An Implementation of FFT, DCT, and Other Transforms on the TMS320C30

APPLICATION REPORT: SPRA113

Panos Papamichalis
Regional Technology Center
Waltham, Massachusetts
Texas Instruments

Digital Signal Processing Solutions

TEXAS INSTRUMENTS

IMPORTANT NOTICE

TRADEMARKS

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

An Implementation of FFT, DCT, and Other Transforms on the TMS320C30

Abstract

This book describes the several types of transforms and related algorithms used on the TMS320C30 family of digital signal processors. These include:

- The Fast Fourier Transforms (FFTs)
  - the complex radix-2 FFT
  - the complex radix-4 FFT
  - the real valued radix-2
- The Discrete Hartley Transform (DHT)
- The Discrete Cosine Transform (DCT)

The book contains:

- A description of transforms and their implementation on the TMS32030 family of digital signal processors.
- A description and comparison of the different kinds of transforms: the FFTs, the Hartley transform and the Cosine transform
- A description of the features of the TMS320C30 that allow the efficient implementation of these algorithms
- Outlines of specific descriptions of implementations, transforms and TMS320C30 C Compiler facts
- Implementation issues
- Several graphics and tables detailing
  - Forms and flowgraphs of FFTs

- Memory requirements for FFT and Hartley transforms
- Differences in FFT and DCT timing

The end of the book contains 17 appendices with actual TMS320C30 source code for performing transforms.

Product Support

World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. New users must register with TI&ME before they can access the data sheet archive. TI&ME allows users to build custom information pages and receive new product updates automatically via email.

Email

For technical issues or clarification on switching products, please send a detailed email to *dsph@ti.com.* Questions receive prompt attention and are usually answered within one business day.

This report describes the implementation of several Fast Fourier Transforms (FFTs) and related algorithms on the TMS320C30. The TMS320C30 is the first device in the third generation of 32-bit floating-point Digital Signal Processors (DSPs) in the Texas Instruments TMS320 family. The algorithms considered here are the complex radix-2 FFT, the complex radix-4 FFT, the real-valued radix-2 FFT (both forward and inverse transforms), the Discrete Hartley Transform (DHT), and the Discrete Cosine Transform (DCT). These transforms have many applications, such as in image processing, sonar, and radar.

The introduction briefly describes transforms and their implementation on the TMS320 family of processors. Next, the different kinds of FFTs (including the real FFT), the closely-related Hartley transform, and the Cosine transform are described and compared. This is followed by a description of the TMS320C30 features that permit efficient implementations of these algorithms. Then, specific implementations, transforms, and TMS320C30 C Compiler facts are outlined. Finally, the report discusses some implementation issues, and the appendices list actual TMS320C30 code for performing transforms.

The powerful architecture and instruction set of the TMS320C30 permit flexible and compact coding of the algorithms in assembly language while preserving close correspondence to a high-level language implementation. The efficiency of the architecture and the speed of the device make faster realization of real and complex transforms possible. With the availability of a C compiler, these routines can be put in C-callable form and used as faster versions of FFT C functions.

Introduction

The Fast Fourier Transform (FFT) is an important tool used in Digital Signal Processing (DSP) applications. Its development by Cooley and Tuckey gave impetus to the establishment of DSP as an independent discipline. The well-structured form of the FFT has also made it one of the benchmarks in assessing the performance of number-crunching devices and systems.

In recent years, because of the popularity of this signal-processing tool, there have been efforts to improve its performance by advances both at the algorithmic level and in hardware implementation. Researchers have been developing efficient algorithms to increase the execution speed of FFTs while keeping requirements for memory size low. On the other hand, developers of VLSI systems are including features in their designs that improve system performance for applications requiring FFTs. In particular, single-chip programmable DSP devices, currently available or under development, can realize FFTs with speeds that allow the implementation of very complex systems in realtime.

The Texas Instruments TMS320 family consists of five generations of programmable digital signal processors. The TMS32010 introduced the first generation, which today encompasses more than twelve devices with various speeds, interfacing capabilities, and price/performance combinations. FFT implementations on the TMS32010 can be found in the appendix of the book by Burrus and Parks [1].

The second-generation TMS320 devices (the TMS32020, the TMS320C25, and their spinoffs) enhanced the architecture and speed capabilities of the first generation. Examples of FFT programs implemented on the TMS32020 can be found in an application report in the book *Digital Signal Processing Applications with the TMS320 Family* [2]. Such programs are easily extended to the TMS320C25 because of the code compatibility between devices.

The architectural and speed improvements on the processors from one generation to the next have made the FFT computation faster and the programming easier. These advantages have reached a new high level in the third generation. The TMS320C30 is the first device in the third generation, and this report examines implementation of the FFT algorithms on it. The fourth generation (TMS320C4x) is a new set of floating-point devices, while the fifth generation (TMS320C5x) is a continuation of the fixed-point devices. Since software compatibility is maintained within the fixed-point and the floating-point devices, the existing FFT implementations will also be applicable to these new generations.

The Fourier Transform of an analog signal $x(t)$, given as

$$X(\omega) = \int_{-\infty}^{\infty} x(t)\, e^{-j\omega t} dt \tag{1}$$

determines the frequency content of the signal $x(t)$. In other words, for every frequency, the Fourier transform $X(\omega)$ determines the contribution of a sinusoid of that frequency in the composition of the signal $x(t)$. For computations on a digital computer, the signal $x(t)$ is sampled at discrete-time instants. If the input signal is digitized, a sequence of numbers $x(n)$ is available instead of the continuous-time signal $x(t)$. Then, the Fourier transform takes the form

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)\, e^{-j\omega n} \tag{2}$$

The resulting transform $X(e^{j\omega})$ is a periodic function of ω, and it needs to be computed for only one period. The actual computation of the Fourier transform of a stream of data presents difficulties because $X(e^{j\omega})$ is a continuous function in ω. Since the transform must be computed at discrete points, the properties of the Fourier transform led to the definition of the *Discrete Fourier Transform* (DFT), given by

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-\frac{j2\pi kn}{N}} \tag{3}$$

When $x(n)$ consists of N points $x(0)$, $x(1)$, . . ., $x(N-1)$, the frequency-domain representation is given by the set of N points $X(k)$, $k=0,1,\ldots,N-1$. Equation (3) is often written in the form

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{nk} \tag{4}$$

where $W_N^{nk} = e^{-j\,2\pi nk/N}$. The factor W_N is sometimes referred to as the *twiddle factor*. A detailed description of the DFT can be found in references [1,3,4]. The computational requirements of the DFT increase rapidly with increasing block size N, having an impact on the real-time system performance. This problem was alleviated with the development of special fast algorithms, collectively known as Fast Fourier Transform (FFT). With an FFT, the computational burden increases much less rapidly with N, and for any given N, the FFT computational load, measured in terms of required multiplications and additions, is smaller than a brute-force computation of the DFT.

The definition of the FFT is identical to the DFT: only the method of computation differs. To achieve the efficiency of an FFT, it is important that N be a highly composite number. Typically, the length N of the FFT is a power of 2: $N = 2^M$, and the whole algorithm breaks down into a repeated application of an elementary transform known as a *butterfly*. If N is not a power of 2, the sequence $x(n)$ is appended with enough zeroes to make the total length a power of 2. Again, references [1,3,4] contain a detailed development of the FFT. Reference [2] also discusses the same topic.

Different Forms of the FFT

Over the years, researchers have developed different forms of FFT for more efficient computation. Special cases, such as those in which the input is a sequence of real numbers, have been investigated, and even more sophisticated algorithms have been developed. The general form of the FFT *butterfly* is given in Figure 1.

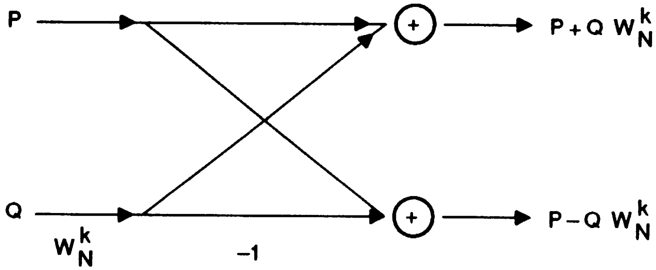

Figure 1. Radix-2 Butterfly for Decimation in Time

If the inputs to the butterfly are the two complex numbers P and Q, the outputs will be the complex numbers P' and Q', such that

$$P' = P + Q\,W_N^k \tag{5}$$

and

$$Q' = P - Q\,W_N^k \tag{6}$$

The quantities P, Q, and P', Q' represent different points in the array being transformed, and they may or may not occupy adjacent locations in that array. In an in-place computation, the result P' will overwrite P, and Q' will overwrite Q. W_N^k represents again the twiddle factor, and its exponent is determined by the location of the corresponding butterfly in the FFT algorithm.

Figure 2 shows an alternate form of the same FFT butterfly.

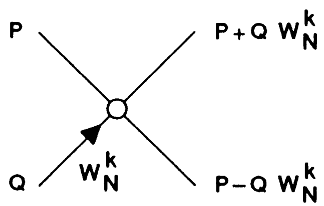

Figure 2. Alternate Form of Radix-2 Butterfly for Decimation in Time.

Although the notation is now less descriptive, it creates a clearer picture when several butterflies are put together to form an FFT. Using the first notation, Figure 3 is the flowgraph of an 8-point FFT example.

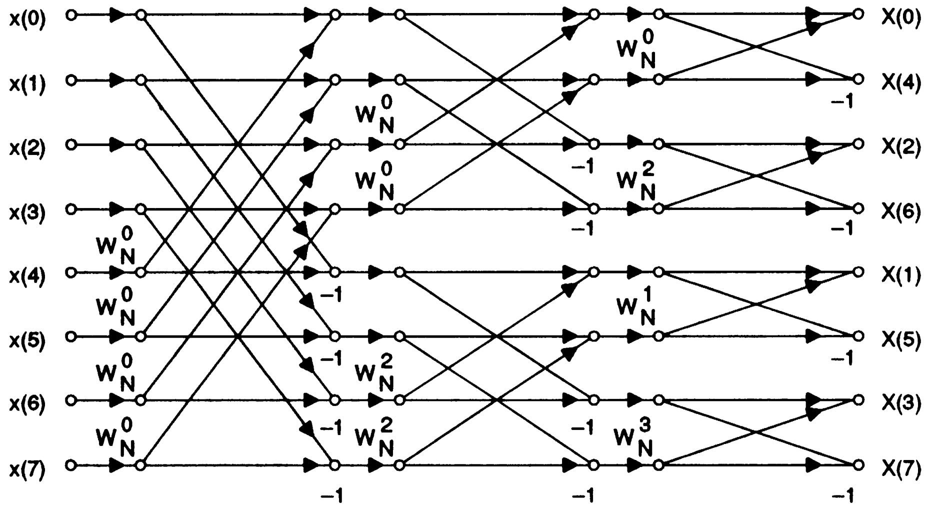

Figure 3. Example of 8-Point FFT with Decimation in Time.

Note that the input sequence *x(n)* is in the correct order, while the output *X(k)* is scrambled. Actually, this scrambling occurs in a very systematic way, called bit-reversed order: If you express the indices of a scrambled sequence in binary and you reverse this number, the result is the order that this particular point occupies. For instance, *X*(3) occupies the sixth position in the output (when counting from the zero position). In binary form, $3_{10} = 011_2$, and if bit-reversed, you get $110_2 = 6_{10}$, which is the position that *X*(3) occupies. It turns out that the third position is occupied by *X*(6), and to restore the correct order at the output, you need only to swap these two numbers.

The same procedure can be repeated with all the scrambled numbers not occupying the position that their index suggests. If the input sequence *x(n)* is rearranged to appear in bit-reversed form, the output *X(k)* appears in the correct order, as shown in Figure 4.

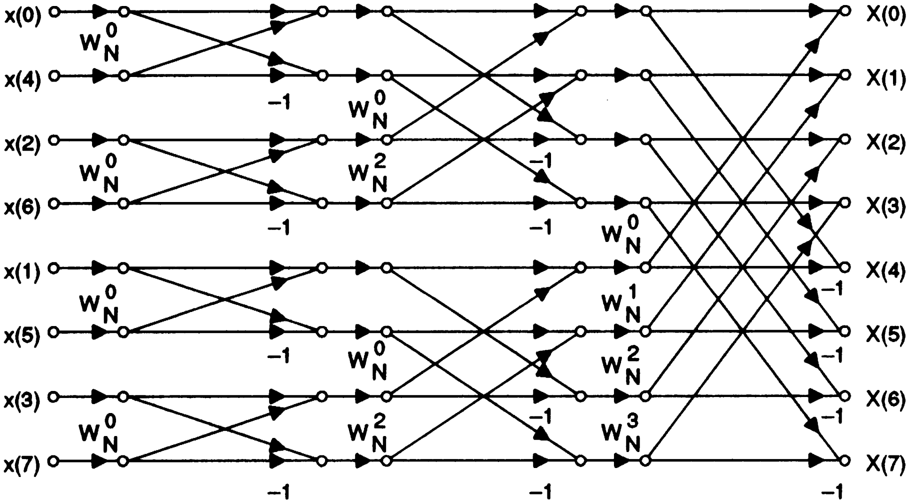

Figure 4. Alternate Form of 8-Point FFT with Decimation in Time. The Input Is in Bit-Reversed Order and the Output Is in the Correct Order.

Since the only difference between Figures 3 and 4 is a rearrangement of the butterflies, the computational load and the final results are identical. In terms of implementation, this rearrangement means that the nesting of the two innermost loops in the FFT routine is interchanged.

The butterflies and the FFT configurations presented thus far implement the FFT with a *decimation in time*. This terminology essentially describes a way of grouping the terms of the DFT definition; see Equation (3). An alternative way of grouping the DFT terms together is called *decimation in frequency*. Figures 5 and 6 show the same example of an 8-point FFT: Figure 5 with the input in correct order and the output in bit-reversed order, and Figure 6 vice-versa, and using the decimation in frequency (DIF).

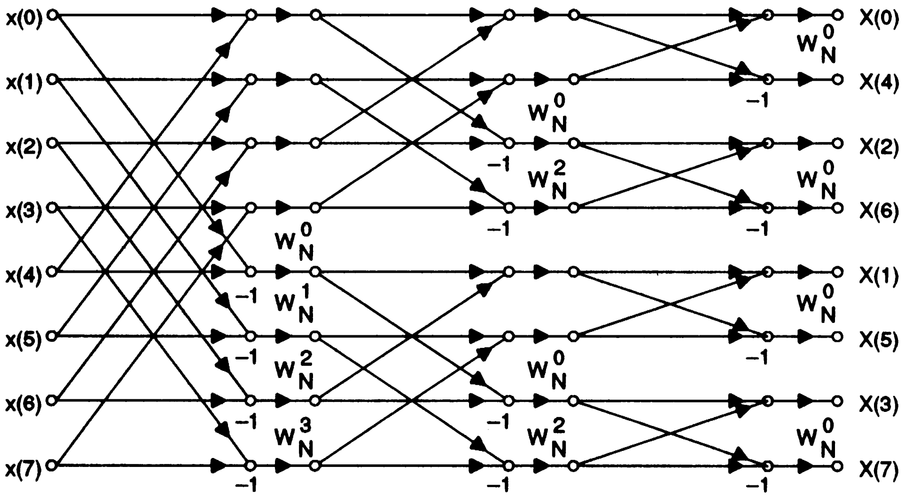

Figure 5. Example of an 8-Point FFT with Decimation in Frequency.

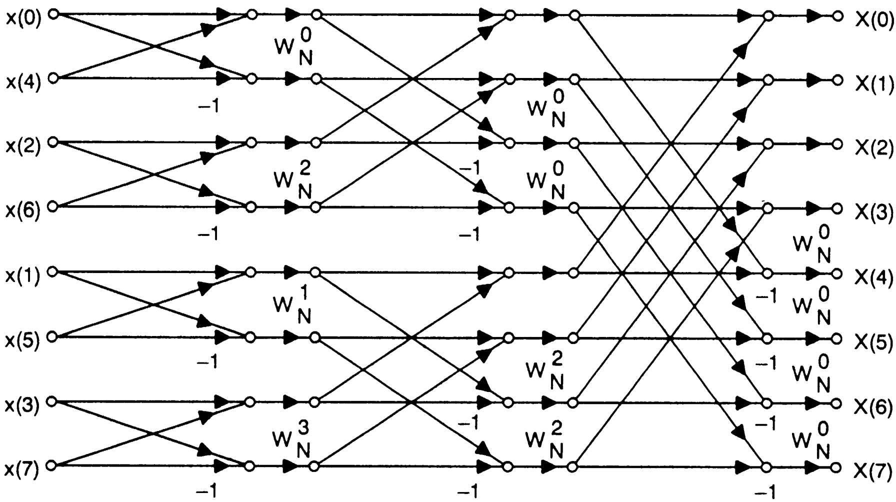

Figure 6. Alternate Form of 8-Point FFT with Decimation in Frequency. The Input Is in Bit-Reversed Order and the Output Is in the Correct Order

Pictorially, the difference between decimation in time and decimation in frequency is that the twiddle factor appears at the input of the butterfly in the first, and at the output in the second. Otherwise, the two methods are identical in terms of results. However, depending on what is the most convenient order of getting the twiddle factors and where the longest-span butterfly appears, you may prefer one method over the other.

The butterfly shown in Figure 1 (or Figure 2) is the smallest element in a radix-2 FFT. The radix of the FFT represents the number of inputs that are combined in a butterfly. The Fast Fourier Transform is usually explained around the radix-2 algorithm for conceptual simplicity. If, however, higher-order radices are used, more computational savings can be achieved. These savings increase with the radix, but there is very little improvement above radix 4. That's why the radix-2 and radix-4 FFTs are the most commonly used algorithms.

In radix-4 FFT, each butterfly has 4 inputs and 4 outputs, essentially combining two stages of a radix-2 algorithm in one. Figure 7 shows this combination graphically.

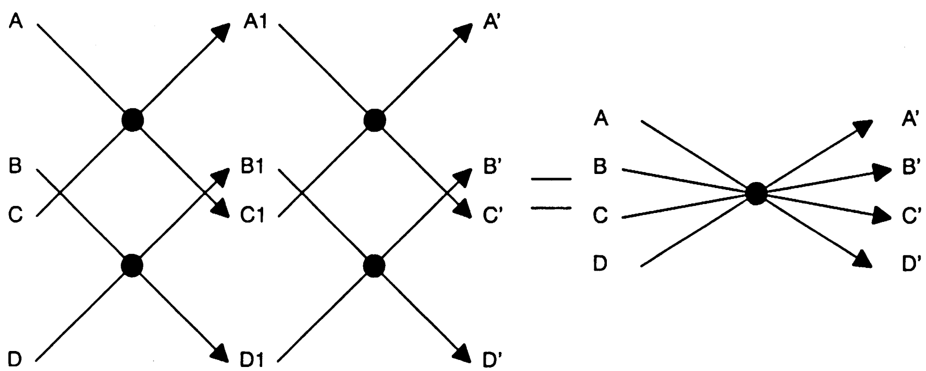

Figure 7. Butterfly for Radix-4, Decimation-in-Time FFT.

Although four radix-2 butterflies are combined into one radix-4 butterfly, the computational load of the latter is less than four times the load of a radix-2 butterfly. Examples of radix-4, 16-point FFTs are shown in Figures 8 and 9 for decimation in time and decimation in frequency, respectively.

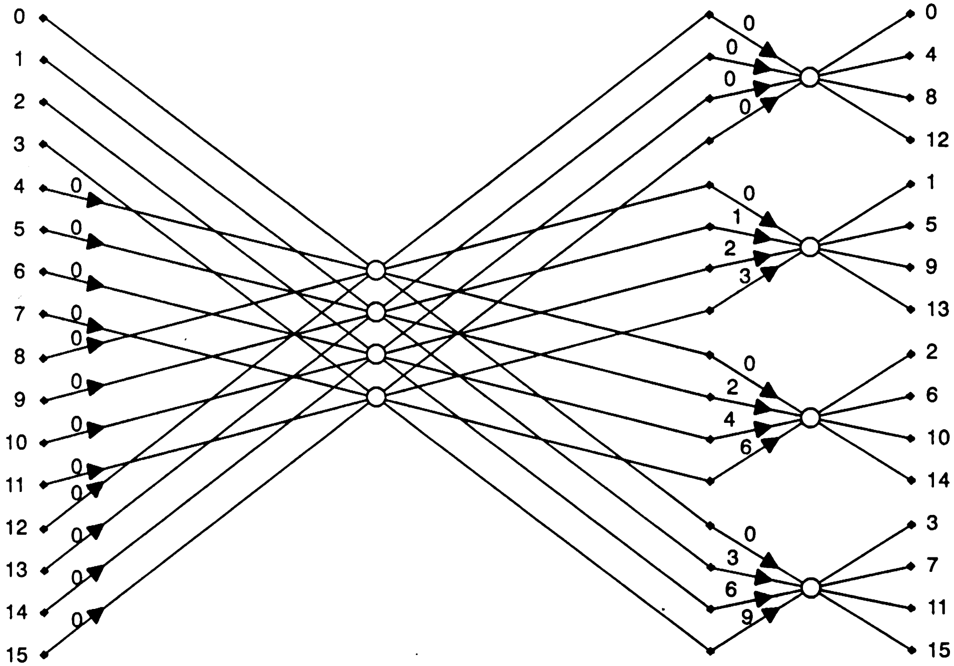

Figure 8. Example of a 16-Point, Radix-4, Decimation-in-Time FFT.

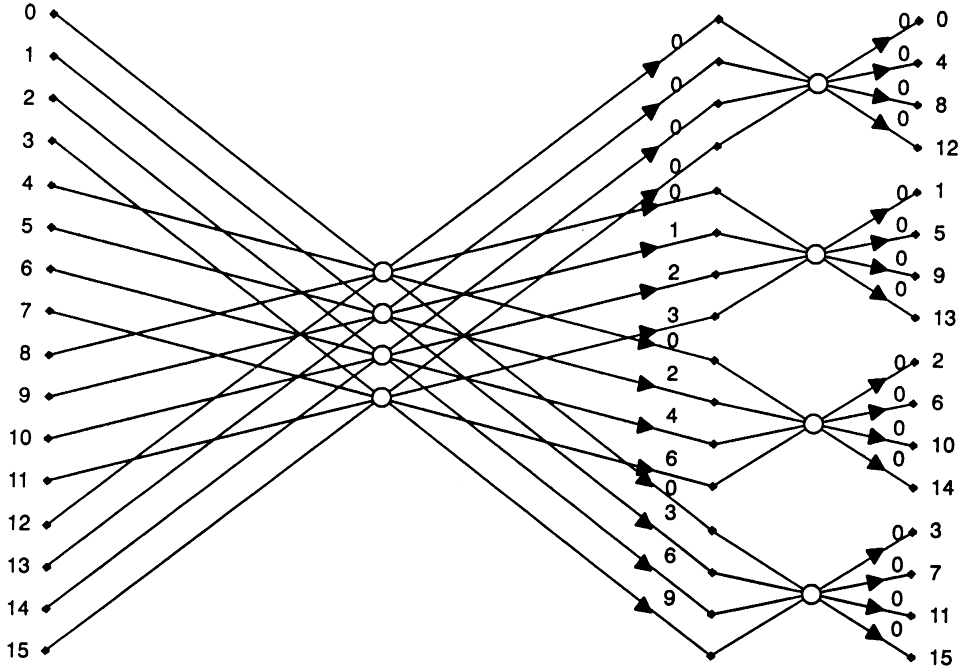

Figure 9. Example of a 16-Point, Radix-4, Decimation-in-Frequency FFT.

These configurations take the incoming sequence in order and produce the frequency-domain result in digit-reversed form. It is a simple matter to rearrange the FFT and have the input in digit-reversed form and the output in order.

Digit reversal is similar to bit reversal, except that the number whose digits are reversed is written in base 4 (equal to the radix) rather than base 2. For example, the output value $X(14)$ in a 16-point, radix-4 FFT occupies position eleven (again starting from zero) because $14_{10} = 32_4$ and, reversing the digits of the number, $23_4 = 11_{10}$. To restore the output to the correct order, the contents of locations with digit-reversed indices should be swapped. However, since the TMS320C30 has a special bit-reversed addressing mode, it is desirable to have the output of the radix-4 computation in bit-reversed rather than digit-reversed form. This is accomplished quite simply if, in each radix-4 butterfly, the two middle output legs are interchanged. That is, whenever the output of the butterfly is the four numbers A', B', C', and D', instead of storing them in that order, store them in the order A', C', B', and D', as shown in Figure 10.

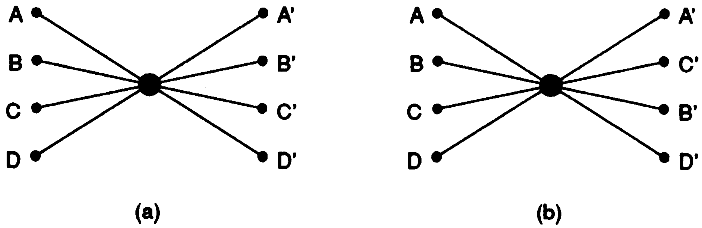

Figure 10. Radix-4 Butterflies. (a) Regularly-Ordered Output, (b) Bit-Reversed Output.

References [5, 6] explain why this simple rearrangement puts the result in bit-reversed order.

Features of the TMS320C30

The TMS320C30 is the first device introduced in the third generation of the TMS320 Digital Signal Processors [7,8]. It has many architectural features that permit very efficient implementation of algorithms. Some of those features pertinent to the FFT implementation are discussed in this section.

The two most salient characteristics of the TMS320C30 device are its high speed (60-ns cycle time) and floating-point arithmetic. The higher speed makes the implementation of real-time application easier than in earlier processors, even when the other architectural advantages are not considered. Each instruction executes in a single cycle under mild pipeline restrictions. The device automatically takes care of any potential conflicts. The pipeline should be observed closely (e.g., using the trace capability of the simulator) only if code optimization for speed is required.

The floating-point capability permits the handling of numbers of high dynamic range without concern for overflows. In FFT programs, in particular, the computed values tend to increase from one stage to the next, as discussed in reference [2]. Then, the fixed-point arithmetic will cause overflows if the incoming numbers are large enough and no provisions are made for scaling. All these considerations are eliminated with the floating-point capability of the TMS320C30. The TMS320C30 performs floating-point arithmetic with the same speed as any fixed point operation; no performance is sacrificed for this feature.

There are eight extended-precision registers, R0—R7, that can be used as accumulators or general-purpose registers, and eight auxiliary registers, AR0—AR7, for addressing and integer arithmetic. For many applications, these registers are sufficient for temporary storage of values, and there is no need to use memory locations. This is the case with the radix-2 FFT algorithm, where no locations are required other than those for the transformation of incoming data to be transformed. Also, arithmetic using these registers greatly increases the programming efficiency. The two index registers, IR0 and IR1, are used for indexing the contents of the auxiliary registers AR0—AR7, thus making the access of the butterfly legs and the twiddle factors easy.

A powerful structure in the TMS320C30 is the block-repeat capability that has the form

```
        RPTB    LABEL
        put instructions here
LABEL   last instruction
```

Whatever occurs after the RPTB instruction and up to the LABEL is repeated one time more than the number included in the repeat counter register, RC. The RC register must be initialized before entering the block-repeat construct. The net effect is that the repeated code behaves as if it were straight-line coded (no penalty for looping), with program size equal to the one in looped code. In this way, the FFT butterfly, being the core of the program, can be implemented in a block-repeat form, thereby saving execution time while preserving the clarity of the program and conserving program space.

A bit-reversed addressing mode is available to eliminate the need for swapping memory locations at the beginning or the end of the FFT (depending on the FFT type). When you use this addressing mode, you access a sequence of data points in bit-reversed order rather than sequentially, and you can recover the points in the correct order during retrieval of the data instead of spending extra cycles to accomplish it in software.

Implementation of Radix-2 and Radix-4 Complex FFTs

Because of the powerful architecture and the instruction set of the TMS320C30, the assembly language program follows closely the flow of a high-level language program; this makes it easy to read and debug. It also keeps the size of the program small and reduces the requirements for program memory. Appendix A presents an example of code for a Radix-2 complex FFT, while Appendix B is a radix-4 complex FFT. The program memory requirements for these programs (as well as others to be discussed later) are given in Table 1.

Table 1. Program Memory Requirements for the Core of the FFT and Hartley Transforms

Routine Type	Program Size
Radix-2, complex FFT	50 words
Radix-4, complex FFT	170 words
Radix-2, real FFT	68 words
Radix-2, real inverse FFT	76 words
Hartley transform	71 words

The numbers in the table correspond only to the core program and do not include the sine/cosine tables for the twiddle factors, any input/output, or any bit-reversing operations. Note also that they are independent of the FFT data size.

The data memory requirements are, of course, dependent on the FFT size. The maximum length of a complex, radix-2 FFT that can be implemented entirely on the internal memory of the TMS320C30 is 1024 points. In the present implementation, the 1024-point radix-4 FFT requires a few more locations (about 7) than are available on-chip.

The code (provided in the appendices) has been written to be independent of the FFT length. The length N, together with the sine/cosine tables for the twiddle factors, should be provided separately to maintain the generic nature of the core FFT program. An example of a file with the sine/cosine tables for a 64-point FFT is given in the Appendix F. Note that the FFT size and the number of stages are declared .global in both files (i.e., the main routine and the file with the table) so that the core program gets the actual values during linking.

To reduce the storage requirements of a sine/cosine table, a full sine and a cosine cycle are overlapped. The table stores 5/4 of a full sine wave, with the cosine table starting with a phase delay of 1/4 cycle from the sine table. This table size is larger than actually needed, and it is selected merely for testing convenience of the algorithms. The minimum table size for a radix-2 complex FFT includes 1/2 of a full sine wave, and 1/2 of a full cosine wave. If these two half waves are combined using the above quarter-cycle phase delay, the minimum table size for this kind of FFT is 3/4 of a full sine wave. For instance, for a 1024-point FFT, the table can be the first 768 points of a sine wave, where a full cycle would be 1024 points. In the case of a radix-4 complex FFT, the minimum table size should include 3/4 of a sine and 3/4 of a cosine wave. Overlapping these requirements, we get the minimum table size of a radix-4 algorithm to be one full sine wave.

An example of a linking file is also included in Appendix F to show how the different segments are assigned. For a complete description of the assembler and linker, consult the corresponding manual [6].

The timing of the FFT routines was done using the cycle-counting capability of the TMS320C30 simulator. For the conversion of the number of cycles into seconds, a cycle time of 50 ns was used. The timing refers only to the core FFT computation, ignoring read-in and write-out requirements, since such requirements are application-dependent. Also, no bit reversal is counted (although it may be included in the program), since it is performed as part of the read-in or read-out. Table 2 gives the timing for the different FFT routines and for the Hartley transform.

Table 2. FFT Timing in Milliseconds†

Transform Size	Radix-2 Complex FFT	Radix-4 Complex FFT	Radix-2 Real FFT	Radix-2 Real Inverse FFT	Hartley Transform
64	0.101	0.103	0.047	0.053	0.068
128	0.211	—	0.099	0.110	0.151
256	0.453	0.520	0.215	0.241	0.336
512	0.991	—	0.476	0.535	0.943
1024	2.175	2.533	1.055	1.193	2.025
1024	1.972				

†Improvements have been made and are shown in this table. You may obtain the latest code from the BBS, (713) 274-2323.

The last entry in this table represents the timing of the radix-2, DIT routine generated at the University of Erlangen [18] and given in Appendix A. These numbers are typically used for benchmarking.

Implementation of Real FFT

The development of FFT algorithms is centered mostly around the assumption that the input sequence consists of complex numbers (as does the output). This assumption guarantees the generality of the algorithm. However, in a large number of actual applications, the input is a sequence of real numbers. If this condition is taken into consideration, additional computational savings can be achieved because the FFT of a real sequence demonstrates the following symmetries: Assuming that the FFT output $X(k)$ is complex,

$$X(k) = R(k) + j\,I(k) \tag{7}$$

and that the sequence has length N, R(k) and I(k) should satisfy the following relations:

$$R(k) = R(N-k), \; k = 1, \ldots, N/2-1 \tag{8}$$
$$I(k) = -I(N-k), \; k = 1, \ldots, N/2-1 \tag{9}$$
$$I(0) = I(N/2) = 0. \tag{10}$$

In other words, the real part of the transform is symmetric around zero frequency, while the imaginary part is antisymmetric. Similar conditions hold if the transform is expressed in terms of magnitude and phase.

The savings are due to the fact that not all points need to be computed. Since the not-computed points do not need to be saved either, there are also storage savings. An efficient algorithm for real-valued FFTs is described in [10]. This algorithm was implemented in the present study in such a way that, given the sequence of N *real* numbers $x(0), x(1), \ldots, x(N-1)$, the resulting FFT, consisting of complex numbers, is stored as $R(0), R(1), \ldots, R(N/2), I(N/2-1), I(N/2-2), \ldots, I(1)$. $R(k)$ and $I(k)$ represent the real and imaginary parts of the complex number $X(k)$. Figure 11 shows the memory arrangement for the FFT. Note that the input to the real FFT should be bit-reversed, but the bit reversal can be done as the data is brought in. With this arrangement, an N-point FFT uses exactly N memory locations. If the full array $X(k)$ is needed, the following relations should be used:

$$X(0) = R(0) \tag{11}$$
$$X(k) = R(k) + j\, I(k),\; K = 1, \ldots, N/2-1 \tag{12}$$
$$X(N/2) = R(N/2) \tag{13}$$
$$X(k) = R(N-k) - j\, I(N-k),\; k = N/2+1, \ldots, N-1 \tag{14}$$

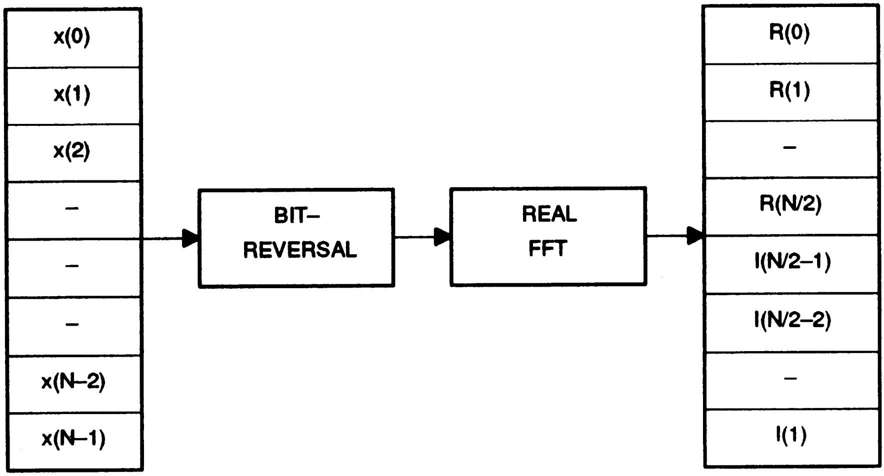

Figure 11. Memory Arrangement of a Real FFT.

It is expected that, in most signal processing applications, there will be no need to reconstruct the full $X(k)$ array and that the output shown in Figure 11 will be sufficient for any further processing.

Appendix C contains TMS320C30 routines implementing a radix-2 real FFT and its inverse. The implementation of the forward transformation is based on the FORTRAN programs contained in [10]. The inverse transformation assumes that the input data are given in the order presented at the output of the forward transformation and produces a time signal in the proper order (i.e., bit-reversing takes place at the end of the program). Viewed another way, the inverse real FFT operates as shown in Figure 11 but with the arrows reversed (and inverse FFT taking the place of the FFT).

The timing for the real-valued FFT (both forward and inverse) is included in Table 2, and the corresponding program sizes are shown in Table 1. As you can see, the real-valued FFT is considerably faster than the corresponding complex FFT because not all the computations need be performed. Furthermore, there are data storage savings because only half the values must be stored. As a result, the maximum length of real-valued FFT that can be implemented on the TMS320C30 without using any external memory is 2048 points. Of course, if all the values are needed, they can be recovered using the symmetry conditions mentioned earlier. To achieve the efficiencies of real FFT and not use any extra memory locations during the computation, the decimation-in-time method is applied [10]. Decimation in time requires the bit-reversal operation in the forward transform to be performed at the beginning of the program rather than at the end. The reverse is true for bit-reversing in the inverse transform.

The Discrete Hartley Transform

Another transform that has attracted attention recently is the Discrete Hartley Transform (DHT)[11, 12]. The DHT is applicable to real-valued signals and is closely related to the real-valued FFT. Comparison of references [10] and [12] describing the implementation of the two algorithms on FORTRAN programs shows that their implementation on the TMS320C30 should be similar. And indeed, this is the case.

The DHT pair is defined for a real-valued sequence $x(n)$, $n = 0, \ldots, N-1$, by the following equations:

$$H(k) = \sum_{n=0}^{N-1} x(n)\, cas(2\pi k\, n \,/\, N), \quad k=0, \ldots, N-1 \qquad (15)$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} H(k)\, cas(2\pi k\, n \,/\, N), \quad k=0, \ldots, N-1 \qquad (16)$$

where $cas(x) = cos(x) + sin(x)$. The DHT demonstrates a symmetry that is convenient for implementations: The same program can be used for both the forward and the inverse transforms, and the result is correct within a scale factor. Also, the real FFT and the DHT can be derived from each other [12].

A radix-2 Hartley transform was implemented on the TMS320C30, and the corresponding code is included in Appendix D. This code follows the structure of the real FFT in Appendix C. Tables 1 and 2 show the program memory requirements and the timing for the execution of Hartley transforms of different sizes. The sine/cosine table sizes are the same as in the case of a real FFT.

The Discrete Cosine Transform

The Discrete Cosine Transform (DCT), since its introduction in 1974 [13], has gained popularity in speech and image processing applications because of its near-optimal behavior. This discussion is based on the paper by Lee [14]. The DCT code was developed and implemented by Paul Wilhelm of the University of Washington.

If $x(n)$, $n=0, \ldots, N-1$ is a time-domain signal and $X(k)$ is the corresponding DCT, $x(n)$ and $X(k)$ are related by the following equations:

$$x(k) = \frac{2}{N} \sum_{n=0}^{N-1} e(k)\, x(n) \cos \frac{(2k+1)\pi n}{2N} \tag{17}$$

$$x(n) = \sum_{k=0}^{N-1} e(k)\, X(k) \cos \frac{(2k+1)\pi n}{2N} \tag{18}$$

$$e(0) = 1/\sqrt{2} \tag{19}$$

$$e(k) = 1, \quad \text{for } k \neq 0 \tag{20}$$

Appendix E shows an implementation of the DCT based on the paper by Lee [14]. The appendix contains the algorithms for both the forward and the inverse transformations and an example of a table for a 16-point DCT. Note that, because of the structure of the algorithm, the cosine table needed contains actually the inverses of the cosines (within a scale factor), and it is not stored in the natural order. Instead, it is generated by the following C pseudocode:

```
for [k=2, i=0; k=N/2; k*=2]
    for [j=k/2; j<N/2; j+=k]{
        cos_table[i++] = 1/[2*cos[j*pi/[2*N]]];
        cos_table[i++] = 1/[2*cos[[N-j]*pi/[2*N]]];
    }
cos_table[N-2] = cos[pi/4];
cos_table[N-1] = 2/N;
```

The last entry to the table is not part of the cosine itself; it is a constant that is used by the algorithm, and it is placed at the end of the cosine table for convenience.

Table 3 shows the timing of the forward and inverse transforms for different transform lengths. The difference in the timing between the forward and the inverse transforms is due to the fact that more time was expended to optimize the performance of the inverse transform. Since four of the smallest butterflies were done simultaneously in the center program loop, the minimum permissible array size to be transformed is 8.

Table 3. DCT Timing in Milliseconds

Transform Size	Forward Transform	Inverse Transform
16	0.019	0.017
64	0.875	0.073
128	0.192	0.161
256	0.418	0.347
512	0.912	0.754
1024	1.982	1.652

Other Related Transforms

In addition to the FFT types mentioned earlier (complex, real, decimation-in-time, decimation-in-frequency, etc.), newer forms of the FFT have been developed to reduce the computational load. One of the latest in the literature is the *Split-Radix* FFT. The Split-Radix FFT [16] has the lowest number of multiplies and adds of any known algorithm. It achieves this efficiency by combining certain radix-2 and radix-4 butterflies, but, as a result, the classical concept of FFT stages is lost. The new structure uses a rather complicated indexing scheme, which is the price paid for the reduced multiplies/adds. Since, on the TMS320C30, multiplies/adds are not more expensive computationally than any other operation, the indexing scheme wipes out the gains of the reduced arithmetic. Actually, an implementation of the split-radix FFT showed it to be slower than the radix-2 FFT, one of the main reasons being that the block-repeat structure could no longer be used effectively.

Very often, there is a question on what the different benchmark numbers mean. A useful comparison of execution times for different algorithms on different machines has been made [17]. Table 4 presents a small segment of the resulting information that is relevant to the present discussion: the timing in seconds for the radix-8, mix-radix, and split-radix algorithms that were implemented on various machines. Different operating systems and compilers have been used, as shown. The execution times of Table 4 should be compared with the 0.0010055 s that it takes to implement a 1024-point, radix-2, real FFT on a TMS320C30. As can be seen, the TMS320C30 compares favorably to all the other machines investigated.

Table 4. Execution Times in Seconds for a 1024-Point Real FFT. The Numbers Should Be Compared with 0.001055 s of a 1024-Point Real FFT on the TMS320C30

Machine	Radix-8	Mix-radix	Split-radix
VAX 750 UNIX BSD4.2 f77	0.3634	0.3902	0.3021
VAX 750 UNIX BSD4.2 f77 −O	0.2376	0.2948	0.2089
VAX 750 UNIX BSD4.3 f77	0.2545	0.2600	0.2371
VAX 750 UNIX BSD4.3 f77 −O	0.1825	0.2127	0.1672
VAX 785 ULTRIX f77	0.1046	0.1107	0.1101
VAX 785 ULTRIX f77 −O	0.0796	0.0943	0.0811
VAX 785 VMS FOR/NOOPTM	0.0767	0.0871	0.0975
VAX 785 VMS FOR/OPTM	0.0539	0.0641	0.0633
VAX 8600 VMS FOR/OPTM	0.0217	0.0243	0.0235
MICROVAX VMS FOR/NOOPTM	0.1671	0.1846	0.1864
MICROVAX VMS FOR/OPTM	0.1299	0.1527	0.1419
DEC-10 TOPS-10 FOR/NOOPTM	0.0940	0.1184	0.0991
DEC-10 TOPS-10 FOR/OPTM	0.0885	0.1110	0.0845
CDC 855 FTN5,OPT = 0	0.0277	0.0319	0.0338
CDC 855 FTN5,OPT = 1	0.0277	0.0316	0.0337
CDC 855 FTN5,OPT = 2	0.0182	0.0171	0.0151
CDC 855 FTN5,OPT = 3	0.0180	0.0173	0.0150
SUN 3/50 UNIX BSD4.2 f77 −O -f68881	0.2518	0.3365	0.2103
SUN 3/50 UNIX BSD4.2 f77 −f68881	0.2806	0.3897	0.2802
SUN 3/50 UNIX BSD4.2 f77 −O	0.7586	1.047	0.6955
SUN 3/50 UNIX BSD4.2 f77	0.7476	1.029	0.7033
SUN 3/160 UNIX BSD4.2 f77	0.6037	0.6895	0.5660
SUN 3/160 UNIX BSD4.2 f77 −pfa	0.0983	0.1060	0.0946
SUN 3/260 UNIX BSD4.3 f77	0.3689	0.4126	0.3390
SUN 3/260 UNIX BSD4.3 f77 −O	0.3530	0.4142	0.3297
Pyramid 90X UNIX BSD4.2 f77 −O	0.2053	0.2244	0.1416
Pyramid 90X UNIX BSD4.2 f77	0.2206	0.2457	0.1326
HP-1000 21MX-E FTN7X	0.9400	1.248	0.9478
Apple MAC Microsoft FOR	2.6670	3.1600	2.8260
AST PC Microsoft FOR	1.5040	2.0800	1.4630

The TMS320C30 C Compiler

The C compiler for the TMS320C30 permits easy porting of high-level language programs to the DSP device. If the CPU loading of a particular application is not very high, the C compiler can create programs that run on the TMS320C30 in real time. If, however, the result is non-realtime, it may be necessary to use assembly language for more efficient coding.

In most cases, only a portion of the code needs to be written in assembly language. Typically, there are a few code segments where the device spends most of the time and which, when optimized in assembly language, yield the necessary performance improvement. By following the conventions outlined in the run-time environment of the C compiler [15], you can write these time-critical routines in assembly language and call them in a C program. This is also true for the FFT routines. In appendices A, B, and C, the radix-2, radix-4, and real FFT routines mentioned earlier are also put in a C-callable form by adding the necessary interface at the beginning and the end of the code. The tables with the sines and cosines are again assumed to be supplied during link time.

Issues in FFT Implementation

There are many ways of actually implementing the FFT code (and the other transformations), taking into consideration the different possibilities of program locations, the data locations, the ways of input and output, etc. Since it is impractical to cover every possible case, this report has concentrated on a configuration in which the use of external memory is minimized. With the source code and additional explanations provided, you should be able to customize the FFT implementation for a particular application.

Use of External Memory

In these implementations, only on-chip memory was used, and that's why the maximum transform size considered was 1024 points long (2048 for a real transform). Often, though, applications call for use of external memory for program or data or both. When external memory is used, the structure of the code does not change at all; it is only the timing that may be affected.

Fast external memory can be selected so that no wait states are necessary. But even when there are no wait states, accessing external memory may impose some limitations. For instance, you can make only one external memory access in a full cycle, but you can make two accesses of internal memory in each cycle. Also, because of mutliplexing of the busses, pipeline conflicts may arise if both program and data are placed on the same external port. Resolution of such conflicts causes extra cycles for the execution. The section on pipelining in the *TMS320C30 User's Guide* explains in detail what kind of potential conflicts may occur.

To minimize or avoid such conflicts, there are some simple steps that the designer can take. The TMS320C30 has three separate memory areas (one on-chip, one accessed by the primary bus, and one accessed by the expansion bus) that can be combined. For instance, the program can be placed on the expansion port and the data on the primary port. Or the data can first be brought into internal memory and then operated upon. Alternatively, the program may be relocated to internal memory. A related approach is to use the cache. All the transforms are implemented as loops that are executed many times. If you activate the on-chip cache after the first access of the code, the instructions execute from the cache instead of the external memory.

If there are additional conflicts, they can typically be resolved by some rearrangement of the code. For instance, consecutively writing to external memory takes two cycles per write. If, however, a write is followed by some internal operation, then the second cycle of the write is transparent, and the actual cost is one cycle.

Bit Reversal

The TMS320C30 has a special form of the indirect addressing mode for the bit-reversing operation that is required at the beginning or the end of an FFT. Through this addressing mode, the scrambled data are accessed in their proper order. This addressing mode works as follows:

Let ARn (n=0..7) be the auxiliary register pointing to the array with scrambled data. The index register IR0 contains a number equal to one-half the size of the FFT. Then, after every access of the data, ARn is incremented by IR0 using the construct

```
*ARn++(IR0)B
```

This causes the contents of ARn to be incremented by the contents of IR0, but if there is a carry in this incrementing, the carry propagates to the right instead of to the left. The result is the generation of the addresses in a bit-reversed order. The bit-reversed addressing mode works correctly if the array with the data is aligned in memory so that the first memory address is a multiple of the FFT size. This can be achieved if the first memory address has zeros for the last M bits, where $M = log_2N$, with N being the FFT size. For example, in the case of a 1024-point FFT, the last 10 bits of the memory address of the first datum should be zeros.

In the implementation of the complex FFT, the output is complex even when the input is real. So, there is a need to consider both the real and the imaginary parts of the data array. The above description of the bit-reversed addressing mode assumed that the real and the imaginary parts are stored as separate arrays in the memory. In this case, each of the arrays (real or imaginary parts) can be accessed as described. However, in most cases (including this report), the real and imaginary points alternate in the same array.

In this arrangement, the following simple modification achieves the same goal: set IR0 equal to *N* instead of *N/2*, and access the *N* points of the transform. At every access, the auxiliary register is pointing to the real part of the FFT. The imaginary part is located in the next higher location, and it can be easily accessed.

With the bit-reversed addressing mode, the unscrambling of the data can take place when the FFT result is accessed for further processing or for I/O. It is possible, though, that certain applications demand the reordering of the data in the same array. Such a rearrangement can be done very simply for a complex FFT with the following code.

```
; DO THE BIT-REVERSING EXPLICITLY

        LDI   @FFTSIZ,RC     ; RC = FFT SIZE
        SUBI  1,RC           ; RC SHOULD BE ONE LESS THAN DESIRED #
        LDI   @FFTSIZ,IR0    ; IR0 = FFT SIZE
        LDI   @INPUT,AR0
        LDI   @INPUT,AR1
*
        RPTB  BITRV
        CMPI  AR1,AR0        ; EXCHANGE LOCATIONS ONLY
        BGE   CONT           ;     IF AR0AR1
        LDF   *AR0,R0        ;
||      LDF   *AR1,R1        ;     EXCHANGE REAL PARTS
        STF   R0,*AR1        ;
||      STF   R1,*AR0        ;
        LDF   *+AR0,R0       ;
||      LDF   *+AR1,R1       ;     EXCHANGE IMAGINARY PARTS
        STF   R0,*+AR1       ;
||      STF   R1,*+AR0       ;
CONT    NOP   *AR0++(2)
BITRV   NOP   *AR1++(IR0)B
```

Note that AR1 is pointing to the bit-reversed version of the address contained in AR0. For real-valued FFT, or for FFTs that store the real and the imaginary parts in separate arrays, the real-FFT routine in Appendix C contains a modified example of the above code.

Use of DMA

If the signal to be transformed arrives as a continuous stream of data, the DMA could be used to collect the new data while the data already collected are processed. In this case, the data source address of the DMA points to the memory location corresponding to a serial port, or to another port associated with an external device. The destination is a memory space designated for storage.

There are two ways to use such buffers. One possibility is to designate one buffer as the temporary storage and the other buffer as the working area. When the storage buffer receives the necessary amount of data, the data is transferred to the working area, and the DMA starts refilling the storage buffer. Alternatively, the two buffers are considered equivalent: when the processor finishes processing and outputting the data from one and the DMA has filled the other, the two buffers switch functions; i.e., the DMA starts filling the first buffer while the CPU is processing the data in the buffer just filled.

Test Vector

For testing purposes, a vector with 64 (quasi-random) data points and the corresponding FFT values is given in Appendix F. In this way, if any of the routines is implemented, the test vectors can be used to verify the correct functionality of the routines. Together with the test vectors, Appendix C gives a sine/cosine table for a 64-point transform, and the linking file for such a transform.

Summary

This report examined implementations of fast transforms on the Texas Instruments TMS320C3x floating-point devices. The transforms considered were several forms of the FFT, the Discrete Hartley Transform, and the Discrete Cosine Transform. Because of the powerful architecture of the device, the implementation was done easily and efficiently. It was shown that a TMS320C30 executes the FFTs several times faster than large computers such as VAX and SUN workstations. With the availability of the C compiler, these routines can be put in C-callable form and be used to compute the corresponding transforms efficiently.

Appendices

Appendices A to F contain the TMS320C30 assembly language programs for the different algorithms considered. The contents of the appendices are as follows:

Appendix A: Radix-2 Complex FFT.
composed of

- A1: Generic Program to Do a Looped-Code Radix-2 FFT Computation on the TMS320C30.
- A2: fft_2 - Radix-2 Complex FFT to Be Called as a C Function.
- A3: Complex, Radix-2 DIT FFT - R2DIT.ASM.
- A4: Complex, Radix-2 DIT FFT - R2DITB.ASM.
- A5: TWID1KBR.ASM - Table with Twiddle Factors for a FFT up to a Length of 1024 Complex Points.

Appendix B: Radix-4 Complex FFT.
composed of

- B1: Generic Program to Do a Looped-Code Radix-4 FFT on the TMS320C30.
- B2: fft_4 - Radix-4 Complex FFT to Be Called as a C Function.

Appendix C: Radix-2 Real FFT.
composed of

- C1: Generic Program to Do a Radix-2 Real FFT Computation on the TMS320C30.
- C2: fft_rl - Radix-2 Real FFT to Be Called as a C Function.
- C3: Generic Program to Do a Radix-2 Real Inverse FFT Computation on the TMS320C30.

Appendix D: Discrete Hartley Transform.
composed of

- D1: Generic Program to Do a Radix-2 Hartley Transform on the TMS320C30.

Appendix E: Discrete Cosine Transform.
composed of

- E1: A Fast Cosine Transform.
- E2: A Fast Cosine Transform (Inverse Transform).
- E3: FCT Cosine Tables File.
- E4: Data File.

Appendix F: Test Vectors, 64-Point Sine Table, Link Command File.
composed of

F1: Example of a 64-Point Vector to Test the FFT Routines.
F2: File to Be Linked with the Source Code for a 64-Point, Radix-4 FFT.
F3: Link Command File.

The first three appendices contain the code for the radix-2, complex radix-4, and real radix-2 FFT transformations. These routines are given in both the regular form and in a C-callable form. Furthermore, the contents of a file with the twiddle factors are given, as well as an example of a link command file for a 64-point FFT. Note that the source code of these routines can be downloaded from the TI DSP bulletin board (BBS) by calling (713) 274-2323. For questions regarding the BBS, call the TI DSP hotline at (713) 274-2320.

Acknowledgements

Mr. Raimund Meyer and Mr. Karl Schwarz (Lehrstuhl fur Nachrichtentechnik, University of Erlangen) provided the fast routines of Appendix A to do 1024-point, radix-2, DIT FFT. Mr. Paul Wilhelm of the University of Washington provided the routines for the Fast Cosine Transform (FCT) together with the related explanations and the test vector in Appendix E. Their contributions are gratefully acknowledged.

References

[1] Burrus, C. S., and Parks, T.W. *DFT/FFT and Convolution Algorithms*, John Wiley and Sons, New York, 1985.

[2] Lin, K. -S., Ed. *Digital Signal Processing Applications with the TMS320 Family*, Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[3] Oppenheim, A. V. and Schafer R.W. *Digital Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[4] Rabiner, L.W., and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[5] Burrus, C.S. "Unscrambling for Fast DSP Algorithms," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-36, No. 7, pp. 1086—1087, July 1988.

[6] Papamichalis, Panos E., and Burrus, C.S. "Conversion of Digit-Reversed to Bit-Reversed Order in FFT Algorithms," *Proceedings of 1989 IEEE International Conference on Acoustics, Speech, and Signal Processing*, May 1989.

[7] *Third-Generation TMS320 User's Guide*, Texas Instruments, Inc., Dallas, Texas, August 1988.

[8] Papamichalis, Panos E., and Simar, Ray Jr. "The TMS320C30 Floating-Point Digital Signal Processor," *IEEE Micro*, Vol. 8, No. 6, pp. 13—29, December1988.

[9] *TMS320C30 Assembly Language Tools User's Guide*, Texas Instruments Inc., Dallas, Texas, July 1987.

[10] Sorensen, H.V., Jones, D.L., Heideman, M.T., and Burrus, C.S. "Real-Valued Fast Fourier Transform Algorithms", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-35, No. 6, pp. 849—863, June 1987.

[11] Bracewell, R.N. "The Fast Hartley Transform," *Proceedings of IEEE*, Vol. 72, No. 8, pp. 1010—1018, August 1984.

[12] Sorensen, H.V., Jones, D.L., Burrus, C.S., and Heideman, M.T. "On Computing the Discrete Hartley Transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-33, No. 4, pp. 1231—1238, October 1985.

[13] Ahmed, N., Natarajan, T., and Rao, K.R. "Discrete Cosine Transform," *IEEE Transactions on Computers*, Vol. C-23, pp. 90—93, January 1974.

[14] B. G. Lee, "FCT - A Fast Cosine Transform," *Proceedings of 1984 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 28A.3.1—28A.3.4, March 1984.

[15] *TMS320C30 C Compiler Reference Guide*, Texas Instruments Inc., Dallas, Texas, December 1988.

[16] Sorensen, H.V., Heideman, M.T., and Burrus, C.S. "On Computing the Split-Radix FFT," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-34, No. 1, pp. 152—156, February 1986.
[17] Sorensen, H.V.and Burrus, C.S. "Computer Dependency of FFT Algorithms", *Proceedings of ASILOMAR*, 1987.
[18] Schuessler, H.W., Meyer, R., and Schwarz, K. "FFT Implementation on DSP Chips—Theory and Practice," *Proposal for the 1990 IEEE International Conference on Acoustics, Speech, and Signal Processing.*

Appendix A. Radix-2 Complex FFT

Appendix A1. Generic Program to Do a Looped-Code Radix-2 FFT Computation on the TMS320C30

```
*    GENERIC PROGRAM TO DO A LOOPED-CODE RADIX-2 FFT COMPUTATION ON THE
*    TMS320C30.
*
*    THE PROGRAM IS TAKEN FROM THE BURRUS & PARKS BOOK, P. 111. THE (COMPLEX)
*    DATA RESIDE IN INTERNAL MEMORY. THE COMPUTATION IS DONE IN-PLACE, BUT THE
*    RESULT IS MOVED TO ANOTHER MEMORY SECTION TO DEMONSTRATE THE BIT-REVERSED
*    ADDRESSING. THE TWIDDLE FACTORS ARE SUPPLIED IN A TABLE PUT IN A .DATA
*    SECTION. THIS DATA IS INCLUDED IN A SEPARATE FILE TO PRESERVE THE GENERIC
*    NATURE OF THE PROGRAM. FOR THE SAME PURPOSE, THE SIZE OF THE FFT N AND
*    LOG2(N) ARE DEFINED IN A .GLOBL DIRECTIVE AND SPECIFIED DURING LINKING.
*
*    AUTHOR: PANOS E. PAPAMICHALIS
*            TEXAS INSTRUMENTS          JULY 16, 1987
*
        .GLOBL  FFT                 ; ENTRY POINT FOR EXECUTION
        .GLOBL  N                   ; FFT SIZE
        .GLOBL  M                   ; LOG2(N)
        .GLOBL  SINE                ; ADDRESS OF SINE TABLE
*
INP     .USECT  "IN",1024           ; MEMORY WITH INPUT DATA
        .BSS    OUTP,1024           ; MEMORY WITH OUTPUT DATA
*
      .TEXT
*
*    INITIALIZE
*
        .WORD   FFT                 ; STARTING LOCATION OF THE PROGRAM
*
        .SPACE  100                 ; RESERVE 100 WORDS FOR VECTORS, ETC.
*
FFTSIZ  .WORD   N
LOGFFT  .WORD   M
SINTAB  .WORD   SINE
INPUT   .WORD   INP
OUTPUT  .WORD   OUTP
*
FFT:    LDP     FFTSIZ              ; COMMAND TO LOAD DATA PAGE POINTER
*
        LDI     @FFTSIZ,IR1
        LSH     -2,IR1              ; IR1=N/4, POINTER FOR SIN/COS TABLE
        LDI     0,AR6               ; AR6 HOLDS THE CURRENT STAGE NUMBER
        LDI     @FFTSIZ,IR0
        LSH     1,IR0               ; IR0=2*N1 (BECAUSE OF REAL/IMAG)
        LDI     @FFTSIZ,R7          ; R7=N2
        LDI     1,AR7               ; INITIALIZE REPEAT COUNTER OF FIRST
*                                   ; LOOP
        LDI     1,AR5               ; INITIALIZE IE INDEX (AR5=IE)
*
*    OUTER LOOP
*
LOOP:   NOP     *++AR6(1)           ; CURRENT FFT STAGE
        LDI     @INPUT,AR0          ; AR0 POINTS TO X(I)
        ADDI    R7,AR0,AR2          ; AR2 POINTS TO X(L)
        LDI     AR7,RC
        SUBI    1,RC                ; RC SHOULD BE ONE LESS THAN DESIRED #
*
*    FIST LOOP
*
        RPTB    BLK1
        ADDF    *AR0,*AR2,R0        ; R0=X(I)+X(L)
        SUBF    *AR2++,*AR0++,R1    ; R1=X(I)-X(L)
        ADDF    *AR2,*AR0,R2        ; R2=Y(I)+Y(L)
        SUBF    *AR2,*AR0,R3        ; R3=Y(I)-Y(L)
        STF     R2,*AR0--           ; Y(I)=R2  AND...
||      STF     R3,*AR2--           ;   Y(L)=R3
BLK1    STF     R0,*AR0++(IR0)      ; X(I)=R0  AND...
||      STF     R1,*AR2++(IR0)      ;   X(L)=R1 AND AR0,2 = AR0,2 + 2*N1
*
*    IF THIS IS THE LAST STAGE, YOU ARE DONE
*
        CMPI    @LOGFFT,AR6
        BZD     END
*
*    MAIN INNER LOOP
*
        LDI     2,AR1               ; INIT LOOP COUNTER FOR INNER LOOP
        LDI     @SINTAB,AR4         ; INITIALIZE IA INDEX (AR4=IA)
INLOP:  ADDI    AR5,AR4             ; IA=IA+IE; AR4 POINTS TO COSINE
        LDI     AR1,AR0
        ADDI    2,AR1               ; INCREMENT INNER LOOP COUNTER
        ADDI    @INPUT,AR0          ; (X(I),Y(I)) POINTER
        ADDI    R7,AR0,AR2          ; (X(L),Y(L)) POINTER
        LDI     AR7,RC
        SUBI    1,RC                ; RC SHOULD BE ONE LESS THAN DESIRED #
        LDF     *AR4,R6             ; R6=SIN
*
*    SECOND LOOP
*
        RPTB    BLK2
        SUBF    *AR2,*AR0,R2        ; R2=X(I)-X(L)
        SUBF    *+AR2,*+AR0,R1      ; R1=Y(I)-Y(L)
        MPYF    R2,R6,R0            ; R0=R2*SIN AND...
||      ADDF    *+AR2,*+AR0,R3      ; R3=Y(I)+Y(L)
        MPYF    R1,*+AR4(IR1),R3    ; R3=R1*COS AND...
||      STF     R3,*+AR0            ; Y(I)=Y(I)+Y(L)
        SUBF    R0,R3,R4            ; R4=R1*COS-R2*SIN
        MPYF    R1,R6,R0            ; R0=R1*SIN AND...
||      ADDF    *AR2,*AR0,R3        ; R3=X(I)+X(L)
        MPYF    R2,*+AR4(IR1),R3    ; R3=R2*COS AND...
||      STF     R3,*AR0++(IR0)      ; X(I)=X(I)+X(L) AND AR0=AR0+2*N1
        ADDF    R0,R3,R5            ; R5=R2*COS+R1*SIN
BLK2    STF     R5,*AR2++(IR0)      ; X(L)=R2*COS+R1*SIN, INCR AR2 AND...
||      STF     R4,*+AR2            ;   Y(L)=R1*COS-R2*SIN
*
        CMPI    R7,AR1
        BNE     INLOP               ; LOOP BACK TO THE INNER LOOP
```

```
*
        LSH     1,AR7           ; INCREMENT LOOP COUNTER FOR NEXT TIME
*
        LSH     1,AR5           ; IE=2*IE
        LDI     R7,IR0          ; N1=N2
        LSH     -1,R7           ; N2=N2/2
        BR      LOOP            ; NEXT FFT STAGE
*
*   STORE RESULT OUT USING BIT-REVERSED ADDRESSING
*
END:    LDI     @FFTSIZ,RC      ; RC=N
        SUBI    1,RC            ; RC SHOULD BE ONE LESS THAN DESIRED #
        LDI     @FFTSIZ,IR0     ; IR0=SIZE OF FFT=N
        LDI     2,IR1
        LDI     @INPUT,AR0
        LDI     @OUTPUT,AR1

        RPTB    BITRV
        LDF     *++AR0(1),R0
::      LDF     *AR0++(IR0)B,R1
BITRV   STF     R0,*++AR1(1)
::      STF     R1,*AR1++(IR1)
*
SELF    BR      SELF            ; BRANCH TO ITSELF AT THE END
        .END
```

Appendix A2. fft_2 – Radix-2 Complex FFT to Be Called as a C Function

```
* NAME:
*    fft_2 --- RADIX-2 COMPLEX FFT TO BE CALLED AS A C FUNCTION.
*
* SYNOPSIS:
*    INT  fft_2(N, M, DATA)
*    INT  N      FFT SIZE: N=2**M
*    INT  M      NUMBER OF STAGES = LOG2(N)
*    FLOAT *DATA    ARRAY WITH INPUT AND OUTPUT DATA
*
* DESCRIPTION:
*    GENERIC FUNCTION TO DO A RADIX-2 FFT COMPUTATION ON THE 320C30.
*    THE DATA ARRAY IS 2*N-LONG, WITH REAL AND IMAGINARY VALUES ALTERNATING.
*    THE PROGRAM IS BASED ON THE FORTRAN PROGRAM IN THE BURRUS AND PARKS
*    BOOK, P. 111.
*
*    THE COMPUTATION IS DONE IN PLACE, AND THE ORIGINAL DATA IS DESTROYED.
*    BIT REVERSAL IS IMPLEMENTED AT THE END OF THE FUNCTION. IF THIS IS NOT
*    NECESSARY, THIS PART CAN BE COMMENTED OUT.
*
*    THE SINE/COSINE TABLE FOR THE TWIDDLE FACTORS IS EXPECTED TO BE SUPPLIED
*    DURING LINK TIME, AND IT SHOULD HAVE THE FOLLOWING FORMAT:
*
*         .GLOBAL   _sine
*         .DATA
*         _sine     .FLOAT  VALUE1  = sin(0*2*pi/N)
*         .FLOAT    VALUE  = sin(1*2*pi/N)
*         ......
*         .FLOAT    VALUE(5N/4)  = sin((5*N/4-1)*2*pi/N)
*
*    THE VALUES VALUE1, VALUE2, ETC., ARE THE SAME WAVE VALUES. FOR AN
*    N-POINT FFT, THERE ARE N+N/4 VALUES FOR A FULL AND A QUARTER PERIOD OF
*    THE SINE WAVE. IN THIS WAY, A FULL SINE AND COSINE PERIOD ARE AVAILABLE
*    (SUPERIMPOSED).
*
*    STACK STRUCTURE UPON THE CALL:
*
*                  +--------------+
*         -FP(4)   :    DATA      :
*         -FP(3)   :    M         :
*         -FP(2)   :    N         :
*         -FP(1)   : RETURN ADDR  :
*         -FP(0)   :    OLD FP    :
*                  +--------------+
*
*    REGISTERS USED: R0, R1, R2, R3, R4, R5, R6, R7, AR0, AR1, AR2, AR4, AR5
*                    AR6, AR7, IR0, IR1, RS, RE, RC
*
*    AUTHOR: PANOS E. PAPAMICHALIS
*            TEXAS INSTRUMENTS          OCTOBER 13, 1987
*
****************************************************************************
```

```
*
FP        .set      AR3
*
          .GLOBL    _fft_2               ; ENTRY POINT FOR EXECUTION
          .GLOBL    _sine                ; ADDRESS OF SINE TABLE
*
          .BSS      FFTSIZ,1
          .BSS      LOGFFT,1
          .BSS      INPUT,1
*
          .TEXT
*
SINTAB    .word     _sine
*
*   INITIALIZE C FUNCTION
*
_fft_2:   PUSH      FP                   ; SAVE DEDICATED REGISTERS
          LDI       SP,FP
          PUSH      R4
          PUSH      R5
          PUSHF     R6
          PUSHF     R7
          PUSH      AR4
          PUSH      AR5
          PUSH      AR6
          PUSH      AR7
*
          LDI       *-FP(2),R0           ; MOVE ARGUMENTS TO LOCATIONS MATCHING
          STI       R0,@FFTSIZ           ; THE NAMES IN THE PROGRAM
          LDI       *-FP(3),R0
          STI       R0,@LOGFFT
          LDI       *-FP(4),R0
          STI       R0,@INPUT
*
*   INITIALIZE FFT ROUTINE
*
          LDI       @FFTSIZ,IR1
          LSH       -2,IR1               ; IR1=N/4, POINTER FOR SIN/COS TABLE
          LDI       0,AR6                ; AR6 HOLDS THE CURRENT STAGE NUMBER
          LDI       @FFTSIZ,IR0
          LSH       1,IR0                ; IR0=2*N1 (BECAUSE OF REAL/IMAG)
          LDI       @FFTSIZ,R7           ; R7=N2
          LDI       1,AR7                ; INITIALIZE REPEAT COUNTER OF FIRST
*                                        ;   LOOP
          LDI       1,AR5                ; INITIALIZE IE INDEX (AR5=IE)
*
*   OUTER LOOP
*
LOOP:     NOP       *++AR6(1)            ; CURRENT FFT STAGE
          LDI       @INPUT,AR0           ; AR0 POINTS TO X(I)
          ADDI      R7,AR0,AR2           ; AR2 POINTS TO X(L)
          LDI       AR7,RC
          SUBI      1,RC                 ; RC SHOULD BE ONE LESS THAN DESIRED #
*
```

```
*   FIST LOOP
*
        RPTB    BLK1
        ADDF    *AR0,*AR2,R0          ; R0=X(I)+X(L)
        SUBF    *AR2++,*AR0++,R1      ; R1=X(I)-X(L)
        ADDF    *AR2,*AR0,R2          ; R2=Y(I)+Y(L)
        SUBF    *AR2,*AR0,R3          ; R3=Y(I)-Y(L)
        STF     R2,*AR0--             ; Y(I)=R2  AND...
||      STF     R3,*AR2--             ;   Y(L)=R3
BLK1    STF     R0,*AR0++(IR0)        ; X(I)=R0  AND...
||      STF     R1,*AR2++(IR0)        ;   X(L)=R1 AND AR0,2 = AR0,2 + 2*N1
*
*   IF THIS IS THE LAST STAGE, YOU ARE DONE
*
        CMPI    @LOGFFT,AR6
        BZD     END
*
*   MAIN INNER LOOP
*
        LDI     2,AR1                 ; INIT LOOP COUNTER FOR INNER LOOP
        LDI     @SINTAB,AR4           ; INITIALIZE IA INDEX (AR4=IA)
INLOP:  ADDI    AR5,AR4               ; IA=IA+IE; AR4 POINTS TO COSINE
        LDI     AR1,AR0
        ADDI    2,AR1                 ; INCREMENT INNER LOOP COUNTER
        ADDI    @INPUT,AR0            ; (X(I),Y(I)) POINTER
        ADDI    R7,AR0,AR2            ; (X(L),Y(L)) POINTER
        LDI     AR7,RC
        SUBI    1,RC                  ; RC SHOULD BE ONE LESS THAN DESIRED #
        LDF     *AR4,R6               ; R6=SIN
*
*   SECOND LOOP
*
        RPTB    BLK2
        SUBF    *AR2,*AR0,R2          ; R2=X(I)-X(L)
        SUBF    *+AR2,*+AR0,R1        ; R1=Y(I)-Y(L)
        MPYF    R2,R6,R0              ; R0=R2*SIN AND...
||      ADDF    *+AR2,*+AR0,R3        ;   R3=Y(I)+Y(L)
        MPYF    R1,*+AR4(IR1),R3      ; R3=R1*COS AND...
||      STF     R3,*+AR0              ;   Y(I)=Y(I)+Y(L)
        SUBF    R0,R3,R4              ; R4=R1*COS-R2*SIN
        MPYF    R1,R6,R0              ; R0=R1*SIN AND...
||      ADDF    *AR2,*AR0,R3          ;   R3=X(I)+X(L)
        MPYF    R2,*+AR4(IR1),R3      ; R3=R2*COS AND...
||      STF     R3,*AR0++(IR0)        ;   X(I)=X(I)+X(L) AND AR0=AR0+2*N1
        ADDF    R0,R3,R5              ; R5=R2*COS+R1*SIN
BLK2    STF     R5,*AR2++(IR0)        ; X(L)=R2*COS+R1*SIN, INCR AR2 AND...
||      STF     R4,*+AR2              ;   Y(L)=R1*COS-R2*SIN
*
        CMPI    R7,AR1
        BNE     INLOP                 ; LOOP BACK TO THE INNER LOOP
*
        LSH     1,AR7                 ; INCREMENT LOOP COUNTER FOR NEXT TIME
*
        LSH     1,AR5                 ; IE=2*IE
        LDI     R7,IR0                ; N1=N2
        LSH     -1,R7                 ; N2=N2/2
        BR      LOOP                  ; NEXT FFT STAGE
*
*   DO THE BIT-REVERSING OF THE OUTPUT
*
END:    LDI     @FFTSIZ,RC            ; RC=N
        SUBI    1,RC                  ; RC SHOULD BE ONE LESS THAN DESIRED #
        LDI     @FFTSIZ,IR0           ; IR0=SIZE OF FFT=N
        LDI     @INPUT,AR0
        LDI     @INPUT,AR1
*
        RPTB    BITRV
        CMPI    AR0,AR1
        BGE     CONT
        LDF     *AR0,R0
||      LDF     *AR1,R1
        STF     R0,*AR1
||      STF     R1,*AR0
        LDF     *+AR0(1),R0
||      LDF     *+AR1(1),R1
        STF     R0,*+AR1(1)
||      STF     R1,*+AR0(1)
CONT    NOP     *++AR0(2)
BITRV   NOP     *AR1++(IR0)B
*
*   RESTORE THE REGISTER VALUES AND RETURN
*
        POP     AR7
        POP     AR6
        POP     AR5
        POP     AR4
        POPF    R7
        POPF    R6
        POP     R5
        POP     R4
        POP     FP
        RETS
```

Appendix A3. Complex, Radix-2 DIT FFT – R2DIT.ASM

```
                COMPLEX, RADIX-2 DIT FFT  :  R2DIT.ASM
             ------------------------------------------

   GENERIC PROGRAM FOR A FAST LOOPED-CODE RADIX-2 DIT FFT COMPUTATION
                        ON THE TMS320C30

   WRITTEN BY: RAIMUND MEYER, KARL SCHWARZ              19.07.89
               LEHRSTUHL FUER NACHRICHTENTECHNIK
               UNIVERSITAET ERLANGEN-NUERNBERG
               CAUERSTRASSE 7, D-8520 ERLANGEN, FRG

   THE (COMPLEX) DATA RESIDE IN INTERNAL MEMORY. THE COMPUTATION IS DONE
   IN-PLACE, BUT THE RESULT IS MOVED TO ANOTHER MEMORY SECTION TO
   DEMONSTRATE THE BIT-REVERSED ADDRESSING.

   FOR THIS PROGRAM THE MINIMUM FFTLENGTH IS 32 POINTS BECAUSE OF THE
   SEPARATE STAGES.

   FIRST TWO PASSES ARE REALIZED AS A FOUR BUTTERFLY LOOP SINCE THE
   MULTIPLIES ARE TRIVIAL. THE MULTIPLIER IS ONLY USED FOR A LOAD IN
   PARALLEL WITH AN ADDF OR SUBF.

************************************************************************

   EXAMPLE FOR A 1024-POINT FFT (EXCLUDING BIT REVERSAL):

   MEMORY SIZE:
        PROGRAM                       =    229 WORDS
        DATA (TWIDDLE FACTORS)        =    512 WORDS

   CYCLES PER BUTTERFLY:
        STAGES 1 AND 2                =      4
        STAGES 3 TO  8                =      8
        STAGE 9                       =      8.25
        STAGE 10                      =      8.5

   AVERAGE CYCLES/BUTTERFLY            =     7.275
   TOTAL BUTTERFLYCYCLES               = 37248
   INITIALIZATION OVERHEAD             =  2181  =  5.55 % OF TOTAL TIME
   TOTAL NUMBER OF INSTRUCTION CYCLES = 39429
   TOTAL TIME FOR A 1024 POINT FFT     =     2.36 ms (EXCLUDING BIT
                                                          REVERSAL)

************************************************************************
```

```
************************************************************************

   THIS PROGRAM INCUDES FOLLOWING FILES:
   -------------------------------------
   THE FILE 'TWIDIKBR.ASM' CONSISTS OF TWIDDLE FACTORS

   THE TWIDDLE FACTORS ARE STORED IN BITREVERSED ORDER AND WITH A TABLE
   LENGTH OF N/2 (N = FFTLENGTH).
   EXAMPLE: SHOWN FOR N=32, WN(n) = COS(2*PI*n/N) - j*SIN(2*PI*n/N)

   ADDRESS   COEFFICIENT
    0          R(WN(0)) = COS(2*PI*0/32) = 1
    1         -I(WN(0)) = SIN(2*PI*0/32) = 0
    2          R(WN(4)) = COS(2*PI*4/32) = 0.707
    3         -I(WN(4)) = SIN(2*PI*4/32) = 0.707
    .                          .
    .                          .
   12          R(WN(3)) = COS(2*PI*3/32) = 0.831
   13         -I(WN(3)) = SIN(2*PI*3/32) = 0.556
   14          R(WN(7)) = COS(2*PI*7/32) = 0.195
   15         -I(WN(7)) = SIN(2*PI*7/32) = 0.981

   WHEN GENERATED FOR A FFT LENGTH OF 1024, THE TABLE IS FOR ALL
   AVAILABLE FFT OF LESS OR EQUAL LENGTH.

   THE MISSING TWIDDLE FACTORS (WN(),WN(),....) ARE GENERATED BY USING
   THE SYMMETRY WN(N/4+n) = -j*WN(n). THIS CAN BE EASILY REALIZED BY
   CHANGING REAL- AND IMAGINARY PART OF THE TWIDDLE FACTORS AND BY
   NEGATING THE NEW REAL PART.

   TO CHANGE THE FFT LENGTH, ONLY THE PARAMETERS IN THE HEADER OF
   TWIDIKBR.ASM AND THE INPUT AND OUTPUT VECTOR LENGTHS NEED TO BE
   ALTERED.
************************************************************************
                                                  +
   AR + j AI ---------------------------------------------- AR' + j AI'
                                            \        / +
                                             \      /
                                               \ /
                                               / \
                                             /      \
                                            /        \ +
   BR + j BI ---- ( COS - j SIN ) ------------------------- BR' + j BI'
                                                  -

   TR = BR * COS + BI * SIN
   TI = BR * SIN - BI * COS
   AR'= AR + TR
   AI'= AI - TI
   BR'= AR - TR
   BI'= AI + TI

************************************************************************
```

```
*
        .global FFT
        .global N
        .global NHALB
        .global NVIERT
        .global NATCHEL
        .global M
        .global SINE
*
        .BSS    INP,2048          ; INPUT VECTOR LENGTH = 2N (DEPENDS
*                                 ;   ON N)
        .BSS    OUTP,2048         ; OUTPUT VECTOR LENGTH = 2N (DEPENDS
*                                 ;   ON N)
*
        .text
*
FFTSIZ  .word   N
FG4M2   .word   NVIERT-2
FG4M3   .word   NVIERT-3
FG8M2   .word   NATCHEL-2
FG2     .word   NHALB
FG2M3   .word   NHALB-3
LOGFFT  .word   M
SINTAB  .word   SINE
SINTM1  .word   SINE-1
SINTP2  .word   SINE+2
INPUT   .word   INP
INPUTP2 .word   INP+2
OUTPUT  .word   OUTP

*   AR0 : AR + AI
*   AR1 : BR + BI
*   AR2 : CR + CI + CR' + CI'
*   AR3 : DR + DI
*   AR4 : AR' + AI'
*   AR5 : BR' + BI'
*   AR6 : DR' + DI'
*   AR7 : FIRST TWIDDLE FACTOR = 1
*
FFT:    LDP     FFTSIZ            ; LOAD PAGE POINTER
        LDI     @FG2,IR0          ; IR0 = N/2 = OFFSET BETWEEN INPUTS
        LDI     @SINTAB,AR7       ; AR7 POINTS TO TWIDDLE FACTOR 1
        LDI     @INPUT,AR0        ; AR0 POINTS TO AR
        ADDI    IR0,AR0,AR1       ; AR1 POINTS TO BR
        ADDI    IR0,AR1,AR2       ; AR2 POINTS TO CR
        ADDI    IR0,AR2,AR3       ; AR3 POINTS TO DR
        LDI     AR0,AR4           ; AR4 POINTS TO AR'
        LDI     AR1,AR5           ; AR5 POINTS TO BR'
        LDI     AR3,AR6           ; AR6 POINTS TO DR'
        LDI     2,IR1             ; ADDRESS OFFSET
        LSH     -1,IR0            ; IR0 = N/4 = NUMBER OF R4-BUTTERFLIES
        LDI     IR0,RC
        SUBI    2,RC
*
*   FIRST 2 STAGES AS RADIX-4 BUTTERFLY
*
*   FILL PIPELINE
*
        ADDF    *AR2,*AR0,R4          ; R4 = AR + CR
        SUBF    *AR2,*AR0++,R5        ; R5 = AR - CR
        ADDF    *AR1,*AR3,R6          ; R6 = DR + BR
        SUBF    *AR1++,*AR3++,R7      ; R7 = DR - BR
        ADDF    R6,R4,R0              ; AR' = R0 = R4 + R6
        MPYF    *AR3++,*AR7,R1        ; R1 = DI , BR' = R3 = R4 - R6
||      SUBF    R6,R4,R3
        ADDF    R1,*AR1,R0            ; R0 = BI + DI , AR' = R0
||      STF     R0,*AR4++
        SUBF    R1,*AR1++,R1          ; R1 = BI - DI , BR' = R3
||      STF     R3,*AR5++
        ADDF    R1,R5,R2              ; CR' = R2 = R5 + R1
        MPYF    *++AR2,*AR7,R1        ; R1 = CI , DR' = R3 = R5 - R1
||      SUBF    R1,R5,R3
        ADDF    R1,*AR0,R2            ; R2 = AI + CI , CR' = R2
||      STF     R2,*AR2++(IR1)
        SUBF    R1,*AR0++,R6          ; R6 = AI - CI , DR' = R3
||      STF     R3,*AR6++
        ADDF    R0,R2,R4              ; AI' = R4 = R2 + R0
*
*   RADIX-4 BUTTERFLY LOOP
*
        RPTB    BLK1
        MPYF    *AR2--,*AR7,R0        ; R0 = CR , (BI' = R2 = R2 - R0)
||      SUBF    R0,R2,R2
        MPYF    *AR1++,*AR7,R1        ; R1 = BR , (CI' = R3 = R6 + R7)
||      ADDF    R7,R6,R3
        ADDF    R0,*AR0,R4            ; R4 = AR + CR , (AI' = R4)
||      STF     R4,*AR4++
        SUBF    R0,*AR0++,R5          ; R5 = AR - CR , (BI' = R2)
||      STF     R2,*AR5++
        SUBF    R7,R6,R7              ; (DI' = R7 = R6 - R7)
        ADDF    R1,*AR3,R6            ; R6 = DR + BR , (DI' = R7)
||      STF     R7,*AR6++
        SUBF    R1,*AR3++,R7          ; R7 = DR - BR , (CI' = R3)
||      STF     R3,*AR2++
        ADDF    R6,R4,R0              ; AR' = R0 = R4 + R6
        MPYF    *AR3++,*AR7,R1        ; R1 = DI , BR' = R3 = R4 - R6
||      SUBF    R6,R4,R3
        ADDF    R1,*AR1,R0            ; R0 = BI + DI , AR' = R0
||      STF     R0,*AR4++
        SUBF    R1,*AR1++,R1          ; R1 = BI - DI , BR' = R3
||      STF     R3,*AR5++
        ADDF    R1,R5,R2              ; CR' = R2 = R5 + R1
        MPYF    *++AR2,*AR7,R1        ; R1 = CI , DR' = R3 = R5 - R1
||      SUBF    R1,R5,R3
        ADDF    R1,*AR0,R2            ; R2 = AI + CI , CR' = R2
||      STF     R2,*AR2++(IR1)
        SUBF    R1,*AR0++,R6          ; R6 = AI - CI , DR' = R3
||      STF     R3,*AR6++
```

```
BLK1    ADDF    R0,R2,R4            ; AI' = R4 = R2 + R0
*
*   CLEAR PIPELINE
*
        SUBF    R0,R2,R2            ; BI' = R2 = R2 - R0
        ADDF    R7,R6,R3            ; CI' = R3 = R6 + R7
        STF     R4,*AR4             ; AI' = R4 , BI' = R2
||      STF     R2,*AR5
        SUBF    R7,R6,R7            ; DI' = R7 = R6 - R7
        STF     R7,*AR6             ; DI' = R7 , CI' = R3
||      STF     R3,*--AR2

*
*   THIRD TO LAST OF STAGE 2
*

        LDI     @FG2,IR1
        LDI     IR0,AR5
        SUBI    1,AR5
        LDI     1,AR6

STUFE   LDI     @SINTAB,AR7         ; POINTER TO TWIDDLE FACTOR
        LDI     0,AR4               ; GROUP COUNTER
        LDI     @INPUT,AR0          ; UPPER REAL BUTTERFLY INPUT
        LDI     AR0,AR2             ; UPPER REAL BUTTERFLY OUTPUT
        ADDI    IR0,AR0,AR3         ; LOWER REAL BUTTERFLY OUTPUT
        LDI     AR3,AR1             ; LOWER REAL BUTTERFLY INPUT
        LSH     1,AR6               ; DOUBLE GROUP COUNT
        LSH     -2,AR5              ; HALF BUTTERFLY COUNT
        LSH     1,AR5               ; CLEAR LSB
        LSH     -1,IR0              ; HALF STEP FROM UPPER TO LOWER REAL
*                                   ;   PART
        LSH     -1,IR1
        ADDI    1,IR1               ; STEP FROM OLD IMAGINARY TO NEW REAL
*                                   ;   VALUE
        LDF     *AR1++,R6           ; DUMMY LOAD, ONLY FOR ADDRESS UPDATE
||      LDF     *AR7,R7             ; R7 = COS
*
GRUPPE
*
*   FILL PIPELINE                   ; AR0 = UPPER REAL BUTTERFLY INPUT
*                                   ; AR1 = LOWER REAL BUTTERFLY INPUT
*                                   ; AR2 = UPPER REAL BUTTERFLY OUTPUT
*                                   ; AR3 = LOWER REAL BUTTERFLY OUTPUT
*                                   ; THE IMAGINARY PART HAS TO FOLLOW
        LDF     *++AR7,R6           ; R6 = SIN
        MPYF    *AR1--,R6,R1        ; R1 = BI * SIN
||      ADDF    *++AR4,R0,R3        ; DUMMY ADDF FOR COUNTER UPDATE
        MPYF    *AR1,R7,R0          ; R0 = BR * COS
        MPYF    *AR1++,*AR7--,R0    ; R3 = TR = R0 + R1 , R0 = BR * SIN
||      ADDF    R0,R1,R3
        MPYF    *AR1++,R7,R1        ; R1 = BI * COS , R2 = AR - TR
||      SUBF    R3,*AR0,R2
        ADDF    *AR0++,R3,R5        ; R5 = AR + TR , BR' = R2
||      STF     R2,*AR3++
        LDI     AR5,RC
************************************************************************
*                                                                      *
*   FIRST BUTTERFLY-TYPE:                                              *
*                                                                      *
*       TR = BR * COS + BI * SIN                                       *
*       TI = BR * SIN - BI * COS                                       *
*       AR'= AR + TR                                                   *
*       AI'= AI - TI                                                   *
*       BR'= AR - TR                                                   *
*       BI'= AI + TI                                                   *
*                                                                      *
************************************************************************
*
        RPTB    BFLY1
*
        MPYF    *+AR1,R6,R5         ; R5 = BI * SIN , (AR' = R5)
||      STF     R5,*AR2++
        SUBF    R1,R0,R2            ; (R2 = TI = R0 - R1)
        MPYF    *AR1,R7,R0          ; R0 = BR * COS , (R3 = AI + TI)
||      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++,R4        ; (R4 = AI - TI , BI' = R3)
||      STF     R3,*AR3++
        ADDF    R0,R5,R3            ; R3 = TR = R0 + R5
        MPYF    *AR1++,R6,R0        ; R0 = BR * SIN , R2 = AR - TR
||      SUBF    R3,*AR0,R2
        MPYF    *AR1++,R7,R1        ; R1 = BI * COS , (AI' = R4)
||      STF     R4,*AR2++
BFLY1   ADDF    *AR0++,R3,R5        ; R5 = AR + TR , BR' = R2
||      STF     R2,*AR3++
*
*   SWITCH OVER TO NEXT GROUP
*
        SUBF    R1,R0,R2            ; R2 = TI = R0 - R1
        ADDF    R2,*AR0,R3          ; R3 = AI + TI , AR' = R5
||      STF     R5,*AR2++
        SUBF    R2,*AR0++(IR1),R4   ; R4 = AI - TI , BI' = R3
||      STF     R3,*AR3++(IR1)
        NOP     *AR1++(IR1)         ; ADDRESS UPDATE
        MPYF    *AR1--,R7,R1        ; R1 = BI * COS , AI' = R4
||      STF     R4,*AR2++(IR1)
        MPYF    *AR1,R6,R0          ; R0 = BR * SIN
        MPYF    *AR1++,*AR7++,R0    ; R3 = TR = R1 - R0 , R0 = BR * COS
||      SUBF    R0,R1,R3
        MPYF    *AR1++,R6,R1        ; R1 = BI * SIN , R2 = AR - TR
||      SUBF    R3,*AR0,R2
        ADDF    *AR0++,R3,R5        ; R5 = AR + TR , BR' = R2
||      STF     R2,*AR3++
        LDI     AR5,RC
```

```
*************************************************************************
*                                                                       *
*    SECOND BUTTERFLY-TYPE:                                             *
*                                                                       *
*        TR = BI * COS - BR * SIN                                       *
*        TI = BI * SIN + BR * COS                                       *
*        AR'= AR + TR                                                   *
*        AI'= AI - TI                                                   *
*        BR'= AR - TR                                                   *
*        BI'= AI + TI                                                   *
*                                                                       *
*************************************************************************
*
        RPTB    BFLY2
*
        MPYF    *+AR1,R7,R5         ; R5 = BI * COS , (AR' = R5)
||      STF     R5,*AR2++
        ADDF    R1,R0,R2            ; (R2 = TI = R0 + R1)
        MPYF    *AR1,R6,R0          ; R0 = BR * SIN , (R3 = AI + TI)
||      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++,R4        ; (R4 = AI - TI , BI' = R3)
||      STF     R3,*AR3++
        SUBF    R0,R5,R3            ; TR = R3 = R5 - R0
        MPYF    *AR1++,R7,R0        ; R0 = BR * COS , R2 = AR - TR
||      SUBF    R3,*AR0,R2
        MPYF    *AR1++,R6,R1        ; R1 = BI * SIN , (AI' = R4)
||      STF     R4,*AR2++
BFLY2   ADDF    *AR0++,R3,R5        ; R5 = AR + TR , BR' = R2
||      STF     R2,*AR3++
*
*    CLEAR PIPELINE
*
        ADDF    R1,R0,R2            ; R2 = TI = R0 + R1
        ADDF    R2,*AR0,R3          ; R3 = AI + TI
||      STF     R5,*AR2++           ; AR' = R5
        CMPI    AR6,AR4
        BNED    GRUPPE              ; DO FOLLOWING 3 INSTRUCTIONS
        SUBF    R2,*AR0++(IR1),R4   ; R4 = AI - TI , BI' = R3
||      STF     R3,*AR3++(IR1)
        LDF     *++AR7,R7           ; R7 = COS
||      STF     R4,*AR2++(IR1)      ; AI' = R4
        NOP     *AR1++(IR1)         ; BRANCH HERE
*
*    END OF THIS BUTTERFLY GROUP
*
        CMPI    4,IR0               ; JUMP OUT AFTER LD(N)-3 STAGE
        BNZ     STUFE
*
*
*     SECOND TO LAST STAGE
*
        LDI     @INPUT,AR0          ; UPPER INPUT
        LDI     AR0,AR2             ; UPPER OUTPUT
        ADDI    IR0,AR0,AR1         ; LOWER INPUT
        LDI     AR1,AR3             ; LOWER OUTPUT
        LDI     @SINTP2,AR7         ; POINTER TO TWIDDLE FACTOR
        LDI     5,IR0               ; DISTANCE BETWEEN TWO GROUPS
        LDI     @FG8M2,RC
*
*    FILL PIPELINE
*
*    1. BUTTERFLY: w^0
*
        ADDF    *AR0,*AR1,R2        ; AR' = R2 = AR + BR
        SUBF    *AR1++,*AR0++,R3    ; BR' = R3 = AR - BR
        ADDF    *AR0,*AR1,R0        ; AI' = R0 = AI + BI
        SUBF    *AR1++,*AR0++,R1    ; BI' = R1 = AI - BI
*
*    2. BUTTERFLY: w^0
*
        ADDF    *AR0,*AR1,R6        ; AR' = R6 = AR + BR
        SUBF    *AR1++,*AR0++,R7    ; BR' = R7 = AR - BR
        ADDF    *AR0,*AR1,R4        ; AI' = R4 = AI + BI
        SUBF    *AR1++(IR0),*AR0++(IR0),R5   ; BI' = R5 = AI - BI
        STF     R2,*AR2++           ; (AR' = R2)
||      STF     R3,*AR3++           ; (BR' = R3)
        STF     R0,*AR2++           ; (AI' = R0)
||      STF     R1,*AR3++           ; (BI' = R1)
        STF     R6,*AR2++           ; AR' = R6
||      STF     R7,*AR3++           ; BR' = R7
        STF     R4,*AR2++(IR0)      ; AI' = R4
||      STF     R5,*AR3++(IR0)      ; BI' = R5
*
*    3. BUTTERFLY: w^N/4
*
        ADDF    *AR0++,*+AR1,R5     ; AR' = R5 = AR + BI
        SUBF    *AR1,*AR0,R4        ; AI' = R4 = AI - BR
        ADDF    *AR1++,*AR0--,R6    ; BI' = R6 = AI + BR
        SUBF    *AR1++,*AR0++,R7    ; BR' = R7 = AR - BI
*
*    4. BUTTERFLY: w^N/4
*
        ADDF    *+AR1,*++AR0,R3     ; AR' = R3 = AR + BI
        LDF     *-AR7,R1            ; R1 = 0 (FOR INNER LOOP)
||      LDF     *AR1++,R0           ; R0 = BR (FOR INNER LOOP)
        SUBF    *AR1++(IR0),*AR0++,R2   ; BR' = R2 = AR - BI
        STF     R5,*AR2++           ; (AR' = R5)
||      STF     R7,*AR3++           ; (BR' = R7)
        STF     R6,*AR3++           ; (BI' = R6)
*
*    5. TO M. BUTTERFLY:
*
        RPTB    BF2END
*
        LDF     *AR7++,R7           ; R7 = COS , ((AI' = R4))
||      STF     R4,*AR2++
        LDF     *AR7++,R6           ; R6 = SIN , (BR' = R2)
||      STF     R2,*AR3++
```

```
        MPYF    *++AR1,R6,R5        ; R5 = BI * SIN , (AR' = R3)
||      STF     R3,*AR2++
        ADDF    R1,R0,R2            ; (R2 = TI = R0 + R1)
        MPYF    *AR1,R7,R0          ; R0 = BR * COS , (R3 = AI + TI)
||      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++(IR0),R4   ; (R4 = AI - TI , BI' = R3)
||      STF     R3,*AR3++(IR0)
        ADDF    R0,R5,R3            ; R3 = TR = R0 + R5
        MPYF    *AR1++,R6,R0        ; R0 = BR * SIN , R2 = AR - TR
||      SUBF    R3,*AR0,R2
        MPYF    *AR1++,R7,R1        ; R1 = BI * COS , (AI' = R4)
||      STF     R4,*AR2++(IR0)
        ADDF    *AR0++,R3,R5        ; R5 = AR + TR , BR' = R2
||      STF     R2,*AR3++
*
        MPYF    *++AR1,R6,R5        ; R5 = BI * SIN , (AR' = R5)
||      STF     R5,*AR2++
        SUBF    R1,R0,R2            ; (R2 = TI = R0 - R1)
        MPYF    *AR1,R7,R0          ; R0 = BR * COS , (R3 = AI + TI)
||      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++,R4        ; (R4 = AI - TI , BI' = R3)
||      STF     R3,*AR3++
        ADDF    R0,R5,R3            ; R3 = TR = R0 + R5
        MPYF    *AR1++,R6,R0        ; R0 = BR * SIN , R2 = AR - TR
||      SUBF    R3,*AR0,R2
        MPYF    *AR1++(IR0),R7,R1   ; R1 = BI * COS , (AI' = R4)
||      STF     R4,*AR2++
        ADDF    *AR0++,R3,R3        ; R3 = AR + TR , BR' = R2
||      STF     R2,*AR3++
*
        MPYF    *++AR1,R7,R5        ; R5 = BI * COS , (AR' = R3)
||      STF     R3,*AR2++
        SUBF    R1,R0,R2            ; (R2 = TI = R0 - R1)
        MPYF    *AR1,R6,R0          ; R0 = BR * SIN , (R3 = AI + TI)
||      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++(IR0),R4   ; (R4 = AI - TI , BI' = R3)
||      STF     R3,*AR3++(IR0)
        SUBF    R0,R5,R3            ; R3 = TR = R5 - R0
        MPYF    *AR1++,R7,R0        ; R0 = BR * COS , R2 = AR - TR
||      SUBF    R3,*AR0,R2
        MPYF    *AR1++,R6,R1        ; R1 = BI * SIN , (AI' = R4)
||      STF     R4,*AR2++(IR0)
        ADDF    *AR0++,R3,R5        ; R5 = AR + TR , BR' = R2
||      STF     R2,*AR3++
*
        MPYF    *++AR1,R7,R5        ; R5 = BI * COS , (AR' = R5)
||      STF     R5,*AR2++
        ADDF    R1,R0,R2            ; (R2 = TI = R0 + R1)
        MPYF    *AR1,R6,R0          ; R0 = BR * SIN , (R3 = AI + TI)
||      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++,R4        ; (R4 = AI - TI , y(L) = BI' = R3)
||      STF     R3,*AR3++
        SUBF    R0,R5,R3            ; R3 = TR = R5 - R0
        MPYF    *AR1++,R7,R0        ; R0 = BR * COS , R2 = AR - TR
||      SUBF    R3,*AR0,R2
BF2END  MPYF    *AR1++(IR0),R6,R1   ; R1 = BI * SIN , R3 = AR + TR
||      ADDF    *AR0++,R3,R3
*
*       CLEAR PIPELINE

        STF     R2,*AR3++           ; BR' = R2 , AI' = R4
||      STF     R4,*AR2++
        ADDF    R1,R0,R2            ; R2 = TI = R0 + R1
        ADDF    R2,*AR0,R3          ; R3 = AI + TI , AR' = R3
||      STF     R3,*AR2++
        SUBF    R2,*AR0,R4          ; R4 = AI - TI , BI' = R3
||      STF     R3,*AR3
        STF     R4,*AR2             ; AI' = R4
*
*       LAST STAGE
*
        LDI     @INPUT,AR0          ; UPPER INPUT
        LDI     AR0,AR2             ; UPPER OUTPUT
        LDI     @INPUTP2,AR1        ; LOWER INPUT
        LDI     AR1,AR3             ; LOWER OUTPUT
        LDI     @SINTP2,AR7         ; POINTER TO TWIDDLE FACTORS
        LDI     3,IR0               ; GROUP OFFSET
        LDI     @FG4M2,RC
*
*       FILL PIPELINE
*
*       1. BUTTERFLY: w^0
*
        DDF     *AR0,*AR1,R6        ; AR' = R6 = AR + BR
        SUBF    *AR1++,*AR0++,R7    ; BR' = R7 = AR - BR
        ADDF    *AR0,*AR1,R4        ; AI' = R4 = AI + BI
        SUBF    *AR1++(IR0),*AR0++(IR0),R5   ; BI' = R5 = AI - BI
*
*       2. BUTTERFLY: w^N/4
*
        ADDF    *++AR1,*AR0,R3      ; AR' = R3 = AR + BI
        LDF     *-AR7,R1            ; R1 = 0 (FOR INNER LOOP)
||      LDF     *AR1++,R0           ; R0 = BR (FOR INNER LOOP)
        SUBF    *AR1++(IR0),*AR0++,R2   ; BR' = R2 = AR - BI
        STF     R6,*AR2++           ; (AR' = R6)
||      STF     R7,*AR3++           ; (BR' = R7)
        STF     R5,*AR3++(IR0)      ; (BI' = R5)
*
*       3. TO M. BUTTERFLY:
*
        LDF     *AR7++,R7           ; R7 = COS , (AI' = R4)
||      STF     R4,*AR2++(IR0)
        LDF     *AR7++,R6           ; R6 = SIN , (BR' = R2)
||      STF     R2,*AR3++
        MPYF    *++AR1,R6,R5        ; R5 = BI * SIN , (AR' = R3)
||      STF     R3,*AR2++
        ADDF    R1,R0,R2            ; (R2 = TI = R0 + R1)
        MPYF    *AR1,R7,R0          ; R0 = BR * COS , (R3 = AI + TI)
```

```
||      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++(IR0),R4   ; (R4 = AI - TI , BI' = R3)
||      STF     R3,*AR3++(IR0)
        ADDF    R0,R5,R3            ; R3 = TR = R0 + R5
        MPYF    *AR1++,R6,R0        ; R0 = BR * SIN , R2 = AR - TR
||      SUBF    R3,*AR0,R2
        MPYF    *AR1++(IR0),R7,R1   ; R1 = BI * COS , (AI' = R4)
||      STF     R4,*AR2++(IR0)
        ADDF    *AR0++,R3,R3        ; R3 = AR + TR , BR' = R2
||      STF     R2,*AR3++
*
        MPYF    *+AR1,R7,R5         ; R5 = BI * COS , (AR' = R3)
||      STF     R3,*AR2++
        SUBF    R1,R0,R2            ; (R2 = TI = R0 - R1)
        MPYF    *AR1,R6,R0          ; R0 = BR * SIN , (R3 = AI + TI)
||      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++(IR0),R4   ; (R4 = AI - TI , BI' = R3)
||      STF     R3,*AR3++(IR0)
        SUBF    R0,R5,R3            ; R3 = TR = R0 - R5
        MPYF    *AR1++,R7,R0        ; R0 = BR * COS , R2 = AR - TR
||      SUBF    R3,*AR0,R2
BFLEND  MPYF    *AR1++(IR0),R6,R1   ; R1 = BI * SIN , R3 = AR + TR
||      ADDF    *AR0++,R3,R3
*
*   CLEAR PIPELINE
*
        STF     R2,*AR3++           ; BR' = R2 , (AI' = R4)
||      STF     R4,*AR2++(IR0)
        ADDF    R1,R0,R2            ; R2 = TI = R0 + R1
        ADDF    R2,*AR0,R3          ; R3 = AI + TI , AR' = R3
||      STF     R3,*AR2++
        SUBF    R2,*AR0,R4          ; R4 = AI - TI , BI' = R3
||      STF     R3,*AR3
        STF     R4,*AR2             ; AI' = R4
*
*   END OF FFT
*
*   BIT REVERSAL
*
        LDI     @FFTSIZ,IR0
        LDI     2,IR1
        LDI     @INPUT,AR0
        LDI     @OUTPUT,AR1
        LDI     @FFTSIZ,RC
        SUBI    2,RC
*
        LDF     *+AR0(1),R0
        RPTB    BITRV
        LDF     *AR0++(IR0)B,R1
||      STF     R0,*+AR1(1)
BITRV   LDF     *+AR0(1),R0
||      STF     R1,*AR1++(IR1)
        LDF     *AR0++(IR0)B,R1
||      STF     R0,*+AR1(1)
        STF     R1,*AR1
*
END:    NOP
        NOP
        NOP
        NOP
*
SELF    BR      SELF
        .end
*
```

Appendix A4. Complex, Radix-2 DIT FFT – R2DITB.ASM

```
APPENDIX A4

            COMPLEX, RADIX-2 DIT FFT  :  R2DITB.ASM
         -------------------------------------------

GENERIC PROGRAM FOR A FAST LOOPED-CODE RADIX-2 DIT FFT COMPUTATION
                  ON THE TMS320C30

WRITTEN BY: RAIMUND MEYER, KARL SCHWARZ               24.07.89
            LEHRSTUHL FUER NACHRICHTENTECHNIK
            UNIVERSITAET ERLANGEN-NUERNBERG
            CAUERSTRASSE 7, D-8520 ERLANGEN, FRG

THE (COMPLEX) DATA RESIDE IN INTERNAL MEMORY. THE COMPUTATION IS DONE
IN-PLACE, BUT THE RESULT IS MOVED TO ANOTHER MEMORY SECTION TO
DEMONSTRATE THE BIT-REVERSED ADDRESSING.

FOR THIS PROGRAM THE MINIMUM FFT LENGTH IS 32 POINTS BECAUSE OF
THE SEPARATE STAGES.

FIRST TWO PASSES ARE REALIZED AS A FOUR BUTTERFLY LOOP SINCE THE
MULTIPLIES ARE TRIVIAL. THE MULTIPLIER IS ONLY USED FOR A LOAD IN
PARALLEL WITH AN ADDF OR SUBF.
```

```
EXAMPLE FOR A 1024-POINT FFT (WITH BIT REVERSAL) :

MEMORY SIZE :
    PROG                             =  231 WORDS
    DATA                             =  512 WORDS

CYCLES PER BUTTERFLY :
    STAGES 1 AND 2                   =    4
    STAGES 3 TO  8                   =    8
    STAGE 9                          =    8.25
    STAGE 10                         =   10.5 (DUE TO EXT. MEMORY WAITS)

AVERAGE CYCLES/BUTTERFLY             =    7.475
TOTAL BUTTERFLYCYCLES                = 38272
INITIALIZATION OVERHEAD              =  2185  =  5.4 % OF TOTAL TIME
TOTAL NUMBER OF INSTRUCTION CYCLES = 40457
TOTAL TIME FOR A 1024 POINT FFT      =    2.42 ms (INCLUDING BIT
                                                          REVERSAL)
```

```
THIS PROGRAM INCUDES FOLLOWING FILES:
-------------------------------------

THE FILE 'TWID1KBR.ASM' CONSISTS OF TWIDDLE FACTORS

THE TWIDDLE FACTORS ARE STORED IN BIT REVERSED ORDER AND WITH A TABLE
LENGTH OF N/2 (N = FFTLENGTH).

EXAMPLE: SHOWN FOR N=32, WN(n) = COS(2*PI*n/N) - j*SIN(2*PI*n/N)

ADDRESS   COEFFICIENT
 0          R(WN(0)) = COS(2*PI*0/32) = 1
 1         -I(WN(0)) = SIN(2*PI*0/32) = 0
 2          R(WN(4)) = COS(2*PI*4/32) = 0.707
 3         -I(WN(4)) = SIN(2*PI*4/32) = 0.707
 .                              .
 .                              .
12          R(WN(3)) = COS(2*PI*3/32) = 0.831
13         -I(WN(3)) = SIN(2*PI*3/32) = 0.556
14          R(WN(7)) = COS(2*PI*7/32) = 0.195
15         -I(WN(7)) = SIN(2*PI*7/32) = 0.981

WHEN GENERATED FOR A FFT LENGTH OF 1024, THE TABLE IS FOR ALL
AVAILABLE FFT OF LESS OR EQUAL LENGTH.

THE MISSING TWIDDLE FACTORS (WN(),WN(),....) ARE GENERATED BY USING
THE SYMMETRY WN(N/4+n) = -j*WN(n). THIS CAN BE EASILY REALIZED, BY
CHANGING REAL- AND IMAGINARY PART OF THE TWIDDLE FACTORS AND BY
NEGATING THE NEW REAL PART.

TO CHANGE THE FFT LENGTH ONLY THE PARAMETERS IN THE HEADER OF
TWID1KBR.ASM AND THE INPUT AND OUTPUT VECTOR LENGTHS NEED TO BE
ALTERED.
```

```
                                          +
AR + j AI ------------------------------------------------ AR' + j AI'
                                   \        / +
                                    \      /
                                     \    /
                                      /  \
                                     /    \
                                    /      \ +
BR + j BI ---- ( COS - j SIN ) --------------------------- BR' + j BI'
                                          -

TR = BR * COS + BI * SIN
TI = BR * SIN - BI * COS
AR'= AR + TR
AI'= AI - TI
BR'= AR - TR
BI'= AI + TI
```

```
*
        .global  FFT
        .global  N
        .global  NHALB
        .global  NVIERT
        .global  NACHTEL
        .global  M
        .global  SINE

        .bss     INP,2048            ; INPUT VECTOR LENGTH = 2N (DEPENDS
*                                    ;   ON N)
        .bss     OUTP,2048           ; OUTPUT VECTOR LENGTH = 2N (DEPENDS
*                                    ;   ON N)
*
        .text
*
FFTSIZ  .word    N
FG4M2   .word    NVIERT-2
FG4M3   .word    NVIERT-3
FG8M2   .word    NACHTEL-2
FG2     .word    NHALB
FG2M3   .word    NHALB-3
LOGFFT  .word    M
SINTAB  .word    SINE
SINTM1  .word    SINE-1
SINTP2  .word    SINE+2
INPUT   .word    INP
INPUTP2 .word    INP+2
OUTPUT  .word    OUTP
OUTP1   .word    OUTP+1
*
*    AR0 : AR + AI
*    AR1 : BR + BI
*    AR2 : CR + CI + CR' + CI'
*    AR3 : DR + DI
*    AR4 : AR' + AI'
*    AR5 : BR' + BI'
*    AR6 : DR' + DI'
*    AR7 : FIRST TWIDDLE FACTOR = 1
*
FFT:    LDP      FFTSIZ              ; LOAD PAGE POINTER
        LDI      @FG2,IR0            ; IR0 = N/2 = OFFSET BETWEEN INPUTS
        LDI      @SINTAB,AR7         ; AR7 POINTS TO TWIDDLE FACTOR 1
        LDI      @INPUT,AR0          ; AR0 POINTS TO AR
        ADDI     IR0,AR0,AR1         ; AR1 POINTS TO BR
        ADDI     IR0,AR1,AR2         ; AR2 POINTS TO CR
        ADDI     IR0,AR2,AR3         ; AR3 POINTS TO DR
        LDI      AR0,AR4             ; AR4 POINTS TO AR'
        LDI      AR1,AR5             ; AR5 POINTS TO BR'
        LDI      AR3,AR6             ; AR6 POINTS TO DR'
        LDI      2,IR1               ; ADDRESS OFFSET
        LSH      -1,IR0              ; IR0 = N/4 = NUMBER OF R4-BUTTERFLIES
        LDI      IR0,RC
        SUBI     2,RC
*
*   FIRST 2 STAGES AS RADIX-4 BUTTERFLY
*
*   FILL PIPELINE
*
        ADDF     *AR2,*AR0,R4        ; R4 = AR + CR
        SUBF     *AR2,*AR0++,R5      ; R5 = AR - CR
        ADDF     *AR1,*AR3,R6        ; R6 = DR + BR
        SUBF     *AR1++,*AR3++,R7    ; R7 = DR - BR
        ADDF     R6,R4,R0            ; AR' = R0 = R4 + R6
        MPYF     *AR3++,*AR7,R1      ; R1 = DI , BR' = R3 = R4 - R6
||      SUBF     R6,R4,R3
        ADDF     R1,*AR1,R0          ; R0 = BI + DI , AR' = R0
||      STF      R0,*AR4++
        SUBF     R1,*AR1++,R1        ; R1 = BI - DI , BR' = R3
||      STF      R3,*AR5++
        ADDF     R1,R5,R2            ; CR' = R2 = R5 + R1
        MPYF     *+AR2,*AR7,R1       ; R1 = CI , DR' = R3 = R5 - R1
||      SUBF     R1,R5,R3
        ADDF     R1,*AR0,R2          ; R2 = AI + CI , CR' = R2
||      STF      R2,*AR2++(IR1)
        SUBF     R1,*AR0++,R6        ; R6 = AI - CI , DR' = R3
||      STF      R3,*AR6++
        ADDF     R0,R2,R4            ; AI' = R4 = R2 + R0
*
*   RADIX-4 BUTTERFLY LOOP
*
        RPTB     BLK1
        MPYF     *AR2--,*AR7,R0      ; R0 = CR , (BI' = R2 = R2 - R0)
||      SUBF     R0,R2,R2
        MPYF     *AR1++,*AR7,R1      ; R1 = BR , (CI' = R3 = R6 + R7)
||      ADDF     R7,R6,R3
        ADDF     R0,*AR0,R4          ; R4 = AR + CR , (AI' = R4)
||      STF      R4,*AR4++
        SUBF     R0,*AR0++,R5        ; R5 = AR - CR , (BI' = R2)
||      STF      R2,*AR5++
        SUBF     R7,R6,R7            ; (DI' = R7 = R6 - R7)
        ADDF     R1,*AR3,R6          ; R6 = DR + BR , (DI' = R7)
||      STF      R7,*AR6++
        SUBF     R1,*AR3++,R7        ; R7 = DR - BR , (CI' = R3)
||      STF      R3,*AR2++
        ADDF     R6,R4,R0            ; AR' = R0 = R4 + R6
        MPYF     *AR3++,*AR7,R1      ; R1 = DI , BR' = R3 = R4 - R6
||      SUBF     R6,R4,R3
        ADDF     R1,*AR1,R0          ; R0 = BI + DI , AR' = R0
||      STF      R0,*AR4++
        SUBF     R1,*AR1++,R1        ; R1 = BI - DI , BR' = R3
||      STF      R3,*AR5++
        ADDF     R1,R5,R2            ; CR' = R2 = R5 + R1
        MPYF     *+AR2,*AR7,R1       ; R1 = CI , DR' = R3 = R5 - R1
||      SUBF     R1,R5,R3
        ADDF     R1,*AR0,R2          ; R2 = AI + CI , CR' = R2
||      STF      R2,*AR2++(IR1)
        SUBF     R1,*AR0++,R6        ; R6 = AI - CI , DR' = R3
```

```
::      STF     R3,*AR6++
BLK1    ADDF    R0,R2,R4            ; AI' = R4 = R2 + R0
*
*   CLEAR PIPELINE

        SUBF    R0,R2,R2            ; BI' = R2 = R2 - R0
        ADDF    R7,R6,R3            ; CI' = R3 = R6 + R7
        STF     R4,*AR4             ; AI' = R4 , BI' = R2
::      STF     R2,*AR5
        SUBF    R7,R6,R7            ; DI' = R7 = R6 - R7
        STF     R7,*AR6             ; DI' = R7 , CI' = R3
::      STF     R3,*--AR2
*
*   THIRD TO LAST-2 STAGE
*
        LDI     @FG2,IR1
        LDI     IR0,AR5
        SUBI    1,AR5
        LDI     1,AR6
*
STUFE   LDI     @SINTAB,AR7         ; POINTER TO TWIDDLE FACTOR
        LDI     0,AR4               ; GROUP COUNTER
        LDI     @INPUT,AR0          ; UPPER REAL BUTTERFLY INPUT
        LDI     AR0,AR2             ; UPPER REAL BUTTERFLY OUTPUT
        ADDI    IR0,AR0,AR3         ; LOWER REAL BUTTERFLY OUTPUT
        LDI     AR3,AR1             ; LOWER REAL BUTTERFLY INPUT
        LSH     1,AR6               ; DOUBLE GROUP COUNT
        LSH     -2,AR5              ; HALF BUTTERFLY COUNT
        LSH     1,AR5               ; CLEAR LSB
        LSH     -1,IR0              ; HALF STEP FROM UPPER TO LOWER REAL
*                                   ;   PART
        LSH     -1,IR1
        ADDI    1,IR1               ; STEP FROM OLD IMAGINARY TO NEW REAL
*                                   ;   VALUE
        LDF     *AR1++,R6           ; DUMMY LOAD, ONLY FOR ADDRESS UPDATE
::      LDF     *AR7,R7             ; R7 = COS
*
GRUPPE
*
*   FILL PIPELINE                   ; AR0 = UPPER REAL BUTTERFLY INPUT
*                                   ; AR1 = LOWER REAL BUTTERFLY INPUT
*                                   ; AR2 = UPPER REAL BUTTERFLY OUTPUT
*                                   ; AR3 = LOWER REAL BUTTERFLY OUTPUT
*                                   ; THE IMAGINARY PART HAS TO FOLLOW
        LDF     *++AR7,R6           ; R6 = SIN
        MPYF    *AR1--,R6,R1        ; R1 = BI * SIN
::      ADDF    *++AR4,R0,R3        ; DUMMY ADDF FOR COUNTER UPDATE
        MPYF    *AR1,R7,R0          ; R0 = BR * COS
        MPYF    *AR1++,*AR7--,R0    ; R3 = TR = R0 + R1 , R0 = BR * SIN
::      ADDF    R0,R1,R3
        MPYF    *AR1++,R7,R1        ; R1 = BI * COS , R2 = AR - TR
::      SUBF    R3,*AR0,R2
        ADDF    *AR0++,R3,R5        ; R5 = AR + TR , BR' = R2
::      STF     R2,*AR3++

        LDI     AR5,RC
*
*   FIRST BUTTERFLY-TYPE:
*
*   TR = BR * COS + BI * SIN
*   TI = BR * SIN - BI * COS
*   AR'= AR + TR
*   AI'= AI - TI
*   BR'= AR - TR
*   BI'= AI + TI*
*
        RPTB    BFLY1
*
        MPYF    *+AR1,R6,R5         ; R5 = BI * SIN , (AR' = R5)
::      STF     R5,*AR2++
        SUBF    R1,R0,R2            ; (R2 = TI = R0 - R1)
        MPYF    *AR1,R7,R0          ; R0 = BR * COS , (R3 = AI + TI)
::      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++,R4        ; (R4 = AI - TI , BI' = R3)
::      STF     R3,*AR3++
        ADDF    R0,R5,R3            ; R3 = TR = R0 + R5
        MPYF    *AR1++,R6,R0        ; R0 = BR * SIN , R2 = AR - TR
::      SUBF    R3,*AR0,R2
        MPYF    *AR1++,R7,R1        ; R1 = BI * COS , (AI' = R4)
::      STF     R4,*AR2++
BFLY1   ADDF    *AR0++,R3,R5        ; R5 = AR + TR , BR' = R2
::      STF     R2,*AR3++
*
*   SWITCH OVER TO NEXT GROUP
*
        SUBF    R1,R0,R2            ; R2 = TI = R0 - R1
        ADDF    R2,*AR0,R3          ; R3 = AI + TI , AR' = R5
::      STF     R5,*AR2++
        SUBF    R2,*AR0++(IR1),R4   ; R4 = AI - TI , BI' = R3
::      STF     R3,*AR3++(IR1)
        NOP     *AR1++(IR1)         ; ADDRESS UPDATE
        MPYF    *AR1--,R7,R1        ; R1 = BI * COS , AI' = R4
::      STF     R4,*AR2++(IR1)
        MPYF    *AR1,R6,R0          ; R0 = BR * SIN
        MPYF    *AR1++,*AR7++,R0    ; R3 = TR = R1 - R0 , R0 = BR * COS
::      SUBF    R0,R1,R3
        MPYF    *AR1++,R6,R1        ; R1 = BI * SIN , R2 = AR - TR
::      SUBF    R3,*AR0,R2
        ADDF    *AR0++,R3,R5        ; R5 = AR + TR , BR' = R2
::      STF     R2,*AR3++
        LDI     AR5,RC
*
*   SECOND BUTTERFLY-TYPE:
*
*   TR = BI * COS - BR * SIN
*   TI = BI * SIN + BR * COS
*   AR'= AR + TR
*   AI'= AI - TI
*   BR'= AR - TR
```

```
*    BI'= AI + TI
*
        RPTB    BFLY2

        MPYF    *++AR1,R7,R5            ; R5 = BI * COS , (AR' = R5)
::      STF     R5,*AR2++
        ADDF    R1,R0,R2                ; (R2 = TI = R0 + R1)
        MPYF    *AR1,R6,R0              ; R0 = BR * SIN , (R3 = AI + TI)
::      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++,R4            ; (R4 = AI - TI , BI' = R3)
::      STF     R3,*AR3++
        SUBF    R0,R5,R3                ; TR = R3 = R5 - R0
        MPYF    *AR1++,R7,R0            ; R0 = BR * COS , R2 = AR - TR
::      SUBF    R3,*AR0,R2
        MPYF    *AR1++,R6,R1            ; R1 = BI * SIN , (AI' = R4)
::      STF     R4,*AR2++
BFLY2   ADDF    *AR0++,R3,R5            ; R5 = AR + TR , BR' = R2
::      STF     R2,*AR3++
*
*    CLEAR PIPELINE
*
        ADDF    R1,R0,R2                ; R2 = TI = R0 + R1
        ADDF    R2,*AR0,R3              ; R3 = AI + TI
::      STF     R5,*AR2++               ; AR' = R5
        CMPI    AR6,AR4
        BNED    GRUPPE                  ; DO FOLLOWING 3 INSTRUCTIONS
        SUBF    R2,*AR0++(IR1),R4       ; R4 = AI - TI , BI' = R3
::      STF     R3,*AR3++(IR1)
        LDF     *++AR7,R7               ; R7 = COS
::      STF     R4,*AR2++(IR1)          ; AI' = R4
        NOP     *AR1++(IR1)             ; BRANCH HERE
*
*    END OF THIS BUTTERFLY GROUP
*
        CMPI    4,IR0                   ; JUMP OUT AFTER LD(N)-3 STAGE
        BNZ     STUFE
*
*    SECOND TO LAST STAGE
*
        LDI     @INPUT,AR0              ; UPPER INPUT
        LDI     AR0,AR2                 ; UPPER OUTPUT
        ADDI    IR0,AR0,AR1             ; LOWER INPUT
        LDI     AR1,AR3                 ; LOWER OUTPUT
        LDI     @SINTP2,AR7             ; POINTER TO TWIDDLE FACTOR
        LDI     5,IR0                   ; DISTANCE BETWEEN TWO GROUPS
        LDI     @FG8M2,RC
*
*    FILL PIPELINE
*
*    1. BUTTERFLY: w^0
*
        ADDF    *AR0,*AR1,R2            ; AR' = R2 = AR + BR
        SUBF    *AR1++,*AR0++,R3        ; BR' = R3 = AR - BR
        ADDF    *AR0,*AR1,R0            ; AI' = R0 = AI + BI
        SUBF    *AR1++,*AR0++,R1        ; BI' = R1 = AI - BI
*
*    2. BUTTERFLY: w^0
*
        ADDF    *AR0,*AR1,R6            ; AR' = R6 = AR + BR
        SUBF    *AR1++,*AR0++,R7        ; BR' = R7 = AR - BR
        ADDF    *AR0,*AR1,R4            ; AI' = R4 = AI + BI
        SUBF    *AR1++(IR0),*AR0++(IR0),R5   ; BI' = R5 = AI - BI
        STF     R2,*AR2++               ; (AR' = R2)
::      STF     R3,*AR3++               ; (BR' = R3)
        STF     R0,*AR2++               ; (AI' = R0)
::      STF     R1,*AR3++               ; (BI' = R1)
        STF     R6,*AR2++               ; AR' = R6
::      STF     R7,*AR3++               ; BR' = R7
        STF     R4,*AR2++(IR0)          ; AI' = R4
::      STF     R5,*AR3++(IR0)          ; BI' = R5
*
*    3. BUTTERFLY: w^M/4
*
        ADDF    *AR0++,*+AR1,R5         ; AR' = R5 = AR + BI
        SUBF    *AR1,*AR0,R4            ; AI' = R4 = AI - BR
        ADDF    *AR1++,*AR0--,R6        ; BI' = R6 = AI + BR
        SUBF    *AR1++,*AR0++,R7        ; BR' = R7 = AR - BI
*
*    4. BUTTERFLY: w^M/4
*
        ADDF    *+AR1,*++AR0,R3         ; AR' = R3 = AR + BI
        LDF     *-AR7,R1                ; R1 = 0 (FOR INNER LOOP)
::      LDF     *AR1++,R0               ; R0 = BR (FOR INNER LOOP)
        SUBF    *AR1++(IR0),*AR0++,R2   ; BR' = R2 = AR - BI
        STF     R5,*AR2++               ; (AR' = R5)
::      STF     R7,*AR3++               ; (BR' = R7)
        STF     R6,*AR3++               ; (BI' = R6)
*
*    5. TO M. BUTTERFLY:
*
        RPTB    BF2END
*
        LDF     *AR7++,R7               ; R7 = COS , ((AI' = R4))
::      STF     R4,*AR2++
        LDF     *AR7++,R6               ; R6 = SIN , (BR' = R2)
::      STF     R2,*AR3++
        MPYF    *++AR1,R6,R5            ; R5 = BI * SIN , (AR' = R3)
::      STF     R3,*AR2++
        ADDF    R1,R0,R2                ; (R2 = TI = R0 + R1)
        MPYF    *AR1,R7,R0              ; R0 = BR * COS , (R3 = AI + TI)
::      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++(IR0),R4       ; (R4 = AI - TI , BI' = R3)
::      STF     R3,*AR3++(IR0)
        ADDF    R0,R5,R3                ; R3 = TR = R0 + R5
        MPYF    *AR1++,R6,R0            ; R0 = BR * SIN , R2 = AR - TR
::      SUBF    R3,*AR0,R2
        MPYF    *AR1++,R7,R1            ; R1 = BI * COS , (AI' = R4)
::      STF     R4,*AR2++(IR0)
```

```
        ADDF    *AR0++,R3,R5        ; R5 = AR + TR , BR' = R2
||      STF     R2,*AR3++
*
        MPYF    *++AR1,R6,R5        ; R5 = BI * SIN , (AR' = R5)
||      STF     R5,*AR2++
        SUBF    R1,R0,R2            ; (R2 = TI = R0 - R1)
        MPYF    *AR1,R7,R0          ; R0 = BR * COS , (R3 = AI + TI)
||      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++,R4        ; (R4 = AI - TI , BI' = R3)
||      STF     R3,*AR3++
        ADDF    R0,R5,R3            ; R3 = TR = R0 + R5
        MPYF    *AR1++,R6,R0        ; R0 = BR * SIN , R2 = AR - TR
||      SUBF    R3,*AR0,R2
        MPYF    *AR1++(IR0),R7,R1   ; R1 = BI * COS , (AI' = R4)
||      STF     R4,*AR2++
        ADDF    *AR0++,R3,R3        ; R3 = AR + TR , BR' = R2
||      STF     R2,*AR3++
*
        MPYF    *++AR1,R7,R5        ; R5 = BI * COS , (AR' = R3)
||      STF     R3,*AR2++
        SUBF    R1,R0,R2            ; (R2 = TI = R0 - R1)
        MPYF    *AR1,R6,R0          ; R0 = BR * SIN , (R3 = AI + TI)
||      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++(IR0),R4   ; (R4 = AI - TI , BI' = R3)
||      STF     R3,*AR3++(IR0)
        SUBF    R0,R5,R3            ; R3 = TR = R5 - R0
        MPYF    *AR1++,R7,R0        ; R0 = BR * COS , R2 = AR - TR
||      SUBF    R3,*AR0,R2
        MPYF    *AR1++,R6,R1        ; R1 = BI * SIN , (AI' = R4)
||      STF     R4,*AR2++(IR0)
        ADDF    *AR0++,R3,R5        ; R5 = AR + TR , BR' = R2
||      STF     R2,*AR3++
*
        MPYF    *++AR1,R7,R5        ; R5 = BI * COS , (AR' = R5)
||      STF     R5,*AR2++
        ADDF    R1,R0,R2            ; (R2 = TI = R0 + R1)
        MPYF    *AR1,R6,R0          ; R0 = BR * SIN , (R3 = AI + TI)
||      ADDF    R2,*AR0,R3
        SUBF    R2,*AR0++,R4        ; (R4 = AI - TI , y(L) = BI' = R3)
||      STF     R3,*AR3++
        SUBF    R0,R5,R3            ; R3 = TR = R5 - R0
        MPYF    *AR1++,R7,R0        ; R0 = BR * COS , R2 = AR - TR
||      SUBF    R3,*AR0,R2
BF2END  MPYF    *AR1++(IR0),R6,R1   ; R1 = BI * SIN , R3 = AR + TR
||      ADDF    *AR0++,R3,R3
*
*   CLEAR PIPELINE
*
        STF     R2,*AR3++           ; BR' = R2 , AI' = R4
||      STF     R4,*AR2++
        ADDF    R1,R0,R2            ; R2 = TI = R0 + R1
        ADDF    R2,*AR0,R3          ; R3 = AI + TI , AR' = R3
||      STF     R3,*AR2++
        SUBF    R2,*AR0,R4          ; R4 = AI - TI , BI' = R3
||      STF     R3,*AR3
        STF     R4,*AR2             ; AI' = R4
*
*   LAST STAGE WITH INTEGRATED BIT REVERSAL
*
        LDI     @INPUT,AR0          ; UPPER INPUT
        LDI     @OUTPUT,AR2         ; REAL OUTPUT !!!
        LDI     @INPUTP2,AR1        ; LOWER INPUT
        LDI     @OUTP1,AR3          ; IMAGINARY OUTPUT !!!
        LDI     @SINTP2,AR7         ; POINTER TO TWIDDLE FACTORS
        LDI     @FFTSIZ,IR0         ; BIT REVERSAL
        LDI     3,IR1               ; GROUP OFFSET
        LDI     @FG4M2,RC
*
*   FILL PIPELINE
*
*   1. BUTTERFLY: w^0
*
        ADDF    *AR0,*AR1,R6        ; AR' = R6 = AR + BR
        SUBF    *AR1++,*AR0++,R7    ; BR' = R7 = AR - BR
        SUBF    *AR1,*AR0,R4        ; BI' = R4 = AI - BI
        ADDF    *AR1++(IR1),*AR0++(IR1),R5   ; AI' = R5 = AI + BI
*
*   2. BUTTERFLY: w^M/4
*
        UBF     *++AR1,*AR0,R3      ; BR' = R3 = AR - BI
        LDF     *-AR7,R1            ; R1 = 0 (FOR INNER LOOP)
||      LDF     *AR1++,R0           ; R0 = BR (FOR INNER LOOP)
        ADDF    *AR1++(IR1),*AR0++,R2   ; AR' = R2 = AR + BI
        STF     R6,*AR2++(IR0)b     ; (AR' = R6)
||      STF     R5,*AR3++(IR0)b     ; (AI' = R5)
        STF     R7,*AR2++(IR0)b     ; (BR' = R7)
*
*   3. TO M. BUTTERFLY:
*
        PTB     BFLEND
*
*   17 CYCLES IF FFT SIZE <1024 DUE TO THE USE OF INTERNAL MEMORY FOR BIT
*   REVERSAL, 21 CYCLES IF FFT SIZE = 1024 DUE TO THE USE OF EXTERNAL MEMORY
*   FOR BIT REVERSAL
*
        LDF     *AR7++,R7           ; R7 = COS , ((BI' = R4))
||      STF     R4,*AR3++(IR0)B
        LDF     *AR7++,R6           ; R6 = SIN , (AR' = R2)
||      STF     R2,*AR2++(IR0)B
        MPYF    *++AR1,R6,R5        ; R5 = BI * SIN , (BR' = R3)
||      STF     R3,*AR2++(IR0)B
        ADDF    R1,R0,R2            ; (R2 = TI = R0 + R1)
        MPYF    *AR1,R7,R0          ; R0 = BR * COS , (AI' = R3 = AI - TI)
||      SUBF    R2,*AR0,R3
        ADDF    R2,*AR0++(IR1),R4   ; (BI' = R4 = AI + TI , AI' = R3)
||      STF     R3,*AR3++(IR0)B
        ADDF    R0,R5,R3            ; R3 = TR = R0 + R5
        MPYF    *AR1++,R6,R0        ; R0 = BR * SIN , AR' = R2 = AR + TR
```

```
||      ADDF    R3,*AR0,R2
        MPYF    *AR1++(IR1),R7,R1   ; R1 = BI * COS , (BI' = R4)
||      STF     R4,*AR3++(IR0)B
        SUBF    R3,*AR0++,R3        ; BR' = R3 = AR - TR , AR' = R2
||      STF     R2,*AR2++(IR0)B
*
        MPYF    *++AR1,R7,R5        ; R5 = BI * COS , (BR' = R3)
||      STF     R3,*AR2++(IR0)B
        SUBF    R1,R0,R2            ; (R2 = TI = R0 - R1)
        MPYF    *AR1,R6,R0          ; R0 = BR * SIN , (AI' = R3 = AI - TI)
||      SUBF    R2,*AR0,R3
        ADDF    R2,*AR0++(IR1),R4   ; (BI' = R4 = AI + TI , AI' = R3)
||      STF     R3,*AR3++(IR0)B
        SUBF    R0,R5,R3            ; R3 = TR = R0 - R5
        MPYF    *AR1++,R7,R0        ; R0 = BR * COS , AR' = R2 = AR + TR
||      ADDF    R3,*AR0,R2
BFLEND  MPYF    *AR1++(IR1),R6,R1   ; R1 = BI * SIN , BR' = R3 = AR - TR
||      SUBF    R3,*AR0++,R3
*
*   CLEAR PIPELINE
*
        STF     R2,*AR2++(IR0)B     ; AR' = R2 , (BI' = R4)
||      STF     R4,*AR3++(IR0)B
        ADDF    R1,R0,R2            ; R2 = TI = R0 + R1
        SUBF    R2,*AR0,R3          ; AI' = R3 = AI - TI , BR' = R3
||      STF     R3,*AR2
        ADDF    R2,*AR0,R4          ; BI' = R4 = AI + TI , AI' = R3
||      STF     R3,*AR3++(IR0)B
        STF     R4,*AR3             ; BI' = R4
*
*   END OF FFT
*
END:    NOP
        NOP
        NOP
        NOP
*
SELF    BR      SELF
        .end
*
```

Appendix A5. TWID1KBR.ASM—Table With Twiddle Factors for a FFT up to a Length of 1024 Complex Points.

```
********************************************************************************
*                                                                              *
*     APPENDIX A5                                                              *
*                                                                              *
*     TITLE: TWID1KBR.ASM                                                      *
*                                                                              *
*     TABLE WITH TWIDDLE FACTORS FOR A FFT UP TO A LENGTH OF 1024 COMPLEX      *
*     POINTS.                                                                  *
*                                                                              *
*     FILE TO BE LINKED WITH THE SOURCE CODE : R2DIT.ASM OR R2DITB.ASM         *
*                                                                              *
*     WRITTEN BY : RAIMUND MEYER AND KARL SCHWARZ          14.07.89            *
*                  LEHRSTUHL FUER NACHRICHTENTECHNIK                           *
*                  UNIVERSITAET ERLANGEN-NUERNBERG                             *
*                                                                              *
*     LENGTH OF TWIDDLE FACTOR TABLE : 512 REAL VALUES (=1024 FFT)             *
*                                                                              *
********************************************************************************
*
          .global   sine
          .global   n
          .global   nhalb
          .global   nviert
          .global   nachtel
          .global   m
*
n         .set      1024                ; FFT-LENGTH n
nhalb     .set      512                 ; n/2
nviert    .set      256                 ; n/4
nachtel   .set      128                 ; n/8
m         .set      10                  ; NUMBER OF STAGES = ld(n)
*
* ANOTHER EXAMPLE OF FFT-LENGTH n = 32:
* ONLY THE FIRST 16 VALUES OF THE TABLE ARE NEEDED
*
*n        .set      2
*nhalb    .set      16
*nviert   .set      8
*nachtel  .set      4
*m        .set      5
*
          .data
*
sine
          .float    1.00000000000000e+000
          .float    0.00000000000000e+000
          .float    7.07106781186548e-001
          .float    7.07106781186548e-001
          .float    9.23879532511287e-001
          .float    3.82683432365090e-001
          .float    3.82683432365090e-001
          .float    9.23879532511287e-001
          .float    9.80785280403230e-001
```

```
          .
          .
          .float    7.11432195745216e-001
          .float    7.02754744457225e-001
          .float    6.13588464915452e-003
          .float    9.99981175282601e-001
```

Appendix B. Radix-4 Complex FFT

Appendix B1. Generic Program to Do a Looped-Code Radix-4 FFT on the TMS320C30

```
*****************************************************************************
*
*    APPENDIX B1
*
*   GENERIC PROGRAM TO DO A LOOPED-CODE RADIX-4 FFT COMPUTATION ON THE
*   TMS320C30.
*
*   THE PROGRAM IS TAKEN FROM THE BURRUS AND PARKS BOOK, P. 117. THE COMPLEX
*   DATA RESIDE IN INTERNAL MEMORY, AND THE COMPUTATION IS DONE IN-PLACE.
*
*   THE TWIDDLE FACTORS ARE SUPPLIED IN A TABLE PUT IN A .DATA SECTION. THIS
*   DATA IS INCLUDED IN A SEPARATE FILE TO PRESERVE THE GENERIC NATURE OF THE
*   PROGRAM. FOR THE SAME PURPOSE, THE SIZE OF THE FFT N AND LOG4(N) ARE
*   DEFINED IN A .GLOBL DIRECTIVE AND SPECIFIED DURING LINKING.
*
*   IN ORDER TO HAVE THE FINAL RESULT IN BIT-REVERSED ORDER, THE TWO MIDDLE
*   BRANCHES OF THE RADIX-4 BUTTERFLY ARE INTERCHANGED DURING STORAGE. NOTE
*   THIS DIFFERENCE WHEN COMPARING WITH THE PROGRAM IN P. 117 OF THE BURRUS
*   AND PARKS BOOK.  ,
*
*   AUTHOR: PANOS E. PAPAMICHALIS
*           TEXAS INSTRUMENTS                               AUGUST 23, 1987
*
*****************************************************************************
*
        .GLOBL  FFT             ; ENTRY POINT FOR EXECUTION
        .GLOBL  N               ; FFT SIZE
        .GLOBL  M               ; LOG4(N)
        .GLOBL  SINE            ; ADDRESS OF SINE TABLE
*
INP     .USECT  "IN",1024       ; MEMORY WITH INPUT DATA
*
        .TEXT
*
*   INITIALIZE
*
        .WORD   FFT             ; STARTING LOCATION OF THE PROGRAM
*
        .SPACE  100             ; RESERVE 100 WORDS FOR VECTORS, ETC.
*
TEMP    .WORD   $+2
STORE   .WORD   FFTSIZ          ; BEGINNING OF TEMP STORAGE AREA
        .WORD   N
        .WORD   M
        .WORD   SINE
        .WORD   INP
*
        .BSS    FFTSIZ,1        ; FFT SIZE
        .BSS    LOGFFT,1        ; LOG4(FFTSIZ)
        .BSS    SINTAB,1        ; SINE/COSINE TABLE BASE
        .BSS    INPUT,1         ; AREA WITH INPUT DATA TO PROCESS
        .BSS    STAGE,1         ; FFT STAGE #
        .BSS    RPTCNT,1        ; REPEAT COUNTER
        .BSS    IEINDX,1        ; IE INDEX FOR SINE/COSINE
        .BSS    LPCNT,1         ; SECOND-LOOP COUNT
        .BSS    JT,1            ; JT COUNTER IN PROGRAM, P. 117
        .BSS    IA1,1           ; IA1 INDEX IN PROGRAM, P. 117
*
FFT:
*                               ; INITIALIZE DATA LOCATIONS
        LDP     TEMP            ; COMMAND TO LOAD DATA PAGE POINTER
        LDI     @TEMP,AR0
        LDI     @STORE,AR1
        LDI     *AR0++,R0       ; XFER DATA FROM ONE MEMORY TO THE
*                               ;   OTHER
        STI     R0,*AR1++
        LDI     *AR0++,R0
        STI     R0,*AR1++
        LDI     *AR0++,R0
        STI     R0,*AR1++
        LDI     *AR0,R0
        STI     R0,*AR1
*
        LDP     FFTSIZ          ; COMMAND TO LOAD DATA PAGE POINTER
        LDI     @FFTSIZ,R0
        LDI     @FFTSIZ,IR0
        LDI     @FFTSIZ,IR1
        LDI     0,AR7
        STI     AR7,@STAGE      ; @STAGE HOLDS THE CURRENT STAGE
*                               ;   NUMBER
        LSH     1,IR0           ; IR0=2*N1 (BECAUSE OF REAL/IMAG)
        LSH     -2,IR1          ; IR1=N/4, POINTER FOR SIN/COS TABLE
        LDI     1,AR7
        STI     AR7,@RPTCNT     ; INITIALIZE REPEAT COUNTER OF FIRST
*                               ;   LOOP
        LSH     -2,R0
        STI     AR7,@IEINDX     ; INITIALIZE IE INDEX
        ADDI    2,R0
        STI     R0,@JT          ; JT=R0/2+2
        SUBI    2,R0
        LSH     1,R0            ; R0=N2
*
*   OUTER LOOP
*
LOOP:
        LDI     @INPUT,AR0      ; AR0 POINTS TO X(I)
        ADDI    R0,AR0,AR1      ; AR1 POINTS TO X(I1)
        ADDI    R0,AR1,AR2      ; AR2 POINTS TO X(I2)
        ADDI    R0,AR2,AR3      ; AR3 POINTS TO X(I3)
        LDI     @RPTCNT,RC
        SUBI    1,RC            ; RC SHOULD BE ONE LESS THAN DESIRED #
*
*   FIST LOOP
*
        RPTB    BLK1
        ADDF    *+AR0,*+AR2,R1  ; R1=Y(I)+Y(I2)
        ADDF    *+AR3,*+AR1,R3  ; R3=Y(I1)+Y(I3)
        ADDF    R3,R1,R6        ; R6=R1+R3
```

```
        SUBF    *+AR2,*+AR0,R4     ; R4=Y(I)-Y(I2)
        STF     R6,*+AR0           ; Y(I)=R1+R3
        SUBF    R3,R1              ; R1=R1-R3
        LDF     *AR2,R5            ; R5=X(I2)
||      LDF     *+AR1,R7           ; R7=Y(I1)
        ADDF    *AR3,*AR1,R3       ; R3=X(I1)+X(I3)
        ADDF    R5,*AR0,R1         ; R1=X(I)+X(I2)
||      STF     R1,*+AR1           ; Y(I1)=R1-R3
        ADDF    R3,R1,R6           ; R6=R1+R3
        SUBF    R5,*AR0,R2         ; R2=X(I)-X(I2)
||      STF     R6,*AR0++(IR0)     ; X(I)=R1+R3
        SUBF    R3,R1              ; R1=R1-R3
        SUBF    *AR3,*AR1,R6       ; R6=X(I1)-X(I3)
        SUBF    R7,*+AR3,R3        ; -R3=Y(I1)-Y(I3)  !!!
||      STF     R1,*AR1++(IR0)     ; X(I1)=R1-R3
        SUBF    R6,R4,R5           ; R5=R4-R6
        ADDF    R6,R4              ; R4=R4+R6
        STF     R5,*+AR2           ; Y(I2)=R4-R6
||      STF     R4,*+AR3           ; Y(I3)=R4+R6
        SUBF    R3,R2,R5           ; R5=R2-R3  !!!
        ADDF    R3,R2              ; R2=R2+R3  !!!
BLK1    STF     R5,*AR2++(IR0)     ; X(I2)=R2-R3  !!!
||      STF     R2,*AR3++(IR0)     ; X(I3)=R2+R3  !!!
*
*   IF THIS IS THE LAST STAGE, YOU ARE DONE
*
        LDI     @STAGE,AR7
        ADDI    1,AR7
        CMPI    @LOGFFT,AR7
        BZD     END
        STI     AR7,@STAGE         ; CURRENT FFT STAGE
*
*   MAIN INNER LOOP
*
        LDI     1,AR7
        STI     AR7,@IA1           ; INIT IA1 INDEX
        LDI     2,AR7
        STI     AR7,@LPCNT         ; INIT LOOP COUNTER FOR INNER LOOP
INLOP:
        LDI     2,AR6              ; INCREMENT INNER LOOP COUNTER
        ADDI    @LPCNT,AR6
        LDI     @LPCNT,AR0
        LDI     @IA1,AR7
        ADDI    @IEINDX,AR7        ; IA1=IA1+IE
        ADDI    @INPUT,AR0         ; (X(I),Y(I)) POINTER
        STI     AR7,@IA1
        ADDI    R0,AR0,AR1         ; (X(I1),Y(I1)) POINTER
        STI     AR6,@LPCNT
        ADDI    R0,AR1,AR2         ; (X(I2),Y(I2)) POINTER
        ADDI    R0,AR2,AR3         ; (X(I3),Y(I3)) POINTER
        LDI     @RPTCNT,RC
        SUBI    1,RC               ; RC SHOULD BE ONE LESS THAN DESIRED #
        CMPI    @JT,AR6            ; IF LPCNT=JT, GO TO
        BZD     SPCL               ;   SPECIAL BUTTERFLY
        LDI     @IA1,AR7
        LDI     @IA1,AR4
        ADDI    @SINTAB,AR4        ; CREATE COSINE INDEX AR4
        ADDI    AR4,AR7,AR5
        SUBI    1,AR5              ; IA2=IA1+IA1-1
        ADDI    AR7,AR5,AR6
        SUBI    1,AR6              ; IA3=IA2+IA1-1
*
*   SECOND LOOP
*
        RPTB    BLK2
        ADDF    *+AR2,*+AR0,R3     ; R3=Y(I)+Y(I2)
        ADDF    *+AR3,*+AR1,R5     ; R5=Y(I1)+Y(I3)
        ADDF    R5,R3,R6           ; R6=R3+R5
        SUBF    *+AR2,*+AR0,R4     ; R4=Y(I)-Y(I2)
        SUBF    R5,R3              ; R3=R3-R5
        ADDF    *AR2,*AR0,R1       ; R1=X(I)+X(I2)
        ADDF    *AR3,*AR1,R5       ; R5=X(I1)+X(I3)
        MPYF    R3,*+AR5(IR1),R6   ; R6=R3*CO2
||      STF     R6,*+AR0           ; Y(I)=R3+R5
        ADDF    R5,R1,R7           ; R7=R1+R5
        SUBF    *AR2,*AR0,R2       ; R2=X(I)-X(I2)
        SUBF    R5,R1              ; R1=R1-R5
        MPYF    R1,*AR5,R7         ; R7=R1*SI2
||      STF     R7,*AR0++(IR0)     ; X(I)=R1+R5
        SUBF    R7,R6              ; R6=R3*CO2-R1*SI2
        SUBF    *+AR3,*+AR1,R5     ; R5=Y(I1)-Y(I3)
        MPYF    R1,*+AR5(IR1),R7   ; R7=R1*CO2
||      STF     R6,*+AR1           ; Y(I1)=R3*CO2-R1*SI2
        MPYF    R3,*AR5,R6         ; R6=R3*SI2
        ADDF    R7,R6              ; R6=R1*CO2+R3*SI2
        ADDF    R5,R2,R1           ; R1=R2+R5
        SUBF    R5,R2              ; R2=R2-R5
        SUBF    *AR3,*AR1,R5       ; R5=X(I1)-X(I3)
        SUBF    R5,R4,R3           ; R3=R4-R5
        ADDF    R5,R4              ; R4=R4+R5
        MPYF    R3,*+AR4(IR1),R6   ; R6=R3*CO1
||      STF     R6,*AR1++(IR0)     ; X(I1)=R1*CO2+R3*SI2
        MPYF    R1,*AR4,R7         ; R7=R1*SI1
        SUBF    R7,R6              ; R6=R3*CO1-R1*SI1
        MPYF    R1,*+AR4(IR1),R6   ; R6=R1*CO1
||      STF     R6,*+AR2           ; Y(I2)=R3*CO1-R1*SI1
        MPYF    R3,*AR4,R7         ; R7=R3*SI1
        ADDF    R7,R6              ; R6=R1*CO1+R3*SI1
        MPYF    R4,*+AR6(IR1),R6   ; R6=R4*CO3
||      STF     R6,*AR2++(IR0)     ; X(I2)=R1*CO1+R3*SI1
        MPYF    R2,*AR6,R7         ; R7=R2*SI3
        SUBF    R7,R6              ; R6=R4*CO3-R2*SI3
        MPYF    R2,*+AR6(IR1),R6   ; R6=R2*CO3
||      STF     R6,*+AR3           ; Y(I3)=R4*CO3-R2*SI3
        MPYF    R4,*AR6,R7         ; R7=R4*SI3
        ADDF    R7,R6              ; R6=R2*CO3+R4*SI3
BLK2    STF     R6,*AR3++(IR0)     ; X(I3)=R2*CO3+R4*SI3
*
```

```
        CMPI    @LPCNT,R0
        BP      INLOP           ; LOOP BACK TO THE INNER LOOP
        BR      CONT
*
*   SPECIAL BUTTERFLY FOR W=J
*
SPCL    LDI     IR1,AR4
        LSH     -1,AR4          ; POINT TO SIN(45)
        ADDI    @SINTAB,AR4     ; CREATE COSINE INDEX AR4=CO21
*
        RPTB    BLK3
        ADDF    *AR2,*AR0,R1    ; R1=X(I)+X(I2)
        SUBF    *AR2,*AR0,R2    ; R2=X(I)-X(I2)
        ADDF    *++AR2,*++AR0,R3 ; R3=Y(I)+Y(I2)
        SUBF    *++AR2,*++AR0,R4 ; R4=Y(I)-Y(I2)
        ADDF    *AR3,*AR1,R5    ; R5=X(I1)+X(I3)
        SUBF    R1,R5,R6        ; R6=R5-R1
        ADDF    R5,R1           ; R1=R1+R5
        ADDF    *++AR3,*++AR1,R5 ; R5=Y(I1)+Y(I3)
        SUBF    R5,R3,R7        ; R7=R3-R5
        ADDF    R5,R3           ; R3=R3+R5
        STF     R3,*++AR0       ; Y(I)=R3+R5
||      STF     R1,*AR0++(IR0)  ; X(I)=R1+R5
        SUBF    *AR3,*AR1,R1    ; R1=X(I1)-X(I3)
        SUBF    *++AR3,*++AR1,R3 ; R3=Y(I1)-Y(I3)
        STF     R6,*++AR1       ; Y(I1)=R5-R1
||      STF     R7,*AR1++(IR0)  ; X(I1)=R3-R5
        ADDF    R3,R2,R5        ; R5=R2+R3
        SUBF    R2,R3,R2        ; R2=-R2+R3 !!!
        SUBF    R1,R4,R3        ; R3=R4-R1
        ADDF    R1,R4           ; R4=R4+R1
        SUBF    R5,R3,R1        ; R1=R3-R5
        MPYF    *AR4,R1         ; R1=R1*CO21
        ADDF    R5,R3           ; R3=R3+R5
        MPYF    *AR4,R3         ; R3=R3*CO21
||      STF     R1,*++AR2       ; Y(I2)=(R3-R5)*CO21
        SUBF    R4,R2,R1        ; R1=R2-R4  !!!
        MPYF    *AR4,R1         ; R1=R1*CO21
||      STF     R3,*AR2++(IR0)  ; X(I2)=(R3+R5)*CO21
        ADDF    R4,R2           ; R2=R2+R4  !!!
        MPYF    *AR4,R2         ; R2=R2*CO21  !!!
BLK3    STF     R1,*++AR3       ; Y(I3)=-(R4-R2)*CO21  !!!
||      STF     R2,*AR3++(IR0)  ; X(I3)=(R4+R2)*CO21  !!!
*
*
        CMPI    @LPCNT,R0
        BPD     INLOP           ; LOOP BACK TO THE INNER LOOP
*
CONT    LDI     @RPTCNT,AR7
        LDI     @IEINDX,AR6
        LSH     2,AR7           ; INCREMENT REPEAT COUNTER FOR NEXT
*                               ;   TIME
        STI     AR7,@RPTCNT
        LSH     2,AR6           ; IE=4*IE
        STI     AR6,@IEINDX
        LDI     R0,IR0          ; N1=N2
        LSH     -3,R0
        ADDI    2,R0
        STI     R0,@JT          ; JT=N2/2+2
        SUBI    2,R0
        LSH     1,R0            ; N2=N2/4
        BR      LOOP            ; NEXT FFT STAGE
*
*   STORE RESULT OUT USING BIT-REVERSED ADDRESSING
*
END:    LDI     @FFTSIZ,RC      ; RC=N
        SUBI    1,RC            ; RC SHOULD BE ONE LESS THAN DESIRED #
        LDI     @FFTSIZ,IR0     ; IR0=SIZE OF FFT=N
        LDI     2,IR1
        LDI     @INPUT,AR0
        LDP     STORE
        LDI     @STORE,AR1
*
        RPTB    BITRV
        LDF     *++AR0(1),R0
||      LDF     *AR0++(IR0)B,R1
BITRV   STF     R0,*++AR1(1)
||      STF     R1,*AR1++(IR1)
*
SELF    BR      SELF            ; BRANCH TO ITSELF AT THE END
        .END
```

Appendix B2. fft_4 – Radix-4 Complex FFT to Be Called as a C Function

```
*
*    APPENDIX B2
*
*    NAME: fft_4 --- RADIX-4 COMPLEX FFT TO BE CALLED AS A C FUNCTION.
*
*    SYNOPSIS:
*         int    fft_4(N, M, DATA)
*         int    N        FFT SIZE: N=4**M
*         int    M        NUMBER OF STAGES = LOG4(N)
*         float  *data    ARRAY WITH INPUT AND OUTPUT DATA
*
*    DESCRIPTION:
*         GENERIC FUNCTION TO DO A RADIX-4 FFT COMPUTATION ON THE TMS320C30.
*         THE DATA ARRAY IS 2*N-LONG, WITH REAL AND IMAGINARY VALUES ALTER-
*         NATING. THE PROGRAM IS BASED ON THE FORTRAN PROGRAM IN THE BURRUS
*         AND PARKS BOOK, P. 117.
*
*         IN ORDER TO HAVE THE FINAL RESULT IN BIT-REVERSED ORDER, THE TWO
*         MIDDLE BRANCHES OF THE RADIX-4 BUTTERFLY ARE INTERCHANGED DURING
*         STORAGE. NOTE THIS DIFFERENCE WHEN COMPARING WITH THE PROGRAM ON
*         P. 117. THE COMPUTATION IS DONE IN-PLACE, AND THE ORIGINAL DATA IS
*         DESTROYED. BIT REVERSAL IS IMPLEMENTED AT THE END OF THE FUNCTION.
*         IF THIS IS NOT NECESSARY, THIS PART CAN BE COOOENTED OUT. THE
*         SINE/COSINE TABLE FOR THE TWIDDLE FACTORS IS EXPECTED TO BE SUPPLIED
*         DURING LINK TIME, AND IT SHOULD HAVE THE FOLLOWING FORMAT:
*
*                   .global   _sine
*                   .data
*         _sine     .float    value1  = sin(0*2*pi/N)
*                   .float    value2  = sin(1*2*pi/N)
*        ......
*                   .float    value(5N/4) = sin((5*N/4-1)*2*pi/N)
*
*         THE VALUES value1, value2, ETC., ARE THE SINE WAVE VALUES. FOR AN
*         N-POINT FFT, THERE ARE N+N/4 VALUES FOR A FULL AND A QUARTER PERIOD
*         OF THE SINE WAVE. IN THIS WAY, A FULL SINE AND COSINE PERIOD ARE
*         AVAILABLE (SUPERIMPOSED).
*
*    STACK STRUCTURE UPON THE CALL:
*                   +--------------+
*         -FP(4)    :    DATA      :
*         -FP(3)    :    M         :
*         -FP(2)    :    N         :
*         -FP(1)    : RETURN ADDR  :
*         -FP(0)    :   OLD FP     :
*                   +--------------+
*
*    REGISTERS USED: R0, R1, R2, R3, R4, R5, R6, R7, AR0, AR1, AR2, AR3, AR4,
*                    AR5, AR6, AR7, IR0, IR1, RS, RE, RC
*
*    AUTHOR: PANOS E. PAPAMICHALIS
*            TEXAS INSTRUMENTS                          OCTOBER 13, 1987
*
*****************************************************************************
*
FP        .SET      AR3
*
          .GLOBL    _FFT_4              ; ENTRY POINT FOR EXECUTION
          .GLOBL    _SINE               ; ADDRESS OF SINE TABLE
*
          .BSS      FFTSIZ,1
          .BSS      LOGFFT,1
          .BSS      INPUT,1
*
          .TEXT
*
SINTAB    .word     _SINE
*
*    INITIALIZE C FUNCTION
*
_fft_4:   PUSH      FP                  ; SAVE DEDICATED REGISTERS
          LDI       SP,FP
          PUSH      R4
          PUSH      R5
          PUSHF     R6
          PUSHF     R7
          PUSH      AR4
          PUSH      AR5
          PUSH      AR6
          PUSH      AR7
*
          LDI       *-FP(2),R0          ; MOVE ARGUMENTS TO LOCATIONS MATCHING
          STI       R0,@FFTSIZ          ; THE NAMES IN THE PROGRAM
          LDI       *-FP(3),R0
          STI       R0,@LOGFFT
          LDI       *-FP(4),R0
          STI       R0,@INPUT
*
*    INITIALIZE FFT ROUTINE
*
          .BSS      STAGE,1             ; FFT STAGE #
          .BSS      RPTCNT,1            ; REPEAT COUNTER
          .BSS      IEINDX,1            ; IE INDEX FOR SINE/COSINE
          .BSS      LPCNT,1             ; SECOND-LOOP COUNT
          .BSS      JT,1                ; JT COUNTER IN PROGRAM, P. 117
          .BSS      IA1,1               ; IA1 INDEX IN PROGRAM, P. 117
*
          LDI       @FFTSIZ,R0
          LDI       @FFTSIZ,IR0
          LDI       @FFTSIZ,IR1
          LDI       0,AR7
          STI       AR7,@STAGE          ; @STAGE HOLDS THE CURRENT STAGE
*                                       ;   NUMBER
          LSH       1,IR0               ; IR0=2*N1 (BECAUSE OF REAL/IMAG)
          LSH       -2,IR1              ; IR1=N/4, POINTER FOR SIN/COS TABLE
          LDI       1,AR7
          STI       AR7,@RPTCNT         ; INITIALIZE REPEAT COUNTER OF FIRST
*                                       ;   LOOP
```

```
        LSH     -2,R0
        STI     AR7,@IEINDX         ; INITIALIZE IE INDEX
        ADDI    2,R0
        STI     R0,@JT              ; JT=R0/2+2
        SUBI    2,R0
        LSH     1,R0                ; R0=N2
*
*   OUTER LOOP
*
LOOP:
        LDI     @INPUT,AR0          ; AR0 POINTS TO X(I)
        ADDI    R0,AR0,AR1          ; AR1 POINTS TO X(I1)
        ADDI    R0,AR1,AR2          ; AR2 POINTS TO X(I2)
        ADDI    R0,AR2,AR3          ; AR3 POINTS TO X(I3)
        LDI     @RPTCNT,RC
        SUBI    1,RC                ; RC SHOULD BE ONE LESS THAN DESIRED #
*
*   FIST LOOP
*
        RPTB    BLK1
        ADDF    *++AR0,*++AR2,R1    ; R1=Y(I)+Y(I2)
        ADDF    *++AR3,*++AR1,R3    ; R3=Y(I1)+Y(I3)
        ADDF    R3,R1,R6            ; R6=R1+R3
        SUBF    *++AR2,*++AR0,R4    ; R4=Y(I)-Y(I2)
        STF     R6,*++AR0           ; Y(I)=R1+R3
        SUBF    R3,R1               ; R1=R1-R3
        LDF     *AR2,R5             ; R5=X(I2)
||      LDF     *++AR1,R7           ; R7=Y(I1)
        ADDF    *AR3,*AR1,R3        ; R3=X(I1)+X(I3)
        ADDF    R5,*AR0,R1          ; R1=X(I)+X(I2)
||      STF     R1,*++AR1           ; Y(I1)=R1-R3
        ADDF    R3,R1,R6            ; R6=R1+R3
        SUBF    R5,*AR0,R2          ; R2=X(I)-X(I2)
||      STF     R6,*AR0++(IR0)      ; X(I)=R1+R3
        SUBF    R3,R1               ; R1=R1-R3
        SUBF    *AR3,*AR1,R6        ; R6=X(I1)-X(I3)
        SUBF    R7,*++AR3,R3        ; -R3=Y(I1)-Y(I3)  !!!
||      STF     R1,*AR1++(IR0)      ; X(I1)=R1-R3
        SUBF    R6,R4,R5            ; R5=R4-R6
        ADDF    R6,R4               ; R4=R4+R6
        STF     R5,*++AR2           ; Y(I2)=R4-R6
||      STF     R4,*++AR3           ; Y(I3)=R4+R6
        SUBF    R3,R2,R5            ; R5=R2-R3  !!!
        ADDF    R3,R2               ; R2=R2+R3  !!!
BLK1    STF     R5,*AR2++(IR0)      ; X(I2)=R2-R3  !!!
||      STF     R2,*AR3++(IR0)      ; X(I3)=R2+R3  !!!
*
*   IF THIS IS THE LAST STAGE, YOU ARE DONE
*
        LDI     @STAGE,AR7
        ADDI    1,AR7
        CMPI    @LOGFFT,AR7
        BZD     END
        STI     AR7,@STAGE          ; CURRENT FFT STAGE
```

```
*
*   MAIN INNER LOOP
*
        LDI     1,AR7
        STI     AR7,@IA1            ; INIT IA1 INDEX
        LDI     2,AR7
        STI     AR7,@LPCNT          ; INIT LOOP COUNTER FOR INNER LOOP
INLOP:
        LDI     2,AR6               ; INCREMENT INNER LOOP COUNTER
        ADDI    @LPCNT,AR6
        LDI     @LPCNT,AR0
        LDI     @IA1,AR7
        ADDI    @IEINDX,AR7         ; IA1=IA1+IE
        ADDI    @INPUT,AR0          ; (X(I),Y(I)) POINTER
        STI     AR7,@IA1
        ADDI    R0,AR0,AR1          ; (X(I1),Y(I1)) POINTER
        STI     AR6,@LPCNT
        ADDI    R0,AR1,AR2          ; (X(I2),Y(I2)) POINTER
        ADDI    R0,AR2,AR3          ; (X(I3),Y(I3)) POINTER
        LDI     @RPTCNT,RC
        SUBI    1,RC                ; RC SHOULD BE ONE LESS THAN DESIRED #
        CMPI    @JT,AR6             ; IF LPCNT=JT, GO TO
        BZD     SPCL                ;   SPECIAL BUTTERFLY
        LDI     @IA1,AR7
        LDI     @IA1,AR4
        ADDI    @SINTAB,AR4         ; CREATE COSINE INDEX AR4
        ADDI    AR4,AR7,AR5
        SUBI    1,AR5               ; IA2=IA1+IA1-1
        ADDI    AR7,AR5,AR6
        SUBI    1,AR6               ; IA3=IA2+IA1-1
*
*   SECOND LOOP
*
        RPTB    BLK2
        ADDF    *++AR2,*++AR0,R3    ; R3=Y(I)+Y(I2)
        ADDF    *++AR3,*++AR1,R5    ; R5=Y(I1)+Y(I3)
        ADDF    R5,R3,R6            ; R6=R3+R5
        SUBF    *++AR2,*++AR0,R4    ; R4=Y(I)-Y(I2)
        SUBF    R5,R3               ; R3=R3-R5
        ADDF    *AR2,*AR0,R1        ; R1=X(I)+X(I2)
        ADDF    *AR3,*AR1,R5        ; R5=X(I1)+X(I3)
        MPYF    R3,*++AR5(IR1),R6   ; R6=R3*CO2
||      STF     R6,*++AR0           ; Y(I)=R3+R5
        ADDF    R5,R1,R7            ; R7=R1+R5
        SUBF    *AR2,*AR0,R2        ; R2=X(I)-X(I2)
        SUBF    R5,R1               ; R1=R1-R5
        MPYF    R1,*AR5,R7          ; R7=R1*SI2
||      STF     R7,*AR0++(IR0)      ; X(I)=R1+R5
        SUBF    R7,R6               ; R6=R3*CO2-R1*SI2
        SUBF    *++AR3,*++AR1,R5    ; R5=Y(I1)-Y(I3)
        MPYF    R1,*++AR5(IR1),R7   ; R7=R1*CO2
||      STF     R6,*++AR1           ; Y(I1)=R3*CO2-R1*SI2
        MPYF    R3,*AR5,R6          ; R6=R3*SI2
        ADDF    R7,R6               ; R6=R1*CO2+R3*SI2
```

```
        ADDF    R5,R2,R1            ; R1=R2+R5
        SUBF    R5,R2               ; R2=R2-R5
        SUBF    *AR3,*AR1,R5        ; R5=X(I1)-X(I3)
        SUBF    R5,R4,R3            ; R3=R4-R5
        ADDF    R5,R4               ; R4=R4+R5
        MPYF    R3,*+AR4(IR1),R6    ; R6=R3*CO1
||      STF     R6,*AR1++(IR0)      ; X(I1)=R1*CO2+R3*SI2
        MPYF    R1,*AR4,R7          ; R7=R1*SI1
        SUBF    R7,R6               ; R6=R3*CO1-R1*SI1
        MPYF    R1,*+AR4(IR1),R6    ; R6=R1*CO1
||      STF     R6,*+AR2            ; Y(I2)=R3*CO1-R1*SI1
        MPYF    R3,*AR4,R7          ; R7=R3*SI1
        ADDF    R7,R6               ; R6=R1*CO1+R3*SI1
        MPYF    R4,*+AR6(IR1),R6    ; R6=R4*CO3
||      STF     R6,*AR2++(IR0)      ; X(I2)=R1*CO1+R3*SI1
        MPYF    R2,*AR6,R7          ; R7=R2*SI3
        SUBF    R7,R6               ; R6=R4*CO3-R2*SI3
        MPYF    R2,*+AR6(IR1),R6    ; R6=R2*CO3
||      STF     R6,*+AR3            ; Y(I3)=R4*CO3-R2*SI3
        MPYF    R4,*AR6,R7          ; R7=R4*SI3
        ADDF    R7,R6               ; R6=R2*CO3+R4*SI3
BLK2    STF     R6,*AR3++(IR0)      ; X(I3)=R2*CO3+R4*SI3
*
        CMPI    @LPCNT,R0
        BP      INLOP               ; LOOP BACK TO THE INNER LOOP
        BR      CONT
*
*   SPECIAL BUTTERFLY FOR W=J
*
SPCL    LDI     IR1,AR4
        LSH     -1,AR4              ; POINT TO SIN(45)
        ADDI    @SINTAB,AR4         ; CREATE COSINE INDEX AR4=CO21
*
        RPTB    BLK3
        ADDF    *AR2,*AR0,R1        ; R1=X(I)+X(I2)
        SUBF    *AR2,*AR0,R2        ; R2=X(I)-X(I2)
        ADDF    *+AR2,*+AR0,R3      ; R3=Y(I)+Y(I2)
        SUBF    *+AR2,*+AR0,R4      ; R4=Y(I)-Y(I2)
        ADDF    *AR3,*AR1,R5        ; R5=X(I1)+X(I3)
        SUBF    R1,R5,R6            ; R6=R5-R1
        ADDF    R5,R1               ; R1=R1+R5
        ADDF    *+AR3,*+AR1,R5      ; R5=Y(I1)+Y(I3)
        SUBF    R5,R3,R7            ; R7=R3-R5
        ADDF    R5,R3               ; R3=R3+R5
        STF     R3,*+AR0            ; Y(I)=R3+R5
||      STF     R1,*AR0++(IR0)      ; X(I)=R1+R5
        SUBF    *AR3,*AR1,R1        ; R1=X(I1)-X(I3)
        SUBF    *+AR3,*+AR1,R3      ; R3=Y(I1)-Y(I3)
        STF     R6,*+AR1            ; Y(I1)=R5-R1
||      STF     R7,*AR1++(IR0)      ; X(I1)=R3-R5
        ADDF    R3,R2,R5            ; R5=R2+R3
        SUBF    R2,R3,R2            ; R2=-R2+R3 !!!
        SUBF    R1,R4,R3            ; R3=R4-R1
        ADDF    R1,R4               ; R4=R4+R1
        SUBF    R5,R3,R1            ; R1=R3-R5
        MPYF    *AR4,R1             ; R1=R1*CO21
        ADDF    R5,R3               ; R3=R3+R5
        MPYF    *AR4,R3             ; R3=R3*CO21
||      STF     R1,*+AR2            ; Y(I2)=(R3-R5)*CO21
        SUBF    R4,R2,R1            ; R1=R2-R4  !!!
        MPYF    *AR4,R1             ; R1=R1*CO21
||      STF     R3,*AR2++(IR0)      ; X(I2)=(R3+R5)*CO21
        ADDF    R4,R2               ; R2=R2+R4  !!!
        MPYF    *AR4,R2             ; R2=R2*CO21  !!!
BLK3    STF     R1,*+AR3            ; Y(I3)=-(R4-R2)*CO21  !!!
||      STF     R2,*AR3++(IR0)      ; X(I3)=(R4+R2)*CO21  !!!
*
        CMPI    @LPCNT,R0
        BPD     INLOP               ; LOOP BACK TO THE INNER LOOP
*
CONT    LDI     @RPTCNT,AR7
        LDI     @IEINDX,AR6
        LSH     2,AR7               ; INCREMENT REPEAT COUNTER FOR NEXT
*                                   ;   TIME
        STI     AR7,@RPTCNT
        LSH     2,AR6               ; IE=4*IE
        STI     AR6,@IEINDX
        LDI     R0,IR0              ; N1=N2
        LSH     -3,R0
        ADDI    2,R0
        STI     R0,@JT              ; JT=N2/2+2
        SUBI    2,R0
        LSH     1,R0                ; N2=N2/4
        BR      LOOP                ; NEXT FFT STAGE
*
*   DO THE BIT-REVERSING OF THE OUTPUT
*
END:    LDI     @FFTSIZ,RC          ; RC=N
        SUBI    1,RC                ; RC SHOULD BE ONE LESS THAN DESIRED #
        LDI     @FFTSIZ,IR0         ; IR0=SIZE OF FFT=N
        LDI     @INPUT,AR0
        LDI     @INPUT,AR1
*
        RPTB    BITRV
        CMPI    AR0,AR1
        BGE     CONT
        LDF     *AR0,R0
||      LDF     *AR1,R1
        STF     R0,*AR1
||      STF     R1,*AR0
        LDF     *+AR0(1),R0
||      LDF     *+AR1(1),R1
        STF     R0,*+AR1(1)
||      STF     R1,*+AR0(1)
CONT    NOP     *++AR0(2)
BITRV   NOP     *AR1++(IR0)B
*
*   RESTORE THE REGISTER VALUES AND RETURN
```

```
*
        POP     AR7
        POP     AR6
        POP     AR5
        POP     AR4
        POPF    R7
        POPF    R6
        POP     R5
        POP     R4
        POP     FP
        RETS
```

Appendix C.Radix-2 Real FFT

Appendix C1. Generic Program to Do a Radix-2 Real FFT Computation on the TMS320C30

```
*
*     APPENDIX C1
*
*     GENERIC PROGRAM TO DO A RADIX-2 REAL FFT COMPUTATION ON THE TMS320C30
*
*     THE PROGRAM IS TAKEN FROM THE PAPER BY SORENSEN ET AL., JUNE 1987 ISSUE
*     OF THE TRANSACTIONS ON ASSP.
*
*     THE (REAL) DATA RESIDE IN INTERNAL MEMORY. THE COMPUTATION IS DONE
*     IN-PLACE. THE BIT REVERSAL IS DONE AT THE BEGINNING OF THE PROGRAM.
*
*     THE TWIDDLE FACTORS ARE SUPPLIED IN A TABLE PUT IN A .DATA SECTION. THIS
*     DATA IS INCLUDED IN A SEPARATE FILE TO PRESERVE THE GENERIC NATURE OF THE
*     PROGRAM. FOR THE SAME PURPOSE, THE SIZE OF THE FFT N AND LOG2(N) ARE
*     DEFINED IN A .GLOBL DIRECTIVE AND SPECIFIED DURING LINKING. THE LENGTH OF
*     THE TABLE IS N/4 + N/4 = N/2.
*
*     AUTHOR: PANOS E. PAPAMICHALIS
*             TEXAS INSTRUMENTS                         SEPTEMBER 8, 1987
*
        .GLOBL  FFT                     ; ENTRY POINT FOR EXECUTION
        .GLOBL  N                       ; FFT SIZE
        .GLOBL  M                       ; LOG2(N)
        .GLOBL  SINE                    ; ADDRESS OF SINE TABLE
*
INP     .USECT  "IN",1024               ; MEMORY WITH INPUT DATA
        .BSS    OUTP,1024               ; MEMORY WITH OUTPUT DATA
*
        .TEXT
*
*     INITIALIZE
*
        .WORD   FFT                     ; STARTING LOCATION OF THE PROGRAM
*
        .SPACE  100                     ; RESERVE 100 WORDS FOR VECTORS, ETC.
*
FFTSIZ  .WORD   N
LOGFFT  .WORD   M
SINTAB  .WORD   SINE
INPUT   .WORD   INP
OUTPUT  .WORD   OUTP
*
FFT:    LDP     FFTSIZ                  ; COMMAND TO LOAD DATA PAGE POINTER
*
*     DO THE BIT-REVERSING AT THE BEGINNING
*
        LDI     @FFTSIZ,RC              ; RC=N
        SUBI    1,RC                    ; RC SHOULD BE ONE LESS THAN DESIRED #
        LDI     @FFTSIZ,IR0
        LSH     -1,IR0                  ; IR0=HALF THE SIZE OF FFT=N/2
        LDI     @INPUT,AR0
        LDI     @INPUT,AR1
*
        RPTB    BITRV
        CMPI    AR1,AR0                 ; XCHANGE LOCATIONS ONLY
        BGE     CONT                    ; IF AR0<AR1
        LDF     *AR0,R0
::      LDF     *AR1,R1
        STF     R0,*AR1
::      STF     R1,*AR0
CONT    NOP     *AR0++
BITRV   NOP     *AR1++(IR0)B
*
*     LENGTH-TWO BUTTERFLIES
*
        LDI     @INPUT,AR0              ; AR0 POINTS TO X(I)
        LDI     IR0,RC                  ; REPEAT N/2 TIMES
        SUBI    1,RC                    ; RC SHOULD BE ONE LESS THAN DESIRED #
*
        RPTB    BLK1
        ADDF    *+AR0,*AR0++,R0         ; R0=X(I)+X(I+1)
        SUBF    *AR0,*-AR0,R1           ; R1=X(I)-X(I+1)
BLK1    STF     R0,*-AR0                ; X(I)=X(I)+X(I+1)
::      STF     R1,*AR0++               ; X(I+1)=X(I)-X(I+1)
*
*     FIRST PASS OF THE DO-20 LOOP (STAGE K=2 IN DO-10 LOOP)
*
        LDI     @INPUT,AR0              ; AR0 POINTS TO X(I)
        LDI     2,IR0                   ; IR0=2=N2
        LDI     @FFTSIZ,RC
        LSH     -2,RC                   ; REPEAT N/4 TIMES
        SUBI    1,RC                    ; RC SHOULD BE ONE LESS THAN DESIRED #
*
        RPTB    BLK2
        ADDF    *+AR0(IR0),*AR0++(IR0),R0    ; R0=X(I)+X(I+2)
        SUBF    *AR0,*-AR0(IR0),R1      ; R1=X(I)-X(I+2)
        NEGF    *+AR0,R0                ; R0=-X(I+3)
::      STF     R0,*-AR0(IR0)           ; X(I)=X(I)+X(I+2)
BLK2    STF     R1,*AR0++(IR0)          ; X(I+2)=X(I)-X(I+2)
::      STF     R0,*+AR0                ; X(I+3)=-X(I+3)
*
*     MAIN LOOP (FFT STAGES)
*
        LDI     @FFTSIZ,IR0
        LSH     -2,IR0                  ; IR0=INDEX FOR E
        LDI     3,R5                    ; R5 HOLDS THE CURRENT STAGE NUMBER
        LDI     1,R4                    ; R4=N4
        LDI     2,R3                    ; R3=N2
LOOP    LSH     -1,IR0                  ; E=E/2
        LSH     1,R4                    ; N4=2*N4
        LSH     1,R3                    ; N2=2*N2
*
*     INNER LOOP (DO-20 LOOP IN THE PROGRAM)
*
        LDI     @INPUT,AR5              ; AR5 POINTS TO X(I)
INLOP   LDI     IR0,AR0
        ADDI    @SINTAB,AR0             ; AR0 POINTS TO SIN/COS TABLE
        LDI     R4,IR1                  ; IR1=N4
```

```
*
        LDI     AR5,AR1
        ADDI    1,AR1                 ; AR1 POINTS TO X(I1)=X(I+J)
        LDI     AR1,AR3
        ADDI    R3,AR3                ; AR3 POINTS TO X(I3)=X(I+J+N2)
        LDI     AR3,AR2
        SUBI    2,AR2                 ; AR2 POINTS TO X(I2)=X(I-J+N2)
        ADDI    R3,AR2,AR4            ; AR4 POINTS TO X(I4)=X(I-J+N1)
*
        LDF     *AR5++(IR1),R0        ; R0=X(I)
        ADDF    *+AR5(IR1),R0,R1      ; R1=X(I)+X(I+N2)
        SUBF    R0,*++AR5(IR1),R0     ; R0=-X(I)+X(I+N2)
||      STF     R1,*-AR5(IR1)         ; X(I)=X(I)+X(I+N2)
        NEGF    R0                    ; R0=X(I)-X(I+N2)
        NEGF    *++AR5(IR1),R1        ; R1=-X(I+N4+N2)
||      STF     R0,*AR5               ; X(I+N2)=X(I)-X(I+N2)
        STF     R1,*AR5               ; X(I+N4+N2)=-X(I+N4+N2)
*
*   INNERMOST LOOP
*
        LDI     @FFTSIZ,IR1
        LSH     -2,IR1                ; IR1=SEPARATION BETWEEN SIN/COS TBLS
        LDI     R4,RC
        SUBI    2,RC                  ; REPEAT N4-1 TIMES
*
        RPTB    BLK3
        MPYF    *AR3,*+AR0(IR1),R0    ; R0=X(I3)*COS
        MPYF    *AR4,*AR0,R1          ; R1=X(I4)*SIN
        MPYF    *AR4,*+AR0(IR1),R1    ; R1=X(I4)*COS
||      ADDF    R0,R1,R2              ; R2=X(I3)*COS+X(I4)*SIN
        MPYF    *AR3,*AR0++(IR0),R0   ; R0=X(I3)*SIN
        SUBF    R0,R1,R0              ; R0=-X(I3)*SIN+X(I4)*COS !!!
        SUBF    *AR2,R0,R1            ; R1=-X(I2)+R0 !!!
        ADDF    *AR2,R0,R1            ; R1=X(I2)+R0 !!!
||      STF     R1,*AR3++             ; X(I3)=-X(I2)+R0 !!!
        ADDF    *AR1,R2,R1            ; R1=X(I1)+R2
||      STF     R1,*AR4--             ; X(I4)=X(I2)+R0 !!!
        SUBF    R2,*AR1,R1            ; R1=X(I1)-R2
||      STF     R1,*AR1++             ; X(I1)=X(I1)+R2
BLK3    STF     R1,*AR2--             ; X(I2)=X(I1)-R2
*
        SUBI    @INPUT,AR5
        ADDI    R4,AR5                ; AR5=I+N1
        CMPI    @FFTSIZ,AR5
        BLTD    INLOP                 ; LOOP BACK TO THE INNER LOOP
        ADDI    @INPUT,AR5
        NOP
        NOP
*
        ADDI    1,R5
        CMPI    @LOGFFT,R5
        BLE     LOOP
        NOP
        NOP
        NOP
        NOP
*
END     BR      END                   ;BRANCH TO ITSELF AT THE END
        .END
```

Appendix C2. fft_rl – Radix-2 Real FFT to Be Called as a C Function

```
*
*   APPENDIX C2
*
*   NAME:
*       fft_rl --- RADIX-2 REAL FFT TO BE CALLED AS A C FUNCTION.
*
*   SYNOPSIS:
*       int fft_rl(N, M, data)
*       int N       FFT SIZE: N=2**M
*       int M       NUMBER OF STAGES = LOG2(N)
*       float *data     ARRAY WITH INPUT AND OUTPUT DATA
*
*   DESCRIPTION:
*       GENERIC FUNCTION TO DO A RADIX-2 FFT COMPUTATION ON THE TMS320C30.
*       THE DATA ARRAY IS N-LONG, WITH ONLY REAL DATA. THE OUTPUT IS STORED
*       IN THE SAME LOCATIONS WITH REAL AND IMAGINARY POINTS R AND I AS
*       FOLLOWS: R(0), R(1),..., R(N/2), I(N/2-1),..., I(1)
*
*       THE PROGRAM IS BASED ON THE FORTRAN PROGRAM IN THE PAPER BY SORENSEN
*       ET AL., JUNE 1987 ISSUE OF TRANS. ON ASSP. THE COMPUTATION IS DONE
*       IN-PLACE, AND THE ORIGINAL DATA IS DESTROYED. BIT REVERSAL IS
*       IMPLEMENTED AT THE BEGINNING OF THE FUNCTION. IF THIS IS NOT
*       NECESSARY, THIS PART CAN BE COMMENTED OUT.
*
*       THE SINE/COSINE TABLE FOR THE TWIDDLE FACTORS IS EXPECTED TO BE
*       SUPPLIED DURING LINK TIME, AND IT SHOULD HAVE THE FOLLOWING FORMAT:
*
*
*               .global   _sine
*               .data
*       _sine   .float  value1  = sin(0*2*pi/N)
*               .float  value2  = sin(1*2*pi/N)
*     ......
*               .float  value(N/2)    = cos((N/4)*2*pi/N)
*
*       THE VALUES value1 TO value(N/4) ARE THE FIRST QUARTER OF THE SINE
*       PERIOD ANDd value(N/4+1) TO value(N/2) ARE THE FIRST QUARTER OF THE
*       COSINE PERIOD.
*
*       STACK STRUCTURE UPON THE CALL:
*                  +--------------+
*       -FP(4)     :    DATA      :
*       -FP(3)     :     M        :
*       -FP(2)     :     N        :
*       -FP(1)     : RETURN ADDR  :
*       -FP(0)     :   OLD FP     :
*                  +--------------+
*
*   REGISTERS USED: R0, R1, R2, R3, R4, R5, AR0, AR1, AR2, AR4, AR5, IR0,
*                   IR1, RS, RE, RC
*
*   AUTHOR: PANOS E. PAPAMICHALIS
*           TEXAS INSTRUMENTS                          OCTOBER 13, 1987
*
****************************************************************************
```

```
*
FP      .SET    AR3
*
        .GLOBL  _FFT_RL         ; ENTRY POINT FOR EXECUTION
        .GLOBL  _SINE           ; ADDRESS OF SINE TABLE
*
        .BSS    FFTSIZ,1
        .BSS    LOGFFT,1
        .BSS    INPUT,1
*
        .TEXT
*
SINTAB  .word   _SINE
*
*   INITIALIZE C FUNCTION
*
_FFT_RL: PUSH   FP              ; SAVE DEDICATED REGISTERS
        LDI     SP,FP
        PUSH    R4
        PUSH    R5
        PUSH    AR4
        PUSH    AR5
*
        LDI     *-FP(2),R0      ; MOVE ARGUMENTS TO LOCATIONS MATCHING
        STI     R0,@FFTSIZ      ;   THE NAMES IN THE PROGRAM
        LDI     *-FP(3),R0
        STI     R0,@LOGFFT
        LDI     *-FP(4),R0
        STI     R0,@INPUT
*
*   DO THE BIT REVERSING AT THE BEGINNING
*
        LDI     @FFTSIZ,RC      ; RC=N
        SUBI    1,RC            ; RC SHOULD BE ONE LESS THAN DESIRED #
        LDI     @FFTSIZ,IR0
        LSH     -1,IR0          ; IR0=HALF THE SIZE OF FFT=N/2
        LDI     @INPUT,AR0
        LDI     @INPUT,AR1
*
        RPTB    BITRV
        CMPI    AR1,AR0         ; XCHANGE LOCATIONS ONLY
        BGE     CONT            ; IF AR0<AR1
        LDF     *AR0,R0
||      LDF     *AR1,R1
        STF     R0,*AR1
||      STF     R1,*AR0
CONT    NOP     *AR0++
BITRV   NOP     *AR1++(IR0)B
*
*   LENGTH-TWO BUTTERFLIES
*
        LDI     @INPUT,AR0      ; AR0 POINTS TO X(I)
        LDI     IR0,RC          ; REPEAT N/2 TIMES
        SUBI    1,RC            ; RC SHOULD BE ONE LESS THAN DESIRED #
```

```
*
        RPTB    BLK1
        ADDF    *++AR0,*AR0++,R0      ; R0=X(I)+X(I+1)
        SUBF    *AR0,*--AR0,R1        ; R1=X(I)-X(I+1)
BLK1    STF     R0,*-AR0              ; X(I)=X(I)+X(I+1)
||      STF     R1,*AR0++             ; X(I+1)=X(I)-X(I+1)
*
*   FIRST PASS OF THE DO-20 LOOP (STAGE K=2 IN DO-10 LOOP)
*
        LDI     @INPUT,AR0            ; AR0 POINTS TO X(I)
        LDI     2,IR0                 ; IR0=2=N2
        LDI     @FFTSIZ,RC
        LSH     -2,RC                 ; REPEAT N/4 TIMES
        SUBI    1,RC                  ; RC SHOULD BE ONE LESS THAN DESIRED #
*
        RPTB    BLK2
        ADDF    *++AR0(IR0),*AR0++(IR0),R0    ; R0=X(I)+X(I+2)
        SUBF    *AR0,*--AR0(IR0),R1   ; R1=X(I)-X(I+2)
        NEGF    *++AR0,R0             ; R0=-X(I+3)
||      STF     R0,*-AR0(IR0)         ; X(I)=X(I)+X(I+2)
BLK2    STF     R1,*AR0++(IR0)        ; X(I+2)=X(I)-X(I+2)
||      STF     R0,*++AR0             ; X(I+3)=-X(I+3)
*
*   MAIN LOOP (FFT STAGES)
*
        LDI     @FFTSIZ,IR0
        LSH     -2,IR0                ; IR0=INDEX FOR E
        LDI     3,R5                  ; R5 HOLDS THE CURRENT STAGE NUMBER
        LDI     1,R4                  ; R4=N4
        LDI     2,R3                  ; R3=N2
LOOP    LSH     -1,IR0                ; E=E/2
        LSH     1,R4                  ; N4=2*N4
        LSH     1,R3                  ; N2=2*N2
*
*   INNER LOOP (DO-20 LOOP IN THE PROGRAM)
*
        LDI     @INPUT,AR5            ; AR5 POINTS TO X(I)
INLOP   LDI     IR0,AR0
        ADDI    @SINTAB,AR0           ; AR0 POINTS TO SIN/COS TABLE
        LDI     R4,IR1                ; IR1=N4
*
        LDI     AR5,AR1
        ADDI    1,AR1                 ; AR1 POINTS TO X(I1)=X(I+J)
        LDI     AR1,AR3
        ADDI    R3,AR3                ; AR3 POINTS TO X(I3)=X(I+J+N2)
        LDI     AR3,AR2
        SUBI    2,AR2                 ; AR2 POINTS TO X(I2)=X(I-J+N2)
        ADDI    R3,AR2,AR4            ; AR4 POINTS TO X(I4)=X(I-J+N1)
*
        LDF     *AR5++(IR1),R0        ; R0=X(I)
        ADDF    *++AR5(IR1),R0,R1     ; R1=X(I)+X(I+N2)
        SUBF    R0,*++AR5(IR1),R0     ; R0=-X(I)+X(I+N2)
||      STF     R1,*-AR5(IR1)         ; X(I)=X(I)+X(I+N2)
        NEGF    R0                    ; R0=X(I)-X(I+N2)
        NEGF    *++AR5(IR1),R1        ; R1=-X(I+N4+N2)
||      STF     R0,*AR5               ; X(I+N2)=X(I)-X(I+N2)
        STF     R1,*AR5               ; X(I+N4+N2)=-X(I+N4+N2)
*
*   INNERMOST LOOP
*
        LDI     @FFTSIZ,IR1
        LSH     -2,IR1                ; IR1=SEPARATION BETWEEN SIN/COS TBLS
        LDI     R4,RC
        SUBI    2,RC                  ; REPEAT N4-1 TIMES
*
        RPTB    BLK3
        MPYF    *AR3,*++AR0(IR1),R0   ; R0=X(I3)*COS
        MPYF    *AR4,*AR0,R1          ; R1=X(I4)*SIN
        MPYF    *AR4,*++AR0(IR1),R1   ; R1=X(I4)*COS
||      ADDF    R0,R1,R2              ; R2=X(I3)*COS+X(I4)*SIN
        MPYF    *AR3,*AR0++(IR0),R0   ; R0=X(I3)*SIN
        SUBF    R0,R1,R0              ; R0=-X(I3)*SIN+X(I4)*COS !!!
        SUBF    *AR2,R0,R1            ; R1=-X(I2)+R0 !!!
        ADDF    *AR2,R0,R1            ; R1=X(I2)+R0 !!!
||      STF     R1,*AR3++             ; X(I3)=-X(I2)+R0 !!!
        ADDF    *AR1,R2,R1            ; R1=X(I1)+R2
||      STF     R1,*AR4--             ; X(I4)=X(I2)+R0 !!!
        SUBF    R2,*AR1,R1            ; R1=X(I1)-R2
||      STF     R1,*AR1++             ; X(I1)=X(I1)+R2
BLK3    STF     R1,*AR2--             ; X(I2)=X(I1)-R2
*
        SUBI    @INPUT,AR5
        ADDI    R4, AR5               ; AR5=I+N1
        CMPI    @FFTSIZ,AR5
        BLTD    INLOP                 ; LOOP BACK TO THE INNER LOOP
        ADDI    @INPUT,AR5
        NOP
        NOP
*
        ADDI    1,R5
        CMPI    @LOGFFT,R5
        BLE     LOOP
*
*       RESTORE THE REGISTER VALUES AND RETURN
*
        POP     AR5
        POP     AR4
        POP     R5
        POP     R4
        POP     FP
        RETS
```

Appendix C3. Generic Program to Do a Radix-2 Real Inverse FFT Computation on the TMS320C30

```
*
*    APPENDIX C3
*
*    GENERIC PROGRAM TO DO A RADIX-2 REAL INVERSE FFT COMPUTATION ON THE
*    TMS320C30.
*
*    THE (REAL) DATA RESIDE IN INTERNAL MEMORY. THE COMPUTATION IS DONE
*    IN-PLACE. THE BIT REVERSAL IS DONE AT THE BEGINNING OF THE PROGRAM. THE
*    INPUT DATA ARE STORED IN THE FOLLOWING ORDER:
*
*    RE(0), RE(1),..., RE(N/2), IM(N/2-1),..., IM(1)
*
*    THE TWIDDLE FACTORS ARE SUPPLIED IN A TABLE PUT IN A .DATA SECTION. THIS
*    DATA IS INCLUDED IN A SEPARATE FILE TO PRESERVE THE GENERIC NATURE OF THE
*    PROGRAM. FOR THE SAME PURPOSE, THE SIZE OF THE FFT N AND LOG2(N) ARE
*    DEFINED IN A .GLOBL DIRECTIVE AND SPECIFIED DURING LINKING. THE LENGTH OF
*    THE TABLE IS N/4 + N/4 = N/2.
*
*    AUTHOR: PANOS PAPAMICHALIS                            DECEMBER 21, 1988
*            TEXAS INSTRUMENTS
*
        .GLOBL  IFFT            ; ENTRY POINT FOR EXECUTION
        .GLOBL  N               ; FFT SIZE
        .GLOBL  M               ; LOG2(N)
        .GLOBL  SINE            ; ADDRESS OF SINE TABLE
*
        .BSS    INP,1024        ; MEMORY WITH INPUT DATA
*
        .TEXT
*
*    INITIALIZE
*
        .WORD   IFFT            ; STARTING LOCATION OF THE PROGRAM
*
        .SPACE  100             ; RESERVE 100 WORDS FOR VECTORS, ETC.
*
FFTSIZ  .WORD   N
LOGFFT  .WORD   M
SINTAB  .WORD   SINE
INPUT   .WORD   INP
*
IFFT:   LDP     FFTSIZ          ; COMMAND TO LOAD DATA PAGE POINTER
*
*    MAIN LOOP (FFT STAGES)
*
        LDI     1,IR0           ; IR0=INDEX FOR E
        LDI     3,R5            ; R5 HOLDS THE CURRENT STAGE NUMBER
        LDI     @FFTSIZ,R3
        LSH     -1,R3           ; R3=N1/2=N2
        LDI     @FFTSIZ,R4
        LSH     -2,R4           ; R4=N1/4=N4
*
*    INNER LOOP
*
LOOP    LDI     @INPUT,AR5              ; AR5 POINTS TO X(I)
        LDI     IR0,AR0
        ADDI    @SINTAB,AR0             ; AR0 POINTS TO SIN/COS TABLE
INLOP   LDI     R4,IR1                  ; IR1=N4
*
        LDI     AR5,AR1
        ADDI    1,AR1                   ; AR1 POINTS TO X(I1)=X(I+J)
        LDI     AR1,AR3
        ADDI    R3,AR3                  ; AR3 POINTS TO X(I3)=X(I+J+N2)
        LDI     AR3,AR2
        SUBI    2,AR2                   ; AR2 POINTS TO X(I2)=X(I-J+N2)
        ADDI    R3,AR2,AR4              ; AR4 POINTS TO X(I4)=X(I-J+N1)
*
        NOP     *++AR5(IR1)             ; POINT TO X(I+N4)
        ADDF    *-AR5(IR1),*++AR5(IR1),R0
        SUBF    *++AR5(IR1),*-AR5(IR1),R1
        STF     R0,*-AR5(IR1)           ; X(I)=X(I)+X(I+N2)
        STF     R1,*++AR5(IR1)          ; X(I+N2)=X(I)-X(I+N2)
||      LDF     *AR5,R0
        MPYF    2.0,R0
        STF     R0,*-AR5(IR1)           ; X(I+N4)=2*X(I+N4)
||      LDF     *++AR5(IR1),R1
        MPYF    -2.0,R1
        STF     R1,*AR5++(IR1)          ; X(I+N4+N2)=-X(I+N4+N2)*2
*
*    INNERMOST LOOP
*
        LDI     @FFTSIZ,IR1
        LSH     -2,IR1                  ; IR1=SEPARATION BETWEEN SIN/COS TBLS
        LDI     R4,RC
        SUBI    2,RC                    ; REPEAT N4-1 TIMES
*
        RPTB    BLK3
        SUBF    *AR2,*AR1,R1            ; R1=T1=X(I1)-X(I2)
        ADDF    *AR2,*AR1,R0
        MPYF    R1,*++AR0(IR1),R0       ; R0=T1*COS
||      STF     R0,*AR1++               ; X(I1)=X(I1)+X(I2)
        ADDF    *AR3,*AR4,R2            ; R2=T2=X(I3)+X(I4)
        SUBF    *AR3,*AR4,R6
        MPYF    R2,*AR0,R6              ; R6=T2*SIN
||      STF     R6,*AR2--               ; X(I2)=X(I4)-X(I3)
        SUBF    R6,R0
        MPYF    R2,*++AR0(IR1),R6       ; R6=T2*COS
||      STF     R0,*AR3++               ; X(I3)=T1*COS-T2*SIN
        MPYF    R1,*AR0++(IR0),R0       ; R0=T1*SIN
        ADDF    R6,R0
BLK3    STF     R0,*AR4--               ; X(I4)=T1*SIN+T2*COS
*
        SUBI    @INPUT,AR5
        CMPI    @FFTSIZ,AR5
        BLTD    INLOP                   ; LOOP BACK TO THE INNER LOOP
        ADDI    @INPUT,AR5
        LDI     IR0,AR0
        ADDI    @SINTAB,AR0             ; AR0 POINTS TO SIN/COS TABLE
```

```
*
        ADDI    1,R5
        CMPI    @LOGFFT,R5
        BLED    LOOP
        LSH     1,IR0               ; E=E*2
        LSH     -1,R4               ; N4=N4/2
        LSH     -1,R3               ; N2=N2/2
*
*   LAST PASS OF THE MAIN LOOP
*
        LDI     @INPUT,AR0          ; AR0 POINTS TO X(I)
        LDI     2,IR0               ; IR0=2=N2
        LDI     @FFTSIZ,RC
        LSH     -2,RC               ; REPEAT N/4 TIMES
        SUBI    1,RC                ; RC SHOULD BE ONE LESS THAN DESIRED #
*
        LDF     *+AR0(IR0),R0       ; R0=X(I+2)
        RPTB    BLK2
        ADDF    R0,*AR0++(IR0),R1   ; R1=X(I)+X(I+2)
        SUBF    R0,*-AR0(IR0),R1    ; R1=X(I)-X(I+2)
::      STF     R1,*-AR0(IR0)       ; X(I)=X(I)+X(I+2)
        STF     R1,*AR0++           ; X(I+2)=X(I)-X(I+2)
::      LDF     *-AR0,R1
        MPYF    2.0,R1              ; R1=2.0*X(I+1)
        STF     R1,*-AR0(IR0)       ; X(I+1)=2.0*X(I+1)
::      LDF     *AR0++,R1
        MPYF    -2.0,R1             ; R1=-2.0*X(I+3)
BLK2    STF     R1,*-AR0            ; X(I+3)=-2.0*X(I+3)
::      LDF     *+AR0(IR0),R0       ; R0=X(I+4+2)
*
*   LENGTH-TWO BUTTERFLIES
*
        LDI     @INPUT,AR0          ; AR0 POINTS TO X(I)
        LDI     @FFTSIZ,RC
        LSH     -1,RC               ; REPEAT N/2 TIMES
        SUBI    1,RC                ; RC SHOULD BE ONE LESS THAN DESIRED #
*
        RPTB    BLK1
        ADDF    *+AR0,*AR0++,R0     ; R0=X(I)+X(I+1)
        SUBF    *AR0,*-AR0,R1       ; R1=X(I)-X(I+1)
BLK1    STF     R0,*-AR0            ; X(I)=X(I)+X(I+1)
::      STF     R1,*AR0++           ; X(I+1)=X(I)-X(I+1)
*
*   DO THE BIT REVERSING AT THE END
*
        LDI     @FFTSIZ,RC          ; RC=N
        SUBI    1,RC                ; RC SHOULD BE ONE LESS THAN DESIRED #
        LDI     @FFTSIZ,IR0
        LSH     -1,IR0              ; IR0=HALF THE SIZE OF FFT=N/2
        LDI     @INPUT,AR0
        LDI     @INPUT,AR1
*
        RPTB    BITRV
        CMPI    AR1,AR0             ; XCHANGE LOCATIONS ONLY
        BGE     CONT                ; IF AR0<AR1
        LDF     *AR0,R0
::      LDF     *AR1,R1
        STF     R0,*AR1
::      STF     R1,*AR0
CONT    NOP     *AR0++
BITRV   NOP     *AR1++(IR0)B
*
END     BR      END                 ; BRANCH TO ITSELF AT THE END
        .END
```

Appendix D. Discrete Hartley Transform

Appendix D1. Generic Program to Do a Radix-2 Hartley Transform on the TMS320C30

```
*     APPENDIX D1
*
*     GENERIC PROGRAM TO DO A RADIX-2 HARTLEY TRANSFORM ON THE TMS320C30.
*
*     THE PROGRAM IS TAKEN FROM THE PAPER BY SORENSEN ET AL., OCT 1985 ISSUE
*     OF THE TRANSACTIONS ON ASSP.
*
*     THE (REAL) DATA RESIDE IN INTERNAL MEMORY. THE COMPUTATION IS DONE
*     IN-PLACE. THE BIT-REVERSAL IS DONE AT THE BEGINNING OF THE PROGRAM.
*
*     THE TWIDDLE FACTORS ARE SUPPLIED IN A TABLE PUT IN A .DATA SECTION. THIS
*     DATA IS INCLUDED IN A SEPARATE FILE TO PRESERVE THE GENERIC NATURE OF THE
*     PROGRAM. FOR THE SAME PURPOSE, THE SIZE OF THE FHT N AND LOG2(N) ARE
*     DEFINED IN A .GLOBL DIRECTIVE AND SPECIFIED DURING LINKING. THE LENGTH OF
*     THE TABLE IS N/4 + N/4 = N/2.
*
*     AUTHOR: PANOS PAPAMICHALIS                          DECEMBER 14, 1988
*             TEXAS INSTRUMENTS
*
         .GLOBL   FHT                  ; ENTRY POINT FOR EXECUTION
         .GLOBL   N                    ; FHT SIZE
         .GLOBL   M                    ; LOG2(N)
         .GLOBL   SINE                 ; ADDRESS OF SINE TABLE
*
         .BSS     INP,1024             ; MEMORY WITH INPUT DATA
*
         .TEXT
*
*     INITIALIZE
*
         .WORD    FHT                  ; STARTING LOCATION OF THE PROGRAM
*
         .SPACE   100                  ; RESERVE 100 WORDS FOR VECTORS, ETC.
*
FHTSIZ   .WORD    N
LOGFHT   .WORD    M
SINTAB   .WORD    SINE
INPUT    .WORD    INP
*
FHT:     LDP      FHTSIZ               ; COMMAND TO LOAD DATA PAGE POINTER
*
*     DO THE BIT REVERSING AT THE BEGINNING
*
         LDI      @FHTSIZ,RC           ; RC=N
         SUBI     1,RC                 ; RC SHOULD BE ONE LESS THAN DESIRED #
         LDI      @FHTSIZ,IR0
         LSH      -1,IR0               ; IR0=HALF THE SIZE OF FHT=N/2
         LDI      @INPUT,AR0
         LDI      @INPUT,AR1
*
         RPTB     BITRV
         CMPI     AR1,AR0              ; XCHANGE LOCATIONS ONLY
         BGE      CONT                 ;   IF AR0<AR1
         LDF      *AR0,R0
||       LDF      *AR1,R1
         STF      R0,*AR1
||       STF      R1,*AR0
CONT     NOP      *AR0++
BITRV    NOP      *AR1++(IR0)B
*
*     LENGTH-TWO BUTTERFLIES
*
         LDI      @INPUT,AR0           ; AR0 POINTS TO X(I)
         LDI      IR0,RC               ; REPEAT N/2 TIMES
         SUBI     1,RC                 ; RC SHOULD BE ONE LESS THAN DESIRED #
*
         RPTB     BLK1
         ADDF     *+AR0,*AR0++,R0      ; R0=X(I)+X(I+1)
         SUBF     *AR0,*-AR0,R1        ; R1=X(I)-X(I+1)
BLK1     STF      R0,*-AR0             ; X(I)=X(I)+X(I+1)
||       STF      R1,*AR0++            ; X(I+1)=X(I)-X(I+1)
*
*     FIRST PASS OF THE DO-30 LOOP (STAGE K=2 IN DO-20 LOOP)
*
         LDI      @INPUT,AR0           ; AR0 POINTS TO X(J)
         LDI      2,IR0                ; IR0=2=N2
         LDI      @FHTSIZ,RC
         LSH      -2,RC                ; REPEAT N/4 TIMES
         SUBI     1,RC                 ; RC SHOULD BE ONE LESS THAN DESIRED #
*
         RPTB     BLK2
         ADDF     *+AR0(IR0),*AR0++(IR0),R0    ; R0=X(J)+X(L2)
         SUBF     *AR0,*-AR0(IR0),R1   ; R1=X(J)-X(L2)
         STF      R0,*-AR0(IR0)        ; X(J)=X(J)+X(L2)
||       LDF      *+AR0,R0             ; R0=X(L4)
         ADDF     R0,*-AR0,R1          ; R1=X(L3)+X(L4)
||       STF      R1,*AR0++            ; X(L2)=X(J)-X(L2)
         SUBF     R0,*-AR0(IR0),R1     ; R1=X(L3)-X(L4)
||       STF      R1,*-AR0(IR0)        ; X(L3)=X(L3)+X(L4)
BLK2     STF      R1,*AR0++            ; X(L4)=X(L3)-X(L4)
*
*     MAIN LOOP (FHT STAGES)
*
         LDI      @FHTSIZ,IR0
         LSH      -2,IR0               ; IR0=INDEX FOR E
         LDI      3,R5                 ; R5 HOLDS THE CURRENT STAGE NUMBER
         LDI      1,R4                 ; R4=N4
         LDI      2,R3                 ; R3=N2
LOOP     LSH      -1,IR0               ; E=E/2
         LSH      1,R4                 ; N4=2*N4
         LSH      1,R3                 ; N2=2*N2
*
*     INNER LOOP (DO-30 LOOP IN THE PROGRAM)
*
         LDI      @INPUT,AR5           ; AR5 POINTS TO X(J)
INLOP    LDI      IR0,AR0
         ADDI     @SINTAB,AR0          ; AR0 POINTS TO SIN/COS TABLE
         LDI      R4,IR1               ; IR1=N4
```

```
*
        LDI     AR5,AR1
        ADDI    1,AR1               ; AR1 POINTS TO X(L1)=X(J+I-1)
        LDI     AR1,AR3
        ADDI    R3,AR3              ; AR3 POINTS TO X(L3)=X(L1+N2)
        LDI     AR3,AR2
        SUBI    2,AR2               ; AR2 POINTS TO X(L2)=X(J-I+1+N2)
        ADDI    R3,AR2,AR4          ; AR4 POINTS TO X(L4)=X(L2+N2)
*
        LDF     *AR5++(IR1),R0      ; R0=X(J)
        ADDF    *+AR5(IR1),R0,R1    ; R1=X(J)+X(L2)
        SUBF    R0,*++AR5(IR1),R0   ; R0=-X(J)+X(L2)
||      STF     R1,*-AR5(IR1)       ; X(J)=X(J)+X(L2)
        NEGF    R0                  ; R0=X(J)-X(L2)
        STF     R0,*AR5             ; X(L2)=X(J)-X(L2)
||      LDF     *+AR5(IR1),R0       ; R0=X(L4)
        ADDF    R0,*-AR5(IR1),R1    ; R1=X(L3)+X(L4)
        SUBF    R0,*-AR5(IR1),R1    ; R1=X(L3)-X(L4)
||      STF     R1,*-AR5(IR1)       ; X(L3)=X(L3)+X(L4)
        STF     R1,*+AR5(IR1)       ; X(L4)=X(L3)-X(L4)
*
*   INNERMOST LOOP
*
        LDI     @FHTSIZ,IR1
        LSH     -2,IR1              ; IR1=SEPARATION BETWEEN SIN/COS TBLS
        LDI     R4,RC
        SUBI    2,RC                ; REPEAT N4-1 TIMES
*
        RPTB    BLK3
        MPYF    *AR3,*+AR0(IR1),R0  ; R0=X(L3)*COS
        MPYF    *AR4,*AR0,R1        ; R1=X(L4)*SIN
        MPYF    *AR4,*+AR0(IR1),R1  ; R1=X(L4)*COS
||      ADDF    R0,R1,R2            ; R2=X(L3)*COS+X(L4)*SIN=T1
        MPYF    *AR3,*AR0++(IR0),R0 ; R0=X(L3)*SIN
        SUBF    R1,R0,R0            ; R0=X(L3)*SIN-X(I4)*COS=T2
        SUBF    R0,*AR2,R1          ; R1=X(L2)-T2
        ADDF    *AR2,R0,R1          ; R1=X(L2)+T2
||      STF     R1,*AR4--           ; X(L4)=X(L2)-T2
        ADDF    *AR1,R2,R1          ; R1=X(L1)+T1
||      STF     R1,*AR2--           ; X(L2)=X(L2)+T2
        SUBF    R2,*AR1,R1          ; R1=X(L1)-T1
||      STF     R1,*AR1++           ; X(L1)=X(L1)+T1
BLK3    STF     R1,*AR3++           ; X(L3)=X(L1)-T1
*
        SUBI    @INPUT,AR5
        ADDI    R3,AR5              ; AR5=I+N1
        CMPI    @FHTSIZ,AR5
        BLTD    INLOP               ; LOOP BACK TO THE INNER LOOP
        ADDI    @INPUT,AR5
        NOP
        NOP
*
        ADDI    1,R5
        CMPI    @LOGFHT,R5
        BLE     LOOP
*
END     BR      END                 ;BRANCH TO ITSELF AT THE END
        .END
```

Appendix E. Discrete Cosine Transform

Appendix E1. A Fast Cosine Transform

```
*
*   APPENDIX E1
*
*   A FAST COSINE TRANSFORM
*
*   BASED ON THE ALGORITHM OUTLINED BY BYEONG GI LEE IN HIS ARTICLE, FCT - A
*   FAST COSINE TRANSFORM, PUBLISHED IN THE PROCEEDINGS OF THE IEEE INTER-
*   NATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, SAN
*   DIEGO, CA, 19-21 MARCH 1984, P 28A.3/1-4 VOL. 2, (CH1954-5/84/0000-0299).
*
*   LEE'S ALGORITHM HAS BEEN MODIFIED TO ALLOW NATURAL ORDER TIME DOMAIN
*   COEFFICIENTS RATHER THAN THE LESS ORDERED INPUT SUGGESTED IN HIS ARTICLE.
*
*   THE FREQUENCY DOMAIN COEFFICIENTS ARE IN BIT REVERSE ORDER. THIS IS AN IN
*   PLACE CALCULATION.
*
*   AUTHOR: PAUL WILHELM
*
*
        .global   FCT             ; FAST COSINE TRANSFORM ENTRY POINT.
        .global   M               ; LENGTH OF DATA ENTRY.
        .global   COS_TAB         ; TABLE OF COSINE COEFFICIENTS.
        .global   COEFF           ; TABLE OF INPUT DATA.
*
        .text
*
FCTSIZE .word     M
_COS    .word     COS_TAB
_DATA   .word     COEFF
*
FCT:
        LDI       @FCTSIZE,AR0    ; LOAD DATA LENGTH.
        LDI       @FCTSIZE,BK     ; SET BLOCK SIZE FOR CIRCULAR
*                                 ;   ADDRESSING.
        LDI       @_DATA,AR6      ; LOAD DATA POINTER.
        LDI       @_COS,AR7       ; LOAD COSINE TABLE POINTER.
        LDI       AR0,IR1         ; INITIALIZE INDEX REGISTERS FOR FIRST
        LDI       -1,IR0          ;   BUTTERFLY SERIES.
        LDI       AR6,AR1         ; INITIALIZE DATA POINTERS.
        ADDI3     AR6,AR0,AR2
        SUBI      1,AR2
        LSH3      IR0,AR0,AR3
        LDI       1,AR5           ; INITIALIZE 2'S POWER COUNTER.
        ADDI      AR6,AR3         ; FINISH DATA POINTER INITIALIZATION.
        ADDI3     IR0,AR3,AR4
        ADDI3     IR0,AR5,RC      ; RC SHOULD BE ONE LESS THAN COUNT
*                                 ;   DESIRED.
*
*   FIRST LOOP SERIES
*
*   THIS LOOP SERIES DOES ALL THE BUTTERFLY STAGES EXCEPT THE FINAL ONE.
*
        RPTB      END_CENTER_LOOP
*
OUTSIDE_LOOP:                     ; TWO BUTTERFLIES ARE CALCULATED AT
MIDDLE_LOOP:                      ;   THE SAME TIME.
*
        LDF       *AR2,R2         ; GET LOWER HALF OF EACH BUTTERFLY.
||      LDF       *AR3,R3         ;   (THIS ALLOWS FOR MORE PARALLEL
*                                 ;   COMMANDS LATER)
        SUBF3     *AR3,*AR4,R1    ; SUBTRACT SECOND BUTTERFLY DATA.
        SUBF3     *AR2,*AR1,R0    ; SUBTRACT FIRST BUTTERFLY DATA.
        MPYF3     R1,*++AR7,R1    ; MULTIPLY 2ND SUBTRACTION RESULT BY
||      ADDF3     R3,*AR4,R3      ;   COSINE COEFFICIENT. ADD SECOND
*                                 ;   BUTTERFLY DATA.
        MPYF3     R0,*--AR7,R0    ; MULTIPLY 1ST SUBTRACTION RESULT BY
||      ADDF3     R2,*AR1,R2      ;   COSINE COEFFICIENT. ADD FIRST
*                                 ;   BUTTERFLY DATA.
        STF       R1,*AR2++(IR1)% ; SAVE 2ND MULTIPLY RESULT IN LOWER
||      STF       R3,*AR4++(IR1)% ;   HALF IF BUTTERFLY. SAVE 2ND
*                                 ;   ADDITION IN UPPER 2ND BUTTERFLY.
END_CENTER_LOOP:
*
        STF       R0,*AR3++(IR1)% ; SAVE 1ST MULTIPLY IN LOWER HALF OF
||      STF       R2,*AR1++(IR1)% ;   2ND BUTTERFLY. SAVE 1ST ADDITION
*                                 ;   IN UPPER 1ST BUTTERFLY.
*
*   END OF CENTER LOOP OF FIRST LOOP SERIES.
*
        ADDI3     IR0,AR5,RC      ; UPDATE REPEAT COUNTER FOR NEXT BLOCK
*                                 ;   REPEAT.
        ADDF3     *AR3++,*AR2--,R0 ; UPDATE DATA POINTERS.
        CMPI      AR3,AR2         ; HAVE BUTTERFLIES BEEN COMPLETED?
        BGTD      MIDDLE_LOOP     ; DELAYED BRANCH, IF NOT.
        ADDF3     *AR1++,*AR4--,R0 ; UPDATE FINAL TWO POINTERS FOR NEXT
*                                 ;   REPEAT.
        ADDI      2,AR7           ; UPDATE COSINE COEFFICIENT POINTER.
        OR        0100H,ST        ; SET REPEAT MODE. (FASTER THAN USING
*                                 ;   RPTB WHEN START AND END ADDRESS
*                                 ;   ARE STILL GOOD)
*
*   DELAY BRANCH FROM HERE TO MIDDLE_LOOP.
*
        LSH       -1,IR1          ; UPDATE INDEX REGISTER. (DIVIDE BY 2)
        LDI       AR6,AR1         ; REINITIALIZE DATA POINTERS.
        ADDI      IR0,AR6,AR2
        ADDI      IR1,AR2
        CMPI      2,IR1           ; IS FIRST BUTTERFLY SERIES COMPLETE?
        BGTD      OUTSIDE_LOOP    ; DELAY BRANCH, IF NOT.
        LSH       1,AR5           ; MULTIPLY 2'S POWER COUNTER BY 2.
        SUBI3     IR0,AR4,AR3     ; CONTINUE REINITIALIZING DATA
*                                 ;   POINTERS.
        ADDI3     IR0,AR5,RC      ; SET REPEAT COUNTER FOR REPEAT BLOCK.
*
*   END OF FIRST LOOP SERIES.
*
*   FINAL BUTTERFLY STAGE LOOP.
*
```

```
*    INCLUDES LAST BUTTERFLIES AND FIRST STAGE OF BIT REVERSE ADDITIONS.
*
        LDI     4,IR1                ; INITIALIZE INDEX REGISTER.
        ADDI    1,AR3                ; SET UP DATA POINTERS.
        LSH     -1,AR5
        ADDI    3,AR4
        ADDI3   IR0,AR5,RC           ; INITIALIZE REPEAT COUNTER.
        MPYF3   *AR7,*++AR7,R4       ; CALCULATE (2/M)*COS(PI/4).
*                                    ;   (I.E.-> (SQRT(2))/M THIS VALUE IS
*                                    ;   CALLED, S, BELOW.)
        RPTB    END_2ND_LOOP         ; TWO BUTTERFLIES ARE CALCULATED PER
*                                    ;   LOOP.
*
        SUBF3   *AR2,*AR1,R0         ; SUBTRACT 1ST BUTTERFLY DATA.
        SUBF3   *AR4,*AR3,R1         ; SUBTRACT 2ND BUTTERFLY DATA.
        MPYF3   R0,R4,R0             ; MULTIPLY 1ST SUBTRACTION RESULT
||      ADDF3   *AR3++(IR1),*AR4++(IR1),R3    ; BY S. ADD 2ND BUTTERFLY
*                                    ;   DATA.
        MPYF3   R1,R4,R1             ; MULTIPLY 2ND SUBTRACTION RESULT
||      ADDF3   *AR1++(IR1),*AR2++(IR1),R2    ;   BY S. ADD 1ST BUTTERFLY
*                                    ;   DATA.
        MPYF3   R3,*++AR7,R3         ; MULTIPLY 2ND ADDITION RESULT BY
||      STF     R0,*-AR2(IR1)        ;   7071. SAVE 1ST SUBTRACTION IN
                                     ;   LOWER 1/2 OF 1ST BUTTERFLY.
        MPYF3   R2,*++AR7,R2         ; MULTIPLY 1ST ADDITION RESULT BY
||      STF     R1,*-AR4(IR1)        ;   .7071 SAVE 2ND SUBTRACTION IN
*                                    ;   LOWER 1/2 OF 2ND BUTTERFLY.
        ADDF3   R3,R1,R3             ; ADD 2ND SUBTRACTION MULTIPLY TO 2ND
*                                    ;   ADDITION MULTIPLY.
        STF     R2,*-AR1(IR1)        ; SAVE 1ST ADDITION MULTIPLY IN UPPER
*                                    ;   1/2 OF BUTTERFLY.
*
END_2ND_LOOP:
*
        STF     R3,*-AR3(IR1)        ; SAVE 2ND ADDITION MULTPLY IN UPPER
*                                    ;   1/2 OF UPPER BUTTERFLY.
*
*    END OF FINAL BUTTERFLY STAGE LOOP.
*
*    BIT REVERSE ADDITION LOOP SERIES.
*
*    THIS LOOP SERIES DOES ALL OF THE BIT REVERSE ADDITIONS AT THE END OF FAST
*    COSINE TRANSFORM.
*
        LDI     2,IR0                ; INITIALIZE INDEX REGISTERS AND DATA
        LDI     AR6,AR1              ;   POINTERS FOR FINAL ADDITION
        ADDI    4,AR1                ;   SERIES.
        LDI     AR1,AR2
        LDI     8,IR1
*
LAST_OUTSIDE_LOOP:
*
        LDI     AR2,AR4              ; UPDATE POINTERS AND COUNTERS.
        LSH     -1,AR5
        LDI     AR5,RC               ; SET UP REPEAT COUNTER.
        ADDF3   *AR2++(IR0)B,*AR4++(IR0)B,R0    ; DATA POINTER UPDATE.
        LDI     AR1,R4               ; USE INITIAL AR1 VALUE AS INNER LOOP
*                                    ;   CONTROL.
        SUBI    1,RC
        NOP     *AR4++(IR0)B         ; CONTINUE UPDATING POINTERS.
        LDI     AR2,AR3
*
        RPTB    END_INSIDE           ; TWO ADDITIONS ARE DONE IN EACH LOOP.
*
LAST_INSIDE_LOOP:
*
        ADDF3   *AR1,*AR2++(IR1)%,R0   ; ADD FIRST TWO DATA.
        ADDF3   *AR3,*AR4++(IR1)%,R1   ; ADD SECOND TWO DATA.
        STF     R0,*AR1++(IR1)%      ; SAVE FIRST ADDITION.
*
END_INSIDE:
*
        STF     R1,*AR3++(IR1)%      ; SAVE SECOND ADDITION.
*
*    END OF INSIDE LOOP FOR LAST LOOP SERIES.
*
        ADDF3   *AR1++(IR0)B,*AR2++(IR0)B,R0    ; UPDATE DATA POINTERS.
        ADDF3   *AR3++(IR0)B,*AR4++(IR0)B,R0
        ADDF3   *AR3++(IR0)B,*AR4++(IR0)B,R0
        ADDF3   *AR1++(IR0)B,*AR2++(IR0)B,R0
        CMPI    R4,AR4               ; IS THIS LOOP COMPLETE?
        BNED    LAST_INSIDE_LOOP     ; DELAYED BRANCH, IF NOT.
        LDI     AR5,RC               ; SET UP REPEAT COUNTER.
        SUBI    1,RC
        OR      0100H,ST             ; SET REPEAT MODE.
*
*    BRANCH DELAYED TO LAST_INSIDE_LOOP.
*
        RPTB    LAST_BLOCK           ; SINCE THERE ARE AN ODD NUMBER OF
        ADDF3   *AR1,*AR2++(IR1)%,R0   ;   ADDITIONS, THE FINAL ONES ARE
*                                    ;   DONE NOW.
*
LAST_BLOCK:
*
        STF     R0,*AR1++(IR1)%      ; SAVE ADDITION.
*
*    END OF LAST REPEAT BLOCK.
*
        LSH     1,IR0                ; MULTIPLY IR0 BY 2.
        ADDI    IR0,R4               ; UPDTEE INNER LOOP CONTROL REGISTER.
        CMPI    1,AR5                ; ARE CALCULATIONS COMPLETE ?
        BGTD    LAST_OUTSIDE_LOOP    ; DELAYED BRANCH, IF NOT.
        LDI     R4,AR2               ; UPDATE DATA POINTERS.
        LDI     R4,AR1
        LSH     1,IR1                ; MULTIPLY IR1 BY 2.
*
*    DELAYED BRANCH TO LAST_OUTSIDE_LOOP.
*
```

```
*    END OF LAST LOOP SERIES.
*
*    MULTIPLY COEFFICIENT ZERO BY .5, IF NOT ZERO.
*
        LDF     *AR6,R0         ; SET ZERO FLAG IF *AR6 = 0.
        BEQD    DONT_STORE      ; IF COEFFICIENT IS ZERO, DON'T DO
*                               ;   THIS.
        LSH     24,AR5          ; USE INTEGER MATH FOR FLOAT DIVIDE
*                               ;   BY 2.
        SUBI3   AR5,*AR6,AR1
        NOP
*
*    DELAYED BRANCH FROM HERE IF VALUE IS NOT TO BE STORED.
*
        STI     AR1,*AR6        ; STORE, IF EXPONENT WASN'T -128.
*
DONT_STORE:
*
        RETS
```

Appendix E2. A Fast Cosine Transform (Inverse Transform)

```
*
*    APPENDIX E2
*
*    A FAST COSINE TRANSFORM (INVERSE TRANSFORM)
*
*    BASED ON THE ALGORITHM OUTLINED BY BYEONG GI LEE IN HIS ARTICLE, FCT - A
*    FAST COSINE TRANSFORM, PUBLISHED IN THE PROCEEDINGS OF THE IEEE Inter-
*    NATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, SAN
*    DIEGO, CA, 19-21 MARCH 1984, P 28A.3/1-4 VOL 2., (CH1954-5/84/0000-0299).
*
*    LEE'S ALGORITHM HAS BEEN MODIFIED TO ALLOW NATURAL ORDER TIME DOMAIN
*    COEFFICIENTS.
*
*    THE FREQUENCY DOMAIN COEFFICIENTS ARE IN BIT REVERSE ORDER. THIS IS AN IN
*    PLACE CALCULATION.
*
*    AUTHOR: PAUL WILHELM
*
        .global  IFCT                ; INVERSE FAST COSINE TRANSFORM ENTRY
*                                    ;   POINT.
        .global  M                   ; LENGTH OF ARRAY TO BE TRANSFORMED.
        .global  COEFF               ; TABLE OF COSINE COEFFICIENTS.
        .global  COS_TAB             ; TABLE OF ARRAY DATA TO BE
*                                    ;   TRANSFORMED.
*
        .text
*
FCTSIZE .word    M
_DATA   .word    COEFF
_COS    .word    COS_TAB
*
IFCT:
        LDI      @FCTSIZE,AR0        ; LOAD ARRAY SIZE.
        LDI      @FCTSIZE,BK         ; LOAD BLOCK SIZE FOR CIRCULAR
*                                    ;   ADDRESSING
        LDI      @_DATA,AR6          ; LOAD POINTER TO DATA TABLE.
        LDI      @_COS,AR7           ; LOAD POINTER TO COSINE TABLE.
        ADDI     AR0,AR7             ; POINT TO LAST COSINE VALUE IN TABLE.
        SUBI     2,AR7
        LDI      AR0,IR0             ; INITIALIZE INDEX REGISTERS FOR BIT
        LSH      -2,IR0              ;   REVERSED ADDITION SEQUENCE.
        LDI      AR0,IR1
        LDI      AR6,AR1             ; INITIALIZE DATA POINTERS.
        ADDI     IR0,AR1
*
*    START OF BIT REVERSED ADDITION LOOP SERIES.
*
OUTSIDE:                             ; TOP OF OUTSIDE LOOP FOR BIT REVERSED
*                                    ;   ADDITIONS.
        ADDI     IR0,AR1             ; UPDATE DATA POINTERS AND REPEAT
*                                    ;   COUNTER.
        LDI      AR1,AR2
        LDI      IR0,RC
        SUBI     2,RC
        NOP      *AR2++(IR0)B
        ADDF3    *AR1++(IR0)B,*AR2++(IR0)B,R0   ; FIND FIRST SUM. (MAKES
        LDI      AR1,AR3             ;   MIDDLE LOOP MORE EFFICIENT)
        LDI      AR2,AR4
        LDI      AR1,AR5
        ADDF3    *AR3++(IR0)B,*AR4++(IR0)B,R1   ; DUMMY ADD TO UPDATE
*                                    ;   POINTERS.
        LSH      -1,IR0              ; UPDATE INDEX REGISTER.
*
        RPTB     END_CENTER          ; TOP OF INNER MOST LOOP.
*
MIDDLE:                              ; TOP OF MIDDLE LOOP.
*
        LDF      *AR3,R3             ; GET UPPER HALF OF SECOND ADDITION.
        ADDF3    *AR1,*AR2++(IR0)B,R1   ; DO FIRST ADDITION.
        STF      R0,*AR1++(IR0)B     ; STORE ADDITION DONE THE LAST LOOP OR
*                                    ;   WHEN INITIALIZATION WAS DONE ABOVE
END_CENTER:
*
        ADDF3    R3,*AR4++(IR0)B,R0  ; DO SECOND ADDITION.
||      STF      R1,*AR3++(IR0)B     ; STORE FIRST ADDITION.
*
*    END OF INNER MOST LOOP.
*
        ADDF3    *AR3++(IR1)%,*AR4++(IR1)%,R2   ; DUMMY ADD TO UPDATE
*                                    ;   POINTERS.
        LDF      *AR3++(IR0)B,R3     ; GET VALUE FOR LAST ADDITION.
||      LDF      *AR2++(IR0)B,R2     ; DUMMY ADD TO UPDATE POINTER.
        ADDF3    R3,*AR4++(IR0)B,R0  ; DO LAST ADDITION.
||      STF      R0,*AR1++(IR0)B     ; STORE NEXT TO LAST ADDITION.
        ADDF3    *AR1++(IR1)%,*AR2++(IR1)%,R2   ; DUMMY ADD TO UPDATE
*                                    ;   POINTERS.
        LDI      IR0,RC              ; UPDATE REPEAT COUNTER.
        CMPI     AR1,AR5             ; IS MIDDLE LOOP COMPLETE ?
        BNED     MIDDLE              ; IF NOT, DO DELAYED BRANCH.
        LSH      1,RC
        SUBI     2,RC
        OR       0100H,ST            ; SET REPEAT MODE.
*                                    ;   (START/STOP ADDRESSES STILL OK)
*
*    DELAY BRANCH FROM HERE TO MIDDLE.
*
        CMPI     1,IR0               ; IS OUTSIDE LOOP COMPLETE ?
        BGTD     OUTSIDE             ; IF NOT, DO DELAYED BRANCH.
        LDI      AR6,AR1             ; PREPARE TO UPDATE POINTERS AT TOP OF
*                                    ;   LOOP.
        ADDI     IR0,AR1
        LSH      -1,IR1              ; UPDATE INDEX REGISTER.
*
*    DELAY BRANCH FROM HERE TO OUTSIDE.
*
*    END OF BIT REVERSED ADDITION LOOP SERIES.
*
*    START OF CENTER BUTTERFLY LOOP.
```

```
*
*    THIS LOOP INCLUDES THE LAST BIT REVERSED ADDITION STAGE, THE FIRST
*    BUTTERFLY, AND THE COSINE MULTIPLICATIONS FOR THE SECOND BUTTERFLY
*    SERIES.
*
        SUBI    3,AR2               ; UPDATE DATA POINTER FOR THIS LOOP.
        LDI     8,IR1               ; INITIALIZE INDEX REGISTER.
        LDI     AR0,RC              ; INITIALIZE REPEAT COUNTER.
        LSH     -3,RC
        LDF     *AR7--,R7           ; GET COSINE PI/4.
        SUBI    1,RC
        LDI     RC,AR5              ; SAVE REPEAT COUNTER FOR LATER USE.
*
        RPTB    END_CENTER_LOOP     ; FOUR BUTTERFLIES ARE DONE EACH CYCLE
*                                   ;   THROUGH THIS LOOP.
*
        ADDF3   *+AR2,*AR2,R4       ; BIT REVERSED ADDITION FOR 2ND
*                                   ;   BUTTERFLY.
        MPYF3   *AR1,R7,R5          ; COSINE  PI/4 TIMES LOWER HALF OF 1ST
*                                   ;   BUTTERFLY.
        MPYF3   R7,R4,R0            ; COSINE PI/4 TIMES LOWER HALF OF 2ND
*                                   ;   BUTTERFLY.
||      ADDF3   *AR4,*-AR4,R3       ; BIT REVERSED ADDITION FOR 4TH
*                                   ;   BUTTERFLY.
        ADDF3   R5,*-AR1,R4         ; ADD UPPER HALF OF 1ST BUTTERFLY.
        MPYF3   *+AR7,R3,R1         ; COSINE PI/4 TIMES LOWER HALF OF 4TH
*                                   ;   BUTTERFLY.
||      ADDF3   R0,*AR2,R2          ; ADD UPPER HALF OF 2ND BUTTERFLY.
        SUBF3   R5,*-AR1,R5         ; SUBTRACT LOWER HALF OF 1ST
*                                   ;   BUTTERFLY.
        MPYF3   *-AR7,R2,R0         ; MULTIPLY UPPER HALF OF 2ND BUTTERFLY
*                                   ;   BY COSINE COEFFIEIENT.
||      SUBF3   R0,*AR2,R2          ; SUBTRACT LOWER HALF OF 2ND
*                                   ;   BUTTERFLY.
        STF     R4,*-AR1            ; STORE UPPER HALF OF 1ST BUTTERFLY.
||      STF     R5,*AR1++(IR1)%     ; STORE LOWER HALF OF 1ST BUTTERFLY.
        STF     R0,*+AR2            ; STORE LOWER HALF OF 2ND BUTTERFLY.
        MPYF3   *AR3,R7,R4          ; COSINE PI/4 TIMES LOWER HALF OF 3RD
*                                   ;   BUTTERFLY.
        MPYF3   *AR7,R2,R0          ; MULTIPLY LOWER HALF OF 2ND BUTTERFLY
*                                   ;   BY COSINE COEFFICIENT
||      SUBF3   R1,*-AR4,R3         ; SUBTRACT LOWER HALF OF 4TH
*                                   ;   BUTTERFLY.
        ADDF3   R4,*-AR3,R5         ; ADD UPPER HALF OF 3RD BUTTERFLY.
        MPYF3   *AR7,R3,R1          ; MULTIPLY ;OWER HALF OF 4TH BUTTERFLY
*                                   ;   BY COSINE COEFFICIENT
||      ADDF3   R1,*-AR4,R3         ; ADD UPPER HALF OF 4TH BUTTERFLY.
        SUBF3   R4,*-AR3,R4         ; SUBTRACT LOWER HALF OF 3RD
*                                   ;   BUTTERFLY.
        MPYF3   *-AR7,R3,R1         ; MULTIPLY UPPER HALF OF 4TH BUTTERFLY
*                                   ;   BY COSINE COEFFICIENT.
||      STF     R1,*-AR4            ; STORE UPPER HALF OF 4TH BUTTERFLY.
        STF     R0,*AR2++(IR1)%     ; STORE UPPER HALF OF 2ND BUTTERFLY.
||      STF     R5,*-AR3            ; STORE UPPER HALF OF 3RD BUTTERFLY.
*
END_CENTER_LOOP:
*
        STF     R1,*AR4++(IR1)%     ; STORE LOWER HALF OF 4TH BUTTERFLY.
||      STF     R4,*AR3++(IR1)%     ; STORE LOWER HALF OF 3RD BUTTERFLY.
*
*    END OF CENTER BUTTERFLY LOOP.
*
*    START NEXT TO LAST LOOP SERIES.
*
*    THIS SERIES OF LOOPS DOES ALL BUT THE LAST BUTTERFLY STAGE. ALL THE
*    COSINE COEFFICIENT MULTIPLICATIONS ARE DONE, INCLUDING THE MULTI-
*    PLICATIONS FOR THE LAST BUTTERFLY STAGE. (THIS PROGRAM FLOW ALLOWS FOR
*    FAST EXECUTION.)
*
        SUBI    2,AR7               ; UPDATE COSINE COEFFICIENT POINTER.
        SUBI    1,AR4               ; UPDATE DATA POINTER.
        LDI     AR5,RC              ; RELOAD REPEAT COUNTER.
        LDF     *AR7--,R5           ; GET COSINE COEFFICIENTS.
        LDF     *AR7--,R4
*
        RPTB    END_NTL             ; TWO BUTTERFLIES ARE CALCULATED PER
*                                   ;   CYCLE THROUGH THE INNER LOOP.
*
NTL_LOOP:
*
        SUBF3   *AR4,*AR3,R6        ; SUBTRACT LOWER HALF OF 2ND
*                                   ;   BUTTERFLY.
        ADDF3   *AR4,*AR3,R7        ; ADD UPPER HALF OF 2ND BUTTERFLY.
        MPYF3   R5,R6,R0            ; MULTIPLY UPPER HALF OF 2ND BUTTERFLY
*                                   ;   BY COSINE COEFFICIENT.
||      ADDF3   *AR2,*AR1,R2        ; ADD UPPER HALF OF 1ST BUTTERFLY.
        MPYF3   R4,R7,R1            ; MULTIPLY LOWER HALF OF 2ND BUTTERFLY
*                                   ;   BY COSINE COEFFIEICENT.
||      SUBF3   *AR2,*AR1,R3        ; SUBTRACT LOWER HALF OF 1ST
*                                   ;   BUTTERFLY.
        STF     R0,*AR3++(IR1)%     ; STORE UPPER HALF OF 2ND BUTTERFLY.
||      STF     R2,*AR1++(IR1)%     ; STORE UPPER HALF OF 1ST BUTTERFLY.
*
END_NTL:
*
        STF     R1,*AR4++(IR1)%     ; STORE LOWER HALF OF 1ST BUTTERFLY.
||      STF     R3,*AR2++(IR1)%     ; STORE LOWER HALF OF 2ND BUTTERFLY.
*
*    END OF CENTER LOOP OF NEXT TO LAST SERIES.
*
        LDI     AR5,RC              ; RELOAD REPEAT COUNTER.
        LDF     *AR7--,R5           ; GET NEW COSINE COEFFICIENTS. (FYI-
        LDF     *AR7--,R4           ;   THE LAST TIME, THIS WILL FETCH
*                                   ;   FROM MEMORY BELOW THE COSINE
*                                   ;   TABLE.)
        CMPI    AR1,AR6             ; HAS MIDDLE LOOP BEEN COMPLETED ?
        BNED    NTL_LOOP            ; IF NOT, BRANCH DELAYED.
        ADDF3   *AR4++,*AR3--,R0    ; DUMMY ADDS TO UPDATE DATA POINTERS.
```

```
        ADDF3   *AR2++,*AR1--,R0
        OR      0100H,ST            ; SET REPEAT MODE. (START/STOP
*                                   ;   ADDRESSES ARE STILL GOOD.)
*
*   BRANCH DELAY FROM HERE TO NTL_LOOP.
*
        LDI     AR3,AR1             ; UPDATE DATA POINTERS.
        ADDI3   IR1,AR1,AR3
        LSH     1,IR1               ; UPDATE INDEX REGISTER.
        CMPI    IR1,AR0             ; IS THIS LOOP SERIES COMPLETE ?
        BGED    NTL_LOOP            ; IF NOT, BRANCH DELAYED.
        ADDI3   IR0,AR3,AR4         ; UPDATE DATA POINTER.
        LSH     -1,AR5              ; UPDATE REPEAT COUNTER.
        LDI     AR5,RC
*
*   DELAYED BRANCH FROM HERE TO NTL_LOOP.
*
*   END OF NEXT TO LAST LOOP SERIES.
*
*   START OF THE LAST LOOP.
*
*   THE LAST LOOP IS THE LAST BUTTERFLY STAGE WITHOUT THE COSINE COEFFICIENT
*   MULTIPLICATIONS, WHICH HAVE ALREADY BEEN DONE.
*
        LDI     2,IR1               ; INITIALIZE INDEX REGISTER.
        ADDI3   IR0,AR2,AR4         ; INITIALIZE DATA POINTERS.
        SUBI3   IR0,AR1,AR3
        LDI     AR0,RC              ; INITIALIZE REPEAT COUNTER.
        LSH     -2,RC
        SUBI    1,RC
*
        RPTB    END_LAST_LOOP       ; TWO BUTTERFLIES ARE DONE FOR EACH
*                                   ;   CYCLE THROUGH THE LOOP.
*
        LDF     *AR4,R0             ; GET VALUE FOR LOWER HALF OF 2ND
*                                   ;   BUTTERFLY.
        ADDF3   *AR2,*AR1,R1        ; ADD UPPER HALF OF 1ST BUTTERFLY.
        SUBF3   *AR2,*AR1,R2        ; SUBTRACT LOWER HALF OF 1ST
*                                   ;   BUTTERFLY.
        ADDF3   R0,*AR3,R3          ; ADD UPPER HALF OF 2ND BUTTERFLY.
||      STF     R1,*AR1--(IR1)      ; STORE UPPER HALF OF 1ST BUTTERFLY.
        SUBF3   R0,*AR3,R4          ; SUBTRACT LOWER HALF OF 2ND
*                                   ;   BUTTERFLY.
||      STF     R2,*AR2++(IR1)      ; STORE LOWER HALF OF 1ST BUTTERFLY.
        STF     R3,*AR3--(IR1)      ; STORE UPPER HALF OF 2ND BUTTERFLY.
*
END_LAST_LOOP:
*
        STF     R4,*AR4++(IR1)      ; STORE LOWER HALF OF 2ND BUTTERFLY.
*
*   END OF LAST LOOP, AND INVERSE COSINE TRANSFORM.
*
        RETS
        .end
```

Appendix E3. FCT Cosine Tables File

```
*
*    APPENDIX E3
*
*    FCT COSINE TABLES FILE
*
*    TO BE LINKED WITH FCT SOURCE CODE FOR 32 POINT FCT.
*
*    COEFFICIENTS ARE  1/(2 * COS(N*PI/2M)), WHERE N IS A NUMBER FROM 1 to
*    M-1. M IS THE ORDER OF THE TRANSFORM.
*
*    FOR A 32 POINT FCT, N IS IN THE FOLLOWING ORDER:
*         1, 15, 3, 13, 5, 11, 7, 9,
*         2, 14, 6, 10,
*         4, 12,
*         8
*
*    THE LAST VALUE IN THE TABLE IS 2/M.
*
*
          .global   COS_TAB
          .global   M
*
M         .set      16
*
          .data
*
COS_TAB
          .float    0.5024193
          .float    5.1011487
          .float    0.5224986
          .float    1.7224471
          .float    0.5669440
          .float    1.0606777
          .float    0.6468218
          .float    0.7881546
          .float    0.5097956
          .float    2.5629154
          .float    0.6013449
          .float    0.8999762
          .float    0.5411961
          .float    1.3065630
          .float    0.7071068
          .float    0.1250000
          .end
```

Appendix E4. Data File

```
*
*    APPENDIX E4
*
*    DATA FILE
*
          .global   COEFF
*
          .data
*
COEFF
          .float    137.0
          .float    249.0
          .float    105.0
          .float    217.0
          .float    73.0
          .float    185.0
          .float    41.0
          .float    153.0
          .float    9.0
          .float    121.0
          .float    233.0
          .float    89.0
          .float    201.0
          .float    57.0
          .float    169.0
          .float    25.0
          .end
```

Appendix F. Test Vectors, 64-Point Sine Table, Link Command File

Appendix F1. Example of a 64-Point Vector to Test the FFT Routines

```
*
*    APPENDIX F1
*
*    EXAMPLE OF A 64-POINT VECTOR TO TEST THE FFT ROUTINES

        X =

             0.2113
             0.0824
             0.7599
             0.0087
             0.8096
             0.8474
             0.4524
             0.8075
             0.4832
             0.6135
             0.2749
             0.8807
             0.6538
             0.4899
             0.7741
             0.9626
             0.9933
             0.8360
             0.7469
             0.0378
             0.4237
             0.2613
             0.2403
             0.3405
             0.1167
             0.6250
             0.5510
             0.3550
             0.4943
             0.0365
             0.2260
             0.8159
             0.2284
             0.6553
             0.0621
             0.7075
             0.2408
             0.6907
             0.1062
             0.2640
             0.7034
             0.4021
             0.6553
             0.9700
             0.0380
             0.0988
             0.2560
             0.5598
             0.9166
             0.1402
             0.7054
             0.0178
             0.2611
             0.1358
             0.0503
             0.5782
             0.2432
             0.9448
             0.5876
             0.7256
             0.2849
             0.6767
             0.8642
             0.1943
*
*    64-POINT FFT CORRESPONDING TO VECTOR X
*
        Y =

            30.3774
             1.7780 - 2.5584i
            -1.0376 - 2.3999i
            -1.0123 + 2.4889i
             0.6594 + 2.3639i
            -1.5228 - 0.7527i
            -3.8171 - 0.2050i
            -2.7096 + 1.2841i
             2.1622 - 1.6863i
             0.2879 + 1.8671i
            -1.5479 + 1.6298i
            -0.6366 - 0.1176i
             2.2902 + 1.5549i
            -2.4837 - 0.5842i
            -1.7338 + 0.0738i
            -0.2180 - 0.4726i
            -0.2104 + 0.4897i
            -1.7473 - 1.0213i
             0.1233 - 2.3915i
            -0.6415 - 1.1144i
            -2.7719 - 0.4802i
            -0.0063 - 0.3885i
            -0.7163 + 1.5682i
             0.3218 - 1.3316i
            -0.7823 + 1.0607i
            -0.2553 + 2.8270i
            -1.0813 - 2.7861i
             3.4869 + 1.9485i
             3.0352 + 1.3855i
             3.2099 + 2.3564i
            -1.9511 - 0.7714i
             1.8755 + 0.2867i
```

-1.5474
1.8755 - 0.2867i
-1.9511 + 0.7714i
3.2099 - 2.3564i
3.0352 - 1.3855i
3.4869 - 1.9485i
-1.0813 + 2.7861i
-0.2553 - 2.8270i
-0.7823 - 1.0607i
0.3218 + 1.3316i
-0.7163 - 1.5682i
-0.0063 + 0.3885i
-2.7719 + 0.4802i
-0.6415 + 1.1144i
0.1233 + 2.3915i
-1.7473 + 1.0213i
-0.2104 - 0.4897i
-0.2180 + 0.4726i
-1.7338 - 0.0738i
-2.4837 + 0.5842i
2.2902 - 1.5549i
-0.6366 + 0.1176i
-1.5479 - 1.6298i
0.2879 - 1.8671i
2.1622 + 1.6863i
-2.7096 - 1.2841i
-3.8171 + 0.2050i
-1.5228 + 0.7527i
0.6594 - 2.3639i
-1.0123 - 2.4889i
-1.0376 + 2.3999i
1.7780 + 2.5584i

Appendix F2. File to Be Linked with the Source Code for a 64-Point, Radix-4 FFT.

```
*
*       APPENDIX F2
*
*       FILE TO BE LINKED WITH THE SOURCE CODE FOR A 64-POINT, RADIX-4 FFT.
*
        .globl    SINE
        .globl    N
        .globl    M
*
N       .set      64
M       .set      6
*
        .data
*
SINE
        .float    0.000000
        .float    0.098017
        .float    0.195090
        .float    0.290285
        .float    0.382683
        .float    0.471397
        .float    0.555570
        .float    0.634393
        .float    0.707107
        .float    0.773010
        .float    0.831470
        .float    0.881921
        .float    0.923880
        .float    0.956940
        .float    0.980785
        .float    0.995185
COSINE
        .float    1.000000
        .float    0.995185
        .float    0.980785
        .float    0.956940
        .float    0.923880
        .float    0.881921
        .float    0.831470
        .float    0.773010
        .float    0.707107
        .float    0.634393
        .float    0.555570
        .float    0.471397
        .float    0.382683
        .float    0.290285
        .float    0.195090
        .float    0.098017
        .float    0.000000
        .float    -0.098017
        .float    -0.195090
        .float    -0.290285
        .float    -0.382683
        .float    -0.471397
        .float    -0.555570
        .float    -0.634393
        .float    -0.707107
        .float    -0.773010
        .float    -0.831470
        .float    -0.881921
        .float    -0.923880
        .float    -0.956940
        .float    -0.980785
        .float    -0.995185
        .float    -1.000000
        .float    -0.995185
        .float    -0.980785
        .float    -0.956940
        .float    -0.923880
        .float    -0.881921
        .float    -0.831470
        .float    -0.773010
        .float    -0.707107
        .float    -0.634393
        .float    -0.555570
        .float    -0.471397
        .float    -0.382683
        .float    -0.290285
        .float    -0.195090
        .float    -0.098017
        .float    0.000000
        .float    0.098017
        .float    0.195090
        .float    0.290285
        .float    0.382683
        .float    0.471397
        .float    0.555570
        .float    0.634393
        .float    0.707107
        .float    0.773010
        .float    0.831470
        .float    0.881921
        .float    0.923880
        .float    0.956940
        .float    0.980785
        .float    0.995185
```

Appendix F3. Link Command File

```
*
*    APPENDIX F3
*
*
*    LINK COMMAND FILE
*
*    DO NOT TYPE IN THESE FIRST SEVEN LINES
-o 12opt64.out
12fopt.obj
sin64.obj

SECTIONS
{
    .text : {}
    .data : {}
    IN 809800h : { 12fopt.obj(IN) }
    .bss 809C00h: {}
}
```