Analysis Tool allows the user to repeat the analysis presented here after substituting other values for the parameters. The Benchmark Analysis Tool also allows the user to specify weightings to be applied to the individual benchmark functions for purposes of computing aggregate results, and to add data for new or proprietary processors. For more information on the Benchmark Analysis Tool, contact BDTI.

Table 8.0-3. Benchmark Parameters

DSP Building Block Functions	Fixed Parameters	Variable Parameters
Real Block FIR		N=40, T=16
Single-Sample FIR	N=1	T=16
Complex Block FIR		N=40, T=16
LMS Adaptive FIR	N=1	T=16
Two-Biquad IIR	N=1, B=2	
Vector Dot Product		N=40
Vector Add		N=40
Vector Maximum		N=40, M=5
Control		
256-Point FFT	N=256	
Viterbi		
Bit Unpack		

Key: N = input vector length, B = number of biquad sections,
T = number of taps, M = number of new relative maxima.

Notation

Table 8.0-4 and Table 8.0-5 define the notation used throughout this chapter. Every table and figure presented in this chapter uses this notation.

Symbol	Definition	Meaning
MP[p,b]	Program memory use	Total program memory use for processor p on benchmark b (in bytes). The program memory use reported here is for the main benchmark body, and does not include the power-up initialization section.
MC[p,b]	Constant data memory use	Constant data memory use for processor p on benchmark b (in bytes).
MD[p,b]	Non-constant data memory use	Non-constant data memory use for processor p on benchmark b (in bytes).
MPC[p,b]	ROMable memory use, MP[p,b] + MC[p,b]	ROMable memory use for processor p on benchmark b (in bytes). ROMable memory includes program memory and constant data memory.
MPCD[p,b]	Total memory use, MPC[p,d] + MD[p,b]	Total memory use for processor p on benchmark b (in bytes). Total memory use includes program memory use, constant data memory use, and non-constant data memory use.
AMP[b]	Average program memory use	Average program memory use on benchmark b. (Average across all processors, in bytes.)
AMC[b]	Average constant data memory use	Average constant data memory use on benchmark b. (Average across all processors, in bytes.)
AMD[b]	Average non-constant data memory use	Average non-constant data memory use on benchmark b. (Average across all processors, in bytes.)
AMPC[b]	Average ROMable memory use	Average ROMable memory use on benchmark b. (Average across all processors, in bytes.)
AMPCD[b]	Average total memory use	Average total memory use on benchmark b. (Average across all processors, in bytes.)

TABLE 8.0-4. Notation used in presentation of benchmark results (1 of 2).

Symbol	Definition	Meaning
p	Processor	Index of processors in tables. In each table or figure, p=1 for the first processor listed, p=2 for the second processor listed, and so on.
b	Benchmark	Index of benchmark functions in tables. In each table or figure, b=1 for the first benchmark listed, b=2 for the second, and so on.
C[p,b]	Instruction Cycle Count	Instruction cycles required for processor p to complete benchmark b. Includes cycles for the main benchmark body, but *not* power-up cycles (see text).
I[p]	Instruction Cycle Time	Instruction cycle time for processor p (for single-cycle instructions), in nanoseconds.
T[p,b]	Execution Time, C[p,b] * I[p] / 1000	Execution time for processor p to complete benchmark b, in microseconds. (The factor of 1000 converts from nanoseconds to microseconds.)
P[p]	Power	Estimated "typical" power consumption for processor p, in watts.
E[p,b]	Energy, P[p] * T[p,b]	Energy consumption for processor p on benchmark b. The result of multiplying execution time by typical power consumption, in watt-microseconds.
D[p]	Cost	Quoted unit cost for processor p based on a purchase of 10,000 units, in U.S. dollars.
AC[b]	Average Cycles	Average number of instruction cycles required for processors to complete benchmark b. (Average across all processors.)
AT[b]	Average Time	Average time required for processors to complete benchmark b, in microseconds. (Average across all processors.)
DT[p,b]	Cost-Execution Time Product D[p] * T[p,b]	The result of multiplying the cost of processor p by the execution time required for the processor to complete benchmark b. This is a figure of merit reflecting both execution time and cost in microsecond-dollars.
ADT[b]	Average Cost-Execution Time Product	Average cost-execution time product for benchmark b, across all processors, in microsecond-dollars.
AE[b]	Average Energy	Average energy consumption for benchmark b, across all processors, in watt-microseconds.

TABLE 8.0-5. Notation used in presentation of benchmark results (2 of 2).

8.1 Instruction Cycle Counts

In this section we present and analyze the BDTI Benchmark™ results in terms of the number of instruction cycles required for each processor to complete individual benchmark functions. These results form the foundation for subsequent metrics of execution time, cost-execution time, and energy use. They also provide a basis for extending the scope of subsequent sections' analysis to specific processor family members that were not selected for our benchmarking, but may be of interest to particular readers. For example, new execution time results can be calculated for a new family member with a higher clock speed by referring to the cycle count results in this section.

In most cases, cycle count results apply uniformly to all members of a particular processor family. This is also true for memory use results, but is not the case for other benchmark metrics, such as execution time, where results are specific to one family member. For a few processors, the cycle count results shown in this section do not apply to all family members. In particular, the Lucent DSP164xx results do not apply to the DSP162xx, because the former processor provides some instructions not found in the latter. In addition, the cycle count results shown for the Texas Instruments TMS320C62xx and TMS320C67xx do not apply to members of these families that rely on caches. Similarly, the results for the SC140 and MSC8101 do not apply directly to the recently announced Lucent StarPro 2000, which is a multiprocessor device based on the SC140 core and which uses caches.

As mentioned above, the cycles counts reported are for the main benchmark body. We exclude the power-up cycle counts from our analysis because the time spent in one-time processor setup code is usually inconsequential to overall performance in an application.

Instruction cycle counts by themselves are not especially useful as a performance measure. Only when combined with information about the *rate* at which each processor can execute instructions do cycle counts yield meaningful information about processor performance. (We refer to this rate as the processor *instruction cycle rate*.) Such a combination is the next step in our analysis and appears in Section 8.2, *Execution Times*.

Nevertheless, we include detailed processor instruction cycle count information in this section for two reasons. First, instruction cycle counts give some indication of the power of the architecture of a given processor. A processor with a more powerful architecture will execute a benchmark function in fewer cycles than a processor with a less powerful architecture. Of course, executing a function in fewer instruction cycles may or may not translate into executing the function more quickly, depending on the relative instruction cycle rates of the processors being compared. Second, processor instruction rates increase over time, as processor manufacturers improve their fabrication processes and circuit designs. Users may therefore want to use more recent processor instruction cycle rates to update the analysis presented in later sections of this report.

The benchmark instruction cycle count information presented in this section is organized as follows.

- **Benchmark instruction cycle count formulas and calculated values.**
  Benchmark instruction cycle count formulas and the values that result from substituting our selected values for each of the variable parameters are shown in Tables 8.1-1A and 8.1-1B, respectively. Note that each of these tables extends over two facing pages. The cycle count values are also plotted in Figures 8.1-1 through 8.1-12. Power-up cycles are excluded from these results, as discussed earlier.
- **Normalized total instruction cycle count values.**
  As the first step in creating aggregate cycle count results that illustrate each processor's overall cycle counts, we normalize the instruction cycle count for each processor on each benchmark. This normalization is done by dividing each processor's cycle count on a given benchmark by the average over all processors on that benchmark. This is done so that benchmarks that are inherently more time consuming are not automatically weighted more heavily when we aggregate the results for each processor over all benchmarks. The normalized total instruction cycle count values for each processor on each benchmark are shown in Table 8.1-2.
- **Total normalized instruction cycle count values through all benchmarks.**
  Figure 8.1-13 and the last row of Table 8.1-2 show the result of adding together the normalized instruction cycle counts for each processor across all benchmarks. This creates an overall processor instruction cycle count metric. Note that this approach applies a uniform weighting to each of the benchmark cycle counts. This weighting is arbitrary and will not be the best for most applications. Readers may wish to repeat our analysis with different weightings applied to the individual benchmarks, as called for by the application at hand. BDTI offers the Benchmark Analysis Tool™ to assist users in customizing our benchmark analysis in this way.

Note that in this section and in the later sections of this chapter, there may appear to be small errors in some calculated values. These small discrepancies are due to the fact that many values are displayed with limited precision, although full precision is used for calculations. For example, a value of 1.15 may be printed as 1.2. If two such values are added (i.e., 1.15 + 1.15), the correct result (2.3) is printed, rather than the sum of the rounded-off values (2.4).

Analysis of Results

This section presents analysis of the cycle counts for each benchmark. In this analysis, we focus on those processors that have especially large or small cycle counts for each benchmark and explain why this is the case. Bear in mind that while instruction cycle counts provide useful information about the power of a processor's architecture and instruction set, they are not reliable indicators of performance. To determine a processor's performance, one must take into account the instruction cycle time of that processor. In many cases, manufacturers make trade-offs between offering a powerful instruction set (requiring more complex hardware) and achieving a short instruction cycle time (using simpler hardware). This means that processors with large benchmark cycle counts may have much faster instruction cycle times than processors with small cycle counts. In Section 8.2 we combine instruction cycle time information with the cycle counts presented in this section to determine each processor's execution time on the benchmark functions.

Throughout the analysis in this section we present simplified descriptions of each of the benchmark functions. Detailed definitions of the benchmark functions are included in the BDTI Benchmarks specification.

General Observations

The more work a processor can get done per cycle on a given benchmark, the lower the cycle count. Overall, the newest processors, which use VLIW and/or SIMD techniques, achieve the highest parallelism on the BDTI Benchmarks and therefore the lowest cycle counts. Conventional DSPs, including newly introduced architectures like the Analog Devices ADSP-219x-C and Motorola DSP5685x, generally achieve less parallelism and therefore have higher cycle counts.

It is important to note, however, that different processors employ different kinds of parallelism, and therefore relative cycle counts can vary significantly from one benchmark to another. For example, some of the benchmarks, such as the Real Block FIR, are dominated by arithmetic computations, and are readily parallelized. On these benchmarks, highly parallel architectures tend to have the lowest cycle counts. In other benchmarks, even those with rather similar overall functionality, like the Single-Sample FIR, initialization and housekeeping operations play an important (in some cases, dominant) role in determining cycle counts. The Single-Sample FIR benchmark, for example, requires a total of 16 multiply-accumulate operations. The DSPs with the most parallel data paths can complete these MAC operations in a mere four cycles. Thus, tasks such as initializing address pointers, launching loops, and priming software pipelines typically account for most of the cycles in a Single-Sample FIR benchmark implementation on such processors. This suggests (and the benchmark results that follow confirm) that it is critical to evaluate a processor using benchmarks that are representative of the kinds of tasks that dominate the application at hand.

In the following paragraphs, we analyze the cycle count results for each benchmark function individually. As stated earlier, we focus on processors with unusually low or high cycle counts, and explain the reasons for these results. The cycle count results for each benchmark function are plotted in Figures 8.1-1 through 8.1-12. Further information on processor features that affect cycle count results can be found in Chapter 7, *Processor Analyses*.

Real Block FIR (Figure 8.1-1)

The Real Block FIR filter benchmark executes a 16-tap filter on 40 samples of input data. The benchmark nominally consists of an outer loop that is repeated 40 times (once for each input sample), and an inner loop that is repeated 16 times (once for each filter tap). Each output must be scaled. (Floating-point processors are exempt from this requirement since on floating-point processors coefficients can generally be scaled without loss of precision.) Additionally, processors must update the filter delay line; this can be done either by explicit copying of data or via a circular buffer.

The arithmetic-intensive nature of the Real Block FIR benchmark, and the ample parallelism available in the algorithm, mean that processors with highly parallel data paths (in particular, with more than one multiplier) can obtain low cycle counts. Conventional DSP processors (which provide a single MAC unit) generally require one instruction cycle per filter tap to compute the filter convolution, and one instruction cycle for scaling and storing the result. Conventional DSPs therefore spend at least 680 instruction cycles ([16 + 1] × 40) on the core of this benchmark, and require additional instruction cycles for initialization and housekeeping tasks. Processors that sport two multipliers can perform the convolution at a rate of two filter taps per instruction cycle. Assuming that one instruction cycle is needed for scaling and storing the result, two-multiplier processors spend at least 360 instruction cycles in the core of this benchmark. Similarly, processors with four multipliers can perform the convolution at a rate of four filter taps per instruction cycle, and require at least 200 cycles for the benchmark core. Note that every extra overhead cycle spent inside the outer loop costs cycles each time the outer loop is executed; most processors require some cycles in the outer loop for housekeeping and branching back to the start of the loop.

As expected, the conventional architectures uniformly require significantly more cycles on this benchmark compared with the enhanced conventional and VLIW-based processors. Among conventional DSPs, the Texas Instruments TMS320C54xx achieves the lowest cycle count: 730 cycles, quite close to the minimum of 680 cycles predicted above for the benchmark core on single-multiplier processors. The TMS320C54xx achieves its relatively low cycle count on this benchmark by making use of a specialized instruction (primarily intended for LMS filter implementations) to compute the convolution. This reduces the overhead in the outer loop by allowing the output to be scaled and stored in parallel with a multiplication operation in a single instruction cycle. In contrast, the cycle counts of the Analog Devices ADSP-218x and the Motorola DSP563xx, DSP568xx, and DSP5685x range from 855 to 983. These higher cycle counts are due to the fact that these processors spend more cycles on set-up and housekeeping overhead (for example, reloading address registers for each iteration of the outer loop) than the other conventional DSPs included here. Compared with the older Motorola DSP563xx, the Motorola DSP5685x achieves a lower cycle count; this is due to the fact that fewer cycles are required to initiate a hardware loop on the DSP5685x.

Also as expected, the processors with the most extensive data path parallelism have the lowest cycle counts on this benchmark. The processors with four multipliers— the StarCore SC140 and Texas Instruments TMS320C64xx-C—have the lowest cycle counts by a significant margin. The cycle counts of the two-multiplier processors are roughly twice as large those of the four-multiplier processors.

Not all of the highly parallel processors achieve their parallelism in the same way. For example, among dual-multiplier processors, the Texas Instruments TMS320C62xx and TMS320C67xx use the VLIW approach of executing multiple instructions in parallel, while the Analog Devices ADSP-2116x-C uses its SIMD feature to perform two multiplies in parallel under the control of a single instruction. The ADSP-2116x-C, however, spends a significant number of cycles in the outer loop to set up software pipelining and for explicitly enabling and disabling the SIMD mode; this gives it a higher cycle count than the other two-multiplier processors, which can perform some of the overhead operations in parallel with other operations.

Single-Sample FIR (Figure 8.1-2)

The Single-Sample FIR filter benchmark runs a 16-tap FIR filter on a single real input sample, producing a single output sample per invocation of the function. The processor must properly maintain the filter delay line (typically via a circular buffer), and fixed-point processors must scale the output. Because the benchmark only processes one sample of data at a time, initialization and housekeeping operations play a much larger role than they do in the cycle counts for the block-oriented benchmarks. Therefore, this benchmark is more reflective of processors' performance on housekeeping types of operations (e.g., setting up buffers and loops, loading and saving registers) than is the block-oriented FIR filter benchmark, which tends to emphasize processors' arithmetic capabilities. Processors with deep pipelines often suffer higher cycle counts on this benchmark, because pipeline latencies can increase the number of cycles required to perform initialization and housekeeping operations.

The inner loop of the benchmark requires 16 multiply-accumulate operations. Thus, a conventional DSP processor with one multiplier typically requires at least 16 cycles to perform the inner loop. A DSP with two multipliers typically requires at least 8 cycles, and a DSP with four multipliers typically requires at least four cycles for the inner loop. Among the processors benchmarked here, the lowest cycle count (14 cycles) is achieved by the StarCore SC140, which provides four multipliers. Approximately half of these 14 cycles are consumed by set-up and housekeeping operations. This supports the observation, mentioned above, that the set-up and housekeeping work required by this benchmark represents a large share of the work in the benchmark.

Other processors with notably low cycle counts on the Single-Sample FIR benchmark include the Lucent DSP164xx and Analog Devices ADSP-2116x-C (with cache preloaded). These processors utilize their dual multipliers effectively on this benchmark, and are efficient in the set-up and housekeeping code, in part due to their relatively shallow pipelines.

The Texas Instruments TMS320C64xx-C, despite its four multipliers, has a cycle count that is slightly higher than that of the two-multiplier Lucent DSP164xx on this benchmark, even with the TMS320C64xx-C caches preloaded. This difference is due to the higher number of cycles required by the TMS320C64xx-C for initialization and housekeeping code, which in turn is due partly to the deep pipeline of the TMS320C64xx-C. When its level-one caches are not preloaded, the estimated additional cycles required for loading the cache raise the TMS320C64xx cycle count substantially on this benchmark, yielding the highest cycle count of all of the benchmarked processors.

Texas Instruments' floating-point TMS320C67xx has an even deeper pipeline than does the TMS320C64xx, and this contributes to its notably high cycle count on this benchmark. In addition, on the TMS320C67xx, circular addressing does not support loading of two elements at a time, and this limitation hinders the processor in the inner loop of this benchmark.

The Analog Devices ADSP-219x-C also has a high cycle count on this benchmark, about 35% higher than the cycle count of its predecessor, the ADSP-218x. The relatively high cycle count of the ADSP-219x-C is due primarily to cycles required to initialize circular buffers on this processor and to stall cycles attributable to the deeper pipeline of the ADSP-219x-C.

Complex Block FIR (Figure 8.1-3)

The Complex Block FIR filter benchmark is similar to the Block FIR filter benchmark except that the input, output, and filter coefficient values are complex. As with the Real Block FIR, the filter parameters are 16 taps and 40 input samples, and (on fixed-point processors) the outputs are scaled. Processors must update the filter delay line.

Computing one tap of a complex FIR filter requires one complex multiplication, which consists of four real multiplies and two real additions. This requires four instruction cycles per tap on a processor capable of one multiply-accumulate per instruction cycle. Including two instruction cycles for scaling and storing results after each filter kernel, conventional DSP processors require at least 2,640 instruction cycles for the core of this benchmark. Processors with two multipliers are—ideally—capable of performing a complex multiplication using two instruction cycles per tap. On these processors, at least 1,280 instruction cycles are required in the benchmark core, not including cycles required for scaling and storing results. Similarly, on processors with 4 multipliers, at least 640 cycles are required for the benchmark core. As with the real FIR filter, extra cycles spent within the outer loop add significantly to the benchmark cycle count.

As in the Real Block FIR benchmark, the conventional architectures uniformly require significantly more cycles on this benchmark compared with the enhanced conventional and VLIW-based processors. Among conventional DSPs, the Analog Devices ADSP-218x and ADSP-219x-C and the Motorola DSP568xx have relatively high cycle counts. As in the Real Block FIR benchmark, these higher cycle counts are largely due to the fact that these processors spend more cycles on set-up and housekeeping overhead (for example, reloading address registers for each iteration of the outer loop) than the other conventional DSPs included here. In addition, the ADSP-218x is hampered by having only one accumulator; ideally, two accumulators would be used, one to accumulate the real part and one to accumulate the imaginary part of the running sum.

Also as expected, the processors with the most extensive data path parallelism have the lowest cycle counts on this benchmark. The processors with four multipliers—the StarCore SC140 and Texas Instruments TMS320C64xx-C—have the lowest cycle counts by a significant margin. The cycle counts of the two-multiplier processors are roughly twice as large those of the four-multiplier processors.

LMS Adaptive Filter (Figure 8.1-4)

The LMS benchmark implements a least mean-square adaptive FIR filter, which is an FIR filter whose coefficients are adjusted after each new output sample is produced. Each execution requires computing one output point of an FIR filter, computing an error signal (by subtracting the output from a supplied reference signal), and then, for each filter coefficient, multiplying a scaled version of the error value by the corresponding entry in the delay line, adding this product to the coefficient, and writing the new coefficient value back to memory (after rounding on fixed-point processors). As with the FIR filter benchmarks, the processor must also properly maintain the delay line; this is usually done via a circular buffer. The LMS Adaptive FIR filter is a single-sample benchmark, like the Single-Sample FIR benchmark. This means that it processes a single sample of input data per invocation of the function, and that initialization and housekeeping operations play a significant role in determining a processor's cycle count.

The StarCore SC140 has a markedly low cycle count on this benchmark—less than half that of the processor with the next-highest result, the Texas Instruments TMS320C64xx-C (with caches preloaded), which along with the TMS320C62xx, also has a low cycle count on this benchmark. The SC140 and TMS320C64xx-C achieve similar levels of parallelism in the inner loop of this benchmark. In fact, the TMS320C64xx-C has a slightly lower cycle count in the inner loop (3/4 cycle per tap) than does the SC140 (which requires about 1 cycle per tap). The significant difference in the cycle counts of these two processors is mostly explained by two factors. First, the deep pipeline of the TMS320C64xx-C and the resulting latencies associated with some key operations cause delays in the availability of certain results. Second, the MAC instructions of the SC140 are quite well suited to the computation required in this benchmark, including support for scaling and rounding of results. The TMS320C64xx-C uses its dot-product instruction, which is not quite as well matched to the needs of this benchmark; additional instructions are thus required for rounding and scaling of results. As a result of these differences, while the TMS320C64xx-C does have the second-lowest cycle count of the benchmarked processors on the LMS benchmark, its cycle count is about twice that of the SC140, and only about 10% less than that of the TMS320C62xx. Note that while the cycle counts of the TMS320C64xx-C and TMS320C62xx on this benchmark are higher than that of the SC140, they are lower than those of all of the other processors benchmarked, and are about 35% below the average.

Conventional DSPs generally have relatively high cycle counts on the LMS Adaptive Filter benchmark, due to their lower levels of parallelism. An exception to this is the Analog Devices ADSP-2106x-C; it takes advantage of the many general-purpose registers, flexible instruction set, and good support for operand-unrelated parallel moves to compute the FIR filter convolution in parallel with the coefficient update in two instruction cycles per tap, significantly fewer cycles than most conventional DSPs. The enhanced conventional ADSP-2116x-C cannot make efficient use of its SIMD mode in this benchmark; it uses the same implementation as the ADSP-2106x-C.

Among conventional DSPs, the Motorola DSP563xx and Analog Devices ADSP-219x-C have notably high cycle counts on this benchmark. Both of these processors compute one tap of the filter and perform one coefficient update in three cycles, as do most conventional DSPs. Thus, the higher cycle count results of these processors compared to other conventional DSPs are due to tasks outside of the inner loop(s) of the benchmark. In the case of the DSP563xx, the high cycle count is due in part to the cycles required to set up modulo addressing for the three separate circular buffers used. The ADSP-219x-C cycle count is significantly higher than that of its predecessor, the ADSP-218x. This is largely due to the five cycles required to launch a hardware loop on the ADSP-219x-C, and to the fact that some instructions that execute in one cycle on the ADSP-218x require multiple cycles on the ADSP-219x-C.

Two-Biquad IIR Filter (Figure 8.1-5)

The Two-Biquad IIR Filter is a single-sample benchmark, like the LMS Adaptive Filter and Single-Sample FIR Filter. The benchmark implements a cascade of two biquad filters, each of which involves four multiply-accumulate operations and the update of a two-sample delay line. Because practical IIR filters often require coefficient values with magnitudes larger than 1.0, this benchmark, unlike others, requires that processors accommodate coefficient values in the range of -2.0 to slightly less than +2.0. Because fixed-point processors generally use a fractional arithmetic format that accommodates values from -1.0 to almost +1.0, they must perform extra scaling steps to handle larger coefficient values and especially to mix them with data (e.g., input data) that may be represented using a different format. Some processors can implement these scaling steps more easily than others, often through mode bits that cause the processor to perform appropriate scaling when accessing multiplier outputs or when writing results to memory. These issues do not hinder floating-point processors, since they can handle numbers with very large dynamic range automatically.

For conventional DSPs, assuming one multiply-accumulate operation per instruction cycle, and assuming that the delay line update can be done in parallel (either through explicit data moves or modulo address arithmetic), each biquad takes at least 4 instruction cycles, for a total of 8 instruction cycles for the core of this benchmark (4 instruction cycles per biquad times 2 biquads). The cycle counts of the benchmarked processors range from 8 (for the StarCore SC140) to 30 (for the Texas Instruments TMS320C67xx). As with the Single-Sample FIR and LMS Adaptive Filter benchmarks, initialization and housekeeping operations contribute significantly to cycle counts on this benchmark.

The SC140 has the lowest cycle count on this benchmark—approximately half that of the processor with the next-lowest cycle count, the Lucent DSP164xx. The SC140 makes effective use of its data path parallelism, in this benchmark, performing up to four multiply or multiply-accumulate operations in parallel, and also benefits from its simple pipeline.

The Lucent DSP164xx has a relatively low cycle count on this benchmark. The DSP164xx makes good use of its two multipliers, direct support for the fractional number format used in this benchmark, and relatively good support for operand-unrelated parallel moves. These capabilities enable each IIR biquad to be calculated using three instruction cycles. The DSP164xx also benefits from its simple pipeline.

The cycle count of the Texas Instruments TMS320C64xx-C (with caches preloaded) on the IIR benchmark is about equal to the average, despite the processor's extensive parallelism. The higher cycle count of the TMS320C64xx-C is largely due to the processor's deep pipeline. Because of the deep pipeline, some key instructions have multi-cycle latencies. In addition, the TMS320C64xx-C requires a shift instruction to align results before storing them to memory, whereas the SC140 combines the shift with the store operation. Note that the cycle count for the TMS320C64xx-C on this benchmark is slightly higher than that of the TMS320C62xx. The TMS320C64xx-C is compatible with the TMS320C62xx and could use the same implementation of the benchmark. Our analysis uses a different implementation, however, because it has a similar cycle count but significantly lower memory use.

Similarly, although the Texas Instruments TMS320C55xx has significantly increased parallelism compared to the older TMS320C54xx, the two processors have the same cycle count on this benchmark. The TMS320C55xx implementation makes use of the processor's increased parallelism but, in order to do so, requires more elaborate initialization and housekeeping operations. The increased initialization and housekeeping cycles offset the reduction in cycle count due to the increase parallelism.

Despite its extensive parallelism, the Texas Instruments TMS320C67xx has the highest cycle count of the benchmarked processors on the IIR benchmark, roughly 40-50% higher than those of the conventional DSPs. As with the TMS320C64xx-C, the higher cycle count of the TMS320C67xx on this benchmark is largely due to the processor's deep pipeline. The pipeline of the TMS320C67xx is exceptionally long, and the resulting latencies (for example, four cycles for single-precision add and multiply operations) add a significant number of cycles to the cycle count for this benchmark.

If the benchmark were to use a longer filter (that is, more biquad sections), the cycle counts of processors with high levels of parallelism would be lower relative to those of other processors. For example, the TMS320C55xx would have a lower cycle count than the TMS320C54xx, as initialization and housekeeping operations would become a less significant component of the cycle counts.

Vector Dot Product (Figure 8.1-6)

The Vector Dot Product benchmark requires that processors perform a pair-wise multiplication of elements in two vectors and sum the resulting products. The vectors are 40 elements long. Although it has many other applications, the Vector Dot Product is essentially the heart of an FIR filter implementation, without the data movement and buffer management that is normally part of an FIR implementation.

Because this benchmark is comprised mostly of multiply-accumulate operations, a processor's multiply-accumulate throughput is a major factor in determining its cycle count on this benchmark. Since the vector lengths are moderate, though, additional cycles required for initialization, housekeeping, and pipeline latencies can also contribute significantly; this contribution can be particularly noticeable for processors that perform the MAC operations efficiently.

The StarCore SC140 has the lowest cycle count on the Vector Dot Product benchmark, significantly lower than that of the processor with the second-lowest result, the Texas Instruments TMS320C64xx-C. The SC140 uses its full data path parallelism to perform four multiply-accumulates per instruction cycle in parallel with data loads. The dot product computation is performed with no overhead except for two cycles used for adding the partial results. Other overhead cycles are limited to three, for launching a hardware loop, loading address registers, and priming the software pipeline.

The TMS320C64xx-C (with caches preloaded) has the second-lowest cycle count on this benchmark, but its cycle count is about 65% (10 cycles) higher than that of the SC140. Like the SC140, the TMS320C64xx-C uses its full data path parallelism to perform four multiply-accumulates per instruction cycle in parallel with data loads, yielding equivalent throughput in the inner loop. The difference in cycle counts is mostly attributable to pipeline latencies.

Although the Lucent DSP164xx has only two multipliers, its relatively short pipeline and its efficiency in executing the initialization and housekeeping operations in this benchmark give it a low cycle count—just slightly higher than that of the TMS320C64xx-C.

Conventional DSPs generally have relatively high cycle counts on the Vector Dot Product benchmark, due to their lower levels of parallelism. There are significant differences, though, among the cycle counts of the conventional DSPs, ranging from 45 for the Analog Devices ADSP-218x to 56 for the Analog Devices ADSP-219x-C (with cache preloaded). All of the conventional DSPs execute the dot product inner loop at a rate of one multiply-accumulate with two memory loads per cycle; hence, the differences in cycle counts among these processors are attributable to initialization and housekeeping operations, including pipeline delays associated with these operations. The higher cycle count of the ADSP-219x-C is due largely to the multiple cycles requires to launch and terminate a hardware loop.

Although the Texas Instruments TMS320C55xx has significantly increased parallelism compared to the older TMS320C54xx, the two processors have roughly the same cycle count on this benchmark. The reason for this is that the TMS320C55xx cannot make use of its increased parallelism here. The nature of the Vector Dot Product benchmark precludes use the TMS320C55xx's dual multipliers, because the two multipliers cannot share an operand in implementing this algorithm.

Vector Addition (Figure 8.1-7)

The Vector Addition benchmark requires processors to perform pair-wise addition of elements in two vectors, creating a third vector that represents the sum of the first two. Intuitively one might expect that since addition is a simpler operation than multiplication, the Vector Addition benchmark cycle counts would be at least as low, if not lower than those for the Vector Dot Product. However, this is not the case. While DSP processors are specifically optimized for computing dot products, they are often not as well suited to performing vector addition. One difference is that the Vector Add requires a memory write for each pair of input elements processed.

Again on this benchmark, the StarCore SC140 has the lowest cycle count, about 30% lower than that of the processor with the second-lowest cycle count, the Texas Instruments TMS320C64xx-C (with caches preloaded). Both the SC140 and the TMS320C64xx-C execute the inner loop at a rate of eight additions (with associated data moves) every three instruction cycles. As in the Vector Dot Product benchmark, the TMS320C64xx-C's higher cycle count is due to cycles required for initialization and housekeeping operations; the number of cycles required for these non-inner-loop operations is higher on the TMS320C64xx-C due to pipeline latencies.

The benchmarked conventional DSPs, with their limited parallelism and memory bandwidth, all have higher cycle counts that the enhanced conventional and VLIW-based DSPs on this benchmark. Cycle counts for the conventional DSPs are clustered fairly closely together, ranging from 75 for the Texas Instruments TMS320C54xx to 95 for the Analog Devices ADSP-219x-C (with caches preloaded).

Vector Maximum (Figure 8.1-8)

The Vector Maximum benchmark requires a processor to scan a vector of 40 data values to find the largest item in the vector. The processor must determine not only the value of the largest item, but also its position within the input vector. Unlike the benchmark functions described above, the Vector Maximum includes conditional operations (comparing two values and either conditionally executing certain instructions based on the result, or conditionally branching based on the result). Processors that have powerful and flexible conditional execution features achieve lower cycle counts on this benchmark, as do processors with limited conditional execution facilities that happen to be well suited to this benchmark. Refer to BDTI's textbook *DSP Processor Fundamentals*, for a discussion of conditional execution.

The StarCore SC140 has the lowest cycle count on this benchmark, about 20% lower than that of the processor with the second-lowest cycle count (the Lucent DSP164xx). This low cycle count is attributable to the SC140's ability to use the full parallelism of its data path in this benchmark—performing eight 16-bit comparisons and storing the results in a single cycle—and its extensive support for conditional execution of instructions.

The DSP164xx also has a low cycle count on this benchmark. The DSP164xx achieves this low cycle count via highly specialized data path and instruction set features. Owing to these specialized features, in a single cycle the DSP164xx uses its "trace-back encoder" to update one counter, make dual comparisons between two operands (using specialized dual compare instructions), and record which operands were largest. In parallel, two new operands can be loaded. Since these steps are exactly what are required in the Vector Maximum benchmark, the DSP164xx can process the elements at a rate of two elements per instruction cycle.

The Texas Instruments TMS320C62xx and TMS320C64xx-C (with caches preloaded) also achieve low cycle counts on this benchmark, taking advantage of their parallel data paths and conditional instruction execution capabilities. For this benchmark, these processors have less parallelism in their data paths than does the SC140. The TMS320C64xx-C performs four 16-bit comparisons per cycle, compared with eight for the SC140. The TMS320C62xx, unlike the TMS320C64xx-C and the SC140, does not have a maximum instruction that performs a comparison and a conditional assignment. Hence, the TMS320C62xx requires significantly more cycles in the inner loop of this benchmark. However, compared with the TMS320C64xx-C, this disadvantage is offset due to the lower number of housekeeping and initialization operations required by the TMS320C62xx implementation of the benchmark.

The benchmarked conventional DSPs, with their limited parallelism and (in most cases) limited conditional execution capability, all have higher cycle counts that the enhanced conventional and VLIW-based DSPs on this benchmark. Cycle counts for the conventional DSPs vary significantly, however, ranging from 89 for the Motorola DSP563xx and DSP568xx, to 170 for the Analog Devices ADSP-219x-C (with cache preloaded).

The high cycle count of the ADSP-219x-C is mostly due to the lack of support for a maximum compare instruction, which are offered on some current DSP processors. Also, the ADSP-219x-C benchmark implementation, unlike that of the ADSP-218x, does not use a conditional branch instruction in the inner loop, since branches on the ADSP-219x-C incur a significant cycle count penalty. Instead, the ADSP-219x-C replaces the conditional branch with several conditionally executed instructions. This results in an inner loop that requires more cycles than that of the ADSP-218x (because the conditional instructions each consume a cycle regardless of whether they are executed or bypassed). The low cycle count of the DSP563xx relative to other conventional DSPs is attributable to the use of a specialized MAX instruction and a conditionally executed single-move instruction.

Control (Figure 8.1-9)

The Control benchmark is considerably different from most common signal processing benchmarks. The Control benchmark is a contrived series of operations common in decision-making control tasks. The benchmark consists of a "while" loop surrounding a "switch" statement, with the various switch cases performing bit manipulation, data moves, subroutine calls, and stack manipulations. The primary goal for programmers implementing the Control benchmark is minimum memory use, and the secondary goal is speed of execution. (For the other functions in the BDTI Benchmarks, the priority of these goals is reversed.) Implementors are instructed to avoid optimizations that decrease the benchmark's execution time if the optimizations increase the benchmark's memory use. The Control benchmark replaces, and is different from, the FSM benchmark found in earlier BDTI publications.

In this section we compare the processors' cycle counts on the Control benchmark. While these cycle count results can reveal interesting architectural differences, readers should bear in mind that Control benchmark implementations are optimized for minimum memory use, not for minimum cycle counts. Memory usage results are discussed in Section 8.5, *Memory Usage Benchmarking*.

Control benchmark cycle counts for most processors are clustered within about 25% of the average. There is no clear correlation between architecture type and cycle count on this benchmark; for example, both conventional DSPs and VLIW-based DSPs are found among the processors with the highest and lowest cycle counts. In part this is because the Control benchmark does not offer much parallelism, instead emphasizing sequential processing.

The Analog Devices ADSP-218x and StarCore SC140 have the lowest cycle counts on this benchmark; their cycle counts are roughly equal. The ADSP-218x achieves its low instruction cycle count on this control-oriented benchmark through the use of register-to-register parallel moves, good immediate data support, and single-cycle branches and subroutine calls.

The low cycle count of the SC140 is largely due to the SC140's ability to conditionally execute execution sets (groups of instructions executed in parallel) using prefix words. While some of the other processors must make conditional jumps and branches, the SC140 can avoid these in many cases by executing or omitting an execution set based on the condition coded into the prefix word. It consumes only one cycle for such an instruction, which is significantly less than the three or four cycles required for a branch. (The Texas Instruments TMS320C62xx, TMS320C64xx-C, and TMS320C67xx provide a similar capability; they can conditionally execute instructions within execution sets and avoid branching in many cases.) In addition to its conditional execution capabilities, the

SC140 makes efficient use of delayed branches and calls, resulting in a lower cycle count compared to the other benchmarked processors.

The TMS320C62xx and TMS320C67xx use the same implementation for the Control benchmark, and achieve a low cycle count. The processors take advantage of conditional instruction execution, which helps eliminate many costly branches. Also, the processors have good support for immediate operands, which helps reduce cycle counts.

The high Control benchmark cycle count of the Texas Instruments TMS320C55xx is due to branch and subroutine call overhead, which consumes four or five cycles depending on whether the instruction is conditional and if conditional, whether the condition is true. Other processors either have a shorter pipeline or support delayed branches that allow instructions to be placed after a branch to execute in otherwise empty pipeline slots.

The cycle count of the Texas Instruments TMS320C54xx is also high for this benchmark, due to poor support for short immediate operands encoded in the instruction word, and because branches take several instruction cycles to execute on this processor even when delayed branches are used. Similarly, branches on the Motorola DSP563xx take between three and five cycles to execute, inflating cycle counts.

256-Point Fast Fourier Transform (Figure 8.1-10)

Fast Fourier transforms are a class of computationally efficient algorithms for computing the discrete Fourier transform. This transform is used to convert conventional time-domain signals into their frequency-domain representations. Our benchmark function specifies a complex, 256-point FFT. Implementations can use the radix-2 or radix-4 FFT variants, or a combination of the two. The radix-4 variant requires fewer arithmetic operations, but consumes more registers; therefore processors with few registers usually use the radix-2 approach. Within the inner loop of the FFT, a group of multiplication and addition operations called a ***butterfly*** is performed. The butterfly operates on complex numbers and involves computing the sum and the difference of one or more pairs of complex products. In addition to performing the FFT itself, processors must also reorder the input or output data, so that both input and output appear in natural order. (Without this step, the output is generated in a scrambled order, or the input must be provided in a scrambled order.)

Note that the 256-point FFT benchmark used here is different from the 256-point FFT used in earlier BDTI publications. The earlier version did not require reordering, and did not permit use of the radix-4 approach. For this reason, in general it is not meaningful to compare results from the earlier FFT benchmark with results from the current benchmark.

On the FFT benchmark, the Texas Instruments TMS320C64xx-C (with cache preloaded) has the lowest cycle count of the benchmarked processors, regardless of whether its level-one caches are preloaded. The TMS320C64xx-C cycle count is approximately half that of its predecessor, the TMS320C62xx. The TMS320C64xx-C makes good use of its extensive data path parallelism on this benchmark via its new (compared with the

TMS320C62xx) SIMD instructions. For example, the DOTP2 and DOTPN2 instructions on the TMS320C64xx-C are particularly well tailored for the FFT. In addition, new TMS320C64xx-C bit manipulation instructions, such as the bit reversing instruction, are effective for efficiently implementing the data reordering step of the FFT benchmark.

The StarCore SC140 also achieves a low cycle count on the FFT benchmark, approximately 30% higher than that of the TMS320C64xx-C. Like the TMS320C64xx-C, the SC140 makes good use of its data path parallelism on this benchmark. Throughout the core of the benchmark the SC140 performs an average of four arithmetic operations (a combination of multiply, multiply-accumulate, addition, and subtraction) and two move operations (a combination of stores and loads) per instruction cycle. While the SC140 requires the use of all four of its ALU/MAC/bit-field units to perform four arithmetic operations, the TMS320C64xx-C can perform four arithmetic operations (via two SIMD instructions) using only two of its eight execution units, leaving the remaining six execution units free to perform other arithmetic and shift operations. This enables the TMS320C64xx-C to achieve a higher level of parallelism than the SC140 on the FFT benchmark.

Due to the arithmetic-intensive nature of the FFT, and the ample parallelism available in the algorithm, it is not surprising that the conventional architectures generally require significantly more cycles on this benchmark compared with the enhanced conventional and VLIW-based processors.

Among conventional DSPs, the Analog Devices ADSP-2106x-C has an unusually low cycle count. The ADSP-2106x-C provides specialized instructions that assist in efficiently performing the butterfly operation. A single, specialized instruction can compute the sum and the difference of two operands. The ADSP-2116x-C, the enhanced-conventional successor to the ADSP-2106x-C, also benefits from this feature.

The Motorola DSP568xx has the highest cycle count on this benchmark. This is mainly a result of several limitations regarding dual parallel memory moves on this processor: A dual parallel move is limited to a dual read. A dual write, or a parallel read and write, is not possible. Further, only address register R3 can be used as an address register for the second parallel read in an instruction, and in such a read, address register R3 can be incremented or decremented only by one. The processor's cycle counts are further increased because it has no hardware support for bit-reversed addressing.

Viterbi Decoder (Figure 8.1-11)

Viterbi decoding is a technique commonly employed in communications systems to recover a convolutionally encoded signal from a noisy channel. While Viterbi decoding is often thought of as a signal processing algorithm, it is very different from more conventional signal processing algorithms such as filters and transforms. For example, while more conventional DSP algorithms typically make heavy use of operations such as multiplication, Viterbi decoding does not use multiplications, but rather is based on bit-manipulation, comparison, addition, and decision-making operations. The Viterbi Decoder benchmark is composed of two stages. In the first stage, "soft" decision values are used to generate a state transition table. In the second stage, the transition information is back-traced to determine the received bitstream.

The TMS320C64xx-C has the lowest cycle count on the Viterbi Decoder benchmark, regardless of whether its level-one caches are preloaded. This is mostly due to the use of quad SIMD arithmetic and quad SIMD logical (compare) instructions in the first part of the function (during the "add-compare-select" loop), using 8-bit data. The cycle count is also lowered by the use of the application-specific bit deinterleaving instruction (DEAL), which is intended for algorithms like the Viterbi decoder.

The StarCore SC140 also has a very low cycle count on the Viterbi benchmark, just slightly higher than that of the TMS320C64xx-C. The SC140 achieves a low cycle count via special instructions dedicated to Viterbi decoding, such as MAX2VIT and VSL. As mentioned above, the TMS320C64xx-C achieves a low cycle count via use of 8-bit data and associated quad 8-bit compare and add/subtract instructions. The SC140 does not support 8-bit data, but this difference is largely offset by the SC140's application-specific instructions and lower instruction latencies (due to a shallower pipeline).

Due to the specialized nature of the Viterbi algorithm, and the ample parallelism available in the algorithm, highly parallel architectures with specialized support for Viterbi decoding have the lowest cycle counts. In contrast, conventional architectures without specialized Viterbi support have the highest cycle counts.

The Motorola DSP568xx, a conventional DSP, has the highest cycle count on this benchmark by a significant margin. This is due primarily to restrictions on double-word read operations, and to its limited number of data registers. For the add-compare-select operation, the DSP568xx must reload operands to perform successive adds and subtracts since it does not have enough data registers to avoid reloading data words. On other processors with more flexible double-word read capabilities, it takes fewer instructions to do the reload. On other processors with more data registers, the data can be loaded once and used several times.

The Analog Devices ADSP-218x and ADSP-219x-C also have notably high cycle counts on the Viterbi Decoder benchmark. The main reason for this is that the ADSP-218x and ADSP-219x-C processors lack a shift-through-carry operation or other means for efficient bit interleaving. In addition, due to width of the processors' barrel shifter (16 bits), shifting a 32-bit word requires two cycles.

In contrast, the Texas Instruments TMS320C54xx, which is also a conventional DSP, has a cycle count about 70% lower than that of the DSP568xx. This is due in large part to a specialized single-cycle add-compare-select instruction that reduces the cycles required for the first stage of the benchmark.

Bit Unpack (Figure 8.1-12)

Like the Viterbi Decoder benchmark, the Bit Unpack benchmark is not a conventional signal processing algorithm. Rather, the Bit Unpack algorithm requires the processor to unpack data words of varying lengths from a continuous bit stream. This type of operation is common in audio and speech compression applications, where the compressed signal is transmitted as a compacted sequence of words of varying length. Typically, before these words can be processed further, they must separated and stored in distinct memory locations. The Bit Unpack benchmark therefore tests processors' bit-field-manipulation capabilities rather than their arithmetic capabilities.

The StarCore SC140 and the Texas Instruments TMS320C62xx, TMS320C67xx, and TMS320C64xx-C all make use of their extensive parallel resources and conditional instruction execution capabilities on this benchmark. Using conditional execution, these processors implement the Bit Unpack benchmark using a single loop without conditional branches. All other processors benchmarked here use branches. Thus, unlike the other benchmarked processors, the SC140, TMS320C62xx, TMS320C67xx, and TMS320C64xx-C have cycle counts on the Bit Unpack benchmark that are independent of the input data values.

The TMS320C64xx-C (with caches preloaded) has a cycle count that is about 25% lower than that of the SC140. This is primarily due to the ability of the TMS320C64xx-C to perform unaligned 64-bit data accesses, which are not supported on the SC140. In addition, the TMS320C64xx-C data path achieves more parallelism in this benchmark.

The TMS320C62xx lacks support for unaligned 64-bit data accesses, and this is the main reason why its cycle count is higher than that of the TMS320C64xx-C, and about equal to that of the SC140. The floating-point TMS320C67xx uses the same implementation of this benchmark as the fixed-point TMS320C62xx, since floating-point operations are not useful in this benchmark. Hence, the cycle counts of the two processors are equal.

The Texas Instruments TMS320C54xx has the highest cycle count for the Bit Unpack benchmark, about 30% higher than that of typical conventional DSPs benchmarked here. Compared to the other conventional DSPs benchmarked, the TMS320C54xx lacks flexible bit-field manipulation capabilities and thus must perform a larger number of discrete shift-and-logical-operation steps for each bit-field extraction. The TMS320C54xx also has limited register-to-register move support; hence, loading the shift control register takes more cycles than in competing processors that have more flexible register move support. Limited support for conditional instruction execution and operand-unrelated parallel moves also contribute to the high cycle count.

Total Normalized Cycle Counts (Figure 8.1-13)

The total normalized cycle counts provide a summary of the processors' cycle counts across the BDTI Benchmarks. As we would expect, processors with highly parallel data paths and powerful instruction sets achieve lower cycle counts overall than processors without these features. Nevertheless, even among processors with similar levels of parallelism, we find significant differences in cycle counts.

Of the processors benchmarked here, the two with the most extensive parallelism are the StarCore SC140 and the Texas Instruments TMS320C64xx. Not surprisingly, the SC140 has the lowest total normalized cycle count, about 65% below the average. In addition to its extensive parallelism, the SC140 is aided by its short pipeline, which helps reduce overhead cycles, and by its flexible conditional execution capabilities and powerful instructions.

The TMS320C64xx-C (with caches preloaded) has a total normalized cycle count that is about 40% higher than that of the SC140. While the data path parallelism of the TMS320C64xx-C is comparable to (indeed, in some cases, greater than) that of the SC140, the TMS320C64xx-C cycle counts are increased due to its deep pipeline and, in some cases, less powerful instructions.

The processors with the next-highest cycle counts are those with moderate parallelism (for example, two multipliers): the Texas Instruments TMS320C62xx, TMS320C67xx, and TMS320C55xx (all VLIW-based DSPs), and the Analog Devices ADSP-2116x-C and Lucent DSP164xx (both enhanced conventional DSPs). The total normalized cycle counts for these processors span the range between the highly parallel SC140 and TMS320C64xx-C (for example, the total normalized cycle count of the TMS320C62xx is only about 25% higher than that of the TMS320C64xx-C) to the much less parallel conventional architectures like the Analog Devices ADSP-2106x-C (which has a total normalized cycle count about 10% higher than that of the Texas Instruments TMS320C55xx).

Conventional DSPs have total normalized cycle counts that lie within a fairly narrow range, particularly if we exclude the floating-point ADSP-2106x-C with its 48-bit instruction words (which help give this processor an unusually powerful instruction set). The Texas Instruments TMS320C54xx; Motorola DSP563xx, DSP568xx, and DSP5685x; and Analog Devices ADSP-218x and ADSP-219x-C all have total normalized cycle counts that lie within 10% of their average.

It is important to note, however, that the total normalized instruction cycle counts can obscure significant benchmark-to-benchmark variations in relative cycle counts. For example, the Lucent DSP164xx has one of the lowest cycle counts on the Single-Sample FIR filter, but has one of the highest cycle counts on the Bit Unpack benchmark. On the Single-Sample FIR benchmark, the DSP164xx cycle count is about 30% lower than that of the Analog Devices ADSP-218x, while on the Bit Unpack benchmark, the DSP164xx cycle count is about 30% *higher.* This emphasizes the importance of selecting benchmarks that are relevant to the particular application under consideration, as we discussed earlier in this chapter.

Finally, the total normalized cycle counts show a correlation between large instruction word widths (which often correspond to a larger instruction set and more powerful instructions) and low instruction cycle counts. All of the processors with lower than average total normalized cycle counts use 32-bit instructions, mixed 16- and 32-bit instructions, or (in the case of the Texas Instruments TMS320C55xx) variable-length instructions ranging from 8 to 48 bits long. In contrast, the processors with significantly above-average total normalized instruction cycle counts all use 16- or 24-bit instructions.

Table 8.1-1A. Total Benchmark Cycle Formulas

(cont.)

DSP Building Block Functions	ADI 218x	ADI 219x	ADI 219x-C	ADI 2106x	ADI 2106x-C	ADI 2116x	ADI 2116x-C	Lucent 164xx
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	[3]
Real Block FIR	15+N*(5+T)	N*(T+3)+36	N*(T+3)+19	17+N*(7+(T-3))	12+N*(7+(T-3))	26+N*(6+T/2)	18+N*(6+T/2)	N*(T/2+3.5)+22
Single-Sample FIR	14+T	28+T	24+T	15+T	10+T	21+T/2	16+T/2	T/2 + 14
Complex Block FIR	14+N*(14+4*T)	28+N*(22+4*T)	16+N*(22+4*T)	23+N*(5+4*T)	14+N*(5+4*T)	30+N*(6+2*T)	24+N*(6+2*T)	N*(2*T+10)+ 9
LMS Adaptive FIR	18+3*T	39+3*T	31+3*T	32+2*T	23+2*T	32+2*T	23+2*T	(3/2)*T+32
Two-Biquad IIR	21	25	22	23	17	22	16	15
Vector Dot Product	5+N	18+N	16+N	11+N	6+N	15+N/2	10+N/2	N/2+6
Vector Add	6+2*N	17+2*N	15+2*N	8+2*N	6+2*N	9+N	7+N	48
Vector Maximum	10+3*N+M	4*N+10	4*N+10	3+3*N	3+3*N	11+N	11+N	N/2+13
Control	440	618	618	493	493	493	493	567
256-Point FFT	10142	10163	10152	3766	3761	2622	2617	7118
Viterbi	30727	30737	30737	14620	14620	11402	11402	9358
Bit Unpack	379	460	460	270	270	270	270	487

T = # of Taps   N = # of Points   M = # of New Relative Maxima

Table 8.1-1B. Total Benchmark Cycle Counts

(cont.)

DSP Building Block Functions	ADI 218x	ADI 219x	ADI 219x-C	ADI 2106x	ADI 2106x-C	ADI 2116x	ADI 2116x-C	Lucent 164xx
Notes:		[1,2]	[1,2]	[1]	[1]	[1]	[1]	[3]
Real Block FIR	855	796	779	817	812	586	578	482
Single-Sample FIR	30	44	40	31	26	29	24	22
Complex Block FIR	3134	3468	3456	2783	2774	1550	1544	1689
LMS Adaptive FIR	66	87	79	64	55	64	55	56
Two-Biquad IIR	21	25	22	23	17	22	16	15
Vector Dot Product	45	58	56	51	46	35	30	26
Vector Add	86	97	95	88	86	49	47	48
Vector Maximum	135	170	170	123	123	51	51	33
Control	440	618	618	493	493	493	493	567
256-Point FFT	10142	10163	10152	3766	3761	2622	2617	7118
Viterbi	30727	30737	30737	14620	14620	11402	11402	9358
Bit Unpack	379	460	460	270	270	270	270	487

Substituted Values: T = 16   N = 40   M = 5

Notes:

[1] These processors use on-chip caches. Results under processor names denoted with "-C" represent performance with caches preloaded. Results for the processors listed under processor names without "-C" represent performance with caches uninitialized; see text for details.

[2] Projected; as of this writing, these processors were not yet sampling or have not been demonstrated in silicon at their target speed.

[3] Results are for one of the two-on-chip cores.

[4] Results are estimated based on preliminary cache information provided by Texas Instruments.

Table 8.1-1A. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]			[2]		[1,2, 4]	[1,2]	
24+N*(7+T)	N*(T+8)+23	N*(T+5)+23	N*(T+1)/4+13	T*(N+2)+N+18	393	8*ceiling(T/8,1)*N/2+28	279	34+(T*N)/4	13+(15/4)*N+(N*T/2)
18+T	14+T	13+T	T/2+6	T+15	25	T+9	46	(3*T)/8+19	(3*T)/4+31
25+N*(7+4*T)	23+N*(4*(T-2)+22)	25+N*(4*(T-2)+15)	N*(T+0.5)+12	21+N*(13+4*(T-1))	1410	ceiling(T/4,1)*8*N+27	749	35+(T*N)	14+((2*T)+29/2)*N
35+3*T	23+3*(T-1)	22+3*T	(3*T)/4+6	(3*T)+26	63	T*9/8+24	67	(3*T)/4+26	44+(T-4)*(5/4)
20	20	22	8	17	17	16	27	18	30
12+N	N+9	N+7	N/4+5	N+5	48	N/2+11	44	N/4+15	N/2+26
13+2*N	11+N*2	11+N*2	3*(N/8)+4	3*(N/2)+15	48	(N/4)*3+10	46	3*N/8+12	10+N
9+2*N	13+4*(N/2-1)	26+5*(N/2-1)	N/4+16	30+5*((N/2)-1)	71	(N/5)*3+12	56	3/8*N+20	14+((3*N)/4)
840	663	726	443	862	945	479	548	543	479
8824	14860	11708	1632	10406	7386	2492	1444	1246	3557
18029	35983	28533	1938	11264	6877	6436	1830	1741	5320
398	478	453	129	584	249	132	126	97	132

Table 8.1-1B. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]			[2]		[1,2, 4]	[1,2]	
944	983	863	183	730	393	348	279	194	483
34	30	29	14	31	25	25	46	25	43
2865	3143	2865	672	2941	1410	1307	749	675	1874
83	68	70	18	74	63	42	67	38	59
20	20	22	8	17	17	16	27	18	30
52	49	47	15	45	48	31	44	25	46
93	91	91	19	75	48	40	46	27	50
89	89	121	26	125	71	36	56	35	44
840	663	726	443	862	945	479	548	543	479
8824	14860	11708	1632	10406	7386	2492	1444	1246	3557
18029	35983	28533	1938	11264	6877	6436	1830	1741	5320
398	478	453	129	584	249	132	126	97	132

Table 8.1-2. Normalized Benchmark Cycle Counts (cont.)

DSP Building Block Functions	ADI 218x	ADI 219x	ADI 219x-C	ADI 2106x	ADI 2106x-C	ADI 2116x	ADI 2116x-C	Lucent 164xx
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	[3]
Real Block FIR	1.39	1.29	1.26	1.32	1.32	0.95	0.94	0.78
Single-Sample FIR	0.99	1.45	1.31	1.02	0.85	0.95	0.79	0.72
Complex Block FIR	1.45	1.60	1.60	1.29	1.28	0.72	0.71	0.78
LMS Adaptive FIR	1.07	1.41	1.28	1.04	0.89	1.04	0.89	0.91
Two-Biquad IIR	1.06	1.26	1.11	1.16	0.86	1.11	0.81	0.76
Vector Dot Product	1.08	1.39	1.35	1.23	1.11	0.84	0.72	0.62
Vector Add	1.32	1.48	1.45	1.35	1.32	0.75	0.72	0.73
Vector Maximum	1.57	1.98	1.98	1.43	1.43	0.59	0.59	0.38
Control	0.74	1.04	1.04	0.83	0.83	0.83	0.83	0.95
256-Point FFT	1.60	1.61	1.60	0.60	0.59	0.41	0.41	1.12
Viterbi	2.04	2.04	2.04	0.97	0.97	0.76	0.76	0.62
Bit Unpack	1.21	1.47	1.47	0.86	0.86	0.86	0.86	1.55
Total	15.51	18.02	17.49	13.09	12.31	9.81	9.03	9.95

Notes:

[1] These processors use on-chip caches. Results under processor names denoted with "-C" represent performance with caches preloaded. Results for the processors listed under processor names without "-C" represent performance with caches uninitialized; see text for details.

[2] Projected; as of this writing, these processors were not yet sampling or have not been demonstrated in silicon at their target speed.

[3] Results are for one of the two-on-chip cores.

[4] Results are estimated based on preliminary cache information provided by Texas Instruments.

Table 8.1-2. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]			[2]		[1,2, 4]	[1,2]	
1.53	1.59	1.40	0.30	1.18	0.64	0.56	0.45	0.31	0.78
1.12	0.99	0.95	0.46	1.02	0.82	0.82	1.51	0.82	1.41
1.33	1.45	1.33	0.31	1.36	0.65	0.60	0.35	0.31	0.87
1.35	1.10	1.14	0.29	1.20	1.02	0.68	1.09	0.62	0.96
1.01	1.01	1.11	0.40	0.86	0.86	0.81	1.37	0.91	1.52
1.25	1.18	1.13	0.36	1.08	1.15	0.74	1.06	0.60	1.11
1.42	1.39	1.39	0.29	1.15	0.73	0.61	0.70	0.41	0.77
1.03	1.03	1.41	0.30	1.45	0.83	0.42	0.65	0.41	0.51
1.41	1.11	1.22	0.74	1.44	1.58	0.80	0.92	0.91	0.80
1.39	2.35	1.85	0.26	1.64	1.17	0.39	0.23	0.20	0.56
1.20	2.39	1.89	0.13	0.75	0.46	0.43	0.12	0.12	0.35
1.27	1.52	1.44	0.41	1.86	0.79	0.42	0.40	0.31	0.42
15.31	17.12	16.26	4.26	15.01	10.71	7.30	8.85	5.93	10.06

Figure 8.1-1. Total Cycles for Real Block FIR

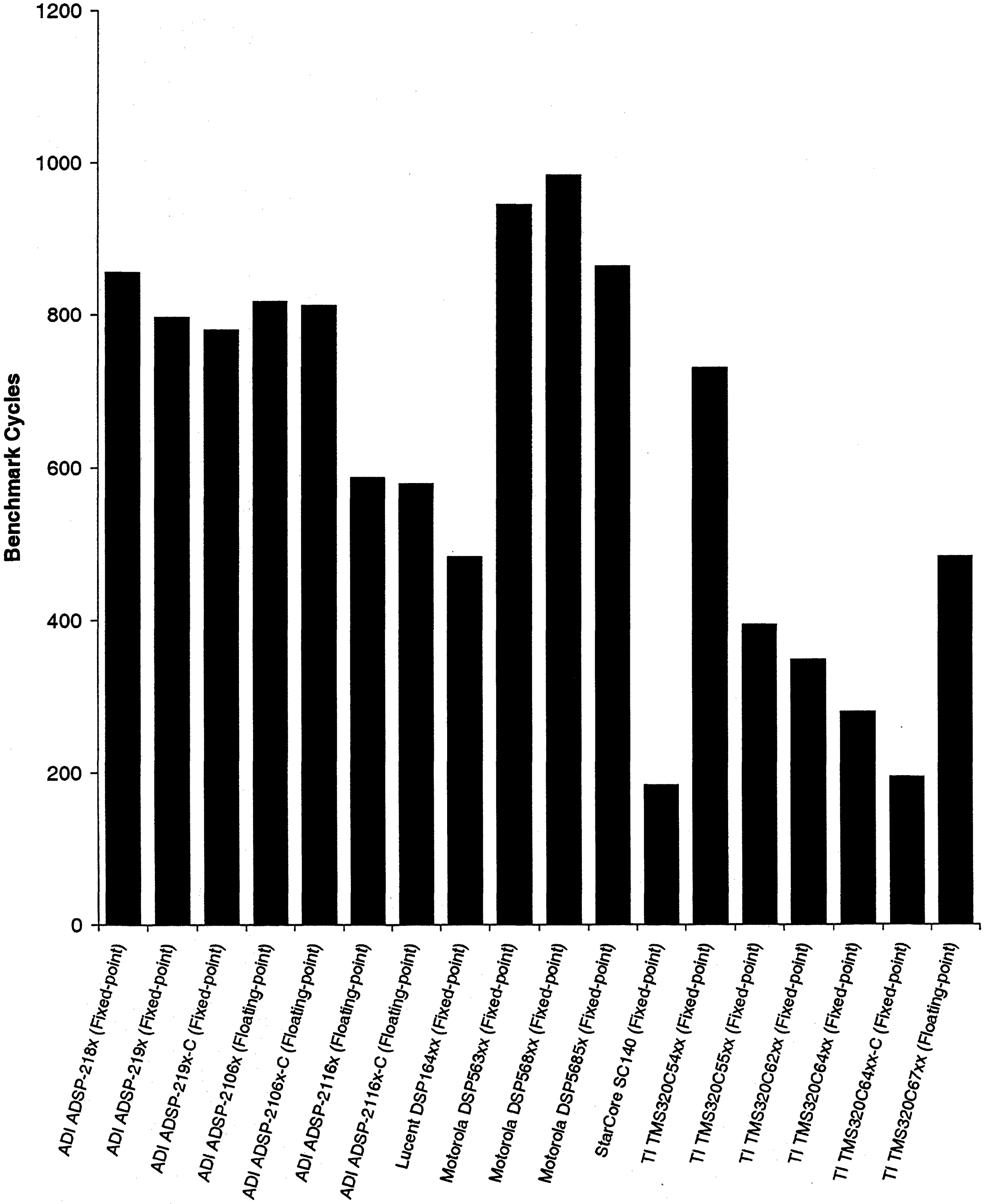


Figure 8.1-2. Total Cycles for Single-Sample FIR

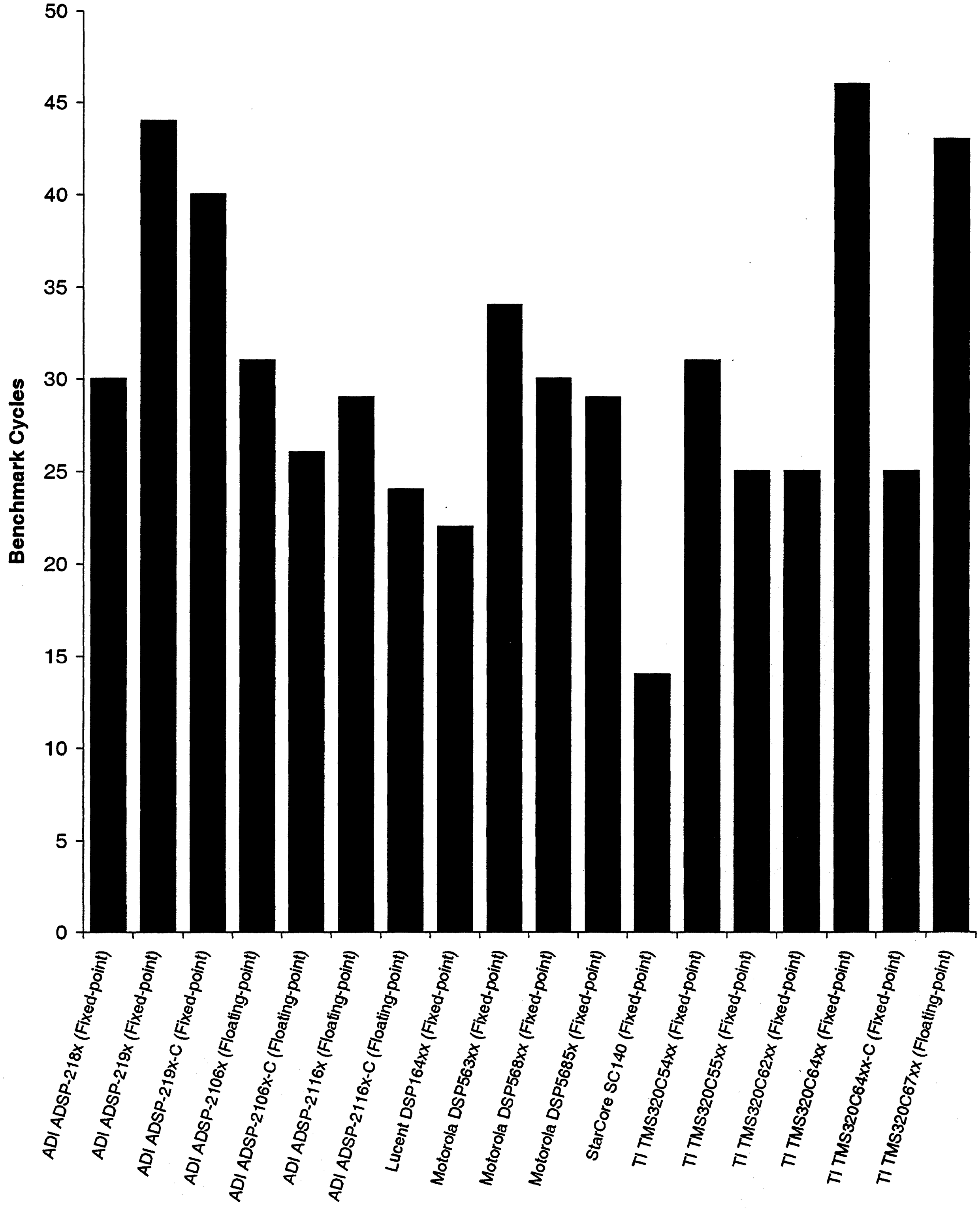


Figure 8.1-3. Total Cycles for Complex Block FIR

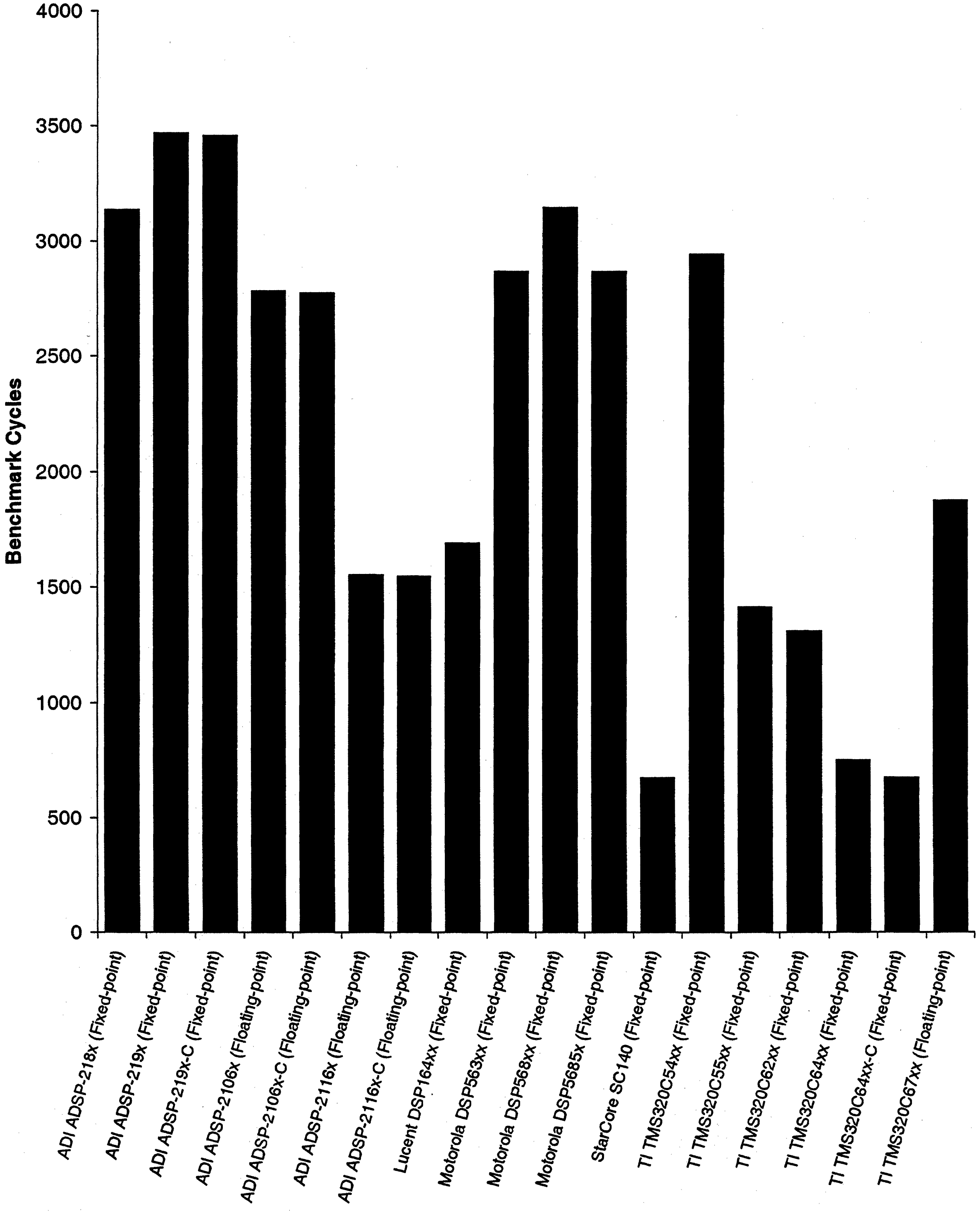


Figure 8.1-4. Total Cycles for LMS Adaptive FIR

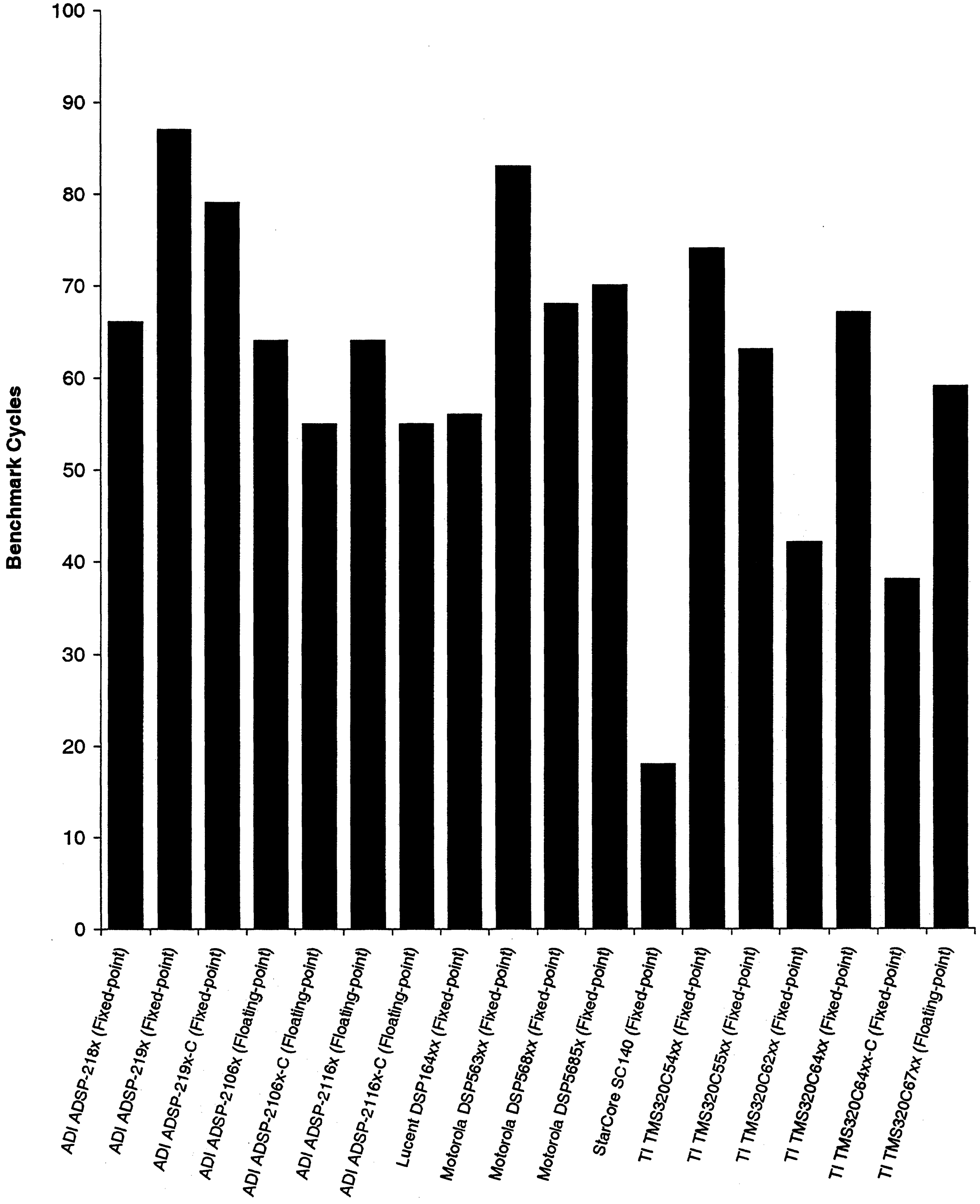


Figure 8.1-5. Total Cycles for Two-Biquad IIR

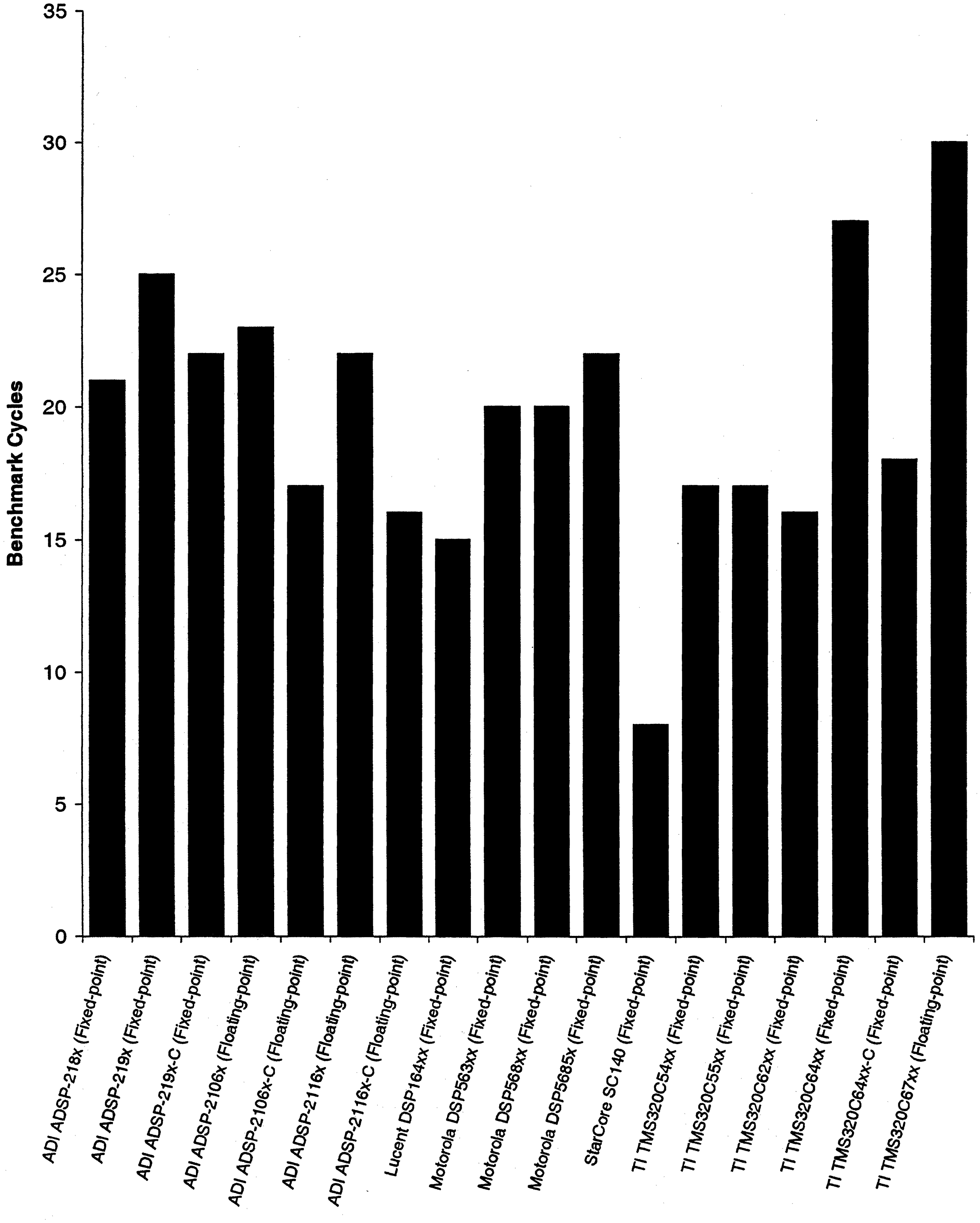


Figure 8.1-6. Total Cycles for Vector Dot Product

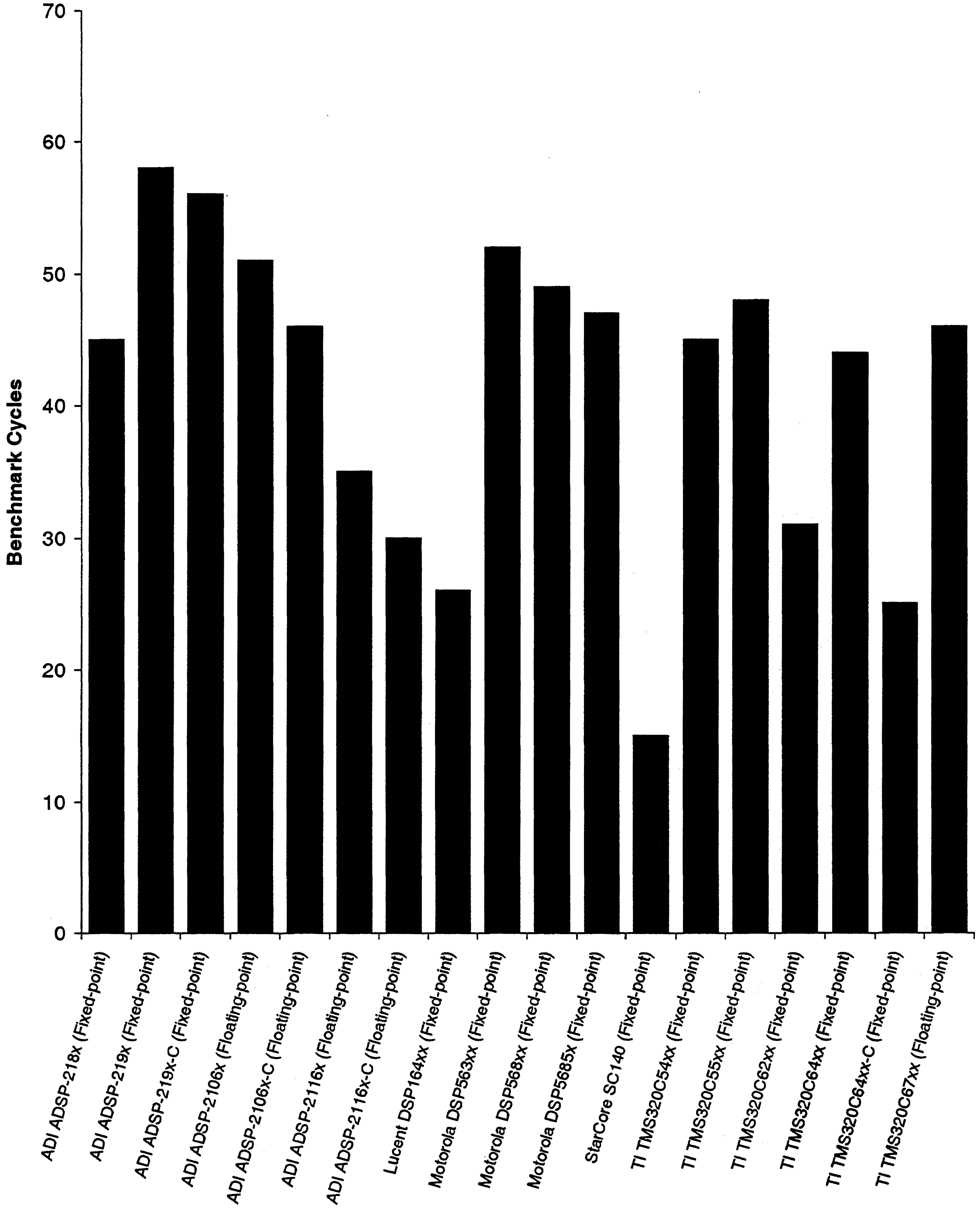


Figure 8.1-7. Total Cycles for Vector Add

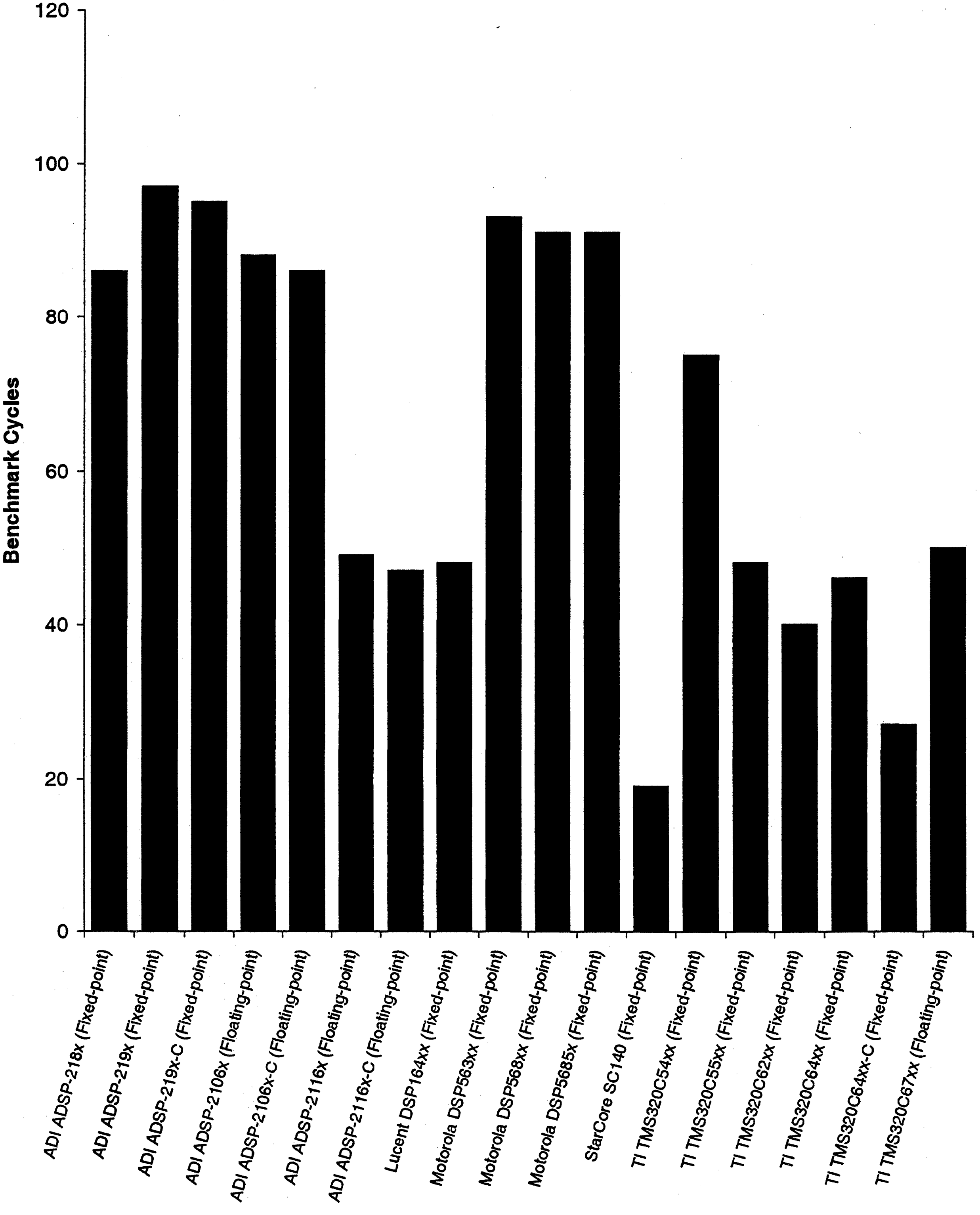


Figure 8.1-8. Total Cycles for Vector Maximum

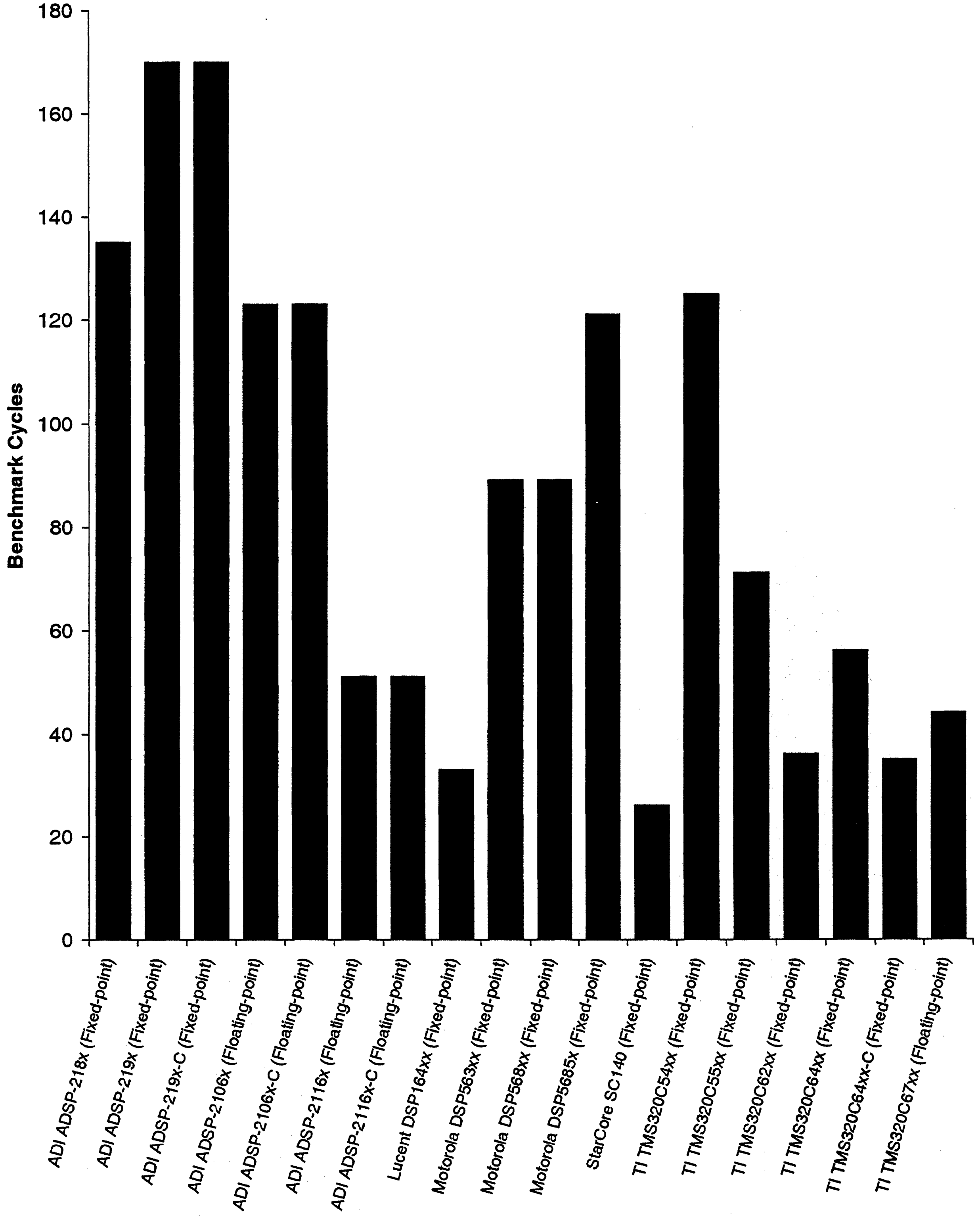


Figure 8.1-9. Total Cycles for Control

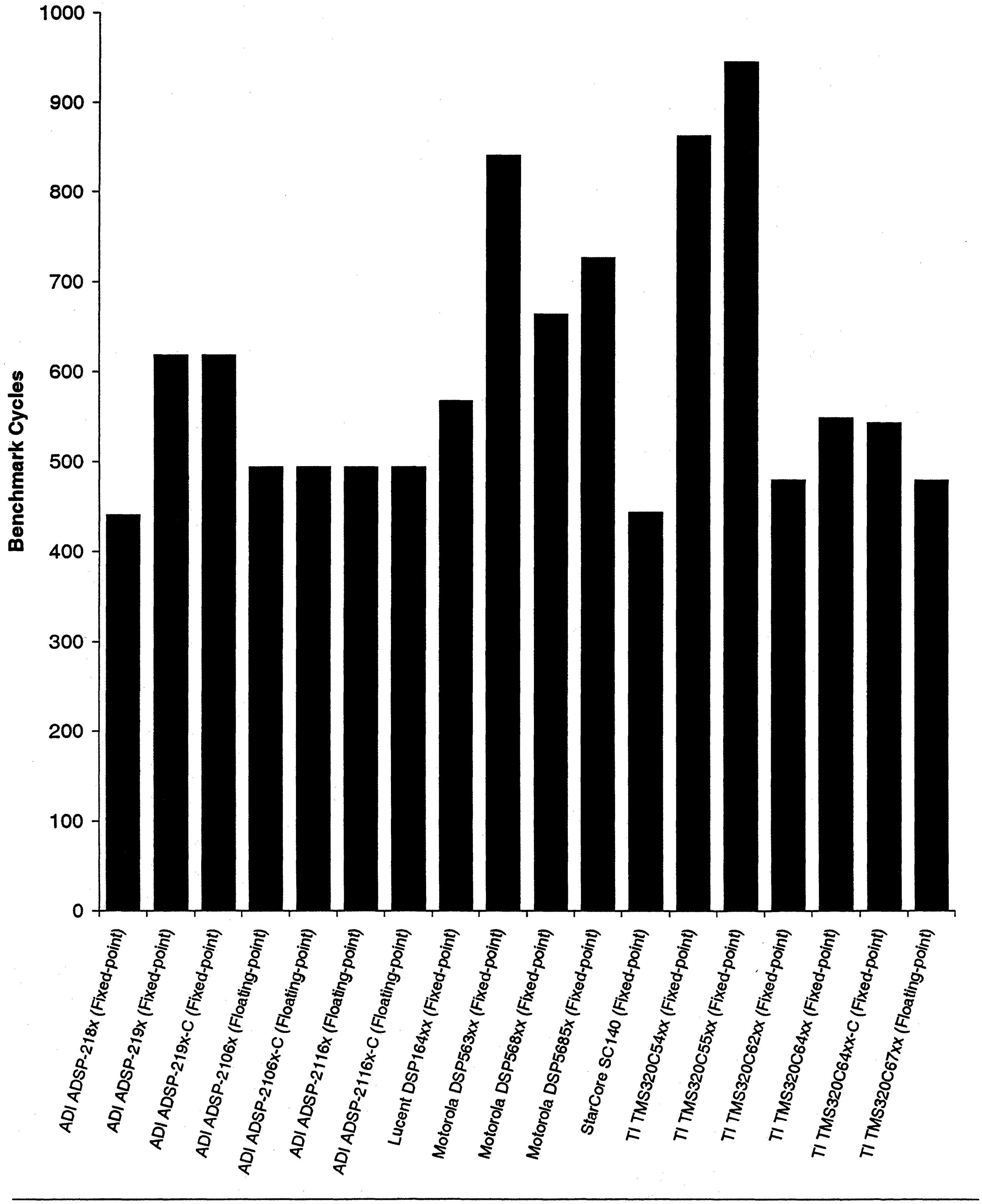


Figure 8.1-10. Total Cycles for 256-Point FFT

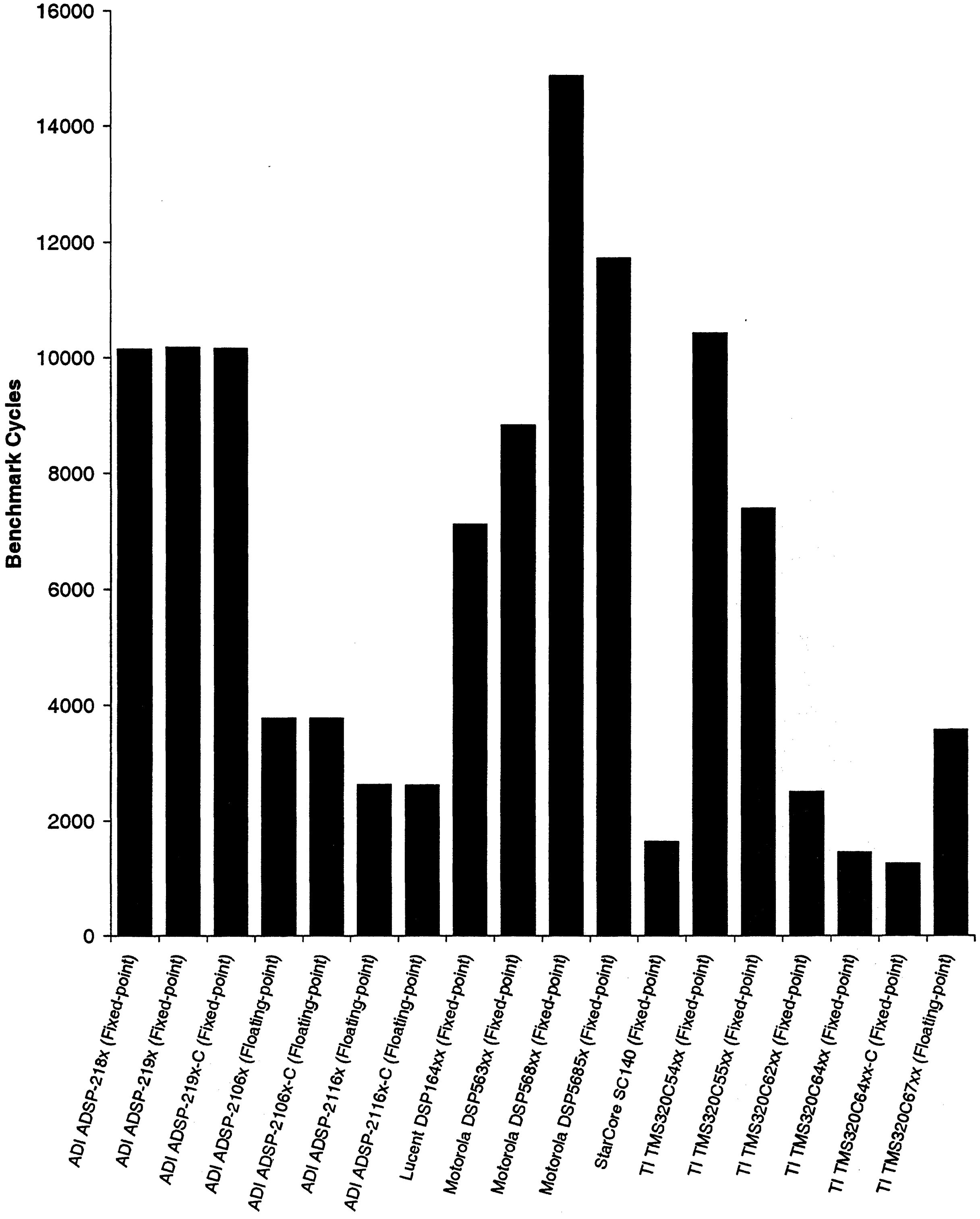


Figure 8.1-11. Total Cycles for Viterbi

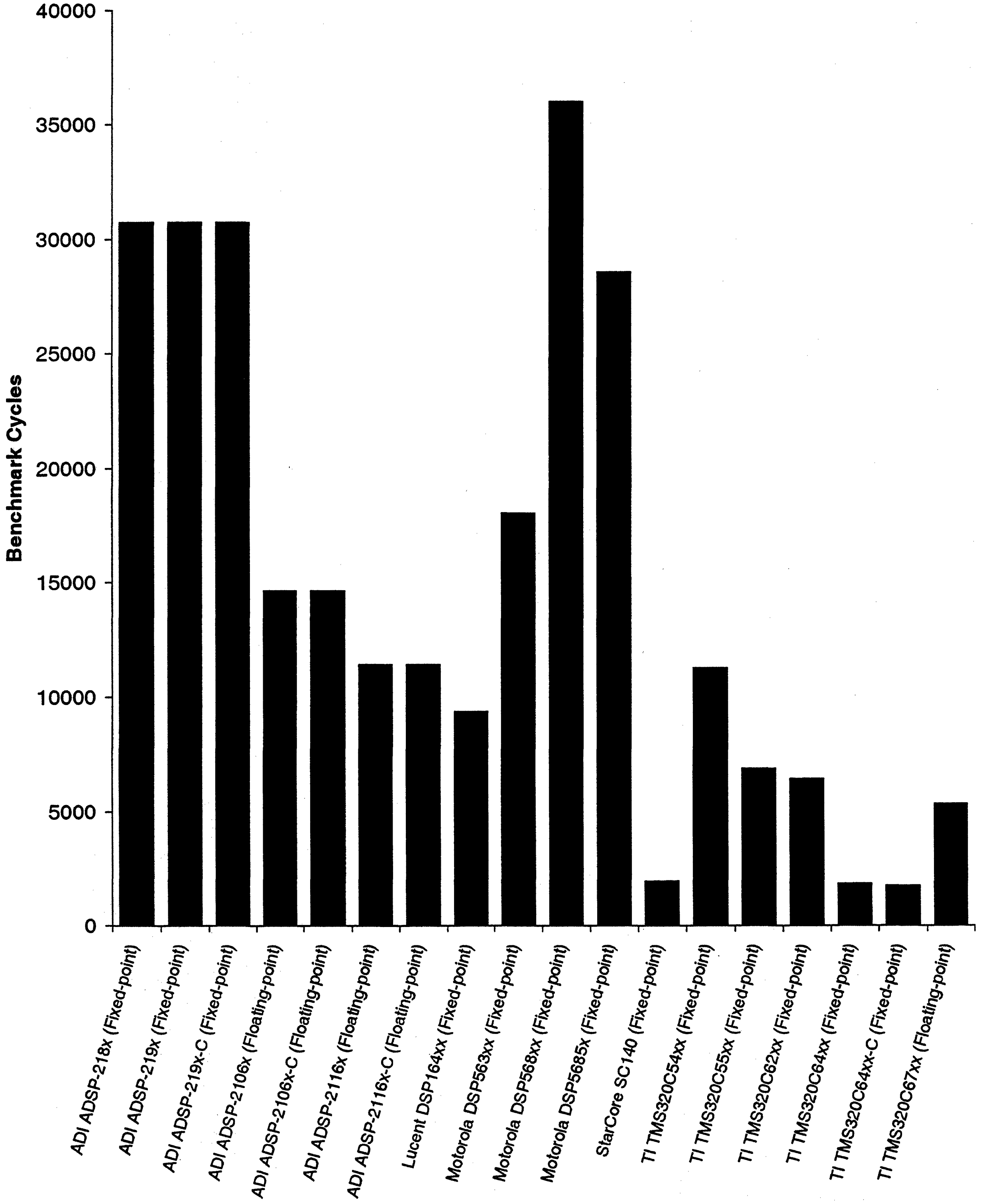


Figure 8.1-12. Total Cycles for Bit Unpack

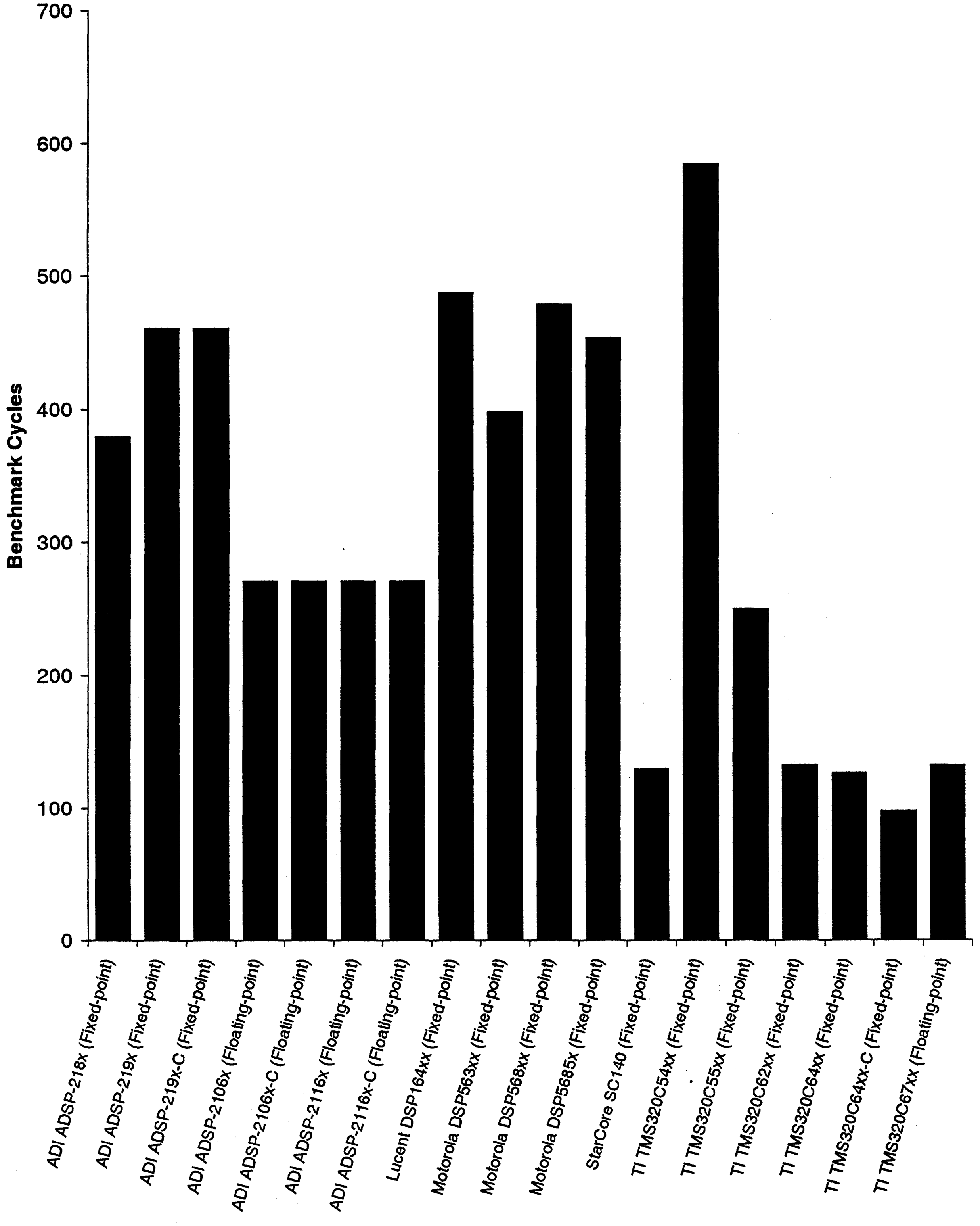
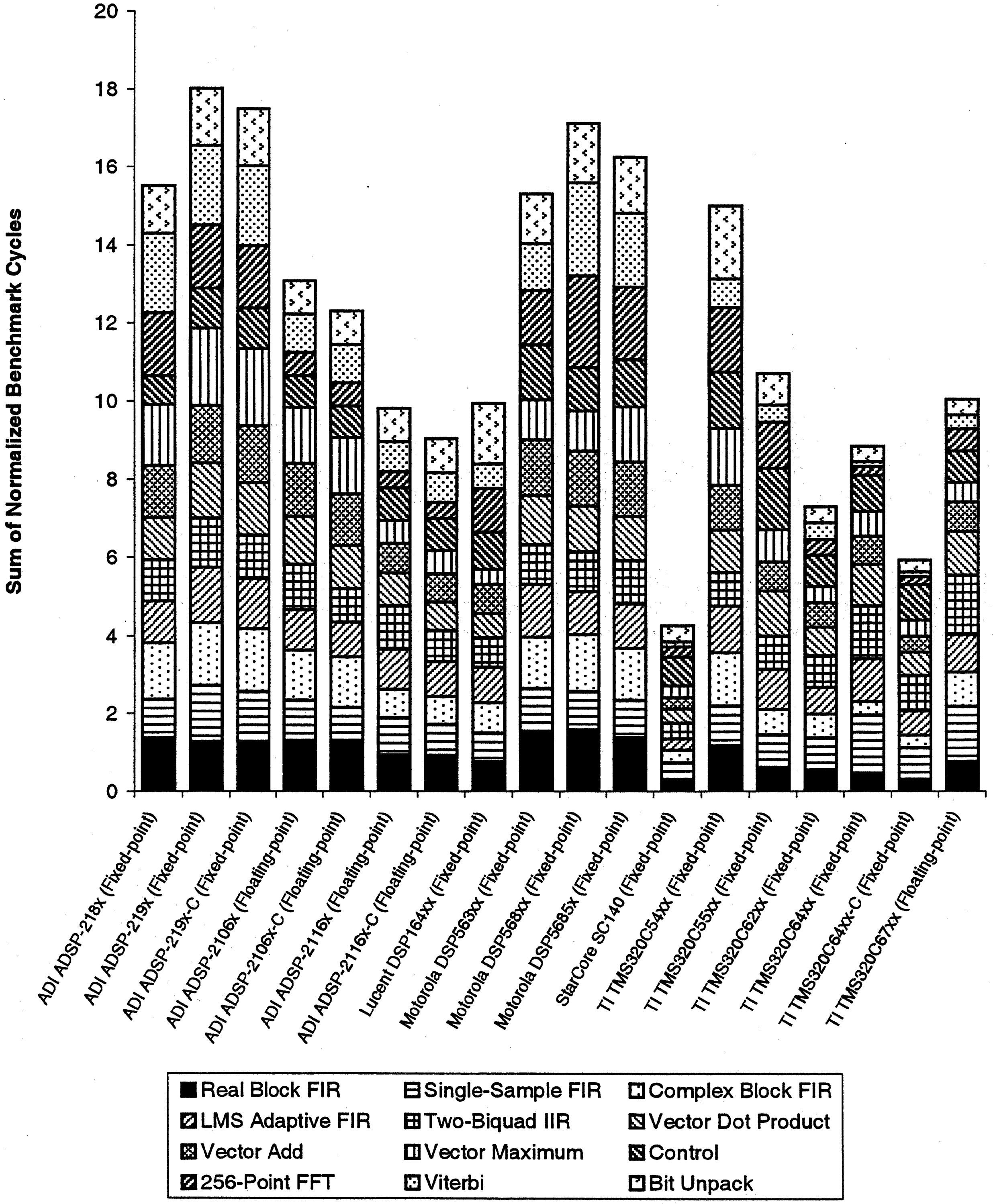


Figure 8.1-13. Normalized Benchmark Cycle Counts
Sum for all b of C[p,b] / AC[b]



8.2 Execution Times

The execution time for a BDTI Benchmark™ function is defined as the amount of time required by the processor to execute the main body of the benchmark. To determine the execution time of a particular benchmark on a given processor, we multiply the number of instruction cycles the processor requires to execute the benchmark by the processor's instruction cycle time.

A processor's instruction cycle time sometimes differs from its master clock period. Further, the ratio of input clock period to instruction cycle time may vary even for a single processor, since some processors have programmable clock dividers or come in versions with different master clock rate divider ratios. As a result, care is required to correctly determine the instruction cycle time for a particular processor.

In Table 8.2-1 we list the instruction cycle time used for each of the selected processor families. The table also gives the processor's instruction cycle rate, the reciprocal of its instruction cycle time. Most of the processors included here were available, at least in sample quantities, at the time of this writing (October, 2000). In such cases, we use the speed of the fastest family member available at the time of this writing. For processors not available yet, we use the vendor's projected clock rate, and mark the results as projected. This includes the Analog Devices ADSP-219x, the Motorola DSP5685x, and the Texas Instruments TMS320C64xx. Table 8.2-1 also includes pricing information that is used in the next section of our analysis.

Note: Except where noted, the instruction cycle times shown here and used in the subsequent analysis are for the fastest version of each processor available in sample quantities as of June 2000, according to the manufacturer. Unit prices are for quantity 10,000 purchases, also as reported by the manufacturer. BDTI has not independently verified this information.

Note that the Analog Devices ADSP-219x, Motorola DSP5685x, and Texas Instruments TMS320C64xx are not available as of this writing. In these cases, we use the vendors' projected speeds, and mark the results as projected.

Readers should be aware that, over time, device speeds tend to increase and prices tend to decrease, even for existing products. In addition, unit prices for DSP processors vary significantly depending upon the volumes purchased. For these reasons, we strongly recommend contacting manufacturers for updated information about prices and maximum device speeds.

The analysis of benchmark execution times is presented as follows:

- **Benchmark execution times**
  The execution time for the fastest version of each processor on each benchmark is shown in Table 8.2-2 and Figures 8.2-1 through 8.2-12. The execution time includes time for the main benchmark body, but not for the power-up section.
- **Normalized benchmark execution times**
  As the first step in creating aggregate execution time results that show each pro-

cessor's overall execution time performance, we normalize the execution time for each processor on each benchmark. This normalization is done by dividing each processor's execution time on a given benchmark by the average over all processors on that benchmark. This is done so that benchmarks that are inherently more time-consuming are not automatically weighted more heavily when we aggregate the performance of each processor over all benchmarks. Normalized benchmark execution times are presented in Table 8.2-3.

- **Total benchmark execution times through all benchmarks**
  Adding together the normalized benchmark execution times for all benchmarks for each processor produces an overall execution-time measure for that processor. This sum effectively applies a uniform weighting to each benchmark for this section of the analysis. Without the normalization step, benchmarks that require more time would tend to be weighted more heavily in the overall results. The results are shown in Figure 8.2-13 and at the bottom of Table 8.2-3.
- **BDTImark2000™ scores**
  The BDTImark2000 is a composite performance metric that is based on a processor's execution time performance on the BDTI Benchmarks. The BDTImark2000 scores differ from the total normalized execution time results in two ways: First, a higher BDTImark2000 score indicates a faster processor, whereas a lower execution time result indicates a faster processor. Second, compared to the total normalized execution time results, the BDTImark2000 is calculated using a different approach for combining results on individual benchmarks. The BDTImark2000 is designed to provide a convenient shorthand for processors' DSP speed, and is far more accurate than MIPS or MFLOPS for this purpose. BDTImark2000 scores are shown in Figure 8.2-14. BDTImark2000 scores are not provided for processors not yet available in silicon. BDTImark2000 scores for additional processors are posted on BDTI's website, *www.BDTI.com.* A white paper that describes the methodology used to develop the BDTImark2000 is also available on the site.

Note that BDTImark2000 replaces the original BDTImark metric. Whereas the original BDTImark was based on an earlier version of the BDTI Benchmarks, the BDTImark2000 is based on the latest version of the BDTI Benchmarks. Hence, BDTImark2000 scores are *not* comparable to earlier BDTImark scores.

BDTImark2000 scores are formulated so that in general, a processor's BDTImark2000 score will be much higher than its BDTImark score; this has been done to help avoid accidental comparisons between BDTImark2000 scores and older BDTImark scores.

Analysis of Results

This section presents a brief analysis of the benchmark execution times, summarized in Figure 8.2-13. As discussed in the preceding paragraphs, benchmark execution time is computed by multiplying a processor's instruction cycle time by the number of instruction cycles required for the processor to complete a given benchmark function.

Before beginning our discussion of execution time results, it is important to note the very wide range of maximum instruction cycle rates of the benchmarked processors. As illustrated in Table 8.2-1, instruction cycle rates range over an order of magnitude, from 40 MHz for the Motorola DSP56F801 (with an 80 MHz clock) to a projected 600 MHz for the Texas Instruments TMS320C64xx. The wide differences in instruction cycle rates, combined with variations in instruction cycle counts, create a very wide range of total normalized execution time results, as presented in Figure 8.2-13. In the total normalized cycle count results, the difference between the lowest result and the highest result is approximately a factor of four. In contrast, in the total normalized execution time results, the difference between the lowest result and the highest result is approximately a factor of *forty.* This large range is not completely unexpected, given that the processors evaluated here are targeting a variety of applications. For example, the Texas Instruments TMS320C64xx-C, which has the fastest total normalized execution time result, is targeting performance-hungry infrastructure applications, while the Motorola DSP56F801, which has the slowest total normalized execution time result, is targeting applications where cost and energy efficiency are primary considerations.

In reviewing the normalized execution times over all benchmarks presented in Figure 8.2-13, a few key points emerge. First, some processors with high cycle counts also have slow instruction cycle rates, and consequently achieve slow execution times. This is the case, for example, with the Motorola DSP56F801. However, processors with high instruction cycle counts do not necessarily achieve the slowest execution times. For example, the Analog Devices ADSP-219x-C has the highest total normalized instruction cycle result of the processors benchmarked here, but this is somewhat offset by its moderately fast projected instruction cycle rate of 160 MHz. As a result, the total normalized execution time result for the ADSP-219x-C is roughly equal to the average of the processors benchmarked here.

Only three floating-point processors are benchmarked here, reflecting the relatively small number of floating-point DSPs available. Among the three benchmarked floating-point DSPs, the Texas Instruments TMS320C6701 has by far the fastest total normalized execution time result, roughly a factor of two faster than the Analog Devices ADSP-21160, and approximately a factor of three faster than the older Analog Devices ADSP-21065L. The total normalized cycle count results for these processors are quite similar; hence, the execution time differences are almost entirely due to instruction cycle rate differences. The TMS320C6701 runs at 167 MHz, while the ADSP-21160 runs at 80 MHz and the ADSP-21065L runs at 66 MHz.

Note that while the floating-point DSPs implement the same benchmark algorithms as the fixed-point devices, the floating-point DSPs provide much better numeric fidelity. Therefore, comparisons between benchmark results for fixed-point processors and floating-point processors should be made with caution. We avoid such comparisons here, except to point out that, in general, floating-point DSPs have significantly slower execution time results than fixed-point DSPs with similar architectures. This is mostly due to the lower instruction cycle rates of the floating-point DSPs.

Among high-performance fixed-point DSPs, the Texas Instruments TMS320C64xx-C (with caches preloaded) has the fastest total normalized execution time result. Note, however, that this result is projected; at the time of this writing Texas Instruments had not announced any products based on the TMS320C64xx architecture. The projected execution time advantage of the TMS320C64xx-C is due to its remarkably high projected instruction cycle rate of 600 MHz; the fastest instruction cycle rate among the other processors benchmarked here is 300 MHz (for the Motorola MSC8101, which uses the SC140 core, and the Texas Instruments TMS320C6203). This factor-of-two advantage in instruction cycle rate allows the TMS320C64xx-C to offset a total normalized instruction cycle count result that is roughly 40% higher than that of the SC140. As a result, the projected total normalized execution time result for the TMS320C64xx-C is roughly 30% faster than that of the MSC8101.

The Texas Instruments TMS320C6203, with its 300 MHz instruction cycle rate, has a total normalized execution time result that is approximately 2.5 times longer (i.e., slower) than that of the TMS320C64xx-C, and roughly 1.7 times longer than the MSC8101. As a group, these three processors are much faster than any of the others benchmarked here. The Lucent DSP16410, the processor with the next-fastest total normalized execution time result, is approximately 2.5 times slower than the TMS320C6203 based on this metric. This large gap is due to a combination of the DSP16410's higher cycle counts and lower instruction cycle rate. Note, however, that the benchmark results used here for the DSP16410 use only one of the processor's two cores.

The Texas Instruments TMS320C5510, which combines attributes of a conventional DSP with attributes of a VLIW-based DSP, has a projected total normalized execution time result that lies between those of the VLIW-based DSPs (which are significantly faster than that of the TMS320C5510) and those of the conventional DSPs (which have total normalized execution time results that range from slightly slower to much slower than that of the TMS320C5510). This is not surprising, given that the TMS320C5510 has a total normalized instruction count that is about average, and a moderate 160 MHz projected clock speed.

Among the benchmarked conventional DSPs, the Texas Instruments TMS320C5416 and Motorola DSP56311 have relatively fast total normalized execution time results, roughly 10-20% lower faster than the average of all processors. Compared with other conventional DSPs, these processors achieve their relatively fast speeds through a combination of cycle counts that are slightly lower than those of other conventional DSPs, and instruction cycle rates that are the highest among this group.

The Motorola DSP568F01 has the slowest total normalized execution time result, trailing the next-slowest processor by over a factor of two. This is due to a combination of high cycle counts and a very slow instruction cycle rate of 40 MHz (with an 80 MHz clock).

Variations on individual benchmarks are often larger than variations in the overall results. For example, on the Real Block FIR benchmark, execution times for fixed-point DSPs range from a projected 0.32 μs for the TMS320C64xx-C to about 25 μs for the DSP56F801; i.e., they vary by a factor of almost 100. Such large variations in performance on individual benchmarks underscore the need to understand the details of a particular application before trying to make a determination as to which processor will provide the best performance.

Table 8.2-1. Processor Speeds and Related Information

Fastest Version of Each Processor, Lowest Price Variant
June 2000 Unit Prices, Qty. 10,000 Purchases (see note 1)

(cont.)

	ADI 2186M	ADI 219x	ADI 219x-C	ADI 21065L	ADI 21065L-C	ADI 21160	ADI 21160-C	Lucent 16410
Notes		[2,3]	[2,3]	[3]	[3]	[3]	[3]	
Instruction Cycle Time of Fastest Shipping Version, ns (= I[p])	13.3	6.3	6.3	15.2	15.2	12.5	12.5	5.9
Speed of Fastest Version, Millions of Instruction Cycles per second (= 1000/I[p])	75.0	160.0	160.0	66.0	66.0	80.0	80.0	170.0
Cost of Processor (Least Expensive at Fastest Speed) (=D[p])	$8.50	N/A	N/A	$25.00	$25.00	$99.00	$99.00	N/A

Notes:

[1] The instruction cycle times shown here are for the fastest version of each processor available in sample quantities as of June 2000, according to the manufacturer. Unit prices are for quantity 10,000 purchases, also as reported by the manufacturer. BDTI has not independently verified this information. In addition, readers should be aware that device speeds are constantly increasing, and prices for existing processor versions are constantly decreasing. Therefore, we strongly recommend contacting manufacturers for updated information about prices and maximum device speeds.

[2] Projected; as of this writing, these processors were not yet sampling or have not been demonstrated in silicon at their target speed.

[3] These processors use on-chip caches. Results under processor names denoted with "-C" represent performance with caches preloaded. Results for the processors listed under processor names without "-C" represent performance with caches uninitialized; see text for details.

Table 8.2-1. (cont.)

Motorola 56311	Motorola 56F801	Motorola 56853	Motorola MSC8101	TI C5416	TI C5510	TI C6203	TI C64xx	TI C64xx-C	TI C6701
		[2]			[2]		[2,3]	[2,3]	
6.7	25.0	8.3	3.3	6.3	6.3	3.3	1.7	1.7	6.0
150.0	40.0	120.0	300.0	160.0	160.0	300.0	600.0	600.0	167.0
$47.70	$8.15	$3.75	$96.00	$33.50	$29.00	$201.42	N/A	N/A	$139.06

Table 8.2-2. Benchmark Execution Times

Fastest Version of Each Processor, Lowest Price Variant
T[p,b] = C[p,b] * I[p] / 1000, microseconds

(cont.)

DSP Building Block Functions	ADI 2186M	ADI 219x	ADI 219x-C	ADI 21065L	ADI 21065L-C	ADI 21160	ADI 21160-C	Lucent 16410
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	[3]
Real Block FIR	11.40	4.98	4.87	12.38	12.30	7.33	7.23	2.84
Single-Sample FIR	0.40	0.28	0.25	0.47	0.39	0.36	0.30	0.13
Complex Block FIR	41.79	21.68	21.60	42.17	42.03	19.38	19.30	9.94
LMS Adaptive FIR	0.88	0.54	0.49	0.97	0.83	0.80	0.69	0.33
Two-Biquad IIR	0.28	0.16	0.14	0.35	0.26	0.28	0.20	0.09
Vector Dot Product	0.60	0.36	0.35	0.77	0.70	0.44	0.38	0.15
Vector Add	1.15	0.61	0.59	1.33	1.30	0.61	0.59	0.28
Vector Maximum	1.80	1.06	1.06	1.86	1.86	0.64	0.64	0.19
Control	5.87	3.86	3.86	7.47	7.47	6.16	6.16	3.34
256-Point FFT	135.23	63.52	63.45	57.06	56.98	32.78	32.71	41.87
Viterbi	409.69	192.11	192.11	221.52	221.52	142.53	142.53	55.05
Bit Unpack	5.05	2.88	2.88	4.09	4.09	3.38	3.38	2.86

Notes:

[1] These processors use on-chip caches. Results under processor names denoted with "-C" represent performance with caches preloaded. Results for the processors listed under processor names without "-C" represent performance with caches uninitialized; see text for details.

[2] Projected; as of this writing, these processors were not yet sampling or have not been demonstrated in silicon at their target speed.

[3] Results are for one of the two on-chip cores.

[4] Results are estimated based on preliminary cache information provided by Texas Instruments; see text.

Table 8.2-2. (cont.)

Motorola 56311	Motorola 56F801	Motorola 56853	Motorola MSC8101	TI C5416	TI C5510	TI C6203	TI C64xx	TI C64xx-C	TI C6701
		[2]			[2]		[1,2,4]	[1,2]	
6.29	24.58	7.19	0.61	4.56	2.46	1.16	0.47	0.32	2.89
0.23	0.75	0.24	0.05	0.19	0.16	0.08	0.08	0.04	0.26
19.10	78.58	23.88	2.24	18.38	8.81	4.36	1.25	1.13	11.22
0.55	1.70	0.58	0.06	0.46	0.39	0.14	0.11	0.06	0.35
0.13	0.50	0.18	0.03	0.11	0.11	0.05	0.05	0.03	0.18
0.35	1.23	0.39	0.05	0.28	0.30	0.10	0.07	0.04	0.28
0.62	2.28	0.76	0.06	0.47	0.30	0.13	0.08	0.05	0.30
0.59	2.23	1.01	0.09	0.78	0.44	0.12	0.09	0.06	0.26
5.60	16.58	6.05	1.48	5.39	5.91	1.60	0.91	0.91	2.87
58.83	371.50	97.57	5.44	65.04	46.16	8.31	2.41	2.08	21.30
120.19	899.58	237.78	6.46	70.40	42.98	21.45	3.05	2.90	31.86
2.65	11.95	3.78	0.43	3.65	1.56	0.44	0.21	0.16	0.79

Table 8.2-3. Normalized Execution Times

Fastest Version of Each Processor, Lowest Price Variant
T[p,b] / AT[b]

DSP Building Block Functions	**ADI 2186M**	**ADI 219x**	**ADI 219x-C**	**ADI 21065L**	**ADI 21065L-C**	**ADI 21160**	**ADI 21160-C**	**Lucent 16410**
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	[3]
Real Block FIR	1.80	0.79	0.77	1.96	1.95	1.16	1.14	0.45
Single-Sample FIR	1.55	1.06	0.97	1.82	1.52	1.40	1.16	0.50
Complex Block FIR	1.94	1.01	1.01	1.96	1.96	0.90	0.90	0.46
LMS Adaptive FIR	1.59	0.98	0.89	1.75	1.51	1.45	1.24	0.60
Two-Biquad IIR	1.62	0.91	0.80	2.02	1.49	1.59	1.16	0.51
Vector Dot Product	1.58	0.95	0.92	2.03	1.84	1.15	0.99	0.40
Vector Add	1.79	0.95	0.93	2.09	2.04	0.96	0.92	0.44
Vector Maximum	2.19	1.29	1.29	2.27	2.27	0.78	0.78	0.24
Control	1.15	0.76	0.76	1.47	1.47	1.21	1.21	0.66
256-Point FFT	2.09	0.98	0.98	0.88	0.88	0.51	0.51	0.65
Viterbi	2.45	1.15	1.15	1.32	1.32	0.85	0.85	0.33
Bit Unpack	1.68	0.95	0.95	1.36	1.36	1.12	1.12	0.95
Total	21.44	11.79	11.42	20.93	19.60	13.08	11.98	6.18

Notes:

[1] These processors use on-chip caches. Results under processor names denoted with "-C" represent performance with caches preloaded. Results for the processors listed under processor names without "-C" represent performance with caches uninitialized; see text for details.

[2] Projected; as of this writing, these processors were not yet sampling or have not been demonstrated in silicon at their target speed.

[3] Results are for one of the two on-chip cores.

[4] Results are estimated based on preliminary cache information provided by Texas Instruments; see text.

Table 8.2-3. (cont.)

Motorola 56311	Motorola 56F801	Motorola 56853	Motorola MSC8101	TI C5416	TI C5510	TI C6203	TI C64xx	TI C64xx-C	TI C6701
		[2]			[2]		[1,2,4]	[1,2]	
1.00	3.89	1.14	0.10	0.72	0.39	0.18	0.07	0.05	0.46
0.88	2.90	0.93	0.18	0.75	0.60	0.32	0.30	0.16	1.00
0.89	3.66	1.11	0.10	0.86	0.41	0.20	0.06	0.05	0.52
1.00	3.07	1.05	0.11	0.84	0.71	0.25	0.20	0.11	0.64
0.77	2.90	1.06	0.15	0.62	0.62	0.31	0.26	0.17	1.04
0.91	3.23	1.03	0.13	0.74	0.79	0.27	0.19	0.11	0.73
0.97	3.56	1.19	0.10	0.73	0.47	0.21	0.12	0.07	0.47
0.72	2.71	1.23	0.11	0.95	0.54	0.15	0.11	0.07	0.32
1.10	3.26	1.19	0.29	1.06	1.16	0.31	0.18	0.18	0.56
0.91	5.75	1.51	0.08	1.01	0.71	0.13	0.04	0.03	0.33
0.72	5.37	1.42	0.04	0.42	0.26	0.13	0.02	0.02	0.19
0.88	3.97	1.25	0.14	1.21	0.52	0.15	0.07	0.05	0.26
10.75	44.26	14.12	1.54	9.90	7.18	2.61	1.62	1.09	6.52

Figure 8.2-1. Execution Time for Real Block FIR (Lower is Better)

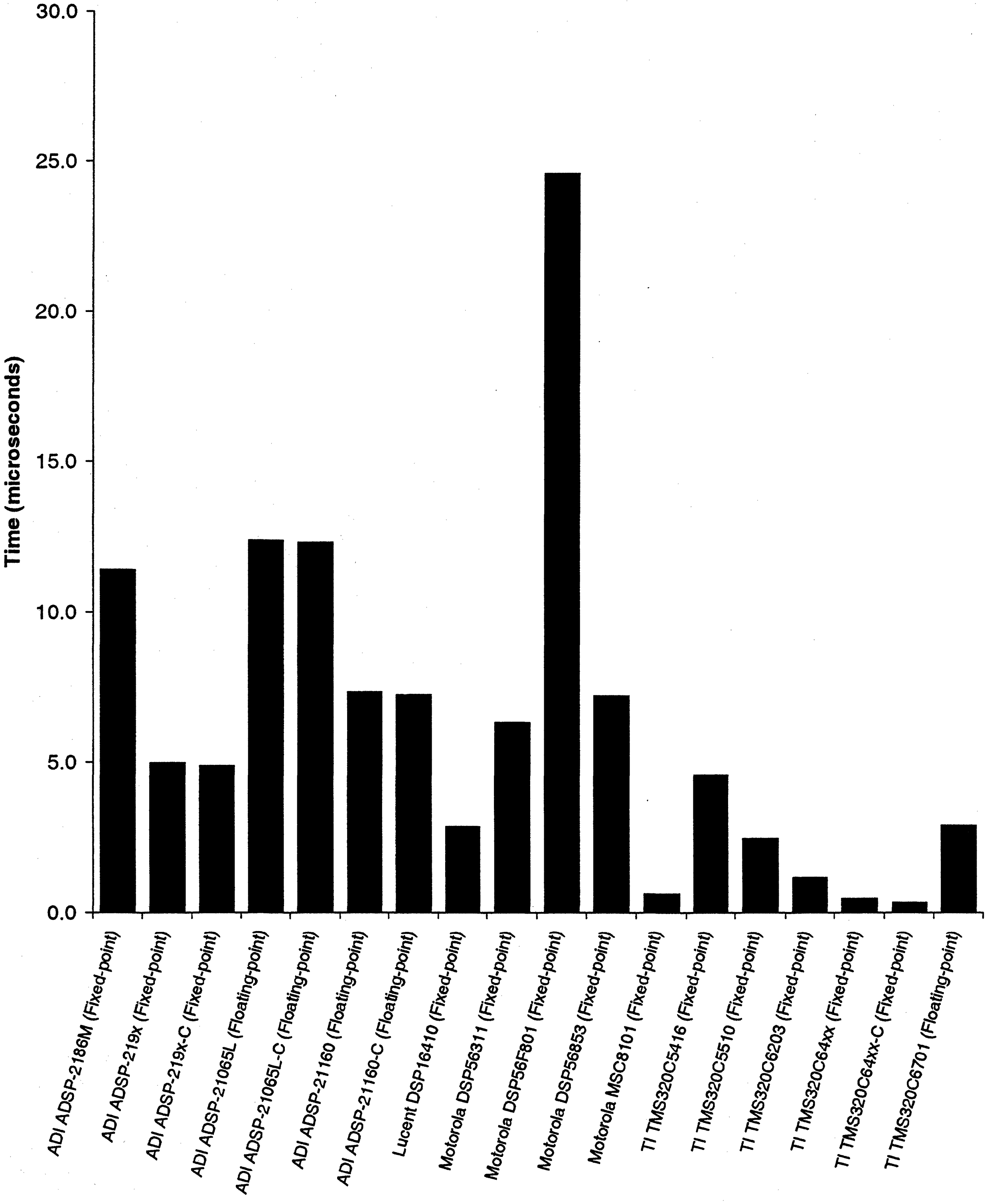


Figure 8.2-2. Execution Time for Single-Sample FIR (Lower is Better)

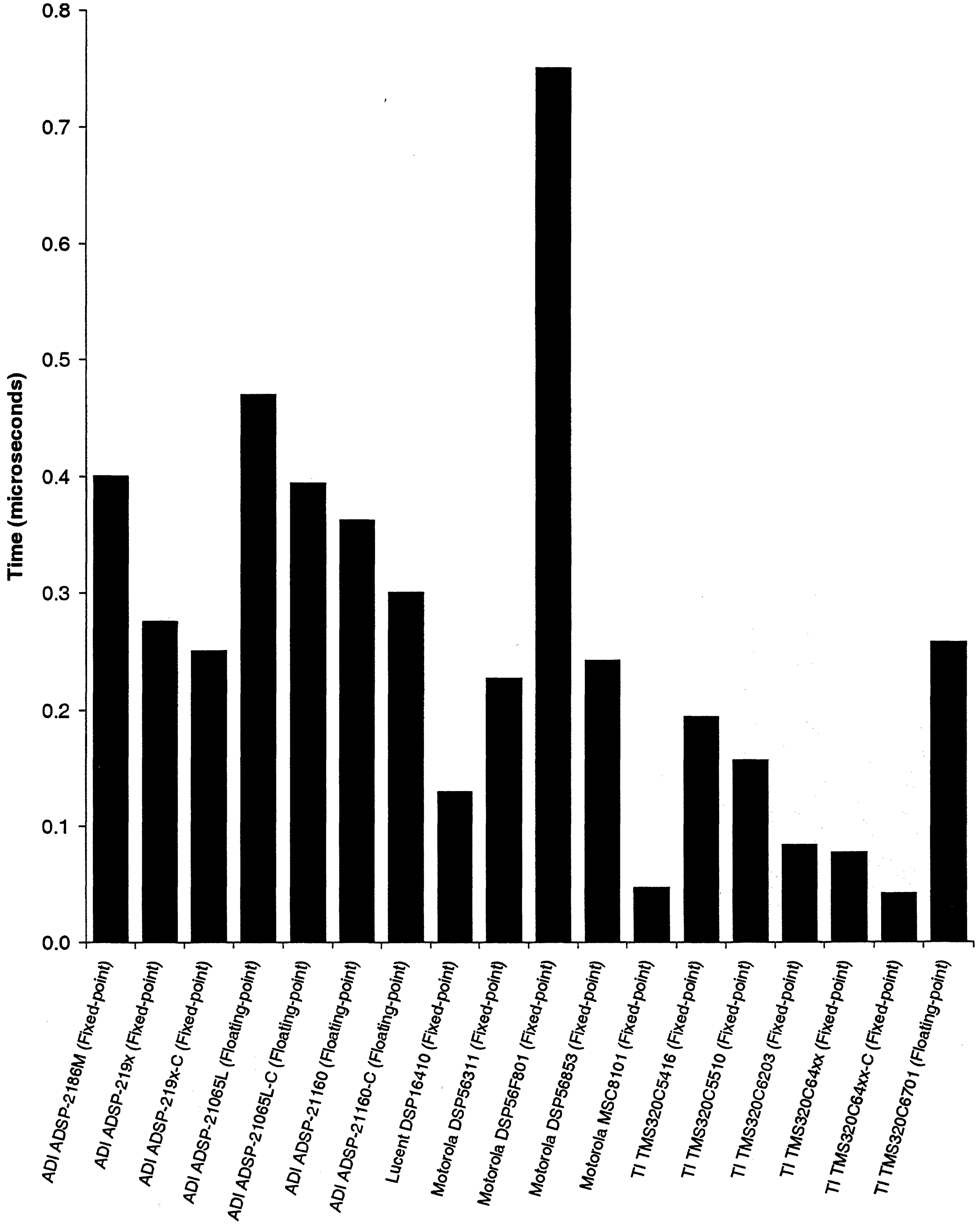


Figure 8.2-3. Execution Time for Complex Block FIR (Lower is Better)

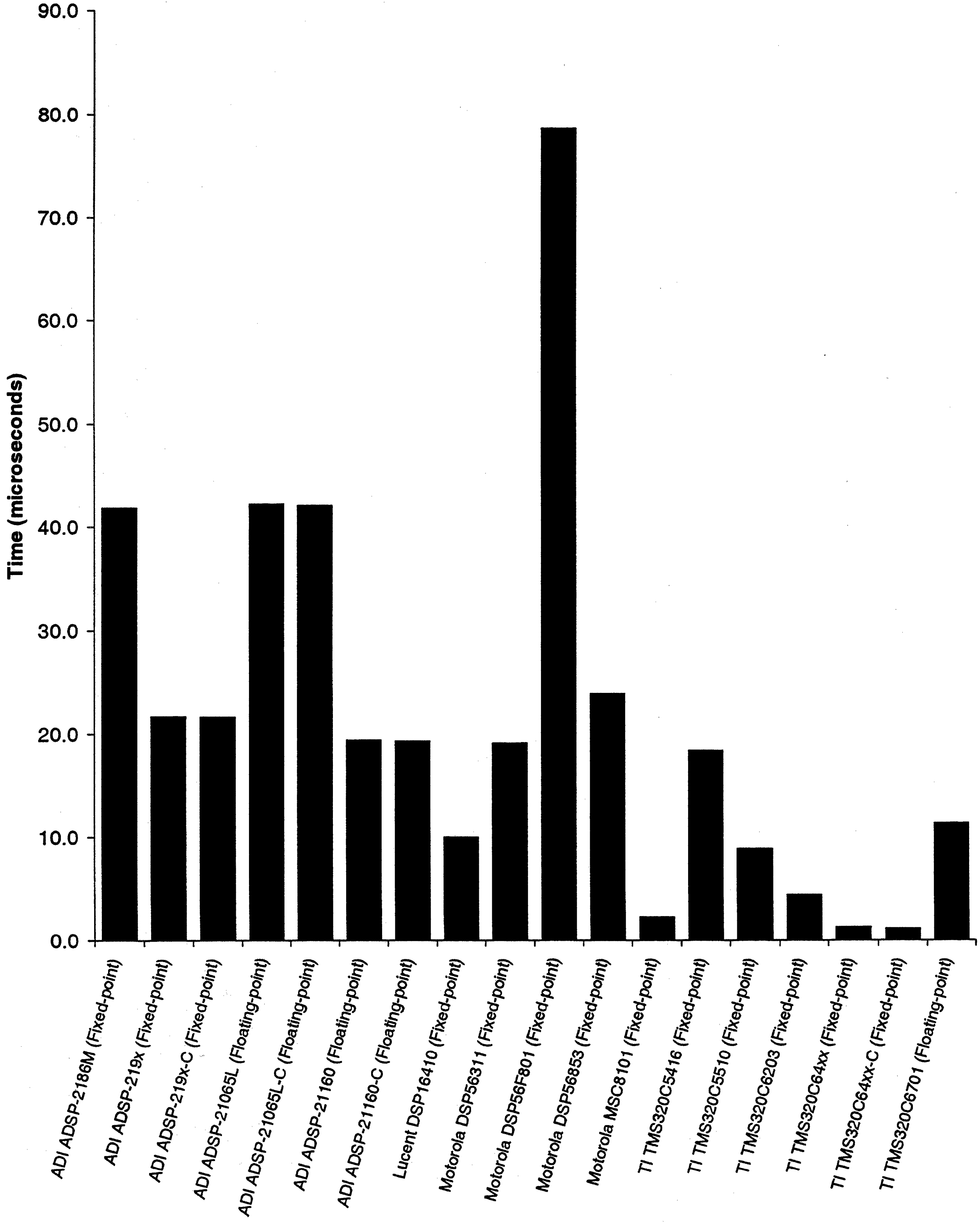


Figure 8.2-4. Execution Time for LMS Adaptive FIR (Lower is Better)

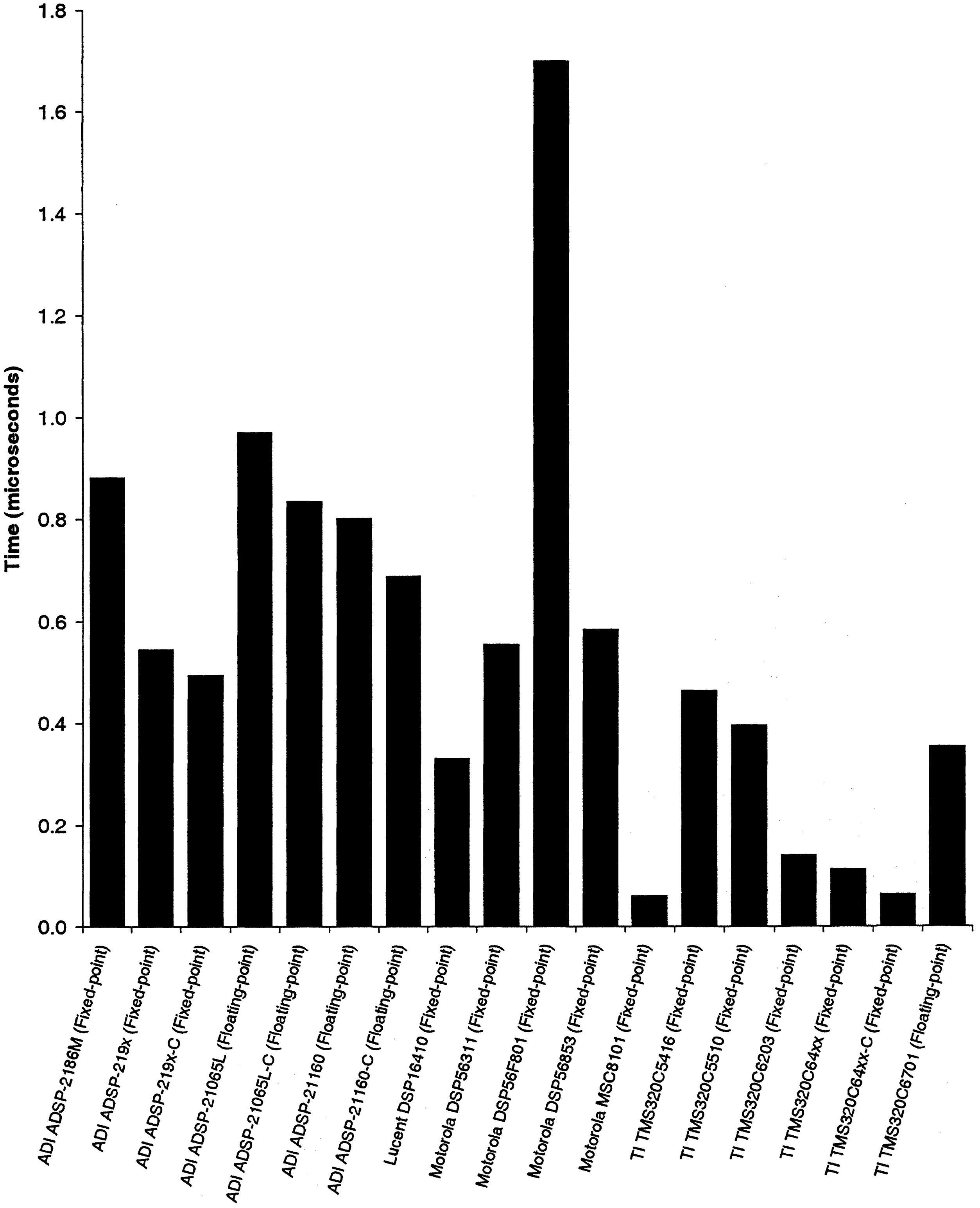


Figure 8.2-5. Execution Time for Two-Biquad IIR (Lower is Better)

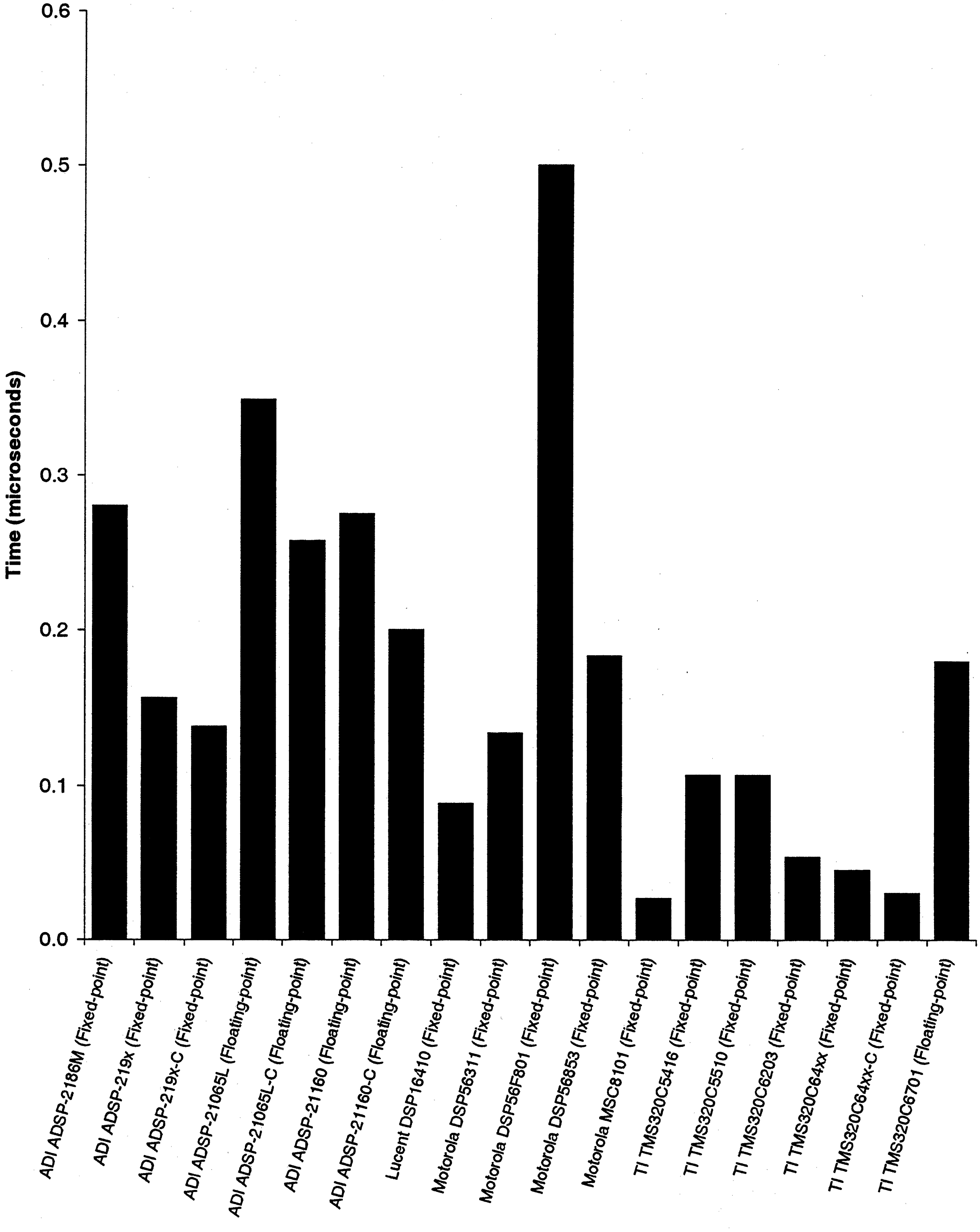


Figure 8.2-6. Execution Time for Vector Dot Product (Lower is Better)

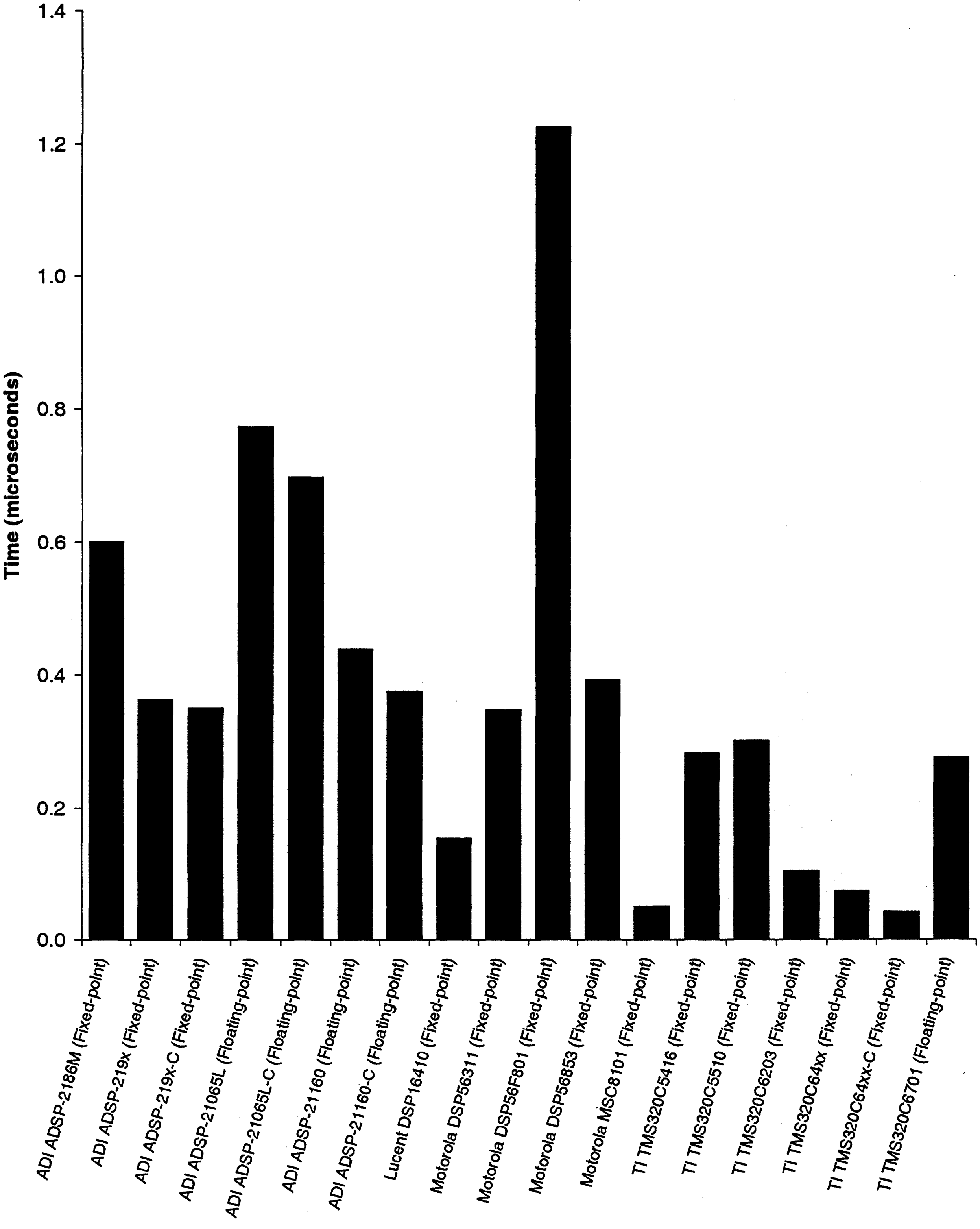


Figure 8.2-7. Execution Time for Vector Add (Lower is Better)

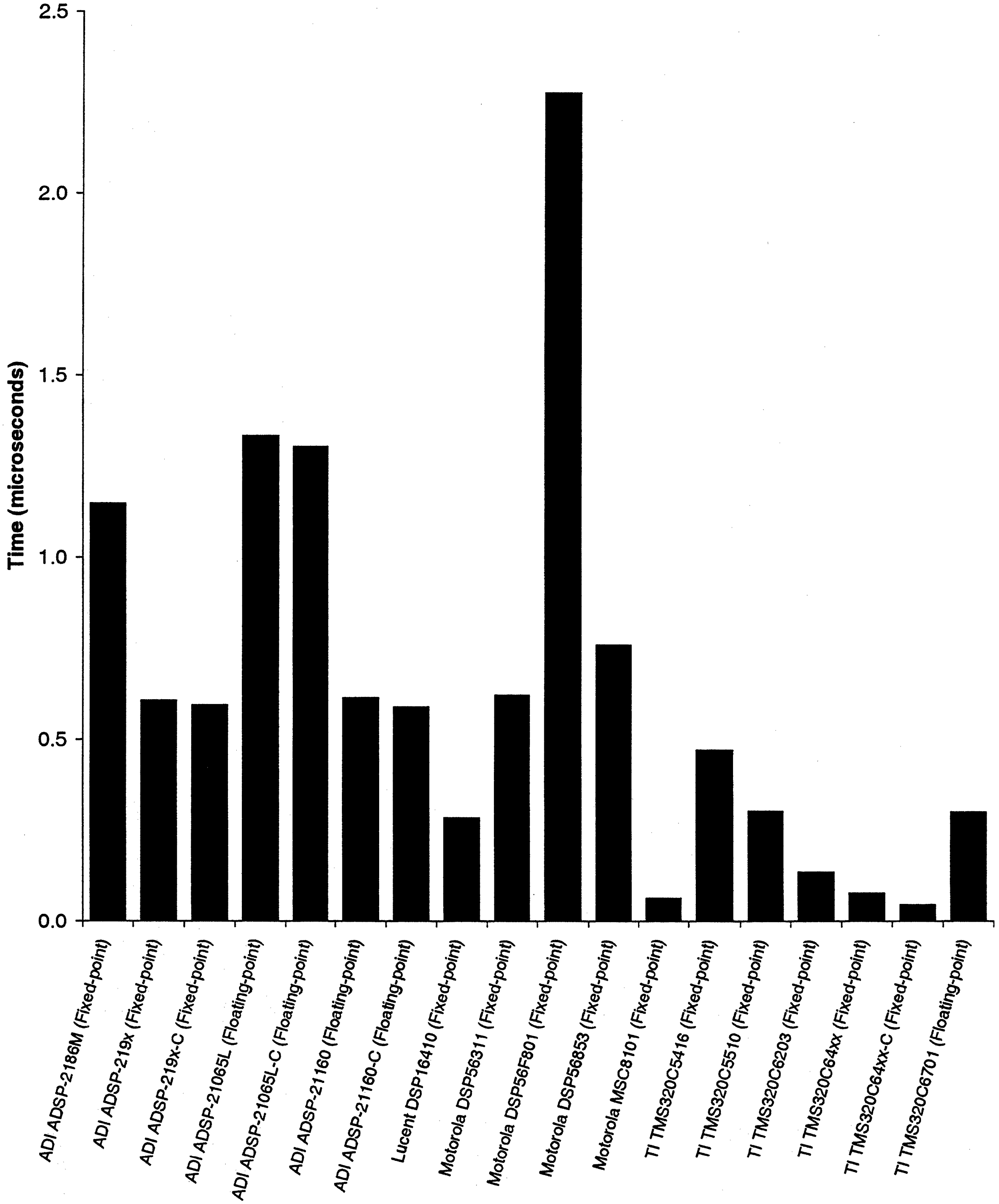


Figure 8.2-8. Execution Time for Vector Maximum
(Lower is Better)

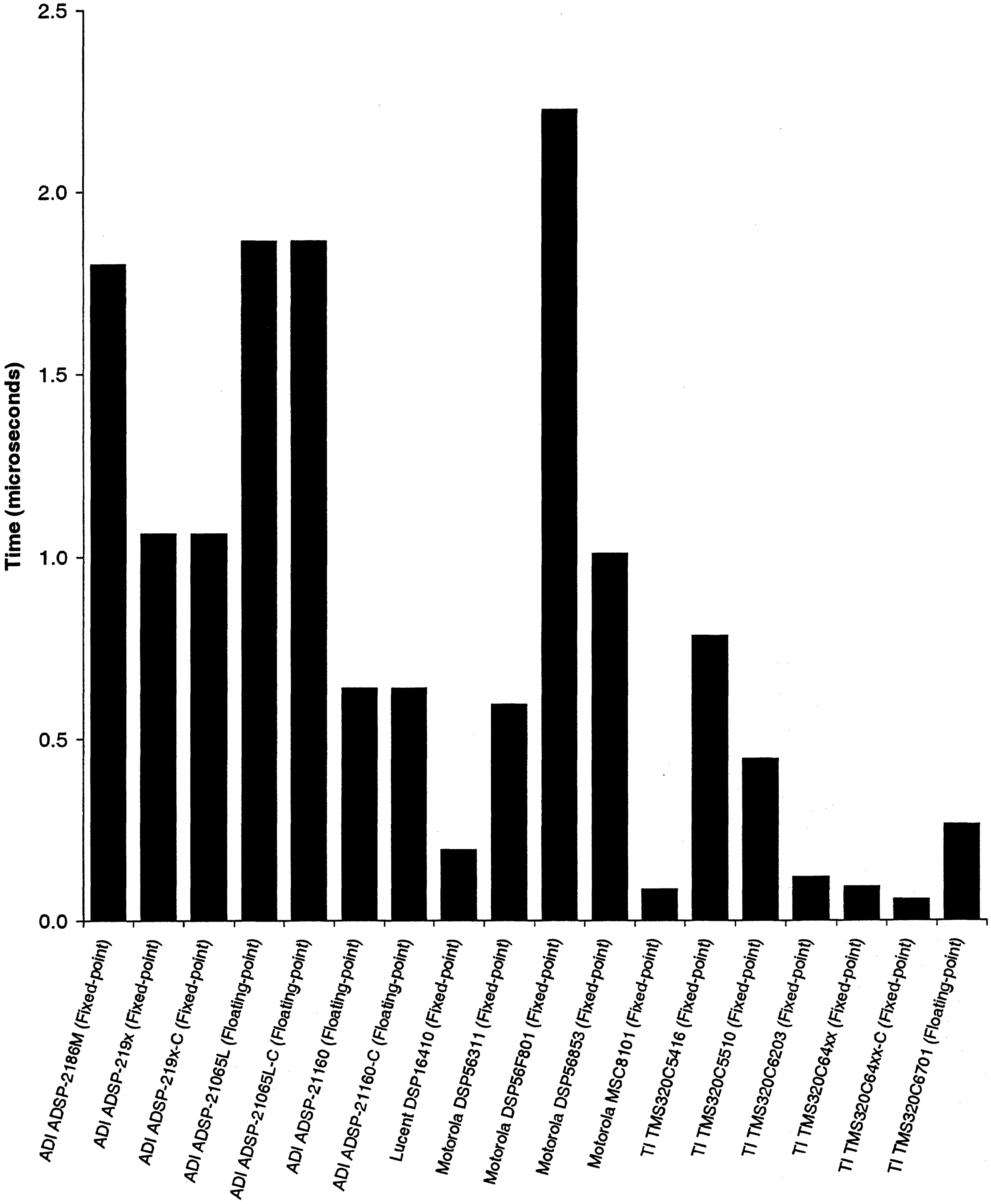


Figure 8.2-9. Execution Time for Control (Lower is Better)

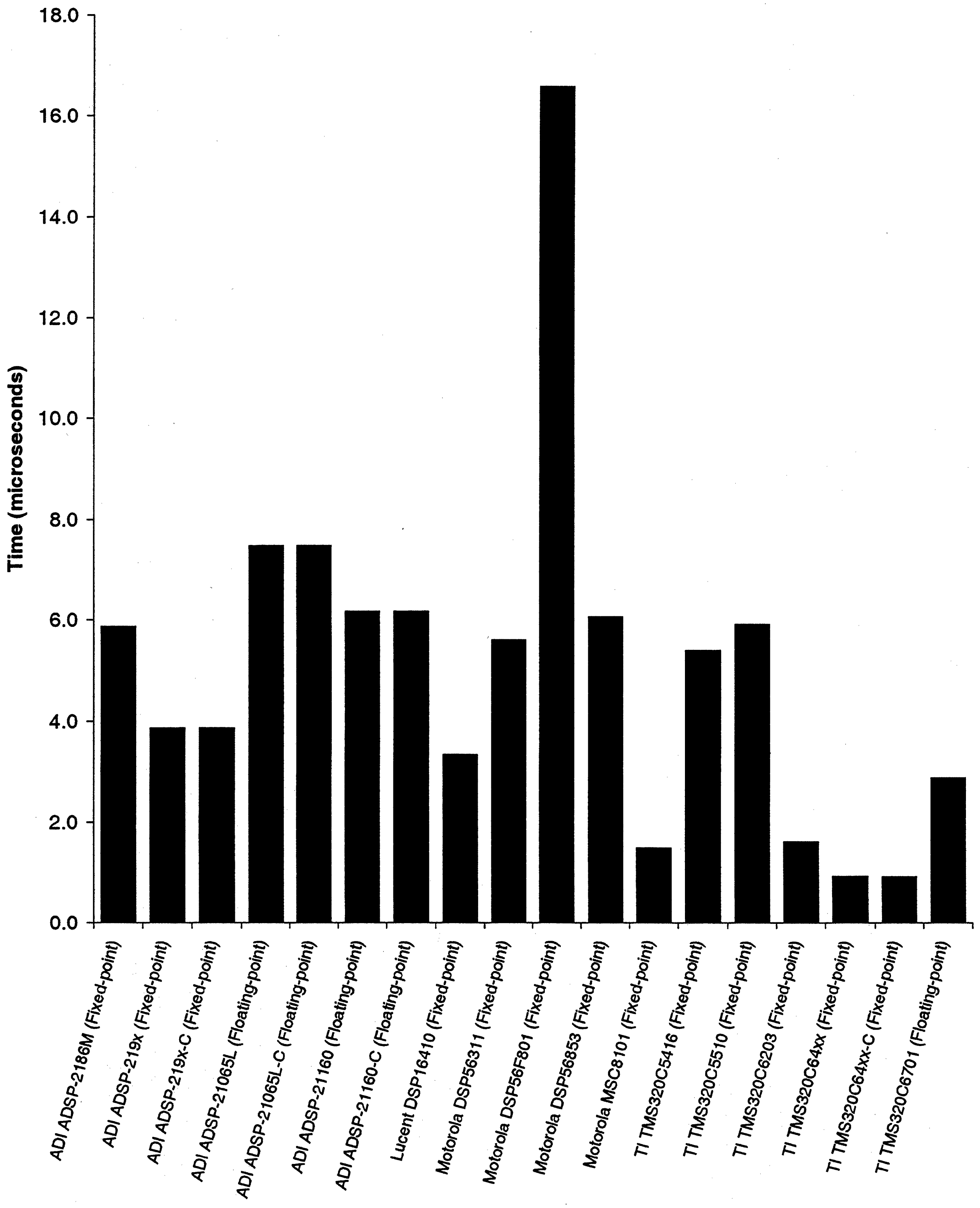


Figure 8.2-10. Execution Time for 256-Point FFT
(Lower is Better)

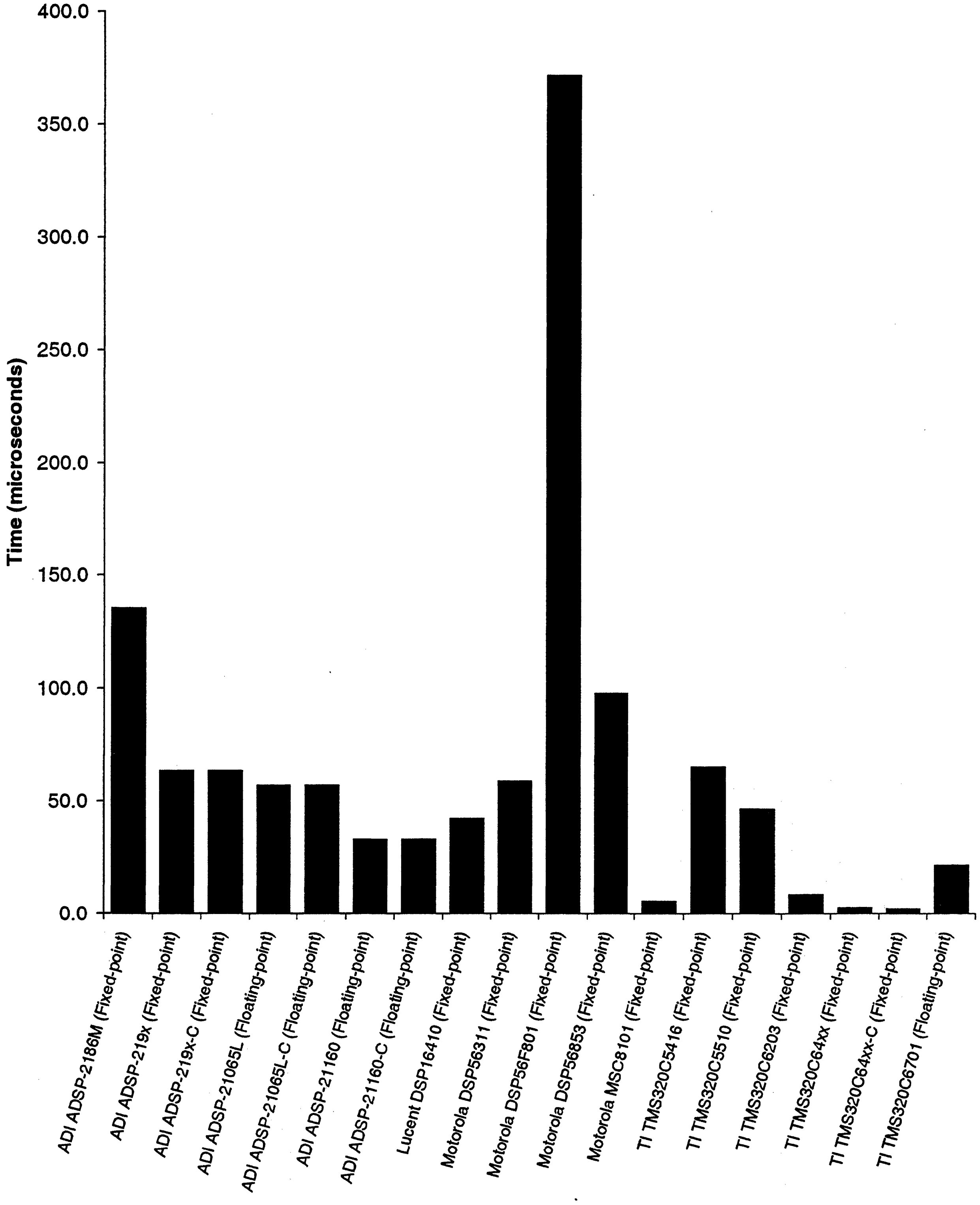


Figure 8.2-11. Execution Time for Viterbi (Lower is Better)

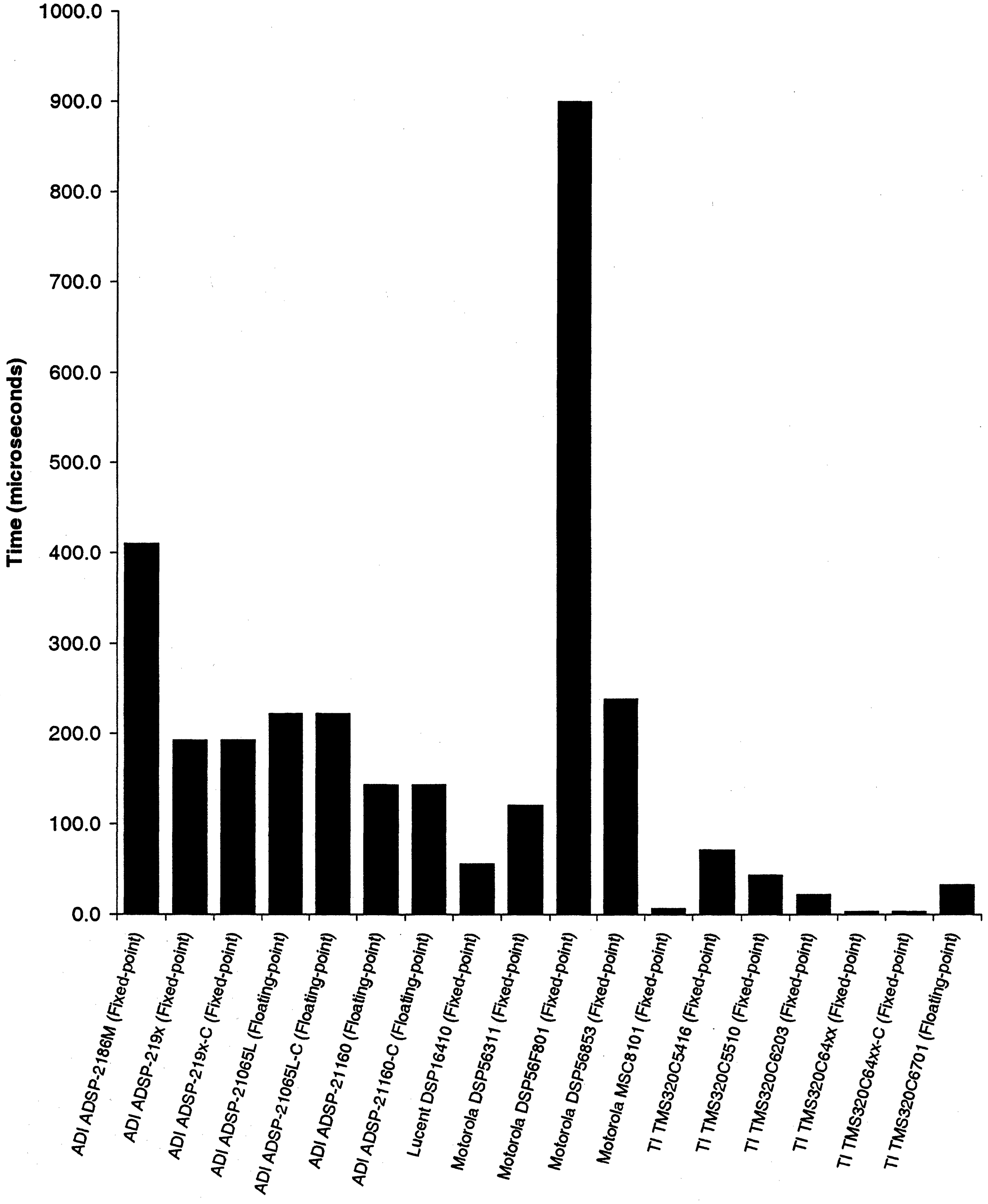


Figure 8.2-12. Execution Time for Bit Unpack
(Lower is Better)

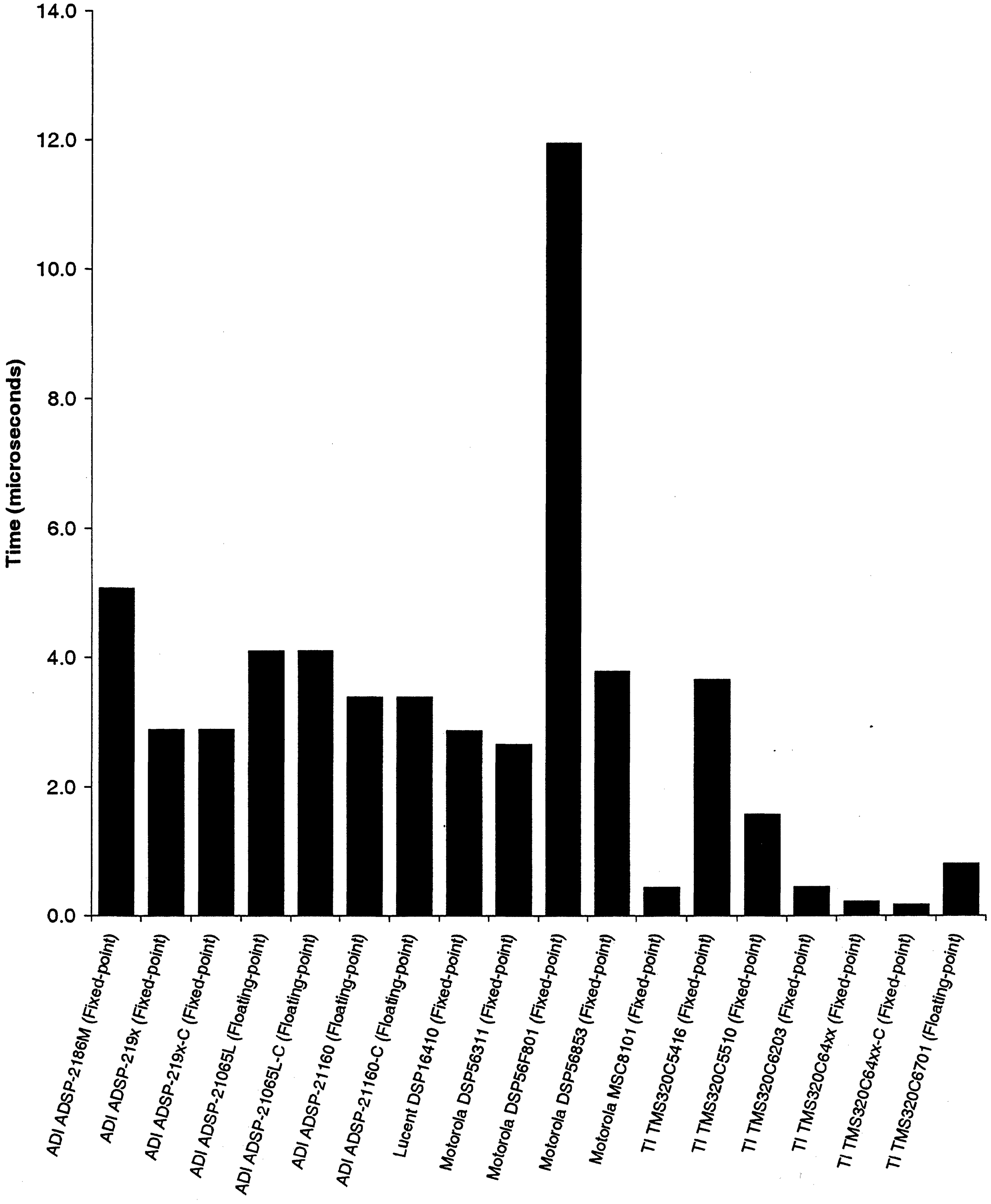


Figure 8.2-13. Normalized Benchmark Execution Times Sum for all b of T[p,b] / AT[b] (Lower is Better)

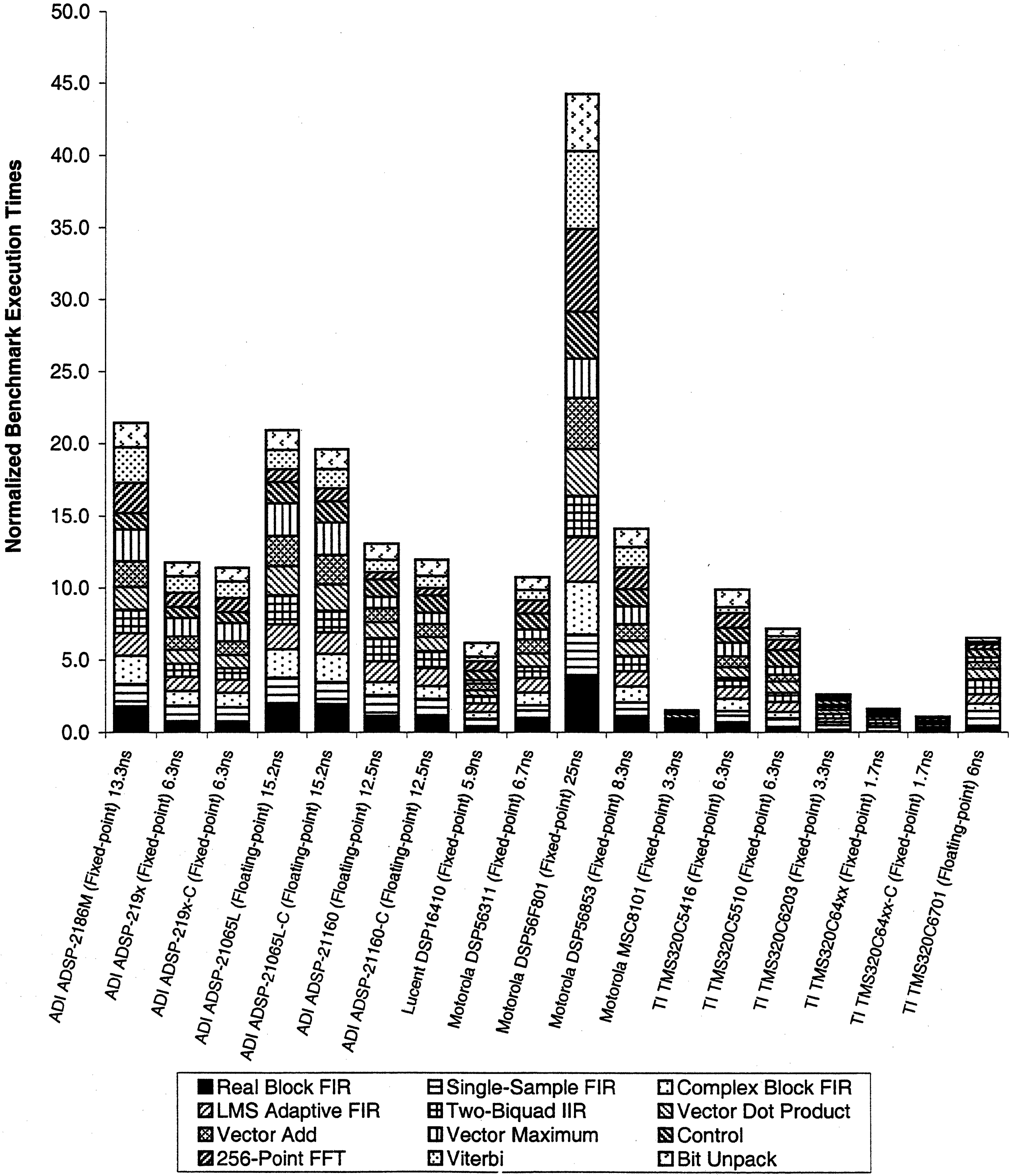
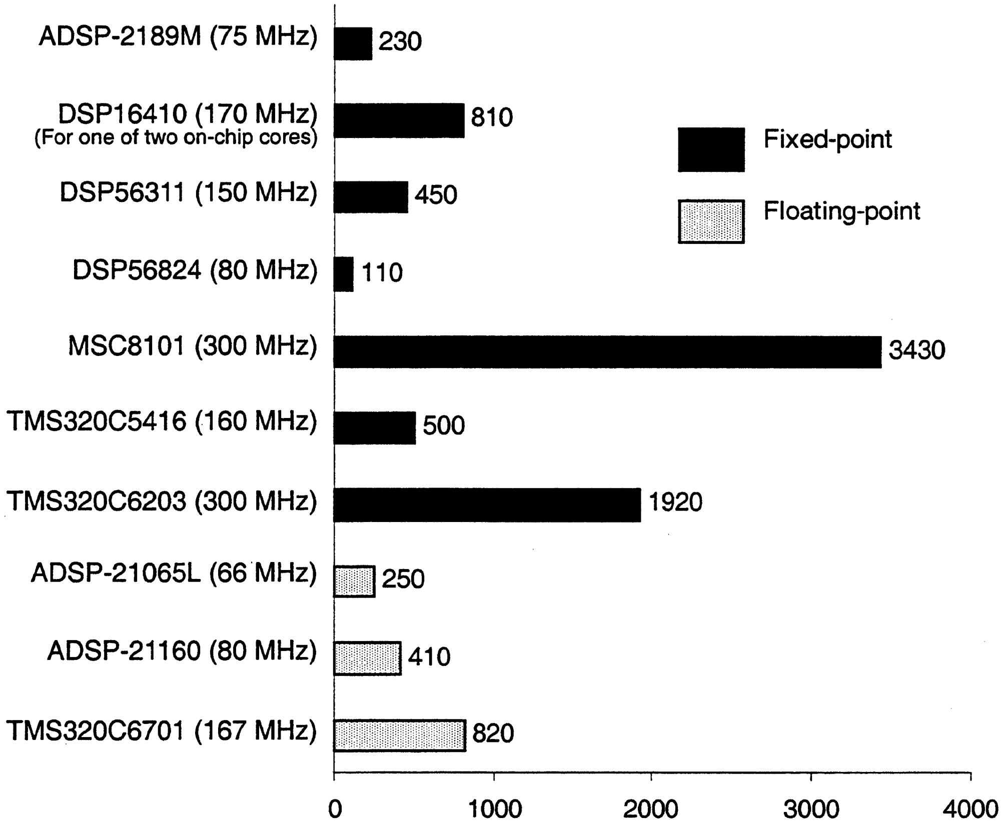


Figure 8.2-14. BDTImark2000 Scores



8.3 Cost-Execution Time Product

In this section, we combine processor execution time results from the previous section with processor cost information to create a cost-performance measure. Specifically, we multiply the benchmark execution times for each processor on each benchmark by the unit cost of the processor, yielding a cost-execution time product. The lower the cost-execution time product, the better the processor's score on this particular figure of merit. We apply equal weighting to cost and execution time in our analysis. *Depending upon the requirements of a given application, different weightings may be appropriate.*

> Since this analysis is based upon execution times from the previous section, the results apply to the same member of each processor family as was used in that section. In particular, this analysis uses prices and speeds from Table 8.2-1. Prices are based on the least expensive packaging option of the fastest version of each processor. In many cases, vendors offer other members of their processor families with reduced performance and reduced cost. Such versions may fare significantly better or worse on the cost-execution time metric compared to the versions used here. BDTI's Benchmark Analysis Tool allows readers to repeat the analysis shown here with different processor family members.

The unit cost data used in this analysis was provided by processor manufacturers or their distributors for June 2000, and is based on orders of 10,000 units. Bear in mind that DSP processor prices change frequently, tending to decrease significantly over time, and are strongly dependent upon the volume purchased. *Therefore, we urge readers to check with vendors to obtain updated pricing information when comparing processors.*

Several of the processors included in earlier sections of our analysis are excluded from this section. Since our analysis of the Lucent DSP164xx single-chip multiprocessor uses only one of the device's processors, it would not be reasonable to include the DSP164xx in this section of the analysis, and it is omitted. At the time of this writing, Analog Devices had not announced a product based on the ASDP-219x architecture, and Texas Instruments had not announced a product based on the TMS320C64xx architecture, so these processors are excluded from our analysis in this section. Pricing for the Motorola DSP56853 is projected, as this product was not available at the time of this writing.

The data and analysis presented in this section are organized as follows:

- **Cost-execution time products for each benchmark.**
  Table 8.3-1 and Figures 8.3-1 through 8.3-12 show the cost-execution time products for each processor on each benchmark. These results are based on the execution times shown in the previous section.

- **Normalized cost-execution time products.**
  As the first step in creating aggregate cost-execution time results that show each processor's overall cost-execution time result, we normalize the cost-execution time product for each processor on each benchmark. This normalization is done by dividing each processor's cost-execution time product on a given benchmark by the average over all processors on that benchmark. This is done so that benchmarks that are inherently more time-consuming are not automatically weighted more heavily when we aggregate the performance of each processor over all benchmarks. Normalized cost-execution time products are shown in Table 8.3-2.
- **Total normalized cost-execution time product over all benchmarks.**
  The result of adding the normalized cost-execution time products over all benchmarks for each processor produces an overall cost-execution time measure for that processor. This sum effectively applies a uniform weighting to each benchmark for this section of the analysis. The results are shown in Figure 8.3-13 and at the bottom of Table 8.3-2.

Analysis of Results

This section presents a brief analysis of the benchmark cost-execution time products, summarized in Figure 8.3-13. As discussed in the preceding paragraphs, the benchmark cost-execution time product is computed by multiplying a processor's execution time for a given benchmark function by the processor's cost. Where noted, these values are projected for processors not available at the time of this writing. Bear in mind that different cost values and different processor family members may be appropriate for your analysis.

In reviewing the normalized cost-execution time products over all benchmarks presented in Figure 8.3-13, several conclusions emerge. First, floating-point processors generally have the highest cost-execution time result, meaning that they are expensive relative to their speed when compared with fixed-point processors. For example, the Analog Devices ADSP-21160 and Texas Instruments TMS320C6701 have the highest total normalized cost-execution time results of the processors benchmarked here. The Analog Devices ADSP-21065L, however, has a total normalized cost-execution time result comparable to those of several fixed-point DSPs.

It must be noted that while the floating-point DSPs implement the same algorithms as the fixed-point devices, they provide much better numeric fidelity. In addition, floating-point processors sometimes have on-chip peripherals and memory beyond what is typical in fixed-point DSPs. Therefore, comparisons between benchmark results for fixed-point processors and floating-point processors should be made with caution.

Turning to fixed-point processors, the Motorola DSP56853 has the lowest total normalized cost-execution time result by a large margin; it should be noted, however, that this result is based on projected pricing, as the DSP56853 was not available at the time of this writing. As presented in Section 8.2, *Execution Times*, the DSP56853 has average speed as measured by its total normalized execution time. Hence, its advantage on the cost-execution time is due to its price, which is a remarkably low $3.75. Similarly, the Analog Devices ADSP-2186M has a very low price of $8.50, and this overcomes its rather slow execution time result to yield a fairly low total normalized cost-execution time result.

The Motorola MSC8101, based on the StarCore SC140, also has a relatively low total normalized cost-execution time result. The price of this device is moderately high, at $96.00, but this disadvantage is overcome by the processor's very fast execution time results.

Texas Instruments' projected $29.00 pricing for the TMS320C5510 is aggressive compared to the current $33.50 price of its lower-performance TMS320C5416. Since on-chip memory is often the primary determinant of silicon area, and the two devices have similar amounts of on-chip memory, we would expect similar prices. Still, it is surprising to see a lower price for the higher performance product. Not surprisingly given these prices, the TMS320C5510 bests the TMS320C5416 in terms of total normalized cost-execution time product by a significant margin.

The total normalized cost-execution time result for the Texas Instruments TMS320C6203 is roughly equal to the average for the processors benchmarked here. It should be noted, though, that there are members of the TMS320C62xx family with much lower prices than the TMS320C6203. These devices have lower instruction cycle rates and less on-chip memory, and in some cases use on-chip caches. This highlights the fact, mentioned earlier, that the cost-execution time products used in this section are based on the fastest version of each processor available as of this writing (or projected for processors not yet available), in the least expensive packaging variant. In most cases, manufacturers offer different versions of their processors at lower speeds and lower prices. Since speed and price do not scale proportionally, choosing a lower-speed version of a processor would in most cases result in a different benchmark cost-execution time product for that processor.

Note also that the choice of processor affects overall system cost. For example, a processor with low cost-execution time products may not have enough on-chip memory for a given application, requiring external memory to be used in the system. Overall system cost may thus be higher than if another processor with more on-chip memory is chosen, although the latter processor may have higher cost-execution time products. We do not take these issues into account in our analysis.

Table 8.3-1. Benchmark Cost-Execution Time Product

Fastest Version of Each Processor, Lowest Price Variant (see note 5)
Benchmark Execution Time * Processor Cost, microsecond-dollars, DT[p,b] = D[p] * T[p,b]

(cont.)

DSP Building Block Functions	ADI 2186M	ADI 219x	ADI 219x-C	ADI 21065L	ADI 21065L-C	ADI 21160	ADI 21160-C	Lucent 16410
Notes		[1]	[1]	[2]	[2]	[2]	[2]	[3]
Real Block FIR	96.9	N/A	N/A	309.5	307.6	725.2	715.3	N/A
Single-Sample FIR	3.4	N/A	N/A	11.7	9.8	35.9	29.7	N/A
Complex Block FIR	355.2	N/A	N/A	1054.2	1050.8	1918.1	1910.7	N/A
LMS Adaptive FIR	7.5	N/A	N/A	24.2	20.8	79.2	68.1	N/A
Two-Biquad IIR	2.4	N/A	N/A	8.7	6.4	27.2	19.8	N/A
Vector Dot Product	5.1	N/A	N/A	19.3	17.4	43.3	37.1	N/A
Vector Add	9.7	N/A	N/A	33.3	32.6	60.6	58.2	N/A
Vector Maximum	15.3	N/A	N/A	46.6	46.6	63.1	63.1	N/A
Control	49.9	N/A	N/A	186.7	186.7	610.1	610.1	N/A
256-Point FFT	1149.4	N/A	N/A	1426.5	1424.6	3244.7	3238.5	N/A
Viterbi	3482.4	N/A	N/A	5537.9	5537.9	14110.0	14110.0	N/A
Bit Unpack	43.0	N/A	N/A	102.3	102.3	334.1	334.1	N/A

Notes:

[1] As of this writing, pricing has not been disclosed for these processors. Hence, they are excluded from this metric.

[2] These processors use on-chip caches. Results under processor names denoted with "-C" represent performance with caches preloaded. Results for the processors listed under processor names without "-C" represent performance with caches uninitialized; see text for details.

[3] This chip contains two cores. Therefore, comparing its cost-execution time result to those of the other processors is not meaningful.

[4] Projected; as of this writing, these processors were not yet sampling or have not been demonstrated in silicon at their target speed.

[5] Please see Table 8.2-1 for information about processors used in this section.

Table 8.3-1. (cont.)

Motorola 56311	Motorola 56F801	Motorola 56853	Motorola MSC8101	TI C5416	TI C5510	TI C6203	TI C64xx	TI C64xx-C	TI C6701
		[4]			[4]		[1]	[1]	
300.2	200.3	27.0	58.6	152.8	71.2	233.2	N/A	N/A	402.2
10.8	6.1	0.9	4.5	6.5	4.5	16.8	N/A	N/A	35.8
911.1	640.4	89.5	215.0	615.8	255.6	875.7	N/A	N/A	1560.5
26.4	13.9	2.2	5.8	15.5	11.4	28.1	N/A	N/A	49.1
6.4	4.1	0.7	2.6	3.6	3.1	10.7	N/A	N/A	25.0
16.5	10.0	1.5	4.8	9.4	8.7	20.8	N/A	N/A	38.3
29.6	18.5	2.8	6.1	15.7	8.7	26.8	N/A	N/A	41.6
28.3	18.1	3.8	8.3	26.2	12.9	24.1	N/A	N/A	36.6
267.1	135.1	22.7	141.8	180.5	171.3	320.9	N/A	N/A	398.9
2806.0	3027.7	365.9	522.2	2178.8	1338.7	1669.6	N/A	N/A	2961.9
5733.2	7331.5	891.7	620.2	2358.4	1246.5	4312.1	N/A	N/A	4429.9
126.6	97.4	14.2	41.3	122.3	45.1	88.4	N/A	N/A	109.9

Table 8.3-2. Normalized Cost-Execution Time Product

Fastest Version of Each Processor, Lowest Price Variant (see note 5)
Cost-Execution Time Product / Average for Benchmark, DT[p,b] / ADT[b]

(cont.)

DSP Building Block Functions	ADI 2186M	ADI 219x	ADI 219x-C	ADI 21065L	ADI 21065L-C	ADI 21160	ADI 21160-C	Lucent 16410
Notes		[1]	[1]	[2]	[2]	[2]	[2]	[3]
Real Block FIR	0.35	N/A	N/A	1.12	1.11	2.62	2.58	N/A
Single-Sample FIR	0.25	N/A	N/A	0.87	0.73	2.64	2.19	N/A
Complex Block FIR	0.40	N/A	N/A	1.20	1.19	2.18	2.17	N/A
LMS Adaptive FIR	0.28	N/A	N/A	0.89	0.77	2.92	2.51	N/A
Two-Biquad IIR	0.26	N/A	N/A	0.94	0.69	2.94	2.13	N/A
Vector Dot Product	0.29	N/A	N/A	1.08	0.98	2.42	2.08	N/A
Vector Add	0.37	N/A	N/A	1.26	1.23	2.29	2.20	N/A
Vector Maximum	0.51	N/A	N/A	1.54	1.54	2.09	2.09	N/A
Control	0.20	N/A	N/A	0.74	0.74	2.42	2.42	N/A
256-Point FFT	0.59	N/A	N/A	0.73	0.73	1.66	1.66	N/A
Viterbi	0.65	N/A	N/A	1.03	1.03	2.63	2.63	N/A
Bit Unpack	0.36	N/A	N/A	0.85	0.85	2.78	2.78	N/A
Total	4.49			12.25	11.59	29.59	27.44	

Notes:

[1] As of this writing, pricing has not been disclosed for these processors. Hence, they are excluded from this metric.

[2] These processors use on-chip caches. Results under processor names denoted with "-C" represent performance with caches preloaded. Results for the processors listed under processor names without "-C" represent performance with caches uninitialized; see text for details.

[3] This chip contains two cores. Therefore, comparing its cost-execution time result to those of the other processors is not meaningful.

[4] Projected; as of this writing, these processors were not yet sampling or have not been demonstrated in silicon at their target speed.

[5] Please see Table 8.2-1 for information about processors used in this section.

Table 8.3-2. (cont.)

Motorola 56311	Motorola 56F801	Motorola 56853	Motorola MSC8101	TI C5416	TI C5510	TI C6203	TI C64xx	TI C64xx-C	TI C6701
		[4]			[4]		[1]	[1]	
1.08	0.72	0.10	0.21	0.55	0.26	0.84	N/A	N/A	1.45
0.80	0.45	0.07	0.33	0.48	0.33	1.23	N/A	N/A	2.64
1.03	0.73	0.10	0.24	0.70	0.29	0.99	N/A	N/A	1.77
0.97	0.51	0.08	0.21	0.57	0.42	1.04	N/A	N/A	1.81
0.69	0.44	0.07	0.28	0.38	0.33	1.16	N/A	N/A	2.69
0.93	0.56	0.08	0.27	0.53	0.49	1.16	N/A	N/A	2.14
1.12	0.70	0.11	0.23	0.59	0.33	1.01	N/A	N/A	1.57
0.94	0.60	0.13	0.28	0.87	0.43	0.80	N/A	N/A	1.21
1.06	0.54	0.09	0.56	0.71	0.68	1.27	N/A	N/A	1.58
1.44	1.55	0.19	0.27	1.12	0.69	0.86	N/A	N/A	1.52
1.07	1.37	0.17	0.12	0.44	0.23	0.80	N/A	N/A	0.83
1.05	0.81	0.12	0.34	1.02	0.38	0.74	N/A	N/A	0.92
12.17	8.98	1.30	3.34	7.96	4.85	11.90			20.14

Figure 8.3-1. Cost-Execution Time Product for Real Block FIR (Lower is Better)

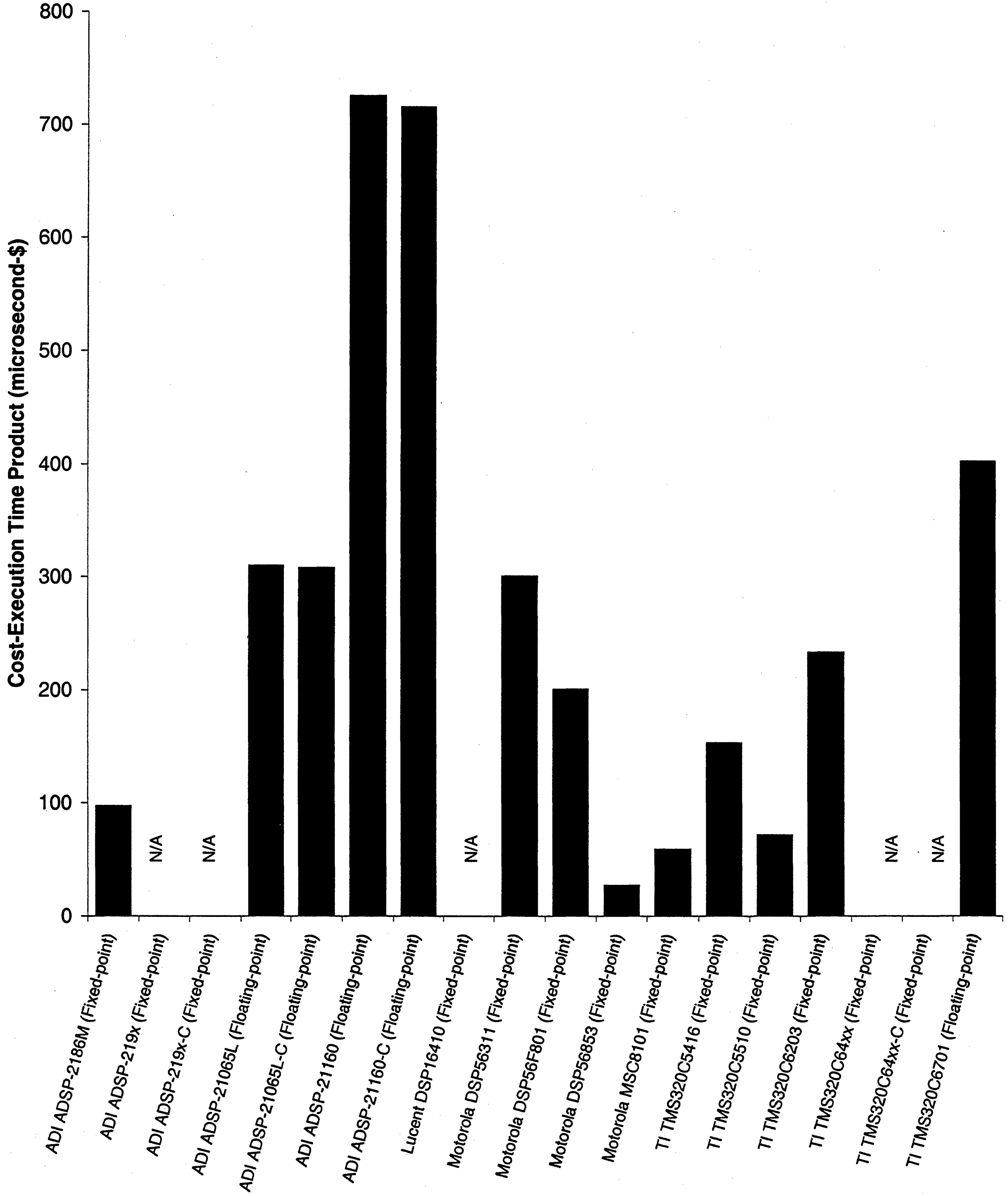


Figure 8.3-2. Cost-Execution Time Product for Single-Sample FIR
(Lower is Better)

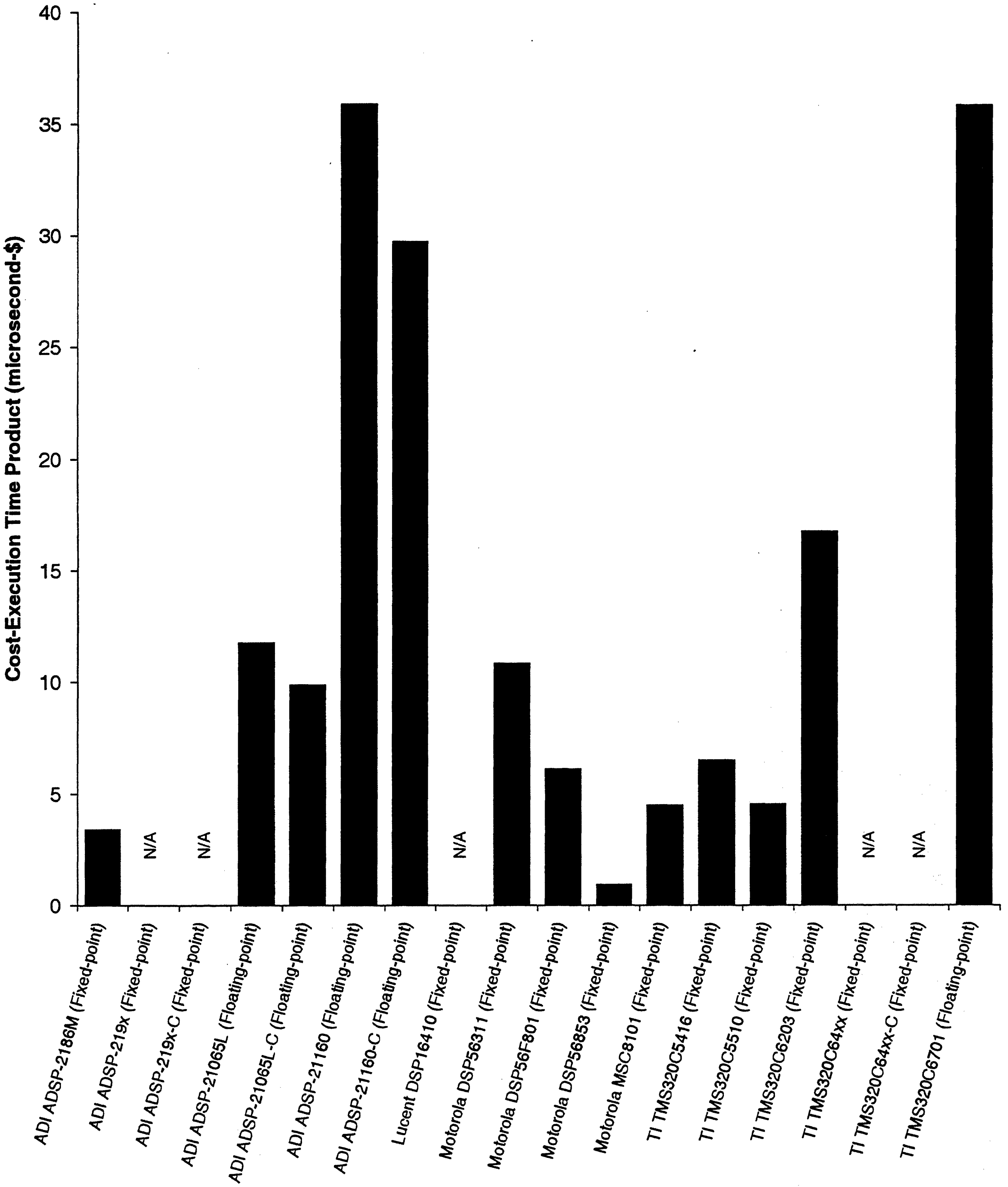


Figure 8.3-3. Cost-Execution Time Product for Complex Block FIR (Lower is Better)

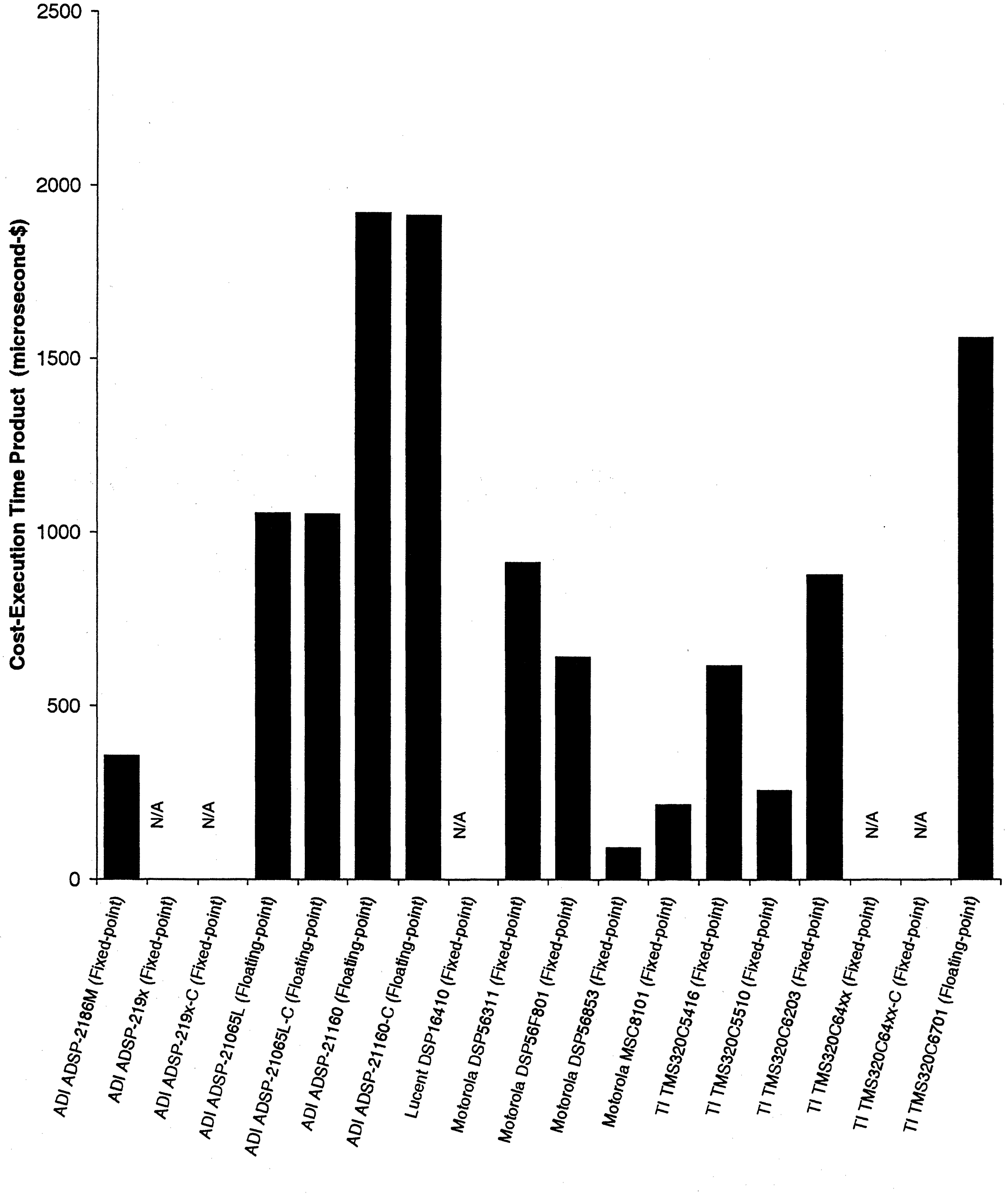


Figure 8.3-4. Cost-Execution Time Product for LMS Adaptive FIR (Lower is Better)

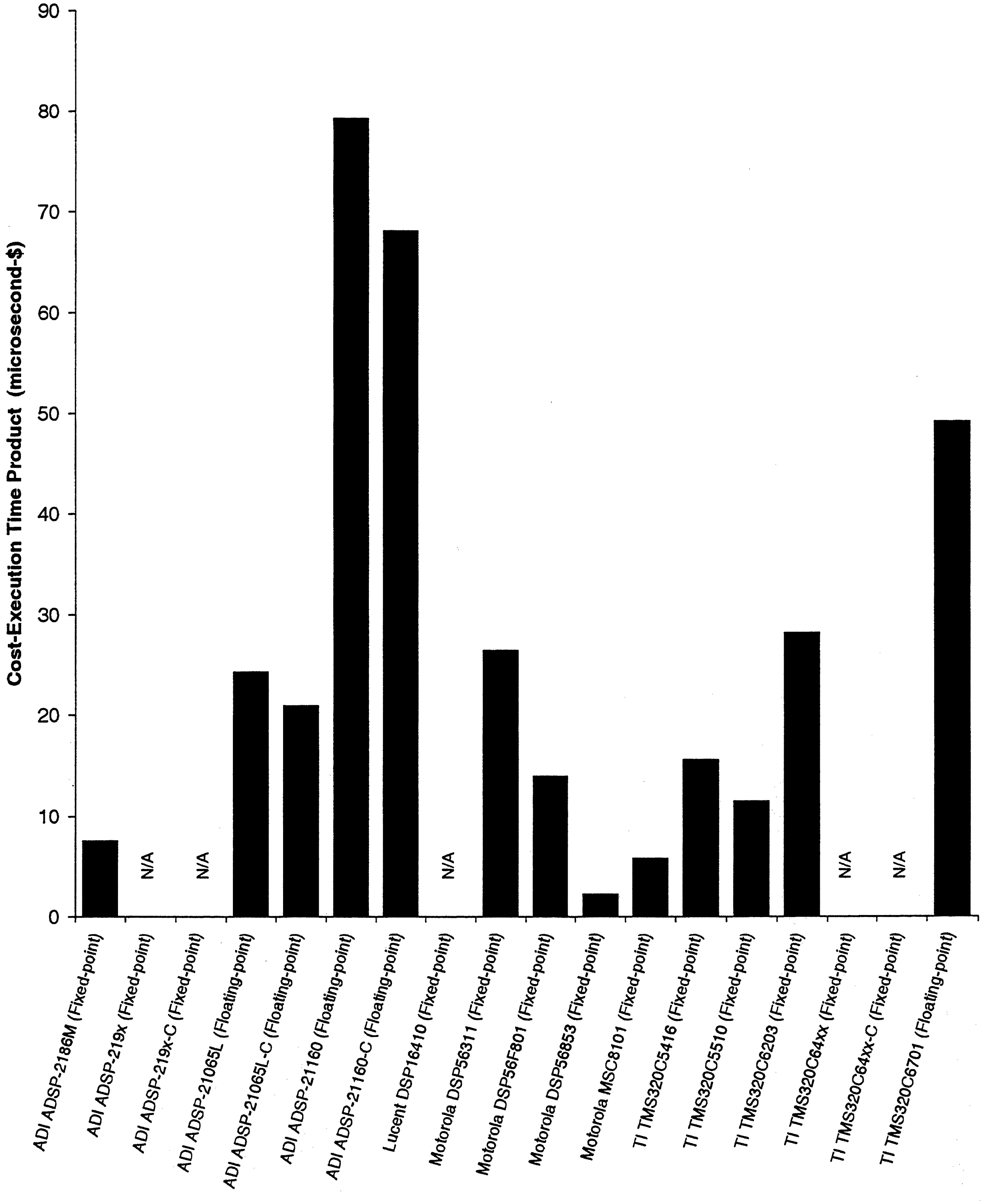


Figure 8.3-5. Cost-Execution Time Product for Two-Biquad IIR (Lower is Better)

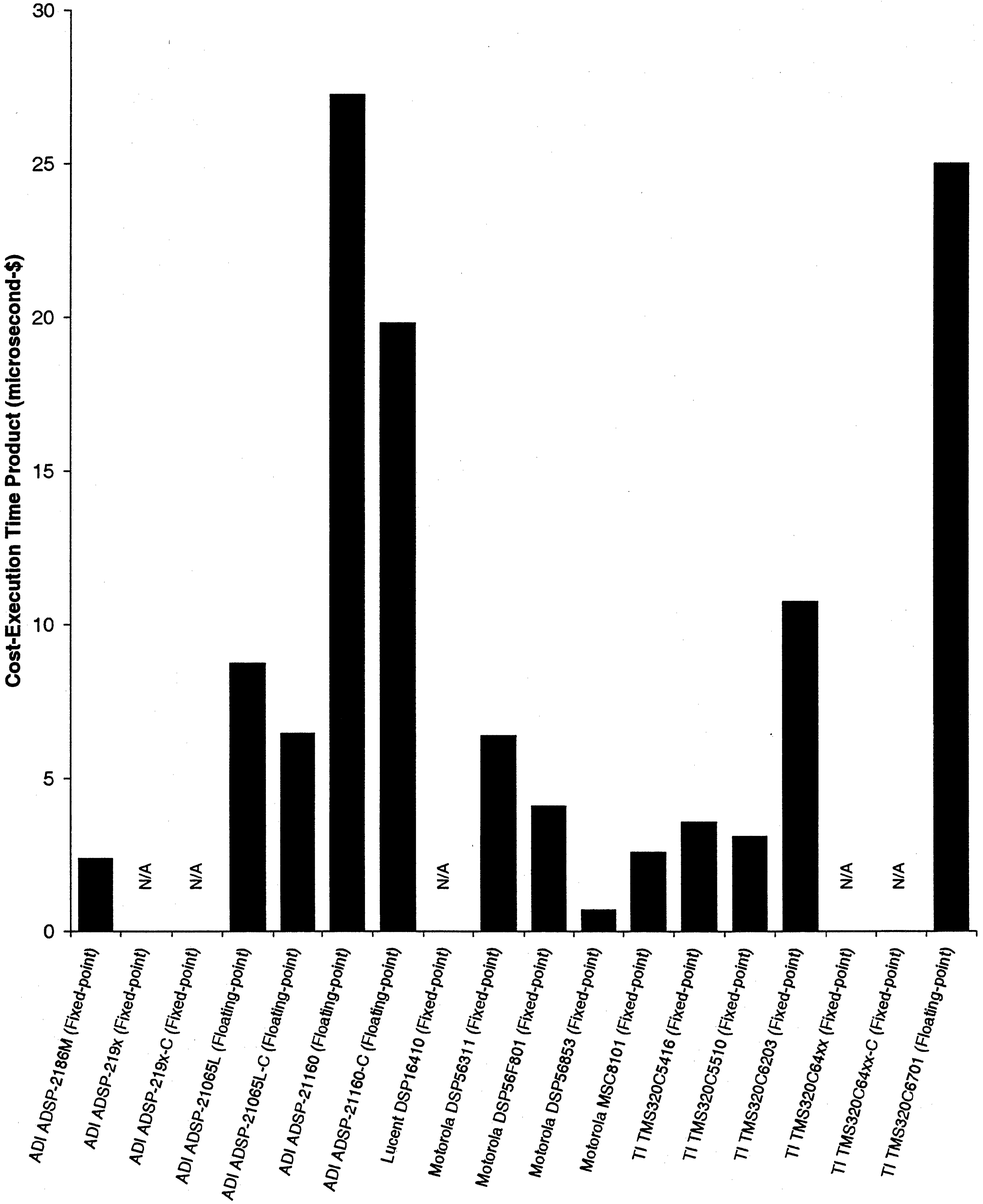


Figure 8.3-6. Cost-Execution Time Product for Vector Dot Product
(Lower is Better)

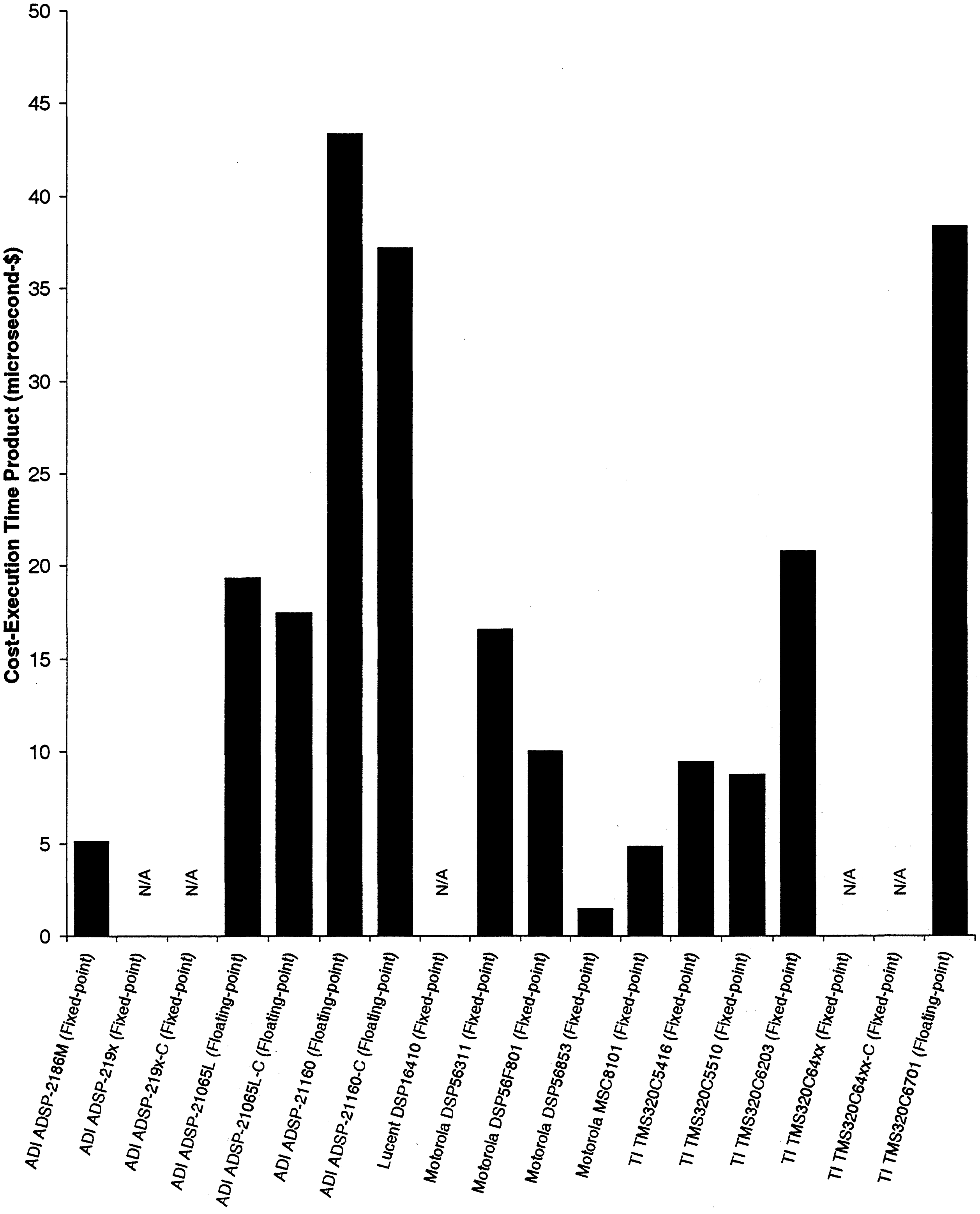


Figure 8.3-7. Cost-Execution Time Product for Vector Add (Lower is Better)

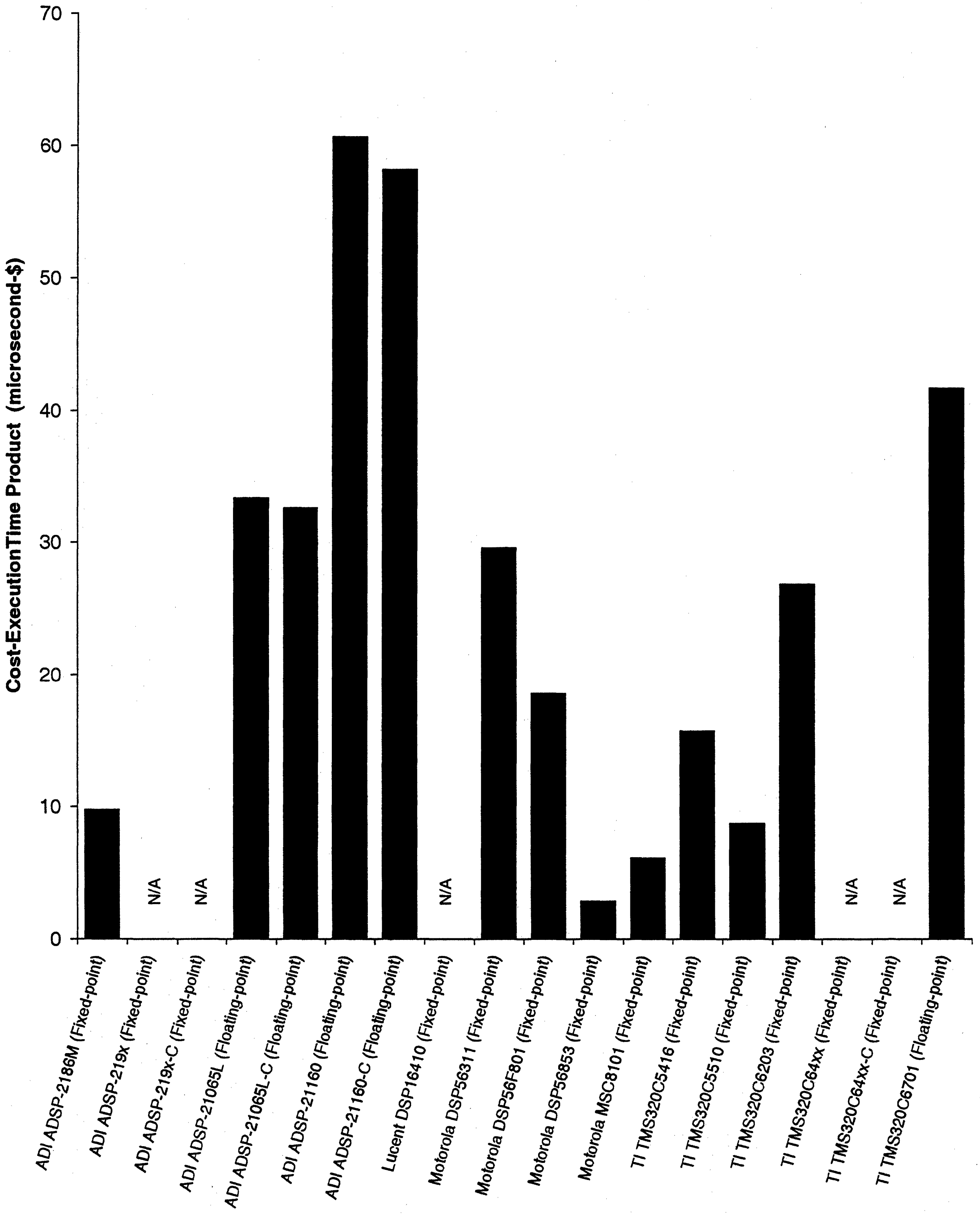


Figure 8.3-8. Cost-Execution Time Product for Vector Maximum (Lower is Better)

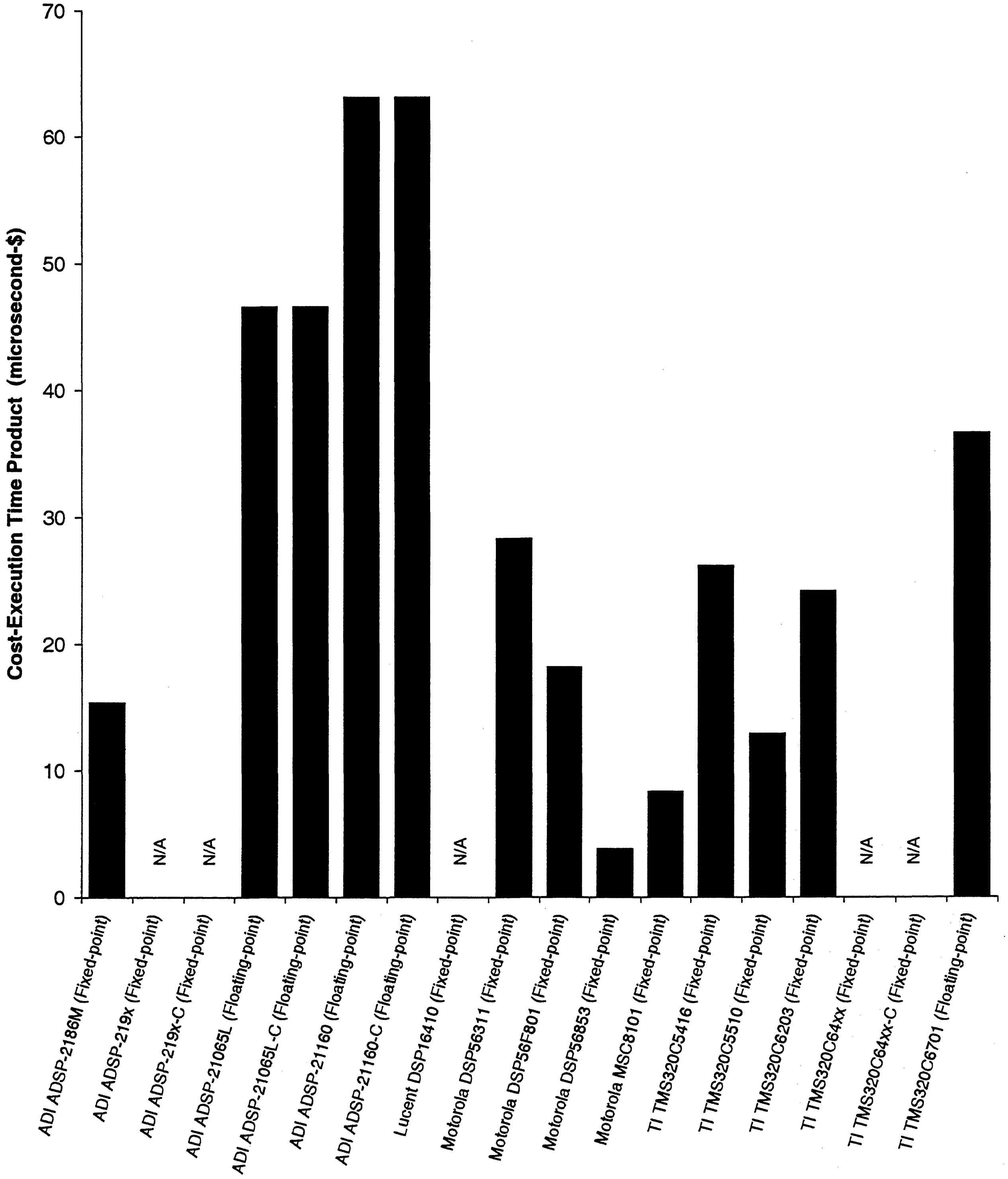


Figure 8.3-9. Cost-Execution Time Product for Control (Lower is Better)

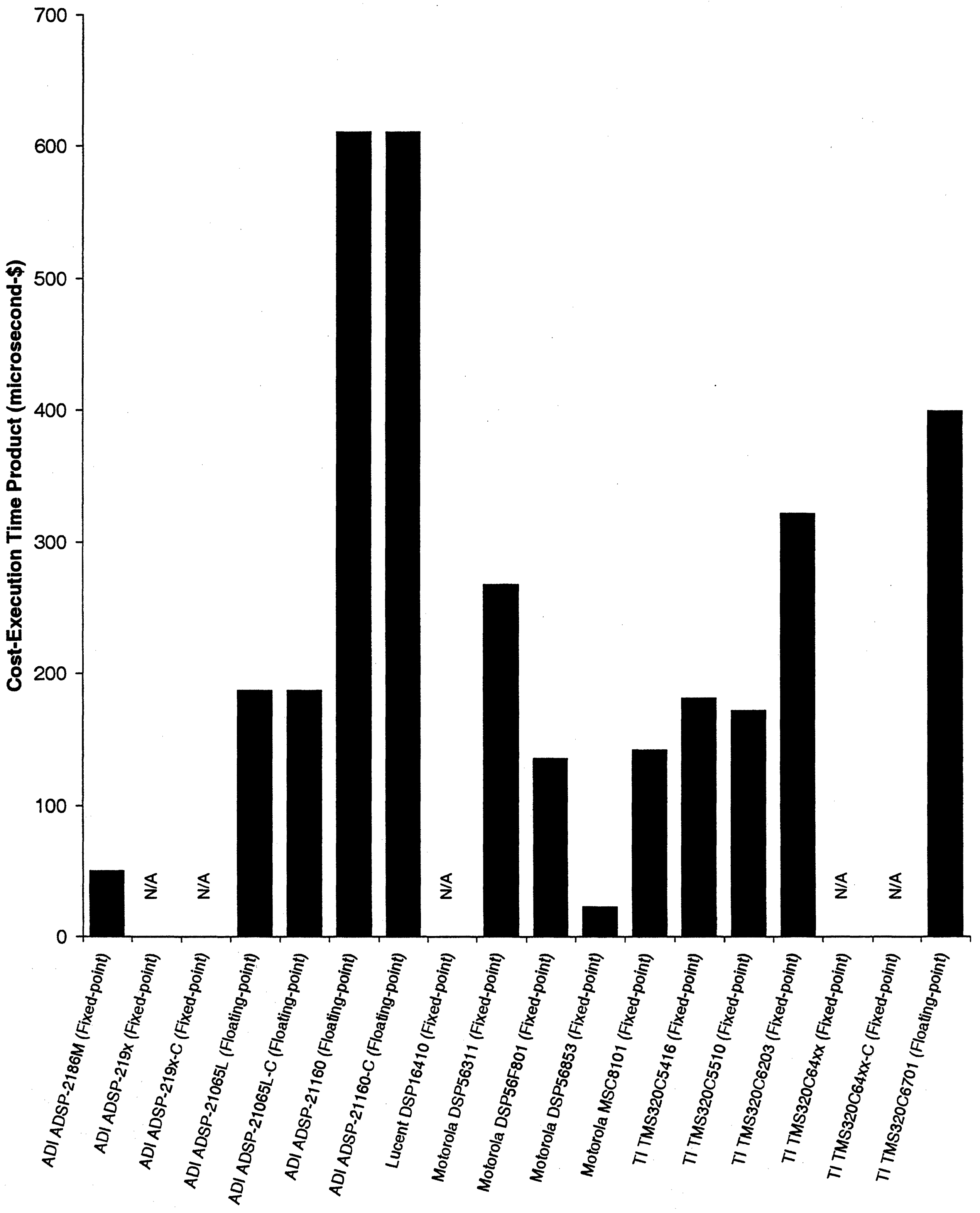


Figure 8.3-10. Cost-Execution Time Product for 256-Point FFT (Lower is Better)

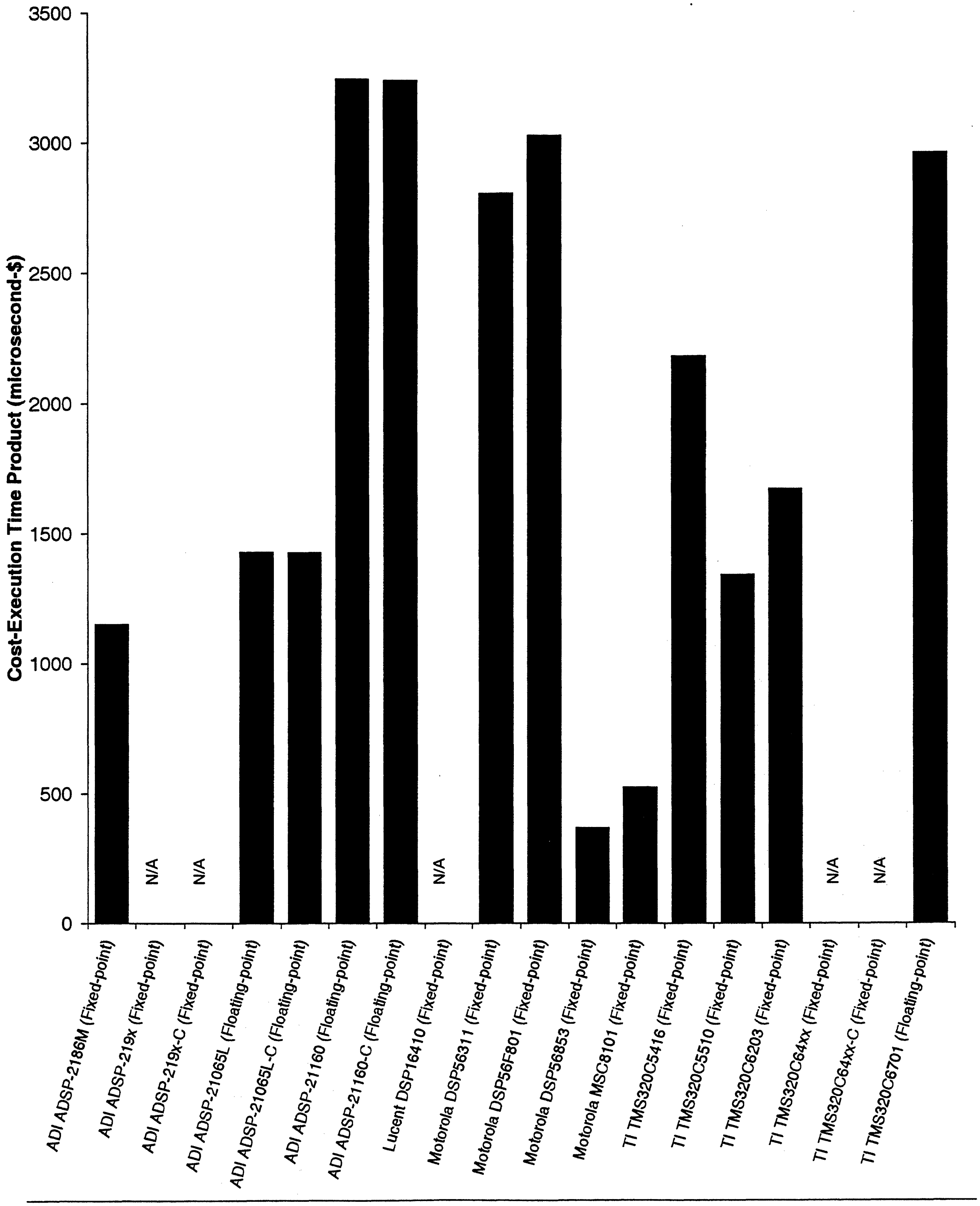


Figure 8.3-11. Cost-Execution Time Product for Viterbi (Lower is Better)

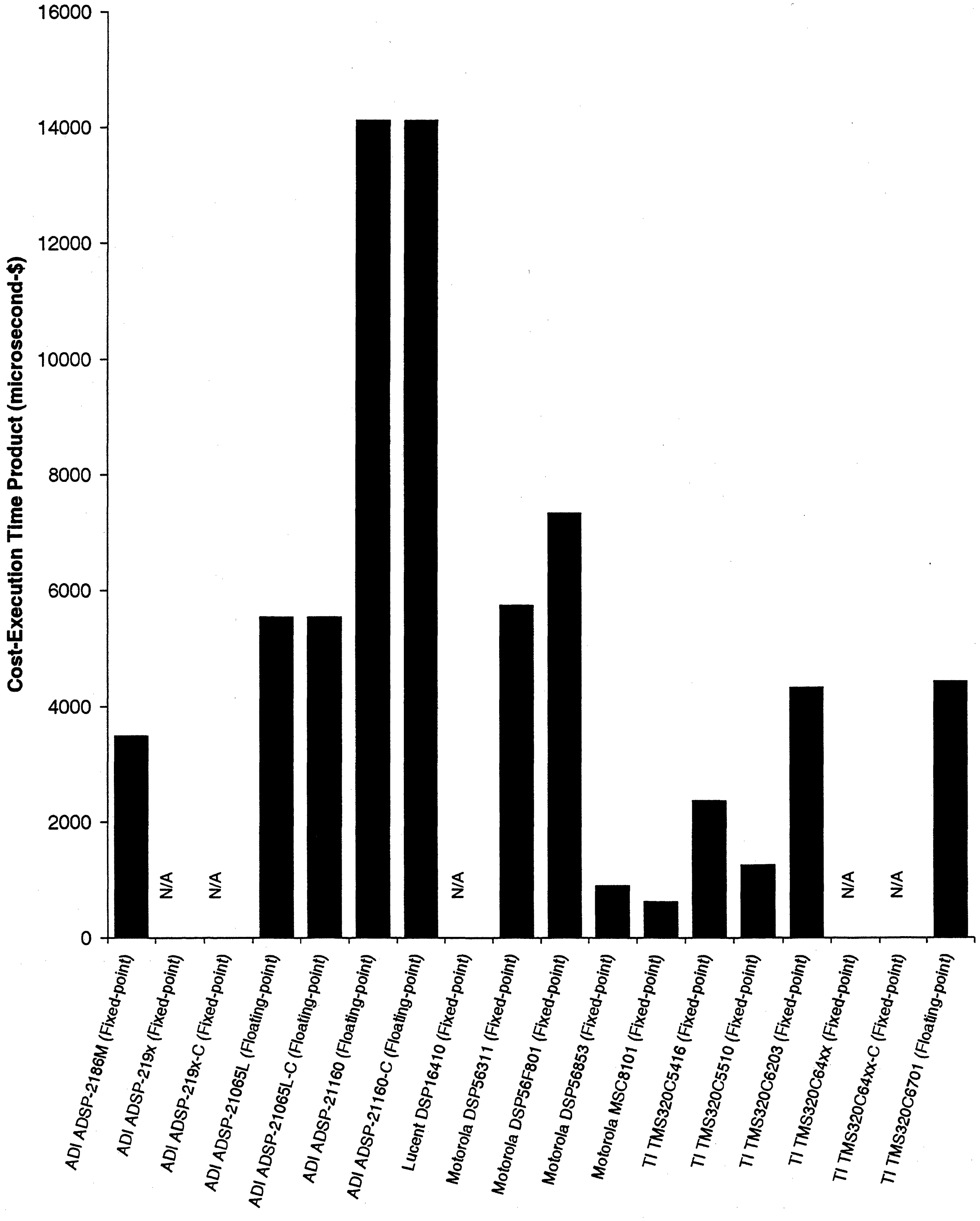


Figure 8.3-12. Cost-Execution Time Product for Bit Unpack (Lower is Better)

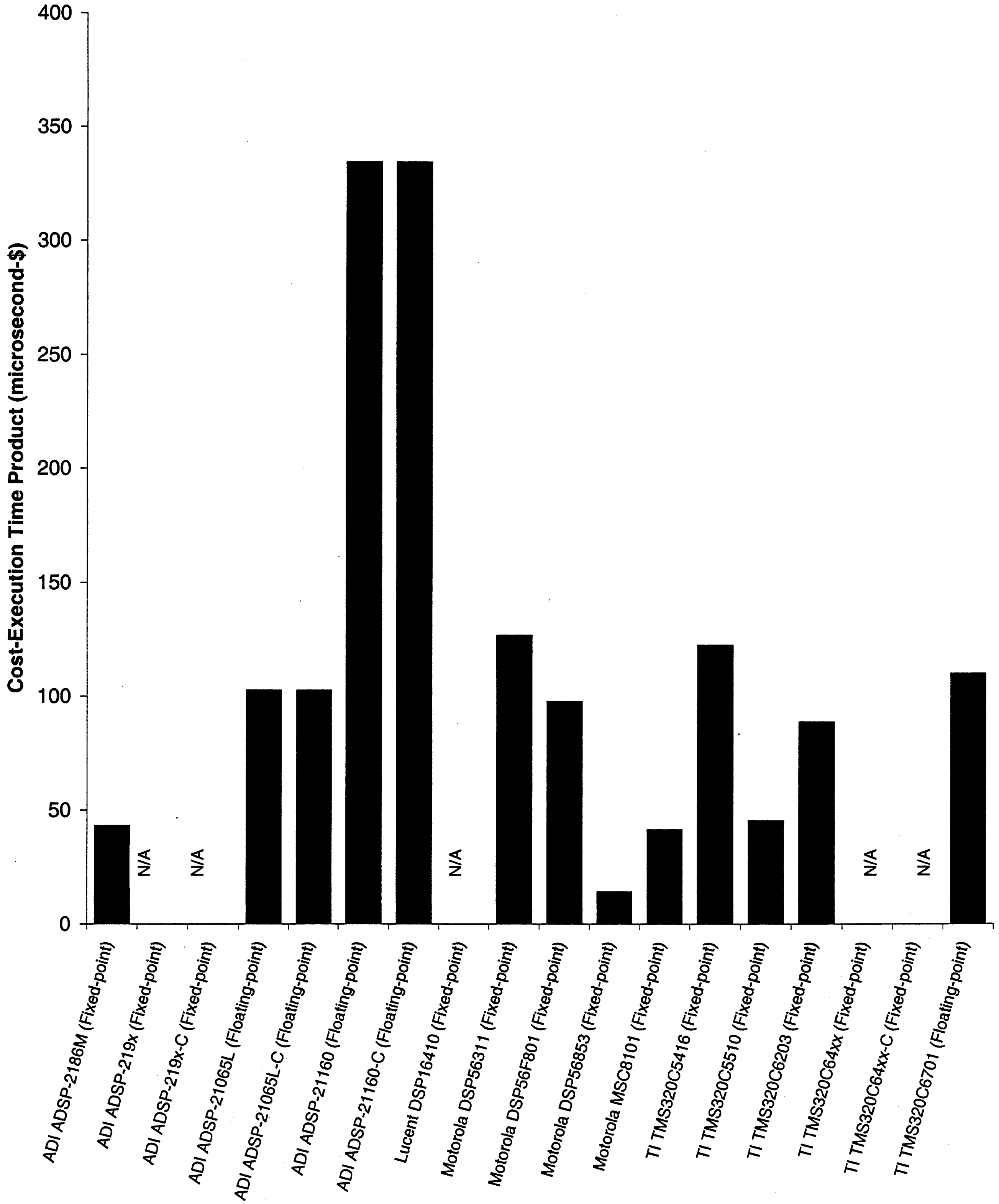
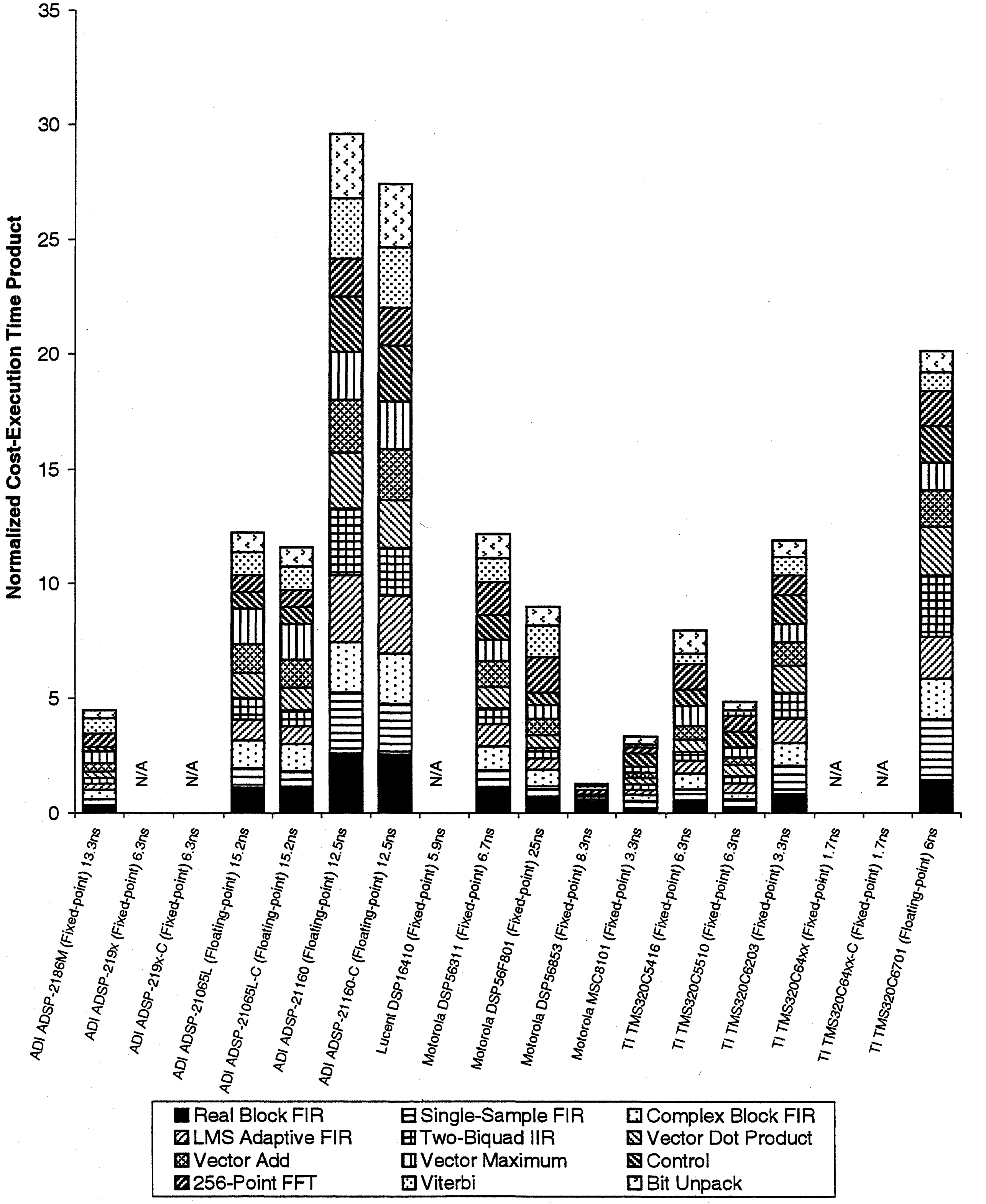


**Figure 8.3-13. Normalized Cost-Execution Time Product
Sum for all b of DT[p,b] / ADT[p,b]
(Lower is Better)**



8.4 Energy Consumption

In many applications, processor power or energy consumption is a key consideration. Power consumption is the rate at which a processor consumes energy. If the power consumption is constant over some period, then the energy consumption is equal to the product of the power consumption multiplied by the time period.

There are two main power-related criteria that might be used to compare processors:

- For many applications, *energy consumption* is a paramount concern. This is especially true of battery-powered applications, where energy consumption translates directly into battery life. The energy consumed for a particular task that is periodically repeated in an application, divided by the period of repetition, yields the average power consumption of the task in the application. This is generally different from a processor's typical power consumption and is often a more useful metric than typical power consumption. Average power consumption may also be a metric of interest if heat dissipation is a concern.
- Some applications are sensitive to the *maximum short-term processor power consumption* because of limits on the maximum amount of power that can be supplied or heat that can be dissipated.

In this section, we analyze and compare processor *energy consumption.* To estimate the energy required for a processor to complete a given benchmark function, we multiply the execution time for the processor on the given benchmark by the estimated typical power consumption for that processor. In most cases, we use manufacturer-supplied "typical" power ratings. In some cases, however, manufacturer-supplied power ratings are skewed due to reduced clock speeds or voltages, or assumptions regarding processor activity. In these cases, we have adjusted the manufacturer-supplied power data to yield what we believe to be a more realistic and comparable power consumption value.

Note that a processor with a lower power consumption may actually consume *more* energy in computing a given benchmark if the processor requires more time to complete the benchmark than another processor. By applying appropriate weightings to the energy consumption values for the various benchmarks, the reader can compute an overall energy consumption estimate that reflects the activities comprising a particular application.

For the energy consumption analysis presented here, the processor family member used may differ from that used in the preceding execution-time analysis. Here, the lowest-voltage member of each family that has enough on-chip memory to run the BDTI Benchmarks™ is used, at its fastest speed. Therefore, the instruction cycle times used here may differ from those used in the previous section.

This analysis is based on manufacturer-supplied power consumption data, and therefore has a higher uncertainty associated with it than other parts of our benchmark analysis. The power consumption data used here was not independently verified, and vendors' definition of "typical" power consumption is sometimes vague. Therefore, small differences between processors' energy consumption benchmark results should not be considered significant. BDTI did analyze power consumption data reported by vendors, and made adjustments where deemed necessary to obtain comparable data.

The analysis presented here does not include power and energy required by the processor to drive signals onto external pins and buses and into other connected devices. The amount of power and energy required for external signals depends on the number and type of external devices connected to the processor and the physical characteristics of the interconnections, as well as the amount of activity on the pins.

Some of the processors included in earlier sections of our analysis are excluded from this section. Since, at the time of this writing, Analog Devices had not announced a product based on the ASDP-219x architecture, and Texas Instruments had not announced a product based on the TMS320C64xx architecture, these processors are excluded from our analysis in this section and the next.

Since the Lucent DSP164xx includes two cores, but only one of the cores is used by the BDTI Benchmarks, the typical power consumption value used in our analysis for this device is equal to half of the typical power consumption value for the entire chip.

Typical power consumption for the Motorola DSP56854 is projected, and should therefore be considered less reliable.

The analysis and results presented in this section are organized as follows:

- **Processor summary power data.**
  Table 8.4-1 lists the typical power consumption for each processor. As mentioned above, we have chosen the lowest-voltage member of each processor family for the analysis in this section. Power consumption data is for the fastest variant of the lowest-voltage version of each processor.
- **Benchmark energy consumption.**
  The estimated energy consumption for each processor on each benchmark is shown in Table 8.4-2 and Figures 8.4-1 through 8.4-12. The energy consumption includes the main benchmark body, but not the power-up section.
- **Normalized benchmark energy consumption.**
  As the first step in creating aggregate energy consumption results that show each processor's overall energy consumption performance, we normalize the energy consumption for each processor on each benchmark. This normalization is done by dividing each processor's energy consumption on a given benchmark by the average over all processors on that benchmark. This is done so that benchmarks that are inherently more time consuming are not automatically weighted more heavily when we aggregate the performance of each processor over all benchmarks. Normalized benchmark energy consumption values are presented in Table 8.4-3.
- **Total normalized energy consumption through all benchmarks.**
  Adding the normalized benchmark energy consumption values for all benchmarks

for each processor produces an overall energy consumption measure for that processor. This sum effectively applies a uniform weighting to each benchmark for this section of the analysis. The results are shown in Figure 8.4-13A and 8.4-13B and at the bottom of Table 8.4-3. Note that Figure 8.4-13B is a magnified version of Figure 8.4-13A; the Y-axis has been scaled so that data for processors with relatively small results are more visible. As a result, the bars representing processors with relatively high results extend beyond the top of the graph and are truncated.

Analysis of Results

This section presents a brief analysis of the benchmark energy consumption results, summarized in Figures 8.4-13A and 8.4-13B. As discussed in the preceding paragraphs, benchmark energy consumption is estimated by multiplying a processor's execution time for a given benchmark function by an estimate of its power consumption.

In reviewing the normalized energy consumption results over all benchmarks presented in Figure 8.4-13A, several important points are revealed. First, floating-point processors generally consume much more energy than their fixed-point counterparts, regardless of variations in operating voltage. All three of the floating-point DSPs benchmarked here have notably high total normalized energy consumption results, at best roughly three times higher than that of the fixed-point DSPs with the highest energy consumption. This is not surprising, since floating-point processors require more complex circuitry for arithmetic operations and usually operate on larger data words, and since these processors generally do not target energy-sensitive applications. However, it must be noted that while the floating-point DSPs implement the same algorithms as the fixed-point devices, they provide much better numeric fidelity. In addition, floating-point processors often have on-chip peripherals and memory beyond what is typical in fixed-point DSPs. Therefore, comparisons between benchmark results for fixed-point processors and floating-point processors should be made with caution.

Although all of the floating-point processors have high energy consumption results relative to the benchmarked fixed-point DSPs, there is significant variation among the floating-point DSPs. Among this group, the Texas Instruments TMS320C6701 has the best energy consumption result; its total normalized energy consumption result is approximately half that of the Analog Devices ADSP-21065L, and roughly one third that of the Analog Devices ADSP-21160M.

Among fixed-point DSPs, there is also significant variation in energy efficiency. The Motorola MSC8101 has the lowest total normalized energy consumption result of the processors benchmarked here. In contrast, the Texas Instruments TMS320C6204 has an energy consumption result approximately eight times higher.

The very low energy consumption result of the MSC8101 is particularly notable considering the processor's high speed (as analyzed in Section 8.2, *Execution Times*). The MSC8101 uses an operating voltage of 1.5 volts, which it shares with many of the other processors benchmarked here.

The Motorola DSP56854 and the Texas Instruments TMS320C5416 and TMS320C5510 also have low energy consumption results. Note that DSP56854 result is based on projected power consumption, as this device has not yet been manufactured. The TMS320C54xx has been among the most energy efficient DSP families for several years, and the TMS320C5510 builds on that legacy, reducing energy consumption by about 10% while also boosting speed. These energy-efficient conventional DSPs benefit from low operating voltages (1.5-1.8 volts) and relatively simple architectures compared with many of the other processors benchmarked here.

Note that the overall energy consumption results can obscure significant differences on the individual benchmarks. For example, on the Real Block FIR benchmark the TMS320C5510 uses about half as much energy as the DSP56854 projected result. On the Control benchmark, however, the DSP56854 is projected to use about half as much energy as the TMS320C5510. Thus, if processing like that found in the Control benchmark consumes a significant percentage of the processor's time in an application and low energy consumption is vital, the DSP56854 may be a better choice than the TMS320C5510. This underscores the importance of understanding the mix of operations that a particular application uses before attempting to choose a processor based on energy consumption.

Table 8.4-1. Processor Power Consumption

Lowest Voltage Version of Each Processor at the Fastest Speed and Lowest Cost

Typical power consumption values in watts, P[p] (see note 1)

(cont.)

	ADI 2186M	**ADI 219x**	**ADI 219x-C**	**ADI 21065L**	**ADI 21065L-C**	**ADI 21160M**	**ADI 21160M-C**	**Lucent 16410**
Notes		[2,4]	[2,4]	[3]	[3]	[3]	[3]	[5]
Voltage	2.5	2.5	2.5	3.3	3.3	2.5	2.5	1.8
Instruction Cycle Time, ns (I[p])	13.3	6.3	6.3	15.2	15.2	12.5	12.5	5.9
Instruction Rate, Millions of Instruction Cycles per second (1/I[p])	75.0	160.0	160.0	66.0	66.0	80.0	80.0	170.0
Typical Power Consumption [P], watts	0.113	N/A	N/A	0.930	0.930	2.350	2.350	0.282

Notes:

[1] The power consumption figures presented here do not include power required to drive external loads; see text.

[2] As of this writing, power consumption has not been disclosed for these processors. Hence, they are excluded from this metric.

[3] These processors use on-chip caches. Results under processor names denoted with "-C" represent performance with caches preloaded. Results for the processors listed under processor names without "-C" represent performance with caches uninitialized.

[4] Projected; as of this writing, these processors were not yet sampling or have not been demonstrated in silicon at their target speed.

[5] Power consumption is for one of the two on-chip cores and half of the on-chip memory; see text.

Table 8.4-1. (cont.)

Motorola 56333	Motorola 56824	Motorola 56854	Motorola MSC8101	TI C5416	TI C5510	TI C6204	TI C64xx	TI C64xx-C	TI C6701
		[4]			[4]		[2,4]	[2,4]	
1.8	3.3	1.8	1.5	1.5	1.6	1.5	1.5	1.5	1.9
6.7	28.6	8.3	3.3	6.3	6.3	5.0	1.7	1.7	6.0
150.0	35.0	120.0	300.0	160.0	160.0	200.0	600.0	600.0	167.0
0.189	0.054	0.065	0.250	0.090	0.109	0.800	N/A	N/A	1.400

Table 8.4-2. Benchmark Energy Consumption

E[p,b] = C[p,b] * I[p] * P[p] / 1000
Based on "Typical" Power Consumption, in watt-microseconds

Lowest Voltage Version of Each Processor at the Fastest Speed and Lowest Cost

(cont.)

DSP Building Block Functions	**ADI 2186M**	**ADI 219x**	**ADI 219x-C**	**ADI 21065L**	**ADI 21065L-C**	**ADI 21160M**	**ADI 21160M-C**	**Lucent 16410**
Notes		[1]	[1]	[2]	[2]	[2]	[2]	[3]
Real Block FIR	1.29	N/A	N/A	11.51	11.44	17.21	16.98	0.80
Single-Sample FIR	0.05	N/A	N/A	0.44	0.37	0.85	0.71	0.04
Complex Block FIR	4.72	N/A	N/A	39.22	39.09	45.53	45.36	2.80
LMS Adaptive FIR	0.10	N/A	N/A	0.90	0.78	1.88	1.62	0.09
Two-Biquad IIR	0.03	N/A	N/A	0.32	0.24	0.65	0.47	0.02
Vector Dot Product	0.07	N/A	N/A	0.72	0.65	1.03	0.88	0.04
Vector Add	0.13	N/A	N/A	1.24	1.21	1.44	1.38	0.08
Vector Maximum	0.20	N/A	N/A	1.73	1.73	1.50	1.50	0.05
Control	0.66	N/A	N/A	6.95	6.95	14.48	14.48	0.94
256-Point FFT	15.28	N/A	N/A	53.07	53.00	77.02	76.87	11.81
Viterbi	46.30	N/A	N/A	206.01	206.01	334.93	334.93	15.52
Bit Unpack	0.57	N/A	N/A	3.80	3.80	7.93	7.93	0.81

Notes:
[1] As of this writing, power consumption has not been disclosed for these processors. Hence, they are excluded from this metric.
[2] These processors use on-chip caches. Results under processor names denoted with "-C" represent performance with caches preloaded. Results for the processors listed under processor names without "-C" represent performance with caches uninitialized.
[3] Power consumption is for one of the two on-chip cores and half of the on-chip memory; see text.
[4] Projected; as of this writing, these processors were not yet sampling or have not been demonstrated in silicon at their target speed.

Table 8.4-2. (cont.)

Motorola 56333	Motorola 56824	Motorola 56854	Motorola MSC8101	TI C5416	TI C5510	TI C6204	TI C64xx	TI C64xx-C	TI C6701
		[4]			[4]		[1]	[1]	
1.19	1.52	0.47	0.15	0.41	0.27	1.39	N/A	N/A	4.05
0.04	0.05	0.02	0.01	0.02	0.02	0.10	N/A	N/A	0.36
3.61	4.85	1.55	0.56	1.65	0.96	5.23	N/A	N/A	15.71
0.10	0.10	0.04	0.02	0.04	0.04	0.17	N/A	N/A	0.49
0.03	0.03	0.01	0.01	0.01	0.01	0.06	N/A	N/A	0.25
0.07	0.08	0.03	0.01	0.03	0.03	0.12	N/A	N/A	0.39
0.12	0.14	0.05	0.02	0.04	0.03	0.16	N/A	N/A	0.42
0.11	0.14	0.07	0.02	0.07	0.05	0.14	N/A	N/A	0.37
1.06	1.02	0.39	0.37	0.48	0.64	1.92	N/A	N/A	4.02
11.12	22.93	6.34	1.36	5.85	5.03	9.97	N/A	N/A	29.82
22.72	55.52	15.46	1.62	6.34	4.68	25.74	N/A	N/A	44.60
0.50	0.74	0.25	0.11	0.33	0.17	0.53	N/A	N/A	1.11

Table 8.4-3. Normalized Energy Consumption

Lowest Voltage Version of Each Processor at the Fastest Speed and Lowest Cost
Energy / Average for Benchmark, E[p,b] / AE[b]

(cont.)

DSP Building Block Functions	ADI 2186M	ADI 219x	ADI 219x-C	ADI 21065L	ADI 21065L-C	ADI 21160M	ADI 21160M-C	Lucent 16410
Notes		[1]	[1]	[2]	[2]	[2]	[2]	[3]
Real Block FIR	0.26	N/A	N/A	2.35	2.33	3.51	3.46	0.16
Single-Sample FIR	0.21	N/A	N/A	2.00	1.68	3.91	3.23	0.17
Complex Block FIR	0.31	N/A	N/A	2.60	2.60	3.02	3.01	0.19
LMS Adaptive FIR	0.22	N/A	N/A	1.98	1.70	4.13	3.55	0.20
Two-Biquad IIR	0.21	N/A	N/A	2.11	1.56	4.21	3.06	0.16
Vector Dot Product	0.23	N/A	N/A	2.43	2.20	3.48	2.98	0.15
Vector Add	0.28	N/A	N/A	2.69	2.63	3.12	2.99	0.17
Vector Maximum	0.37	N/A	N/A	3.16	3.16	2.73	2.73	0.10
Control	0.17	N/A	N/A	1.79	1.79	3.73	3.73	0.24
256-Point FFT	0.56	N/A	N/A	1.96	1.96	2.84	2.84	0.44
Viterbi	0.49	N/A	N/A	2.18	2.18	3.55	3.55	0.16
Bit Unpack	0.28	N/A	N/A	1.86	1.86	3.89	3.89	0.40
Total	3.59			27.12	25.64	42.12	39.03	2.54

Notes:

[1] As of this writing, power consumption has not been disclosed for these processors. Hence, they are excluded from this metric.

[2] These processors use on-chip caches. Results under processor names denoted with "-C" represent performance with caches preloaded. Results for the processors listed under processor names without "-C" represent performance with caches uninitialized.

[3] Power consumption is for one of the two on-chip cores and half of the on-chip memory; see text.

[4] Projected; as of this writing, these processors were not yet sampling or have not been demonstrated in silicon at their target speed.

Table 8.4-3. (cont.)

Motorola 56333	Motorola 56824	Motorola 56854	Motorola MSC8101	TI C5416	TI C5510	TI C6204	TI C64xx	TI C64xx-C	TI C6701
		[4]			[4]		[1]	[1]	
0.24	0.31	0.10	0.03	0.08	0.05	0.28	N/A	N/A	0.83
0.20	0.21	0.07	0.05	0.08	0.08	0.46	N/A	N/A	1.65
0.24	0.32	0.10	0.04	0.11	0.06	0.35	N/A	N/A	1.04
0.23	0.23	0.08	0.03	0.09	0.09	0.37	N/A	N/A	1.09
0.16	0.20	0.08	0.04	0.06	0.08	0.42	N/A	N/A	1.64
0.22	0.26	0.09	0.04	0.09	0.11	0.42	N/A	N/A	1.31
0.25	0.30	0.11	0.03	0.09	0.07	0.35	N/A	N/A	0.91
0.20	0.25	0.12	0.04	0.13	0.09	0.26	N/A	N/A	0.67
0.27	0.26	0.10	0.10	0.12	0.17	0.49	N/A	N/A	1.03
0.41	0.85	0.23	0.05	0.22	0.19	0.37	N/A	N/A	1.10
0.24	0.59	0.16	0.02	0.07	0.05	0.27	N/A	N/A	0.47
0.25	0.36	0.12	0.05	0.16	0.08	0.26	N/A	N/A	0.54
2.92	4.14	1.36	0.53	1.30	1.12	4.30			12.28

Figure 8.4-1. Energy Consumption for Real Block FIR (Lower is Better)

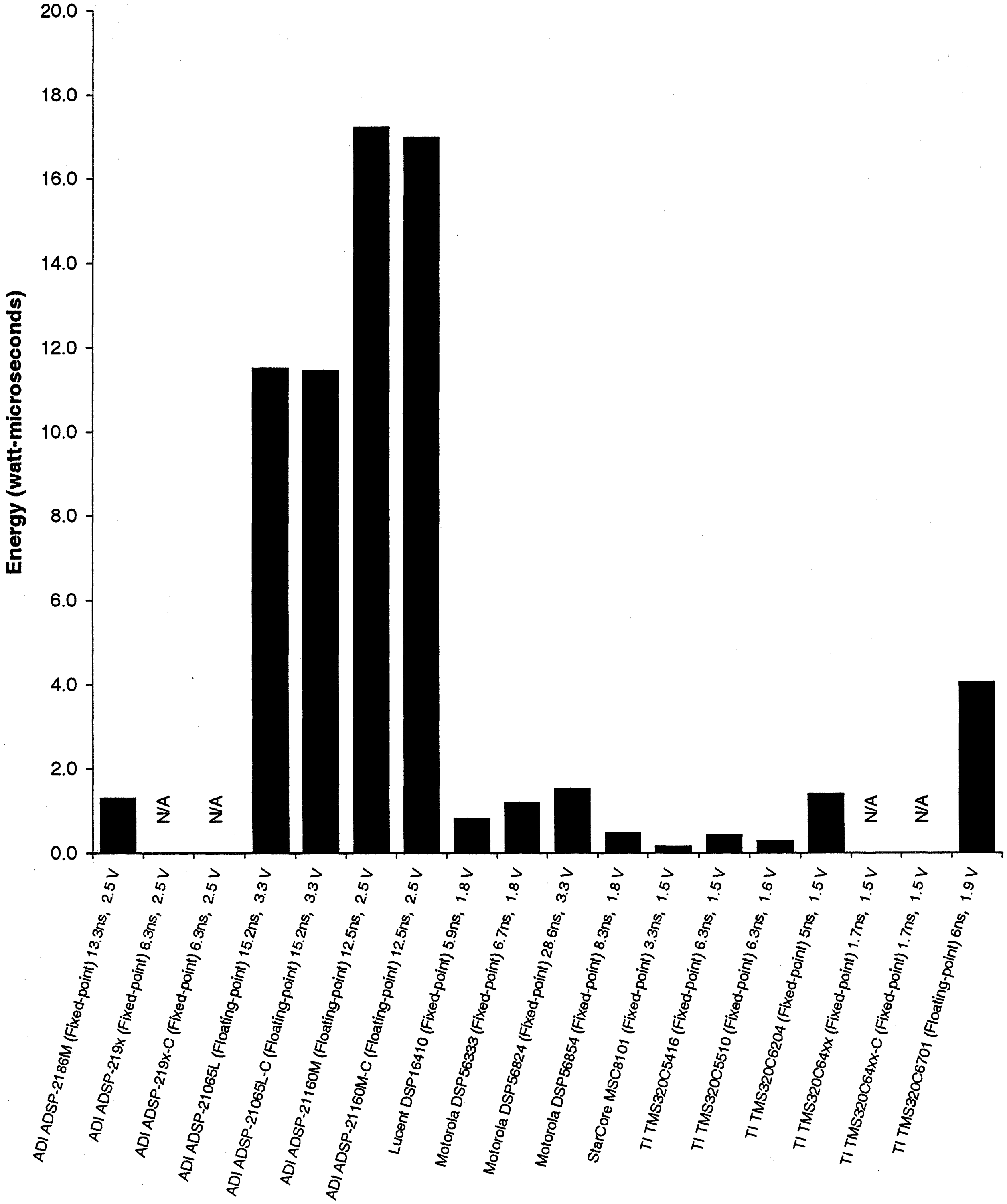


Figure 8.4-2. Energy Consumption for Single-Sample FIR (Lower is Better)

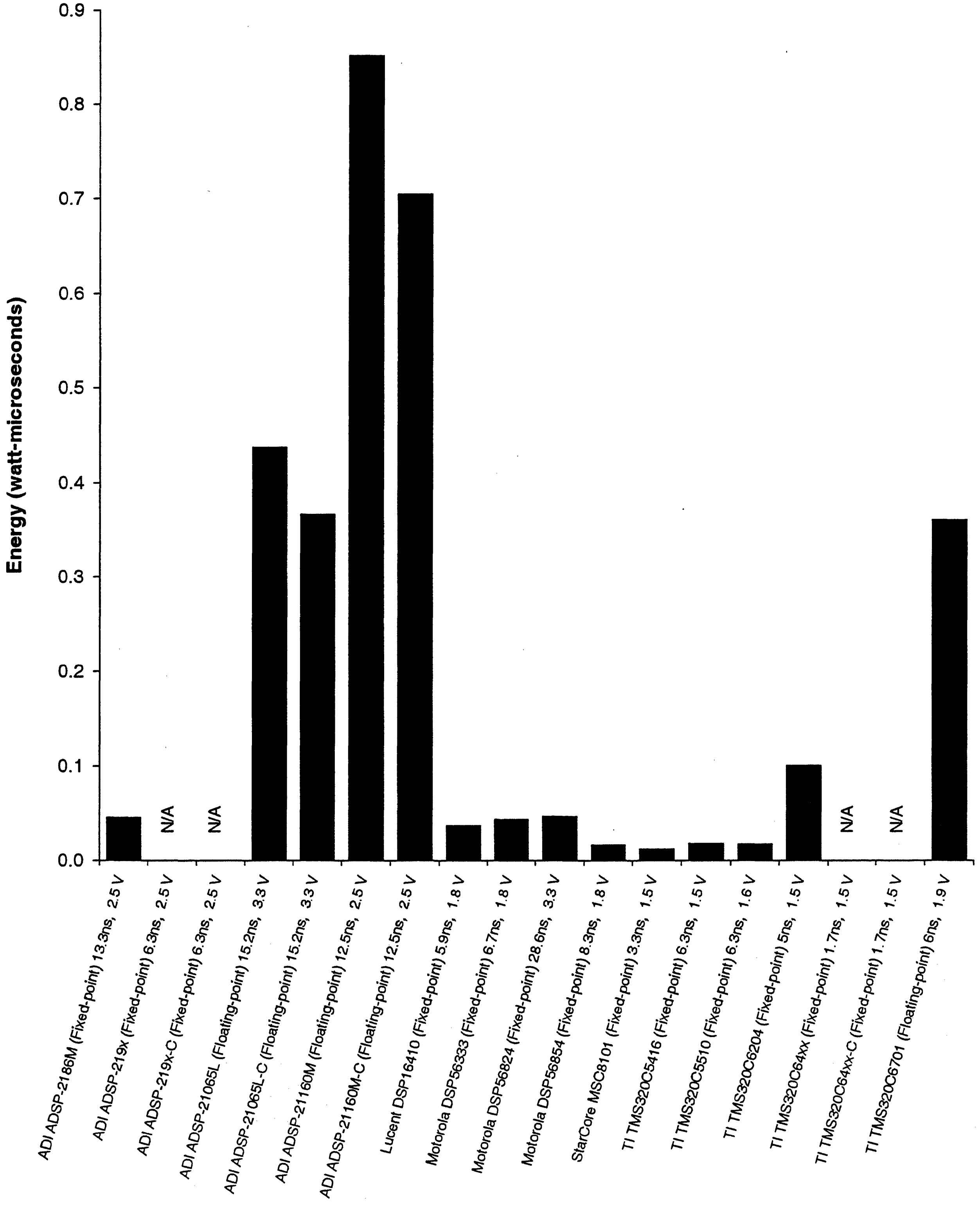


Figure 8.4-3. Energy Consumption for Complex Block FIR (Lower is Better)

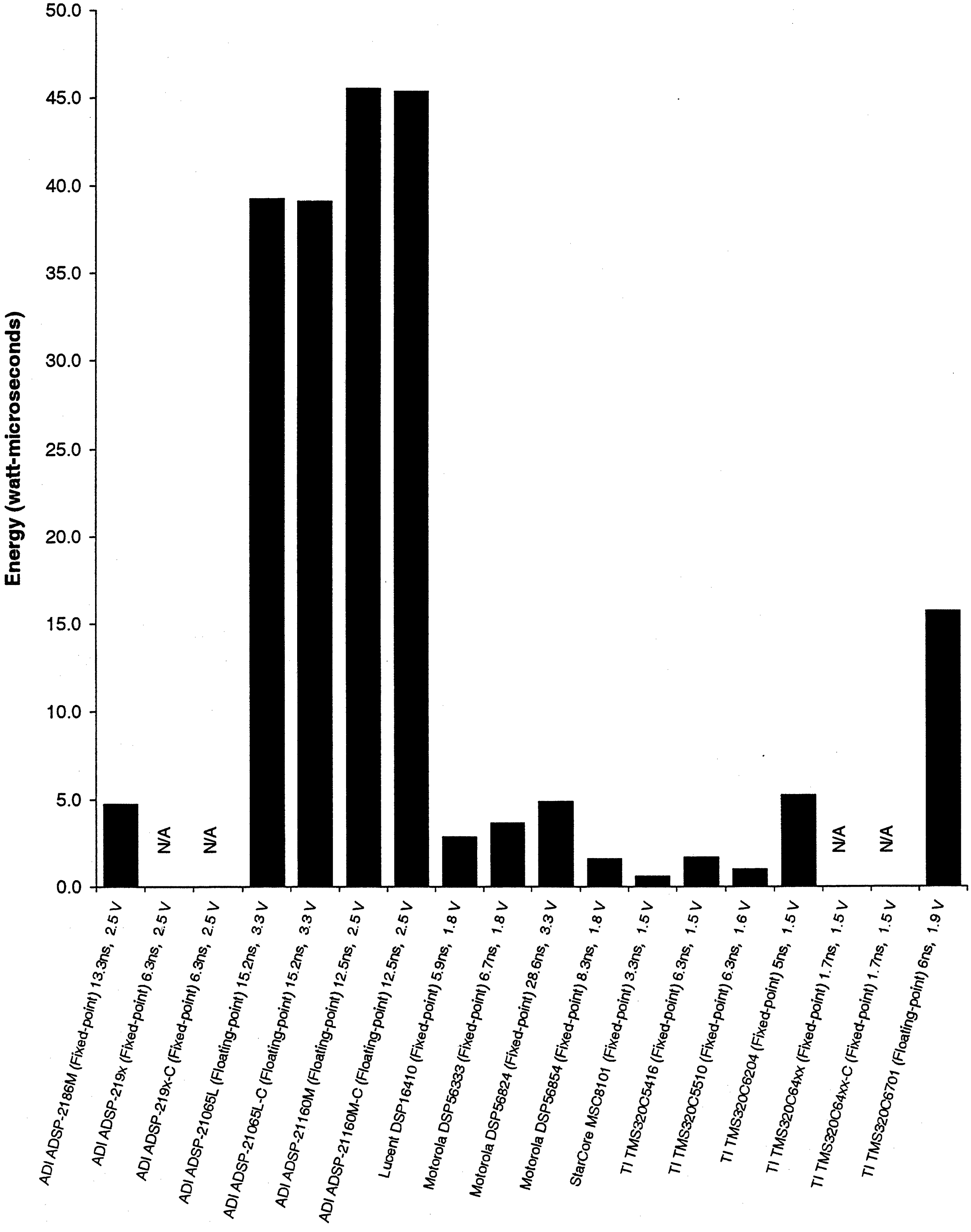


Figure 8.4-4. Energy Consumption for LMS Adaptive FIR (Lower is Better)

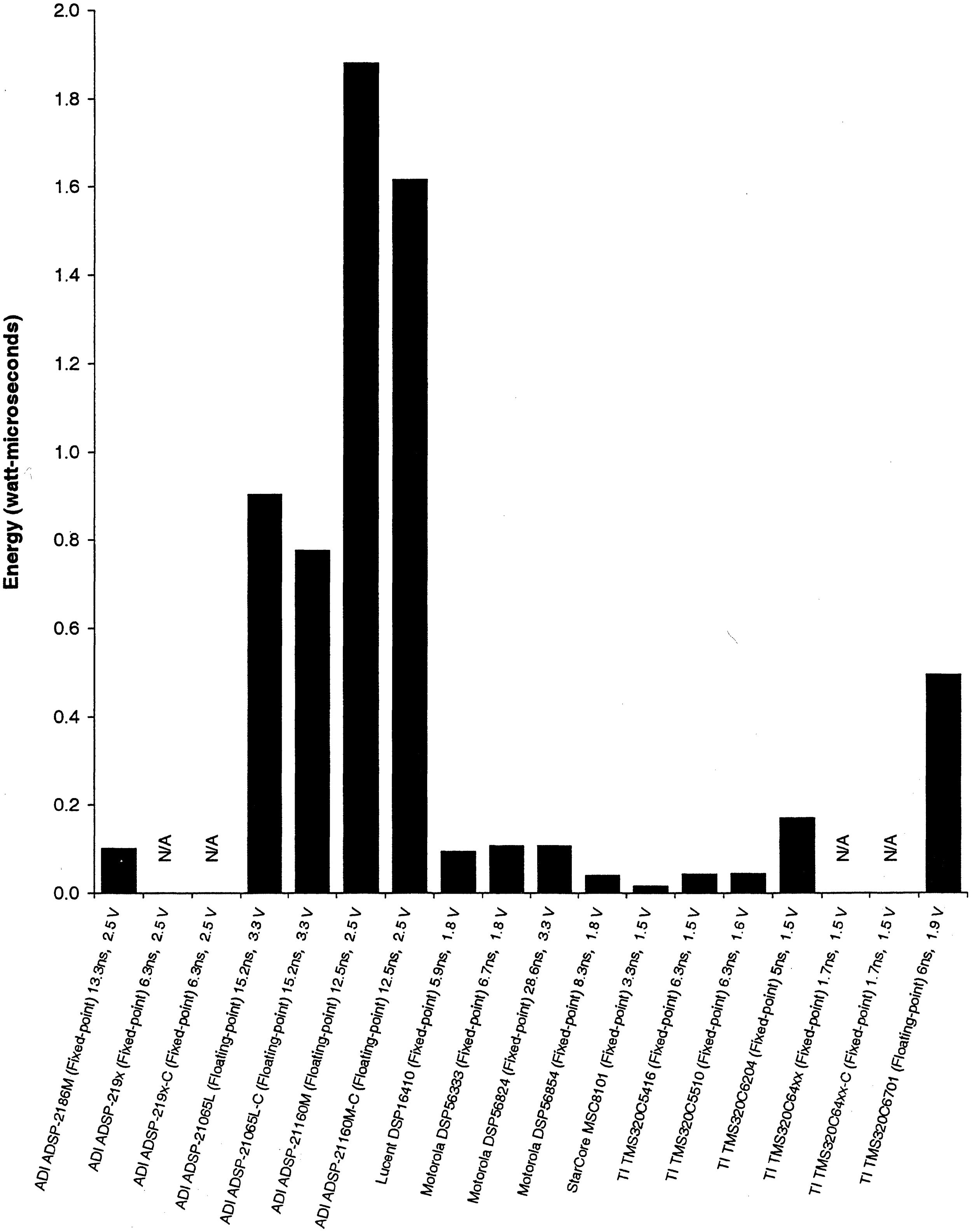


Figure 8.4-5. Energy Consumption for Two-Biquad IIR (Lower is Better)

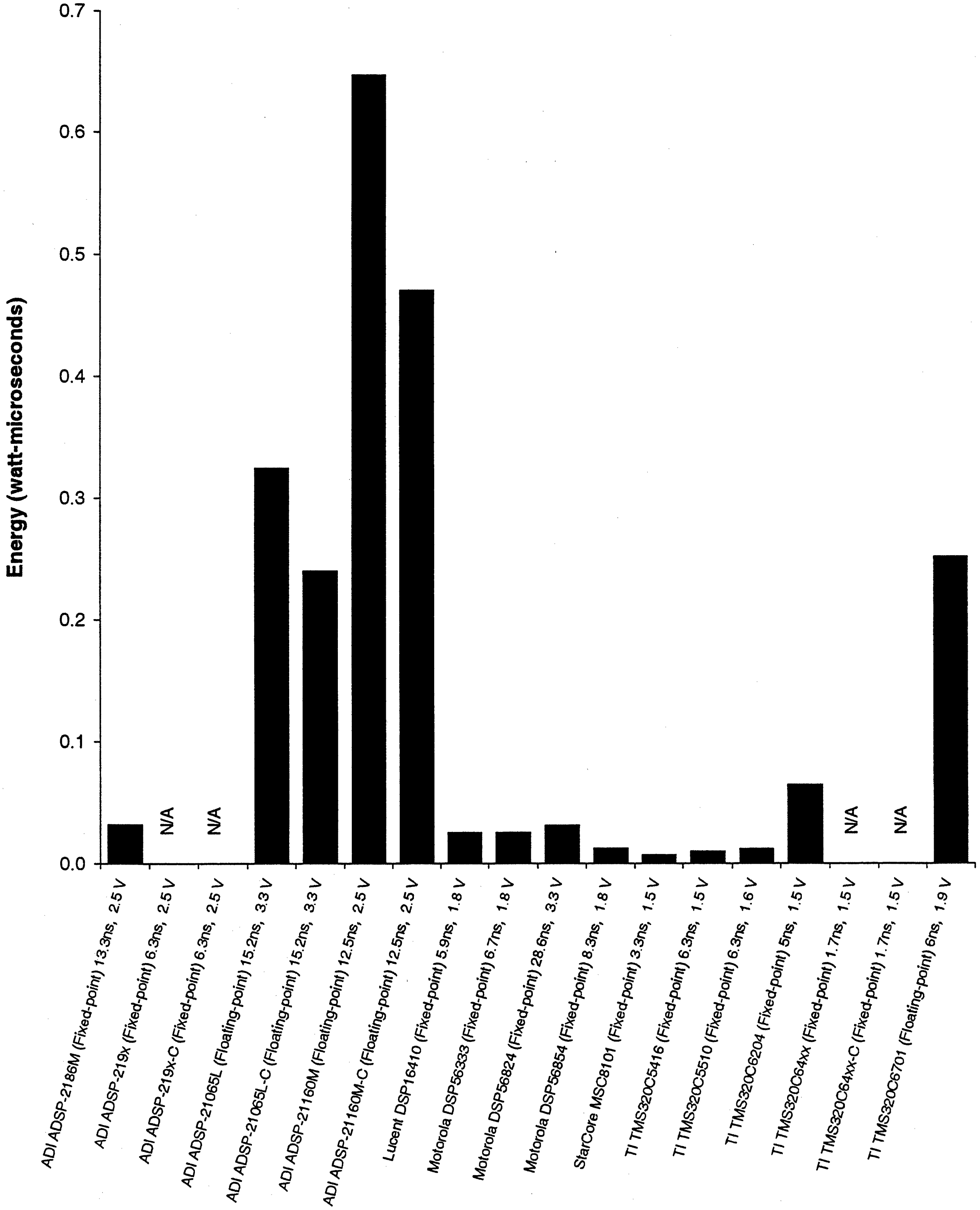


Figure 8.4-6. Energy Consumption for Vector Dot Product (Lower is Better)

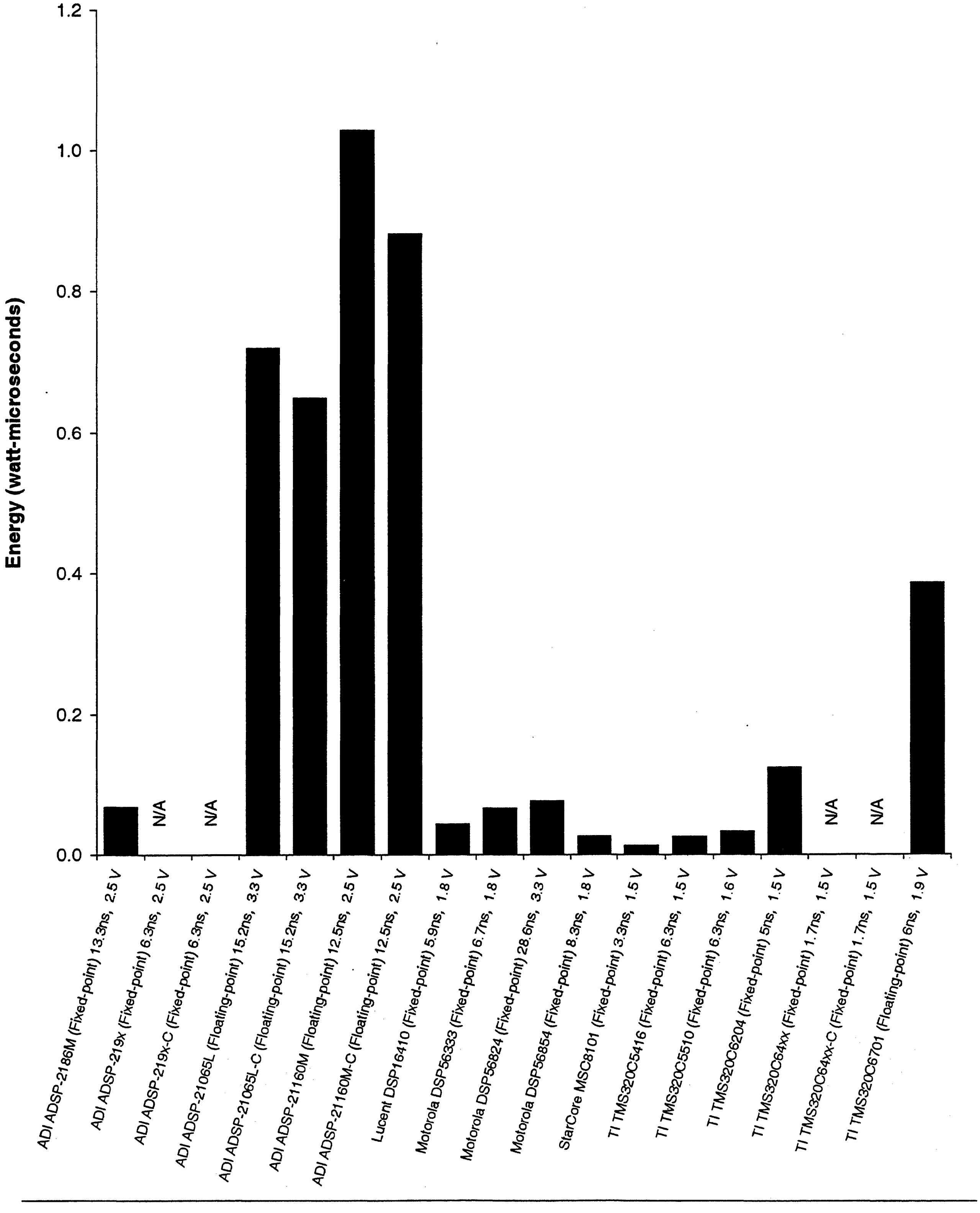


Figure 8.4-7. Energy Consumption for Vector Add (Lower is Better)

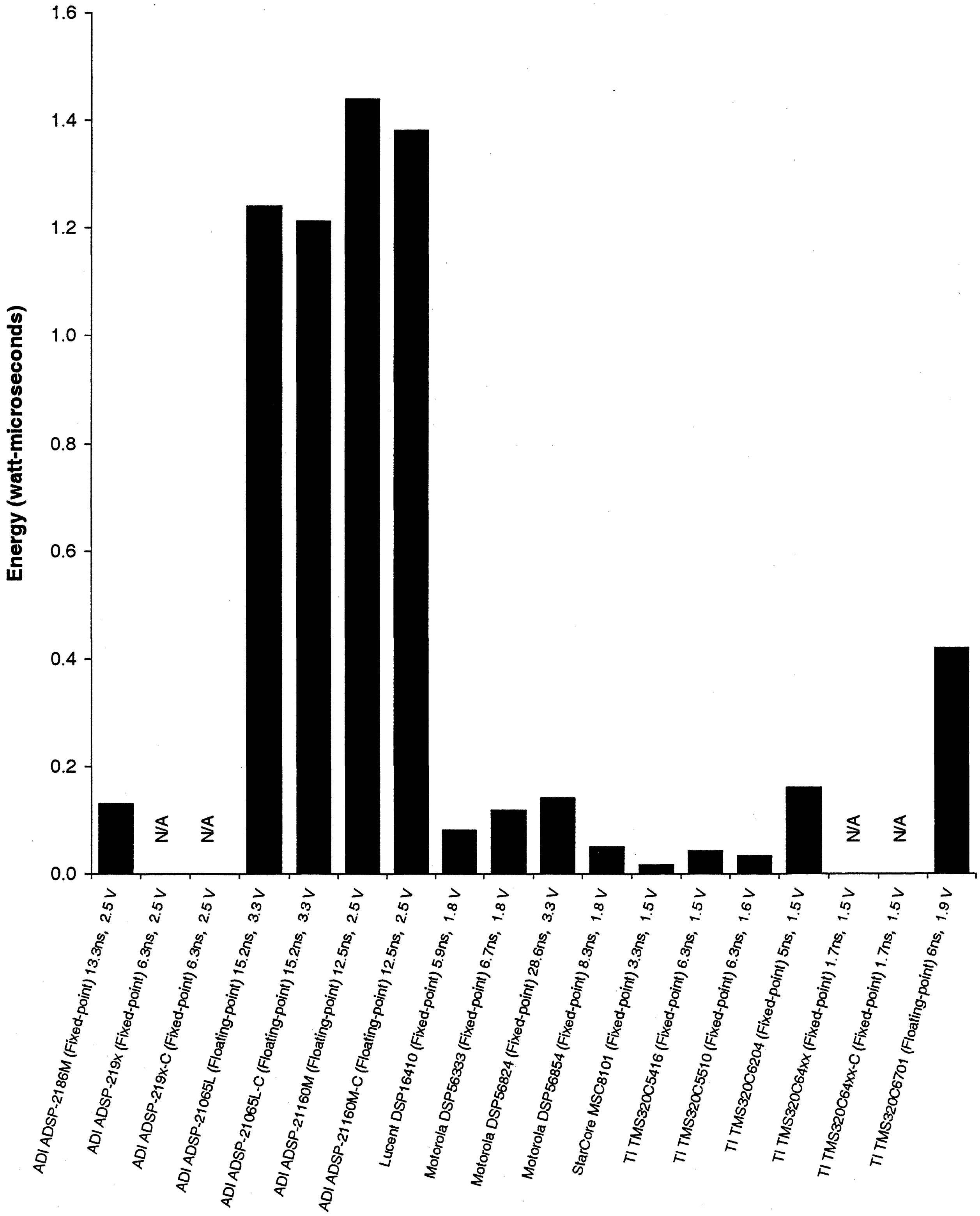


Figure 8.4-8. Energy Consumption for Vector Maximum (Lower is Better)

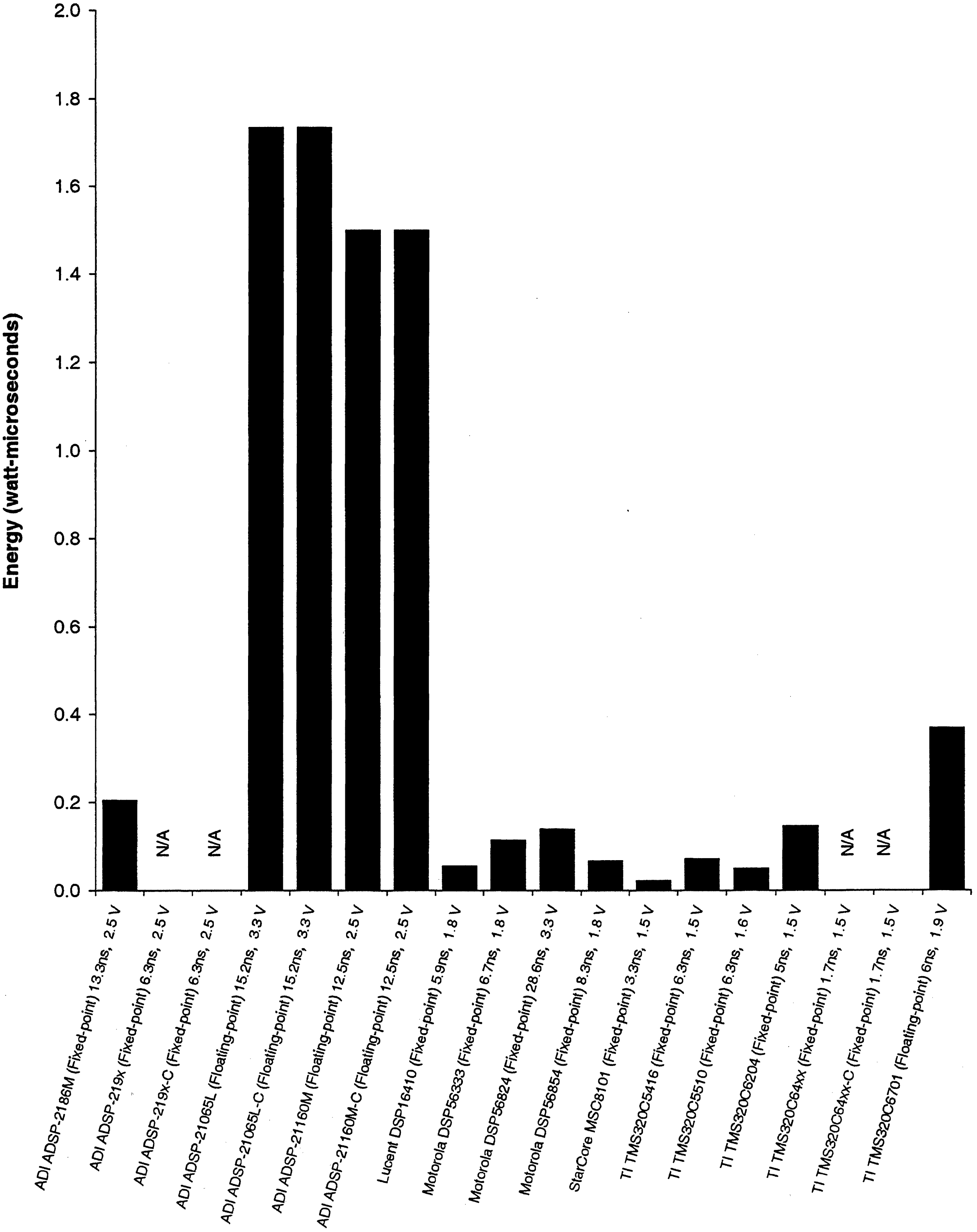


Figure 8.4-9. Energy Consumption for Control (Lower is Better)

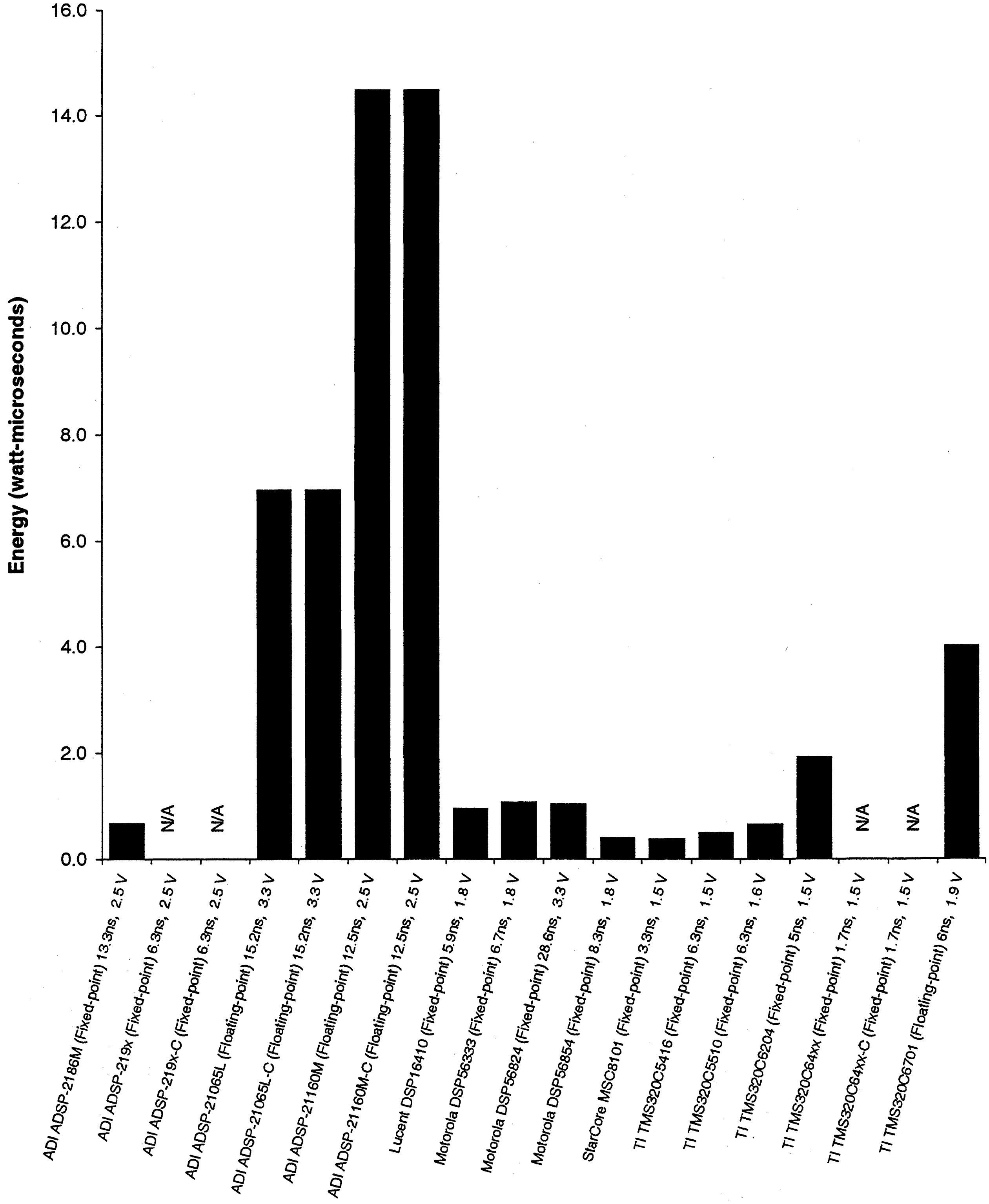


Figure 8.4-10. Energy Consumption for 256-Point FFT (Lower is Better)

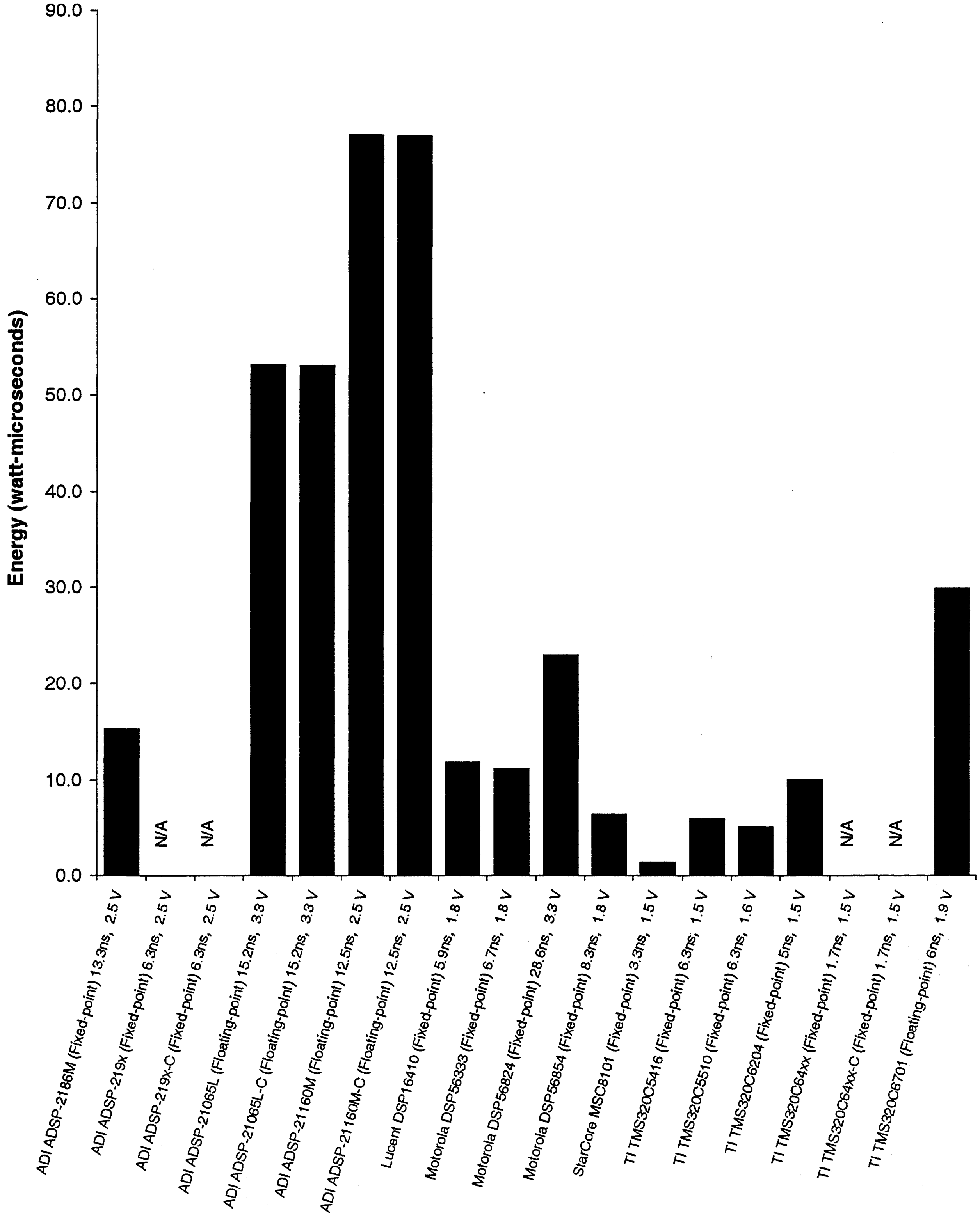


Figure 8.4-11. Energy Consumption for Viterbi (Lower is Better)

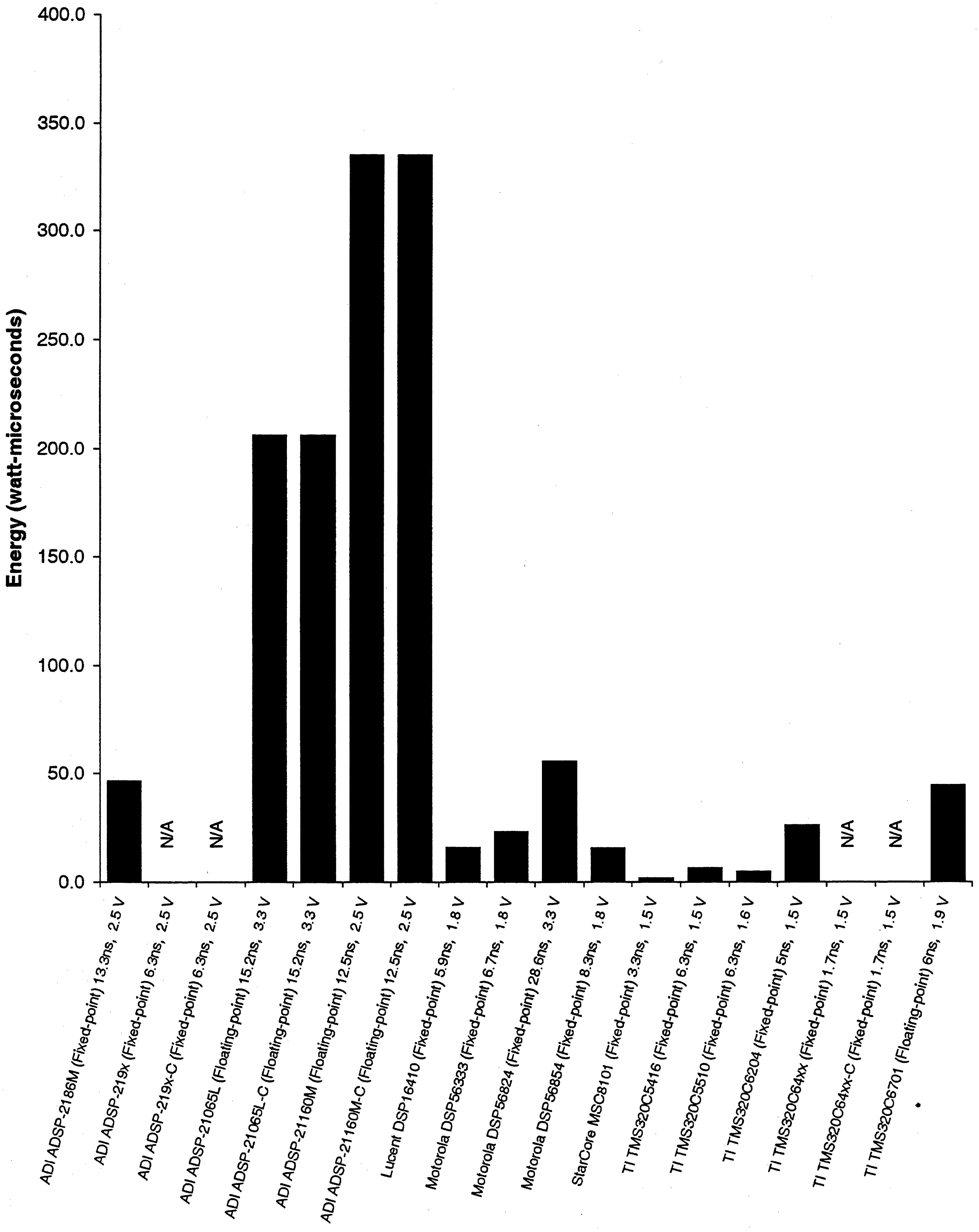


Figure 8.4-12. Energy Consumption for Bit Unpack (Lower is Better)

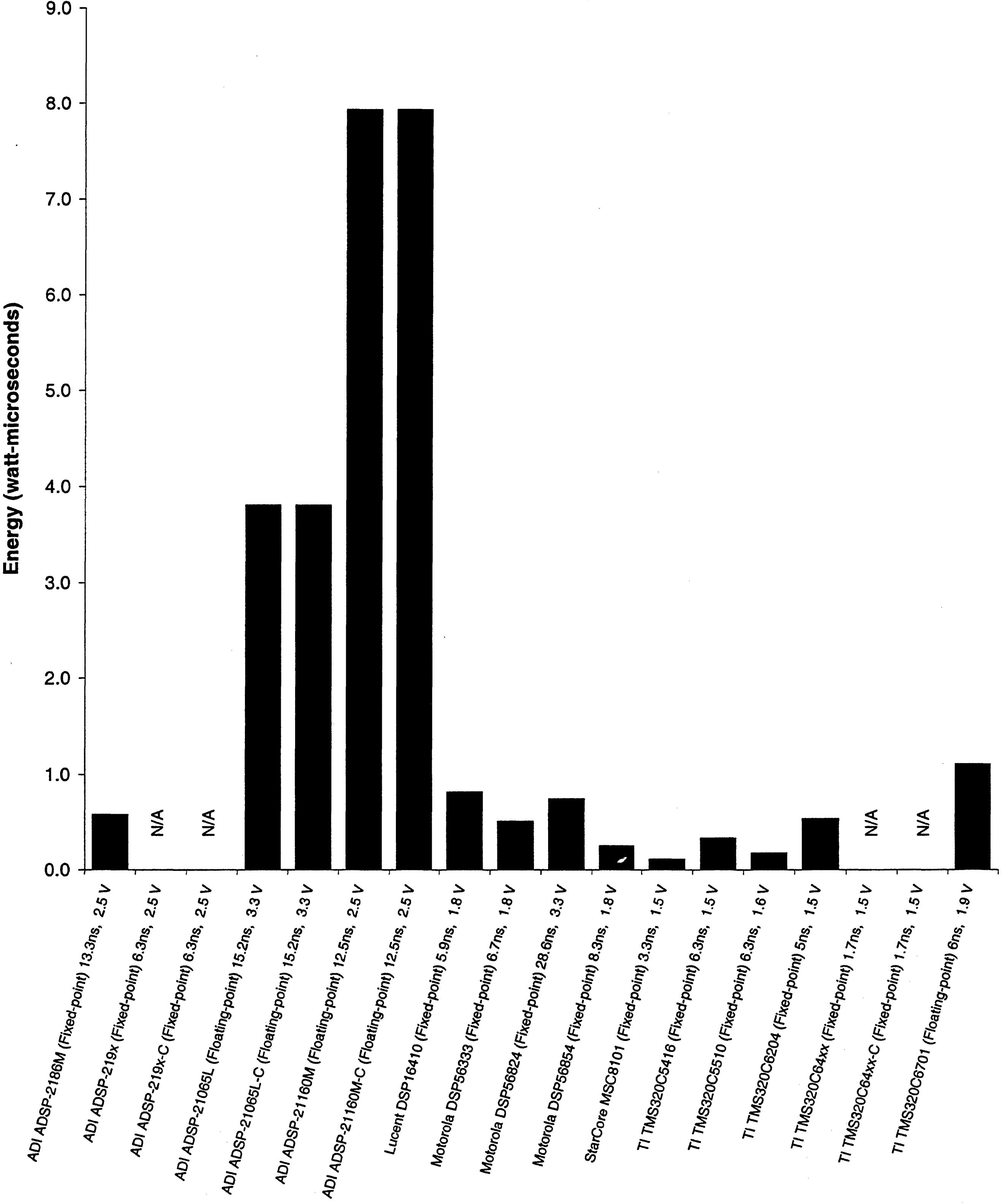


Figure 8.4-13A. Normalized Energy Consumption Sum for all b of E[p,b] / AE[b] (Lower is Better)

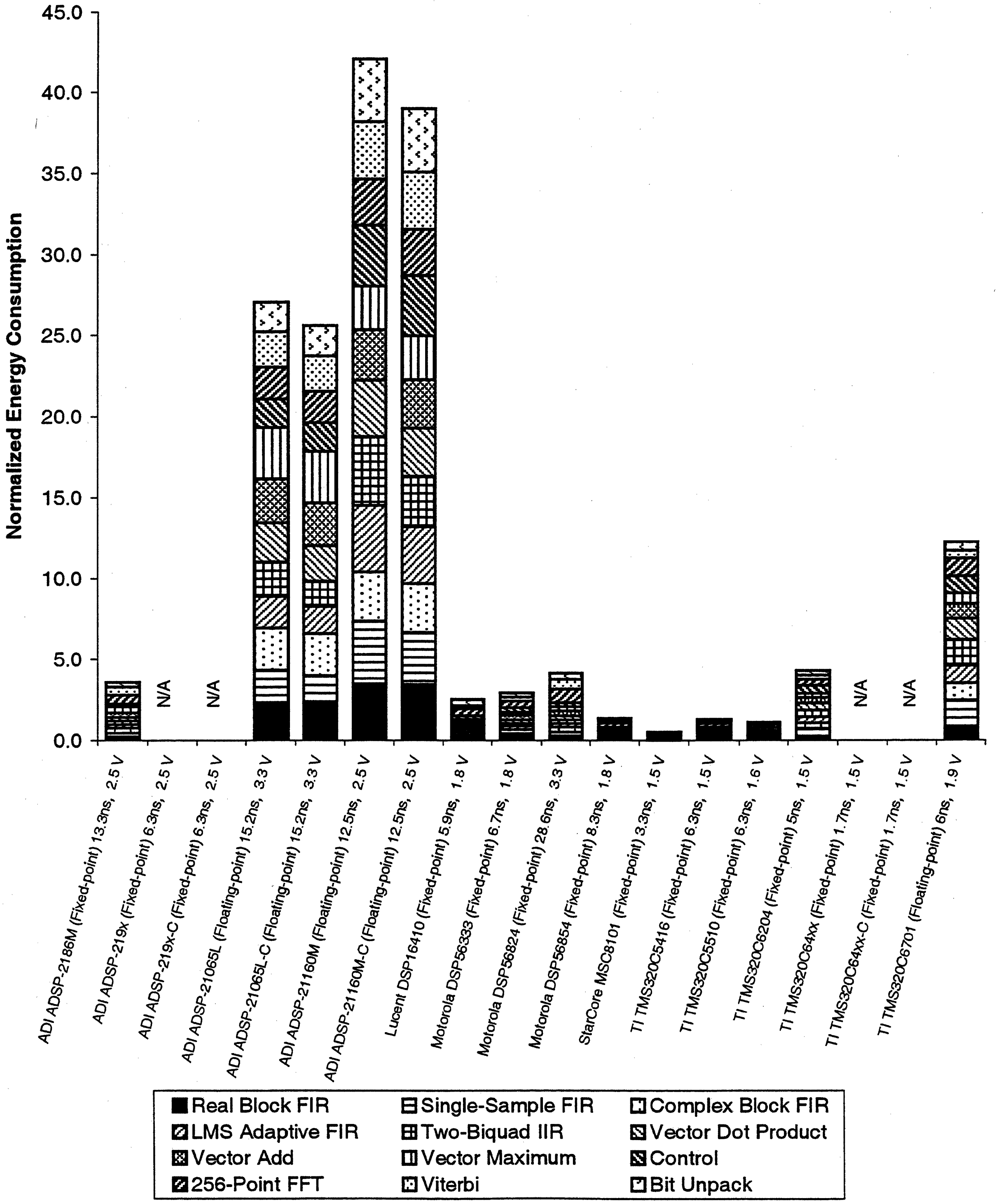
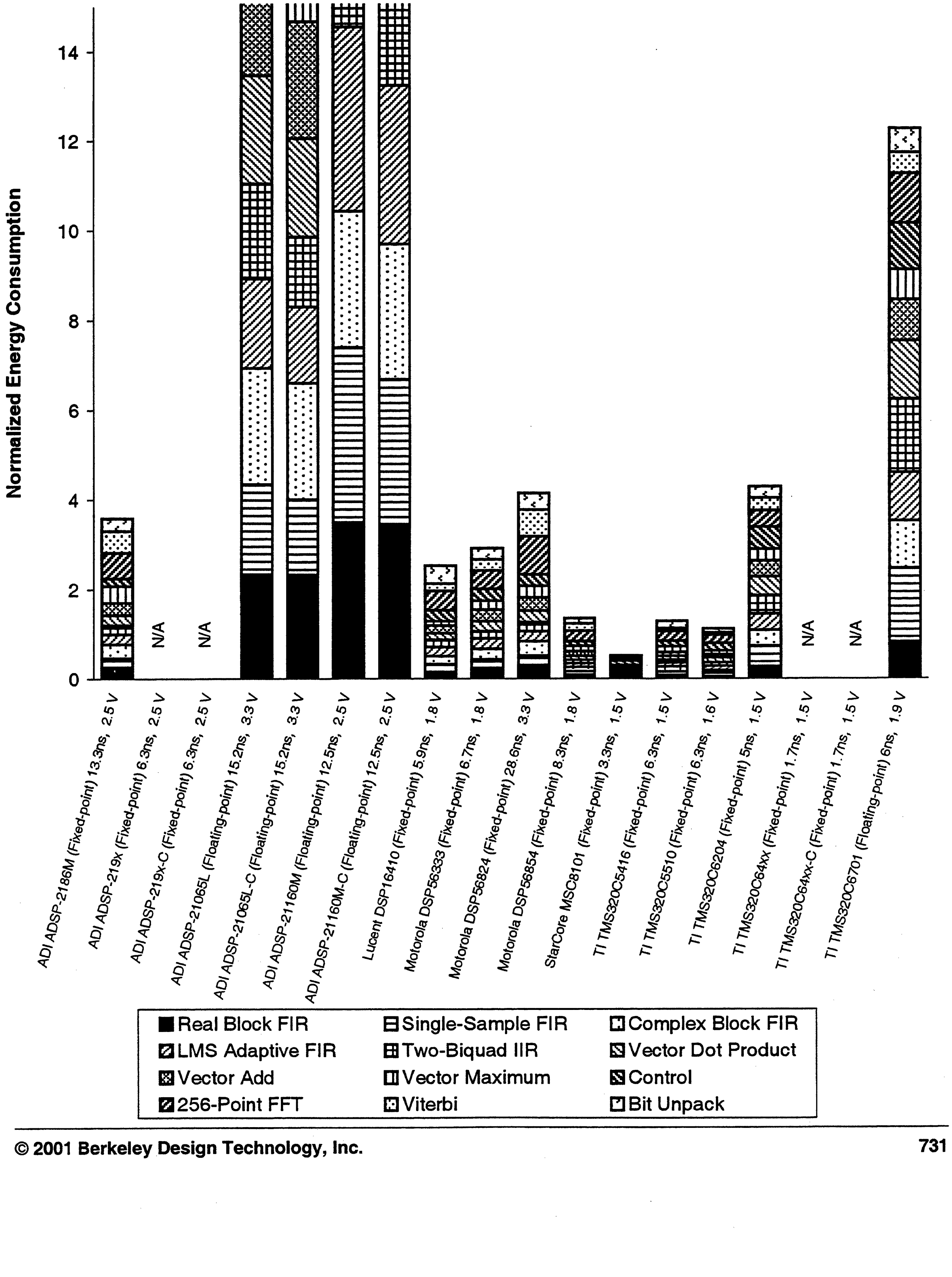


Figure 8.4-13B. Normalized Energy Consumption Sum for all b of E[p,b] / AE[b] (Close-Up) (Lower is Better)



8.5 Memory Usage Benchmarking

The preceding sections of our analysis have focused on evaluating processor execution speed and related figures of merit. However, the memory requirements of an application implementation can have significant impact on overall system cost. Additionally, if application code and data cannot fit entirely in on-chip memory, a significant performance degradation may occur on many processors. Because of these and other factors, memory use efficiency is an important metric for processor selection.

This section of our analysis focuses on processor memory use. We first examine memory use for the Control benchmark, a benchmark which is designed to be representative of typical control code. We next assess ROMable memory use in the BDTI Benchmarks™. Finally, we evaluate total memory use in the BDTI Benchmarks.

Control Benchmark Memory Use

The BDTI Benchmarks™ include one benchmark function specifically designed to evaluate memory use for control-oriented programs. Control-oriented code usually takes up the bulk of an application's code space but only a fraction of the application's processing time. Thus in control-oriented code, memory use is usually a more serious concern than execution speed.

As mentioned above, the Control benchmark replaces the FSM benchmark found in earlier versions of the BDTI Benchmarks.

The Control benchmark is a contrived series of operations common in control and decision processing. The benchmark consists of a "while" loop surrounding a "switch" statement, with the various switch cases performing bit manipulation, data moves, subroutine calls, and stack manipulations. The primary optimization goal for programmers implementing the Control benchmark is minimum memory use, and the secondary goal is speed of execution. (For all other BDTI Benchmarks, the priority of these goals is reversed.) Implementors are instructed to avoid optimizations that decrease the benchmark's execution time if the optimizations increase the benchmark's memory use.

Since the Control benchmark is designed to be representative of decision-making control code, memory use results on the Control benchmark are not necessarily indicative of processors' memory use on signal-processing-intensive tasks. Memory use on signal-processing-intensive tasks is evaluated via the other BDTI Benchmarks.

The two most important processor features that enable good Control benchmark memory performance are:

- **Short immediate data support**. The Control benchmark manipulates a number of relatively small constants. Processors that are able to embed short immediate data into an ALU operation instruction word have a distinct advantage here. For example, a processor that can encode the instruction:

```
CMP  #14,A
```

in a single instruction word typically uses less memory on this benchmark than one that requires multiple instruction words.

Some processors can use short immediate data only in move operations. These processors generally use less memory on the Control benchmark than those that cannot use short immediate data at all, but, due to the overhead of extra move instructions, their advantage is not as great as that of a processor that can use short immediate data as part of an ALU instruction.

- **PC-relative branch support**. The Control benchmark includes about a dozen branch instructions. Processors that are able to use short PC-relative branching (where the destination address is encoded as part of a single branch instruction word, in terms of an offset from the current program counter value) conserve memory related to branch instructions. This is in contrast to processors that must store the destination address as a separate instruction extension word. That is, a processor that can encode the instruction:

```
JLT  test20
```

in a single instruction word (by using a relative branch to destination "test20") generally uses less memory on this benchmark than one that requires two instruction words.

ROMable Memory Use

The cost of ROM is typically much lower than the cost of RAM of the same size and speed. Therefore, when evaluating the memory use of processors for the purpose of analyzing overall system cost, it is important to differentiate between instructions and data that remain constant, and thus can be placed in on- or off-chip ROM, and instructions and data that are modified in the course of executing the application, and thus must be placed in more expensive RAM. We refer to instructions and data that remain constant throughout the execution of an application as *ROMable.*

None of the BDTI Benchmark™ implementations use self-modifying code. Therefore, all program memory used in the BDTI Benchmarks is ROMable. We refer to the amount of memory needed to store benchmark program as *program memory use*. In addition, several benchmarks require filter coefficients or other data that is never modified. We refer to the amount of memory needed to store this data as *constant data memory use*.

Program Memory Use

Computationally intensive DSP algorithm code tends to be fairly compact by nature, so memory use for DSP algorithms is typically less important than memory use for control code. Hence, our analysis of program memory use focuses on the Control benchmark, described above. In all BDTI Benchmarks except the Control benchmark, benchmark code has been optimized for speed first and memory use second.

Programming style strongly affects program memory use. Loop unrolling that excessively increases program memory use is prohibited in the BDTI Benchmarks. However, to achieve low instruction cycle counts, loop unrolling and software pipelining are often used to a limited extent.

The most important architectural features that affect program memory use in DSP algorithms (as opposed to control tasks) are:

- **Instruction word widths**. On the processors we have benchmarked in this report, instruction word widths range from 16 bits to 48 bits. Most processors have fixed instruction word widths, but a few have variable instruction widths. Obviously, a processor with short instruction words uses less program memory per instruction than a processor with wide instruction words.
- **Operations performed within each instruction**. Many DSP processors allow several operations to be specified within a single instruction. The fewer operations that can be performed per instruction, the more instructions are generally required to perform a task, increasing program memory use.
- **Flexibility of instruction set**. On processors with very restricted instruction sets, it may be necessary to use more instructions to perform a given operation than on a processor with a more flexible instruction set. For example, some processors place many restrictions on which registers can be used as operand sources for certain instructions, which may force the programmer to insert extra instructions to move operands into the appropriate registers.

Processors that have flexible instruction sets or that allow several operations to be performed within each instruction often have larger instruction word widths than processors that do not provide such features. Thus, although processors with shorter instruction word widths use less memory per instruction than processors with wider instruction words, they generally use more instructions to accomplish a task.

Constant Data Memory Use

Many DSP algorithms such as fixed-coefficient filters and fast Fourier transforms rely on arrays of coefficients, "twiddle factors," or other data that is not modified. The two primary factors affecting constant data memory use are the data word width and the data set size. For example, processors with wider data words require more memory to store the same number of filter coefficients, and filters with a larger number of taps require more coefficients to be stored.

It is often possible to increase the execution speed of algorithms via the use of look-up tables, sacrificing constant data memory use for speed. In general, we have disallowed such optimizations in the BDTI Benchmarks. However, in a few cases we have allowed an array of filter coefficients to be duplicated in memory if doing so resulted in a significant improvement in speed. Occasionally, we have also allowed benchmark implementations to use a few extra words of constant data memory in order to boost performance. In deciding whether to permit such performance optimizations, we weighed the increase in memory use against the resulting performance gain, and judged whether this trade-off would be reasonable in a typical application.

Total Memory Use

Although many applications use ROM for all instructions and constant data, in some applications this is not possible. For example, in multimedia applications, memory may be dynamically allocated for instructions and data as needed for various multimedia functions. Thus, the system must contain enough RAM to hold the largest amount of instructions and data that may be in use simultaneously. In such applications, *total memory use* is a more useful metric than ROMable memory use.

Non-Constant Data Memory Use

In addition to the program code and constant data included in ROMable memory use as discussed above, total memory use also includes non-constant data. Non-constant data is any data that may need to be updated during execution of the benchmark. This includes delay lines, stack use, input/output buffers, and state variables that must be stored between invocations of the benchmark function.

Note that the amount of non-constant data used by a benchmark function, referred to as *non-constant data memory use*, complements ROMable memory use in two ways. First, non-constant data memory use accounts for the difference between ROMable memory use and total memory use. Second, non-constant data memory use represents the smallest amount of RAM (as opposed to ROM) needed to execute the benchmark.

Like constant data, the two primary factors affecting non-constant data memory use are the data word width and the data set size, although occasionally execution speed optimizations make use of additional non-constant data memory. As with constant data memory, in implementing the BDTI Benchmarks we weighed the increase in memory use of such optimizations against the improvement in performance, and only allowed an optimization if we felt this trade-off would be reasonable in a typical application.

Organization of Results

The memory usage data and analysis presented in this section are organized as follows:

- **Program memory use.**
  The program memory use for each processor on each benchmark is shown in Table 8.5-1 and Figures 8.5-1B through 8.5-12B. (The lower segment of each bar in the figures represents program memory use.) Program memory use includes program memory for the main benchmark body, but not for the power-up section.
- **Normalized program memory use.**
  As the first step in creating aggregate program memory use results that show each processor's overall program memory use, we normalize the program memory use

for each processor on each benchmark. This normalization is done by dividing each processor's memory use on a given benchmark by the average over all processors on that benchmark. This ensures that benchmarks that consume more memory by their nature are not automatically weighted more heavily when we aggregate the program memory use of each processor over all benchmarks. Normalized benchmark program memory use values are presented in Table 8.5-2.

- **Total normalized program memory use for all benchmarks.**
  Adding together the normalized benchmark program memory use for all benchmarks for each processor produces an overall program memory use measure for that processor. This sum effectively applies a uniform weighting to each benchmark for this section of the analysis. Without the normalization step, benchmarks that require more program memory would tend to be weighted more heavily in the overall results. The results are shown in Figure 8.5-13 and at the bottom of Table 8.5-2.
- **Constant data memory use.**
  Tables 8.5-3A and 8.5-3B and Figures 8.5-1B through 8.5-12B show each processor's constant data memory use on each benchmark. (The upper segment of each bar in the figures represents constant data memory usage.)
- **Normalized constant data memory use.**
  Normalized benchmark constant data memory use values for each processor on each benchmark are presented in Table 8.5-4. The normalization of the constant data memory use is performed in the same way as the normalization of program memory use. It should be noted that, on some benchmarks, one processor uses a small amount of constant data while all of the others use none. For example, on the Bit Unpack benchmark, the Texas Instruments TMS320C54xx uses two bytes of constant data while the other processors use none. This phenomenon tends to distort the normalized results. Hence, we focus on the non-normalized constant data memory use results, presented in Table 8.5-3B and Figures 8.5-1B through 8.5-12B, rather than on the normalized constant memory use results.
- **Total normalized constant data memory use for all benchmarks.**
  Adding together the normalized benchmark constant data memory use for all benchmarks for each processor produces an overall constant data memory use measure for that processor. This sum effectively applies a uniform weighting to each benchmark for this section of the analysis. Without the normalization step, benchmarks that require more constant data memory would tend to be weighted more heavily in the overall results. The results are shown in Figure 8.5-14 and at the bottom of Table 8.5-4. As explained above these results are prone to distortions because on some benchmarks most processors use no memory while some processors use a small amount of memory. Hence, we focus on the non-normalized constant data memory use results, presented in Table 8.5-3B and Figures 8.5-1B through 8.5-12B, rather than on the normalized constant memory use results.

- **Non-constant data memory use.**
  Table 8.5-5A and 8.5-5B and Figures 8.5-1A through 8.5-12A show each processor's non-constant data memory use on each benchmark. (The lower segment of each bar in the figures represents non-constant data memory usage.)
- **Normalized non-constant data memory use.**
  Normalized benchmark non-constant data memory use values for each processor on each benchmark are presented in Table 8.5-6. The normalization of the non-constant data memory use is performed in the same way as the normalization of program memory use.
- **Total normalized non-constant data memory use for all benchmarks.**
  Adding together the normalized benchmark non-constant data memory use for all benchmarks for each processor produces an overall non-constant data memory use measure for that processor. This sum effectively applies a uniform weighting to each benchmark for this section of the analysis. Without the normalization step, benchmarks that require more non-constant data memory would tend to be weighted more heavily in the overall results. The results are shown in Figure 8.5-15 and at the bottom of Table 8.5-6.
- **ROMable memory use.**
  Program memory use and constant data memory use are added together to compute each processor's ROMable memory use on each benchmark. These results are shown in Table 8.5-7. Figures 8.5-1B through 8.5-12B show each processor's ROMable memory use on each benchmark, divided into program memory and constant data memory components.
- **Normalized ROMable memory use.**
  Normalized ROMable memory use values for each processor on each benchmark are presented in Table 8.5-8. The normalization of the ROMable memory use values is performed in the same way as the normalization of program memory use.
- **Total normalized ROMable memory use for all benchmarks.**
  Adding together the normalized benchmark ROMable memory use for all benchmarks for each processor produces an overall ROMable memory use measure for that processor. This sum effectively applies a uniform weighting to each benchmark for this section of the analysis. Without the normalization step, benchmarks that require more ROMable memory would tend to be weighted more heavily in the overall results. The results are shown in Figure 8.5-16 and at the bottom of Table 8.5-8.
- **Total memory use.**
  Non-constant data memory use, constant data memory use, and program memory use are added together to compute each processor's total memory use on each benchmark. These results are shown in Table 8.5-9. Figures 8.5-1A through 8.5-12A show the total memory use for each processor on each benchmark,

divided into non-constant data memory, constant data memory, and program memory components.

- **Normalized total memory use.**
  Normalized total memory use values for each processor on each benchmark are presented in Table 8.5-10. The normalization of the total memory use values is performed in the same way as the normalization of program memory use.
- **Overall normalized total memory use for all benchmarks.**
  Adding together the normalized benchmark total memory use for all benchmarks for each processor produces an overall total memory use measure for that processor. This sum effectively applies a uniform weighting to each benchmark for this section of the analysis. Without the normalization step, benchmarks that require more memory would tend to be weighted more heavily in the overall results. The results are shown in Figure 8.5-17 and at the bottom of Table 8.5-10.

As explained earlier in this chapter, for some processors that use caches, we report two sets of results. Memory usage is not affected by the state of the caches. For consistency throughout the report, however, we include the memory usage for these processors in both the non-cache-preloaded versions and the cache-preloaded versions. In calculating the averages of memory usage, though, we count processors with caches as a single processor.

Analysis of Results

Control Benchmark Memory Use

Table 8.5-9 shows processor total memory use (in bytes) for all benchmarks, including the Control benchmark. Figure 8.5-9A presents this information graphically. In the Control benchmark, most processors use only 4-12 bytes of non-constant data memory and no constant data memory. Hence, program memory represents the vast majority of the memory used in this benchmark on each processor. It is therefore not surprising that processors with shorter instruction words generally use less memory in this benchmark than processors with larger instruction words.

The three processors with the lowest Control benchmark memory use results all use 16-bit instructions words. These are the Motorola DSP568xx and DSP5685x, and the StarCore SC140 (which uses 16-bit instructions with optional prefixes that can add 16 or 32 bits to the length of a group of parallel instructions).

The key features that enable the DSP5685x to achieve good code density on control-oriented code are its 16-bit instruction word width, PC-relative branches, and support for short immediate data. The second-lowest total memory usage for the Control benchmark is achieved by the DSP568xx, which requires slightly more program memory. Compared with the DSP568xx, the DSP5685x gains the advantage here due to its ability to load and store long (32-bit) data in one instruction.

The low Control benchmark memory use of the SC140 is particularly noteworthy considering the processor's very high speed. Prior to the SC140, VLIW-based DSPs have generally had relatively high memory use. The main reason for the lower memory use of the SC140 is its efficient encoding of instructions, using 16-bit instructions with optional 16- or 32-bit prefixes that are shared among the instructions in an execution set. For example, on most processors, the loop start and end addresses for hardware loops are either written into a register or embedded into an instruction (in which case the processor writes the values into registers that are invisible to the programmer). For a processor with 16-bit program memory addresses, this results in usage of at least two 16-bit words. These two 16-bit words are also used in the SC140, but they are better utilized because they also carry conditional execution information. Further, in some cases, this conditional execution information can be used to reduce the number of branches—and hence, the number of instructions required to implement the benchmark.

The two worst performers on this benchmark, the ADSP-2106x and ADSP-2116x, use the same code for the Control benchmark. The reason for their high memory usage is that the ADSP-2106x and ADSP-2116x use 48-bit instruction words. To compete with processors with 16-bit instruction words, these processors would have to implement the benchmark using one third as many instructions as the processors with 16-bit instructions. Although the processors' wide instruction words allow flexibility for performing several operations within each instruction, on the Control benchmark this does not make up for the wider instruction words.

The Texas Instruments TMS320C62xx and TMS320C67xx use the same implementation for this benchmark, and have relatively high memory use. This is primarily due to the processors' 32-bit instruction width and the fact that their instructions are fairly simple. Because the Control benchmark is optimized for memory usage rather than speed, the processors do not use a deep software pipeline in this benchmark. Unfortunately, without deep software pipelining, multi-cycle NOP instructions must sometimes be used to fill branch delay slots.

Compared to the TMS320C62xx, the Control benchmark memory use of the TMS320C64xx is about 15% lower. Two architectural changes explain this. First, the TMS320C64xx implementation takes advantage of the new "BNOP" (branch followed by multi-cycle NOP) and "ADDKPC" (save program counter followed by multi-cycle NOP) instructions. Second, on the TMS320C64xx, unlike the TMS320C62xx, groups of instructions to be executed in parallel (referred to as an "execute packet") aren't restricted to be entirely contained in a group of instructions read from memory at one time (referred to as a "fetch packet"). On the TMS320C62xx, an execute packet that would otherwise cross a fetch packet boundary is automatically aligned by the compiler or assembler using NOPs. The NOP instructions are executed in parallel with the other instructions in the execute packet, and don't affect execution time. They do, however, consume program memory.

Note that while the Texas Instruments TMS320C55xx maintains assembly source code compatibility with the older TMS320C54xx, the TMS320C55xx introduces a number of new instructions. The TMS320C55xx achieves a Control benchmark memory use approximately 30% lower than that of the TMS320C54xx, mainly due to its support for PC-relative branches and short immediate constants for arithmetic and logical operations. Although the TMS320C55xx also supports variable-length instructions from 8- to 48-bits long (compared with a fixed 16-bit instruction word length for the TMS320C54xx), in control code the TMS320C55xx mostly uses 16-bit instructions, so this feature does not significantly reduce control code memory use.

ROMable Memory Use

This section presents a brief analysis of the benchmark ROMable memory use results, summarized in Table 8.5-7, Figures 8.5-1B through 8.5-12B, and Figure 8.5-16. Note that in the figures, the ROMable memory use results are divided into program memory and constant data memory components. We first discuss program memory use, then constant data memory use, and finally we combine the two and discuss the overall ROMable memory use of each processor.

When reviewing the total normalized program memory use results presented in Figure 8.5-13, note that the 12 benchmarks are weighted uniformly. This weighting is arbitrary and may not be appropriate for any particular application. For most applications, the Control benchmark should be weighted heavily since control code generally makes up the bulk of an application's program memory use.

In benchmarks other than the Control benchmark, fixed-point conventional DSPs generally have the lowest memory use. This is due in part to their relatively small, 16- or 24-bit instruction word sizes. A more important factor, however, is the relatively simple nature of these processors, including fairly shallow pipelines, and execution of only one instruction at a time. In contrast, on more complex processors that execute multiple instructions in parallel, DSP algorithm implementations typically execute multiple instances of the same instruction in parallel (for example, four MAC instructions), which consumes more program memory. Similarly, in processors with deep pipelines, extensive software pipelining is typically used in DSP algorithm implementations to minimize the effects of pipeline latencies; this amounts to duplicating sequences of instructions, and consumes program memory. These effects are particularly evident in the Texas Instruments TMS320C62xx, TMS320C64xx, and TMS320C67xx. These processors have the highest total normalized program memory use results, approximately double those of the processors with the next-highest results. In addition to the pipeline and parallel instruction execution factors described above, the program memory use of these Texas Instruments processors is increased due to the processors' relatively simple, 32-bit instructions.

The total normalized program memory usage of the TMS320C64xx is about 15% lower than that of the TMS320C62xx, for two reasons. First, unlike on the TMS320C62xx, NOPs aren't required to align execution packets on the TMS320C64xx, increasing code density. Second, in general, the higher parallelism of TMS320C64xx instructions increases code density because the TMS320C64xx can often accomplish more work in a single instruction than can the TMS320C62xx. It should be noted, though, that on some benchmarks the TMS320C64xx uses *more* program memory than does the TMS320C62xx. Both processors' memory usage is increased due to frequent use of loop unrolling to optimize the benchmarks; however, this technique increases program memory usage more on the TMS320C64xx than on the TMS320C62xx. This is because, compared to the TMS320C62xx, the higher level of parallelism on the TMS320C64xx motivates programmers to unroll loops more than with the TMS320C62xx. For this reason, four benchmarks out of twelve require more program memory on the TMS320C64xx than on the TMS320C62xx.

Among conventional DSPs, the Analog Devices ADSP-2106x has an unusually high total normalized program memory use result, owing to its unusually long 48-bit instructions. Excluding the ADSP-2106x, the total normalized program memory use results of the conventional DSPs benchmarked here lie within a narrow range, from approximately 6 for the Texas Instruments TMS320C54xx to approximately 7.5 for the Motorola DSP563xx.

Shifting our attention to constant data memory use, it should be noted that, on some benchmarks, one processor uses a small amount of constant data while all of the others use none. For example, on the Bit Unpack benchmark, the Texas Instruments TMS320C54xx uses two bytes of constant data while the other processors use none. This phenomenon tends to distort the normalized results. Hence, we focus on the non-normalized constant data memory use results, presented in Table 8.5-3B and Figures 8.5-1B through 8.5-12B.

On most benchmarks, all benchmarked processors use the same number of data *words*. Of course, the size of a word varies from one processor to another: the benchmarked floating-point DSPs use 32-bit words, and all of the benchmarked fixed-point DSPs use 16-bit words except for the Motorola DSP563xx, which uses 24-bit words. Thus, in most cases, the most important factor in determining a processor's constant data memory use is its data word size. Processors with 24-bit data words generally use about 1.5 times the constant data memory of processors with 16-bit data words, while processors with 32-bit data words generally use about twice as much constant data memory as processors with 16-bit data words. There are some exceptions to this, though. In particular, constant data memory use on the FFT does vary significantly from processor to processor, even after accounting for different data word sizes. These variations are a consequence of particular speed-oriented optimizations made by the benchmark implementors. Different optimizations are effective on different processors, depending on the details of the processor's capabilities, and different optimizations use different amounts of constant data memory.

For example, on the FFT benchmark, the constant data memory use of the Texas Instruments TMS320C54xx is roughly half that of most of the other processors benchmarked here. On this benchmark, the TMS320C54xx takes advantage of symmetries in the twiddle factors (FFT coefficients) to reduce the amount of constant data memory required, without increasing execution time.

Generally speaking, a processor's constant data memory use can be deduced by knowing the benchmark parameters: for example, if a processor with 16-bit data processes *T* taps in the real block FIR filter benchmark, the processor will require 2*T* bytes of constant data memory for the coefficients. The ADSP-2116x, which uses 32-bit floating-point data, is an exception to this rule. On the real block FIR filter and the Single-Sample FIR filter benchmarks, the ADSP-2116x uses two copies of the filter coefficient array in order to better take advantage of its SIMD architecture. The ADSP-2116x therefore uses twice as much constant data memory on these benchmarks as other DSPs with 32-bit data sizes.

When program memory and constant data memory are combined to yield the ROMable memory use results, the overall normalized results (shown in Figure 8.5-16) are very similar to the overall normalized program memory use results (shown in Figure 8.5-13). This is because seven of the benchmarks use no constant data memory on most processors, and most of the remaining five benchmarks show little variation in constant data memory use.

Comparing the benchmarked processors' overall normalized ROMable memory use, the Texas Instruments TMS320C64xx, TMS320C62xx, and TMS320C67xx have the highest results by a wide margin, due to their very high program memory use, discussed above. (TMS320C67xx ROMable memory use is higher than that of the TMS320C62xx and TMS320C64xx due to higher program memory use and higher data memory use; the latter is due to the fact that the TMS320C67xx uses 32-bit data.) Similarly, the Analog Devices ADSP-2106x and ADSP-2116x have high total normalized ROMable memory use due to their large 48-bit instruction words that increase program memory use, and their 32-bit data word size. The ADSP-2116x is further hindered by the fact that it requires some additional code in order to combine data from its two data paths, and also requires additional constant data as described above.

The processors with the lowest total normalized ROMable memory use are all conventional DSPs: the Texas Instruments TMS320C54xx, the Analog Devices ADSP-218x and ADSP-219x, and the Motorola DSP5685x and DSP568xx. The VLIW-based Texas Instruments TMS320C55xx and the enhanced conventional Lucent DSP164xx also have fairly low total normalized ROMable memory use.

Total Memory Use

This section presents a brief analysis of the benchmark total memory use results summarized in Table 8.5-9, Figures 8.5-1A through 8.5-12A, and Figure 8.5-17. Note that in the figures the total memory use results are divided into non-constant data memory, constant data memory, and program memory components. In this section, we focus our discussion on non-constant data memory use. Constant data memory use and program memory use are discussed above in the analysis of ROMable memory use. We conclude by discussing how non-constant data memory and ROMable memory use combine to yield the overall total memory use of each processor.

Overall normalized non-constant data memory use results are presented in Figure 8.5-15. As with constant data memory use, native data word width is the primary factor determining each processor's non-constant data memory use. Thus, the floating-point DSPs (the Texas Instruments TMS320C67xx and Analog Devices ADSP-2106x and ADSP-2116x) have notably high non-constant data memory use results due to their large 32-bit floating-point native data format. Similarly, the Motorola DSP563xx uses about 50% more non-constant data memory than most 16-bit, fixed-point DSPs due to its 24-bit data word size.

Nearly all of the 16-bit, fixed-point processors have very similar total normalized non-constant data memory results, ranging from approximately 9.3 for the Texas Instruments TMS320C55xx to approximately 10.4 for the Texas Instruments TMS320C62xx. The StarCore SC140 has a total normalized non-constant data memory result that is about 20% higher than those of the other 16-bit fixed-point DSPs. This is due to the SC140's relatively high non-constant memory use on the Control benchmark, which in turn is due to the way in which stack operations work on the SC140. The SC140 can group one or two PUSH instructions in an execution set, and in either case the stack pointer is incremented only once by 8 bytes. However, executing multiple instructions in parallel takes up more memory than issuing each instruction by itself. Thus, there is a trade-off between program memory and non-constant data memory. If the PUSH instructions are issued separately (as is done in the Control benchmark), the size of the required non-constant data memory is significantly increased. Nevertheless, since non-constant data memory use in this benchmark is small relative to program memory use, the SC140's total memory use on the Control benchmark is among the lowest of the benchmarked processors, as discussed above.

Next we discuss the overall normalized total memory use results presented in Figure 8.5-17. Note that in some benchmarks, such as the Complex Block FIR, data makes up the bulk of the benchmark's total memory use on most processors. This is not surprising, since many DSP algorithms are implemented using small loops that process large amounts of data. Note, however, the very large influence of data memory use on total memory use is not typical of most DSP applications. Recall that the normalized total memory use results weight all of the benchmarks uniformly. In contrast, control code accounts for the bulk of program memory use in most applications, so greater weight would typically be appropriate for the Control benchmark compared to other benchmarks. Since the Control benchmark typically uses no constant data and very little non-constant data, a more appropriate weighting would decrease the influence of data memory use on total memory use, making program memory use the dominant factor. Nevertheless, the overall normalized total memory use results reveal a few important points.

The TMS320C67xx takes last place in terms of overall normalized total memory use because it has both extremely high program memory use and a 32-bit, floating-point data format. The TMS320C62xx ties (with the ADSP-2116x) for second-to-last place in terms of overall normalized total memory use despite its 16-bit native data word width, mostly due to its extremely high program memory use. The Analog Devices ADSP-2116x is penalized by its 32-bit, floating-point data format and moderately high program memory use.

The Motorola DSP5685x has the lowest overall normalized total memory use, followed by the Motorola DSP568xx. The low total memory use of these processors is due to their low program memory use and 16-bit native data word width. The Lucent DSP164xx also has a very low overall normalized total memory use result. While the DSP164xx has 16- and 32-bit instructions, most instructions are 16-bits long. This, combined with the processor's 16-bit data word size, explains its low overall normalized total memory use result. Note that several other processors have overall normalized total memory use results just slightly higher than those of the three processors just mentioned. The Texas Instruments TMS320C55xx and TMS320C54xx and the Analog Devices ADSP-218x and ADSP-219x all have overall normalized total memory use results within 10% of that of the DSP5685x.

Table 8.5-1. Program Memory Use

(MP[p,b], bytes) (cont.)

DSP Building Block Functions	ADI 218x	ADI 219x	ADI 219x-C	ADI 2106x	ADI 2106x-C	ADI 2116x	ADI 2116x-C	Lucent 164xx
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	
Real Block FIR	66	108	108	120	120	168	168	90
Single-Sample FIR	48	60	60	78	78	114	114	46
Complex Block FIR	114	114	114	186	186	216	216	98
LMS Adaptive FIR	66	84	84	186	186	186	186	102
Two-Biquad IIR	48	48	48	72	72	72	72	40
Vector Dot Product	18	18	18	54	54	78	78	20
Vector Add	24	24	24	54	54	66	66	32
Vector Maximum	51	42	42	66	66	84	84	56
Control	213	222	222	342	342	342	342	182
256-Point FFT	255	255	255	1020	1020	1074	1074	302
Viterbi	288	306	306	474	474	486	486	188
Bit Unpack	57	57	57	132	132	132	132	102

Table 8.5-2. Normalized Program Memory Use

(MP[p,b]/AMP[b], bytes) (cont.)

DSP Building Block Functions	ADI 218x	ADI 219x	ADI 219x-C	ADI 2106x	ADI 2106x-C	ADI 2116x	ADI 2116x-C	Lucent 164xx
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	
Real Block FIR	0.32	0.53	0.53	0.59	0.59	0.82	0.82	0.44
Single-Sample FIR	0.54	0.68	0.68	0.88	0.88	1.28	1.28	0.52
Complex Block FIR	0.53	0.53	0.53	0.86	0.86	1.00	1.00	0.45
LMS Adaptive FIR	0.34	0.44	0.44	0.97	0.97	0.97	0.97	0.53
Two-Biquad IIR	0.68	0.68	0.68	1.02	1.02	1.02	1.02	0.57
Vector Dot Product	0.29	0.29	0.29	0.86	0.86	1.24	1.24	0.32
Vector Add	0.44	0.44	0.44	1.00	1.00	1.22	1.22	0.59
Vector Maximum	0.44	0.37	0.37	0.57	0.57	0.73	0.73	0.49
Control	0.97	1.01	1.01	1.55	1.55	1.55	1.55	0.83
256-Point FFT	0.39	0.39	0.39	1.55	1.55	1.63	1.63	0.46
Viterbi	0.58	0.62	0.62	0.95	0.95	0.98	0.98	0.38
Bit Unpack	0.55	0.55	0.55	1.27	1.27	1.27	1.27	0.98
Total	6.07	6.50	6.50	12.07	12.07	13.71	13.71	6.55

Notes:

[1] Memory use is the same for both cached and non-cached versions of the chip. Both versions are presented to be consistent with earlier sections, but only one result from each chip is used for normalized calculations.

[2] Projected; as of this writing, these processors were not yet sampling.

Table 8.5-1. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]					[1,2]	[1,2]	
84	64	60	250	54	99	568	624	624	496
45	30	28	114	30	38	176	180	180	256
114	96	82	246	76	103	536	568	568	480
105	46	60	174	66	49	640	248	248	680
42	40	42	86	34	74	140	100	100	148
21	14	14	60	16	22	168	120	120	260
42	24	24	56	36	35	112	100	100	128
30	32	56	176	72	106	288	316	316	232
210	128	122	122	222	152	288	248	248	288
318	578	464	604	236	401	1244	1000	1000	1488
651	926	734	476	248	249	576	460	460	896
72	56	44	112	82	60	184	184	184	184

Table 8.5-2. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]					[1,2]	[1,2]	
0.41	0.31	0.29	1.23	0.27	0.49	2.79	3.06	3.06	2.44
0.51	0.34	0.32	1.28	0.34	0.43	1.98	2.03	2.03	2.88
0.53	0.44	0.38	1.14	0.35	0.48	2.48	2.63	2.63	2.22
0.55	0.24	0.31	0.90	0.34	0.25	3.33	1.29	1.29	3.54
0.60	0.57	0.60	1.22	0.48	1.05	1.99	1.42	1.42	2.10
0.33	0.22	0.22	0.95	0.25	0.35	2.66	1.90	1.90	4.12
0.78	0.44	0.44	1.04	0.67	0.65	2.07	1.85	1.85	2.37
0.26	0.28	0.49	1.53	0.63	0.92	2.51	2.75	2.75	2.02
0.95	0.58	0.55	0.55	1.01	0.69	1.31	1.13	1.13	1.31
0.48	0.88	0.70	0.92	0.36	0.61	1.89	1.52	1.52	2.25
1.31	1.86	1.48	0.96	0.50	0.50	1.16	0.93	0.93	1.80
0.69	0.54	0.42	1.08	0.79	0.58	1.77	1.77	1.77	1.77
7.40	6.71	6.21	12.80	5.98	6.99	25.93	22.27	22.27	28.82

Table 8.5-3A. Constant Data Memory Use Formulas

(MC[p,b], bytes) (cont.)

DSP Building Block Functions	ADI 218x	ADI 219x	ADI 219x-C	ADI 2106x	ADI 2106x-C	ADI 2116x	ADI 2116x-C	Lucent 164xx
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	
Real Block FIR	2*T	2*T	2*T	4*T	4*T	4*T	4*T	2*T
Single-Sample FIR	2*T	2*T	2*T	4*T	4*T	4*T	4*T	2*T
Complex Block FIR	4*T	4*T	4*T	8*T	8*T	8*T	8*T	2*(T*2)
LMS Adaptive FIR	0	0	0	0	0	0	0	0
Two-Biquad IIR	16	16	16	32	32	32	32	16
Vector Dot Product	0	0	0	0	0	0	0	0
Vector Add	0	0	0	0	0	0	0	0
Vector Maximum	0	0	0	0	0	0	0	0
Control	0	0	0	0	0	0	0	0
256-Point FFT	640	640	640	1536	1536	1536	1536	1000
Viterbi	0	0	0	0	0	0	0	0
Bit Unpack	0	0	0	0	0	0	0	0

T=# of Taps

Table 8.5-3B. Constant Data Memory Use

(MC[p,b], bytes) (cont.)

DSP Building Block Functions	ADI 218x	ADI 219x	ADI 219x-C	ADI 2106x	ADI 2106x-C	ADI 2116x	ADI 2116x-C	Lucent 164xx
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	
Real Block FIR	32	32	32	64	64	64	64	32
Single-Sample FIR	32	32	32	64	64	64	64	32
Complex Block FIR	64	64	64	128	128	128	128	64
LMS Adaptive FIR	0	0	0	0	0	0	0	0
Two-Biquad IIR	16	16	16	32	32	32	32	16
Vector Dot Product	0	0	0	0	0	0	0	0
Vector Add	0	0	0	0	0	0	0	0
Vector Maximum	0	0	0	0	0	0	0	0
Control	0	0	0	0	0	0	0	0
256-Point FFT	640	640	640	1536	1536	1536	1536	1000
Viterbi	0	0	0	0	0	0	0	0
Bit Unpack	0	0	0	0	0	0	0	0

Substituted Values: T = 16

Notes:

[1] Memory use is the same for both cached and non-cached versions of the chip. Both versions are presented to be consistent with earlier sections, but only one result from each chip is used for normalized calculations.

[2] Projected; as of this writing, these processors are not yet sampling.

Table 8.5-3A. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]					[1,2]	[1,2]	
3*T	2*T	2*(T+1)	2*T	2*T	2*T	2*T	2*T	2*T	4*T
3*T	2*T	2*T	2*T	2*T	2*T+8	2*T	2*T	2*T	4*T
6*T	4*T	4*T	4*T	4*T	4*T	4*T	4*T	4*T	8*T
0	0	0	0	0	0	0	0	0	0
24	16	16	16	16	16	16	16	16	32
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
768	764	764	812	384	512	800	1024	1024	1568
0	0	0	0	0	2	0	0	0	0
0	0	0	0	2	0	0	0	0	0

Table 8.5-3B. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]					[1,2]	[1,2]	
48	32	34	32	32	32	32	32	32	64
48	32	32	32	32	40	32	32	32	64
96	64	64	64	64	64	64	64	64	128
0	0	0	0	0	0	0	0	0	0
24	16	16	16	16	16	16	16	16	32
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
768	764	764	812	384	512	800	1024	1024	1568
0	0	0	0	0	2	0	0	0	0
0	0	0	0	2	0	0	0	0	0

Table 8.5-4. Normalized Constant Data Memory Use

(MC[p,b]/AMC[b], bytes) (cont.)

DSP Building Block Functions	ADI 218x	ADI 219x	ADI 219x-C	ADI 2106x	ADI 2106x-C	ADI 2116x	ADI 2116x-C	Lucent 164xx
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	
Real Block FIR	0.80	0.80	0.80	1.59	1.59	1.59	1.59	0.80
Single-Sample FIR	0.79	0.79	0.79	1.58	1.58	1.58	1.58	0.79
Complex Block FIR	0.80	0.80	0.80	1.60	1.60	1.60	1.60	0.80
LMS Adaptive FIR	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Two-Biquad IIR	0.80	0.80	0.80	1.60	1.60	1.60	1.60	0.80
Vector Dot Product	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Vector Add	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Vector Maximum	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Control	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
256-Point FFT	0.70	0.70	0.70	1.69	1.69	1.69	1.69	1.10
Viterbi	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Bit Unpack	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Total	8.89	8.89	8.89	13.06	13.06	13.06	13.06	9.28

Notes:

[1] Memory use is the same for both cached and non-cached versions of the chip. Both versions are presented to be consistent with earlier sections, but only one result from each chip is used for normalized calculations.

[2] Projected; as of this writing, these processors are not yet sampling.

Table 8.5-4. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]					[1,2]	[1,2]	
1.20	0.80	0.85	0.80	0.80	0.80	0.80	0.80	0.80	1.59
1.18	0.79	0.79	0.79	0.79	0.99	0.79	0.79	0.79	1.58
1.20	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	1.60
1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1.20	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	1.60
1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
0.84	0.84	0.84	0.89	0.42	0.56	0.88	1.12	1.12	1.72
0.00	0.00	0.00	0.00	0.00	14.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	14.00	0.00	0.00	0.00	0.00	0.00
10.62	9.02	9.07	9.08	22.61	22.95	9.06	9.31	9.31	13.09

Table 8.5-5A. Non-Constant Data Memory Use Formulas

(MD[p,b], bytes) (cont.)

DSP Building Block Functions	**ADI 218x**	**ADI 219x**	**ADI 219x-C**	**ADI 2106x**	**ADI 2106x-C**	**ADI 2116x**	**ADI 2116x-C**	**Lucent 164xx**
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	
Real Block FIR	4*N+2*T+2	4*N+2*T+2	4*N+2*T+2	8*N+4*T+4	8*N+4*T+4	8*N+4*T+16	8*N+4*T+16	2*T+4*N
Single-Sample FIR	2*T+2	2*T+2	2*T+2	4*T+4	4*T+4	8+4*T	8+4*T	2*T
Complex Block FIR	8*N+4*T+2	8*N+4*T+2	8*N+4*T+2	8+16*N+8*T	8+16*N+8*T	16*N+8*T+16	16*N+8*T+16	2*(N*4+1)
LMS Adaptive FIR	4*T+4	4*T+4	4*T+4	8*T+8	8*T+8	8*T+8	8*T+8	(4*T)+8
Two-Biquad IIR	10	10	10	16	16	16	16	8
Vector Dot Product	4*N	4*N	4*N	8*N	8*N	8*N	8*N	4*N
Vector Add	6*N	6*N	6*N	12*N	12*N	12*N	12*N	6*N
Vector Maximum	2*N	2*N	2*N	4*N	4*N	4*(N+2)	4*(N+2)	2*(N+2)
Control	4	4	4	8	8	8	8	12
256-Point FFT	1792	1792	1792	4096	4096	4096	4096	1032
Viterbi	1238	1238	1238	2460	2460	2460	2460	1232
Bit Unpack	188	188	188	348	348	348	348	188

T = # of Taps N = # of Points

Table 8.5-5B. Non-Constant Data Memory Use

(MD[p,b], bytes) (cont.)

DSP Building Block Functions	**ADI 218x**	**ADI 219x**	**ADI 219x-C**	**ADI 2106x**	**ADI 2106x-C**	**ADI 2116x**	**ADI 2116x-C**	**Lucent 164xx**
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	
Real Block FIR	194	194	194	388	388	400	400	192
Single-Sample FIR	34	34	34	68	68	72	72	32
Complex Block FIR	386	386	386	776	776	784	784	322
LMS Adaptive FIR	68	68	68	136	136	136	136	72
Two-Biquad IIR	10	10	10	16	16	16	16	8
Vector Dot Product	160	160	160	320	320	320	320	160
Vector Add	240	240	240	480	480	480	480	240
Vector Maximum	80	80	80	160	160	168	168	84
Control	4	4	4	8	8	8	8	12
256-Point FFT	1792	1792	1792	4096	4096	4096	4096	1032
Viterbi	1238	1238	1238	2460	2460	2460	2460	1232
Bit Unpack	188	188	188	348	348	348	348	188

Substituted Values: T = 16 N = 40

Notes:

[1] Memory use is the same for both cached and non-cached versions of the chip. Both versions are presented to be consistent with earlier sections, but only one result from each chip is used for normalized calculations.

[2] Projected; as of this writing, these processors are not yet sampling.

Table 8.5-5A. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]					[1,2]	[1,2]	
6*N+3*T+3	2*(2*N+T+2)	2*(2*N+T+1)	(4*N)+(2*T)+12	(4*N)+(2*(T-1))	4*N+4*T-4	4+4*N+power(2,	4*N+4*power(2,floor(log(T	4*N+4*power(2,floor(log(T	8*N+8*T
3*T	2*T+2	2*T+2	2*T	(2*T)+2	2*T+8	2*T	power(2, trunc(log(T,2	power(2, trunc(log(T,2	power(2,trunc(log(T,2))+2
6*T+12*N+3	4*(2*N+T+2)	4*(2*N+T)+2	(8*N)+(4*T)+12	(8*N)+(4*T)+4	8*N+12*T-8	(8*N)+(8*T)+4	8*N+4*T+16	8*N+4*T+16	16*N+8*T
6*T+3	2*(2*T+3)	2*(2*T+3)	4*T	4*T	4*T+2	(6*T)+4	8+2*T+power(2,trunc(log(	8+2*T+power(2,trunc(log(	8*T+8
12	8	8	12	8	8	8	8	8	16
6*N	4*N	4*N	4*N	4*N	4*N	4*N	4*N	4*N	8*N
9*N	6*N	6*N	6*N	6*N	6*N+2	6*N	6*N	6*N	12*N
3*N	2*N	2*N	(2*N)+12	(2*N)+4	2*N+4	2*N	2*N+16	2*N+16	4*N
0	12	12	24	4	8	8	8	8	8
3072	1054	1042	2056	2056	1032	2048	2048	2048	4096
2130	1424	1420	1428	1476	466	844	780	780	1310
270	190	188	158	192	188	240	188	188	240

Table 8.5-5B. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]					[1,2]	[1,2]	
291	196	194	204	190	220	228	224	224	448
48	34	34	32	34	40	32	32	32	64
579	392	386	396	388	504	452	400	400	768
99	70	70	64	64	66	100	72	72	136
12	8	8	12	8	8	8	8	8	16
240	160	160	160	160	160	160	160	160	320
360	240	240	240	240	242	240	240	240	480
120	80	80	92	84	84	80	96	96	160
0	12	12	24	4	8	8	8	8	8
3072	1054	1042	2056	2056	1032	2048	2048	2048	4096
2130	1424	1420	1428	1476	466	844	780	780	1310
270	190	188	158	192	188	240	188	188	240

Table 8.5-6. Normalized Non-Constant Data Memory Use

(MD[p,b]/AMD[b], bytes) (cont.)

DSP Building Block Functions	ADI 218x	ADI 219x	ADI 219x-C	ADI 2106x	ADI 2106x-C	ADI 2116x	ADI 2116x-C	Lucent 164xx
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	
Real Block FIR	0.76	0.76	0.76	1.52	1.52	1.57	1.57	0.75
Single-Sample FIR	0.81	0.81	0.81	1.61	1.61	1.71	1.71	0.76
Complex Block FIR	0.78	0.78	0.78	1.57	1.57	1.59	1.59	0.65
LMS Adaptive FIR	0.78	0.78	0.78	1.56	1.56	1.56	1.56	0.83
Two-Biquad IIR	0.95	0.95	0.95	1.51	1.51	1.51	1.51	0.76
Vector Dot Product	0.80	0.80	0.80	1.60	1.60	1.60	1.60	0.80
Vector Add	0.80	0.80	0.80	1.60	1.60	1.60	1.60	0.80
Vector Maximum	0.77	0.77	0.77	1.55	1.55	1.62	1.62	0.81
Control	0.47	0.47	0.47	0.93	0.93	0.93	0.93	1.40
256-Point FFT	0.80	0.80	0.80	1.83	1.83	1.83	1.83	0.46
Viterbi	0.87	0.87	0.87	1.73	1.73	1.73	1.73	0.87
Bit Unpack	0.85	0.85	0.85	1.56	1.56	1.56	1.56	0.85
Total	9.43	9.43	9.43	18.59	18.59	18.82	18.82	9.73

Notes:

[1] Memory use is the same for both cached and non-cached versions of the chip. Both versions are presented to be consistent with earlier sections, but only one result from each chip is used for normalized calculations.

[2] Projected; as of this writing, these processors are not yet sampling.

Table 8.5-6. (cont.)

Motorola	Motorola	Motorola	StarCore	TI	TI	TI	TI	TI	TI
563xx	568xx	5685x	SC140	C54xx	C55xx	C62xx	C64xx	C64xx-C	C67xx
		[2]					[1,2]	[1,2]	
1.14	0.77	0.76	0.80	0.75	0.86	0.90	0.88	0.88	1.76
1.14	0.81	0.81	0.76	0.81	0.95	0.76	0.76	0.76	1.52
1.17	0.79	0.78	0.80	0.79	1.02	0.91	0.81	0.81	1.55
1.14	0.80	0.80	0.73	0.73	0.76	1.15	0.83	0.83	1.56
1.14	0.76	0.76	1.14	0.76	0.76	0.76	0.76	0.76	1.51
1.20	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	1.60
1.20	0.80	0.80	0.80	0.80	0.81	0.80	0.80	0.80	1.60
1.16	0.77	0.77	0.89	0.81	0.81	0.77	0.93	0.93	1.55
0.00	1.40	1.40	2.80	0.47	0.93	0.93	0.93	0.93	0.93
1.37	0.47	0.47	0.92	0.92	0.46	0.92	0.92	0.92	1.83
1.50	1.00	1.00	1.00	1.04	0.33	0.59	0.55	0.55	0.92
1.21	0.85	0.85	0.71	0.86	0.85	1.08	0.85	0.85	1.08
13.37	10.03	9.99	12.15	9.53	9.33	10.37	9.80	9.80	17.42

Table 8.5-7. ROMable Memory Use

(MPC[p,b] = MP[p,b] + MC[p,b], bytes) (cont.)

DSP Building Block Functions	ADI 218x	ADI 219x	ADI 219x-C	ADI 2106x	ADI 2106x-C	ADI 2116x	ADI 2116x-C	Lucent 164xx
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	
Real Block FIR	98	140	140	184	184	232	232	122
Single-Sample FIR	80	92	92	142	142	178	178	78
Complex Block FIR	178	178	178	314	314	344	344	162
LMS Adaptive FIR	66	84	84	186	186	186	186	102
Two-Biquad IIR	64	64	64	104	104	104	104	56
Vector Dot Product	18	18	18	54	54	78	78	20
Vector Add	24	24	24	54	54	66	66	32
Vector Maximum	51	42	42	66	66	84	84	56
Control	213	222	222	342	342	342	342	182
256-Point FFT	895	895	895	2556	2556	2610	2610	1302
Viterbi	288	306	306	474	474	486	486	188
Bit Unpack	57	57	57	132	132	132	132	102
Total	2032	2122	2122	4608	4608	4842	4842	2402

Table 8.5-8. Normalized ROMable Memory Use

(MC[p,b]/AMC[b], bytes) (cont.)

DSP Building Block Functions	ADI 218x	ADI 219x	ADI 219x-C	ADI 2106x	ADI 2106x-C	ADI 2116x	ADI 2116x-C	Lucent 164xx
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	
Real Block FIR	0.40	0.57	0.57	0.75	0.75	0.95	0.95	0.50
Single-Sample FIR	0.62	0.71	0.71	1.10	1.10	1.38	1.38	0.60
Complex Block FIR	0.60	0.60	0.60	1.06	1.06	1.16	1.16	0.55
LMS Adaptive FIR	0.34	0.44	0.44	0.97	0.97	0.97	0.97	0.53
Two-Biquad IIR	0.71	0.71	0.71	1.15	1.15	1.15	1.15	0.62
Vector Dot Product	0.29	0.29	0.29	0.86	0.86	1.24	1.24	0.32
Vector Add	0.44	0.44	0.44	1.00	1.00	1.22	1.22	0.59
Vector Maximum	0.44	0.37	0.37	0.57	0.57	0.73	0.73	0.49
Control	0.97	1.01	1.01	1.55	1.55	1.55	1.55	0.83
256-Point FFT	0.57	0.57	0.57	1.63	1.63	1.66	1.66	0.83
Viterbi	0.58	0.62	0.62	0.95	0.95	0.98	0.98	0.38
Bit Unpack	0.55	0.55	0.55	1.27	1.27	1.27	1.27	0.98
Total	6.51	6.87	6.87	12.86	12.86	14.25	14.25	7.21

Notes:

[1] Memory use is the same for both cached and non-cached versions of the chip. Both versions are presented to be consistent with earlier sections, but only one result from each chip is used for normalized calculations.

[2] Projected; as of this writing, these processors are not yet sampling.

Table 8.5-7. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]					[1,2]	[1,2]	
132	96	94	282	86	131	600	656	656	560
93	62	60	146	62	78	208	212	212	320
210	160	146	310	140	167	600	632	632	608
105	46	60	174	66	49	640	248	248	680
66	56	58	102	50	90	156	116	116	180
21	14	14	60	16	22	168	120	120	260
42	24	24	56	36	35	112	100	100	128
30	32	56	176	72	106	288	316	316	232
210	128	122	122	222	152	288	248	248	288
1086	1342	1228	1416	620	913	2044	2024	2024	3056
651	926	734	476	248	251	576	460	460	896
72	56	44	112	84	60	184	184	184	184
2718	2942	2640	3432	1702	2054	5864	5316	5316	7392

Table 8.5-8. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]					[1,2]	[1,2]	
0.54	0.39	0.39	1.16	0.35	0.54	2.46	2.69	2.69	2.30
0.72	0.48	0.46	1.13	0.48	0.60	1.61	1.64	1.64	2.47
0.71	0.54	0.49	1.05	0.47	0.56	2.02	2.13	2.13	2.05
0.55	0.24	0.31	0.90	0.34	0.25	3.33	1.29	1.29	3.54
0.73	0.62	0.64	1.13	0.55	1.00	1.73	1.28	1.28	1.99
0.33	0.22	0.22	0.95	0.25	0.35	2.66	1.90	1.90	4.12
0.78	0.44	0.44	1.04	0.67	0.65	2.07	1.85	1.85	2.37
0.26	0.28	0.49	1.53	0.63	0.92	2.51	2.75	2.75	2.02
0.95	0.58	0.55	0.55	1.01	0.69	1.31	1.13	1.13	1.31
0.69	0.85	0.78	0.90	0.39	0.58	1.30	1.29	1.29	1.95
1.31	1.86	1.48	0.96	0.50	0.50	1.16	0.93	0.93	1.80
0.69	0.54	0.42	1.07	0.81	0.58	1.76	1.76	1.76	1.76
8.26	7.05	6.68	12.37	6.46	7.23	23.92	20.65	20.65	27.68

Table 8.5-9. Total Memory Use

(MPCD[p,b] = MP[p,b] + MC[p,b] + MD[p,b], bytes) (cont.)

DSP Building Block Functions	ADI 218x	ADI 219x	ADI 219x-C	ADI 2106x	ADI 2106x-C	ADI 2116x	ADI 2116x-C	Lucent 164xx
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	
Real Block FIR	292	334	334	572	572	632	632	314
Single-Sample FIR	114	126	126	210	210	250	250	110
Complex Block FIR	564	564	564	1090	1090	1128	1128	484
LMS Adaptive FIR	134	152	152	322	322	322	322	174
Two-Biquad IIR	74	74	74	120	120	120	120	64
Vector Dot Product	178	178	178	374	374	398	398	180
Vector Add	264	264	264	534	534	546	546	272
Vector Maximum	131	122	122	226	226	252	252	140
Control	217	226	226	350	350	350	350	194
256-Point FFT	2687	2687	2687	6652	6652	6706	6706	2334
Viterbi	1526	1544	1544	2934	2934	2946	2946	1420
Bit Unpack	245	245	245	480	480	480	480	290
Total	6426	6516	6516	13864	13864	14130	14130	5976

Notes:

[1] Memory use is the same for both cached and non-cached versions of the chip. Both versions are presented to be consistent with earlier sections, but only one result from each chip is used for normalized calculations.

[2] Projected; as of this writing, these processors are not yet sampling.

Table 8.5-10. Normalized Total Memory Use

(MPCD[p,b]/AMPCD[b], bytes) (cont.)

DSP Building Block Functions	ADI 218x	ADI 219x	ADI 219x-C	ADI 2106x	ADI 2106x-C	ADI 2116x	ADI 2116x-C	Lucent 164xx
Notes		[1,2]	[1,2]	[1]	[1]	[1]	[1]	
Real Block FIR	0.59	0.67	0.67	1.15	1.15	1.27	1.27	0.63
Single-Sample FIR	0.66	0.73	0.73	1.22	1.22	1.46	1.46	0.64
Complex Block FIR	0.71	0.71	0.71	1.38	1.38	1.43	1.43	0.61
LMS Adaptive FIR	0.48	0.54	0.54	1.15	1.15	1.15	1.15	0.62
Two-Biquad IIR	0.73	0.73	0.73	1.19	1.19	1.19	1.19	0.63
Vector Dot Product	0.68	0.68	0.68	1.42	1.42	1.51	1.51	0.68
Vector Add	0.75	0.75	0.75	1.51	1.51	1.54	1.54	0.77
Vector Maximum	0.60	0.56	0.56	1.04	1.04	1.15	1.15	0.64
Control	0.95	0.99	0.99	1.53	1.53	1.53	1.53	0.85
256-Point FFT	0.71	0.71	0.71	1.75	1.75	1.76	1.76	0.61
Viterbi	0.80	0.80	0.80	1.53	1.53	1.54	1.54	0.74
Bit Unpack	0.75	0.75	0.75	1.47	1.47	1.47	1.47	0.89
Total	8.40	8.62	8.62	16.33	16.33	17.00	17.00	8.32

Notes:

[1] Memory use is the same for both cached and non-cached versions of the chip. Both versions are presented to be consistent with earlier sections, but only one result from each chip is used for normalized calculations.

[2] Projected; as of this writing, these processors are not yet sampling.

Table 8.5-9. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]					[1,2]	[1,2]	
423	292	288	486	276	351	828	880	880	1008
141	96	94	178	96	118	240	244	244	384
789	552	532	706	528	671	1052	1032	1032	1376
204	116	130	238	130	115	740	320	320	816
78	64	66	114	58	98	164	124	124	196
261	174	174	220	176	182	328	280	280	580
402	264	264	296	276	277	352	340	340	608
150	112	136	268	156	190	368	412	412	392
210	140	134	146	226	160	296	256	256	296
4158	2396	2270	3472	2676	1945	4092	4072	4072	7152
2781	2350	2154	1904	1724	717	1420	1240	1240	2206
342	246	232	270	276	248	424	372	372	424
9939	6802	6474	8298	6598	5072	10304	9572	9572	15438

Table 8.5-10. (cont.)

Motorola 563xx	Motorola 568xx	Motorola 5685x	StarCore SC140	TI C54xx	TI C55xx	TI C62xx	TI C64xx	TI C64xx-C	TI C67xx
		[2]					[1,2]	[1,2]	
0.85	0.59	0.58	0.98	0.55	0.70	1.66	1.77	1.77	2.02
0.82	0.56	0.55	1.04	0.56	0.69	1.40	1.42	1.42	2.24
1.00	0.70	0.67	0.89	0.67	0.85	1.33	1.31	1.31	1.74
0.73	0.42	0.47	0.85	0.47	0.41	2.65	1.14	1.14	2.92
0.77	0.63	0.65	1.13	0.57	0.97	1.62	1.23	1.23	1.94
0.99	0.66	0.66	0.84	0.67	0.69	1.25	1.06	1.06	2.20
1.13	0.75	0.75	0.84	0.78	0.78	0.99	0.96	0.96	1.72
0.69	0.51	0.62	1.23	0.71	0.87	1.69	1.89	1.89	1.80
0.92	0.61	0.59	0.64	0.99	0.70	1.29	1.12	1.12	1.29
1.09	0.63	0.60	0.91	0.70	0.51	1.07	1.07	1.07	1.88
1.45	1.22	1.12	0.99	0.90	0.37	0.74	0.65	0.65	1.15
1.05	0.75	0.71	0.83	0.84	0.76	1.30	1.14	1.14	1.30
11.49	8.03	7.96	11.16	8.42	8.31	17.00	14.75	14.75	22.20

Figure 8.5-1A. Real Block FIR Total Memory Use

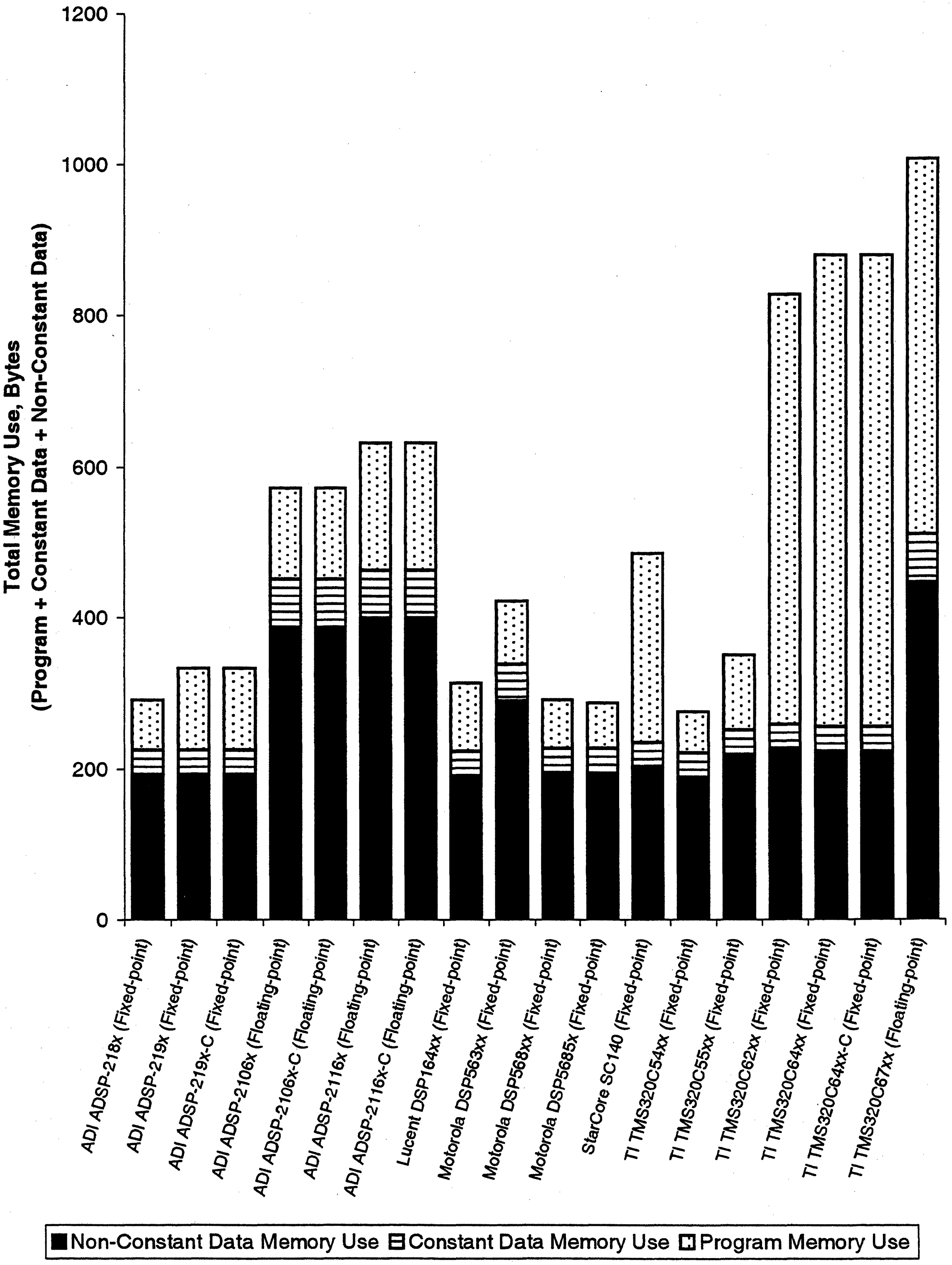


Figure 8.5-1B. Real Block FIR ROMable Memory Use (Program + Constant Data)

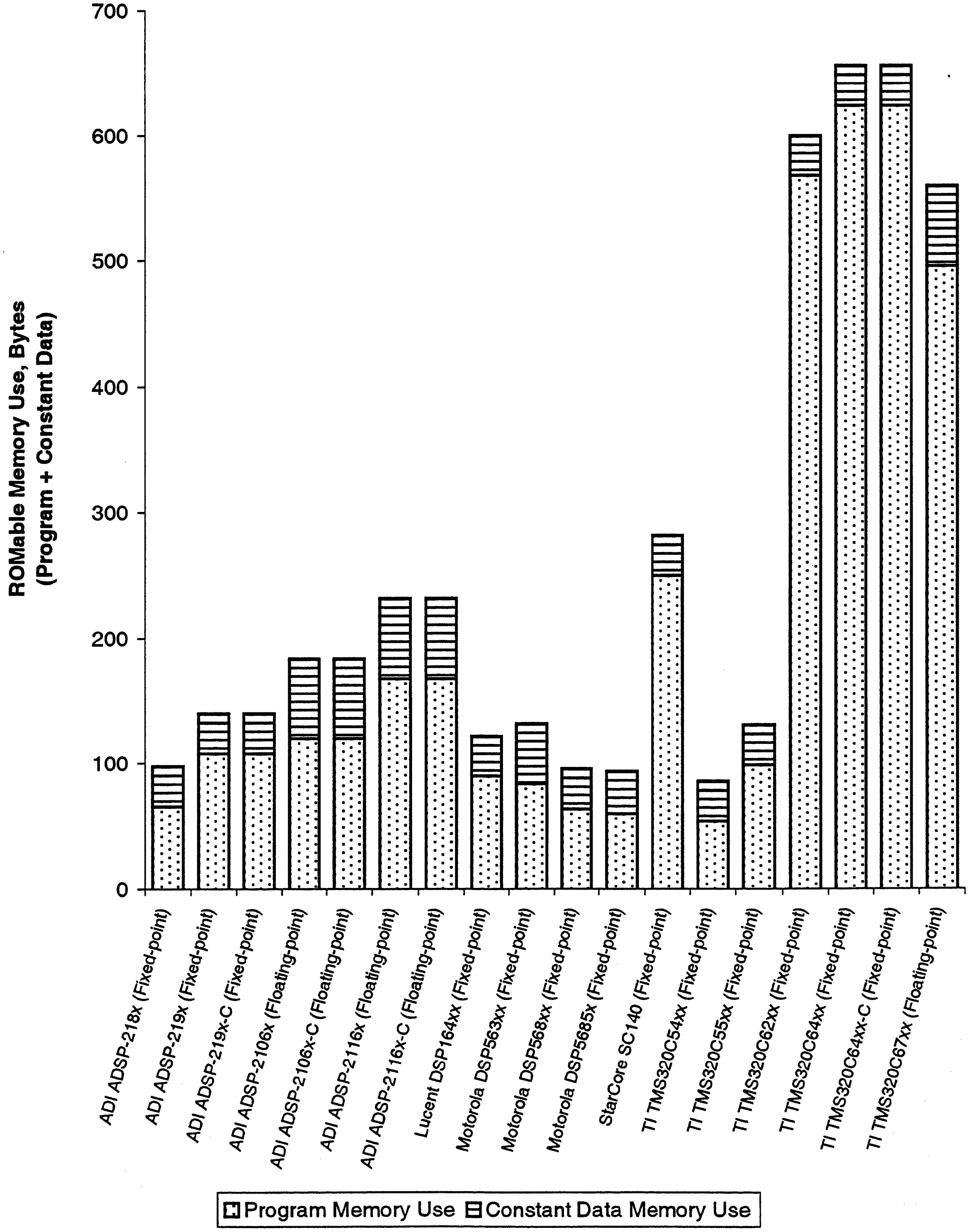


Figure 8.5-2A. Single-Sample FIR Total Memory Use

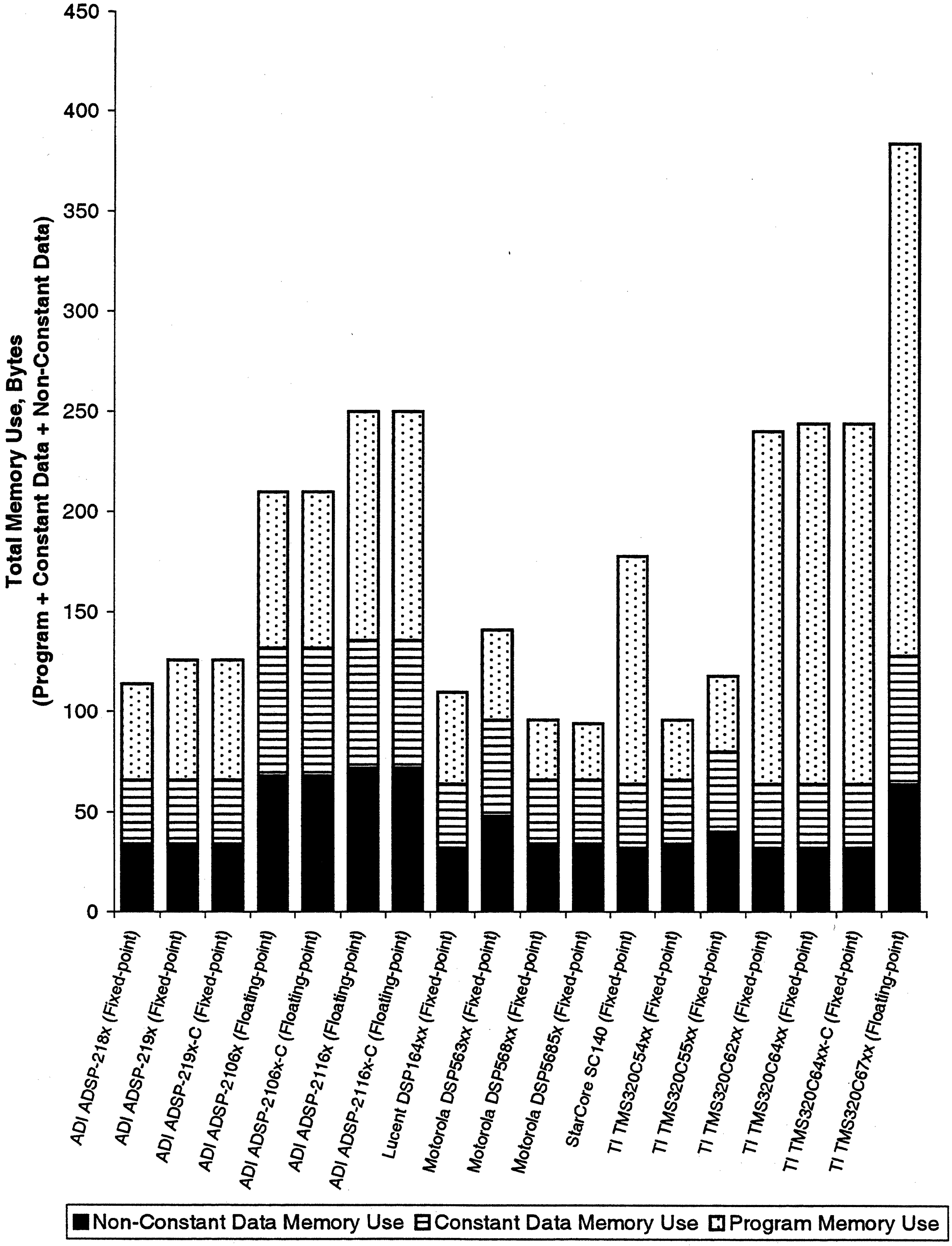


Figure 8.5-2B. Single-Sample FIR ROMable Memory Use (Program + Constant Data)

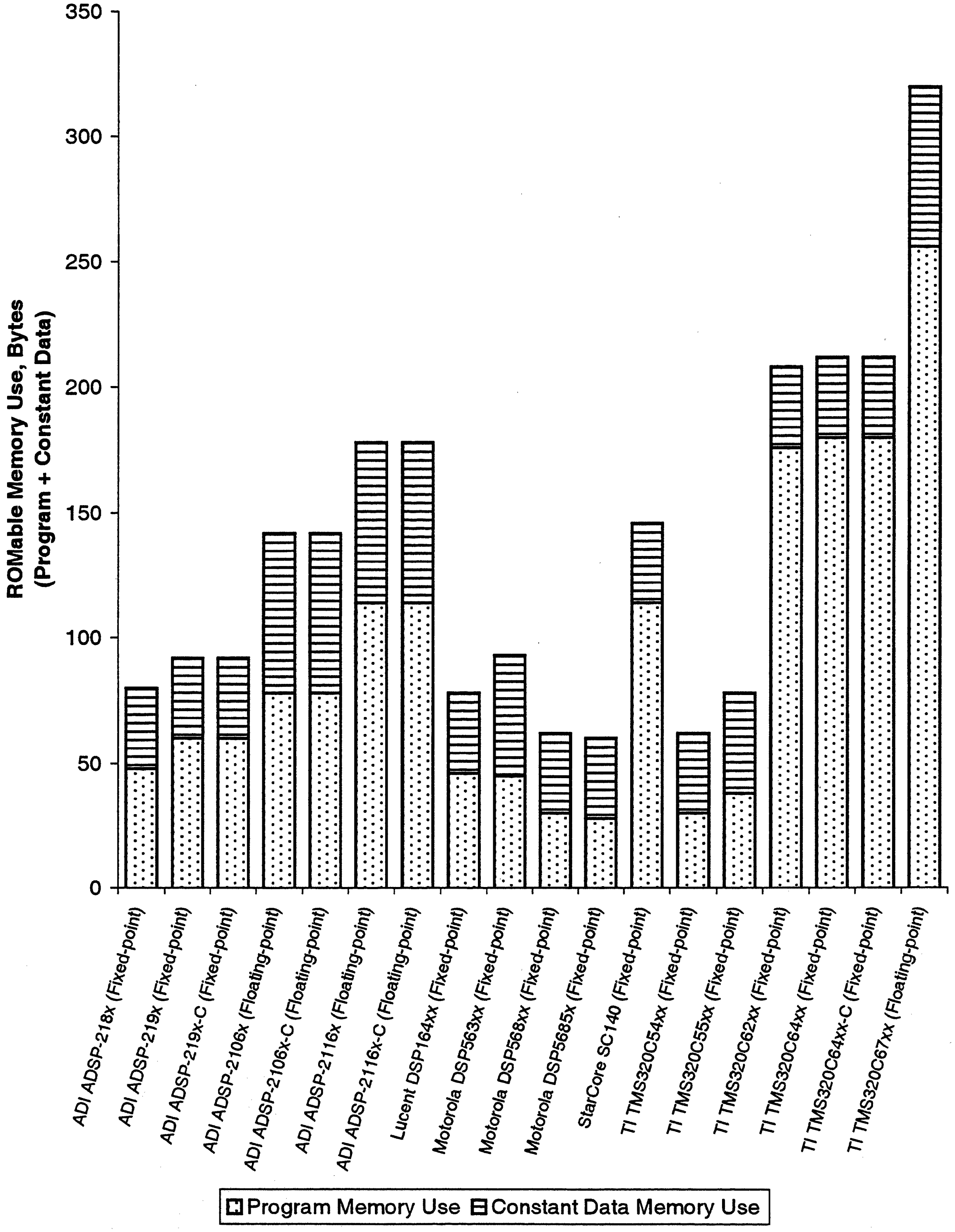


Figure 8.5-3A. Complex Block FIR Total Memory Use

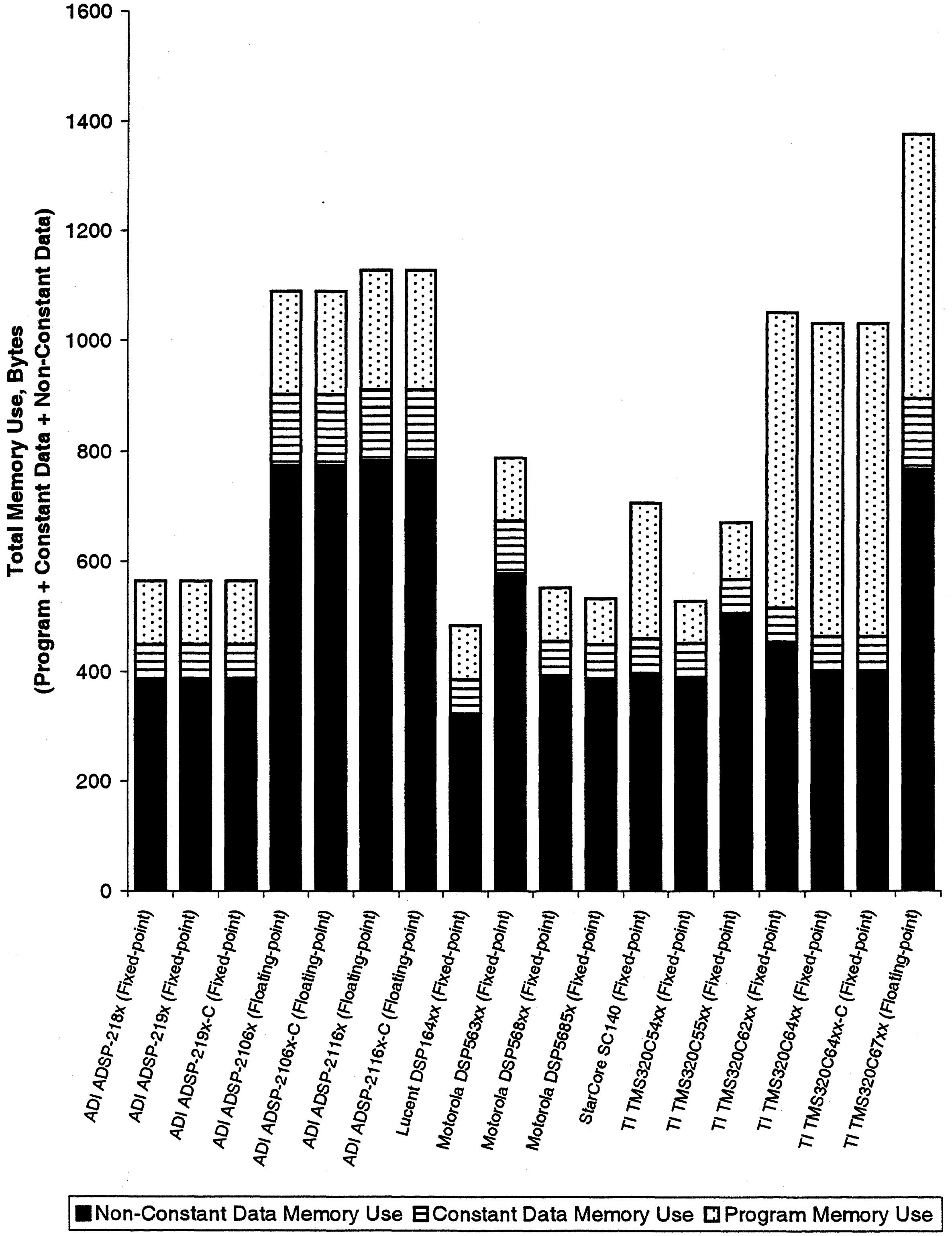


Figure 8.5-3B. Complex Block FIR ROMable Memory Use (Program + Constant Data)

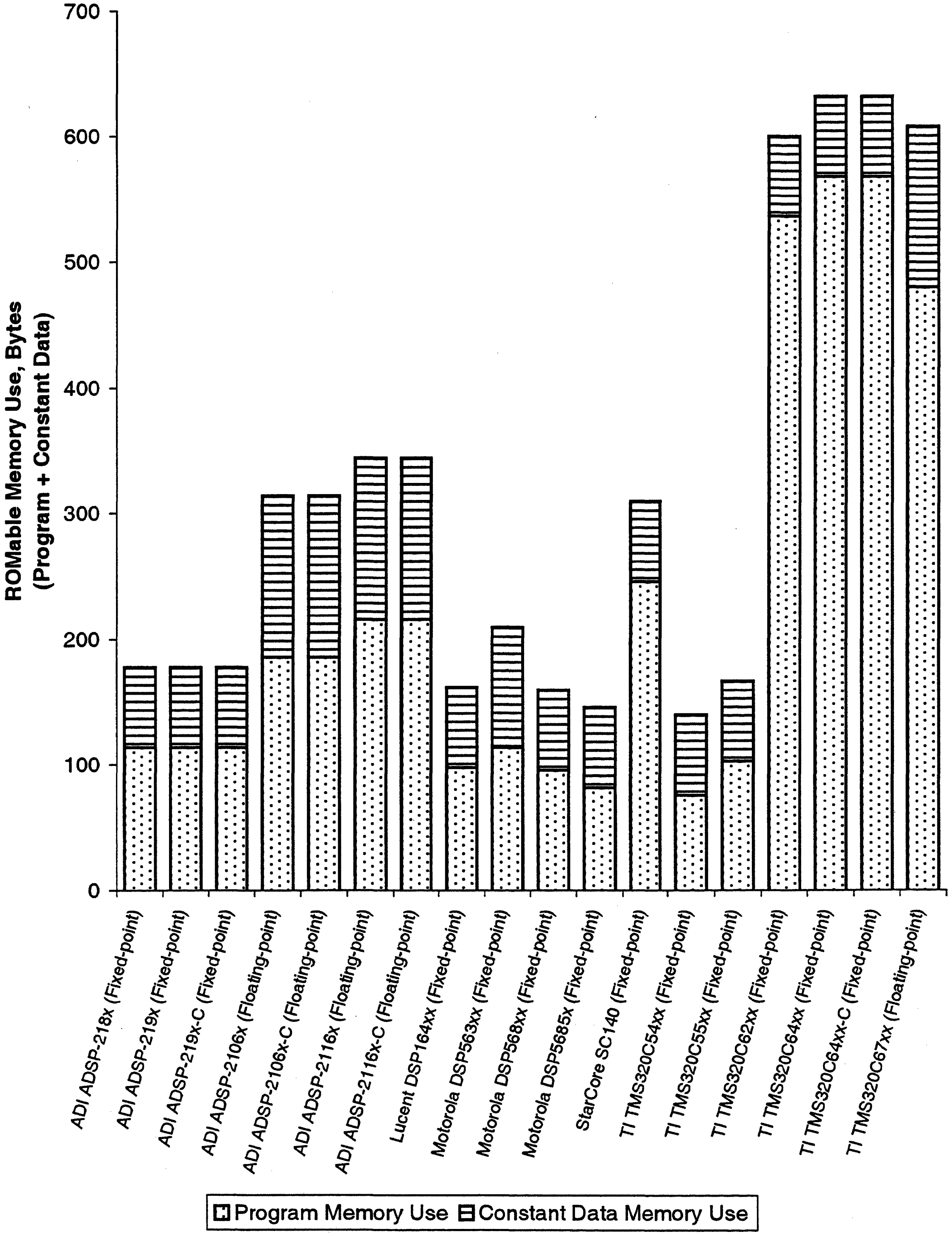


Figure 8.5-4A. LMS Adaptive FIR Total Memory Use

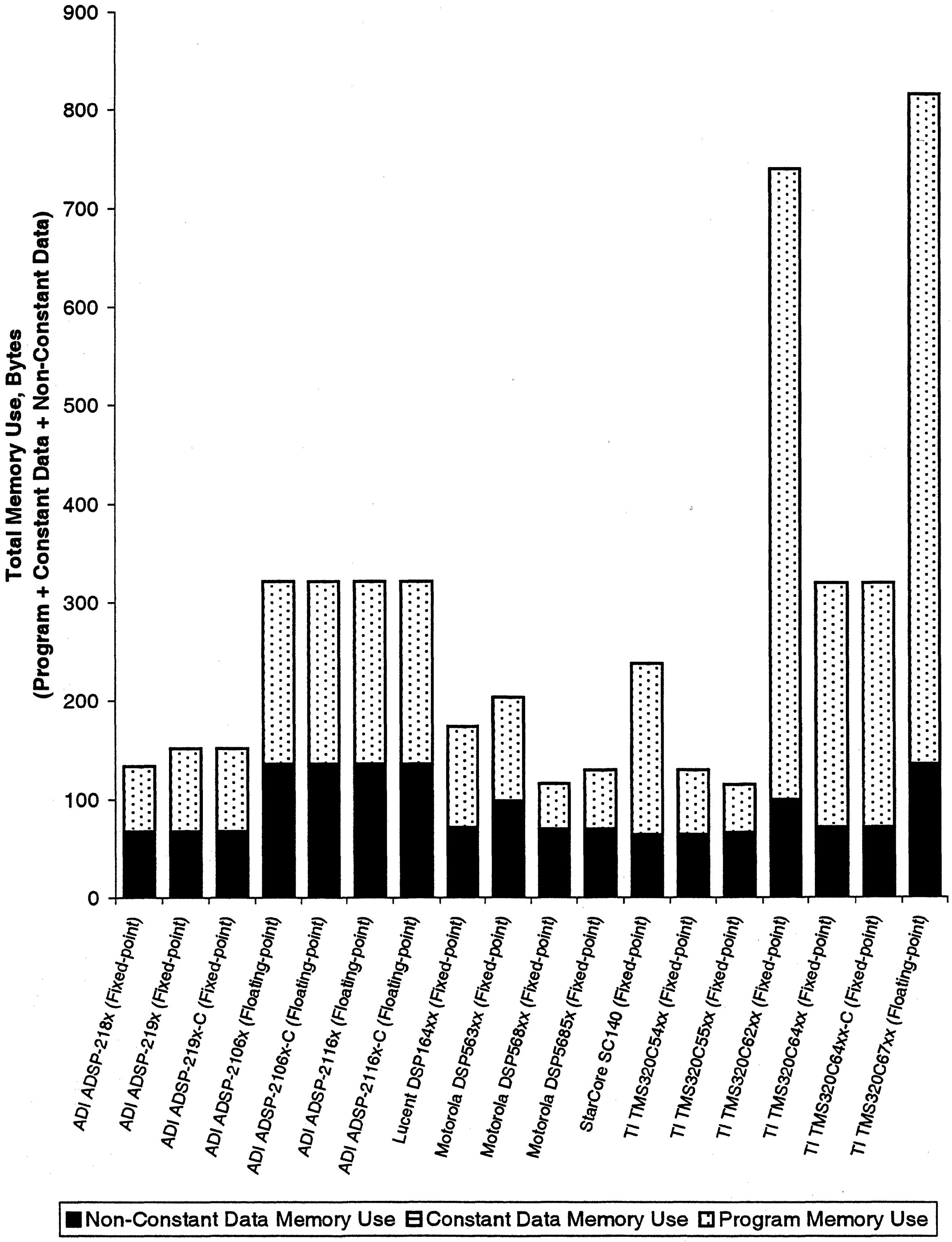


Figure 8.5-4B. LMS Adaptive FIR ROMable Memory Use (Program + Constant Data)

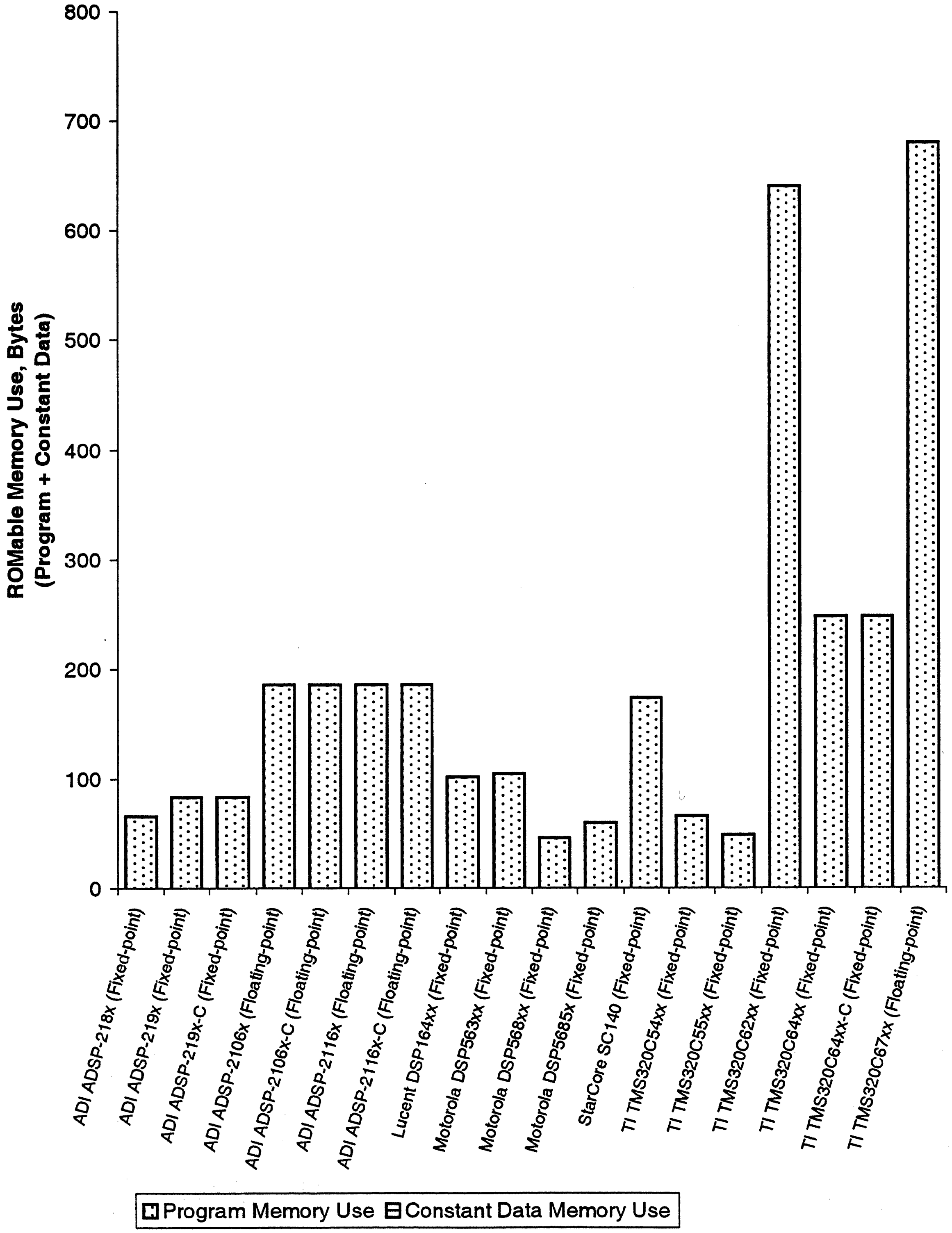


Figure 8.5-5A. Two-Biquad IIR Total Memory Use

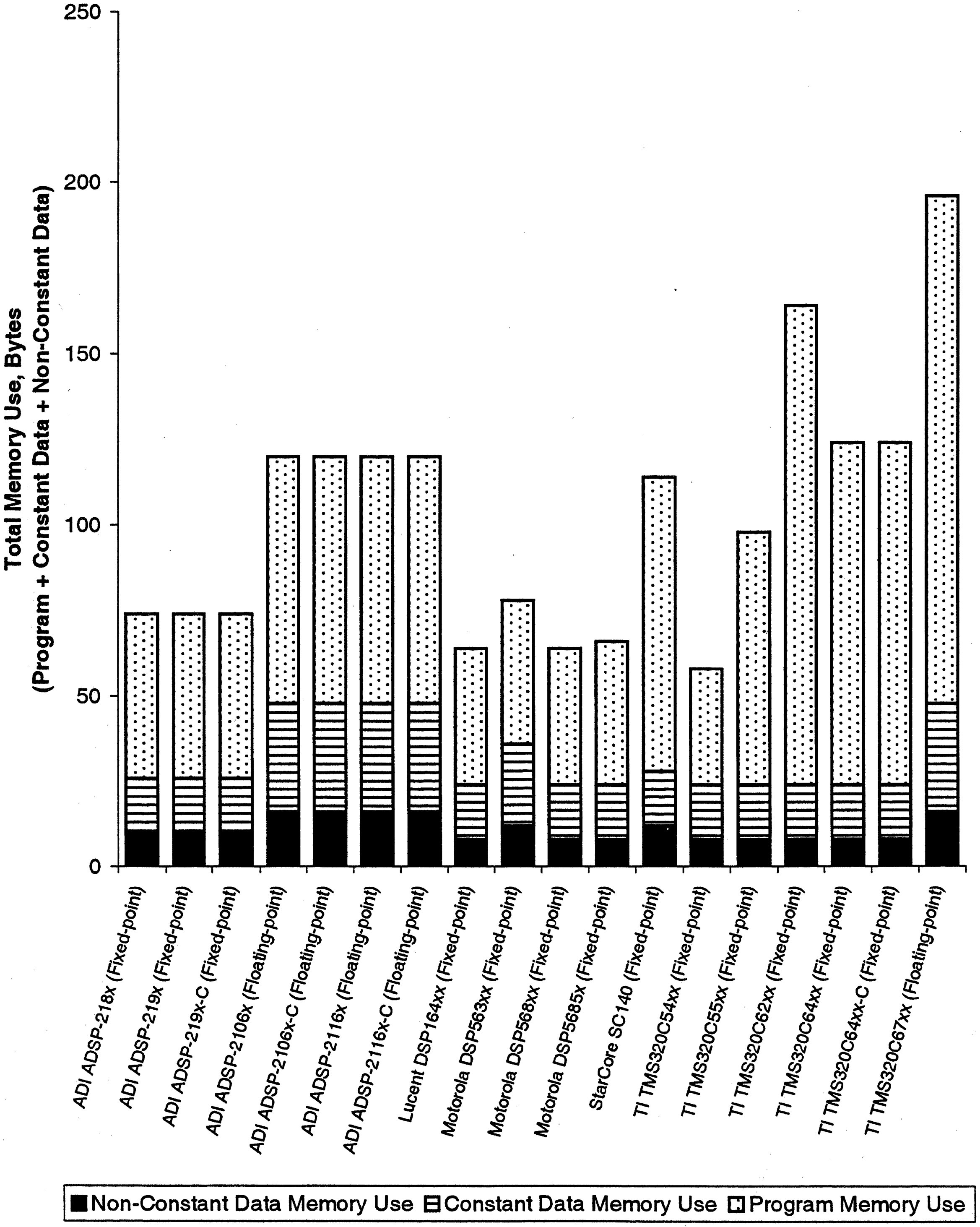


Figure 8.5-5B. Two-Biquad IIR ROMable Memory Use (Program + Constant Data)

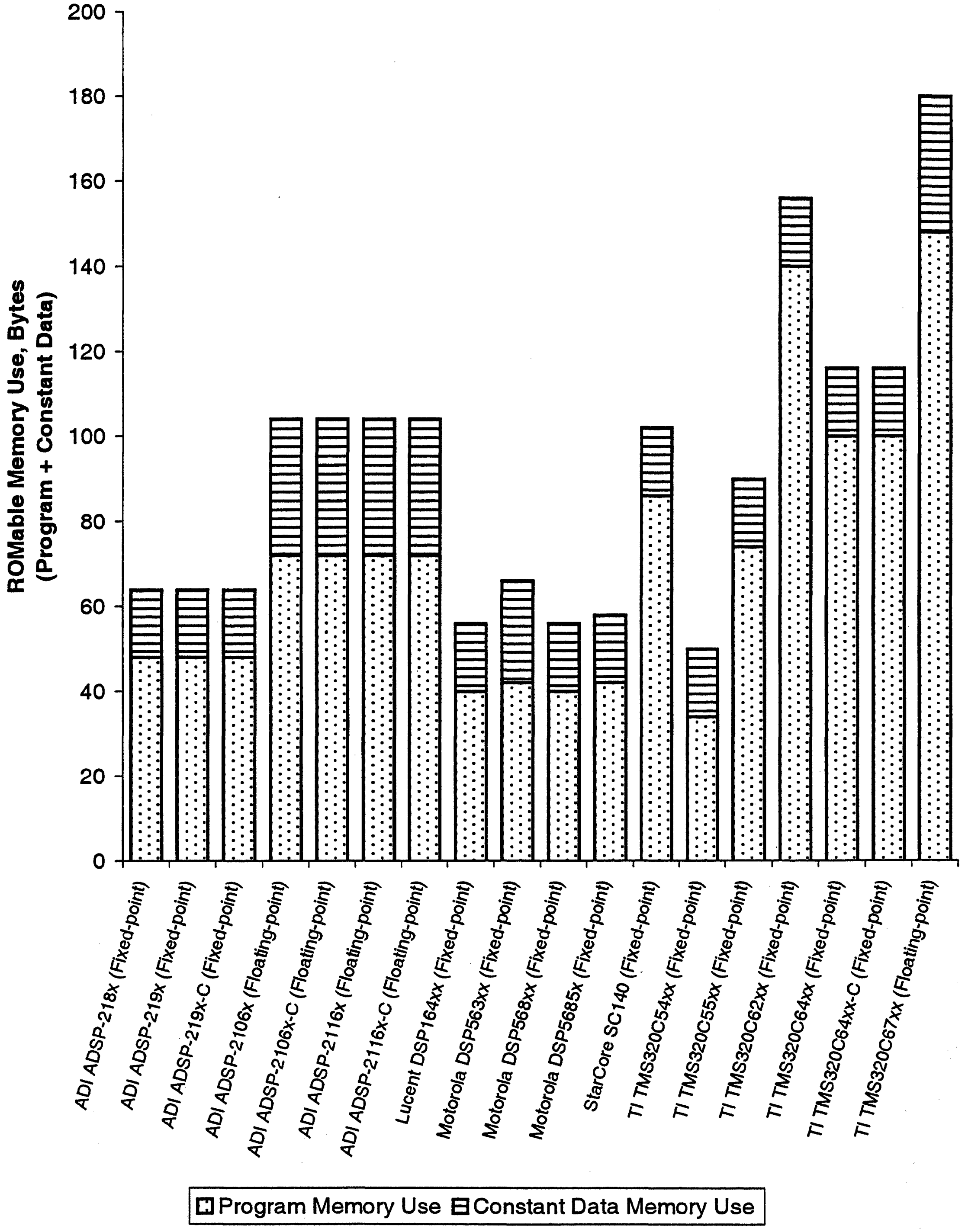


Figure 8.5-6A. Vector Dot Product
Total Memory Use

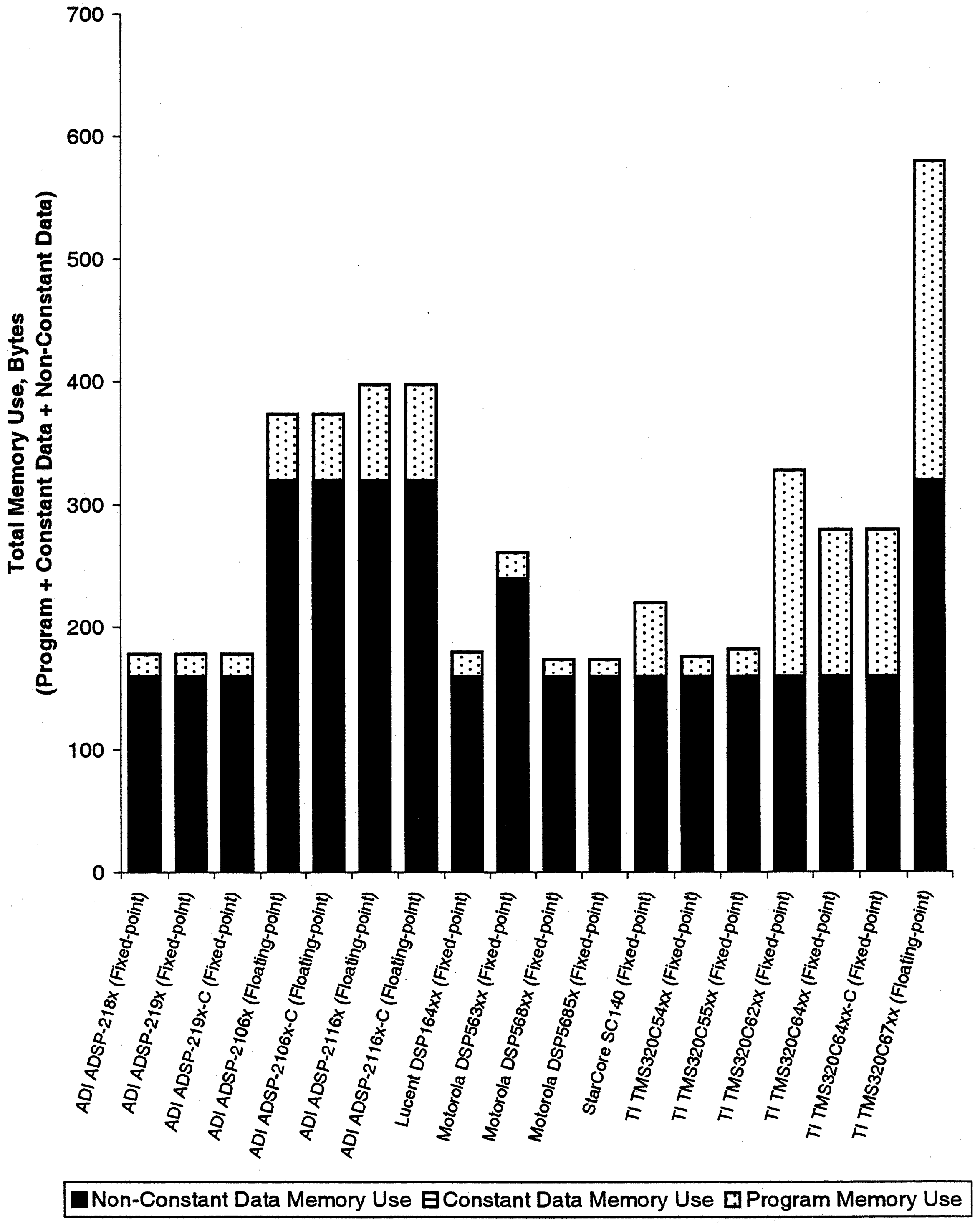


Figure 8.5-6B. Vector Dot Product ROMable Memory Use (Program + Constant Data)

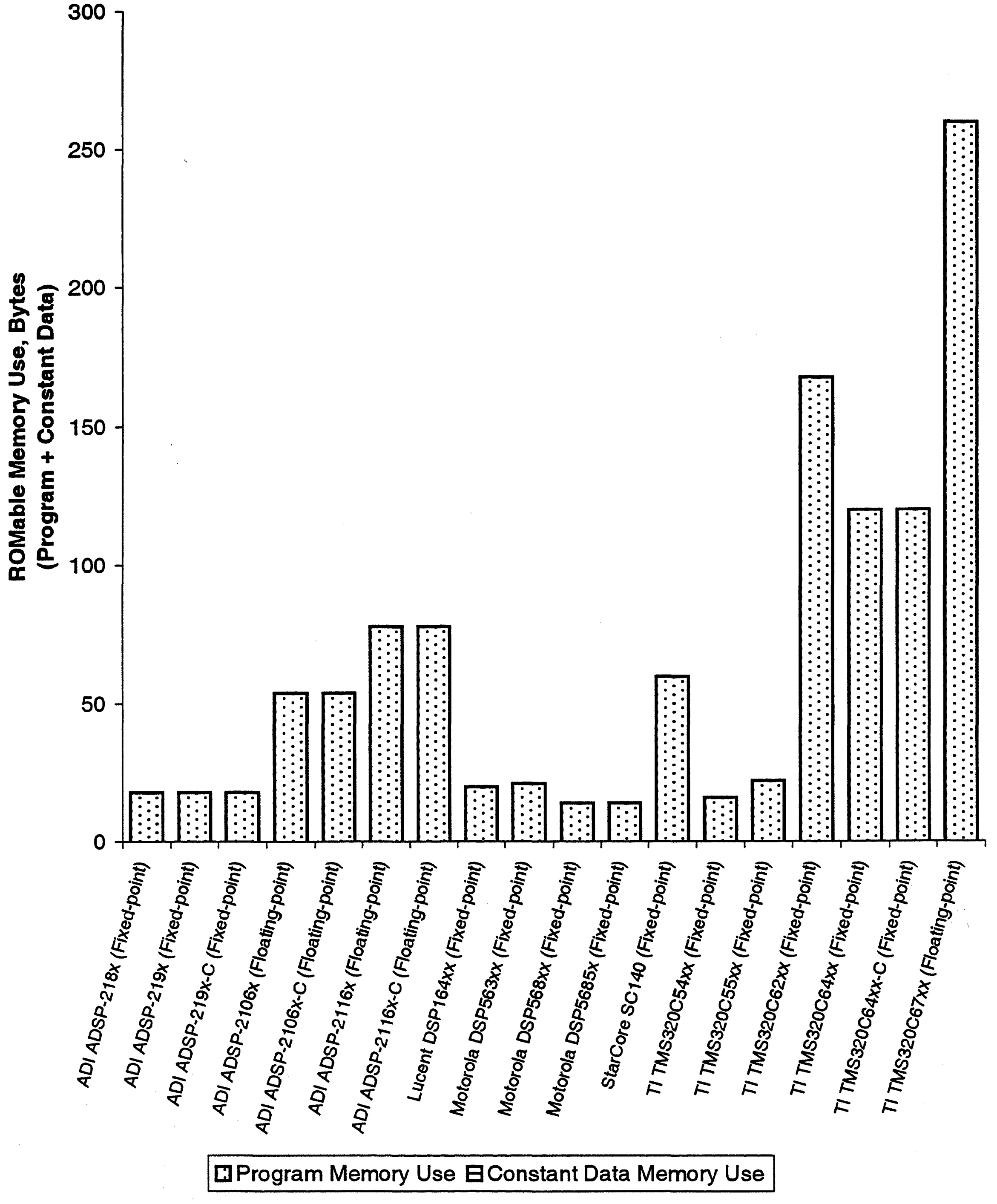


Figure 8.5-7A. Vector Add
Total Memory Use

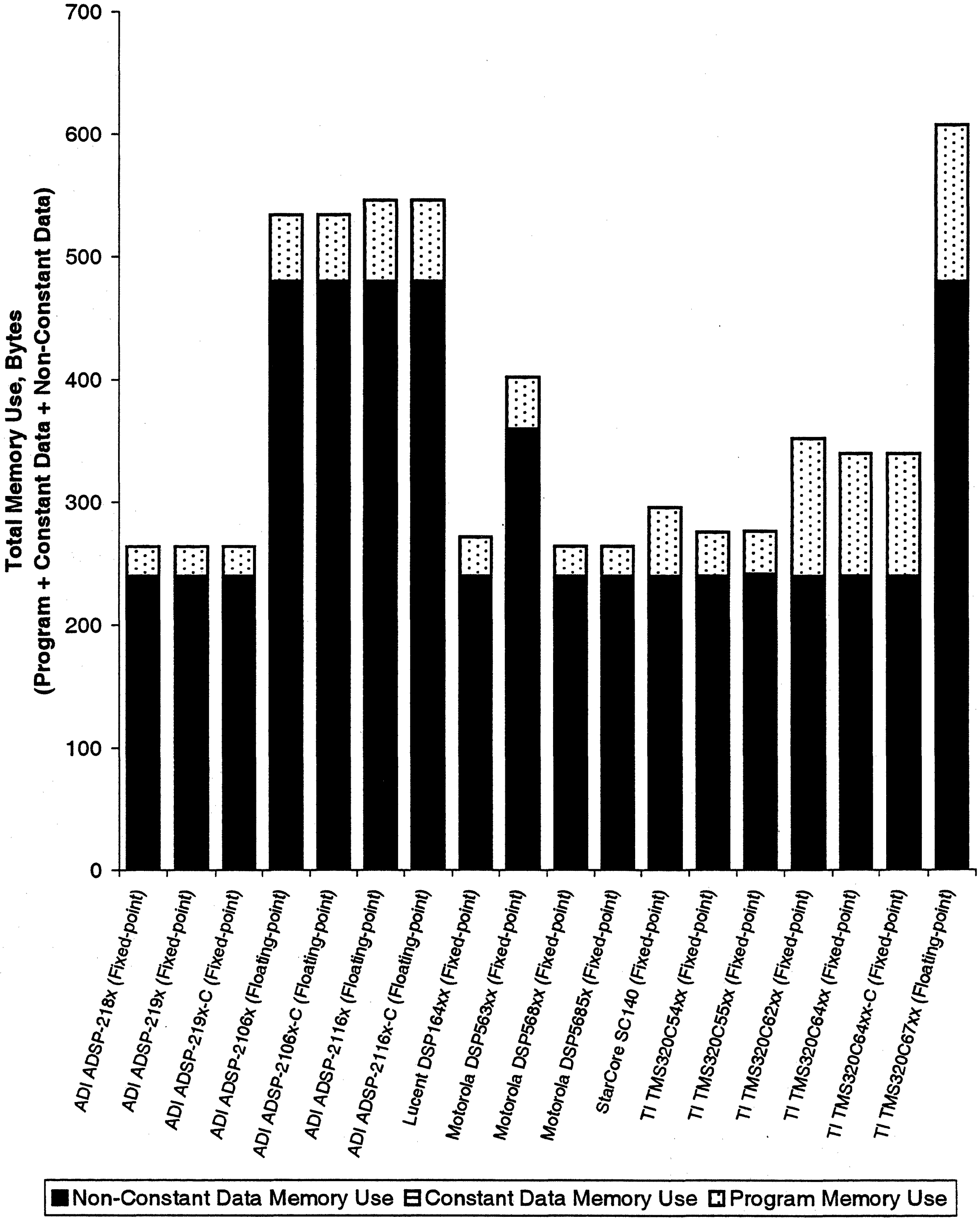


Figure 8.5-7B. Vector Add ROMable Memory Use (Program + Constant Data)

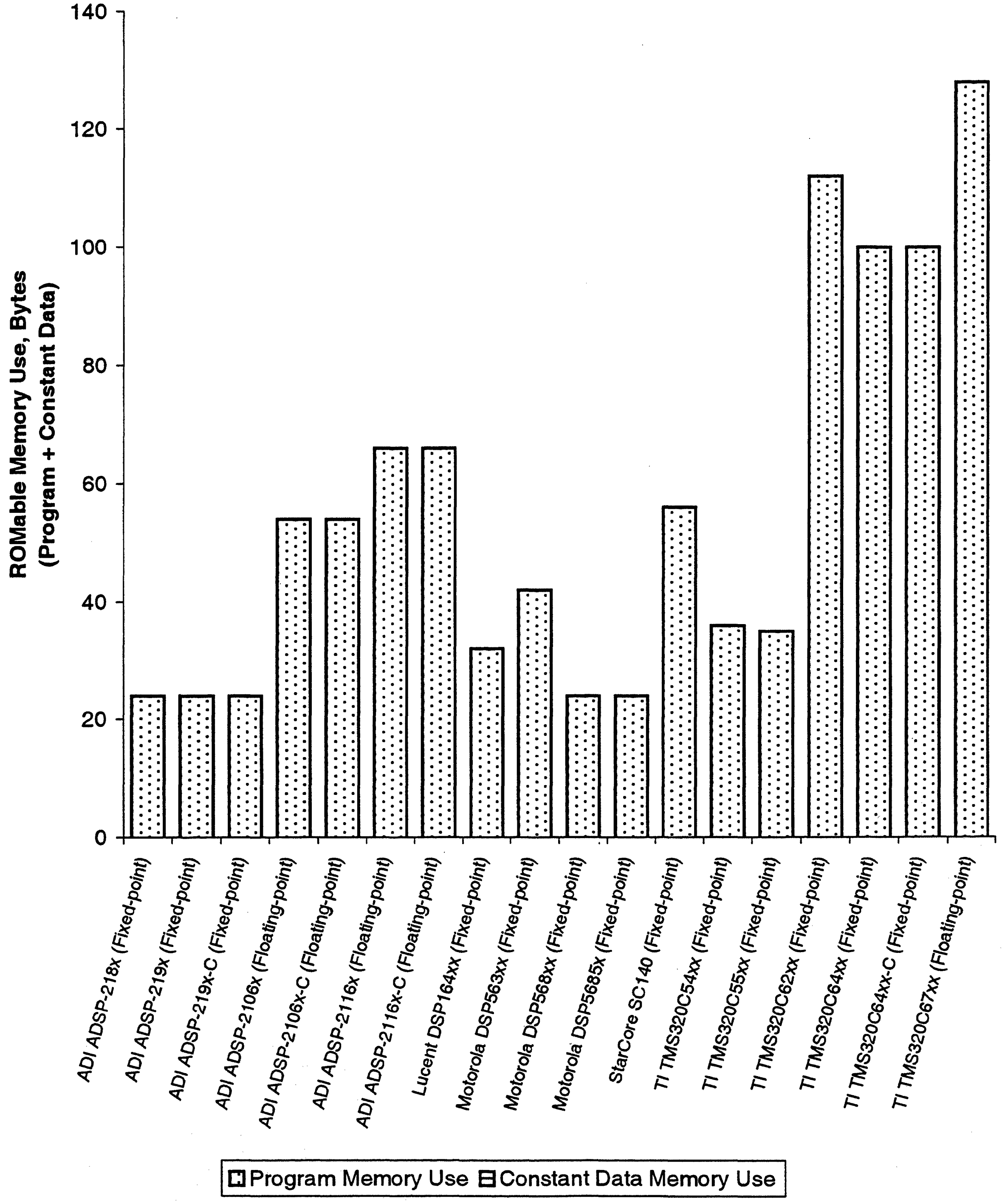


Figure 8.5-8A. Vector Maximum Total Memory Use

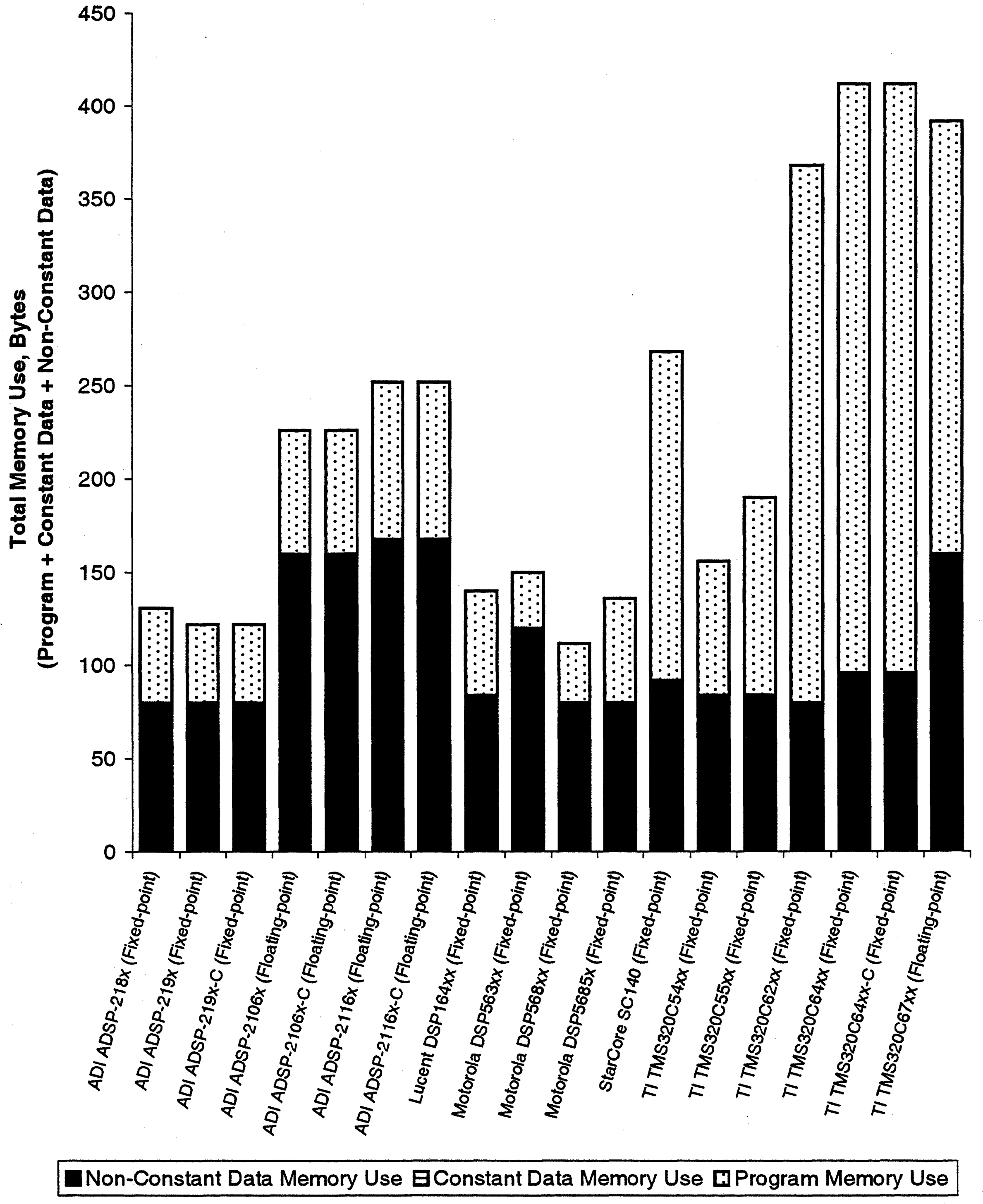


Figure 8.5-8B. Vector Maximum ROMable Memory Use (Program + Constant Data)

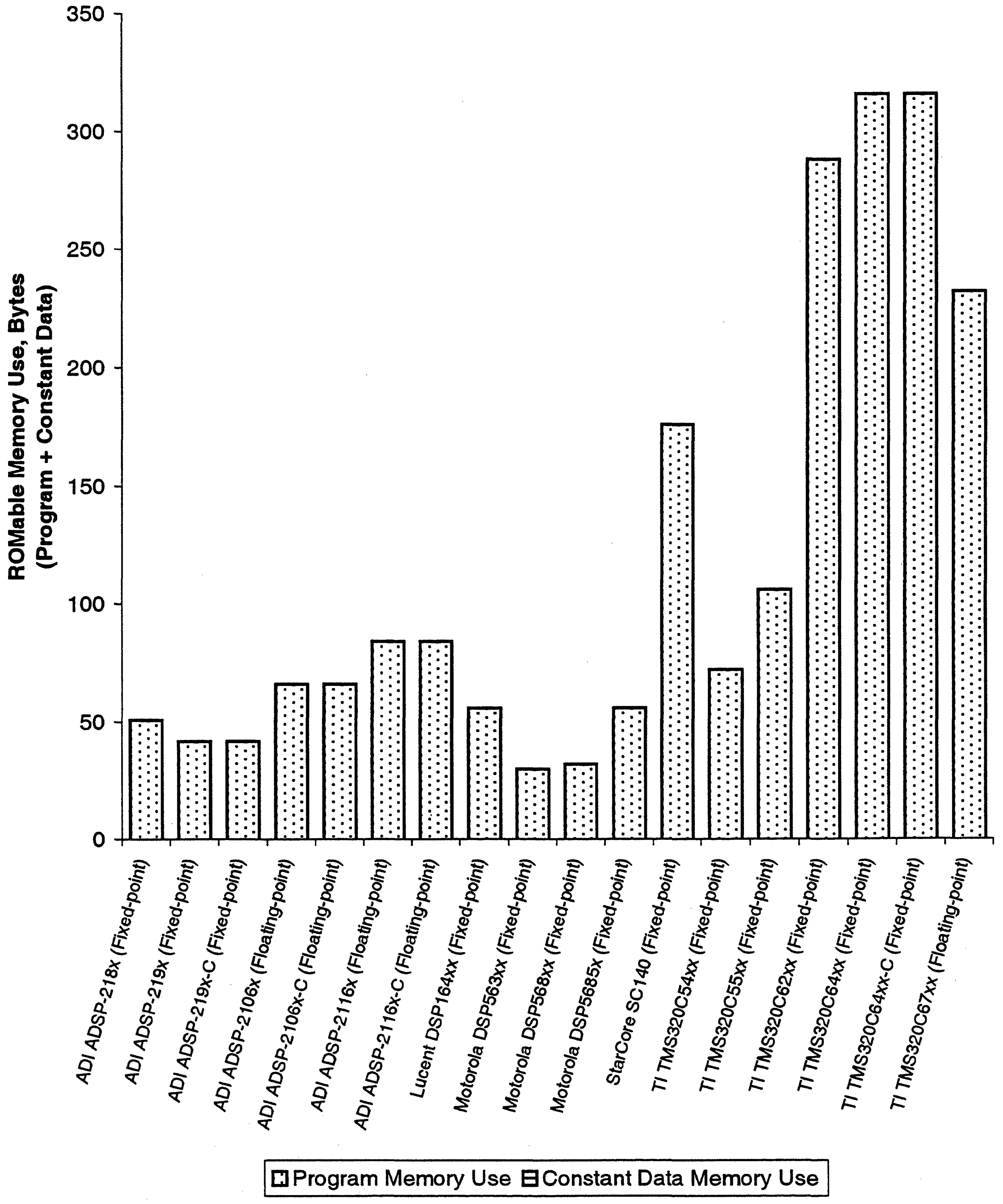


Figure 8.5-9A. Control
Total Memory Use

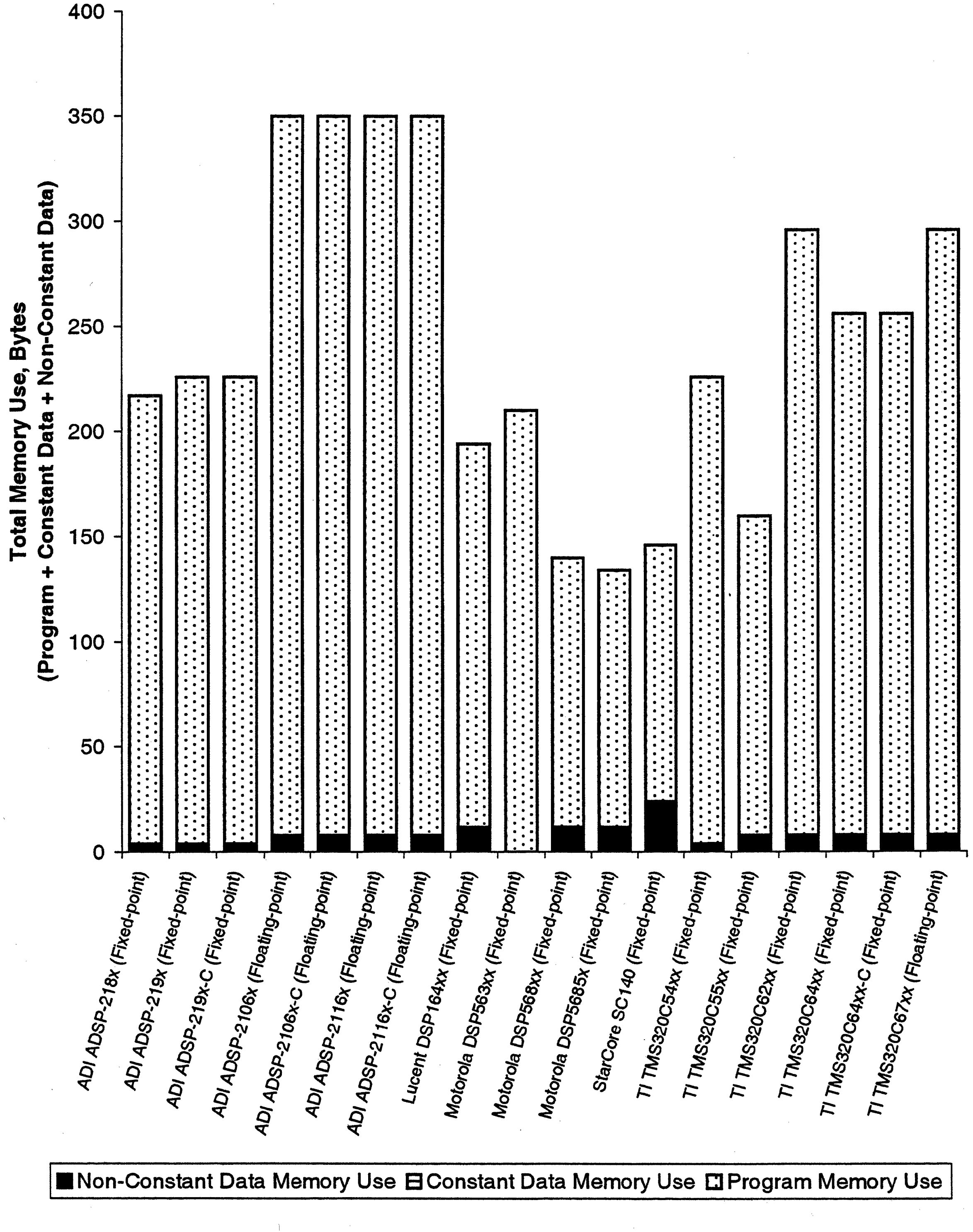


Figure 8.5-9B. Control ROMable Memory Use (Program + Constant Data)

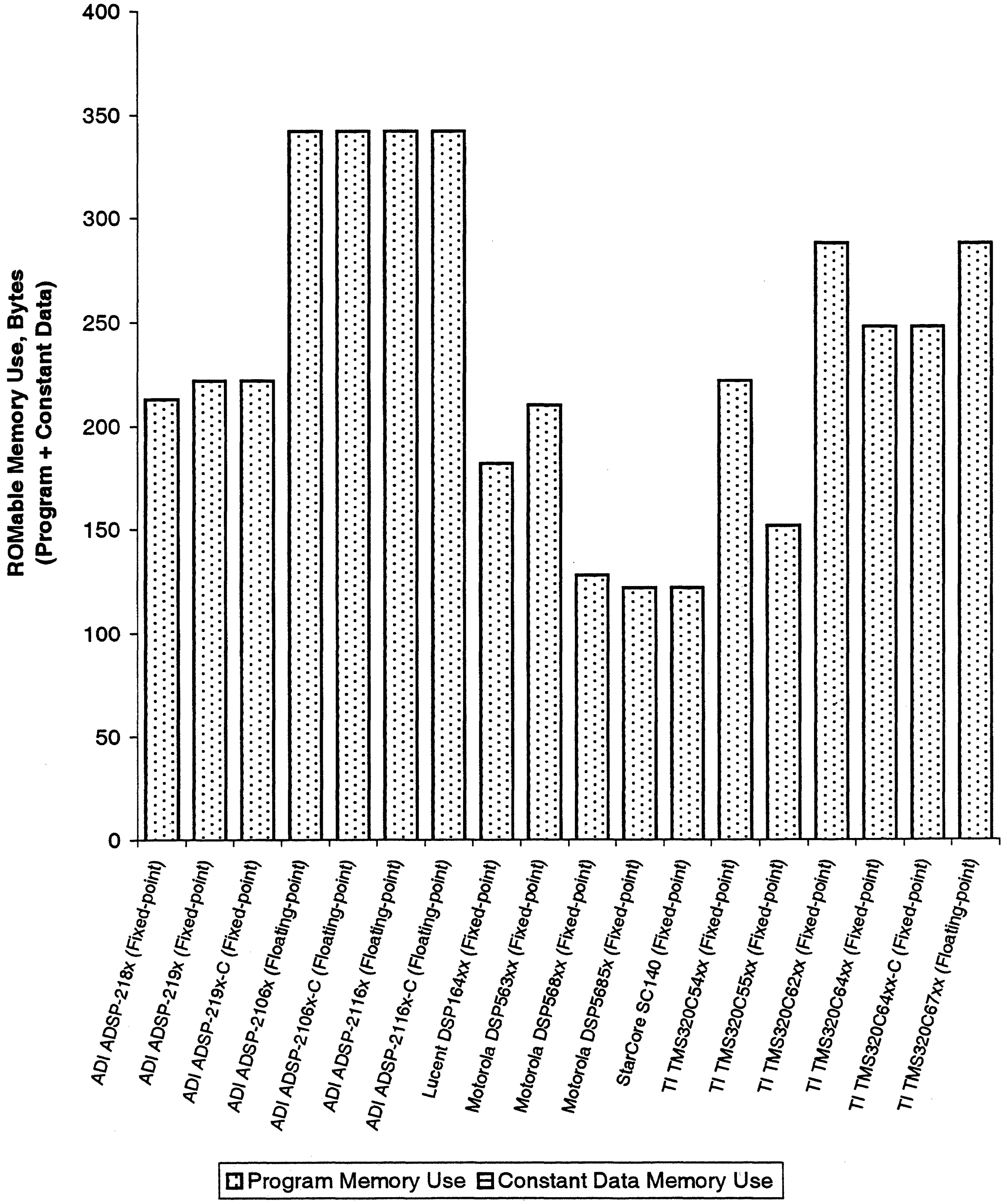


Figure 8.5-10A. 256-Point FFT
Total Memory Use

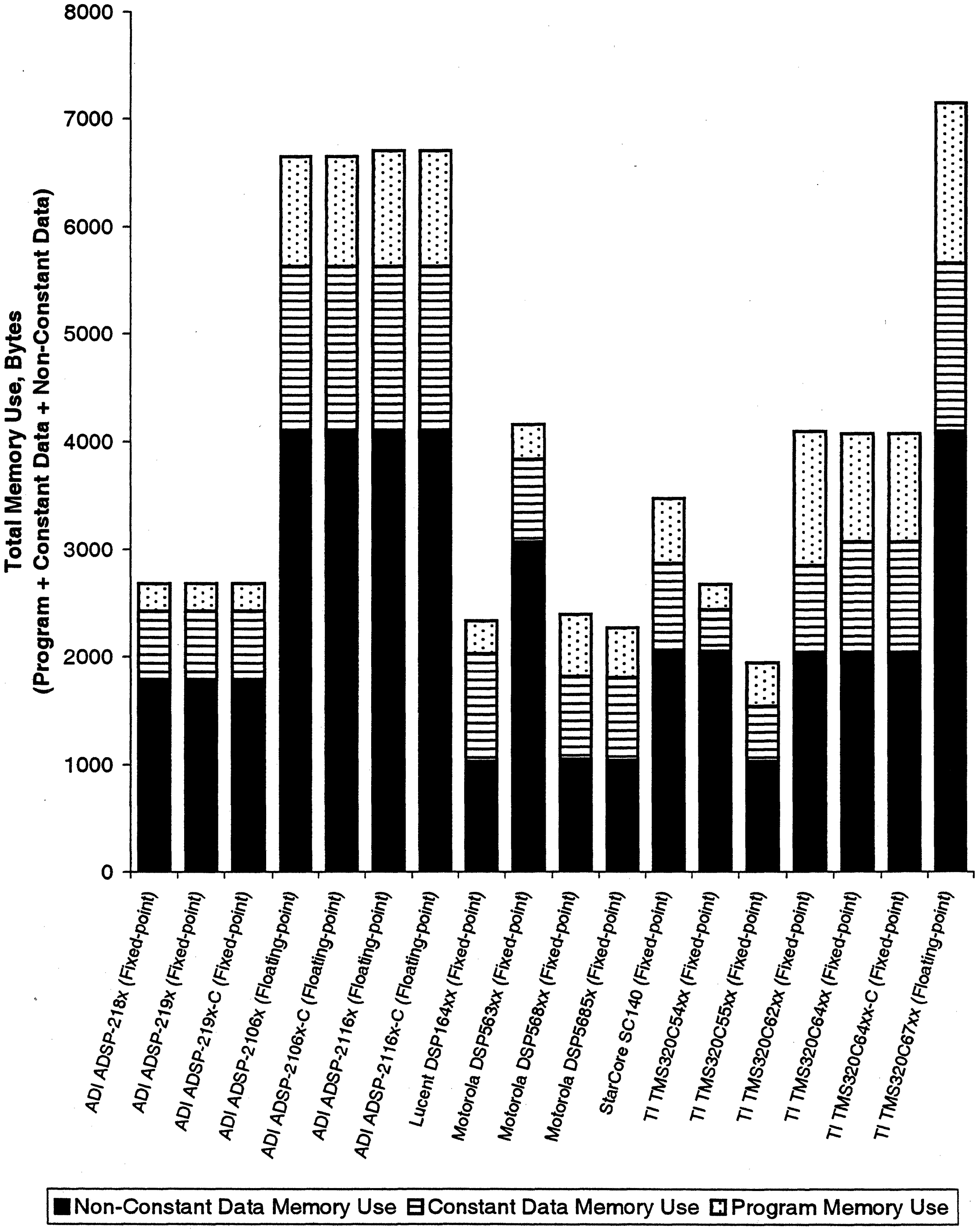


Figure 8.5-10B. 256-Point FFT ROMable Memory Use (Program + Constant Data)

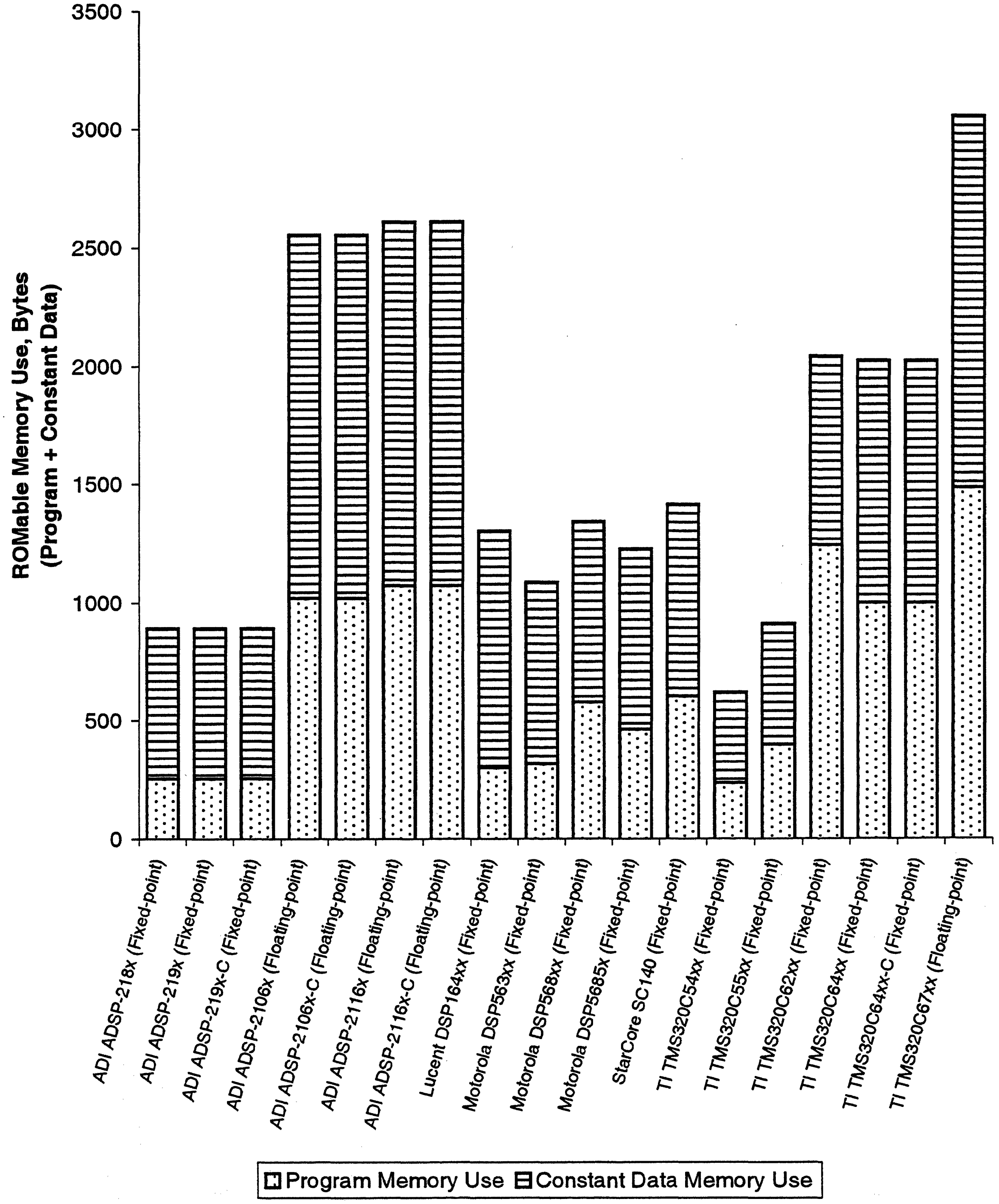


Figure 8.5-11A. Viterbi Total Memory Use

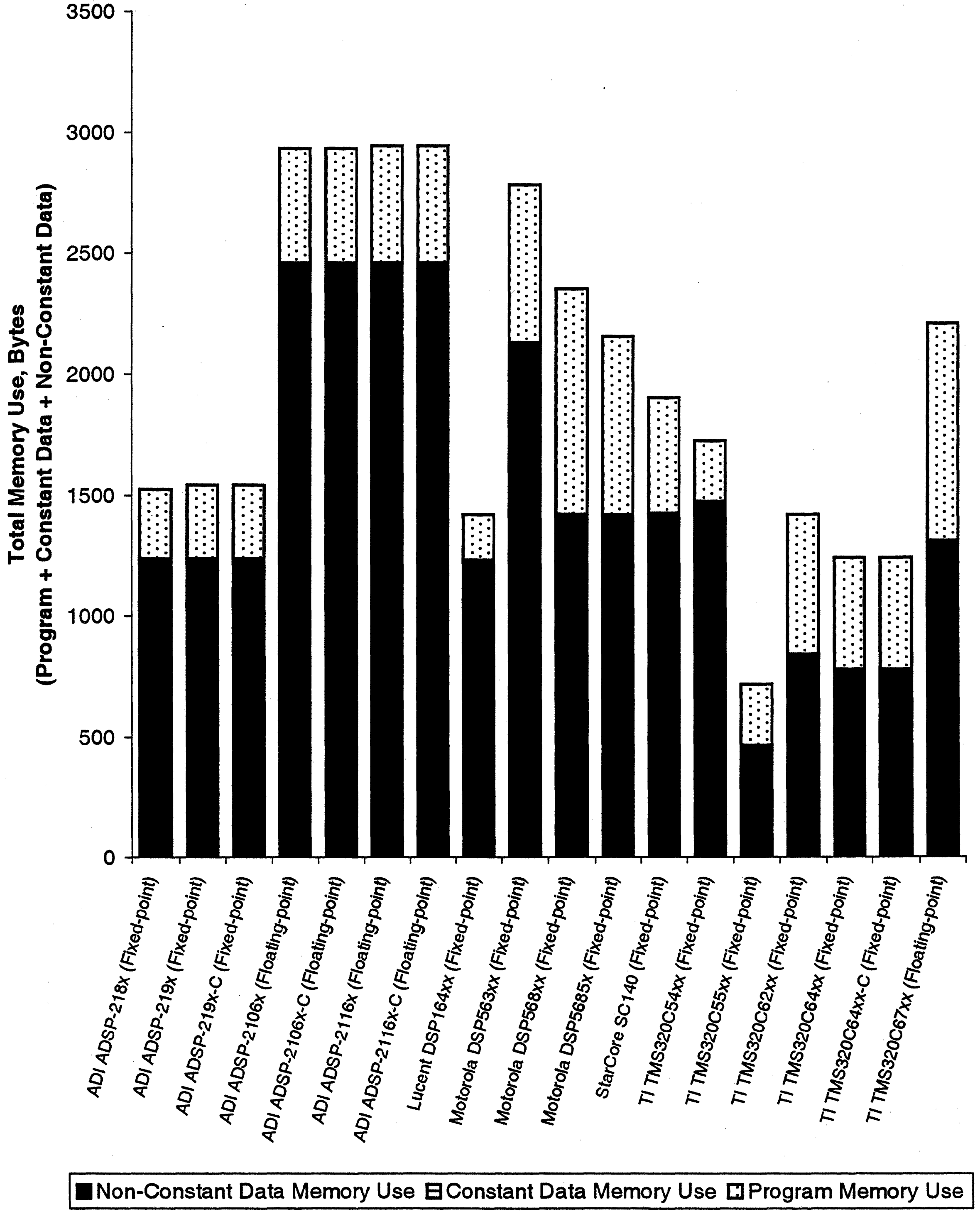


Figure 8.5-11B. Viterbi ROMable Memory Use (Program + Constant Data)

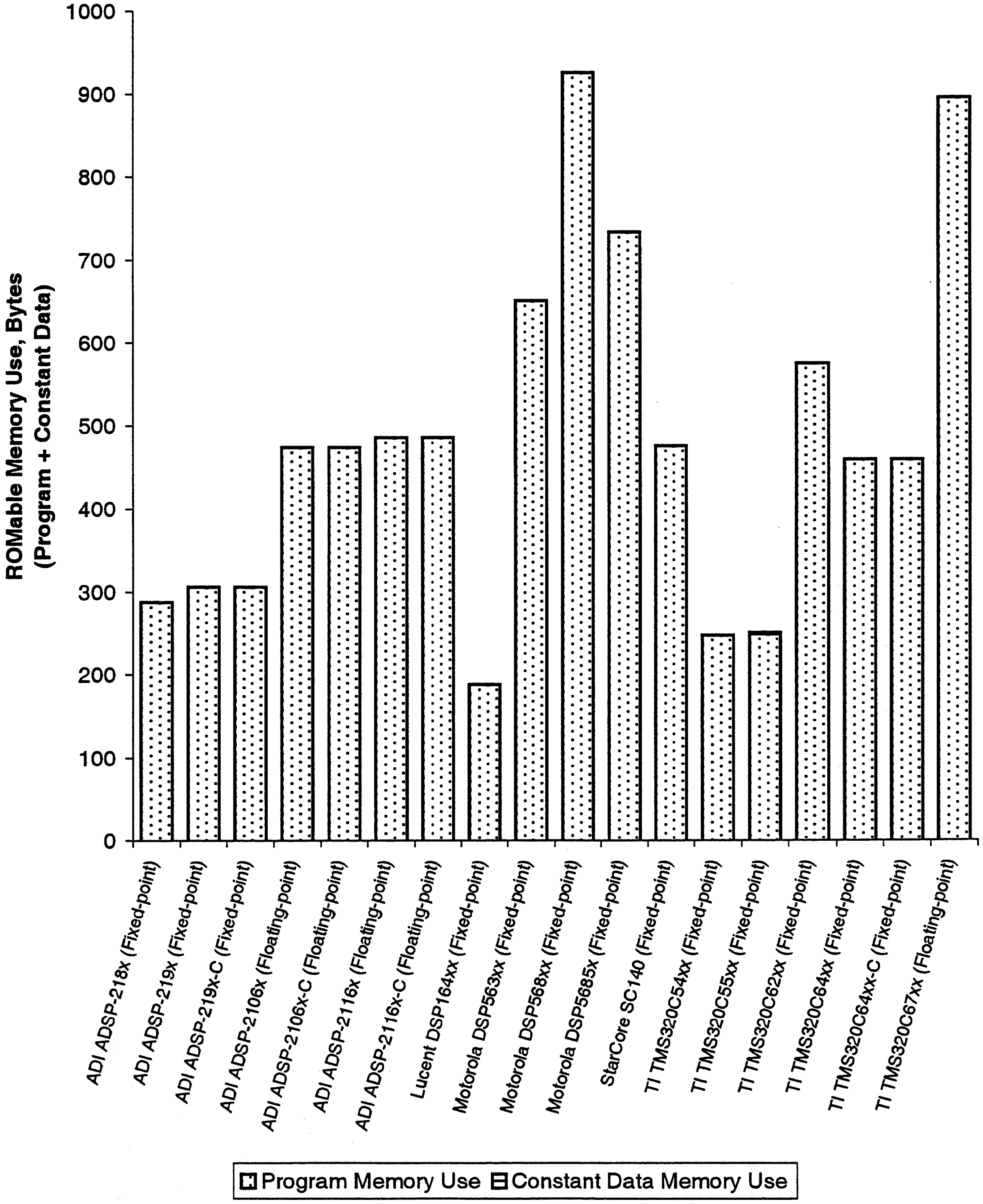


Figure 8.5-12A. Bit Unpack Total Memory Use

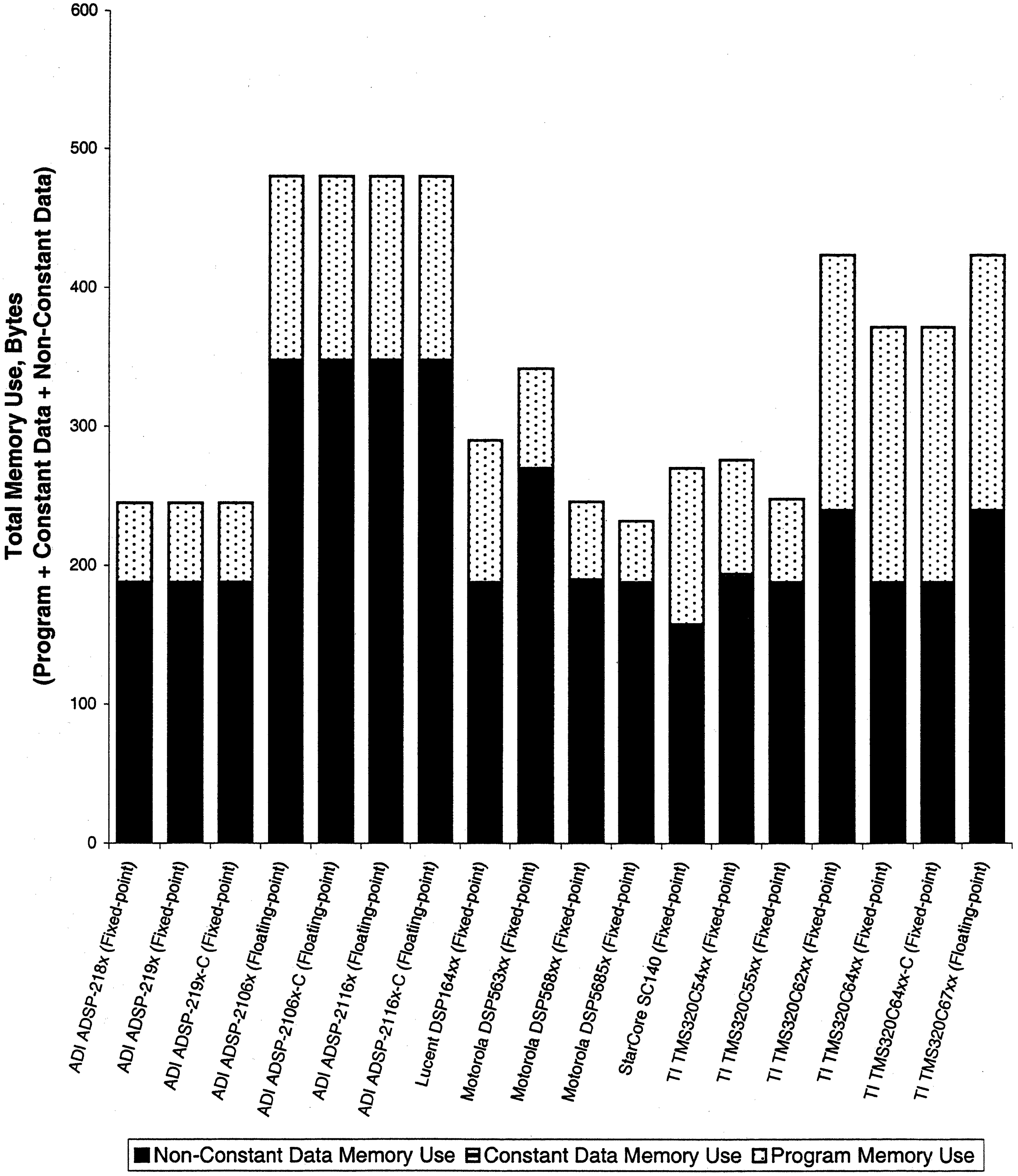


Figure 8.5-12B. Bit Unpack ROMable Memory Use (Program + Constant Data)

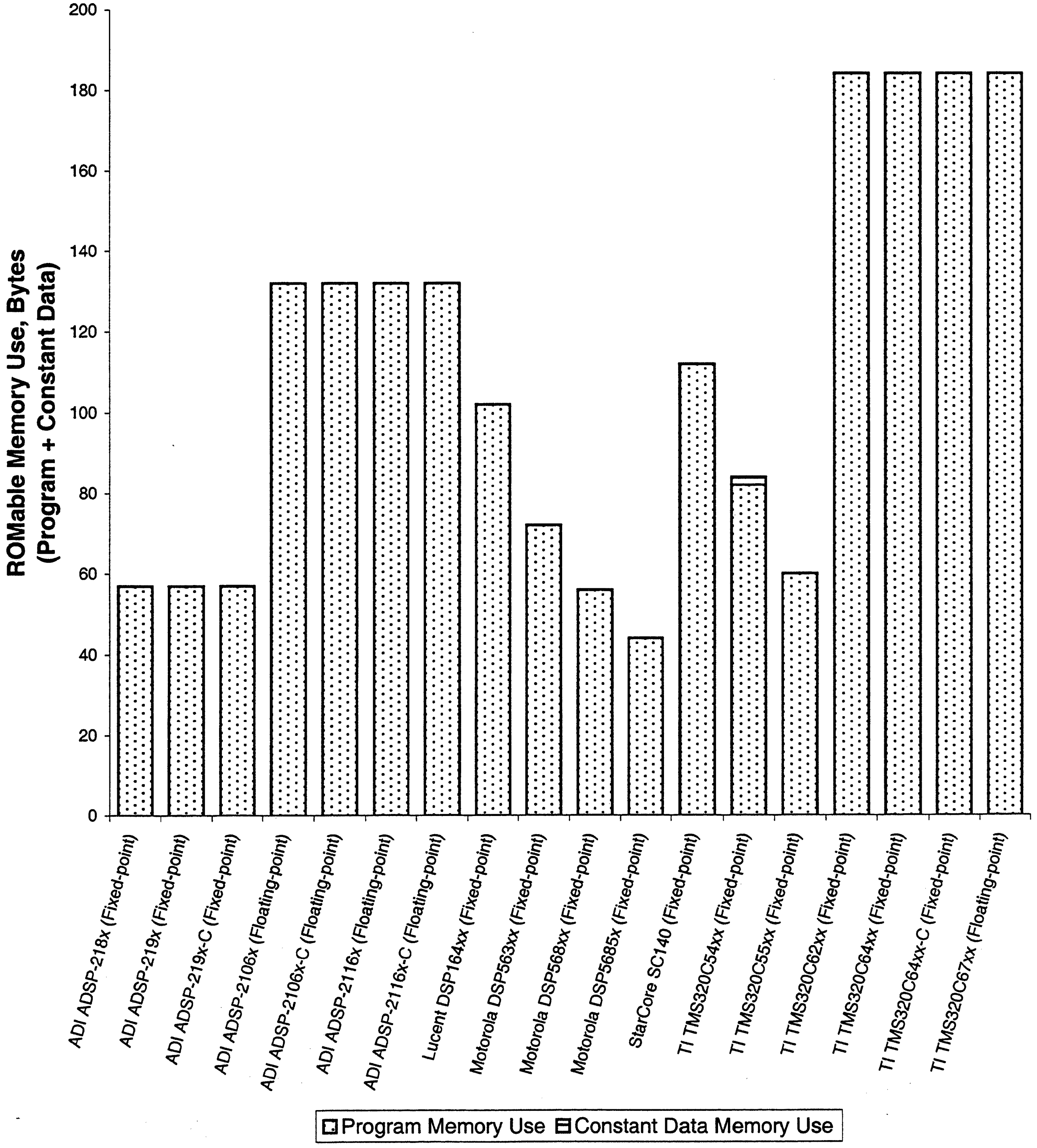


Figure 8.5-13. Normalized Program Memory Use
Sum for All b of MP[p,b]/AMP[b}

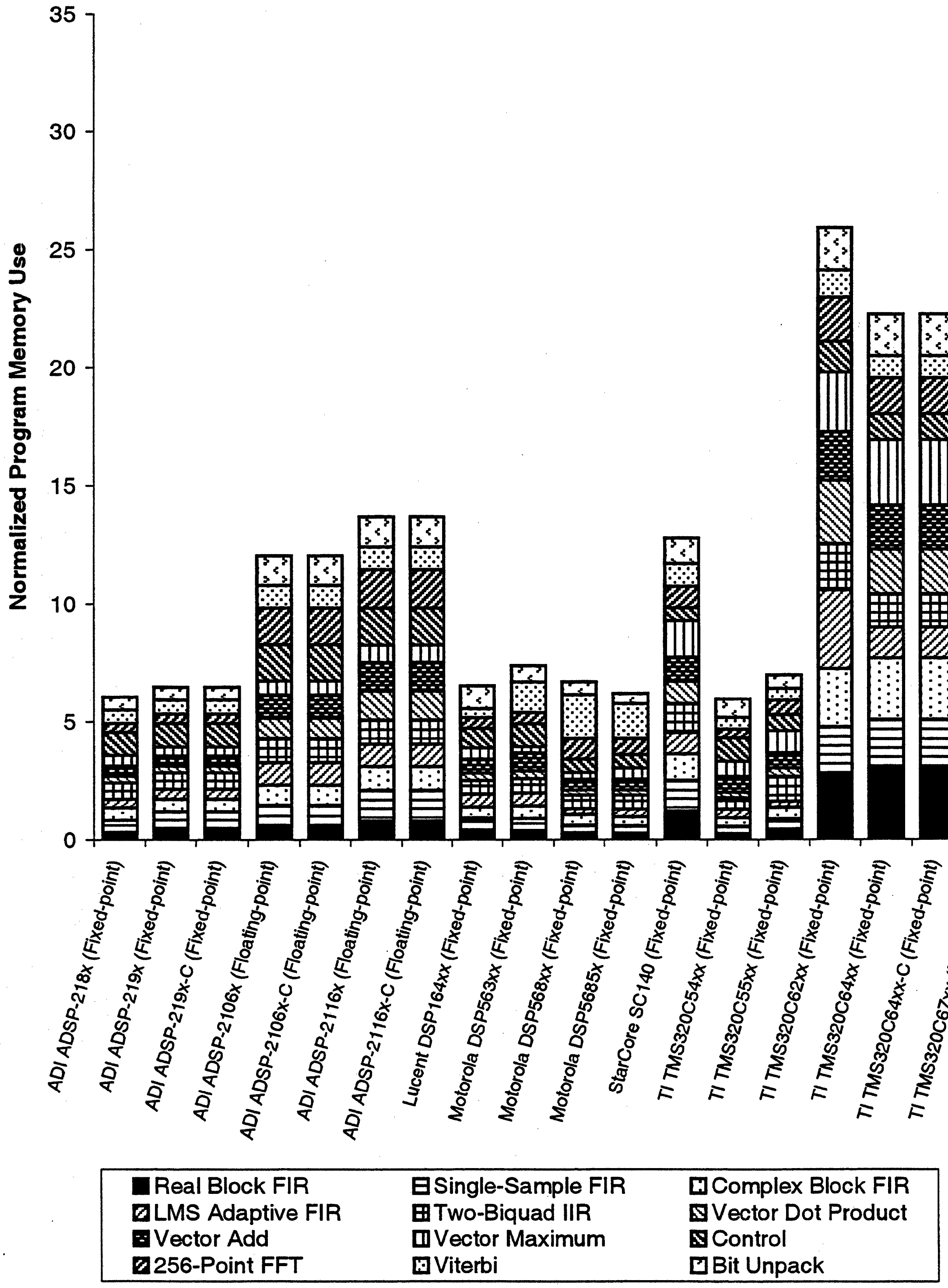


Figure 8.5-14. Normalized Constant Data Memory Use Sum for All b of MC[p,b]/AMC[b]

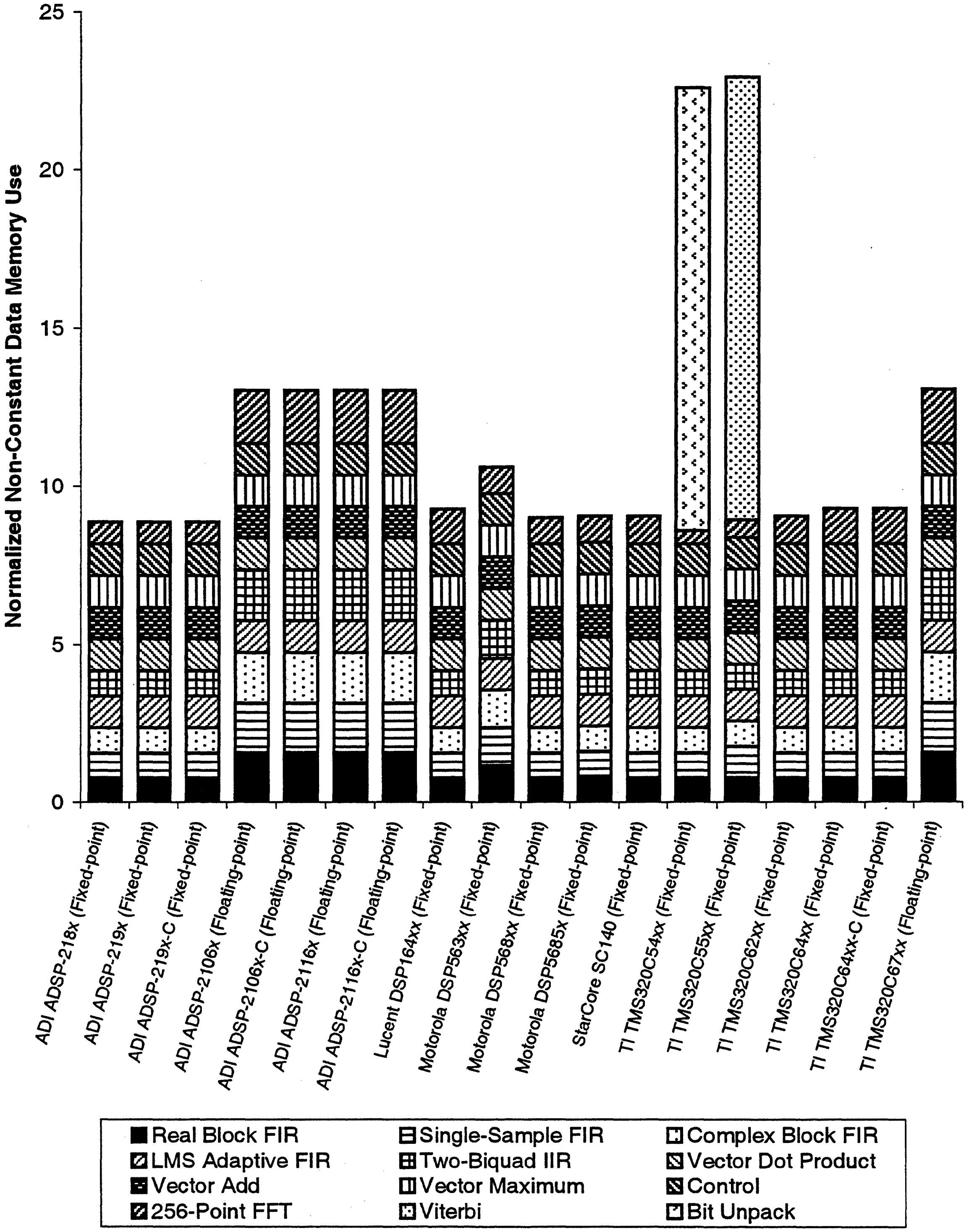


Figure 8.5-15. Normalized Non-Constant Data Memory Use Sum for All b of MD[p,b]/AMD[b]

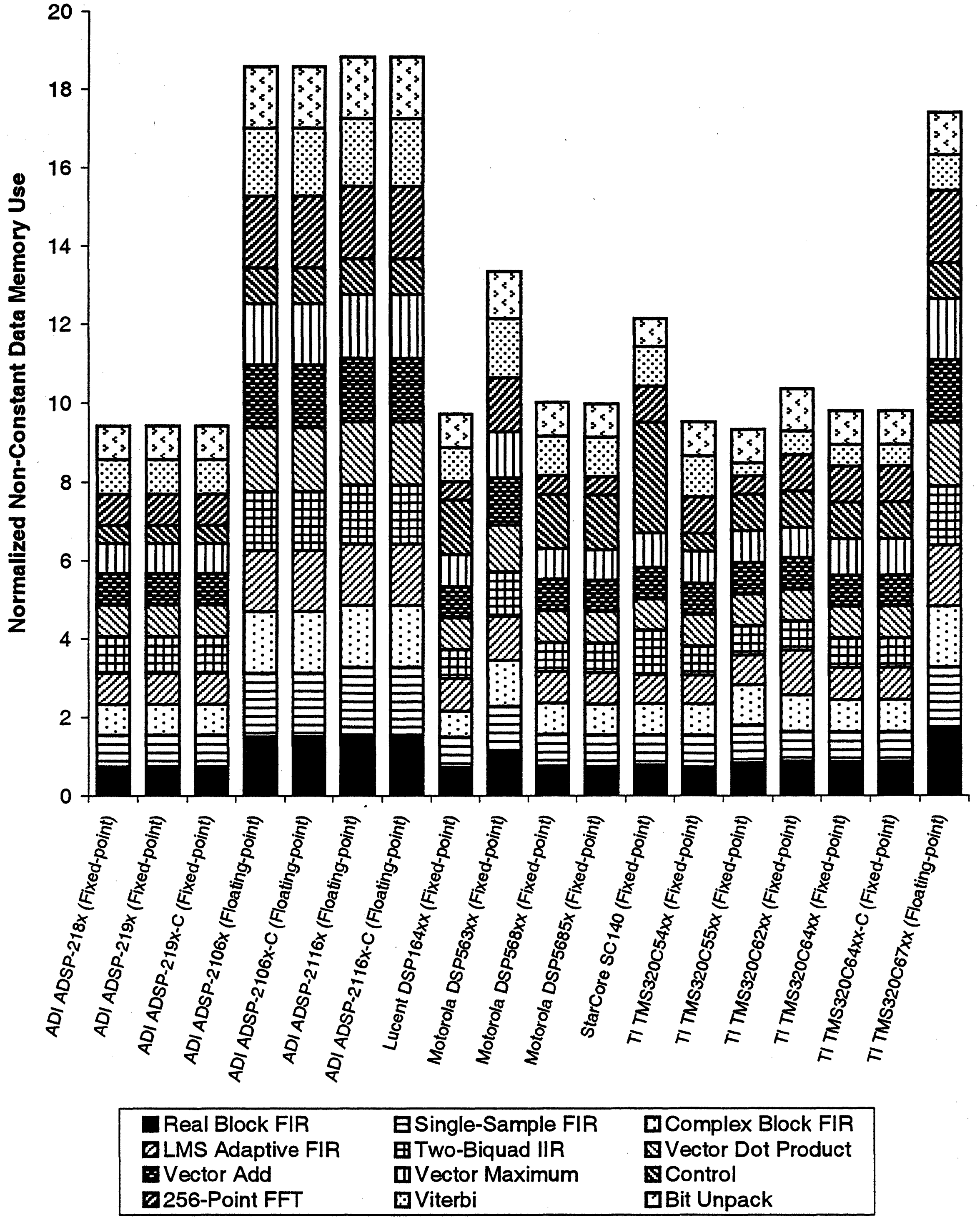


Figure 8.5-16. Normalized ROMable Memory Use (Program + Constant Data) Sum for All b of MPC[p,b]/AMPC[b]

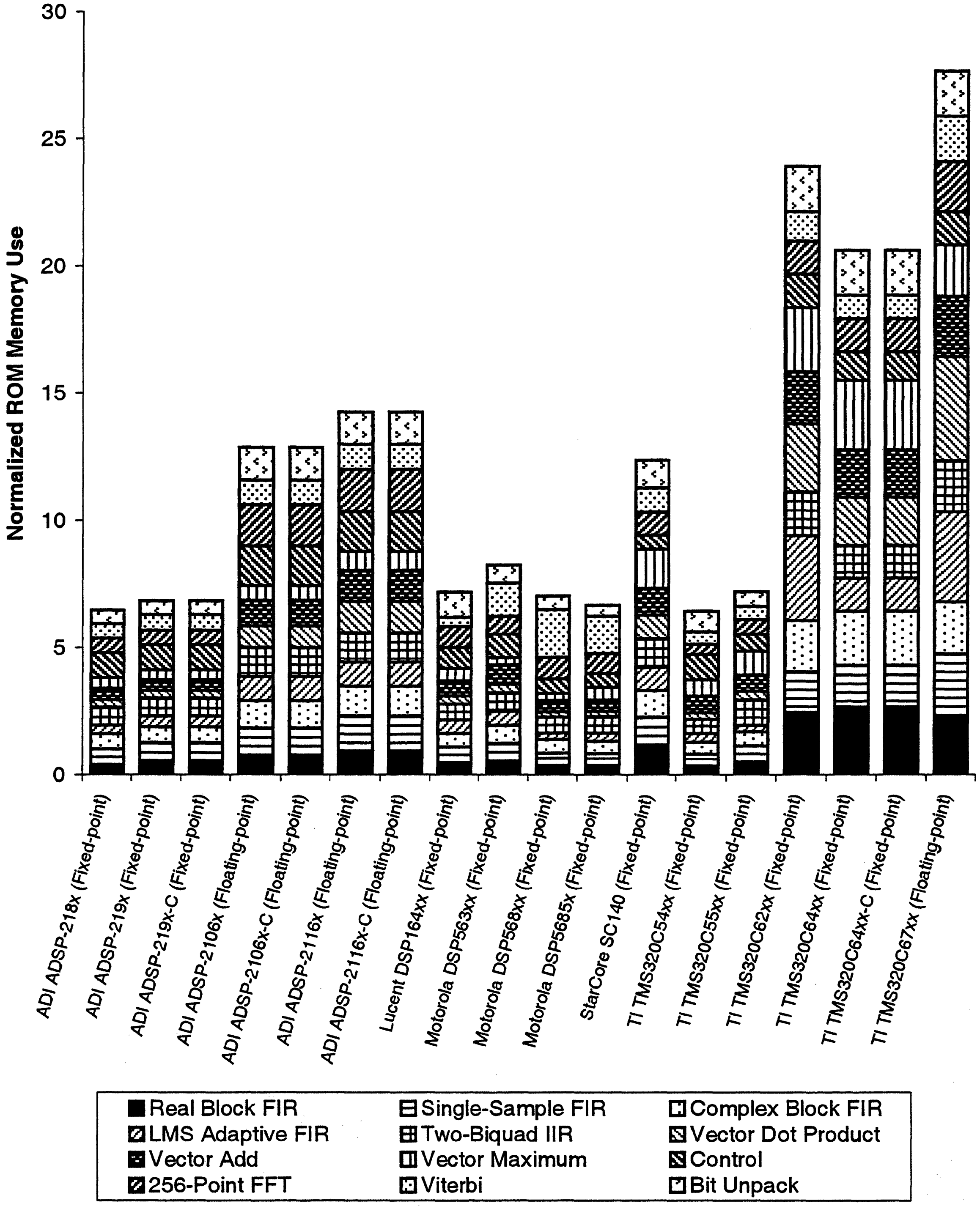
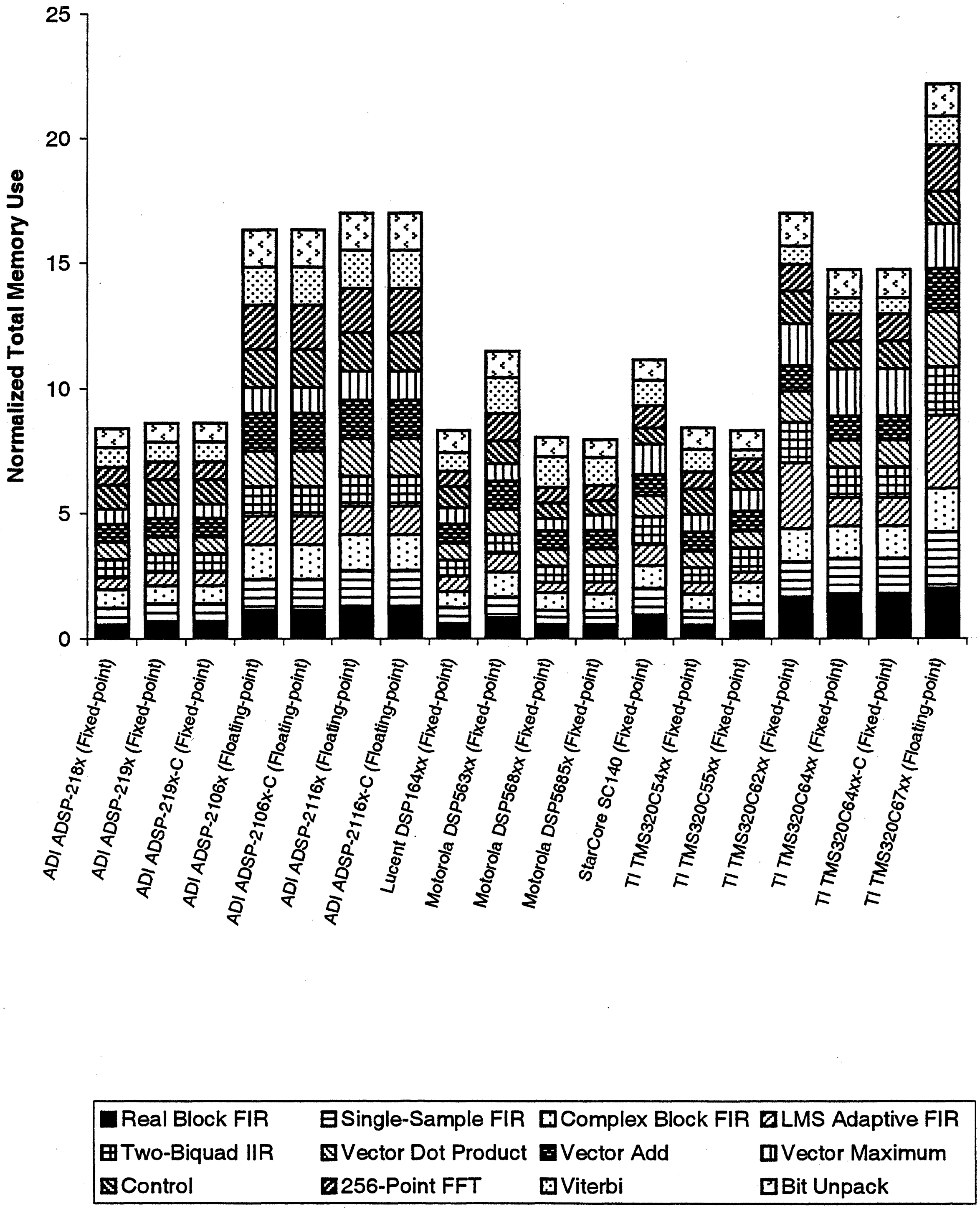


Figure 8.5-17. Normalized Total Memory Use Sum for All b of MPCD[p,b]/AMPCD[b]



9. Conclusions

In this chapter we briefly review the broad themes of this report, summarize our findings, and provide a perspective on the state of the art and the future of DSP processors and applications.

9.1 Comparing Processors

The first commercially successful DSPs appeared in the early 1980s. For the 15 years that followed, DSP processor architectures changed little. Architectural improvements were mostly incremental and vendors concentrated on increasing clock speeds, lowering power consumption, and integrating more on-chip memory and peripherals. As a result, many DSPs shipping in volume today strongly resemble the commercial DSPs of the 1980s.

The past few years, however, have seen dramatic changes in DSP architectures. Today, DSPs use a wide range of architectural techniques. Most major manufacturers have adopted VLIW-based designs for their high-performance processors. At the same time, many of these same vendors also offer conventional and enhanced conventional DSPs. For example, Texas Instruments' product line is based on *nine* different instruction set architectures, ranging from low-cost devices for applications such as disk-drive servo control to high-performance devices for applications like cellular telephone base stations.

What has motivated this rapid diversification in architectures? One key consideration is that, fundamentally, DSP applications themselves are very diverse: DSPs are found in products ranging from hearing aids to satellites; from air conditioners to telephone company central office switches. Not surprisingly, these applications have vastly different requirements. In addition, the market for DSP processors has grown rapidly, to an estimated $6 billion in 2000. The diversity of application requirements and the increasing number of competitors aiming for a piece of this growing market have combined to create the unprecedented range of DSP processor architectures available today.

Even within the somewhat constrained category of DSP processors, choosing among processors has recently become more complex due to the rapidly increasing complexity and expanding diversity of DSP processor architectures. DSP processors now use a wide variety of innovative and complex architectures; understanding these architectures can require a significant investment of time and energy. It is important, though, not to focus exclusively on architectures. A processor's architecture is important in that it contributes to achieving a particular level of speed, cost, energy consumption, programmability, and so forth, but ultimately it is these latter factors, rather than the architecture itself, that determine the suitability of a processor for an application.

Different processors clearly excel in different areas. One important attribute is execution speed. Even among processors with comparable instruction execution rates, application execution speed can vary significantly. Gaining insight into real differences in processor performance is a challenge. In general, performance measures of all kinds should be regarded with caution—this is complex territory, and the temptation to stretch the truth is powerful. Comparisons based on MIPS and MFLOPS are particularly suspect, given the large differences in the amount of useful processing that different processors can accomplish with a single instruction or floating-point operation. It is our firm belief that benchmarks, if carefully chosen and fairly implemented, provide invaluable insights into processor performance. Benchmark results, however, should not be accepted at face value. To draw meaningful conclusions from benchmarks, one must look beneath the surface to understand what the benchmarks really measure. Further, as our benchmark results vividly illustrate, it is critical to use appropriate benchmarks, relevant to the content of the application at hand; relative performance can vary dramatically from one type of algorithm to another.

Additionally, processor users should keep in mind that many features important to processor selection cannot readily be quantified via benchmarking. Applications support, quality of development tools, ability of the vendor to consistently deliver its products, and I/O performance are all examples of processor selection criteria that are not captured in typical benchmark results. Users therefore must weigh both qualitative and quantitative considerations when choosing a processor.

9.2 DSP Processors Trends and Challenges

Achieving Balance

Over the past few years, many vendors have attempted to attract attention by claiming to have produced the "world's fastest DSP." While the veracity of these claims is often questionable, they do raise an important question: Who cares? When comparing processor performance, there is a tendency to focus exclusively on speed. But for embedded applications, often speed is *not* the most important aspect of a processor's performance. In fact, for the most commercially important DSP applications, obtaining adequate processing speed is not the major challenge. Rather, the challenge is obtaining an appropriate level of processing speed while minimizing system cost, energy consumption, memory use, size, and development effort and risk. These factors often must be traded off against one another and different applications place different weights on each factor. Achieving an appropriate balance for a wide range of applications is a key challenge facing processor vendors and users alike. Doing so depends only in part on processor architecture; many other factors, such as fabrication technology and development tools, are also critical.

Not surprisingly, processor vendors make different trade-offs in designing their processors. Many recent DSP processors, such as Texas Instruments' TMS320C62xx, have pursued maximum speed, which is useful for applications such as network infrastructure equipment where many channels aggregated in a single piece of equipment create an enormous appetite for performance. Others, such as Analog Devices' ADSP-2116x, have emphasized compatibility, gaining less dramatic performance improvements but providing a simpler upgrade path for users of previous-generation devices. Still others have pursued energy efficiency, critical in the expanding variety of high-volume, battery-powered DSP applications ranging from MP3 players to hearing aids. The diversity of DSP applications means that there are opportunities for a variety of architectural approaches to succeed. As a result, we expect to see more non-traditional architectures emerge for use in DSP applications in the coming years.

Although much attention has been given recently to the emerging wave of new, complex, high-performance DSP architectures, we should note that the vast majority of the DSPs shipping today use conventional DSP architectures (similar to those of the first commercially successful DSPs); this will continue to be true for some time. Partly this is due to the natural time lags associated with the design and market penetration of new system-level products, but it is also due in part to the fact that for many applications, conventional DSP processors offer a very attractive mix of performance, cost-effectiveness, energy efficiency, and development infrastructure.

Compiler-Friendliness

One of the most interesting and significant changes in DSP processor architecture in the last few years is the importance placed on compiler-friendliness by processor architects. Whereas the mainstream DSPs of 1995 were very poor compiler targets, those of 2000 are in many cases much better compiler targets, increasing the likelihood that users will have access to efficient and robust high-level language compilers. This has become critical as the size of typical DSP application software has increased from hundreds of lines of source code to tens of thousands. Some of the newer VLIW-based DSPs, such as the StarCore SC140, incorporate RISC-like instruction sets, uniform register files, and other features designed to simplify compiler code generation. Of course, designing a compiler-friendly processor architecture does not ensure that an efficient compiler will become available. To help users judge the relative efficiency of compilers, BDTI has recently begun benchmarking C compilers for DSP applications. Our early results show that order-of-magnitude differences in compiler efficiency are common.

Floating-Point DSPs: Uncertain Future

Choices in floating-point DSPs remain relatively narrow. Of the major DSP vendors, only Analog Devices and Texas Instruments offer floating-point DSPs, and there are very few licensable DSP cores with floating-point support. This reflects the fact that floating-point DSPs comprise a very small fraction of the DSP processor market. Going forward, high-performance floating-point DSPs will face increasingly serious competition from high-performance CPUs, such as the Motorola PowerPC 7400, many of which are significantly faster than the DSPs in typical DSP benchmarks.

Growing Competitiveness of General-Purpose Processors

Separate from the high-performance general-purpose processors intended for use as CPUs in computer systems, general-purpose processors that target embedded applications are increasingly offering DSP capabilities.

A wide variety of techniques has been used to add DSP capabilities to existing embedded general-purpose architectures. Some processor designers have made modest enhancements to their processors' instruction set architectures; for example adding a single-cycle multiply-accumulate operation, as has been done on the Integrated Device Technology R46x0 and Motorola ColdFire. Others have extensively renovated their existing architectures; for example, the Hitachi SH-DSP adds a DSP data path, memory architecture, addressing modes, and other features to the SH-2 microcontroller. Other vendors have opted for a co-processor approach, as Massana has with its licensable DSP co-processor cores. In addition, many vendors continue to develop single-chip multiprocessors, combining a DSP core and a microcontroller core in a single device. All of these approaches have the potential to achieve both strong DSP capabilities and strong microcontroller capabilities. In addition, we are seeing the emergence of a class of new, hybrid architectures, such as the Infineon TriCore, designed from the ground up to address both DSP and microcontroller tasks. Such processors are blurring the traditional distinctions between DSPs and general-purpose devices.

Today, general-purpose processors often effectively meet the needs of DSP applications. In some cases, the performance of general-purpose processors in DSP applications is surprisingly strong. In addition, these processors often have much broader tool support than is typically available for DSP processors. These factors, combined with the fact that many existing system product designs already contain one or more general-purpose processors, make it worthwhile for users to consider implementing DSP tasks on general-purpose processors in many cases.

In DSP applications that also contain a large amount of complex, non-DSP software (for example, graphical user interfaces and network protocols), DSP processors may find it increasingly difficult to compete, because the prospect of implementing such non-DSP software on DSPs—with their comparatively limited tools and often idiosyncratic architectures—is unattractive. On the other hand, in applications that consist exclusively or primarily of DSP tasks, general-purpose processors are often at a disadvantage, because their tools may lack features important for DSP software and their software libraries often do not include important DSP components.

Just as general-purpose processors have been adopting features to improve their DSP capabilities, many DSP processors have been adopting features from high-performance CPUs to boost performance and adding other features to improve their microcontroller capabilities. For example, the Motorola DSP568xx and Texas Instruments TMS320C28xx both include features specifically intended to improve the processors' suitability for microcontroller tasks, and the LSI Logic LSI40xZ uses run-time instruction scheduling and branch prediction techniques borrowed from high-performance CPUs.

While it is straightforward for DSP processors to adopt general-purpose processor features to boost performance and enhance functionality, it is not so easy for DSPs to match the strong general-purpose development tools and infrastructure, broad industry familiarity, compatibility, and wide availability associated with the most successful general-purpose processor architectures.

Compatibility

In the world of embedded general-purpose processors, software compatibility has been an important architectural consideration for many years. Hence, today's embedded general-purpose processors often provide binary compatibility with their predecessors. For example, the ARM and MIPS families have maintained compatibility through several generations. In contrast, DSP processor designers have often not maintained software compatibility from one generation to the next. For example, the processors comprising the initial wave of high-performance, VLIW-based DSPs, including the Analog Devices ADSP-TS0xx, the StarCore SC140, and the Texas Instruments TMS320C62xx, are all incompatible with their predecessors.

As DSP applications have become larger (typically consisting of thousands or tens of thousands of lines of source code today), though, and as the "installed base" of existing DSP applications has grown, software compatibility has become an increasingly important concern among DSP users. In addition, as general-purpose processors become more competitive in DSP applications, the lack of compatibility of DSP processors increasingly stands out as a competitive disadvantage. In response to these considerations, some DSP processor designers have recently begun to make compatibility a priority in designing new processors. For example, Texas Instruments' two newest designs, the TMS320C64xx and TMS320C55xx, are both compatible with their predecessors. (The TMS320C64xx is object-code compatible with the TMS320C62xx and the TMS320C55xx is assembly source-code compatible with the TMS320C54xx.) Nevertheless, limited compatibility among different processors from a single vendor (for example, the TMS320C55xx and TMS320C64xx are not software compatible) continues to be a concern for users and a weakness compared to some general-purpose processor families.

Scaling

Scaling is one approach to creating families of processors sharing a single instruction set, but with a wide range of performance, price, and energy efficiency levels. The idea of scaling is to vary the complement of execution units (and, commensurately, memory bandwidth and other resources), while maintaining at least a minimum complement of resources required to efficiently execute the common instruction set. Processors created in this way may have very different levels of parallelism, but are mutually compatible. Among the various types of DSP processor architectures in use today, VLIW-based designs lend themselves more naturally to scaling than do conventional or enhanced conventional architectures.

StarCore has highlighted the scalability of the SC100 instruction set via its recent introduction of the SC110, a scaled-down version of the SC140 containing only one ALU/MAC/bit-field unit (compared with four on the SC140). StarCore expects that processors based on the SC110 core will be competitive for applications such as digital cellular phones. If so, this will be the first VLIW-based DSP from a major DSP processor vendor to address this type of cost- and energy-sensitive application.

For users, scalability can be attractive, allowing a single instruction set—and hence a single set of tools and software—to be used across a range of applications with different speed, price, power, and integration needs. For example, developers of communications systems can often gain significant simplifications by using compatible processors for both base stations and terminals. For vendors, scaling a single instruction set offers efficiencies in the provision of tools, documentation, training, application software, and support.

Examples of scaled DSP processor families are few and recent. As a result, it remains to be seen whether this approach will be successful in creating highly competitive processors for vastly different classes of applications—spanning, for example, very low-cost and low-power applications as well as very speed-hungry applications.

9.3 Business and Infrastructure Trends and Challenges

In this section we briefly discuss some of the significant trends that we have observed related to DSP processors, apart from trends related to the architectures of the devices themselves, which are discussed above.

Increased Competition

As markets for DSP-capable processors continue to grow, it is natural to expect increased competition. This competition comes from many sources. New entrants, large and small, are flooding the DSP processor business. Perhaps the most dramatic example of this is Intel's recent re-entry into the DSP processor business via its architecture collaboration with Analog Devices. In addition, general-purpose processors increasingly pose serious competition for DSP processors in many applications, as discussed above. And for maturing applications, application-specific standard products are often a viable alternative to DSPs. Finally, as we discuss below, alternatives such as FPGAs and custom hardware are attractive for some applications.

For vendors, this increased competition means an increasingly demanding business and technical environment. For users, it means expanded options and more complex choices.

Increasing Collaboration Among Chip Vendors

Over the past two years, two DSP architecture collaborations have been formed among major vendors: Lucent and Motorola have formed StarCore, and Intel and Analog Devices have engaged in a similar joint effort for the design of a next-generation DSP architecture. The results so far from StarCore are promising; Analog Devices and Intel have yet to reveal the results of their collaboration.

If successful, such collaborations offer several benefits. A pooling of resources may allow partners to develop better products faster than they could working separately. Costs can be reduced, because instead of each vendor developing its own tools, application software components, documentation, and so on, these resources can be developed once for the shared architecture. Also, attracting the support of third-party tool and application software developers may be easier if the third-party vendors believe that the shared architecture will create a significant market for their products. Finally, users may be attracted by the potential availability of software-compatible products from the two partner chip vendors.

History shows, however, that such collaborations are tricky to manage. The partner companies must cooperate on architecture development (including planning future generations of products years into the future) while simultaneously competing to sell chip products based on their jointly designed architectures; being competitors and effective collaborators at the same time is a major challenge.

Tools and Infrastructure Emerge as Critical Components of Success

Tools for DSP processor software development and debugging have made significant advances over the past few years. For example, Microsoft Windows-based graphical debuggers are now common, and nearly all vendors have functional C compilers for their DSPs. Most DSPs provide on-chip emulation support with JTAG-compatible interfaces and an increasing number provide code profiling tools.

Nevertheless, as discussed earlier in this chapter, tools continue to be a weak link for DSP processor users. A vast amount of production software for DSP processors (especially fixed-point DSPs) is written in hand-optimized assembly code using very rudimentary tools. As DSP processors have become increasingly powerful, cost-effective, and energy-efficient, the biggest challenge facing users is often the efficient development of hardware and software using these processors. DSP applications have become much larger and more complex, time-to-market windows have become shorter, and skilled developers are in short supply. As a result, for many users, the selection of a processor hinges increasingly on the quality of the available tools, application software components, technical support, and other development infrastructure.

Code-generation tools, in particular, represent the greatest challenge for processor developers. As vendors adopt more powerful and more complex architectures such as the VLIW-based designs discussed above, the need for truly efficient code-generation tools increases. Developing efficient code-generation tools for these complex processors, however, is not straightforward. Thus, we expect both users and vendors to place increased importance on code-generation tools (especially C compilers) in the next few years, paralleling the increased emphasis on compiler-friendliness of new architectures, discussed above.

Software libraries, including building-block functions (like filters and transforms) and application modules (like speech coders and modulators) are essential enablers for processor users who want to deploy products quickly while focusing their development efforts on features that differentiate from—rather than duplicate—those already available elsewhere. DSP processor vendors are actively courting third-party software developers and, as mentioned above, are developing their own application software components in some cases. Nevertheless, the availability of off-the-shelf software remains spotty.

Some vendors (for example, Texas Instruments) have invested aggressively in strengthening their development tools and infrastructure through acquisitions, internal efforts, and facilitating the work of third-party developers; others have not. Relatedly, when a third-party developer considers supporting a new architecture, a key factor in its decision is the number of users expected to create new designs with the architecture. This helps to explain why Texas Instruments, with its inclusive market strategy, broad product line, and large market share, enjoys very strong third-party support while Lucent, with its specialized product line and narrowly focused market strategy, has very little third-party support—despite having the second-largest market share in DSPs. Indeed, we believe that concerns about development tools and infrastructure were a major factor motivating Lucent and Motorola to form their StarCore partnership. The partnership allows the two companies to pool their resources and their user bases to foster internally developed and third-party tools, application software, and other infrastructure.

Factors such as the different market and product orientations of processor vendors, and differing levels of investment, are creating a widening gap between the best-supported and the least-supported DSPs in terms of the quality and capabilities of the available tools and infrastructure. This makes it increasingly critical for users to evaluate these items thoroughly before selecting a processor. Going forward, we expect that success or failure of a new DSP architecture will be increasingly influenced—in some cases decisively—by the quality of the associated tools and other development infrastructure. The importance of tools and infrastructure also helps explain why general-purpose processors are becoming increasingly attractive for DSP applications.

Vertical Integration Accelerates

As applications become larger and more complex, processor vendors are increasingly providing more than just processors. Tools and other development support, mentioned above, are an important example of this. Another example is complementary chips, such as analog-to-digital and digital-to-analog converters designed to integrate easily with a DSP. Yet another, increasingly important example is sophisticated application software, such as xDSL modem software (which Analog Devices, Texas Instruments, and Lucent all offer for selected DSPs). Such software, bundled with the DSP, allows the processor vendor to offer a "black box" application solution to system developers, which in turn helps system developers to successfully create products without mastering all of the associated signal processing content, and without making the sometimes immense investment required to develop high-quality implementations of functions such as modems and audio compression.

As with tools, the past few years have seen accelerating investment by some DSP processor vendors in application software for specific markets. And, as with tools, the availability of appropriate application software is often a critical factor in users' processor-selection decisions. Going forward, we expect to see further increases in the level of investment by DSP processor vendors in application software.

System-on-Chip (SoC) Designs Become Mainstream

More and more of the most important DSP applications place a priority on low cost, low energy consumption, and small size. Ever-improving IC fabrication technology allows more circuitry to be packed into a single small chip. The combination of these trends suggests that an increasing number of products will use specialized chips that combine a DSP core with other significant modules, such as a microcontroller, application-specific accelerators, and custom hardware. Processor vendors have the capacity to design only a very limited number of such specialized devices. Therefore, we expect that an increasing number of system designers and specialty chip companies will create their own DSP core-based SoC devices targeting specific application areas or even a single end-product design.

DSP cores are a less mature technology than packaged DSP processors, and there are still significant barriers to their effective, widespread use. For example, the number of DSP core offerings today is fairly limited, and the required tools and expertise for creating a DSP core-based SoC are often lacking. A major challenge to the widespread use of DSP cores remains the complexity of delivering a DSP core design, along with associated documentation, diagnostics, and tools, to multiple design teams, each with a unique set of design methodologies, know-how, and requirements. Core vendors and EDA tool developers have begun to recognize this problem and are taking steps towards addressing it. For example, the VSI (Virtual Socket Interface) Alliance is developing and promoting standard interfaces for licensable electronic intellectual property such as DSP cores.

Despite many challenges, there is little doubt that DSP cores will play an increasingly important role in the coming years. Unfortunately for users, most of the established DSP processors are either not available as cores, or are available on a very limited basis (for example, only to selected customers, or only for extremely high-volume products). Some vendors of packaged DSP processors do provide "foundry-captive" DSP cores. Texas Instruments, for example, makes some of its cores available for use in high-volume SoCs designed by customers and fabricated by Texas Instruments. Several core specialists, such as DSP Group, offer licensable DSP cores—licensees can combine these cores with building blocks from other vendors or of their own design and fabricate the resulting chip wherever they choose.

Going forward, the proliferation of SoC designs poses a significant challenge for established DSP processor vendors. These processor vendors cannot hope to directly engage with all of the users who wish to create SoC designs using their processors. Therefore, many SoC designers will be forced to look elsewhere for their DSP cores unless the established vendors decide to make their processors available as licensable cores. In recent years, a few DSP processor vendors have begun to offer their architectures as licensable cores. For example, Infineon offers its Carmel architecture as a licensable core, and LSI Logic licenses its LSI40xZ design. These two companies, relatively new to the DSP processor market, hope to increase the chances for success of their architectures by enabling customers to create their own SoC designs without direct involvement from the processor vendor. In addition, Motorola has announced that it will license its new DSP56800E core.

Specialization Continues

As markets for DSP processors expand, it becomes attractive to tailor processors to the needs of specific applications or classes of applications. Processors that are specialized in this way can gain a competitive advantage by focusing resources where they have the most benefit for the application. While this trend is not new, over the past few years we have observed an increase in the number of specialized DSPs.

Specialization can take place in many ways. Among the most obvious are the inclusion of co-processors or accelerators, or the addition of new data path features and instructions to boost performance in certain types of algorithms. Other approaches focus on integrating an appropriate mix of on-chip memory, peripherals, and I/O interfaces. In the extreme, these devices cease to be sold as programmable "processors," and are instead sold as application-specific standard products (ASSPs), with all necessary software provided by the processor vendor.

The degree and specificity of tailoring varies. For example, Motorola's DSP56307 is intended for GSM base stations, and some members of Zoran's ZR38xxx family are designed exclusively for Dolby AC-3 audio decompression applications. In contrast, processors like Texas Instruments' TMS320C54xx have features that make them well suited to wireless telecommunications applications, but most of these features aren't specific to one particular wireless application.

Another manifestation of the trend towards specialization is the size of DSP processor families. Some vendors have created very large families of DSPs based on a common core architecture but offering varying levels of performance and cost and different types of integration and specialization. Perhaps the largest is Analog Devices' ADSP-21xx family, which includes over 30 distinct products, including devices specialized for applications such as motor control.

Some types of specialization, such as integrating application-specific functionality into a processor's data path and instruction set, are inconsistent with the trend towards more RISC-like architectures, which is motivated by the need to create efficient compilers. To resolve this conflict, we expect that in the future many DSP processor vendors will create specialized chips by combining generalized cores with special-purpose co-processors and on-chip peripherals. Other vendors, though, will likely find it worthwhile to sacrifice compiler-friendliness in exchange for performance and efficiency by incorporating application-specific features into the core architecture.

Other, emerging approaches to specialization include enabling users to modify the processor architecture, and combining programmable logic with a processor to provide a "reconfigurable processor." For example, Tensilica, a core licensor, uses the former approach, while Chameleon Systems, a chip vendor, uses the latter. Going forward, we expect that both of these approaches will have success in DSP applications.

9.4 Conclusions

Broadening Options

Today more than ever before, selecting among implementation options for DSP applications is a complex task rife with subtle trade-offs. Users have an unprecedented range of options for implementing DSP tasks, including:

- Conventional DSP processors, such as the Texas Instruments TMS320C54xx.
- Enhanced conventional DSP processors, such as the Analog Devices ADSP-2116x.
- High-performance, VLIW-based DSP processors, such as the Motorola MSC8101.
- High-performance general-purpose processors, such as the Intel Pentium III.
- Embedded-oriented general-purpose processors with DSP enhancements, such as the Hitachi SH3-DSP.
- Customizable processors, such as the ARC processor core.
- Reconfigurable processors, such as the Chameleon Systems CS2000.
- Application-specific standard products, such as MP3 player chips.
- Programmable logic devices, which increasingly are provided with DSP-oriented design tools and intellectual property.
- Custom-designed hardware, which may incorporate hard-wired logic in combination with one or more of the above-listed technologies.

Choosing among this proliferation of offerings can be a daunting task. In many applications, the decision isn't simply a matter of picking the single "best" implementation technology. Rather, the most successful system designs are typically heterogeneous—using multiple technologies, each for the portion of the system where it best meets the needs of the application.

For those applications where a DSP processor is appropriate, no one architecture, and no single processor, can effectively serve the needs of all, or even most users. For this reason, we expect to see continuing specialization and diversification in DSP processor architectures, and we expect that different DSPs will succeed in different applications. General-purpose processors will increasingly challenge DSPs in many applications, and the line between general-purpose processors and DSPs will continue to blur. Nevertheless, we expect that in the next few years, DSPs will dominate in many of the most important, most signal-processing-intensive applications, such as advanced cellular phones, xDSL modems, and voice-over-packet networks.

As a result of the continuing acceleration in DSP processor architectural innovations, the task of comparing and selecting processors is much more complex than it was just a few years ago. Understanding the key attributes of all of the new types of DSP and DSP-enhanced general-purpose architectures can be a complex task; understanding the implications of those attributes for an application can be daunting. Users, therefore, must be prepared to invest significant time in selecting a processor.

For chip vendors, the path ahead includes many new options. Although the leading DSP chip vendors have all adopted VLIW techniques for their fastest devices, those same companies are in many cases continuing to develop more conventional DSP architectures to target applications that demand a different balance of speed, cost, power consumption, and compatibility with previous-generation processors.

Appendix A. Vendor Contact Information

The following tables provide contact information for both vendors of DSP processors and for development tool and library vendors mentioned in this report.

DSP Processor Vendors

Vendor	Address	Telephone, Fax, Electronic Mail, Website	Processor Families
Analog Devices, Inc.	1 Technology Way P. O. Box 9106 Norwood, MA 02062-9106	(781) 329-4700 (800) 262-5643 - Sales (800) 446-6212 - Lit. Faxback Email: cpd_support@analog.com http://www.analog.com	ADSP-218x ADSP-219x ADSP-2106x ADSP-2116x TigerSHARC
Lucent Technologies, Inc.	1247 S. Cedar Crest Blvd. Allentown, PA 18103	(800) 372-2447 (610) 712-4323 (610) 712-4106 - Fax (610) 712-4593 - Tech. Supp. Fax Email: research@stlouis.cp.lucent.com Tech. Supp. web site - http://www.lucent.com/micro/wam/tse http://www.lucent.com	DSP16xxx StarPro
Motorola, Inc.	MD OE314 6501 William Cannon Drive Austin, TX 78735	(800) 521-6274 (847) 538-8099 (512) 895-2030 - Marketing (512) 895-3230 - Tech. Support (512) 895-4665 - Fax Email: help@mot.com http://www.mot.com	DSP563xx DSP568xx DSP5685x MSC81xx
StarCore Technology Center	2100 Riveredge Parkway, Suite 600 Atlanta, GA 30328	(770) 618-2500 (770) 937-4534 http://www.starcore-dsp.com	SC140
Texas Instruments, Inc.	12500 TI Blvd. Dallas, TX 75243	(281) 274-2320 (281) 274-2324 - Fax Email: dsph@ti.com http://www.ti.com	TMS320C2xxx TMS320C3x TMS320C54xx TMS320C55xx TMS320C62xx TMS320C64xx TMS320C67xx

DSP Software and Development Tool Vendors			
Vendor	**Address**	**Telephone, Fax, Electronic Mail, Web Site**	**Products**
Berkeley Design Technology, Inc. (BDTI)	2107 Dwight Way Second Floor Berkeley, CA 94704	(510) 665-1600 (510) 665-1680 Email: info@BDTI.com http://www.BDTI.com	Optimized DSP software; expertise in digital audio
Cadence Design Systems, Inc. (formerly Alta Group)	555 N. Mathilda Avenue Sunnyvale, CA 94086	(408) 733-1595 (408) 523-4601- Fax http://www.altagroup.com	High-level DSP system, software and hardware development tools
DSP Software Engineering, Inc. (Voice-Over-IP solutions acquired by Tellabs)	175 Middlesex Turnpike Bedford, MA 01730	(781) 275-3733 (781) 275-4323 - Fax Email: info@dspse.com http://www.dspse.com	Application libraries for Texas Instruments processors
Domain Technologies, Inc.	1700 Alma Drive Suite 495 Plano, TX 75075	(972) 578-1121 (972) 578-1086- Fax Email: info@domaintec.com http://www.domaintec.com	In-circuit emulators for Motorola DSPs
ENEA OSE Systems, Inc.	101 Metro Drive Suite 680 San Jose, CA 95110	(408) 392-9300 (408) 392-9301 - Fax Email: infor@enea.com http://www.enea.com	Real time OS
GO DSP, A Texas Instruments Company	150 John Street Suite 899 Toronto, Ontario M5V 3E3 Canada	(416) 599-6868 (416) 599-7171 - Fax Email: sales@go-dsp.com http://www.go-dsp.com	GO DSK visual development environment for Texas Instruments DSPs
Green Hills Software, Inc.	30 West Sola Street Santa Barbara, CA 93101	(805) 965-6044 (805)965-6343 - Fax Email: sales@ghs.com http://www.ghs.com	DSP software development tools
ILLICO	2700 Augustine Drive Suite 145 Santa Clara, CA 95054	(408) 980-8170 (408) 980-9327 - Fax http://www.illico.com	Fax, modem software
Metrowerks	9801 Metric Blvd. Austin, TX 78758	(800) 377-5416 (512) 997-4700 (512) 997-4901 - Fax Email: info@metrowerks.com http://www.metrowerks.com	DSP software development tools

DSP Software and Development Tool Vendors

Vendor	Address	Telephone, Fax, Electronic Mail, Web Site	Products
Motorola India Ltd.	Semiconductor Products Sector Area 1B, The Senate No.33A, Ulsoor Road Bangalore 560 042 INDIA	+ 91 80 559 8615 + 91 80 559 4687 - Fax http://www.apspg.com/regional/bangalore.html	DSP software development services
Synopsys	700 E. Middlefield Road Bldg. C Mountain View, CA 94043	(800) 388-9125 (650) 962-5000 http://www.synopsys.com	High-level DSP system, software and hardware development tools
Tasking Inc.	333 Elm Street Dedham, MA 02026-4530	(800) 458-8276 (781) 320-9400 (781) 320-9212 - Fax Email: sales.us@tasking.com or support.us@tasking.com http://www.tasking.com	DSP software development services
Voice Pump, Inc.	299 California Avenue Suite 200 Palo Alto, CA 94306	(650) 323-3232 (650) 323-4222 - Fax Email: sales@voicepump.com http://www.voicepump.com	Real-time speech coding and telecommunications software for various DSPs
Wideband Computers, Inc.	1350 Pear Avenue Mountain View, CA 94043	(650) 962-8722 (650) 962-8790 - Fax Email: info@wideband.com http://www.wideband.com	Optimized software libraries for TI, ADI processors

Note: The above includes only vendors mentioned in this report; it is not a comprehensive list.

References

[BDTI96] Berkeley Design Technology, Inc., *DSP Processor Fundamentals*. Fremont, California: Berkeley Design Technology, Inc., 1996.

[BDTI97] Berkeley Design Technology, Inc., *DSP on General-Purpose Processors*. Fremont, California: Berkeley Design Technology, Inc., 1997.

[Lyon97] Richard G. Lyons, *Understanding Digital Signal Processing*, Reading, Massachusetts: Addison Wesley, 1997.

[Oppe89] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, New Jersey: Prentice-Hall, 1989.

[Stra98] Will Strauss, *DSP Strategies 2002*. Tempe, Arizona: Forward Concepts, 1998.

Bibliography

Here we list a few publications that provide more depth on some of the topics covered in this report. See the References section for publications referenced earlier in this report. Visit BDTI's Web site, *http://www.bdti.com,* for reprints of many articles related to DSP processors.

[All85] Jonathan Allen, "Computer Architecture for Digital Signal Processing," *Proceedings of the IEEE*, Vol. 73, No. 5, May 1985: 852(22).

[Ana90] Analog Devices, DSP Division, *Digital Signal Processing Applications Using the ADSP-2100 Family* (Volume 1). Edited by Amy Mar. Englewood Cliffs, New Jersey: Prentice Hall, 1990.

[Ana95] Analog Devices, DSP Division, *Digital Signal Processing Applications Using the ADSP-2100 Family* (Volume 2). Edited by Jere Babst. Englewood Cliffs, New Jersey: Prentice Hall, 1995.

[Ata91] B. S. Atal, V. Cuperman, and A. Gersho, *Advances in Speech Coding*. Kluwer Academic Publishers, 1991.

[Bur85] C. S. Burrus and T. W. Parks, *DFT/FFT and Convolution Algorithms: Theory and Implementation*, with TMS32010 programs by James F. Potts. New York: Wiley, 1985.

[Cha90] Rulph Chassaing and Darrell W. Horning, *Digital Signal Processing with the TMS320C25*. New York: Wiley, 1990.

[Els90] Mohamed El-Sharkawy, *Real Time Digital Processing Applications with Motorola's DSP56000 Family*; with appendices provided by the Applications Engineering Staff of Motorola's DSP Operation. Englewood Cliffs, New Jersey: Prentice Hall, 1990.

[IEE87] Institute of Electrical and Electronics Engineers, Inc., *IEEE Standard for Radix-Independent Floating-Point Arithmetic*. New York: Institute of Electrical and Electronics Engineers, Inc., 1987.

[Jon88] Douglas L. Jones and Thomas W. Parks, *A Digital Signal Processing Laboratory Using the TMS32010*. Englewood Cliffs, New Jersey: Prentice Hall, 1988.

[Lee88] Edward A. Lee, "Programmable DSP Architectures: Part I," *IEEE ASSP Magazine*, October 1988: pp. 4-19.

[Lee89] Edward A. Lee, "Programmable DSP Architectures: Part II," *IEEE ASSP Magazine*, January 1989: pp. 4-14.

[Lev98] Marcus Levy, "EDN's 1998 DSP-Architecture Directory," *EDN*, vol. 43, no. 9, April 23, 1998: p. 40.

[Man88] M. Morris Mano, *Computer Engineering Hardware Design*. Englewood Cliffs, New Jersey: Prentice-Hall, 1988.

[Shea88] David Shear, "EDN's DSP Benchmarks," *EDN*, September 29, 1988: pp. 126-148.

[Sie82] Daniel P. Sieworek, C. Gordon Bell, and Allen Newell, *Computer Structures: Principles and Examples*. New York: McGraw-Hill, 1982.

Glossary

Absolute addressing
Alternative term for memory-direct addressing. See also: memory-direct addressing.

Accumulator
A register used to hold the output of the ALU or multiply-accumulate unit. DSP processors typically have from one or more accumulators. On fixed-point processors, accumulators are usually at least twice as wide as the processor's native data word width (in bits) and may be wider.

AC-3
A compression scheme for high-fidelity digital audio, developed by Dolby Laboratories. Also called "Dolby Digital."

A/D converter
(Analog-to-digital converter.) A circuit that converts an analog voltage into a numeric (digital) representation.

Address generation unit
A unit responsible for generating effective addresses.

Address match breakpoint
A debugging feature provided by in-circuit emulators and instruction-set simulators. The emulator or simulator halts processor execution when the processor attempts to access program or data from a specified memory address.

Address space
The range of memory addresses that can be selected by an address bus. For example, a 16-bit address bus can be used to select any of 65,536 addresses.

Add-with-carry
An addition where the value of the carry bit from the previous operation is added to the result. See also: subtract-with-borrow.

ADPCM
(Adaptive differential pulse code modulation.) An audio compression technique.

ADSP-21xx
A family of 16-bit, fixed-point DSP processors from Analog Devices, Inc.

ADSP-219x

A family of 16-bit, fixed-point DSP processors from Analog Devices, Inc.

ADSP-21020

A 32-bit, floating-point DSP processor from Analog Devices, Inc.

ADSP-2106x

A family of 32-bit, floating-point DSP processors from Analog Devices, Inc., notable for their communications ports and large on-chip memories. Also referred to as "SHARC."

ADSP-2116x

A family of 32-bit, floating-point DSP processors from Analog Devices, Inc., notable for their dual data paths supporting SIMD operations. Also referred to as "Hammerhead."

AE

(Applications engineer.) An engineer employed by a processor vendor who is responsible for assisting customers with implementing their applications on that vendor's processors.

A-law

A European encoding standard for digital representation of speech signals. Non-uniform quantization levels are used to achieve the effect of compressing the dynamic range prior to quantization. See also: companding, μ-law.

Algorithm

A description of the arithmetic operations to be performed on a signal that does not specify how that arithmetic is to be implemented.

ALU

See arithmetic logic unit.

Analog signal processing

The manipulation of analog signals in the analog domain.

ANSI

(American National Standards Institute.) A standards-setting body. Among other standards, ANSI has defined a standard for the C programming language.

Arithmetic logic unit

A processor execution unit that is responsible for arithmetic (add, subtract, shift, etc.) and logic (*and*, *or*, *not*, *exclusive-or*) operations.

ASIC

(Application-specific integrated circuit.) An integrated circuit intended for use in a particular product or application, typically containing one or more processor cores and a variety of application- or product-specific features, such as algorithm accelerators and specialized peripherals. Also sometimes referred to as an SoC (system-on-chip).

Assembly statement in-lining

The inclusion of an assembly language statement (or statements) in high-level language code. Assembly statement in-lining is a technique commonly used to improve performance of high-level language programs.

ASSP

(Application-specific standard product.) An integrated circuit intended to be used in a range of products within a particular application field. In contrast, an ASIC is often used only in a single product. See also: ASIC, SoC, FASIC.

ATPG

(Automatic test pattern generation.) The automated generation of test data (often called test vectors or test patterns) to be used for production testing of integrated circuits or other hardware components.

BBS

(Bulletin board system.) Many DSP processor vendors provide BBSs accessible by modem that hold source code and application notes for their processors.

Big-endian

The ordering of bytes within a multi-byte data word. In big-endian ordering, bytes within a multi-byte word are arranged most-significant byte first. See also: little-endian.

Biquad filter

A second-order digital filter commonly used in signal processing. Biquads are often used as building blocks in higher-order digital filters.

Bit field

A group of bits. See also: bit-field manipulation.

Bit-field manipulation

Logical or bit operations applied to a group of bits at a time. Bit-field manipulation is a key type of operation in error control coding and decoding.

Bit I/O port

An I/O port in which each pin is individually configurable to be an input or an output and in which each pin can be independently read or written.

Bit-reversed addressing

An addressing mode in which the order of the bits used to form a memory address is reversed. This simplifies reading the output from radix-2 fast Fourier transform algorithms, which produce their results in a scrambled order.

Block floating-point

A technique for computation using a set of mantissas sharing a common exponent.

Block repeat

A hardware looping construct in which a block of instructions is repeated a number of times. See also: hardware loop.

Bootstrap loading

The capability to load and execute code from an off-chip device (such as non-volatile memory or a host processor) at power-up.

Boundary scan

A facility provided by some integrated circuits that allows the values on the IC's pins to be interrogated or driven to specified logic levels through the use of a special serial test port on the device. This is useful for testing the interconnections between ICs on a printed circuit board.

Bounding box

The boundary of a circuit component, such as a DSP processor, SoC, standard cell, or DSP core. The bounding-box definition includes a list of all signals and associated timing and electrical properties.

Branch

A change in a processor's flow of execution to continue execution at a new address.

Breakpoint

An address or condition specified by a debugger that causes the processor to stop execution when the address is reached or the condition is met. See also: address match breakpoint.

Bubble

A delay in instruction execution caused by a pipeline conflict. So called because when depicted graphically the unused pipeline slot appears to "bubble up" towards the execute stage of the pipeline.

Bus

A shared electrical connection that enables access to a resource, such as a memory shared among several hardware subsystems.

Bus-functional simulation model

A partial simulation model of a programmable processor that models activity on the pins or bounding box only. It is not capable of simulating the execution of a program. See also: full-functional simulation model, bounding box.

Butterfly

(1) An operation used in the computation of the fast Fourier transform. A fast Fourier transform butterfly computation involves multiplication, addition, and subtraction of complex numbers. (2) An operation used in the computation of the Viterbi algorithm. A Viterbi butterfly computation involves two additions, two subtractions, and two comparisons. See also: Viterbi decoding.

Byte-addressable

A memory space that can be accessed at any byte-aligned address. See also: word-addressable.

C54XDSP

A 16-bit fixed-point DSP core from IBM Microelectronics. This processor is a clone of Texas Instruments' TMS320C54x.

Cache

A (small) amount of fast memory where recently executed instructions are automatically stored to speed up execution. If the processor accesses instructions stored in the cache, a cache hit is said to occur; otherwise a cache miss occurs. DSP processors typically cache only instructions; however, some DSP families cache both instructions and data.

CAD

(Computer-aided design.)

CAE

(Computer-aided engineering.)

Cascade of biquads

An implementation for IIR filters where the transfer function is factored into second-order terms that are then implemented as biquad filters. See also: IIR.

CD2400

A 16-bit fixed-point DSP core from Clarkspur Design, Inc.

CD2450

A fixed-point DSP core with configurable data word widths from Clarkspur Design, Inc.

cDSP

(Configurable DSP.) Texas Instruments' term for DSP core-based SoCs.

CELP

(Codebook excited linear prediction.) A speech coding technique. CELP usually refers to the CELP algorithm specified in USFS 1016 which compresses speech to 4,800 bits/second. See also: USFS 1016.

Circular addressing

Another term for modulo addressing. See also: modulo addressing.

Circular buffer

A region of memory used as a buffer that appears to wrap around, i.e., the buffer uses a pointer that is automatically reset to the beginning of the buffer if the pointer is advanced beyond the last location in the buffer. Circular buffers are typically implemented in software on general-purpose processors and via modulo addressing on DSPs.

Clock cycle

The time required for one cycle of a processor's master clock. See also: instruction cycle.

Clock divider

A circuit that reduces the frequency of a processor's master clock. Programmable clock dividers allow the programmer to slow the processor's clock during times when full speed operation is not needed, thus reducing power consumption.

Clock doubler

A frequency synthesizer circuit that allows an input clock with frequency of one-half of the processor's desired master clock frequency to be used to generate the master clock. See also: phase-locked loop.

CMOS

(Complementary metal-oxide semiconductor.) A semiconductor technology that results in lower power consumption than other technologies. Most IC technology used today is based on CMOS.

Codec

(Coder-decoder.) An A/D and D/A converter for speech or telephony applications. The term "codec" carries with it an implication that the samples produced and consumed by the device are encoded using companding, but the term is not always used in this strict a fashion.

COFF

(Common object file format.) An object file format developed by AT&T. Most assembly language tools for DSP processors use and generate COFF files.

Companding

Short for compressing-expanding, a technique for reducing the dynamic range of audio signals and then later expanding it again. Companding uses a non-uniform quantization scheme that features finer quantization intervals at lower signal levels (where the input signal probabilisticly spends more time) to achieve a higher signal-to-noise ratio in most cases. Companding is used in A-law and μ-law codecs.

Conditional assignment

An operation in which a value is assigned (e.g., to a register) only if a certain condition is met. Conditional assignment can eliminate the need for a branch in many cases.

Conditional execution

A situation in which an individual instruction is executed or ignored depending upon on a certain condition. Conditional execution can eliminate the need for a branch in many cases.

Conflict wait state

A wait state inserted due to contention for a resource. Similar to a pipeline interlock, except that a pipeline interlock is usually attributable to contention for a resource in the processor's core; a conflict wait state may be attributable to contention for a resource outside of the core, such as an external memory interface. See also: wait state.

Convergent rounding

A rounding technique used to avoid the bias inherent in the conventional round-to-nearest approach. This technique attempts to randomize rounding behavior when the input value to be rounded lies exactly halfway between two output values. In half of these cases the value is rounded up, and in the other half it is rounded down.

Convolutional encoder

An error control coding technique used to encode bits before their transmission over a noisy channel. Used in modems and digital cellular telephony. Convolutional encoding is usually decoded via the Viterbi algorithm. See also: Viterbi decoding.

Core

The central execution units of a processor, excluding such items as on-chip memory and on-chip peripherals. In many cases, DSP processor manufacturers use a common core with different combinations of memory and peripherals to create a family of processors with the same architecture. Some vendors provide DSP cores that their customers can use to create their own customized application-specific ICs.

COSSAP

A block-diagram-based DSP design, simulation, and implementation environment from Synopsys, Inc.

CPP

(C preprocessor.) A preprocessor for the C programming language. CPP expands macros, filters out comments, and resolves conditional compilation directives. It is automatically invoked by the compiler. Some DSP assemblers use CPP to implement macros.

CQFP

(Ceramic quad flat pack.) A type of IC packaging.

Cycle

See instruction cycle or clock cycle.

Cycle stealing

Delaying an operation to allow access to processor resources for another operation, such as DMA.

D950-CORE

A 16-bit, fixed-point DSP core from ST Microelectronics.

D/A converter

(Digital-to-analog converter.) A circuit that outputs an analog voltage given a numeric (digital) representation of the voltage as input.

Data hazard

A condition where an instruction tries to use the result of a previous instruction before that result is available.

Data path

A collection of execution units (adder, multiplier, shifter, etc.) that process data. A processor's data path determines the mathematical operations that can be efficiently performed on that processor.

Data-stationary

An unusual instruction set style for pipelined processors where an instruction specifies the actions that should be performed on a set of data values, even if these actions are distributed over several instruction cycles and coincide with actions specified in other instructions. Contrast with: time-stationary.

DAU

(Data arithmetic unit.) Lucent Technologies' term for the data path of the DSP16xx and DSP16xxx.

Debugger

A front-end program that provides the user interface and much of the functionality of an emulator or instruction-set simulator.

Decibels

(Abbreviated dB.) A base-10 logarithmic expression of a value. When expressing the relative magnitude of a number n in decibels, the formula $20log_{10}(n)$ is used. When expressing the relative power of a signal in decibels, $10log_{10}(n)$ is used.

Delay line

A buffer used to store a fixed number of past samples. Delay lines are used to implement both FIR and IIR filters.

Delayed branch

A branch instruction where the branch actually occurs later than the lexical appearance of the instruction. In other words, one or more instructions appearing after the branch in the program are executed before the branch is executed.

Denormalized number

A floating-point value with a mantissa whose magnitude is below one. Some processors round denormalized numbers to zero. Other processors take special steps to store and process denormalized numbers. Denormalized numbers can be used to extend dynamic range without loss of precision.

Die

A single integrated circuit as a portion of a silicon wafer.

Digitize

Perform analog-to-digital conversion. See also: A/D converter.

DMA

(Direct memory access.) A mechanism by which data can be moved between on-chip peripherals and on-chip memory or between on-chip memory blocks without the processor having to execute data movement instructions.

DRAM

(Dynamic random access memory.) DRAM provides greater memory densities and is less expensive than SRAM, but is also slower and requires external circuitry to refresh it periodically. Some DSP processors provide on-chip DRAM interfaces. See also: SRAM.

DSP

(1) Digital signal processing. (2) Digital signal processor.

DSP16xx

A family of 16-bit, fixed-point DSP processors from Lucent Technologies.

DSP16xxx

A family of 16-bit, fixed-point DSP processors from Lucent Technologies, notable for containing dual multipliers.

DSP32xx

A family of 32-bit, floating-point DSP processors from Lucent Technologies.

DSP32C

A family of 32-bit, floating-point DSP processors from Lucent Technologies.

DSP560xx

A family of 24-bit, fixed-point DSP processors from Motorola, Inc.

DSP561xx

A family of 16-bit, fixed-point DSP processors from Motorola, Inc.

DSP563xx

A family of 24-bit, fixed-point DSP processors from Motorola, Inc.

DSP568xx

A family of 16-bit, fixed-point DSP processors from Motorola, Inc., enhanced with microcontroller capabilities.

DSP568xxE

A family of 16-bit, fixed-point DSP cores from Motorola, Inc., enhanced with microcontroller capabilities.

Dynamic logic

A circuit design technique used to design processors. Processors designed with dynamic logic require a minimum frequency input clock to function correctly. See also: static logic.

E1

A telecom standard for high-speed serial communication.

EDA

(Electronic Design Automation.) CAD tools for electronic design.

EEPROM

(Electrically erasable read-only memory.) An EEPROM can be erased and reprogrammed by the user multiple times. Unlike EPROM, an EEPROM does not require exposure to ultra-violet light to erase its contents. See also: flash EEPROM, EPROM, PROM, ROM.

Effective address

An address used to access memory.

Embedded system

A system containing a processor (for example, a digital signal processor or a general-purpose processor) wherein the processor is not generally reprogrammable by the end-user. For example, a modem containing a DSP processor is an embedded system. A personal computer is not.

EPROM

Erasable programmable read-only memory. An EPROM can be erased (by exposing it to ultra-violet light) and reprogrammed by the user multiple times. See also: EEPROM, flash EEPROM, PROM, ROM.

Event counter

In the context of in-circuit emulators, an event counter counts the number of times a user-specified event occurs while the processor is executing. An event may consist of, for example, access to specified program or data memory addresses, a branch taken by a program, or an external interrupt. Not all in-circuit emulators provide event counters.

Exception

An unplanned-for event that results from a software operation, such as division by zero. On some processors, interrupts are called exceptions.

Exponent

A part of the representation of a floating-point number. See also: floating-point.

Extended precision

(1) The use of data representations that provide higher precision than that of a processor's native format. (2) In the IEEE-754 standard, extended single-precision refers to floating-point numbers that are at least 43 bits wide, and extended double-precision refers to floating-point numbers that are at least 79 bits wide.

Externally requested wait state

Wait states that are requested by an external device. See also: wait state.

FAE

(Field application engineer.) An application engineer based in a vendor's field office.

FASIC

(Function- and application-specific integrated circuit.) An integrated circuit that performs a specialized, high-level function (e.g., speech coding, image compression) that is sold off-the-shelf for use in products of different companies. FASICs are sometimes referred to as ASSPs—application-specific standard products. See also: ASIC, SoC, ASSP.

Fast interrupt

An interrupt where the service routine can only execute one or two instructions but that offers reduced interrupt latency. Fast interrupts are typically used to quickly move data from a peripheral to a memory location or vice versa.

Feature size

An overall indicator of the density of an IC fabrication process. It usually refers to the minimum size of one particular kind of silicon structure or "feature," specifically the minimum length of the "channel, " or active region of a MOS transistor. The sizes of other structures on the IC are usually roughly proportional to the minimum transistor channel length. A smaller feature size translates into a smaller chip.

FFT

(Fast Fourier transform.) A computationally efficient method of estimating the frequency spectrum of a signal. The FFT algorithm is widely used in DSP systems.

Field-programmable gate array

A programmable logic chip having a high density of gates.

FIFO

(First-in, first-out.) A buffer arrangement where the first sample stored in the buffer is the first one to be retrieved. See also: circular buffer.

FIR

(Finite impulse response.) A category of digital filters. As compared to the other common category, IIR filters, FIR filters are generally more expensive to implement, but offer several attractive design characteristics. See also: IIR.

Fixed-point

An arithmetic system where each number is represented using a fixed number of digits, and of these, a fixed subset specifies the integer part, with the remaining subset specifying the fractional part. Contrast with: floating-point.

Flash EEPROM

A type of EEPROM that can be reprogrammed without the use of special voltage levels, making it suitable for use in products where field reprogrammability is important.

Floating-point

An arithmetic system for representing integers or fractions where each number is represented by three fixed-point numbers, one specifying the sign, another the "*mantissa*," and the third the "*exponent*." The value of the number being represented is computed by raising the base (usually 2) to the power given by the exponent and then multiplying it by the mantissa. The sign bit indicates whether the value is positive or negative.

FPGA

See field-programmable gate array.

FTP

(File transfer protocol.) A protocol for transferring files over the Internet. "Anonymous FTP" refers to retrieving or sending files to a public directory of files over the Internet.

Full-functional simulation model

In the context of commercial processor simulation models, a simulation model that models the processor internals and the activities on the bounding box of the processor. See also: bus-functional simulation model.

G.711

An ITU-T standard for encoding and decoding of audio signals. The standard compresses 16-bit samples of audio sampled at 8 kHz (128 kbits/second) to 64 kbits/second. The standard specifies two forms of companding: μ-law (used in North America and Japan) and A-law (used in Europe).

G.721

An ITU-T standard for speech encoding and decoding. The standard is based on ADPCM and compresses speech to 32 kbits/second. It is a subset of the G.723 algorithm. See also: ADPCM, G.723.

G.722

An ITU-T standard for audio encoding and decoding. The standard is based on sub-band ADPCM and compresses a 7-kHz bandwidth audio signal to 64 kbits/second. See also: ADPCM.

G.723

An ITU-T standard for speech encoding and decoding. The standard is based on ADPCM and compresses speech to 24, 32, or 40 kbits/second depending on the quality level desired. G.721 is G.723 running at 32 kbits/second. See also: ADPCM, G.721.

G.728

The ITU-T standard for low-delay CELP, a speech compression technique. G.728 compresses a 4-kHz audio bandwidth speech signal into a 16-kbit/second bit stream. See also: CELP.

Gate array

A digital integrated circuit consisting primarily of a regular array of cells. The final few steps of the fabrication process add customer-specified metal interconnection layers that both define the logic function of each cell and the interconnections between the cells. Because most of the fabrication steps are identical regardless of the application of the gate array, significant economies of scale are possible. See also: field-programmable gate array.

GCC

(GNU C compiler.) A C compiler developed by the Free Software Foundation. GCC forms the basis of many C compilers for DSP processors.

GDB

(GNU debugger.) A C-language source level debugger for use with GCC. See also: GCC.

GNU

(GNU's not UNIX.) The name given by the Free Software Foundation to UNIX-like programs developed independently of AT&T or U.C. Berkeley and protected by a copyright agreement requiring free distribution of source and object code for original GNU software and derivative works.

GSM

(Global System for Mobile Communications.) The standard specifying the pan-European digital cellular telephone network first installed in the early 1990s. GSM also sometimes refers to the GSM full-rate and half-rate speech coder standards. See also: TDMA.

Guard bits

Extra bits in an accumulator used to prevent overflow during accumulation operations. Most DSP processors provide from four to eight guard bits in their accumulators.

Hard real-time

A system that is required to perform certain actions within a strictly delimited set of deadlines.

Hardware loop

A programming construct in which one or more instructions are repeated under the control of specialized hardware that minimizes the time overhead for the repetition.

Hardware stack

A push-down stack implemented in hardware. This facilitates subroutine calls and interrupt servicing. In DSP processors, these are usually small, which limits the nesting depth of subroutine calls or interrupts.

Harvard architecture

A processor architecture with two separate memory address spaces. The processor fetches instructions from one space and data from the other. Most DSP processors are based on variants of the basic Harvard architecture.

Host interface (or port)

A specialized parallel port on a DSP intended to interface easily to a host processor. In addition to data transfer, some host interfaces allow the host processor to force the DSP to execute interrupt service routines, which can be useful for control. See also: host processor.

Host processor

A general-purpose computer or microprocessor. Depending upon the context, a host computer could be a PC or workstation or might be a microprocessor used for control functions in an embedded system.

I^{2}C bus

Inter-integrated circuit bus, a synchronous serial protocol used to connect integrated circuits. The I^{2}C protocol was designed and promoted by Philips.

I^{2}S

A synchronous serial protocol developed by Philips and used for transferring digital audio signals between integrated circuits or between systems.

ICASSP

(International Conference on Acoustics, Speech, and Signal Processing.) One of the main technical conferences in the DSP field.

ICE

(In-circuit emulator.) A common tool for the development of microprocessor-based systems. An ICE usually consists of an adapter that takes the place of the processor in the target system or that connects to the target processor, interface and control electronics, and software running on a host computer. Using an ICE, the engineer can interactively monitor and control the execution of the processor while it runs inside the target system. Many recently designed processors include ICE-like capabilities on-chip, with a serial port for host platform access.

IEEE standard 754

An IEEE standard for floating-point arithmetic. A number of DSP processors support IEEE-754 arithmetic. IEEE 754 is now an international standard, IEC 60559:1989.

IEEE standard 854

An IEEE standard for floating-point arithmetic. A generalized version of IEEE 754, in which the word width is not limited to 32 or 64 bits.

IEEE standard P1149.1

An IEEE standard for boundary-scan testing of integrated circuits. A small serial port conforming to this standard is frequently used on DSP processors to access on-chip debugging facilities. IEEE-P1149.1 is commonly called JTAG. See also: JTAG.

IIR

(Infinite impulse response.) A category of digital filters. As compared to FIR filters, IIR filters generally require less computation to achieve comparable results, but sacrifice certain design characteristics which are often desirable. See also: FIR.

Immediate data

An operand for an instruction that is encoded as a part of the instruction or is encoded in the following instruction word.

Indexed addressing

An addressing mode where the effective address is computed by adding the contents of two registers, or by adding a constant to the contents of a register.

Input/output

Interfaces and devices used for transferring data and control information between devices or systems (e.g., between a processor and peripheral).

Instruction cycle

The time required to execute the fastest instruction on a processor. See also: clock cycle.

Instruction-set simulator

A program that simulates the execution of programs on a specific processor. Instruction-set simulators provide a software view of the processor; that is, they display program instructions, registers, memory, and flags, and allow the user to manipulate register and memory contents.

Interlock

The delay introduced by an interlocking pipeline to resolve a resource conflict or data hazard. See also: interlocking pipeline.

Interlocking pipeline

A pipeline architecture in which instructions that cause data hazards or contention for resources are delayed by some number of instruction cycles.

Interrupt

An event that causes the processor to suspend execution of its current program and begin execution elsewhere in memory.

Interrupt latency

The maximum amount of time from the assertion of an interrupt line to the execution of the first word of the interrupt's service routine, assuming that the processor is in an interruptible state.

I/O

See input/output.

IS-54

A standard for U.S. digital cellular telephony.

IS-95

A standard for U.S. digital cellular telephony. IS-95 uses CDMA.

IS-136

A standard for digital cellular telephony. Also known as IS-54 revision C.

JEDEC

(Joint Electron Device Engineering Council.)

Joule

A unit of energy. One watt is the amount of energy used by a device consuming one joule of energy in one second.

JTAG

The informal name for IEEE-P1149.1. JTAG stands for "Joint Test Action Group," the group that defined the standard. See also: IEEE-P1149.1

Kernel

(1) Software (such as an operating system) that provides services to other programs. (2) A small portion of code that forms the heart of an algorithm.

Latency

The time it takes after an instruction has started execution until the result is available. See also: throughput.

Linker

A program that combines separate object code modules into a single object code module and resolves cross references between modules.

Little-endian

The ordering of bytes within a multi-byte data word. In little-endian ordering, bytes within a multi-byte word are arranged least-significant byte first. See also: big-endian.

Loop unrolling

A programming or compiler strategy whereby instruction sequences that are to be executed multiple times are written repeatedly rather than once within a loop. The overhead of looping is avoided or reduced.

Low voltage

The use of a supply voltage of less than the five volts that used to be standard for digital logic. This is usually done to conserve power, but can also better match a given battery technology.

LPC

(Linear predictive coding.) A speech coding and analysis technique.

LPC-10/LPC-10E

A speech coder based on an LPC algorithm that compresses speech to 2,400 bits/second. See also: LPC, USFS 1015.

M320C25 core

A 16-bit, fixed-point DSP core from 3Soft Corporation.

M320C50 core

A 16-bit, fixed-point DSP core from 3Soft Corporation.

MAC

See multiply-accumulate.

Mantissa

A part of the representation of a floating-point number. See also: floating-point.

Master clock

The highest-frequency clock signal used within a processor. The master clock frequency is typically between one and four times the instruction execution rate of the processor.

MC68356

A single-chip multiprocessor containing both a Motorola DSP56002 and MC68000 processor.

MCM

(Multi-chip module.) A packaging technology that mounts multiple integrated circuits (the dies themselves) directly on a substrate that interconnects them.

MDSP2780

A 16-bit, fixed-point DSP with a 24-bit instruction word from IBM Microelectronics.

Memory-direct addressing

An addressing mode where the address is specified as a constant that forms part of the instruction. For example, "MOV 1234,X0" moves the contents of memory location 1234 into register X0. See also: register-direct addressing, register-indirect addressing.

Meta assembly language

Assembly language that is parameterized with conditional constructs and variables, and does not contain direct specifications of processor registers. Meta assembly language is used in some code generation tools to allow more efficient code to be generated.

Micron

(Abbreviated "µm") A unit of length equal to 10^{-6} meters. Integrated circuit feature sizes are usually specified in microns, and typical sizes range from 0.25 to 0.8 microns. See also: feature size.

Mil

One one-thousandth of an inch. A unit sometimes used for describing integrated circuit die sizes.

Modifier register

A register used in the computation of addresses. Some vendors use this term to refer to a register that contains a value to be added to an address register after an access is performed with that address register (post-incrementing); this definition is used throughout this report. Other vendors use it to refer to a register that is used to configure a processor's address generation unit for a special addressing mode, such as modulo addressing or bit-reversed addressing.

Modulo addressing

An addressing mode where hardware is provided to perform modulo arithmetic when registers used for indirect addressing are incremented. This causes values larger than the modulus value to "wrap around" and is used to implement circular buffers.

MOS

(Metal oxide semiconductor.) MOS is the silicon fabrication process used to implement most programmable DSP processors. The name refers to the three layers making up a transistor. The most common type of MOS used in digital circuits is CMOS, or complementary metal oxide semiconductor.

MQFP

(Metal quad flat pack.) A type of IC package.

µ-law ("mu-law")

An encoding standard for digital representation of speech signals. Non-uniform quantization levels are used to achieve the effect of compressing the dynamic range prior to quantization. See also: companding, A-law.

µPD7701x

A family of 16-bit, fixed-point DSPs with 32-bit instructions from NEC Electronics, Inc.

Multiply-accumulate

The dominant operation in many DSP applications, where operands are multiplied and added to the contents of an accumulator register. Frequently abbreviated to "MAC." See also: multiply-add.

Multiply-add

An important operation in DSP applications, where the operands are multiplied and the result is added to the contents of a register. Multiply-add is distinguished from multiply-accumulate in that the result is written to a different register than the one that provides the addend value for the product. See also: multiply-accumulate.

Multi-ported memory

Memory that can be accessed by two or more units simultaneously.

Multi-precision

Construction of larger data words out of sequences of native-width data words in order to obtain higher precision.

Multirate DSP system

A DSP system which performs operations on signals with different sample rates.

Multitasking

Execution of two or more tasks, either on different processors so that the tasks execute simultaneously or on the same processor using time slots so that the tasks appear to execute simultaneously.

NaN

(Not a number.) The IEEE-754 specifies that floating-point processors should reserve a special representation in their numeric formats to indicate that a register or memory location does not contain a valid number. This representation is referred to as NaN.

Native word width

The number of bits used to represent the largest data operand that can be used in all arithmetic and logical operations provided by the processor's data path.

Nestable interrupts

Interrupts whose service routines can be interrupted.

Normalization

The detection and elimination of redundant sign bits in a fixed-point data word. Normalization is heavily used in block floating-point arithmetic.

Numerical C

A proposed extension to the C language including complex arithmetic and iterators. C compilers featuring Numeric C support are available for ADSP-210xx DSPs.

OakDSPCore

A 16-bit, fixed-point DSP core from DSP Group, Inc.

Object code

Binary instructions and data for use on a programmable processor. Object code is usually produced by an assembler and is often "relocatable," meaning that it does not contain absolute references to particular memory locations.

Off-core

A resource (such as memory or a peripheral) that is not contained within the actual processor core.

OnCE

(On-chip emulation port.) A serial debugging port found on Motorola DSP processors.

On-core

A resource (such as memory or a peripheral) that is contained within the actual processor core.

One-time programmable

Used to refer to a read-only memory (ROM) that can be programmed only once.

Operand-related/operand-unrelated parallel move

See parallel move.

Orthogonal instruction set

An instruction set that is regular and consistent with respect to combinations of operations, data types, and addressing modes.

OTP

See one-time programmable.

Overflow

A situation that occurs when the result of a mathematical operation (typically an add or subtract) requires more bits than are available in the register to which it is to be stored. See also: saturation, wrap-around.

Paged DRAM

A DRAM chip that allows faster than normal access when a group of memory accesses occur within the same region (or page) of memory.

PalmDSPCore

A family of 16-, 20-, and 24-bit cores from DSP Group, Inc.

Parallel move

A movement of data carried out in parallel with the execution of another operation. DSP processors typically provide the ability to move two data values in parallel while executing another operation, although the number of instructions that support parallel moves may be limited.

PC-relative addressing

An addressing mode where the effective address is computed by adding an offset to the current location pointed to by the program counter.

Peripheral

An on-chip unit or external device that performs specialized tasks such as serial or parallel communication, DMA, A/D or D/A conversion, etc.

PGA

(Pin grid array.) A type of integrated circuit package. The external connections are made available on metal pins arranged in a grid.

Phase-locked loop

A feedback system in which an oscillator tracks a periodic input signal. There are many uses for phase-locked loops, including timing recovery in modems and generation of an on-chip master clock at a higher frequency from a lower-frequency signal.

PineDSPCore

A 16-point, fixed-point DSP core from DSP Group, Inc.

Pipeline

An organization of computational hardware in which different stages of the execution of an instruction proceed in parallel for different instructions.

PLL

(Phase-locked loop.) See phase-locked loop.

PQFP

(Plastic quad flat pack.) A type of integrated circuit package.

Pre-/post-increment/decrement

Incrementing or decrementing the contents of a register before or after the register is used for indirect addressing. Pre-increment/decrement means that the contents of the register are modified before the memory access; post-increment/decrement means that the contents of the register are modified after the memory access.

Prioritized interrupts

A scheme used to determine which of several simultaneous interrupts is serviced. Interrupts with higher priority are serviced first.

Profiling

The process of determining the amount of time a processor spends in different sections of a program. The results of profiling are useful during the process of optimizing software for execution speed.

Programmable DSP

An integrated circuit implementing a programmable processor intended for signal processing.

Programmed wait state

A wait state that is automatically generated by the processor when accessing certain ranges of external memory. Most processors that support programmed wait states allow the number of wait states to be configured by the programmer.

PROM

(Programmable read-only memory.) PROM memory can be programmed once by the user after the chip has been fabricated. This is sometimes called one-time-programmable memory. See also: EPROM, ROM.

PWM

(Pulse width modulation.) PWM is used in some control applications. It is also sometimes used as an inexpensive way to implement a D/A converter.

QFP

(Quad flat pack.) A type of integrated circuit package. ICs packaged in QFP packages are typically less expensive than the same ICs in PGA packages.

Quantization error

The difference between the original value and the truncated value.

Quantization noise

Another word for quantization error. The term is used because quantization errors add noise to signals.

Quick interrupt

See fast interrupt.

Real-time breakpoint

A debugging feature provided by in-circuit emulators. The processor executes at full speed and execution is halted when a specified condition (called the breakpoint condition) evaluates true.

Real-time operating system (RTOS)

An operating system that allows the developer to place an upper bound on the amount of time a process must wait to execute after a critical event occurs. Examples of real-time operating systems for DSPs include DSP-BIOS and OSE. UNIX is an example of a non-real-time operating system, in that programs may wait an indefinite amount of time before executing.

Refresh

Periodic access required by dynamic memory (DRAM) to avoid loss of data.

Register

A circuit that holds a set of contiguous bits that are treated as a group. An accumulator is an example of a register.

Register-direct addressing

An addressing mode where operands come from registers, and the registers are identified by constants in the instruction. For example, the instruction "ADD X0,Y0,A", which adds the contents of the X0 and Y0 registers and places their sum in the A register, uses register-direct addressing.

Register file

Another word for the set of registers that supply operands for the data path.

Register-indirect addressing

An addressing mode in which the operand address is contained in a register, and the register is identified in the instruction. For example, the instruction "MOVE (R0),A", which moves the contents of the memory location whose address is stored in register R0 to register A, uses register-indirect addressing. In DSPs, the contents of the register are often modified before or after the address is used.

Relocatable code

Object code that does not contain absolute memory addresses, but instead has symbolic references that a loader can resolve when it loads the program. This allows the program to be loaded into memory at any starting address. See also: object code.

Relocate

The copying of a segment of code from one memory location to another. This requires the code to be relocatable. See also: relocatable code.

Repeat buffer

A (small) buffer where instructions can be loaded to allow fast execution of the instructions a specified number of times. The repeat buffer frees the processor from fetching the instructions from memory.

Reverse-carry arithmetic

An alternative to bit-reversed addressing, where an increment of an address is done by adding a high-order bit and propagating the carry signal in the reverse direction, towards the low-order bit.

RISC

(Reduced instruction set computer.) A computer architecture with simple instructions that can be executed very quickly.

ROM

(Read-only memory.) Mask-programmed ROM, meaning ROM whose memory contents are fixed when the chip is fabricated. See also: PROM, EPROM.

Round-to-nearest

A rounding technique wherein numbers are rounded to the nearest representable number. A value that is equidistant between two representable numbers is always rounded up (or down, depending on the implementation). This introduces bias into calculations.

RTOS

See real-time operating system.

Sample

The value of a signal at a specified time. The outputs of an A/D converter are called samples.

Saturation

A strategy for handling overflow in which the largest representable magnitude is used. Contrast with: wrap-around.

SBSRAM

(Synchronous burst static random-access memory.) Like SRAM, SBSRAM does not require periodic refreshing. SBSRAM is synchronized with a clock signal, allowing faster timing by clocking in a new address while reading the data for the previous address. In addition, SBSRAM includes "burst" transfers where one access to memory can be followed by fast subsequent reads or writes to consecutive memory addresses. See also: SRAM.

Scan-based debugging/emulation

A debugging approach that uses dedicated hardware on a processor to debug the processor while it is operational and in-circuit within its target system.

SC140

A 16-bit VLIW processor core from StarCore, Inc. The core is used by Lucent Technologies and Motorola in their own chip-level products.

Shadow register

An alternative set of registers in addition to the processor's primary register file. Processors that provide shadow registers can switch between the primary registers and the shadow registers. This can be used to efficiently switch between two different contexts.

Signed magnitude

A fixed-point number representation where one bit indicates the sign of the number, and the remaining bits represent the magnitude. The disadvantage of signed magnitude representation is that the value zero has multiple representations.

Sleep mode

A power-conservation mode in which much of the hardware on the DSP is turned off.

SoC

(System-on-Chip.) An integrated circuit intended for use in a particular product or application, typically containing one or more processor cores and a variety of application- or product-specific features, such as algorithm accelerators and specialized peripherals. Also referred to as an ASIC (application-specific IC).

SPDIF

An IEC-958 standard for transferring digitally encoded audio signals.

SPI

(Serial peripheral interface.) A synchronous serial protocol used to connect integrated circuits.

SRAM

(Static random-access memory.) SRAM is often used in DSP systems because it is very fast and does not require periodic refreshing, as DRAM does. However, SRAM is more expensive than DRAM and is not available in as high densities.

Standard cell

A physical layout element in an IC foundry's library, that conforms to certain standard characteristics, such as physical dimensions and electrical drive capability. A standard-cell integrated circuit is constructed by tiling together standard cells and interconnecting them.

Static column DRAM

Another word for paged DRAM.

Static logic

A circuit design technique used to design processors. A processor designed with static logic will run with an arbitrarily low frequency input clock and still function correctly, although more slowly. Because power consumption is proportional to clock frequency in CMOS circuitry, a static processor allows one to reduce power consumption by slowing or stopping the input clock. See also: dynamic logic.

Subroutine

A unit of software that can be invoked from multiple locations in one or more other units of software to perform a specific operation or set of operations. Subroutines allow a programmer to avoid the need for repeatedly specifying often-used sequences of instructions in a program.

Subroutine call

The action by which a processor transfers execution to a subroutine. At a minimum, this usually involves storing the current program counter value (so that execution can be resumed at the correct location when the subroutine completes) and executing a branch instruction.

Subtract-with-borrow

A subtraction where the value of the carry bit from the previous operation is subtracted from the result. See also: add-with-carry.

Superscalar

An architecture in which multiple instructions (usually between two and four) are issued in every instruction cycle and executed in parallel. The determination of which instructions will be executed in parallel ("scheduling") is performed by specialized hardware in the processor, and takes place at run-time.

T1

A telecom standard for high-speed serial communication.

T.4

An ITU-T standard for lossless compression and decompression of two-tone images based on Huffman coding. It is used in Group 3 facsimile machines.

Tap

(1) A fundamental section of an FIR filter, consisting of a coefficient, a multiplication, and a delay line stage. (2) Output of a delay line stage.

Target system

The end system or product in which a processor will be used.

TDMA

(Time-division multiple access.) A multiple access method in which a communication channel is divided into multiple time slots, and only one user transmits in a given time slot.

Throughput

Frequency with which an execution unit, processor, or interface is capable of producing or transferring results. See also: latency.

Time-stationary

An instruction set design for pipelined processors in which an instruction specifies the operations performed by the various pipeline stages in one instruction cycle. The Lucent Technologies DSP16xx and all Motorola DSPs are good examples of this style. Contrast with: data-stationary.

TMS320C1x

A family of 16-bit, fixed-point DSPs from Texas Instruments, Inc.

TMS320C2x

A family of 16-bit, fixed-point DSPs from Texas Instruments, Inc.

TMS320C2xxx

A family of 16-bit, fixed-point DSPs from Texas Instruments, Inc.

TMS320C27xx

A family of 16-bit, fixed-point DSP cores with microcontroller features from Texas Instruments, Inc.

TMS320C3x

A family of 32-bit, floating-point DSPs from Texas Instruments, Inc.

TMS320C4x

A family of 32-bit, floating-point DSPs from Texas Instruments, Inc.

TMS320C5x

A family of 16-bit, fixed-point DSPs from Texas Instruments, Inc. The TMS320C5x is the successor to the TMS320C2x family.

TMS320C54xx

A family of 16-bit, fixed-point DSPs from Texas Instruments, Inc.

TMS320C55xx

A family of 16-bit, fixed-point DSPs from Texas Instruments, Inc.

TMS320C62xx

A family of 16-bit, fixed-point DSPs from Texas Instruments, Inc.

TMS320C64xx

A family of 16-bit, fixed-point DSPs from Texas Instruments, Inc.

TMS320C67xx

A family of 32-bit, floating-point DSPs from Texas Instruments, Inc.

TMS320C8x

A DSP processor from Texas Instruments, Inc. The TMS320C8x contains three or five processors on a single chip: a RISC-based "master processor" and two or four 16-bit, fixed-point DSP processors. The TMS320C8x is intended for image and video processing applications.

TQFP

(Thin quad flat pack.) A type of integrated circuit package similar to, but thinner than, a plastic quad flat pack. TQFP packages are typically used in small, portable electronic systems, such as cellular telephones and pagers. See also: PQFP.

Transducer

A device that converts a physical representation of a signal (e.g., pressure or light) to an electronic representation or vice versa.

Truncation
Discarding the least-significant bits of a value.

Two's complement
The binary representation of numbers most commonly used in DSPs for fixed-point numbers.

Underflow
Underflow occurs when a value is too small to be represented in a certain format. This may happen when two small numbers are multiplied, for example.

Unit delay
A delay of one sample period.

USFS 1015
(United States Federal Standard 1015.) The standard specifying the LPC-10E speech coder. See LPC.

USFS 1016
(United States Federal Standard 1016.) The standard specifying the CELP speech coder. See CELP.

V.22bis
An ITU-T standard for 2,400-bit/second modems.

V.27
An ITU-T standard for 4,800- and 2,400-bit/second facsimile modems.

V.29
An ITU-T standard for 9,600- and 7,200-bit/second facsimile modems.

V.32
An ITU-T standard for 9,600-bit/second modems.

V.32bis
An ITU-T standard for 14,400-bit/second modems.

V.32terbo
A protocol for 19,200-bit/second modems promulgated by Lucent Technologies before the V.34 standard was available.

V.34

An ITU-T standard for 28,800-bit/second modems.

V.42

An ITU-T standard for error correction.

V.42bis

An ITU-T standard for data compression.

Verilog HDL

A hardware description language originally developed by Gateway Design Automation (now part of Cadence Design Systems, Inc.) for use with their digital hardware simulator. Verilog HDL is now a public standard, maintained by Open Verilog International. The language supports modeling of hardware at levels of abstraction ranging from gate level up to very abstract behavioral or performance models. Numerous companies market design entry, simulation, and synthesis tools that process Verilog HDL. See also: VHDL.

VHDL

(VHSIC hardware description language.) A hardware description language specified by IEEE-1076. The language supports modeling of hardware at levels of abstraction ranging from gate level to very abstract behavioral or performance models. Numerous companies market design entry, simulation, and synthesis tools that process VHDL. See also: Verilog HDL.

VHSIC

(Very high-speed integrated circuit.) A U.S. government program aimed at improving integrated circuit technology. The program is now defunct. The VHDL language resulted from this program.

Viterbi decoding (or Viterbi algorithm)

A computationally efficient (but still relatively complex) mechanism for decoding a convolutionally encoded bit stream.

VLIW

(Very large instruction word.) An instruction word encoding technique used on some processors with execution units that can operate in parallel. A VLIW instruction is essentially a compound instruction word created by concatenating a number of smaller instructions that specify the operations to be performed by each of the execution units. By executing a VLIW instruction, all the encoded, smaller instructions are executed in parallel. The determination of which instructions will be executed in parallel ("scheduling") is performed by the programmer or code-generation tool, and hence takes place at compile-time.

VLSI

(Very large-scale integration.)

Von Neumann architecture

A processor with one memory address space used for both instructions and data.

VSELP

(Vector sum excited linear prediction.) A speech coding technique used in the U.S. IS-54 digital cellular telephone system.

Wafer

A disc of silicon on which chip dies are produced.

Wait state

A delay inserted during external memory accesses to give a slow peripheral or memory time to decode the address and retrieve data.

Watt

A unit of power. One watt is the power consumed by a device that uses one joule of energy in one second.

Word-addressable

A memory space that can be addressed only at word-aligned addresses. See also: byte-addressable.

Wrap-around

An overflow strategy in which overflow is ignored, and results are allowed to wrap around the ends of the range of representable numbers. The least significant bits (those that fit the representable range) are preserved.

Z893xx

A family of 16-bit, fixed-point DSPs from Zilog, Inc.

Z894xx

A family of 16-bit, fixed-point DSPs from Zilog, Inc.

ZR38xxx

A family of 20-bit, fixed-point DSPs with 32-bit instruction words from Zoran Corporation.

ZSP164xx

A family of 16-bit, fixed-point DSP processors from ZSP Corporation, notable for its superscalar architecture.

Index