Part Number 800-1191-02
Revision: 1 of 20 October 1986

Engineering Manual

for the

Sun Graphics Processor

Sun Microsystems, Inc.,
2550 Garcia Avenue,
Mountain View,
California 94043
(415) 960-1300

Credits and Trademarks

Multibus is a trademark of Intel Corporation.

Sun Microsystems and **Sun Workstation** are registered trademarks of Sun Microsystems, Incorporated. **Sun-2**, **Sun2**, **Sun-2/xxx**, **Deskside**, **SunStation**, **SunCore**, **SunWindows**, and **DVMA** are trademarks of Sun Microsystems, Incorporated.

UNIX is a trademark of AT&T Bell Laboratories.

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

Contents

Tables

Figures

SUN PROPRIETARY

Preface

Welcome to the Sun Graphics Processor. This manual presents an engineering and hardware description of the Graphics Processor and Graphics Buffer boards.

Summary of Contents

This manual has fifteen chapters and an appendix:

Chapter 1

Introduction — contains a basic overview of the the Graphics Processor, its position in the Sun architecture, and associated Graphics Buffer board.

Chapter 2

Overview — presents a simplified block diagram of the Graphics Processor and Graphics Buffer boards.

Chapters 3-14

Graphics Processor Board Circuitry — gives a detailed description of the circuitry on the Graphics Processor (GP) board.

Chapter 15

Graphics Buffer Board — gives a detailed description of the circuitry on the Graphics Buffer (GB) board.

Appendix A

State Diagrams, PAL listings, and Schematics — contains the state diagrams referred to in this manual, along with the PAL listings and schematics.

Finally, to help us maintain the currency and accuracy of this material we have supplied a reader comment sheet at the end of this guide. Please use the comment sheet to list errors and omissions. Your responses will help a great deal in our efforts to keep our documentation up to date.

Glossary

A few terms are used throughout this document which, without explanation, may seem confusing.

□ Positive Logic — positive logic means that the asserted level (see below) of a signal is a logic 1 (see below also), 2.8 to 4.5 volts for a TTL gate.

□ Asserted — when we say that a signal is ''asserted,'' we mean that it is in its active, or true, state. In positive logic this means that a signal like READ, when asserted, is equal to its most positive state. When a signal like WRITE*, WRITE-, or WRITE\ (the three are synonymous) is asserted it is equal to its most negative state.

- Activated — means the same as "asserted."
- Logic 1 — in positive logic, a logic 1 stands for the more positive of the two voltage levels. A logic 1 in negative logic stands for the more negative of the two voltage levels.
- Logic 0 — in positive logic, a logic 0 stands for the more negative of the two voltage levels. A logic 0 in negative logic stands for the more positive of the two voltage levels.
- Set — means the same as logical 1.
- Clear — means the same as a logical 0.
- ON — when it refers to a switch (or switch section) setting, is synonymous with CLOSED. This means that the signal at the input of the switch (or switch section) is shorted to its output.
- OFF — when it refers to a switch (or switch section) setting, is synonymous with OPEN. This means that the signal at the input of the switch (switch section) is NOT SHORTED (signal is not passed) to its output.
- CLOSED — when it refers to a switch (or switch section) setting, is synonymous with ON. This means that the signal at the input of the switch (switch section) is shorted to its output.
- OPEN — when it refers to a switch (or switch section) setting, is synonymous with OFF. This means that the signal at the input of the switch (switch section) is NOT SHORTED (signal is not passed) to its output.
- DIP — stands for Dual In-line Package, and refers to the physical geometry of the chip (rectangular, with pins on the two longer sides).
- DIP Switch — a multi-sectioned switch which has DIP geometry.
- Switch — a device for making or breaking an electrical circuit. A **switch** may have one or more **sections**, each of which may control a circuit.
- 0x — hexadecimal prefix; the number following this prefix is in hexadecimal.
- PCB — printed circuit board
- TTL — transistor-to-transistor logic

Schematic Conventions

The following conventions are used in this document and on the schematics.

On the schematics, numbers in parenthesis near signal names refer to the schematic page where a source signal originates or a destination of the signal exists. The source and destination of a bus is the point the designers believed to be the "major" source; for example, the AM29116 on the VPBUS and PPBUS.

The components on the schematic are identified by a component name (for example, F74), component designation (for example, UA13) and values if necessary (for example, 1K).

The component designations provide a unique reference name for each device. This unique designator consists of a letter followed by two to four other characters. The first letter of the designation indicates the type of component and is one of:

Table 1 *Component Designations*

Letter	Component Type
C	**Capacitor**
D	**Diode or LED**
J	**Jumper**
P	**Connector**
R	**Resistor**
U	**Integrated Circuit**

The other characters of the component designation tell the grid location of the component on the PCB. For example, UR12 is the IC located at the PCB grid location R-12; RC5 is the resistor located on PCB grid location C-5. Note however that capacitors, jumpers, and connectors do not have a grid location but instead are given sequential numbers without regard to their location on the PCB (e.g. J1, J2, J3, etc.). In very few instances, the PCB grid was not fine enough to uniquely identify components and in those cases a letter A or B was appended to the end of the designations (for example, RM22A and RM22B define the two resistors at grid location M-22). This occurs a total of 5 times on both boards.

An asterisk after a signal name is used to indicate the signal is active low or to distinguish a signal from its complement. Signals of the form Pn.NAME indicate that the signal is part of the Pn connector. For example, P1.AS* is an active low signal attached to the P1 connector.

Pull-up signals are designated by PUn and pull-down signals are designated by PDn. For automatic testing, different pull-up signals are used on the unused asynchronous set and reset inputs of a flip-flop. Also, inputs are not tied directly to ground so that these signals can be toggled and individual components checked even when installed in the circuit.

In this document, a grid location (from the schematic) is used to reference a logic element or signal on a schematic. For example, on schematic sheet 4, the oscillator is located at schematic grid location B8. In this text, these grid locations are in parentheses after the logic element or signal name.

All input signals to PALs are shown on the left side of the device and all the outputs from the PALs are shown on the right side of the device.

Understanding The State Diagrams

The state machines used on the GP and the Graphics Buffer boards are all synchronous sequential state machines implemented in PALs. Each state machine has an associated state diagram. There is a sample figure in the appendix (labelled "Appendix A, Figure 1") which shows an example of a state machine and will be used to explain the nomenclature and standards used in the state diagrams.

The sample state machine has four states, as shown by the large circles. The states are labelled (for reference only) A, B, C, and D. In addition, a state assignment chart is shown which assigns a unique encoding of the two state flip/flops to each of the four states.

The state machine shown has four input signals, three output signals, and two state signals (state flip/flops). The input signals are named RESET, I1, I2, and I3. The output signals are named OUT1, OUT2, and OUT3. OUT1 and OUT2 are registered outputs from the PAL and OUT3 is a combinatorial output from the PAL. The state signals are named FF1 and FF0 and are registered outputs from the PAL. Additionally, there is another signal to the PAL, a clock. The clock signal is used to clock all the flip/flops (registered outputs). The state machine makes a transition on every clock period (although the transition may be to the same state machine state, state A to state A, for example).

The state transitions are shown by the arrows. The input signals which caused the transitions, along with the outputs which will result from the transition, are shown next to the arrows within the brackets. The horizontal line separates the inputs and the outputs; the inputs are shown on the top and the outputs are shown on the bottom.

A brief description of the example diagram follows.

The arrow in state A with the signal RESET above it means that when the signal RESET is active—regardless of any other previous conditions—the state machine will go to state A with none of the outputs activated. This is what will occur after a power on reset or a manual reset.

Once in state A, the state machine has two paths which it can follow. If I1 is active, the state machine will go to state B and activate the output OUT1. Notice that the output OUT1 will become active on the rising edge of the clock which puts the state flip/flops to state B (OUT1 will be active during the clock period where the state machine is in state B). The outputs next to the arrows become valid after the state transition is true for all registered outputs (after going from the state A to state B, the registered output OUT1 will become active while in state B). If I1 is not active (shown by /I1), the state machine will stay in state A and none of the outputs will be activated, shown by 0------0. The notation 0------0 means that none of the outputs will be activated for this state transition.

When in state B the state machine has two possible state transitions. If I1 is active AND I2 is not active, shown by (I1)(/I2), the state machine will go to state C and activate the outputs OUT1 and OUT2, shown by OUT1,OUT2. If I1 AND I2 AND I3 are all active, shown by (I1)(I2)(I3) the state machine will stay in state B and activate OUT2. If I1 AND I2 are active AND I3 is not active, shown by (I1)(I2)(/I3), the state machine will stay in state B but this time will activate the outputs OUT1 and OUT2.

From state C the state machine either stays in state C if I2 is not active (/I2) or it goes to state D if I2 is active (I2). For either case none of the outputs are activated as indicated by 0------0.

From state D the state machine goes to state A right away regardless of any inputs. This condition is shown by the input symbol X------X which means that this transition will occur because all the inputs to the state machine are don't cares. When going from state D to A, the output OUT3 will be activated. Since OUT3 is a combinatorial output from the PAL, it will be active when in state D (in contrast to the registered outputs which, if activated here, would have been active when in state A).

There will be two outputs from the PAL named FF1 and FF0 which correspond to the state assignment table shown. When FF1 is at a logic 1 and FF0 is at a logic 0, it indicates that the state machine is in state A (first line of the state assignment table). If FF1 and FF0 are at logic levels 1 and 1 the state machine is in state B, if they are 0 and 1 the state machine is in state C, and when they are 0 and 0 the state machine is in state D.

Applicable Documents

We emphasize that this manual outlines rather than exhausts many of the topics contained within. References to applicable documents supplied with your system are given throughout; however, and we urge you to read these documents should you need further information.

Table 2 *Sun Documentation*

Sun Part Number	*Description*
800-1190	Graphics Processor Hardware Reference Manual

Table 3 *Vendor Documentation*

Description
AMD Bipolar Microprocessor Logic and Interface Data Book
AMD Bipolar/MOS Memory Data Book
Fairchild Advanced Schottky TTL (FAST) Data Book
Texas Instruments ALS/AS Logic Circuits Data Book
Programmable Array Logic (PAL) Data Book
Weitek 1032/1033 floating-point processor data sheet
4501 FIFO data sheet
VMEbus Specification

1

Introduction

1

Introduction

This document details the design of the Graphics Processor (GP), which includes both the GP board and the optional Graphics Buffer board. The methodology is to explain each schematic page in order. A "higher-level" picture is given in the Graphics Processor Hardware Reference Manual, Sun part number 800-1190. It is assumed that the reader is familiar with the reference manual before using this engineering manual.

The purpose of this document is to provide a low-level record of the hardware design—something analogous to comments in software. It documents what was done and why. It will assist engineers and technicians attempting to learn and understand the hardware. In addition it is a source for generating installation and maintenance manuals and other support documentation.

2

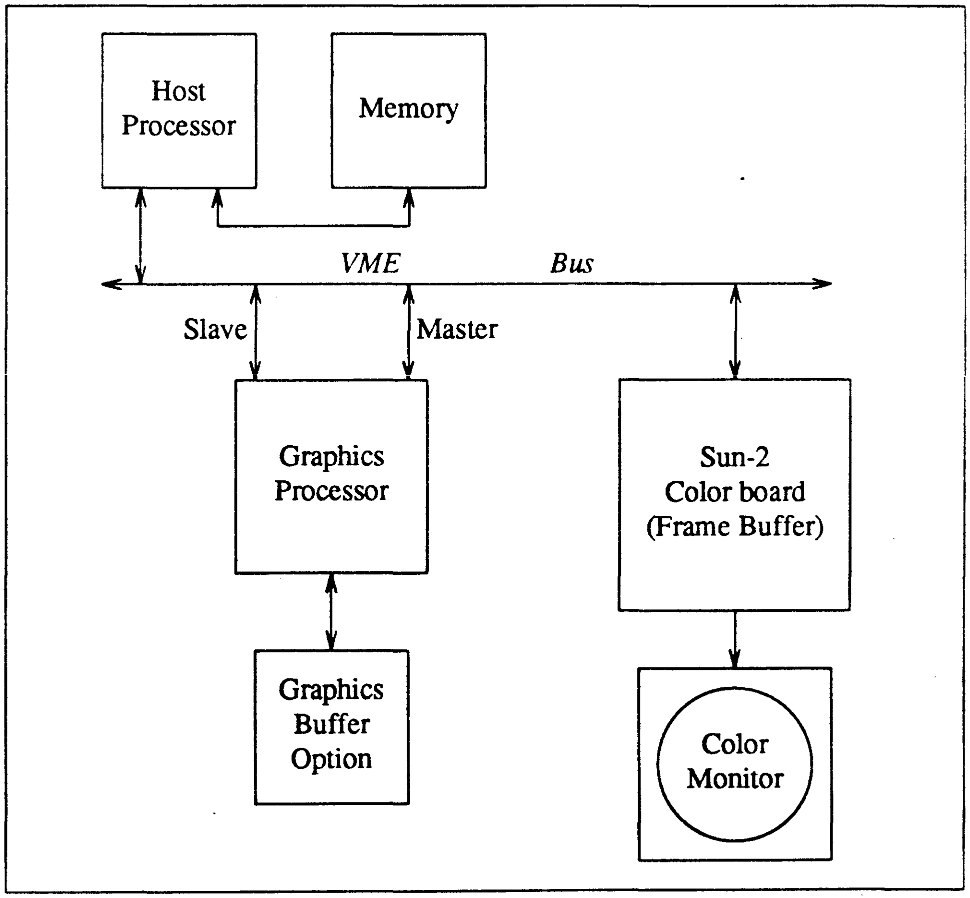
Overview

2

Overview

The Graphics Processor (GP) is a two board set (including the Graphics buffer) that presently installs in a Sun2/160. The GP resides logically between the host processor and the frame buffer, receiving commands and generating pixels. The figure below illustrates a GP installation in the Sun2/160.

Figure 2-1 *Sun-2/160 Color Workstation: System-Level View*



The GP is functionally divided into two sections, the

- Viewing Processor (VP), and the
- Painting Processor (PP).

Much of the implementation of these two sections is similar, including the AM29116 microprocessor, the AM2910A sequencer, the branch logic, and the microcode memory interface. Differences are the subsections that attach to each section's bus. The VP contains the

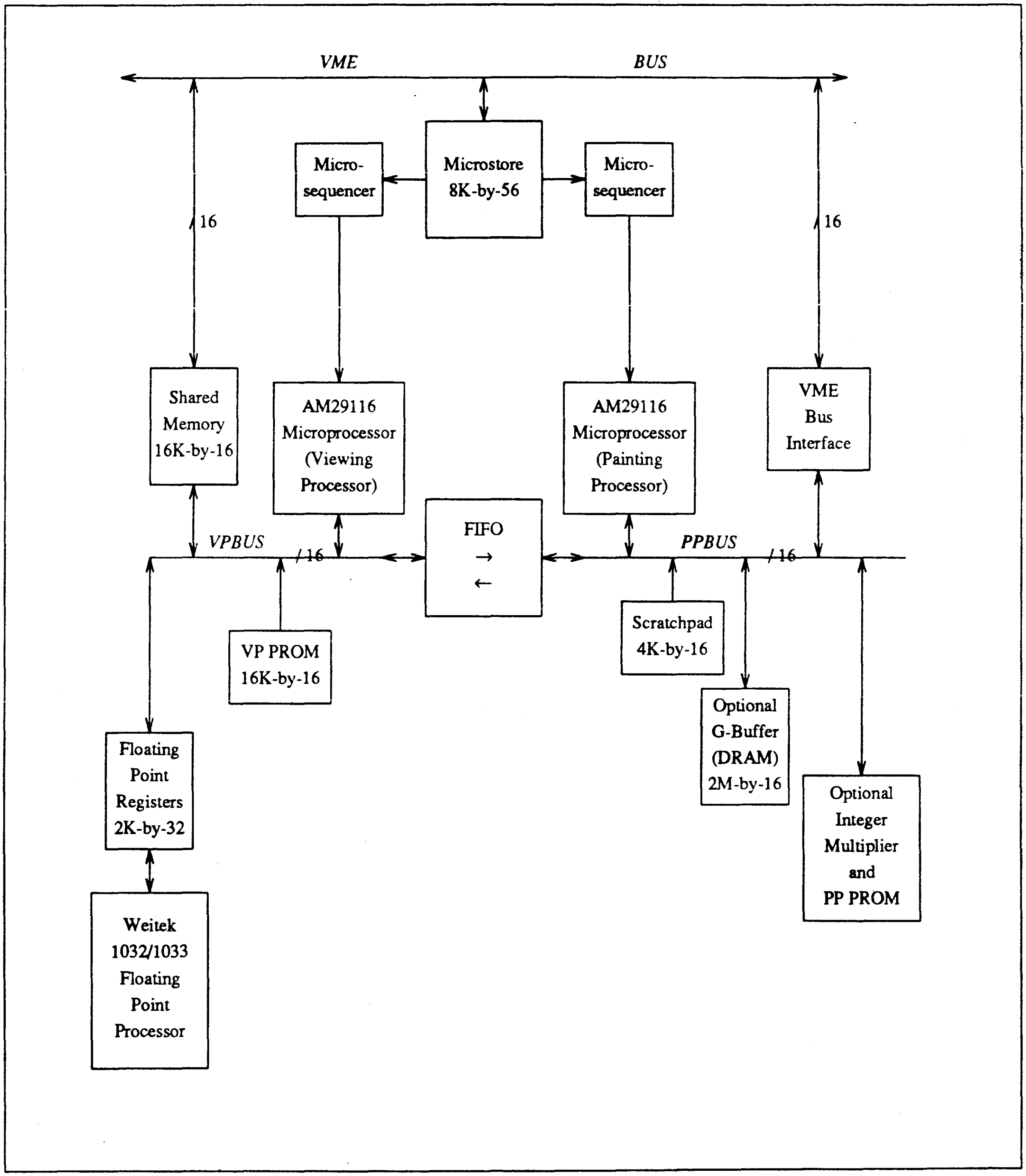
- floating point, and
- shared-memory subsections.

The PP contains the

- VMEbus master interface,
- scratchpad memory,
- Graphics buffer, and
- the integer multiplier.

NOTE *Following is a brief description of each GP subsection and a reference* (in parenthesis) *to the schematic pages on which the logic appears. Refer to the next figure (Graphics Processor block diagram) to see how these subsections interconnect.*

Figure 2-2 *Block Diagram of the Graphics Processor*



2.1. Clock and Reset Circuits *(pages 4-5)*

The GP uses an on-board, free-running oscillator for its basic timing source. Under host program control, the GP clock can be halted (primarily for diagnostics) or the GP can be reset.

2.2. VME Interface *(pages 6-7, 31)*

The VME interface consists of:

- registers accessible by the host processor (board id, control, status, microstore address, and microstore data),
- the logic to interface to the shared-memory,
- the logic to implement the VME protocol,
- the address/data transceivers, and
- the PP VME interface.

The GP is both a VME slave (VME registers and shared-memory) and a VME master. The GP is also a VMEbus interrupter. The slave control logic is primarily on page 7 of the schematics and the master control logic is primarily on page 31.

2.3. Microstore *(pages 8-12)*

The microstore is the program memory for both the VP and the PP. By sharing the available memory bandwidth, both processors access the same memory. The memory is also accessible from the VMEbus (read and write) but only when the two processors are halted. The size of the microstore is 8K-by-56-bits using 28 4K-by-4 chips.

Microstore timing is such that the Viewing Processor and the Painting Processor can share the memory. For each processor, the first half of a cycle is used to determine the address of the next microinstruction. The second half is used to route this address to the microstore, access the memory, and load the fetched instruction into the instruction register. The two processors run 180 degrees out of phase so that the halves of the processor cycles mesh correctly.

2.4. Shared-Memory *(pages 13-14)*

The shared-memory is the vehicle for passing commands, parameters, and any other information between the host processor and the GP. Its bandwidth is shared by VME accesses and by the Viewing Processor; both are allocated independent access (read or write) each VP cycle. The size of the shared memory is 16K 16-bit words.

2.5. Viewing Processor *(pages 15-17)*

The Viewing Processor is implemented with the following:

- 56-bit instruction register (also called pipeline register),
- PALS for VPBUS source/destination encoding and miscellaneous controls,
- AM29116—16-bit bipolar microprocessor,
- AM2910A—microcode sequencer with Bank Select PAL and condition code status register,
- 15-bit branch register, and

- 16-bit general field buffer.

2.6. Floating Point *(pages 18-22)*

The floating point section is built around the Weitek 1032/1033 chip set. It consists of the:

- Weitek chips,
- 2K 32-bit words of floating-point registers,
- pointers into these registers,
- buffers to match timing constraints, and
- the necessary control logic.

Also part of the floating point subsection is the VP PROM (page 23) which is used to store constants necessary for arithmetic operations (such as ''divide'').

The floating point section can be viewed as an attached array processor. It does 32-bit floating-point operations at rates up to 4.16 Mflops. Graphical data (especially 3D) are often defined in floating-point coordinates, manipulated with floating-point operations, and finally converted to integers for display on the screen.

2.7. VP Miscellaneous Logic *(page 23)*

Miscellaneous components of the Viewing Processor are:

- VP PROM—described above in floating point section;
- VP N Register—allows runtime modification of those AM29116 instructions which contain an ''n'' field;
- Interprocessor flag #1 Register—8 bits which can be written by the VP and read by the PP along with three other flags;
- VP Status Register—4 bits which can be written by the VP and read via the VMEbus. This register also controls four LEDs.

2.8. First In First Out (FIFO) Buffer *(page 24)*

The FIFO provides an asynchronous interconnection between the VP and the PP (although it is a synchronous implementation of the FIFO). 16-bit words can be passed from the VP to the PP; or the FIFO can be reversed (under control of the VP) and words passed from the PP to the VP.

2.9. Painting Processor *(pages 25-27)*

The Painting Processor is implemented with the following (note its similarity with the Viewing Processor):

- 56-bit instruction register (also called pipeline register),
- PALS for PPBUS source/destination encoding and miscellaneous controls,
- AM29116—16-bit bipolar microprocessor,
- AM2910A—microcode sequencer with Bank Select PAL and condition code status register,
- 15-bit branch register, and

- 16-bit general field buffer.

2.10. Scratchpad Memory *(page 28)*

The scratchpad memory is general-purpose storage for the PP. The size of the memory is 4K 16-bit words using 4K-by-4 chips.

2.11. PP Miscellaneous Logic *(page 30)*

The miscellaneous logic includes:

- PP N Register—allows runtime modification of AM29116 instructions which contain an ''n'' field;
- Interprocessor Flag #2 Register—8 bits which can be written by the PP and read by the VP along with three other flags;
- PP Status Register—4 bits which can be written by the PP and read via the VME bus. This register also controls four LEDs.
- PP Bus extension—the logic necessary to extend the PP Bus to the Graphics Buffer board containing the Graphics buffer memory, integer multiplier, and PP PROM.

2.12. Graphics Buffer Board

Destination Decode *(page 4)*

On the GP board, a PAL encodes the signals that select the possible sources and destinations on the Graphics Buffer board. A PAL on the Graphics Buffer board decodes the destination encodings into individual signals. The encoded destinations are:

- no destination,
- Graphics buffer high address pointer,
- Graphics buffer low address pointer,
- Graphics buffer write data register,
- multiplier mode register,
- PP PROM pointer,
- multiplier X operand, and
- multiplier Y operand.

Graphics Buffer *(pages 5-11)*

The Graphics buffer has two modes of operation:

- **Normal mode**, which allows for sequential reads or sequential writes. The address pointer is automatically incremented after each access.

NOTE *Normal mode has a sub-mode called* **fill mode**. *In fill mode, each write fills four consecutive memory locations with specified data. Thus the entire memory could be loaded (for instance, a clear memory operation) with the same value in one-fourth the time. When in fill mode, the address pointer is automatically incremented by four and reads are illegal.*

- **Read/modify/write mode**, which allows for a read and then a possible write to the same location before the address pointer is automatically incremented;

this mode is especially useful for the hidden surface removal algorithm.

In both modes (normal and RMW) the actual memory access can be overlapped with other PP activities if properly microcoded.

Integer Multiplier *(pages 12-13)*

The integer multiplier accepts two 16-bit operands and produces a 32-bit result. Rendering algorithms dealing in screen coordinates often require multiplications; hence the inclusion of the integer multiplier. Also included is a PP PROM (similar to the VP PROM) which is used to store constants used in various algorithms.

Notice that unlike the floating point subsection, the integer multiplier does not have associated storage. Whereas the floating point subsection is like an attached array processor, the integer multiplier is more like an integrated co-processor.

3

Connectors, Connection Restrictions *(schematics pages 1, 2, and 3)*

3

Connectors, Connection Restrictions *(schematics pages 1, 2, and 3)*

3.1. Title Page *(page 1)*

Page 1 of the schematics is the title page listing revisions, spares, PAL and PROM part numbers cross-referenced to the IC designations on the schematic and other notes.

3.2. Board/Backplane P-Connectors *(page 2)*

Page 2 shows the three 96-pin connectors that interface the GP board into the backplane. All three rows of P1 and row B of P2 make up the VME bus signals. Row A of P2 is the private bus between the GP and the optional Graphics Buffer board. Rows A and C of P3 are power inputs. Row C of P2 and row B of P3 are unused.

The VME bus defines four bus grant signals. The GP only uses BG3. Therefore, BG0, BG1, and BG2 are connected on the GP board, IN to OUT—that is, they are shorted across the board.

Connection Restrictions

There are restrictions which must be be understood before plugging the GP board-set into the system backplane.

Signals on the private bus between the two boards reside on row A of the P2. If the Graphics Buffer board is installed, the two-board combination must be in slots which provide a dedicated P2-row A bus (since the GP drives some of the signals on P2-row A). On Sun-2/160, this is to be slots 3 and 4. If the Graphics Buffer board is *not* installed, the GP can be plugged into any slot where P2-row A is not being used by some other board. In the Sun Model 160, this can be any one of slots 5 through 12, or slots 3 or 4 *if P2-row A of slots 3 and 4 are not being used by some other board.*

Graphics Buffer Ground

The Graphics Buffer grounds shown on page 2 are used to minimize the noise that may occur on the P2 bus signals between the GP and the Graphics Buffer.

3.3. Capacitors *(page 3)*

The bulk and decoupling capacitors are shown on page 3. One decoupling cap is used for (approximately) each pair of TTL chips (14-, 16-, or 20-pin); the actual number is based on the printed circuit board layout. One cap per memory chip is allocated, and one cap per >20-pin chip is provided.

The ground test points shown on page 3 are merely wire loops consisting of jumpers spread out on the PC board to provide convenient ground connections for a scope probe, etc.

4

Clock and Reset Circuits *(pages 4 and 5)*

4

Clock and Reset Circuits *(pages 4 and 5)*

4.1. Clock Circuit

The Graphics Processor is a synchronous design and the master clock source is the oscillator on page 4. All sections of the design, including the VME bus interface and the graphics buffer memory refresh timing, are dependent on this frequency.

To provide a 120 nanosecond cycle, a 33.333 MHz oscillator is used. The output is routed through jumper J14 (for test engineering—it provides a simple connection for a faster or slower clock) and to two JK flip-flops to divide the frequency by two and four. The basic clocks generated are:

- □ 2XCLK — 60 nsec period, free-running;
- □ VPCLK — 120 nsec period, used to drive the Viewing Processor;
- □ PPCLK — 120 nsec period, used to drive the Painting Processor.

The phase relationships of these signals are shown in the following figure. Note the following:

1. A rising edge of 2XCLK triggers a change in state of both VPCLK and PPCLK.
2. VPCLK and PPCLK are 180 degrees out of phase; thus the first half of a VP cycle is the second half of a PP cycle and vice versa.

Figure 4-1 *Clocks Timing Diagram*

```
Phase relationship of the GP clocks:

          ____      ____      ____      ____
2XCLK  __|    |____|    |____|    |____|    |__

          _________           _________
VPCLK  __|         |_________|         |_________

       __           _________           _________
PPCLK    |_________|         |_________|
```

SUN PROPRIETARY

The clock signals and their usage is as follows:

- 2XCLKZB — 2XCLK for use on the Graphics Buffer board;
- 2XCLK1, 2XCLK2, 2XCLK1* — used for state machine timing and to generate 30 nsec load pulses;
- PPCLK — a free running, ''early'' PPCLK, see page 25;
- VPCLK — a free running, ''early'' VPCLK, see page 15;
- VPCLK* — free running VPCLK complement, needed for timing on page 22;
- VPCLKFR — free running VPCLK;
- PPCLKFR — free running PPCLK;
- VPCLK0, VPCLK1, VPCLK2, VPCLK3, VPCLK4, VPCLK1*, VPCLK3*, VPCLK4* — VPCLKs which can be halted;
- PPCLK0, PPCLK1, PPCLK2, PPCLKZB, PPCLK1* — PPCLKs which can be halted.

The basis of synchronous design is having all components use a common clock. Loading constraints prohibited the use of a single clock on the GP, which is why there are the number of clock signals above. Ideally, all clocks would be identical but practically this is impossible since different paths have different delays. The task then is to minimize the skew between clock edges. The following actions were taken to minimize this skew:

- clocks which are used in the same subsection are routed through the same F240;
- an equal number of gate delays are used wherever possible;
- loads are balanced on the F240's to prevent all inputs switching to the same logic state at the same time (thus preventing ground shifts);
- careful timing analysis using min/max skews was done.

The F240 family of chips are used as the clock drivers because of their high drive current.

Also on page 4 are control circuits which halt the clocks. This is a diagnostic function allowing observation of the GP in a steady state. The clocks which can be halted and their halted states are:

- VPCLK0, VPCLK1, VPCLK2, VPCLK3, VPCLK4 — halted in the logic 1 (high) state;
- VPCLK1*, VPCLK3*, VPCLK4* — halted in the logic 0 (low) state;
- PPCLK0, PPCLK1, PPCLK2, PPCLKZB — halted in the logic 1 (high) state;
- PPCLK1* — halted in the logic 0 (low) state.

The 3-input AND and NAND gates perform the halt function. The halt implementation for VPCLK4 is described below, all others being analogous.

VPCLK4 can be halted either via the VME bus (see page 5) or via a connection to the Hilevel PROM Emulator. Inputs to the 3-input AND gate (D7) on page 4 are the free-running clock PPCLK (the complement of free-running VPCLK), the control line VPHALT* from page 5, and the halt control from the Hilevel test gear. Both of the halt controls can become asserted only when PPCLK is low; therefore the AND gate output will not glitch when a halt control is asserted. If neither halt control is asserted, the output of the AND gate is PPCLK which is inverted through the F240 to become VPCLK4. If either of the halt controls is asserted, the AND output is low and VPCLK4 is halted in the high state.

The Hilevel PROM Emulator plugs into the microstore sockets on the GP board and provides an alternate, known-working method for loading and executing microcode. It also provides trace, breakpoint, and single-step capabilities to assist the debug of hardware and microcode. To enable these capabilities, the GP clock must be routed to the Hilevel which returns a halt control during the proper phase of the clock. These connections are shown on page 4 at grid locations D2, A4, and A3. Wire loop connections consisting of jumpers are provided to attach the Hilevel probes (note that a trace has been placed through these jumpers so that headers and jumpers are not necessary if the Hilevel interface is not going to be used). The S04 inverters provide isolation between the GP and the test gear. The pull-up and pull-down resistors condition the input for use on the GP. Two sets of connections to the Hilevel are provided on the GP board—one for the Viewing Processor and one for the Painting Processor. This means that only one processor can have trace, breakpoint, and single-step capabilities at a time (unless two Hilevel emulators are used). Both processors can execute out of common Hilevel memory, but only one will have these other capabilities.

4.2. Resets

Page 5 contains the reset circuitry, the pull-ups and pull-downs, the halt control accessed via the VME bus, and a flip-flop to prevent the GP from starting an erroneous VME operation after a reset.

Three reset methods are provided.

1. A manual reset (primarily for lab debug) is activated by grounding the input to the ALS00 at D7. A wire loop consisting of a jumper (J13) is provided so that the Hilevel reset line or a switch can be easily attached.

NOTE *A trace has been placed in this jumper so that a header and jumper will not be necessary if Hilevel interface is not going to be used.*

2. Also input to this ALS00 is the VME reset signal SYSRESET* directly from the P1 connector. As specified in the VME specification, this signal is activated on power-up and can be asserted under software control (resetting all VME devices). These two reset sources are ORed together to generate the power-on reset signal: POR*.

3. The third reset method is a software reset for the GP board only. Via the VME bus, the host processor software creates a positive-going pulse on VMERST (C8). This signal is ORed with the power-on reset to generate the

general reset signals RST0*, RST1*, and RST.

The destinations of the reset signals are listed below.

POR*

- VME bus slave control state machine
- REQUEST flip-flop
- WEGATE flip-flop
- WORDXFER flip-flop
- IFLAG flip-flop
- VME bus interrupter state machine
- VME bus master data transfer state machine
- VMERDY flip-flop
- VME bus requester state machine

RST0*

- VPJZ flip-flop
- VPHLT flip-flop
- PPJZ flip-flop
- PPHLT flip-flop
- Interrupt Enable flip-flop
- VP destination (PAL), inhibits loads
- FPFLAG flip-flop
- PPTOVP flip-flop
- PP status register

RST1*

- VP status register
- FIFOs (2)
- FIFO write control state machine
- FIFO read control state machine
- SFIFOMT flip-flop
- SFIFOFULL flip-flop
- PP destination (2 PALs), inhibits loads
- ENSTRTVME flip-flop

RST

- ZBUFRDY* flip-flop

The pull-ups and pull-downs are shown in the lower left corner of page 5. The pull-downs are implemented with an ALS240 for drive capability and because its outputs can be tri-stated allowing test engineering equipment to have control over the pull-downs. If a chip has an input tied directly to ground, this input can not be tested; using the ALS240 allows these inputs to be tested.

The ENSTRTVME flip-flop is used to disable any VME transactions from being started by the GP on the first PP processor clock after a reset. This is necessary because after a reset the contents of the PP instruction register are undefined and may in fact contain the instruction to start a VME transaction on the first rising edge of PPCLK. To prevent this occurrence, this flip-flop is cleared by RST1* and gates the STRTVME signal (from the PAL on page 27, section D1) so that the VMERDY flip-flop (page 31, section B7) can not be activated on the first PPCLK rising edge. The first rising edge of PPCLK sets the ENSTRTVME signal logic high and also loads valid data into the PP instruction register.

The rest of page 5 shows the halt control circuits for the clock. As can be seen, the VP and the PP have independent halt control circuits. Since the VP and PP run/halt circuits are identical, only operation of the VP run/halt circuit will be described.

The ALS175 (D4 & C4) samples the two control inputs from the GP control register (page 6). To put the VP into run-state, the software sets the VPCONT signal. The ALS175 is configured as a differentiator and will detect the rising edge of the VPCONT and a one-clock-period-wide pulse will be generated by the AND gate at D3. This input is then used to set the VPRUN flip-flop; this JK* flip-flop will remain set until a VPHLT signal is received. If VPSTR0 is set coincident with VPCONT, then the VPJZ flip-flop (D1) is set at the same time as the run flip-flop, causing the Viewing Processor to begin execution at microcode location zero. Since the VPJZ flip-flop is a D-type, the VPJZ signal will be asserted for only one clock.

NOTE *VPRUN is the same as VPHALT*. It has been given two names for clarity of discussion.*

To halt the VP, the software sets the VPHLT signal. Similar to above, a one-clock-period-wide pulse is generated which is routed to the K* input of the JK* flip-flop thus resetting the run/halt state.

A table listing the run/halt operations is below. The figure following shows a simple timing diagram displaying the operation of this circuitry.

Table 4-1 *Run/Halt Operations*

Operation	When Running	When Halted
VPCONT	no effect	starts VP at location where last halted
VPCONTwith VPSTR0	continue running at location 0	starts VP at location 0
VPHLT	halts VP	no effect
VPHLT with VPSTR0	halts VP	no effect
VPSTR0	no effect	no effect
VPCONT and VPHLT	halts VP	starts VP at location where last halted
VPCONT and VPHLT with VPSTR0	halts VP	starts VP at location 0

Figure 4-2 *Clock Run Timing Diagram*

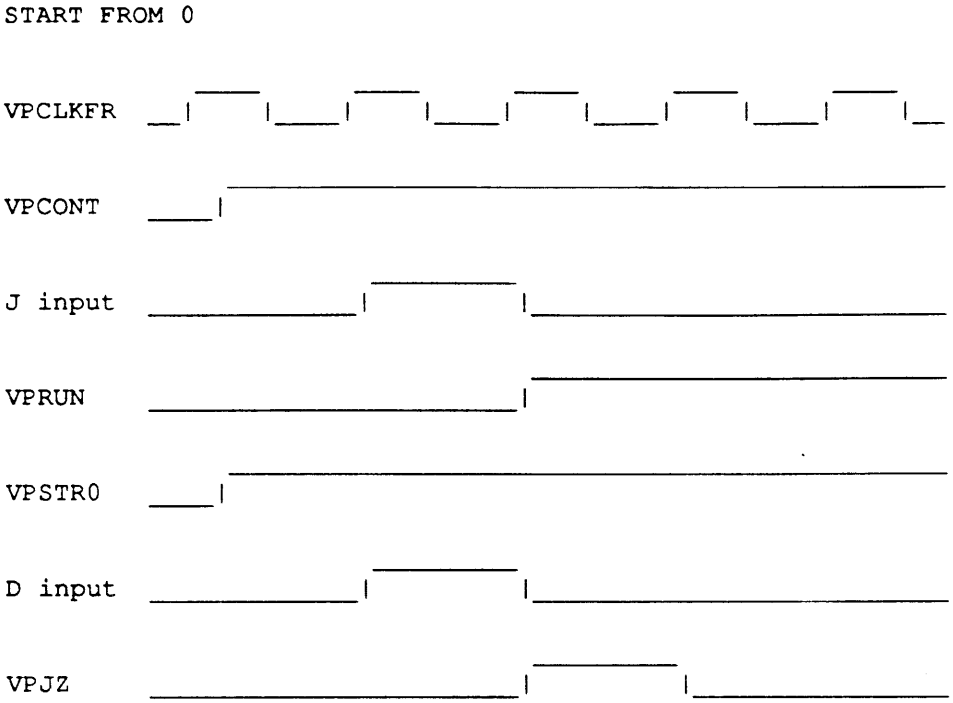
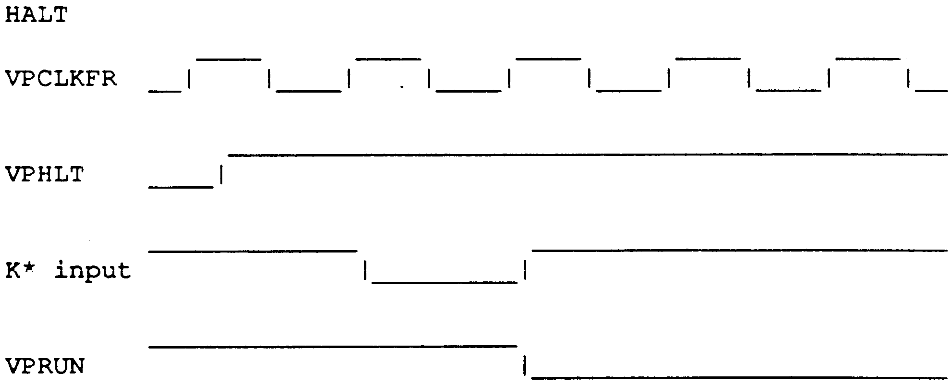


Figure 4-3 *Clock Halt Timing Diagram*



5

VME Interface *(pages 6-8, 13, 29-31)*

5

VME Interface *(pages 6-8, 13, 29-31)*

The VME interface allows the GP to communicate with the other system components in the Sun workstation. The GP contains both a slave and a master VME interface, as defined in the VME specification, along with a VME interrupter.

- The slave interface allows masters on the VME bus to access certain resources of the GP.
- The master interface allows the GP to access resources of slaves on the VME bus.
- The interrupter allows the GP to cause exception processing by the host processor.

The following discussion will detail the implementation of the VME interface of the GP in detail.

5.1. VME Interface: An Overview

Page 6 of the schematics contains the VME address and data and control transceivers, and also some of the GP VME slave registers. The VME slave interface control is on page 7. The VME master interface control and the VME interrupter control is on page 31. Page 8 contains the components necessary to implement the slave interface to the microstore. Page 29 and part of page 30 contains the components necessary to implement the VME master and the VME interrupter. Part of page 13 contains the components necessary to access shared memory from the VME.

5.2. VME Transceivers

The control, address, and data transceivers for the VME bus are shown on page 6 of the schematics. The address, data, and the control modifiers are buffered by F245-type devices. Some of the control lines also on this page are buffered by F244-type devices. These F-type devices were chosen because they satisfy the current-drive requirements of the VME bus.

The VME transceivers are used to provide isolation between the VME bus signals and their corresponding GP internal signals. The GP internal bus corresponding to the VME address bus (P1.A23-P1.A01) is called VMEA<23>-VMEA<01>. The GP internal bus corresponding to the VME data bus (P1.D15-P1.D00) is called VMED<15>-VMED<00>. The GP internal bus corresponding to the VME address modifiers (P1.AM5-P1.AM0) is called AM<5>-AM-<0>. The GP internal signals corresponding to the VME bus control signals P1.WRITE* (read/write indicator), P1.AS* (address strobe), P1.DS1* (data

SUN PROPRIETARY

strobe 1), P1.DS0* (data strobe 0), and P1.LWORD* (long word transfer indicator) are, respectively, VMEWRITE*, VMEAS*, VMEDS1*, VMEDS0*, and VMELWORD*.

P1.A23-P1.A01 *is internally named* VMEA<23>-VMEA<01>

P1.D15-P1.D00 *is internally named* VMED<15>-VMED<00>

P1.AM5-P1.AM0 *is internally named* AM<5>-AM-<0>

P1.WRITE* *is internally named* VMEWRITE*

P1.AS* *is internally named* VMEAS*

P1.DS1* *is internally named* VMEDS1*

P1.DS0* *is internally named* VMEDS0*

P1.LWORD* *is internally named* VMELWORD*

VME Control Transceivers

The control transceivers are implemented with one F245- and one F244-type device. The F245 buffers the VME address modifiers, P1.AM5-P1.AM0, and also the P1.WRITE* and P1.LWORD* signals. The outputs of the F245 are always enabled and only the direction pin is controlled (with signal GPHASVME).

- □ When GPHASVME is active, indicating that the GP is the VME bus master, the GP internal signals drive the VME bus.
- □ When GPHASVME is not active, the VME bus signals drive the internal GP signals.

The input corresponding to P1.LWORD* is just a pull-up resistor. When the GP is the bus master, P1.LWORD* is driven high because the GP does not do long word transfers.

The F244 is used to drive the signals P1.DS1*, P1.DS0*, P1.AS*, P1.LWORD*, and the GP internal signal VMEWRITE*. One section of the F244 always buffers the VME bus signals P1.DS1*, P1.DS0*, P1.AS*, and P1.LWORD* to (respectively) their GP internal versions VMEDS1*, VMEDS0*, VMEAS*, and VMELWORD*. The other section of the F244 enables the GP to drive the VME bus signals P1.AS*, P1.DS1*, and P1.DS0* when it becomes the VME bus master (indicated by the signal GPHASVME* being active). The GP VME master logic (discussed in the section on the *VME Master*) outputs the signals MASTERAS*, MASTERDS1*, MASTERDS0*, and MASTERWRT* which are driven to the VME bus by this F244. The signal VMEWRITE* is driven by the signal MASTERWRT* when the GP is VME bus master and by the VME bus signal P1.WRITE* when the GP is not the VME bus master.

VME Address Transceivers

The address transceivers are implemented with three F245 devices. The B side of the devices are connected directly to the VME P1 bus with the signals named P1.A23-P1.A01. The A side is the GP internal VME address bus and is named VMEA<23>-VMEA<01>. The outputs of the transceivers are always enabled and only the direction pin is controlled. The direction pin is controlled with the signal GPHASVME (GP has control of the VME bus as a master). When GPHASVME is active the GP drives the VME address bus and when GPHASVME is not active the GP is a receiver on the VME address bus.

The address transceivers also drive the signal P1.IACK* (interrupt acknowledge) when the GP is the bus master. The input to this signal is just a pull up resistor because when the GP is the bus master it will not do an interrupt acknowledge and so P1.IACK* is driven high.

VME Data Transceivers

The data transceivers are implemented with two F245 devices. The B side of the devices are connected directly to the VME P1 data bus with the signals named P1.D15-P1.D00. The A side is the GP internal VME data bus and is named VMED<15>-VMED<00>. The outputs of the transceivers are always enabled and only the direction pin is controlled. The direction pin is controlled with the logic implemented using the F10, F08, S04, and the two F00 gates.

The data transceivers are normally, as a default condition, receivers on the VME data bus (the direction pin is a logic low). Under three conditions the transceivers are turned to drive the VME data bus.

1. The first of these conditions is when the GP has become the bus master and is doing a write operation to a slave on the bus. This condition is decoded by the first F00 which ANDs the signals GPHASVME (when active this signal indicates the GP is VME bus master) and VMEREAD* (when not active—logic high—this signal indicates a write operation on the VME bus).
2. The next two conditions in which the data transceivers drive the VME data bus are when either the GP slave is being read by a VME master (indicated by LSLAVEADR* being active) or the GP VME interrupt ID is driven to the VME bus during an interrupt acknowledge cycle (indicated by ENINTID* being active). These conditions are decoded by the F08 gate which ORs the LSLAVEADR* and ENINTID*. The resulting signal is then inverted by the S04 and input to the F10 gate.

The F10 gate does an AND operation on this signal and VMEWRITE* (when not active, indicating a read operation on the VME bus) and DS (when active, indicating that one of the two VME data strobe signals, DS0* and DS1*, is active). The DS signal is used to ensure that the GP does not drive the VME data bus until the data strobe(s) are active and to quit driving the data bus in as little time as possible after the data strobes become inactive. The three conditions decoded are then ORed by the second F00 gate which drives the direction pin of the transceivers.

5.3. VME Slave Control

The slave control logic is on page 7 of the schematics. The slave logic decodes the address from the VME address bus and, if a GP slave port is being addressed, latches that address and activates the signal SREQ which indicates that the VME bus master requests to do a transaction with a GP slave port. The slave control state machine senses this request and issues the appropriate control signals to complete the transaction requested by the VME bus master.

The GP shared memory can accommodate byte or word accesses but the microstore registers can only accommodate word accesses. Longword or 32-bit transfers are not allowed and result in the GP slave control activating P1.BERR* (VME bus error) signal indicating an improper access to the VME bus master. Requests for 8-bit (byte) transfers to the microstore registers are handled as if they were 16-bit transfers. In other words, the GP slave will read and write 16-bit data even though the VME bus master is requesting an 8-bit data transfer. Byte transfers will complete without an error condition being signalled to the VME master.

The GP slave occupies a 64 Kbyte region within the VME standard address space and the starting location of this 64 Kbyte block is selected by setting a hardware DIP switch on the GP. Therefore, the slave can be addressed from the VME bus by having the VME address modifiers equal to 3D hex (0x3D) or 39 hex (0x39) and setting the high eight bits of the VME address equal to the hardware switch on the GP.

The GP slave 64 Kbyte space is divided into two halves; the Microstore registers occupy the first half and shared memory occupies the second half. VME address bit 15, VMEA<15>, selects the Microstore registers when it is a logic low and it selects the shared memory when it is a logic high.

When `VMEA<15> = 1`
    then `Shared Memory` *is selected.*

When `VMEA<15> = 0`
    then `Microstore registers` *are selected.*

When VMEA<15> is a logic low the particular Microstore register is selected by the two least significant bits (LSBs) of the VME address.†

When the first half of the slave space is addressed, the two LSBs of address are used to select one of the four microstore registers. The microstore register addresses are repeated 8K times in the first half of the slave address space because only the two LSBs of address are used. The microstore registers are decoded as follows:

†You may wish to refer to the GP Hardware Reference Manual to get a clearer picture of the organization of the GP slave address space.

Table 5-1 *Decode of the Microstore Registers*

VME address		
VMEA<02>	VMEA<01>	Microstore register addressed
0	0	Board Identification
0	1	GP control/status register
1	0	Microstore address register
1	1	Microstore data register

Slave Address Decode

The slave address decode is implemented with two F521 type 8-bit comparators shown on the lower left of schematic page 7. They compare the value of the 8-bit hardware DIP switch against VME address bits, VMEA<23>-VMEA<16>, and the value of the VME address modifiers, AM5-AM0, against the value of 0x39 or 0x3D, and the value of P1.IACK* (interrupt acknowledge) against a logic high. The output of the comparators is controlled by their enable input. If the enable is not active, then their output can not be activated. The enable input to the comparators is an effective AND function of VMEAS* (address strobe), DS (either data strobe, DS1 or DS0, active), DTACK (data transfer acknowledge), and BERR (bus error). The enable is activated if the VME address strobe, VMEAS*, and one of the data strobes (VMEDS1* or VMEDS0*) is active and neither DTACK nor BERR is active. This AND function is done with the four sections of the F32 gate in the lower left of page 7. The F32 is also used to delay the VMEAS* signal appropriately to ensure that the compare inputs to the F521s are stable before the enable input to them is activated. Because the enable input to the comparators gates the output, these outputs are ensured not to glitch. The outputs of the two comparators are ANDed together by an F02 gate and the resulting signal is called SLAVEADR (slave is being addressed).

If VMEA<15> is low while SLAVEADR is active, then the two LSBs of the VME address, VMEA<02> and VMEA<01>, must be decoded to determine which of the four GP slave microstore registers is being accessed. This is done by the F139 2-to-4 decoder in section D6 of page 7. The F139 is enabled by the F32 gate next to the F139 enable input pin. The output of the F32 is active if the signal SLAVEADR is active and VMEA<15> is a logic low. The outputs of the F139 are the four signals that enable the GP ports to be accessed by the VME master. These signals are:

- □ MSDATA* (microstore data access),
- □ MSADR* (microstore address access),
- □ GPCONREG* (GP control/status register access),
- □ and BRDID* (board identification is being read).

If VMEA<15> is a logic high while SLAVEADR is active, this means that the VME master is accessing shared memory. This condition is decoded by the F00 gate, in section C6 of page 7. The output of this F00 becomes active if the input signal to it (SLAVEADR) is active and the other input to it VMEA<15> is a logic high. The output of the F00 is the signal SHRDMEM* (shared memory is being accessed).

Slave Synchronizer

When the signal SLAVEADR becomes active, indicating that the GP slave is being addressed, it must be synchronized to the clock used by the GP control logic—2XCLK0.

This synchronization is necessary because VME bus accesses are asynchronous to the GP slave control logic. The slave synchronizer, in the lower right of page 7, synchronizes the signal SLAVEADR to the clock 2XCLK0.

The slave synchronizer is implemented with 1 1/2 sections of an F74 D-type flip-flop. The first flip-flop, in section A3 of the schematic, is clocked by the rising edge of SLAVEADR. As stated in previous paragraphs, SLAVEADR will not glitch and it will have one rising edge for every VME access to the GP slave. The output of this F74 is the signal REQUEST. REQUEST is then input to an F11 AND gate whose output is fed into the clock input of the next F74. The function of the F11 will be discussed below, but for right now assume that the other two inputs of this AND gate are both at a logic high so that REQUEST causes a rising edge on its output and clocks the F74. The output of this flip-flop is the signal BUSY (also called CLKVMEAD, clock VME address and data). BUSY is then fed into the D input of the next F74 which synchronizes it to the GP internal clock 2XCLK0. The output of the slave synchronizer is the output from this last flip-flop, the signal SREQ (synchronized VME request).

The three synchronization flip-flops, REQUEST, BUSY, and SREQ are set and reset with different conditions. The REQUEST flip-flop is set by the rising edge of SLAVEADR. When REQUEST is set there is the possibility of two conditions:

1. the BUSY flag is active, indicating that a previous VME access to the slave is still being serviced by the GP, or
2. the BUSY flag is *not* active, indicating that the GP has serviced all previous GP slave accesses and is ready to service another.

Under condition 2, all three inputs to the F11 gate, at the clock input to the BUSY flip-flop, will be at a logic high and the BUSY signal will be activated by REQUEST being activated.

Soon after the BUSY flag becomes active, the REQUEST flip-flop will be deactivated through the two F00 gates on its asynchronous clear input. The first of these F00 gates (page 7) outputs the signal SLVASYNDT* (slave asynchronous DTACK) which will be discussed further in the document. Notice that the clear signal into the REQUEST flip-flop's asynchronous clear input is only a pulse and not a level. The length of this pulse is determined by the delays introduced by the complement of the BUSY flip-flop going through the two F04 gates and the F11 gate back to the F00 which produces SLVASYNDT*. While BUSY is active, it prevents the REQUEST signal from clocking the BUSY flip-flop because it keeps the output of the F11 gate at a logic low.

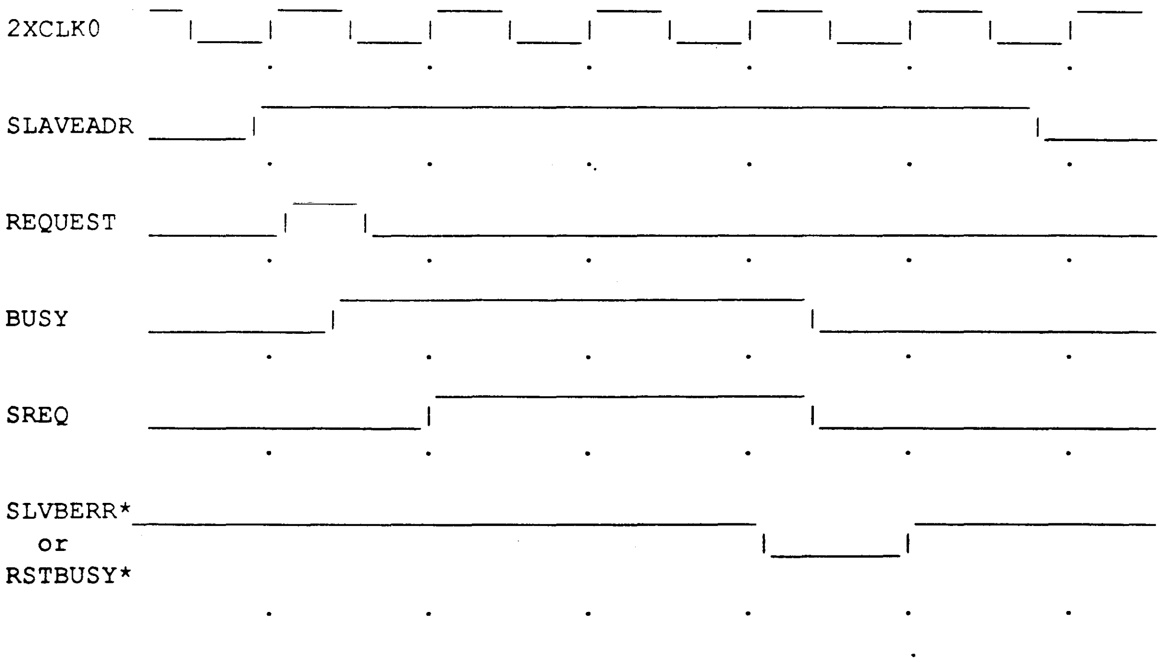
Condition 1 may occur because the GP slave provides an acknowledge to the VME master to complete the VME transaction as soon as possible and (when doing a write to shared memory) before it has actually completed the transaction.

This presents the case where overlapped VME transfers may occur. That is why the BUSY flag, when active, prevents any more REQUEST signals from clocking the BUSY flip-flop. When the GP is completely done with the requested transaction, it activates one of the two signals SLVBERR* (slave bus error) or RSTBUSY* (reset busy). RSTBUSY* and SLVBERR* are ORed together by the F11 gate in section A3 of page 7, and cause the BUSY and the SREQ flags to be deactivated through the asynchronous clear inputs of their respective flip-flops.

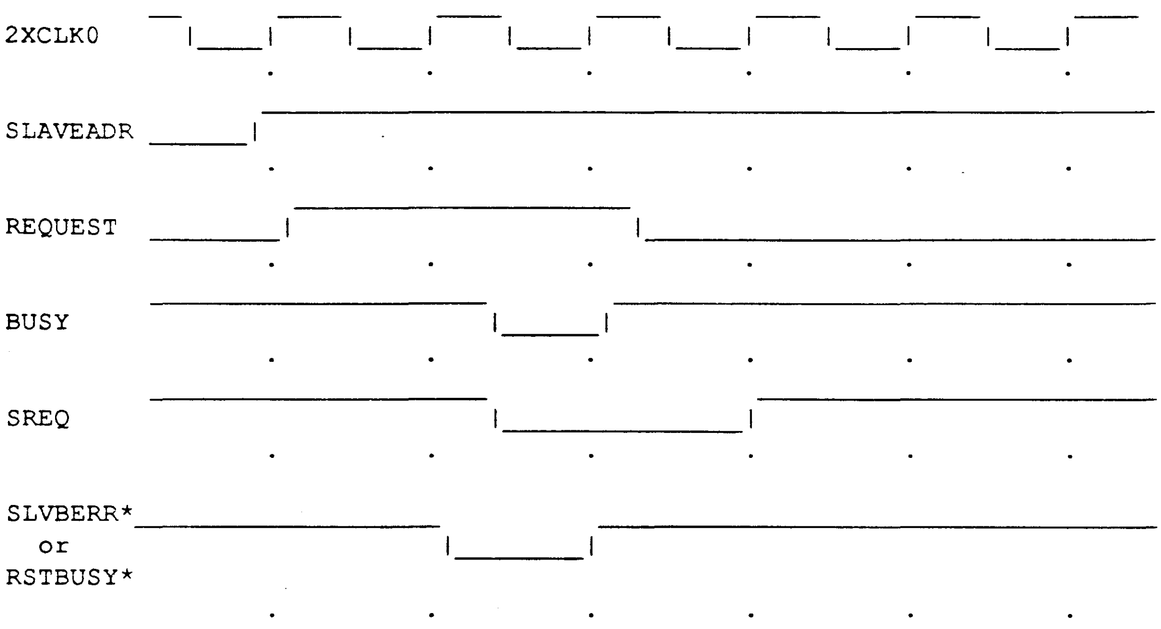
The REQUEST signal is activated by a rising edge on SLAVEADR and is deactivated by the first rising edge on BUSY. If BUSY is not active, it is activated by a rising edge on REQUEST. If BUSY is active, it is deactivated by an active level on either SLVBERR* or RSTBUSY*. If REQUEST is active when SLVBERR* or RSTBUSY* is active, then BUSY will be deactivated for a short time and will immediately be activated by the rising edge of SLVBERR* or RSTBUSY*. If REQUEST is not active when SLVBERR* or RSTBUSY* is deactivating BUSY, then BUSY will be activated by the first rising edge on REQUEST. SREQ is activated by a high logic level on BUSY and it is deactivated by an active level on either SLVBERR* or RSTBUSY*. The figure below shows the timing diagram depicting these different conditions.

Figure 5-1 *Slave Synchronizer Timing Diagram*

Condition 2: REQUEST occurs while BUSY is not active



Condition 1): REQUEST occurs while BUSY is active



Slave Control Latches

There is a latch, F373, in section D4 of page 7. This latch is used to hold the oui put from the slave address decode and slave control register decode during the VME access. The inputs to the latch are:

- the signal which indicates that the GP slave is being accessed, and
- the five signals which indicate which of the GP slave ports are being accessed: MSDATA*, MSADR*, GPCONREG*, BRDID*, and SHRDMEM*.

The output signals from the latch are given the same name as the input signals but with a prefix of "L" (latched). For example, the input signal MSDATA* (microstore data) becomes the output signal LMSDATA* (latched microstore data). The output from this latch is used primarily during VME reads of the GP slave to hold the GP slave data on the VME data bus. When the VME addresses the GP slave for a read, the GP decodes the VME address, routes the appropriate data onto the VME data bus, and issues DTACK* to the VME master indicating that the data requested is on the data bus. After DTACK* is issued, the VME master may remove the VME addresses. However, the data must stay on the bus until the VME data strobes are deactivated. Therefore, this component is used to latch the decoded control signals from the slave address decode from the time when DTACK* is activated until it is deactivated (DTACK* is activated by the GP slave control state machine and is deactivated when both data strobes, VMEDS1* and VMEDS0*, are deactivated).

The F374-type register in section C4 of the schematic is used to hold the decode control lines from the slave address decode and the VME control lines

VMEWRITE*, VMEDS1*, and VMEDS0*, during the VME access. The outputs of this register are used by the slave control state machine to determine what kind of transaction is required over the VME bus. The signals MSDATA* and SHRDMEM* are clocked into the register and become the output signals MSACCESS* (microstore access) and SMACCESS* (shared memory access). The VME control signals VMEWRITE*, VMEDS0*, and VMEDS1*, are input into the register and become SLVWRITE* (slave write), DS0*, and DS1*, respectively, at its output. The inputs to the register are clocked in by a complemented and delayed version of the REQUEST signal discussed above. The falling edge of REQUEST (when it is going inactive from the active state) clocks the inputs into the F374. The two F32 gates on the clock input provide a minimum delay required to meet worst case timing of the F374 (setup time Dn to CP). The outputs from the latch are valid until the next falling edge of REQUEST.

Slave Control State Machine

The slave control state machine is the center of the GP slave control function and is implemented with a 16R6A type PAL (programmable array logic) shown on the top right of page 7. The state diagram for the state machine is shown in the appendix (labelled "Figure 6. GP Slave Control State Machine."). This is a synchronous state machine and is clocked by the signal 2XCLK0.

There are five outputs from the slave control state machine:

- RSTBUSY* (reset busy flip-flop),
- SLVBERR* (slave bus error),
- SLVDTACK* (slave data transfer acknowledge),
- ENWRT* (enable write), and
- DENWRT* (delayed enable write).

RSTBUSY* is used to reset the flip-flops BUSY and SREQ in the slave synchronizer. SLVBERR* is used to activate the P1.BERR* (VME bus error) signal by resetting an F109 J-K flip-flop. SLVDTACK* is used to activate P1.DTACK* (VME data transfer acknowledge) by resetting an F109 J-K flip-flop. ENWRT* is used to clock in the data from shared memory to the shared memory VME-data-out register. DENWRT* is a combinational output from the PAL and is just a delayed version of ENWRT* with the addition of being qualified with the SLVWRITE* signal (DENWRT* is active whenever ENWRT* and SLVWRITE* are active). It is used as the D input of an F74-type flip-flop whose output WEGATE (write enable gate) is used to control the write enable pulse into the shared memory and the microstore.

The slave control state machine has four states labelled A, B, C, and D, as shown on the state diagram. These four states are indicated by bits SFF0 and SFF1 from the state machine.

On power up, the signal POR* causes the state machine to go to state A. The machine stays in state A waiting for the GP slave to be accessed. States B and C are traversed when the VME does a write access to the GP slave. State D is traversed when the VME does a read access to the GP slave.

When the GP board is powered on, the signal POR* (power on reset) becomes active and causes the state machine to go to state A. As long as the GP slave is not accessed by the VME master, the machine stays in state A. When the signal SREQ (synchronous VME request) becomes active, indicating that the VME master wants to access the GP slave, the state machine looks at the signals VMELWORD*, BRDID*, and SLVWRITE* to determine whether to go to state B or state D or to stay in state A.

If VMELWORD* is active when SREQ becomes active, it indicates that the VME master wants to do a long word transfer to the GP slave which is an illegal operation (the GP slave cannot do 32-bit data transfers). For this case, the state machine outputs the signal SLVBERR*, (which activates P1.BERR*), and stays in state A. P1.BERR* will end the transaction and will also indicate to the VME master that an error condition has occurred. The figure below shows the timing diagram for this condition.

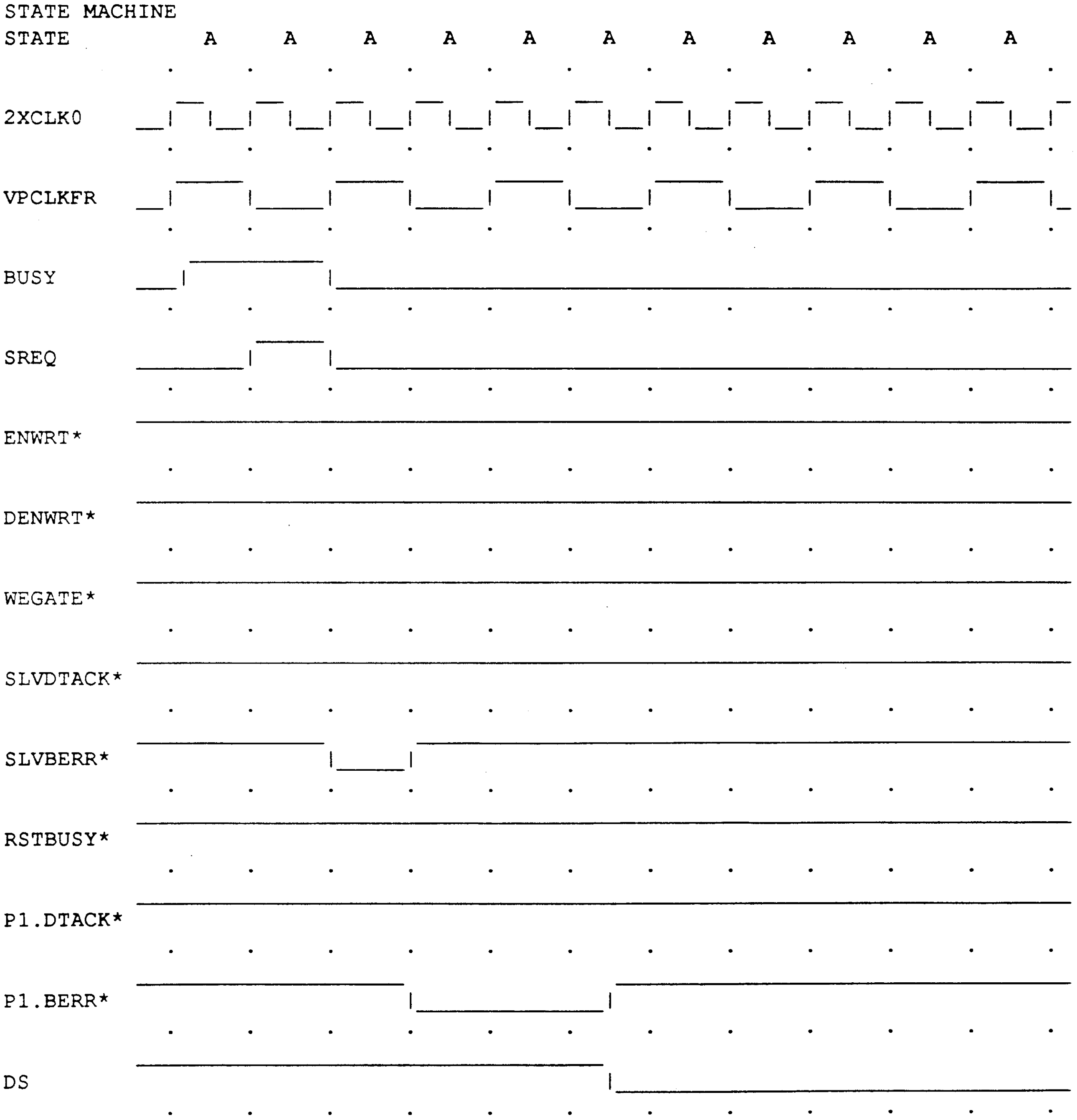
If VMELWORD* is not active when SREQ becomes active in state A, and SLVWRITE* and BRDID* are both active, then the state machine stays in state A and outputs SLVBERR*. This corresponds to the VME master doing a write to the GP board identification, which is a read only port, resulting in a bus error (P1.BERR* activated). The following figure shows the timing diagram for this condition.

Figure 5-2 *Slave Write Control Timing Diagram*

Longword Transfer or Write to the Board ID Register

VMELWORD* is active or

VMELWORD* is not active while BRDID* and SLVWRITE* are active



The following four figures show the timing diagram for GP slave write accesses.

When

- □ SREQ becomes active in state A, and
- □ VMELWORD* and BRDID* are not active, and
- □ SLVWRITE* is active,

then the VME master has requested a write to a GP slave port and the state machine goes to state B. The GP slave port that is being written-to determines which output signals are activated when going from state A to B.

1. If microstore data is being written—indicated by MSACCESS* being active—then the signal ENWRT* is activated.
2. If the microstore address or the GP control register is being written-to—indicated by MSACCESS* and SMACCESS* both being inactive—then the output signals SLVDTACK* and RSTBUSY* are activated.
3. If the shared memory is being written-to—indicated by SMACCESS* being active—the phase of VPCLKFR (free running VPCLK) determines the output.

Figure 5-3 *Slave Write Control Timing Diagram*

```
Microstore Address Write or GP Control Register Write
      VMELWORD*, BRDID*, SMACCESS*, and MSACCESS* are inactive,
      SLVWRITE* is active
```

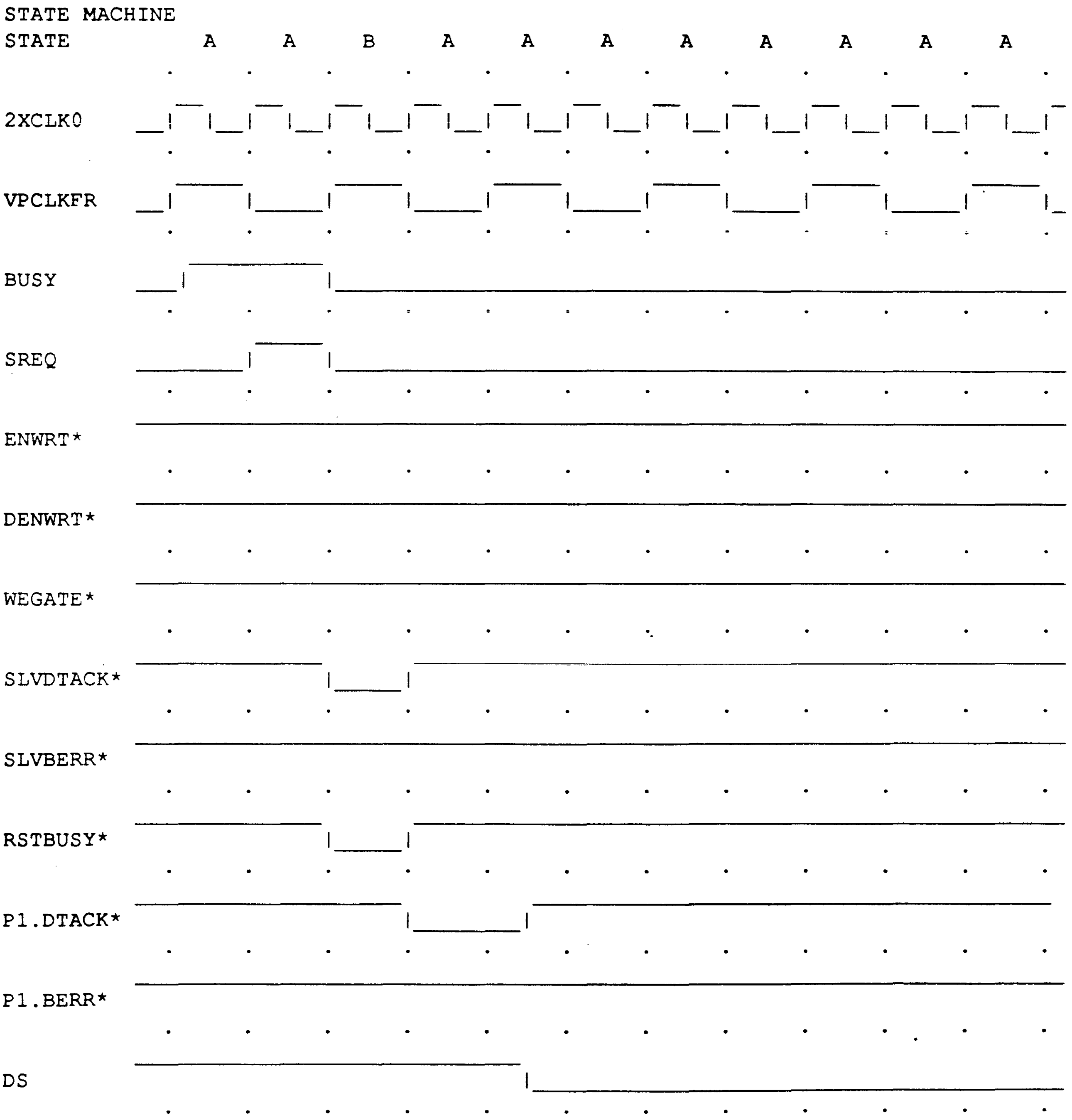


Figure 5-4 *Slave Write Control Timing Diagram*

Microstore Data Write
VMELWORD*, BRDID*, and SMACCESS* are inactive,
MSACCESS* and SLVWRITE* are active

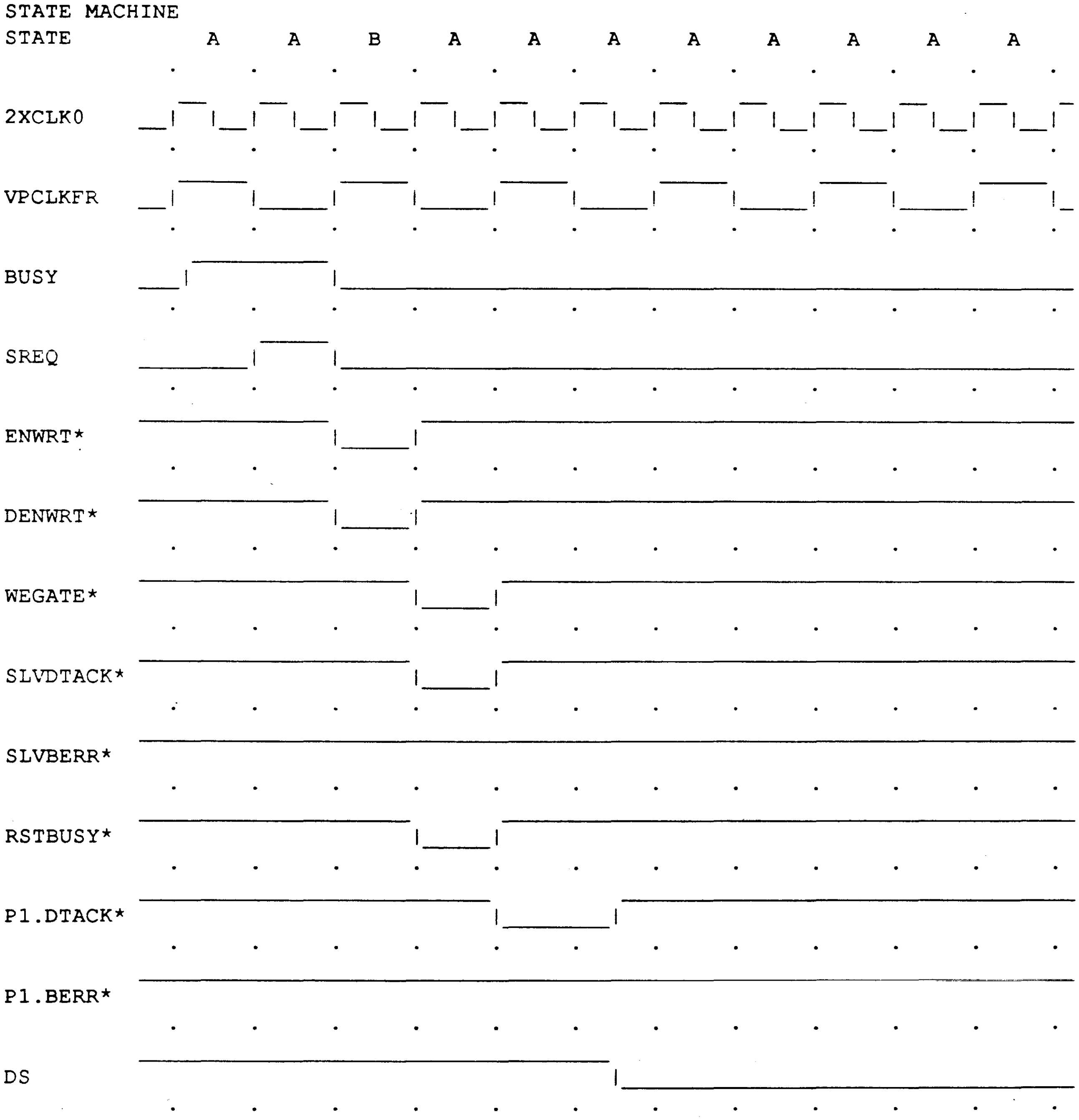


Figure 5-5 *Slave Write Control Timing Diagram*

```
Shared Memory Writes (extra synchronization period in state B)
     VMELWORD* and BRDID* are inactive,
     SLVWRITE* and SMACCESS* are active
```

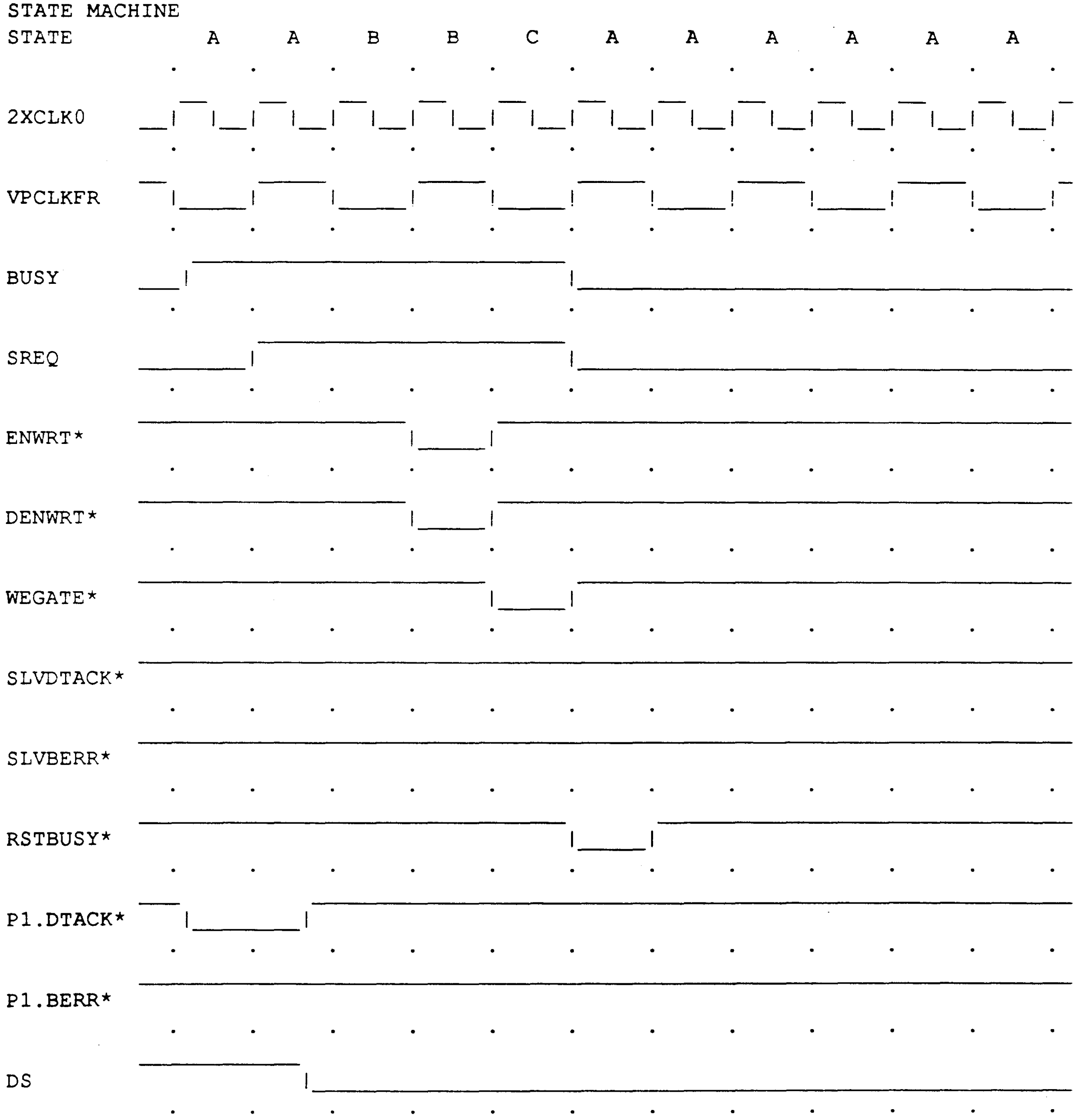


Figure 5-6 *Slave Write Control Timing Diagram*

```
     Shared Memory Writes (no extra synchronization period in state B)
           VMELWORD* and BRDID* are inactive,
           SLVWRITE* and SMACCESS* are active

STATE MACHINE
STATE          A     A     B     C     A     A     A     A     A     A     A
            .     .     .     .     .     .     .     .     .     .     .     .
             __    __    __    __    __    __    __    __    __    __    __    _
2XCLK0    __|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|
            .     .     .     .     .     .     .     .     .     .     .     .
             _____       _____       _____       _____       _____       _____
VPCLKFR   __|     |_____|     |_____|     |_____|     |_____|     |_____|     |_
            .     .     .     .     .     .     .     .     .     .     .     .
              ____________________________
BUSY      ___|                            |_____________________________________
            .     .     .     .     .     .     .     .     .     .     .     .
                   _______________________
SREQ      ________|                       |_____________________________________
            .     .     .     .     .     .     .     .     .     .     .     .
          ______________       _________________________________________________
ENWRT*                  |_____|
            .     .     .     .     .     .     .     .     .     .     .     .
          ______________       _________________________________________________
DENWRT*                 |_____|
            .     .     .     .     .     .     .     .     .     .     .     .
          ____________________       ___________________________________________
WEGATE*                       |_____|
            .     .     .     .     .     .     .     .     .     .     .     .
          ______________________________________________________________________
SLVDTACK*
            .     .     .     .     .     .     .     .     .     .     .     .
          ______________________________________________________________________
SLVBERR*
            .     .     .     .     .     .     .     .     .     .     .     .
          __________________________       _____________________________________
RSTBUSY*                            |_____|
            .     .     .     .     .     .     .     .     .     .     .     .
          ___          _________________________________________________________
P1.DTACK*    |________|
            .     .     .     .     .     .     .     .     .     .     .     .
          ______________________________________________________________________
P1.BERR*
            .     .     .     .     .     .     .     .     .     .     .     .
          ____________
DS                    |_________________________________________________________
            .     .     .     .     .     .     .     .     .     .     .     .
```

Because the VME only has access to shared memory during the second half of the VPCLK period (when it is a logic low), the write enable signal from the slave control state machine, ENWRT*, must be synchronized accordingly. Since 2XCLK0 is twice the frequency of VPCLKFR, VPCLKFR is going to be either in its logic high or logic low state between any state transitions. Therefore, if VPCLKFR is low when transitioning from state A to B, the output signal ENWRT* is activated. If VPCLKFR is high, no output will be generated when going from state A to B.

When in state B and the access is to shared memory, the state machine either:

□ stays in state B for one extra clock outputting ENWRT*, (if ENWRT* was not active the previous cycle), or

□ goes to state C with no output generated, (if ENWRT* was active the previous cycle).

The state machine stays in state C for one period and then goes to state A while outputting RSTBUSY*. When in state B and the access is not to shared memory, the state machine goes back to state A. While going back to state A and the access is to the microstore, the outputs SLVDTACK* and RSTBUSY* are activated. If the access is not to the microstore, then no output is generated while going from state B to A.

The following three figures show the timing diagram for the GP slave read accesses.

Figure 5-7 *Slave Read Control Timing Diagram*

Microstore Address or Data Read, GP Status Register Read, Board ID Read
VMELWORD*, SLVWRITE* and SMACCESS* are inactive

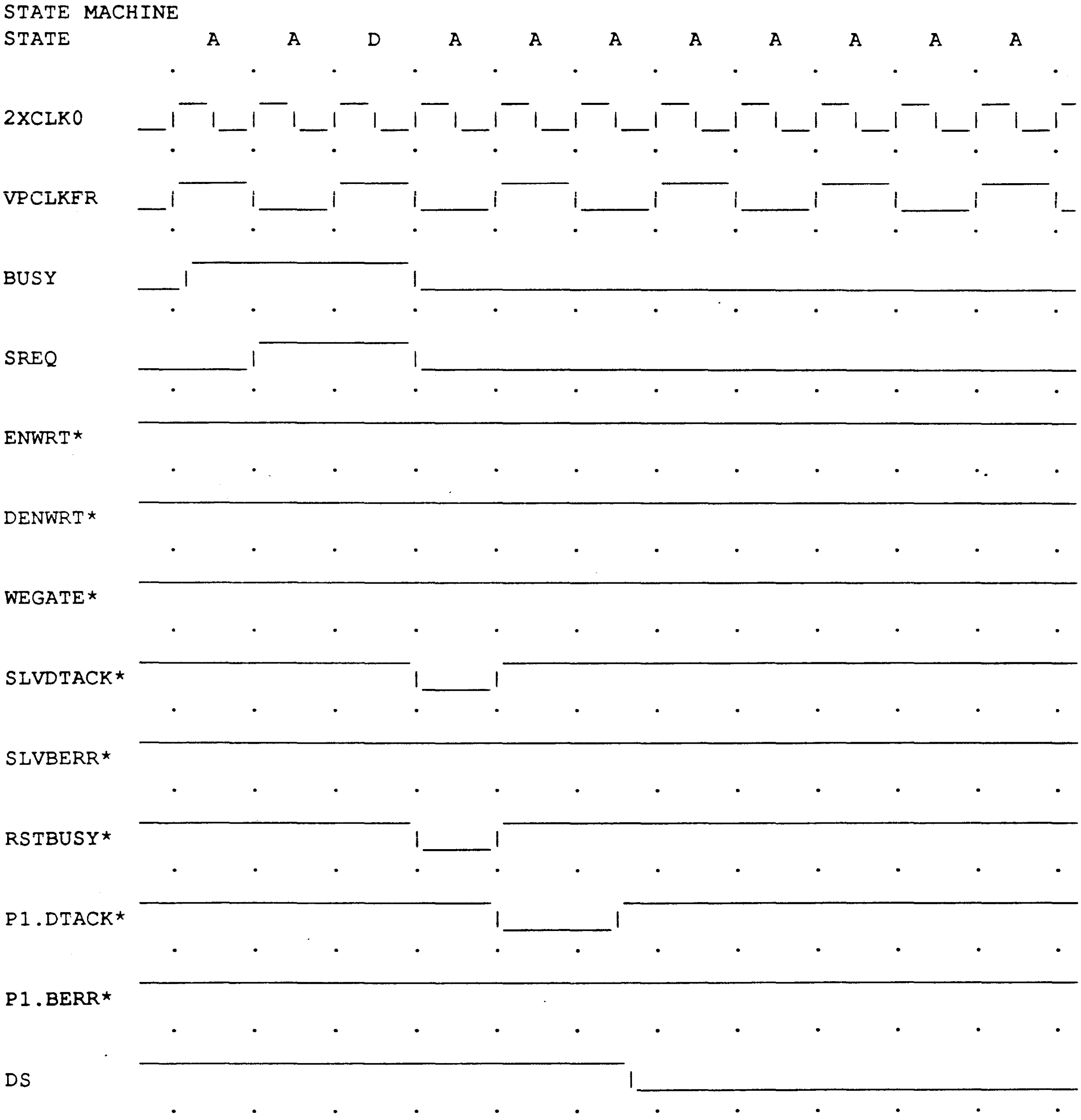


Figure 5-8 *Slave Read Control Timing Diagram*

```
Shared Memory Read (no extra synchronization period in state D)
      VMELWORD* and SLVWRITE* are inactive
      SMACCESS* is active
```

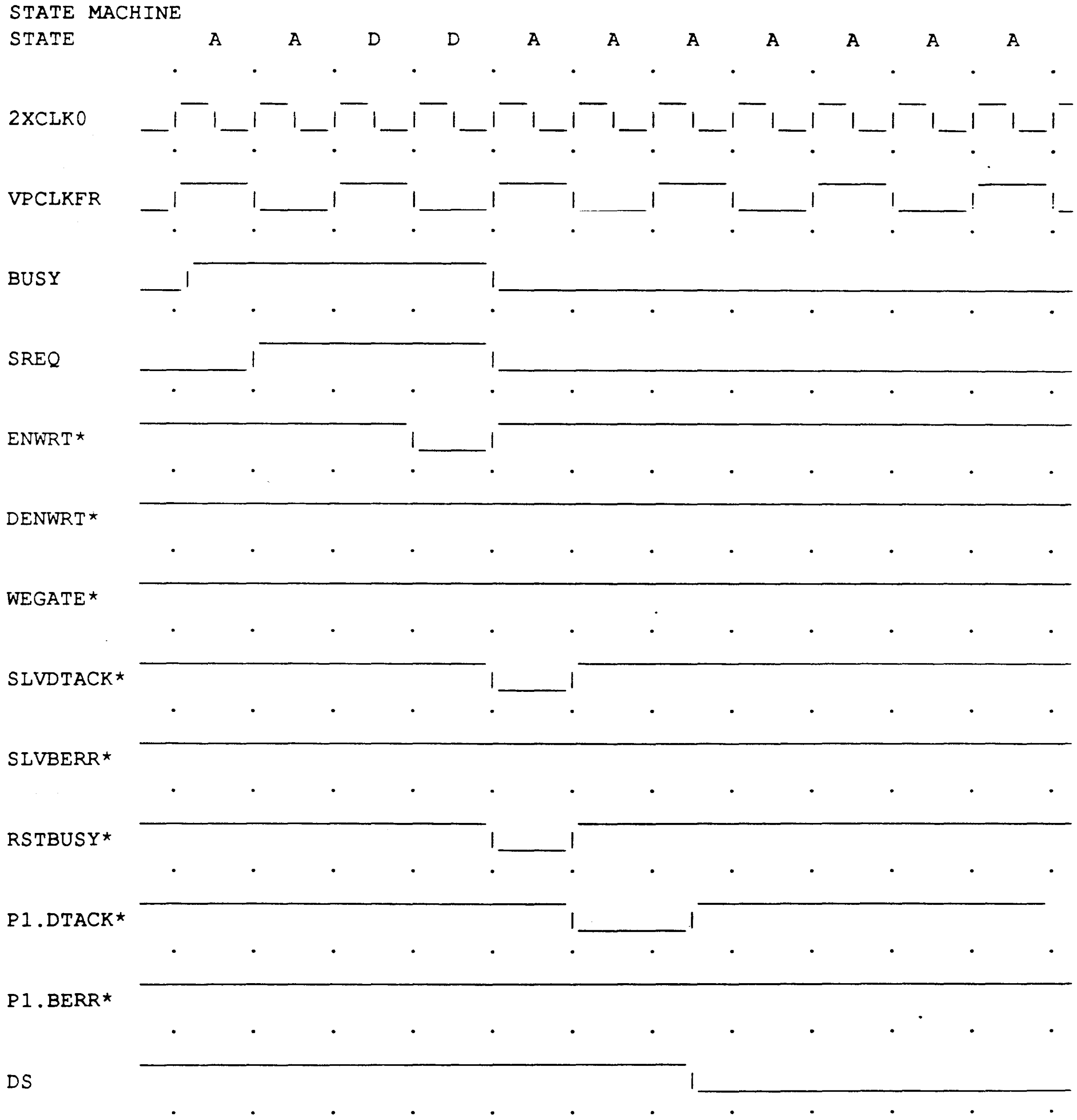
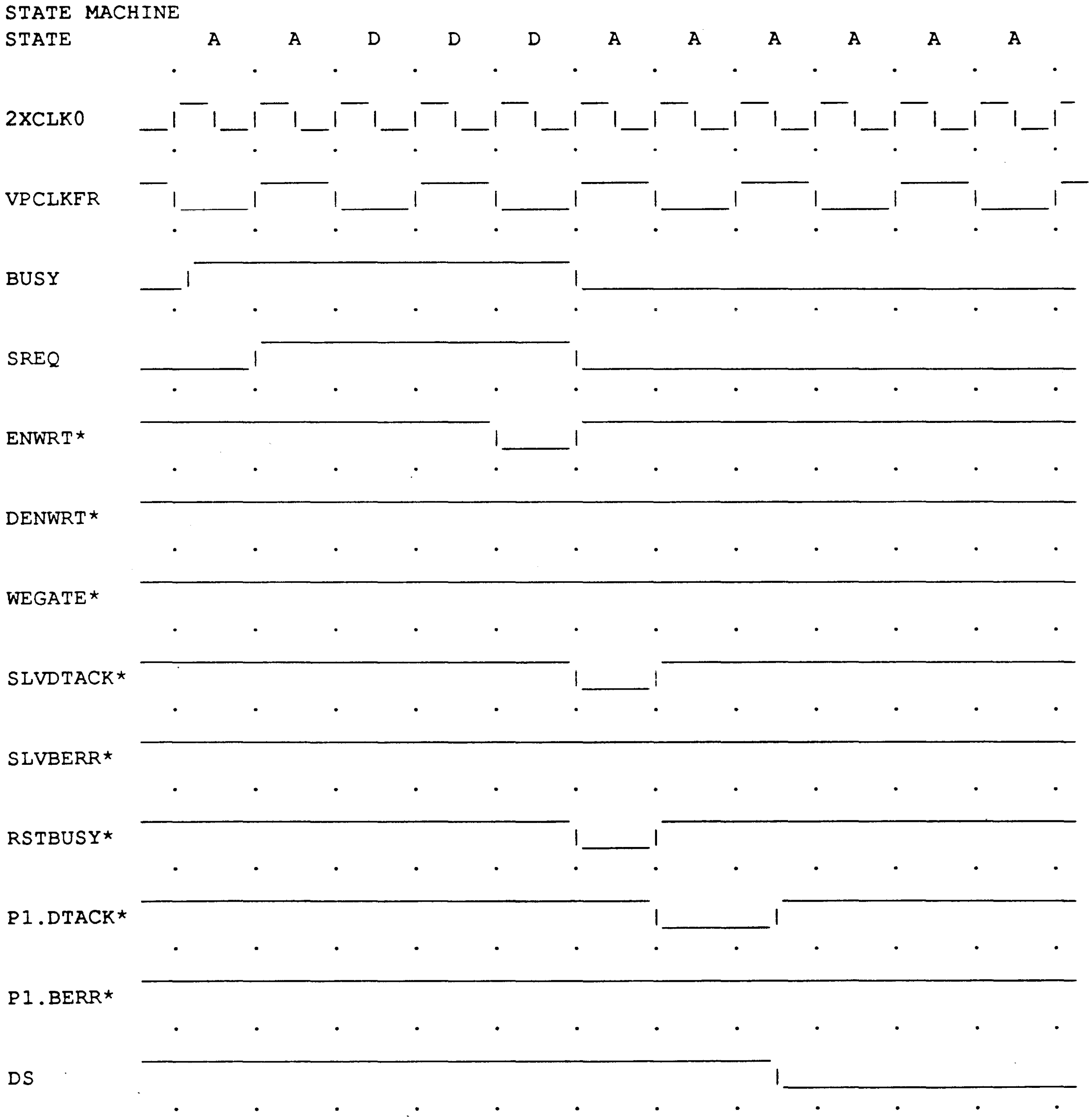


Figure 5-9 *Slave Read Control Timing Diagram*

```
Shared Memory Read (extra synchronization period in state D)
      VMELWORD* and SLVWRITE* are inactive
      SMACCESS* is active
```



- When SREQ becomes active in state A, and VMELWORD*, BRDID*, and SLVWRITE* are not active, then the VME master has requested a read from a GP slave port, whereupon the state machine goes to state D with no outputs activated.
- If the slave port being read is not shared memory, then the state machine goes back to state A while activating SLVDTACK* and RSTBUSY*.
- If the slave port being read is shared memory then the state machine will stay in state D two or three periods, synchronizing the output of ENWRT* with VPCLKFR. After ENWRT* is output the state machine goes to state A while activating SLVDTACK* and RSTBUSY*.

DTACK

The GP slave control logic activates P1.DTACK* to indicate to the VME bus master that the transaction requested has been completed. P1.DTACK* is output from the 7438 open collector gate, as required by the VME specification, driven by the F109 J-K type flip-flop shown in section B2 of page 7. P1.DTACK* is activated for shared memory write accesses by the SSI gates through the asynchronous clear input and for all other GP slave accesses by the SLVDTACK* signal through the K input. P1.DTACK* is also activated by INTDTACK* (interrupt data transfer acknowledge) through the K input. SLVDTACK* and INTDTACK* are ORed by the F00 gate and input to the K input of the flip-flop. The signal INTDTACK* is activated by the GP master interrupt requester and the signal SLVDTACK* is activated by the GP slave control state machine. Both INTDTACK* and SLVDTACK* are one 2XCLK period-long signals and either one being active causes P1.DTACK* to be activated.

The logic to drive the asynchronous clear input is done using the F11, F04, and the F32 gates. The activation of P1.DTACK* through the asynchronous clear input is related with generating P1.DTACK* as soon as possible when shared memory is written to. When the VME master does a write operation to the shared memory, P1.DTACK* is generated before the actual write operation is done. The VME cycle is, therefore, completed very fast. The address and data for the write to shared memory is clocked into registers before the P1.DTACK* is generated so that the operation can be completed without any problems. P1.DTACK* is activated for writes to shared memory by the signal SLVASYNDT* (slave asynchronous data transfer acknowledge), which is generated by the rising edge of BUSY in the slave synchronizer, and the signals VMEREAD*, VMELWORD*, and VMEA<15> being inactive (that is, the condition of shared memory writes that are not long word transfers). P1.DTACK* is deactivated by the condition of both VMEDS0* and VMEDS1* being inactive (that is, P1.DTACK* is deactivated when both of the VME data strobes are deactivated).

Bus Error

When an illegal operation is being requested by the VME master to the GP slave, the GP slave control logic generates the signal P1.BERR*. This signal indicates to the VME master that the transaction requested has not been completed satisfactorily.

The GP slave signals a bus error when a

- 32-bit data transfer is requested, or when a
- write of the board ID is requested by the VME bus master.

The P1.BERR* signal is output from a 7438 open collector gate, as required by the VME specification, driven by the F109 J-K type flip-flop shown in section C2 of page 7. P1.BERR* is activated by the signal SLVBERR* generated by the slave control state machine. P1.BERR* is deactivated by the condition of both VMEDS0* and VMEDS1* being inactive (that is, P1.BERR* is deactivated when both of the VME data strobes are deactivated).

WE Gate

The F74-type flip-flop in section D1 of page 7 just delays the signal DENWRT*, generated from the slave control state machine, by one 2XCLK period and outputs the signal WEGATE and its complement WEGATE* (write enable gate). WEGATE and WEGATE* are the signals used to control the write enable pulse into the VME side of both shared memory and the microstore. The actual use of this flip-flop is to synchronize the rising and falling edges of the write enable pulse to the 2XCLK edges; the direct output of DENWRT* from the slave control state machine PAL had too much delay.

5.4. VME Slave Registers

The registers used when the GP is a VME slave include the:

- board ID,
- GP status/control register,
- microstore address counter,
- microstore data transceiver,
- shared memory VME address pointer,
- shared memory VME data in register, and the
- shared memory VME data out register.

These components serve the purpose of routing and holding data between the VME bus and the GP internal devices. They are controlled by the various signals generated from the VME slave control logic.

Board ID

The board ID (identification) is simply a unique 8-bit binary value set by jumpers which can be routed to the VME bus when it is addressed by the VME bus master. It is shown in the top left corner of schematic page 6. The binary value set by the jumpers distinguishes or identifies the GP board on the VME bus backplane. The value set by the jumpers is isolated from the GP internal VME data bus, VMED<15..00>, by ALS244 buffers. When the GP slave board ID port is read by the VME bus master, the signal LBRDID* becomes active and drives the VMED<15..00> 16-bit data bus—although only the lower byte (VMED<07..00>) is valid. The operation of writing to the GP slave board ID is not allowed and results in the slave control logic asserting bus error.

GP Status/Control Register

The GP status register is shown on left side of page 6 and the GP control register is in the middle of the same page . Both the status and the control registers occupy the same address location on the GP slave address space. When a read of this location is requested the status register is implied and when this location is written the control register is implied. The control register holds various flags which are written by the host processor to control the operation of the GP. The status register holds various flags which are read by the host processor to determine the operating status of the GP.

NOTE *See the GP Hardware Reference Manual (Sun part number 800-1190) for the bit definitions of these registers and see the appropriate sections of this document for the discussion of the function of each signal.*

The control register is implemented using an F109 J-K flip-flop and an ALS374 octal D-type register.

The GP control register clocks in the contents of the VMED bus when the VME bus master does a write access to the GP slave status/control register. This condition is decoded by the two F32 gates and the F04 gate shown in sections A6 and A7 of page 6. The signal GPCONREG* is output from the slave control register decode; the signal VMEWRITE* comes from the VME control transceiver; the signal SLVDTACK* is output from the slave control state machine. Both GPCONREG* and VMEWRITE* occur early in the access cycle and therefore the clock input signal to the ALS374 and the F109 just follows the SLVDTACK* signal (SLVDTACK* is an active low pulse of approximately one 2XCLK period). So the clock line of the Control Register looks like an active high pulse of approximately one 2XCLK period duration. The F109 flip-flop is needed because two of the bits of the Control Register need to be controlled in a certain way. The CLRINTFLG needs to be cleared asynchronously by the IFLAG signal and the INTEN flag is set, reset, and unchanged according to an encoding of two bits of the data bus.

The status register is implemented with two F373-types of octal latches. The outputs of the F373 devices are normally tri-stated, and it is in the flow-through mode. When the VME bus master does a read access of the GP slave status/control register the inputs are latched and the outputs are enabled onto the VMED bus. The condition of the status/control register being read is decoded by the F32 gate in section A8 of page 6. The inputs to this F32 are the signals VMEREAD* (a complemented version of the VMEWRITE* signal from VME control transceiver), and LGPCONREG*, also from the slave control latches. The F373 devices stay in this condition until the VME master deasserts the data strobes which deactivate LGPCONREG*.

Microstore VME Address Counter and Data Transceiver

The microstore VME interface includes the address counter and and the data transceivers on page 8 of the schematics.

The microstore VME address counter is implemented with five F163-type synchronous counters. The counter can be loaded or counted by doing a VME access, or its contents can be read by a VME access. The data inputs to the counter are the GP internal VME data bus signals, VMED<15..0>. The outputs of the counter are the Microstore VME address bits VMEMSA<15..0> and also two other bits (LSBs of the counter) which are used to decode the microstore columns (discussed in the Microstore section of this document).

The counter is loaded when the VME bus master does a write access to the GP slave Microstore Address Register. The counter is enabled to count (once) every time the VME bus master does a read or a write access to the GP slave Microstore Data Register. Both the load and the count operations are done similarly. The load enable or the count enable pin of the F163s are activated and a low-going pulse is applied to the clock input. The following rising edge of this pulse causes load or count to occur.

The logic to control the loading and the counting of the address counter is on the lower left comer of page 8. It consists of F32-, F02-, and F04-types of gates. The load enable input is activated with an output from an F32 gate with inputs SLVWRITE*, from the slave control latches, and MSADR*, from the slave control register decode. When both these signals are active the load enable input becomes active. The load enable input signal also acts as one input to the F02 gate, with the other input being SLVDTACK*. When both become active the output of this F02 becomes a logic high and causes the clock input of the F163s to be pulled to a logic low through another F02 gate. SLVDTACK* is a signal which is active for approximately one 2XCLK period. When SLVDTACK* becomes inactive, the clock input to the F163s accordingly goes back to a logic high. Since the load enable was active, the rising edge on the clock input loads the counter with the data on its the D inputs. The count enable input to the counters is the signal LMSDATA*, from the slave control latches, delayed by two F32 gates and an F04 gate. LMSDATA* is active when the VME master is accessing the microstore data register. LMSDATA* also activates one input of an F02 with the other input being DS*. DS* will already be active when LMSDATA* becomes active so that the clock input to the F163s will be pulled to a logic low through another F02 gate. When DS* goes inactive, the clock input to the counters will, accordingly, go to a logic high. Since the count enable input was already active, the rising edge on the clock will increment the counter.

The contents of the microstore address counter can be read by the VME master by doing a read access to the GP slave microstore address register. This is implemented with two ALS244 buffers shown on the left of page 8. The buffers isolate the outputs of the counter, VMEMSA<15..00>, from the GP internal data bus, VMED<15..00>. The output of the buffers are normally tri-stated. When the signals VMEREAD*, from the VME control transceiver, and LMSADR*, from the slave register decode, become active the outputs of the buffer are enabled. The outputs will be enabled until the LMSADR* becomes inactive.

The microstore data transceivers are implemented with ALS245 devices and are shown on the right of page 8. The microstore is divided into four columns as seen from the VME side (see the Microstore section in this document for a more thorough discussion of this division). The four columns are addressed by the two LSBs of the microstore address counter VMED<00> and VMED<01>, from which are derived VMEMSAC0 and VMEMSAC1. These two LSBs (VMEMSAC0 and VMEMSAC1) are input to the microstore column select which is an F139 decoder shown in section A5 of page 8. This decoder is enabled only when the signal LMSDATA* is active (that is, when the VME bus master is accessing the microstore data). The output from this decoder is the four column enable signals MSCOL3* (column 3), MSCOL2*, MSCOL1*, and MSCOL0*. These column enable signals are used to select the appropriate data transceiver to turn on and the correct write enable to activate. Only one of the column enable signals are active at any one time.

The direction of the microstore data transceivers is controlled by the signal VMEWRITE* and their output enables are controlled with the outputs from an F157 multiplexer. The multiplexer select line is also controlled with the signal VMEWRITE*. The ALS245 transceiver outputs are controlled by one of two signals, depending on whether the VME master is doing a read or a write.

- When the VME master is doing a write, the transceiver outputs are controlled by the write enable signals of the microstore, MSC3WE*..MSC0WE* (microstore column n write enable).
- When the VME master is doing a read, the transceivers are controlled by the column select signals, MSCOL3..0*, from the column select decoder.

When the VME master is doing a read of the microstore, the signal VMEWRITE* is a logic high (deasserted) which allows the transceivers to transmit the microstore data signals, MSTR55..MSTR00, to the VMED bus. Also, the F157 multiplexer selects the column enable outputs from the F139 column select decoder. When the LMSDATA* signal becomes active, one of the four column enable signals, MSCOL3..0*, is activated and passes through the F157 to enable the output of the appropriate transceiver. The transceiver drives the VMED bus until LMSDATA* becomes inactive causing the column enable signals to become inactive.

When the VME master is doing a write, the signal VMEWRITE* is a logic low. This allows the VMED bus to drive the microstore data signals, MSTR55.. MSTR00. The F157 multiplexer selects the microstore write enable signals, MSC3WE*..MSC0WE*. The microstore write enable signals are generated by the four F32 gates in section C8 of schematic page 11. One of the write enable signals is activated if its corresponding column enable signal is active and the signal WEGATE*, from the slave control logic on page 6, is active. The WEGATE* signal is an active low pulse of approximately one 2XCLK period duration. Because the column select signals, MSCOL3*..MSCOL0*, completely overlap WEGATE*, the write enable signals, MSC3WE*..MSC0WE*, will be low-going pulses of approximately one 2XCLK period duration. When one of the write enable signals becomes active, it will pass through the F157 mux and enable the appropriate data transceiver to drive the microstore data lines. The transceiver will drive the data lines until the write enable line is deactivated by WEGATE* going inactive, passing through the F157, and tri-stating the transceiver output.

Shared Memory VME Address Pointer and Data Registers

The shared memory is interfaced to the VME bus with the support of three registers:

- the address pointer,

- the data in register, and
- the data out register.

These registers are shown on page 13 of the schematics.

Shared memory is accessed by the VME master when it addresses the second half of the GP slave address space. When the shared memory is accessed, the VME address is loaded into the address pointer.

If the VME access is a read access, then the shared memory contents (corresponding to the address in the address pointer) are loaded into the data out register. These contents are then enabled onto the VME data bus.

If the VME access is a write access, then the contents of the VME data bus are loaded into the data in register. These contents are then loaded into the location in shared memory which corresponds to the address in the address pointer.

The address pointer is shown section A6 of page 13. The address pointer is implemented using F374 devices. The contents of the GP internal address bus, VMEA<16..01>, are clocked into the register on the rising edge of the signal CLKVMEAD (clock VME address and data). CLKVMEAD is the same signal as BUSY in the slave synchronizer on page 7. (BUSY was changed to CLKVMEAD to give it a name more fitting its application here.) The output enable control of the F374 devices will be discussed in the shared memory section in this document.

The data out register is shown in the top right of page 13. It is also implemented with F374 devices. The output data from the shared memory, SMDO<15..00> (shared memory data out), is clocked into the register with the F32 gate. The inputs to the F32 gate are 2XCLK1 and ENWRT*. ENWRT* is an output from the slave control state machine and is a low going pulse of one 2XCLK period duration. When ENWRT* is active (logic low), the second half of the 2XCLK1 period (when it is also low) causes the clock input of the F374s to become a logic low. When 2XCLK1 goes high the clock input of the F374s goes high causing the data to be loaded into the register. The data from the register is output to the VMED bus when the signals LSHRDMEM*, from the slave control latches, and VMEREAD*, from the VME control transceivers, are both active. The data is output until LSHRDMEM* goes inactive (when the VME data strobes are removed).

5.5. VME Master Control

The VME master control allows the GP to obtain the VME bus as a bus master, supervise data transfers as a bus master, and cause an interrupt over the VME bus. The logical blocks of the GP VME master control are the:

- control signal synchronizer,
- bus requester,
- bus interrupter,
- bus master data transfer controller,

- timeout counter,
- miscellaneous master logic, and the
- PP VME interface.

The PP signals to the GP VME master controller that it wants to use the VME bus by activating the signal VMEBUSY* (VME master is busy). VMEBUSY* is output from the F109 flip-flop in section B6 of page 31. VMEBUSY* is activated by the signal STRTVME* (start VME transfer) from the PP miscellaneous controls. VMEBUSY* is deactivated (and its complement, VMERDY, asserted) by the signal RSTVMEBSY from the Master Data Transfer Controller when the VME transfer has been completed. VMEBUSY* is activated once for every VME transfer that is done by the GP VME master.

While activating VMEBUSY*, the PP processor also updates the signal MASTERWRT* (GP VME master write). This signal indicates to the Data Transfer Controller whether to do a read or a write access over the VME bus. It is output from the F109 flip-flop shown in section C3 of page 31. The MASTERWRT* signal is activated by the signal VMEWR* (write to the VME bus) from the PP miscellaneous controls. MASTERWRT* is deactivated by the signal RSTVMEBSY from the data transfer controller when the VME transfer has been completed.

Control Signal Synchronizer

The control signal synchronizer takes asynchronous VME bus control signals and synchronizes them to 2XCLK, which is used by the GP VME master control logic. The synchronizer is an F374 device which is shown in section A5 of schematic page 6. Clocked by 2XCLK0, the F374 synchronizes the following signals:

- P1.IACKIN* (interrupt acknowledge in),
- P1.BG3IN* (bus grant 3 in),
- VMEAS* (address strobe),
- P1.DTACK* (data transfer acknowledge),
- P1.BERR* (bus error),
- VMEDS0* (data strobe 0), and
- an OR of the four bus request signals, P1.BR3*..P1.BR0*.

The output signals are named similarly, with a prefix of "S":

- SIACKIN*,
- SBGIN*,
- SAS*,
- SDTACK*,
- SBERR*,
- SDS0*, and

□ SBR.

VME Bus Requester

The bus requester senses a request from the PP processor to use the VME bus and sends a request to the VME bus arbiter to obtain the bus. When the VME bus is obtained, the bus requester signals the GP master data transfer controller to carry out the VME bus transfer requested by the PP. The GP bus requester operates on level three and is an ROR (release on request) requester. The bus requester is shown in the lower left of schematic page 31. It is implemented with a state machine and logic in a 16R4A PAL, delay lines, and an F74 type flip-flop.

The state machine is the central control of the bus requester. Figure 9 in the appendix shows the state diagram of the bus requester. There are four states in the state machine.

□ State A implies that the GP does not have control of the VME bus and does not require it.

□ State B means that the GP wants to acquire control of the VME bus but does not yet have it.

□ The state machine goes to state C when it has control of the VME bus. It stays there until the VME transaction requested by the PP is completed by the master data transfer controller.

□ If the state machine is in state D it implies that the transaction requested by the PP has been completed, but, since no other device has requested it, the GP still has control of the VME bus.

When the GP is powered up, the signal POR* becomes active and resets the state machine to state A (signalling the GP does not have control of the VME bus). The state machine stays in state A as long as VMEBUSY* is not active. When the PP processor requests a VME transfer, it activates VMEBUSY* and the state machine goes to state B. When in state B, the signal BR* (bus request) is activated. BR* is inverted and input to a 7438 open collector gate which drives the VME bus request signal P1.BR3*. The state machine keeps asserting BR* and stays in state B until the bus is acquired and the previous controller of the bus completes its last transaction. The VME bus is acquired when the bus grant daisy chain signal P1.BG3IN* reaches the GP board. The last transaction of the previous controller may still be active when the bus is acquired because the bus arbitration can occur while a VME data transaction is active.

Upon sensing the P1.BR3* signal, the bus arbiter will start the bus grant daisy chain. If no other boards between the GP and the bus arbiter have requested the bus, the daisy chain signal P1.BG3IN* will arrive at the GP board. When P1.BG3IN* arrives, a decision must be made by the GP to

1. pass the bus grant out, by activating P1.BG3OUT*, or
2. not to pass it and activate BBSY* indicating that the GP has taken over control of the bus.

In order to make this decision, the VMEBUSY* signal is sampled when the P1.BG3IN* signal arrives at the GP. If VMEBUSY* is active at sample time then the GP will acquire the bus; if it is not active then the GP will propagate the daisy chain by activating P1.BG3OUT*. VMEBUSY* is sampled by using an inverted version P1.BG3IN* to clock an F74 flip-flop which has VMEBUSY* as the D input. The output of this F74 is the signal SBUSY* (synchronized VME busy). P1.BG3IN* is also routed through a 50 ns delay line to become BGIN (bus grant in). The 50 ns delay occurs in parallel with the clocking of the F74, and the metastable output of the F74 will stabilize in 50 ns (metastability may occur because P1.BG3IN* and VMEBUSY* are asynchronous to each other). The two signals SBUSY* and BGIN are all that is necessary to decide whether or not to pass the bus grant out. The bus grant daisy chain is propagated (that is, P1.BG3OUT* activated) if SBUSY* is not active (the sampling did not catch VMEBUSY* as active) and BGIN is active. This is the equation used in the PAL for generating P1.BG3OUT* with the supplanting of BBSY* signal not being active (GP does not control the bus) and the POR* signal not being active (reset deactivates P1.BG3OUT*).

While in state B, the state machine uses SBGIN*, from the control signal synchronizer, to sense that the bus grant daisy chain has propagated to the GP. If SBUSY* is also active, indicating that the daisy chain was not propagated, then the state machine determines that the GP can acquire the bus. The only other condition that must be satisfied before the bus can be controlled is that the previous bus controller must have completed its last transaction. This is determined by sensing the signals SAS*, SDTACK*, and SBERR*. If all three of those signals are inactive it means that the bus is inactive and another transaction can be started. Therefore, when in state B, the bus request signal, BR*, is activated until SBGIN and SBUSY signals are active and SDTACK*, SAS*, and SBERR* signals are inactive. When that occurs the state machine activates the signal BBSY* (bus busy) indicating that the GP has control of the VME bus (GP has become the bus master). BBSY* is inverted and input to a 7438 open collector gate which drives the VME bus busy signal P1.BBSY*.

The state machine activates the signal BBSY* when in state B and stays in state B one more cycle and then goes to state C. It stays in state B this one extra cycle to ensure that BBSY* signal is active for at least two clock periods (it is possible for BBSY* to be deactivated as soon as the state machine goes to state C). This guarantees that BBSY* signal pulse width is greater than 90 ns as required by the VME specification.

When BBSY* is activated, it clears an F74 flip-flop that asserts the signal DISSBUSY* (disable SBUSY*). This F74 is shown in section B7 of page 31. DISSBUSY* in turn deactivates SBUSY*. This is necessary because VME-BUSY* signal will stay active while the GP master data transfer controller is carrying out the VME transfer. During this time it may be necessary to release the bus to another bus requester. If some other board requests the bus, the signal SBR, which is a synchronized version of the OR of the four signals P1.BR3*..P1.BR0* will be activated. The state machine will then deactivate BBSY* which will prompt the bus arbiter to start the bus grant daisy chain. If the daisy chain reaches the GP in this state, the daisy chain should be propagated. The daisy chain input signal, P1.BG3IN*, will be sampling the VMEBUSY* signal—which was already sampled when the bus was being acquired for the GP. This sampling must not be allowed because it would keep the GP from propagating the daisy chain. Therefore DISSBUSY* keeps the sampling F74 output at a logic high, which allows the daisy chain to be propagated. DISSBUSY* was activated by BBSY* going active; it is deactivated by a delayed version of VMEBUSY* going inactive. This delay is 50 ns and is necessary in case BBSY* and VMEBUSY* are deactivated at the same clock edge. The delay line provides enough margin to guarantee deactivation.

After acquiring the bus, the state machine goes to state C and stays there until the GP master data transfer controller completes the transaction requested by the PP. While in state C, the BBSY* signal is activated—unless SBR* (from the control signal synchronizer) is activated. If SBR* *is* activated it means that some other board is requesting to become bus master, whereupon the GP will deactivate BBSY* to let the VME bus arbiter start the arbitration. It is also possible that SBGIN is still active in three cases—from the time the GP obtained the bus, some other board requested the bus, or the GP VME transaction had been completed. For this case, the state machine stays in state C until SBGIN is deactivated.

State C conditions can be described as follows:

1. If BBSY and VMEBUSY are active and SBR is not active, stay in state state C and keep asserting BBSY (the GP VME transaction has not been completed).

2. If BBSY, VMEBUSY, SBR, and SBGIN are active, stay in state C and keep asserting BBSY (the GP VME transaction has not been completed, SBGIN has not been deactivated yet from the time that the GP obtained the bus).

3. If BBSY, VMEBUSY, and SBR are active and SBGIN is not active, stay in state C and deactivate BBSY (the GP VME transaction has not been completed, some other board has requested the bus and SBGIN has been deactivated from the time when the GP obtained the bus).

4. If BBSY and SBGIN are active and VMEBUSY is not active, stay in state C and keep asserting BBSY (the GP VME transaction has been completed, no bus requests arrived during the transaction so the GP still has the bus, SBGIN has not been deactivated yet from the time that the GP obtained the bus, so keep the bus until SBGIN is removed).

5. If VMEBUSY is active and BBSY is not active, stay in state C but don't assert BBSY (BBSY has been deasserted before because of a bus request, the GP VME transaction has not been completed).

6. If BBSY is active and VMEBUSY, SBR, and SBGIN are not active, go to state D and keep asserting BBSY (the GP VME transaction has been completed, no bus requests have arrived or are currently pending, SBGIN has been deactivated from the time when the GP obtained the bus, the GP still has the bus).

7. If BBSY and SBR are active and VMEBUSY and SBGIN are not active, go to state A and deassert BBSY (the GP VME transaction is ending at the same time that a bus request arrives, SBGIN has been deactivated from the time when the GP obtained the bus, so give up bus ownership).

8. If BBSY and VMEBUSY are not active, go to state A and don't assert BBSY (bus ownership has been given away previously, the GP VME transaction is completing).

Condition 6 causes the state machine to go to state D. The state machine stays in state D waiting for either a bus request from another board, (indicated by the SBR* signal being activated), or the PP to start another VME transaction, (indicated by the VMEBUSY* signal being activated).

- If neither VMEBUSY* or SBR* are asserted, the state machine stays in state D and keeps the bus by asserting BBSY*.
- If SBR* becomes active when VMEBUSY* is not active, then the bus is given up by deasserting BBSY* and going to state A.
- If VMEBUSY* becomes active, then bus ownership is kept (that is, BBSY* is kept activated) and the state machine goes to state B. It goes to state B in this case to guarantee that BBSY* is kept active for at least two periods after VMEBUSY* is activated (the state machine will stay in state B for one period and then go to state C).

NOTE *The GP keeps ownership of the bus if, while in state D, both VMEBUSY* and SBR become active at the same time.*

Data Transfer Controller

The data transfer controller carries out the VME transfer that is requested from the PP. When it receives the request to do a transfer (indicated by VMEBUSY* becoming active), and the GP bus requester has obtained the VME bus, the data transfer controller assumes the role of the VME bus master and completes the transaction requested. The data transfer controller is implemented as a state machine and logic contained in a 16R6A PAL shown in the top right of page 31.

There are four outputs from the state machine. The signal RSTVMEBSY* is activated to indicate to the PP and the bus requester that the transfer has been completed. RSTVMEBSY* is used to reset several control flags. Additionally, the state machine outputs the signals MASTERAS* (VME master address strobe), MASTERDS1*, and MASTERDS0* (VME master data strobe 1 and 0). MASTERAS*, MASTERDS1*, and MASTERDS0* are input to the VME control transceivers (page 6) and, when the GP is the VME bus master, are used to drive the VME bus control signals P1.AS*, P1.DS1*, and P1.DS0*, respectively.

The data transfer controller state machine state diagram is shown in Figure 10 of the appendix.

The state machine has four states.

- State A implies that there are no transfer requests.
- In state B, the state machine has started to process a transfer request and may possibly wait until the previous VME slave addressed has removed its control signals. State B is also used to allow a delay to set up the address and data before issuing address and data strobes.
- In state C, the address and data strobes have been issued and the state machine waits for the VME slave being addressed to issue a DTACK or a

BERR, or for the GP VME bus timeout counter to count down.

- When the transaction is complete the state machine goes to state D and stays there as long as the VME bus is not given up by the bus requester.

When the GP is powered up, the signal POR* becomes active and causes the state machine to go to state A. The state machine stays in state A as long as the signal BBSY* is not active. BBSY* is activated by the GP bus requester when the PP requests a VME transaction and the VME bus is obtained. When BBSY* becomes active the state machine goes to state B.

In state B, the state machine may possibly wait for the VME slave from the previous transaction to deactivate its control signals P1.DTACK* and P1.BERR*. The two control signals are input to the state machine after they are synchronized by the control signal synchronizer. If the slave control signals are inactive then the state machine goes to state C.

While going from state B to state C, the state machine looks at the signals WORDXFER (word transfer) and VMEA00 to determine what particular type of a transaction has been requested by the PP.

WORDXFER is a bit that is output from the VME control register on page 29 of the schematics (discussed later in this document). If WORDXFER is active, it indicates that the PP wants to do a 16-bit data transfer. When doing 16-bit transfers the state machine activates both MASTERDS1* and MASTERDS0*. If WORDXFER is not active, it means that the transfer requested is an 8-bit (a byte) data transfer.

If the transfer is a byte transfer, then it could be an upper or lower byte transfer. Upper or lower byte transfers are determined by signal VMEA00. VMEA00 is the LSB of the VME master address register on page 30 of the schematics.

- If VMEA00 is a logic low, the data is transferred on VME data bus bits 15 to 8 while asserting MASTERDS1*.
- If VMEA00 is a logic high, the data is transferred on VME data bus bits 7 to 0 while asserting MASTERDS0*.

Regardless of whether it is a byte or a word transfer, MASTERAS* is always asserted when the state machine goes from state B to state C.

In state C, the state machine outputs MASTERAS* and the appropriate data strobe signals and waits for

1. an acknowledge from the VME slave being addressed, or
2. for the timeout counter to count down.

The VME slave acknowledge signals are synchronized and input as SBERR* and SDTACK*. The timeout counter output is the signal TIMEOUT*. When any one of SBERR*, SDTACK*, or TIMEOUT* is activated, the state machine deactivates the address and data strobes, and activates RSTVMEBSY*. The state machine may stay in state C one more period and then go to state D while activating RSTVMEBSY*. It may stay in state C one more cycle to synchronize RSTVMEBSY* to the correct phase of PPCLK2. RSTVMEBSY* has to be active for a rising edge of PPCLK2 because it is used to reset the VMEBUSY* and MASTERWRT* flip-flops, which are clocked by PPCLK2.

The state machine stays in state D as long as there are no requests from the PP and the GP stays as owner of the VME bus (VMEBUSY* not active and BBSY* active). If the bus requester gives up ownership of the bus (BBSY* not active) the state machine goes to state A. If another request from the PP processor comes in while the GP still has control of the VME bus (VMEBUSY* and BBSY* both active), the state machine goes to state B to service the request.

Timeout Counter and Miscellaneous Logic

The timeout counter is used to limit the amount of time the GP will wait for a slave to respond when the GP is doing a transaction as the VME bus master. This prevents the VME bus from being locked up by an incorrect VME access. An incorrect VME access may be a wrong address output by the GP or it may be the slave being addressed has an error. (Of course the possible conditions are not just these two conditions.) The timeout counter limits the VME transaction to less than 5.4 microseconds.

The timeout counter is implemented with a 6-bit down counter programmed in a 16R8A PAL shown in section B3 of page 31. The counter is controlled by the signal MASTERDS* which is an OR of MASTERDS1* or MASTERDS0* (MASTERDS* is active if any one of the two data strobes are active). When MASTERDS* is not active, the counter keeps loading its initial count value. When MASTERDS* becomes active, the counter begins to count down. If MASTERDS* does not become deactivated by the time the counter counts down to zero, the signal TIMEOUT* is activated inside the PAL. TIMEOUT* is latched and kept in this active state as long as MASTERDS* is active or RSTVMEBSY* is active, because the data transfer controller must sense TIMEOUT*, deactivate MASTERDS*, and activate RSTVMEBSY* before TIMEOUT* is deactivated. TIMEOUT* is deactivated when both RSTVMEBSY* and MASTERDS* are inactive. Therefore, every time the Data Transfer Controller starts a transaction, the timeout counter is loaded with its initial value and continues to count down during the transaction.

The miscellaneous logic generates six signals that are used for controlling the GP VME master data transfers. It is implemented with a 16L8A PAL and an F74 flip-flop shown on the right of page 31. The signals generated by the miscellaneous logic are:

- ILLACCESS* (illegal access),
- GPHASVME* (GP has control of the VME bus as bus master),
- SWRDBYTE* (switch read byte),
- SWWRTBYTE* (switch write byte),
- VMEWRTDOE* (VME master write data output enable), and
- WRTHBYTE* (write high byte).

GPHASVME* is used to control the VME address, data, and control transceivers and also to enable the outputs of the master address register. The PAL equation for GPHASVME is:

```
GPHASVME =  BBSY
            + GPHASVME*/RSTVMEBSY*DVMEBSY
```

GPHASVME* is activated whenever BBSY* is active— whenever the GP has control of the VME bus. When BBSY* is deactivated by the bus requester while the GP data transfer controller is still carrying out the VME transfer, GPHASVME* is kept active until RSTVMEBUSY becomes active. RSTVMEBUSY signals the end of the VME transaction and is shown in the second equation above.

DVMEBSY (VMEBUSY* delayed by one 2XCLK period) is used to deactivate GPHASVME*. It comes into use in the particular case where the VME transaction is completed and VMEBUSY* has been deactivated (at the end of the transaction) and, because there have been no further bus requests, BBSY* is still active. In this state, if a bus request arrives at the bus requester at the same time that the PP processor is trying to start another VME transaction (by activating STRTVME*), BBSY* will be deactivated and VMEBUSY* will be activated on the same clock edge. The bus requester state machine will go to state A, where the VME bus is no longer controlled by the GP, and the data transfer controller will go into state A, because BBSY* will not be active. As a result, the bus requester will try to reacquire the VME bus and, until the VME bus is acquired, the transaction requested by the PP cannot take place. Therefore, in the situation where BBSY* is deactivated and VMEBUSY* is simultaneously activated, GPHASVME* must be deactivated.

All the other signals from the miscellaneous control PAL have the GPHASVME* signal in them—which means they only become active when GPHASVME* is active.

- ILLACCESS* becomes active when a word operation is done with VME address bit 0 (VMEA00) equal to a logic high. It is sent back to the PP as a bit in the VME status register.

- VMEWRTDOE* is active whenever a write operation is being done. It is used to enable the lower 8 bits of data from the master data out register to the internal VME data bus bits VMED<07..00>.

- WRTHBYTE* is active whenever a word write operation is being done. It is used to enable the upper 8 bits of data from the master data out register to the internal VME data bus bits VMED<15..08>.

- SWWRTBYTE* is only active when a byte write operation is being done. The PP always writes the byte data to be sent out to the VME bus in the lower 8 bits of the master data out register. However address bit 0 (VMEA00) determines whether the byte of data should be sent on the lower or upper 8 data bus bits. This signal routes the lower 8 bits of the master data out register to the upper 8 bits of the internal VME data bus bits VMED<15..00>. Thus when byte write operations are done, the byte of data is duplicated on both the upper and lower bytes of the VME data bus.

- □ SWRDBYTE* is active when a byte read operation is being done from the high 8 bits of the VME data bus, VMED<15..08>. The PP always expects the byte data read from the VME bus to be in the lower 8 bits of the master data in register. However address bit 0 (VMEA00) determines whether the byte of data will be in the lower or the upper byte of the VME data bus. This signal routes the upper 8 bits of the VME data bus VMED<15..00> to the lower 8 bits of the master data in register.

VME Bus Interrupter

The bus interrupter is used to raise an interrupt over the VME bus when so requested by the PP. After receiving an interrupt request from the PP, the GP interrupt is initiated by activating the open collector interrupt request line P1.IRQ4* (interrupt request level 4).

When the VME bus interrupt handler senses the request, it starts the interrupt acknowledge daisy chain, P1.IACKIN* (interrupt acknowledge in) and P1.IACKOUT* (interrupt acknowledge out). The interrupt handler also starts a data transfer cycle with P1.IACK* (interrupt acknowledge) activated and the first three LSBs of the VME address bus pointing to the interrupt level being serviced. When the GP receives the interrupt acknowledge daisy chain signal P1.IACKIN*, and the interrupt level being serviced is level 4, it outputs on the VME data bus an 8-bit interrupt identification value and asserts P1.DTACK* to complete the interrupt cycle.

The bus interrupter is controlled by a state machine and logic contained in a 16R4A PAL, an F109 and an F74 flip-flops and a delay line, all shown in the top left of page 31.

Interrupts by the GP are always initiated by the PP activating the signal IFLAG, output from an F109 flip-flop. IFLAG is activated by the signal LDIIDL* (load interrupt identification). LDIIDL* is an active low signal which will be asserted whenever the interrupt ID register (page 29) is loaded, and comes from the PP destination logic on page 25. When IFLAG becomes active it clocks an F74 flip-flop with a logic high activating the signal IREQ* (interrupt request). IREQ* is input to the state machine to start an interrupt request cycle. IREQ* is deactivated by the signal HWCLRIFLG* (hardware clear interrupt flag) from the state machine. IFLAG is deactivated with the CLRINTFLG signal under host software control.

The GP can be set up by the host processor to raise an interrupt over the VME bus or it may be set up for polled interrupts. If the GP is not in the polled interrupt mode it will raise an interrupt over the VME bus every time IFLAG is asserted. The GP is put into polled interrupt mode by the signal INTEN (interrupt enable) from the GP control register on page 6. If INTEN is active, it keeps IREQ* from becoming active, thus preventing the state machine from starting an interrupt cycle over the VME bus. IFLAG is an input to the GP status register which can be read by the host processor at its convenience to sense an interrupt from the GP. After sensing the GP interrupt request, the host processor will clear IFLAG by activating CLRINTFLG (clear interrupt flag) from the GP control register. CLRINTFLG will deactivate IFLAG and, in turn, deassertion of IFLAG will deactivate CLRINTFLG.

The bus interrupter state machine controls the interrupts over the VME bus. The state diagram for the bus interrupter state machine is shown in the appendix, labelled "Figure 11."

The VME interrupter state machine has four states.

- When the state machine is in state A, it implies that there is no interrupt request (IREQ*) pending. When IREQ* becomes active, the state machine goes to state B.
- In state B, the signal IRQ (interrupt request) is activated. IRQ is put through a 7438 open collector gate which drives the VME interrupt request line P1.IRQ4*. The state machine stays in state B until the VME interrupt handler acknowledges the interrupt.
- When the interrupt is acknowledged, the state machine goes to state C and waits until SDS0*, the synchronized version of VMEDS0*, is deactivated.
- The state machine goes to state D and then to state A while activating HWCLRIFLG* signal. HWCLRIFLG* signal deactivates the interrupt request signal IREQ*, completing the interrupt cycle.

The state machine stays in state B, activating IRQ and waiting for either

1. the VME interrupt handler to acknowledge the interrupt or
2. the interrupt request signal IREQ* to be deactivated.

If IREQ* is deactivated while in state B, the state machine goes to state D and then to A, ending the interrupt sequence (for the condition when host software deactivates the GP interrupt request over the VME before the interrupt is acknowledged over the VME).

The interrupt is acknowledged when the signal P1.IACKIN* becomes active and the three LSBs of the VME address, VMEA<03..01>, are encoded to be a 4 (interrupt acknowledge on level 4). P1.IACKIN* is synchronized by the control signal synchronizer (on page 6) to be SIACKIN*. P1.IACKIN* is also used to clock an F74 (section C6 of page 31) which samples IREQ*. IREQ* must be sampled in order to continue the daisy chain to the next board by activating P1.IACKOUT* if the GP is not causing the interrupt (if IREQ* is not active).

The output of this flip-flop is the signal SIREQ* (synchronized interrupt request). SIREQ* can be metastable because IREQ* and P1.IACKIN* are asynchronous. Therefore P1.IACKIN* is put through a 50 ns delay (becoming IACKIN*). SIREQ* is only valid into the PAL if IACKIN* is also active. Additionally, the synchronized data and address strobes, SDS0* and SAS*, must be active for the state machine to determine that the interrupt is being acknowledged. Therefore, when all the signals SAS*, SDS0*, IACKIN*, SIREQ*, and SIACKIN* are active and the address lines VMEA<03..01> are equal to decimal 4 (binary 100), the state machine determines that the GP interrupt has been acknowledged and outputs INTDTACK* (interrupt data transfer acknowledge) while going to state C. INTDTACK* is used on page 29 to clear a flip-flop which results in putting the contents of the interrupt ID register on the VME data bus. INTDTACK* is also used (page 7) to activate P1.DTACK*, acknowledging that the interrupt ID is on the data bus to be read.

After going to state C, the state machine waits for SDS0* to be deactivated before going to state D and then to state A (while activating HWCLRIFLG*). HWCLRIFLG* resets IREQ* and IFLAG indicating to the PP processor that the interrupt is complete and another can take place. This wait for SDS0* to become inactive is to ensure that the PP does not write to the interrupt ID register until the previous interrupt has completed use of its contents.

NOTE *HWCLRIFLG* is encoded as being in state D. When the state machine is in state D, the state flip-flops, INFF1 and INFF0, are equal to 00, which is decoded by an F32 gate to produce HWCLRIFLG*.*

The signal IACKOUT* is an output from the PAL which is passed through two F32 gates (page 31) to produce the interrupt daisy chain output signal P1.IACKOUT*. IACKOUT* is activated when

1. the signals IACKIN* and VMEAS* are active and the address bits VMEA<03..01> are not equal to decimal 4 (binary 100), or
2. the signals IACKIN* and VMEAS* are active and the address bits VMEA<03..01> are equal to decimal 4 (binary 100) but SIREQ* is not active.

IACKOUT* is gated with signals DVMEAS* (delayed VMEAS*) and P1.IACKIN* by two F32 gates before becoming the signal P1.IACKOUT*. DVMEAS* signal is used to prevent glitches on P1.IACKOUT* when it is going active (that is, IACKOUT* becomes active before DVMEAS*). P1.IACKIN* deactivates P1.IACKOUT* one F32 gate delay from when it becomes deactivated.

PP VME Interface

The PP VME interface consists of the registers and associated logic which permits the PP to make an access onto the VME bus. These registers include:

- the interrupt id register,
- the data in and data out registers,
- the address register,
- the VME control register, and
- the VME status register.

With this logic, any VME address in 16 or 24-bit address space can be accessed from the PP, however accesses are primarily made to the color frame buffer.

The VME master address register is shown on the left of page 30. It consists of a counter and a register. The counter is implemented with F169-type up/down counters and the register is implemented with ALS374-type devices. The counter is a 24-bit counter corresponding to 24-bit VME addresses, and is preloaded with values from the PPBUS. Since PPBUS is only 16 bits, the 24-bit counter must be loaded in two instructions. The low 16 bits of the counter are loaded when the signal LDVMELADR* (load VME low address) is active; the upper 8 bits are loaded when the signal LDVMEHADR* (load VME high address) is active.

The counter begins counting when the signal CNTVME* (count the VME address) goes active; the signal DNVME* (count down the address) determines whether the counter increments or decrements. When DNVME* is a logic low the counter decrements.

The contents of the counter are loaded into the ALS374 devices on the rising edge of the signal VMEBSY. VMEBSY is just the complemented version of VMEBUSY* which is activated every time the PP requests a VME transaction.

The outputs of the ALS374s are enabled onto the GP internal VME address bus, VMEA<23..01>, whenever the signal GPHASVME*, from the VME miscellaneous logic, is active.

The VME data in register (page 29) is used to hold the data obtained from the VME bus transaction for the PP to read. It is implemented with ALS374-type of devices and is shown in the lower left of page 29.

When a byte access is done, the PP expects data from the VME bus to be in the lower eight bits of the register regardless of which byte, upper or lower, it is transferred on over the VME bus. To accommodate this there are two F241 devices, which function as multiplexers. The F241s multiplex the upper or lower byte of the VMED bus into the lower eight bits of the register.

The signal SWRDBYTE* (switch read byte) determines whether the upper or lower byte is input to the low byte of the register. When SWRDBYTE* is active, the bits VMED<15..08> are input to the low byte of the register, and when SWRDBYTE* is not active VMED<07..00> is input to the low byte of the register. The input to the upper byte of the register is always the bits VMED<15..08>. The data is loaded into the register on the rising edge of VMERDCLK (VME read clock).

VMERDCLK goes to a logic low everytime a read transaction is requested by the PP and one of the data strobes from the master, MASTERDS0* or MASTERDS1*, is active. When the data strobes are deactivated at the end of the transaction, VMERDCLK goes to a logic high and stays at a logic high until the next read transaction is started. Output from the register is enabled onto the PPBUS when the signal RDVMED* (read VME data) from the PP source logic is active.

The VME data out register (page 29) is used to hold the data that is output when a write access is done by the GP as a VME bus master. It is implemented with ALS374 devices and is shown in the top left of page 29. This register is loaded with data from the PPBUS by the signal LDVMED* (load VME data) from the PP destination logic. The contents of the upper byte of the register are output to the bits VMED<15..08> whenever the signal WRTHBYTE* (write high byte) is active. The lower byte of the register is output to the bits VMED<07..00> whenever the signal VMEWRTDOE* (VME write data output enable). When a byte access is done, the PP loads the byte to be output in the lower eight bits of the register. However, whether the byte should be output on the upper or lower byte of the VME data bus depends upon the address in the address register. The F244 devices are used to enable the lower eight bits of the register into the upper eight bits of the VMED bus. This is accomplished whenever the signal SWWRTBYTE* (switch write byte), from the miscellaneous logic, is active.

The interrupt ID register (page 29) is used to hold the interrupt identification value that is output to the VME bus whenever the GP interrupts over the VME bus. It is implemented with one ALS374 device and one F74 flip-flop. The interrupt ID register is loaded with the data from the PPBUS by the signal LDIID* (load interrupt identification) from the PP destination logic. The contents of the register are enabled to the VME data bus by ENINTID* when the F74 flip-flop is cleared by the signal INTDTACK*. The outputs are enabled until the signal DS (data strobe) becomes inactive (which corresponds to both the data strobes on the VME data bus being inactive).

The VME control register (page 29) is used to hold the address modifier and the flag indicating the type of transfer (byte or word) to be done for the GP VME master interface. The VME control register is implemented with an ALS374 and an F74 shown on the lower right of page 29. The register is loaded with data from the PPBUS by the signal LDVMEC* (load VME control) from the PP destination logic. The address modifier bits are loaded into the ALS374 and output to the GP internal address modifier bus, AM<5..0>, when the signal GPHASVME* is active. The signal WORDXFER (word transfer) is held in the F74 instead of the ALS374 because its value must be known to the VME master control logic before the ALS374 outputs are enabled.

The VME status register (page 29) is used to report the status of the VME master interface to the PP. It is implemented with a 16R8A PAL and an F374 shown on the right of page 29. The PAL is used to latch the flags at the appropriate times using 2XCLK1. The outputs from the PAL are then input to the F374 which clocks them in and outputs them to the PPBUS when the PP requests it.

The F374 was used because the status flags are clocked into the PAL using 2XCLK, which has a rising edge a little sooner than PPCLK (see the section on clock and resets). This means that when the status flags are read there is the possibility of them changing at the PPCLK edge as they are clocked into their destinations. To mitigate this, the PAL outputs are clocked into the F374 on the rising edge of PPCLK1*, which means they will be steady on every subsequent rising edge of PPCLK. Because the PAL is clocked with 2XCLK, the outputs must be synchronized to the correct phase of PPCLK. That is why PPCLK1* is input to the PAL. The status flags are only updated or changed when PPCLK is low (PPCLK1* high). Thus all the flags are synchronous with PPCLK.

All the flags held in the PAL are updated each time a GP VME transfer completes (indicated by the RSTVMEBSY* signal from the data transfer controller). Additionally each flag has an accrued version of the transfer. The flags indicate the status of the last VME transaction while their accrued versions indicate the status of not only the last but all the VME transactions since the last status read by the PP.

The PP reads the flags by activating the signal RDVMES* (read VME status) from the PP source logic. RDVMES* also causes the accrued flags to be cleared to their inactive state. In the case where RSTVMEBSY* and RDVMES* are activated at the same time, the accrued part of the flags are cleared, while the current transaction status is simultaneously latched into them. The flags are described below.

- The TO* (timeout) flag indicates that no slave responded to the transaction within the timeout period.
- The ATO* (accrued timeout) indicates the timeout status of not only the last VME transaction but all the VME transactions since the last status read by the PP. TIMEOUT* activates TO* and ATO*.
- The ERR* (bus error) flag indicates that a bus error acknowledge occurred on the last VME transaction.
- The AERR* (accrued bus error) signal indicates the error status of not only the last VME transaction but all the VME transactions since the last status read by the PP.
- SBERR* (synchronous bus error) is delayed by one clock period to become DSBERR* (delayed synchronous bus error), and is used to activate ERR* and AERR*. This delayed version of SBERR* is used to line up or shift it so that it is valid when RSTVMEBSY* signal is valid.
- The AILLACCESS* (accrued illegal access) flag indicates that a VME transaction had an addressing error caused by the GP itself. This error is caused by executing a word operation with the VME address bit 0 (VMEA00) equal to a logic high. The input signal ILLACCES* is delayed by one clock to become DILLACC* and one more clock to become 2DILLACC*. 2DILLACC* is then used to update AILLACCESS*. The delays are necessary to shift ILLACCES* so that it is valid when RSTVMEBSY* is active.
- The VME Status Register additionally reports the state of the IFLAG* signal to the PP processor by clocking it into the F374.

6

Microstore *(pages 9-12)*

6

Microstore *(pages 9-12)*

The microstore consists of two banks, each made up of fourteen 4K-by-4 memory chips. Each bank is thus 4K of 56-bit words for a maximum of 8K words.

Pages 9 and 10 contain the buffers to multiplex and drive the address lines to the two banks of microstore memory chips. Page 9 is the bank 0 address buffers and page 10 is for bank 1.

There are three sources of microstore address:

1. the Viewing Processor,
2. the Painting Processor, and
3. the VME bus.

For the VP, the address lines are as follows:

- VPMSA00 to VPMSA11 — selects word location within a 4K range;
- VPMSA12 and VPMSA13 — not used, would be used if the RAM array is expanded to use 16K-by-4 chips;
- VPENBL0* — enables bank 0;
- VPENBL1* — enables bank 1.

NOTE *The two VPENBL lines are based on address bit 12. (Address bit 14 would be used if the RAM array was expanded to use 16K-by-4 chips.)*

For the PP, the lines are similar except their names are prefixed by a PP instead of a VP.

For VME accesses, the address comes from the microstore VME address counter, page 8. These lines are as follows:

- VMEMSA<00> to VMEMSA<11> — selects word location within a 4K range;
- VMEMSA<12> and VMEMSA<13> — not used, would be used if the RAM array is expanded to use 16K-by-4 chips;
- VMEENBL1* - selects bank 1.

NOTE *VMEMSA<12> selects bank 0. VMEMSA<12> is complemented to become VMEENBL1* and enable bank 1.*

Because VME accesses to the microstore are 16 bits wide, the 56-bit microstore word must be broken up into four 16-bit columns. Two hidden bits—the two least significant bits of the microstore address counter—select one of these four columns when a VME access is made to the microstore.

The column select lines (as generated by the F139 decoder on the bottom of page 8) are MSCOL0* (microstore column 0), MSCOL1*, MSCOL2*,and MSCOL3*. MSCOL3* selects the microstore data bits MSTR15..00, MSCOL2* selects MSTR31..16, MSCOL1* selects MSTR47..32, and MSCOL0* selects the data bits MSTR55..48. Notice that MSCOL0* only selects 8 bits (the VME will only get 8 valid data bits for accesses to this column). See the VME Interface section of this document for further discussion of VME accesses to the microstore.

The F244 and ALS244 buffer/drivers on pages 9 and 10 perform two functions:

1. provide the capability to drive 14 memory chips, and
2. multiplex the three possible address sources onto two buses: MSA000 to MSA013 for bank 0, and MSA100 to MSA113 for bank 1.

The clocks control the multiplexing.

- If VPCLK0 is not halted, then during the second half of a VP cycle (when VPCLK0 is low and PPCLK0 is high), the VP address is routed to the microstore.
- If PPCLK0 is not halted, then during the second half of a PP cycle (when PPCLK0 is low and VPCLK0 is high), the PP address is routed to the microstore.

Thus half the bandwidth is allocated to the VP and half to the PP. In this way, the two processors share the microstore.†

The VME can access the microstore only when both VPCLK0 and PPCLK0 are halted—in which case both signals are high. (No hardware is provided to enforce this restriction, therefore the software must ensure this happens.) When a VME read or write access is made to the microstore, LMSDATA* is asserted, and the VME address is routed to the microstore. ALS drivers can be used because of the less stringent timing requirements needed by VME accesses.

Pages 11 and 12 are microstore bank 0 and bank 1, respectively. For each bank the address bus is routed through 33 ohm series resistors (to dampen ringing) and to the 14 chips of each bank. The data pins on the 2169 memory chips are bidirectional. As an output, the 56-bit microinstruction goes to both the VP and PP instruction registers. These I/O (input/output) pins are also routed through 33 ohm series resistors and to the VME bus transceivers (page 8) for read/write access via the VME bus.

†F drivers are needed on these paths because of the tight timing in allowing for clock skew (3 nsec), routing the address to the microstore (10 nsec including the delay through a resistor), accessing the desired location (40 nsec), and meeting the set-up time on the instruction register (2 nsec) for a total of 55 nsec—which is 5 nsec less than the 60 nsec allotted.

NOTE *The data pins of the two banks of memory are wired in parallel to take advantage of the tri-state capability of the chips.*

Also shown on page 11 are the microstore write control gates, four AND functions. Four write signals are required to write the 56-bit microstore word.

1. MSC0WE* - writes microinstruction bits 48 to 55 (column 0)
2. MSC1WE* - writes microinstruction bits 32 to 47 (column 1)
3. MSC2WE* - writes microinstruction bits 16 to 31 (column 2)
4. MSC3WE* - writes microinstruction bits 0 to 15 (column 3)

These write enables are routed to both banks of microstore; the chip enables determine which bank is actually written. The signals WEGATE* and MSCOL3*..MSCOL0* generate the write enables. WEGATE* signifies that a VME write to the GP board is active. The MSCOL signals indicate that the write is to the microstore and which column within the microstore is to be written (column designations are defined above).

The write enable signals and the chip enables are also routed through 33 ohm series resistors for signal conditioning.

7

Shared Memory *(pages 13-14)*

7

Shared Memory *(pages 13-14)*

The shared memory is a 16K-by-16-bit static RAM array which is used to pass commands, parameters, and data between the host processor and the VP processor. The registers and control for it are on page 13 and the actual RAM array is on page 14.

The RAM array consists of 2167-45 (16K-by-1-bit) static RAM devices. The array is configured as two 8-bit (byte) banks. The two banks are only necessary for VME slave byte accesses, since the VP processor always makes word accesses to shared memory. Address into the RAM is the shared memory address bus, SMA<13..00>; data into the RAM is the shared memory data in bus, SMDI<15..00>; data out of the RAM is the shared memory data out bus, SMDO<15..00>.

The shared memory is accessed twice every VPCLK period, once for the VP and once for the VME slave interface. The VP has access to the memory in the first half of the VPCLK cycle (when VPCLK is a logic high) and the VME has access to the memory during the second half of the VPCLK cycle (when it is a logic low). See the two figures below for a timing diagram of read and write accesses to the shared memory.

Figure 7-1 *Shared Memory Read Timing*

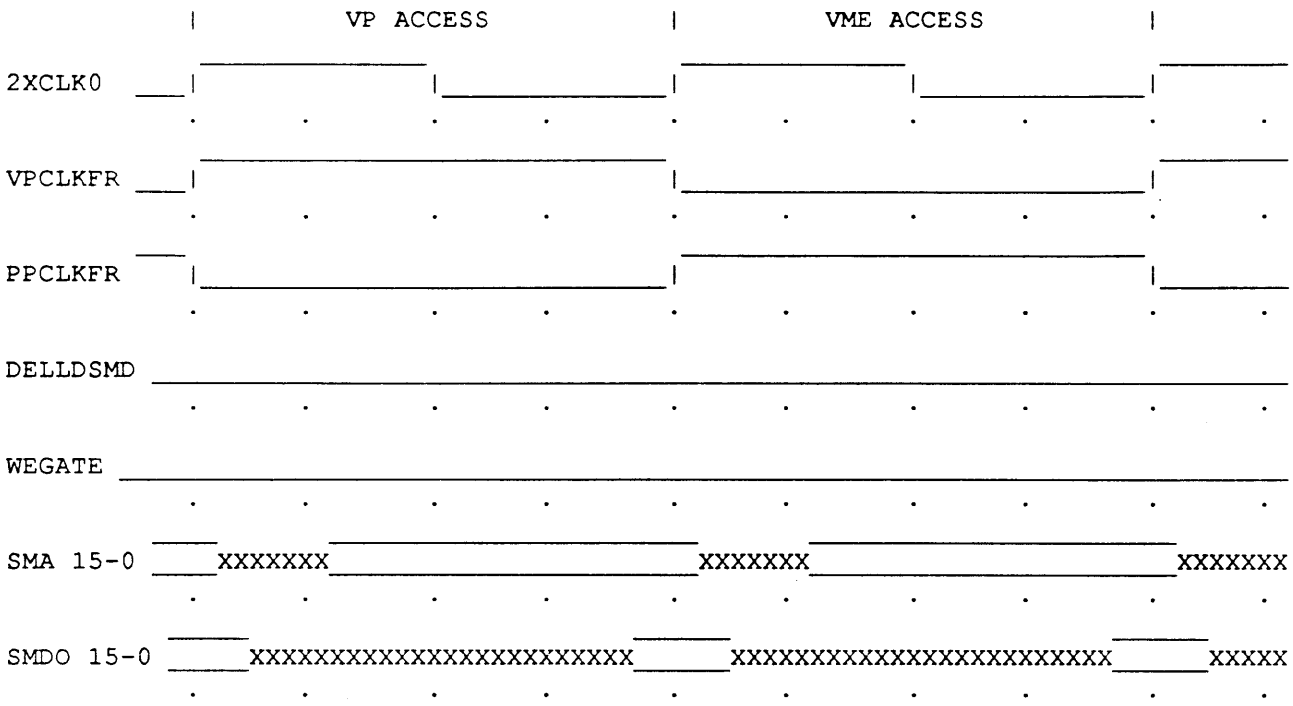
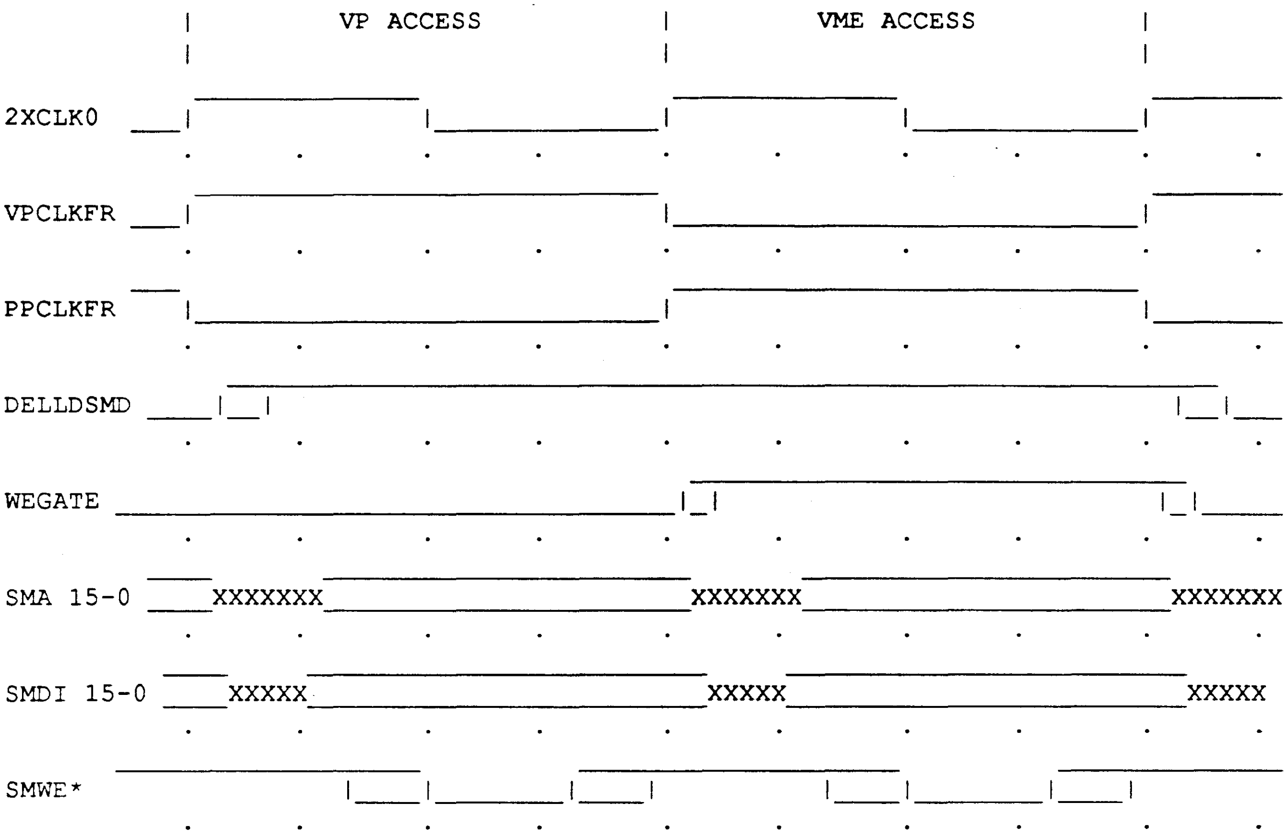


Figure 7-2 *Shared Memory Write Timing*

(Assume that SMACCESS*, DS1*, and DS0* are all active)



- The address and data from the VP and the VME are output to the shared memory address and input data buses, SMADR14..00 and SMDIN15..00, respectively, every half VPCLK period.
- The VP addresses are held and driven by the VP address pointer and the VME addresses are held and driven by the VME address pointer.
- The VP input data is held and driven by the VP data in register and the VME input data is held and driven by the VME data in register.
- When the VP is driving the buses the VME registers are tri-stated and when the VME is driving the busses the VP registers are tri-stated.
- The output enable of the VP and the VME registers are controlled by the signals PPCLKFR (PP clock free running) and VPCLKFR, respectively. PPCLKFR is the complement of VPCLKFR so that when one is a logic high the other is a logic low.

The free running clocks are used here instead of VPCLK and PPCLK because PPCLK and VPCLK may be halted. When PPCLK and VPCLK are halted VME accesses can still occur. There is also the possibility that the VP is halted on an instruction which writes to shared memory; in this case the location in the VP address pointer will be written over and over while the VPCLK is halted. For these reasons the free running clocks are used to keep accessing shared memory regardless of the halt state of the processors.

The address, (from the VP address pointer and the VME address pointer), and t| input data, (from the VP data in register and the VME data in register), are put through series termination resistors before they drive the actual inputs of the RAM array. The address output from the pointers, SMADR15..0, becomes SMA<15..00> after passing through the series resistors.

The data output from the data in registers, SMDIN15..00, becomes SMDI<15..00> after going through the series resistors. SMA<15..00> and SMDI<15..00> drive the RAM address and data inputs directly.†

Loading of data into the VME address pointer and VME data in register, and reading of the data in the VME data out register is covered in the VME Interface section of this document. Refer to that section for information regarding how these registers are loaded and read from the VME. Data from the VME address pointer is enabled to the memories when the signal VPCLKFR is a logic low in the second half of the VPCLK period (page 13). The data from the VME data in register is enabled to the memories when the signal DVPCLK (delayed VPCLK) is a logic low in the second half of the VPCLK period (page 13). DVPCLK is just a delayed version of the VPCLKFR signal—delayed by the F32 gate shown on the top left of page 13. This delay is necessary to enable the data to meet the write cycle data hold time requirements of the memories.

The VP address pointer is implemented as a 16-bit up/down counter with F569 devices (page 13). The counter is loaded with data from the VPBUS. The load

†SMA<15> and SMA<14> are for expansion purposes, and will be used if the RAM array is changed to 64K-by-1 bit devices.

enable is controlled with the signal LDSMP* (load shared memory pointer) from the VP destination logic. The counter can be cleared by the signal CLRSMP* (clear shared memory pointer) from the VP miscellaneous controls. The counter is enabled to count by the signal 2CNTSMP* (second cycle count shared memory pointer). The direction of counting is controlled by the 2DNSMP* (second cycle count down the shared memory pointer). When 2DNSMP* is a logic low, the counter will count down. 2CNTSMP* and 2DNSMP* also come from the VP miscellaneous controls logic.

When a shared memory data write is commanded by the VP, the actual write occurs in the next VPCLK cycle. If the data write command is coincident with the command to count the pointer, the data should be written and the pointer counted after the write is completed (that is, write to the present location in the pointer and then increment the pointer). If the command to count the pointer occurs without the command to write data, then the pointer can be incremented immediately. The logic in the VP miscellaneous control logic implements this to generate the 2CNTSMP* and the 2DNSMP* signals. Data from the counter is enabled to the memories when the signal PPCLKFR is a logic low (first half of the VPCLK period).

The VP data in register is implemented with two F374 devices (page 13). The F374 devices are loaded every VPCLK period with the data from the VPBUS. The data is then loaded into the memories when the write enable is activated (shared memory write control is discussed below). The data from the data in register is enabled to the memories when the signal DVPCLK* (delayed VPCLK) is a logic low (first half of the VPCLK period). DVPCLK* is just a delayed version of the PPCLKFR signal—delayed by the F32 gate shown on the top left of page 13. The delayed version of PPCLKFR signal is necessary to meet the write cycle data hold time requirements of the memories.

The VP data out register (page 13) is used to hold and enable data from the shared memory onto the VPBUS to be read by the VP. It is implemented with two F374 devices. The data output from the memories, SMDO<15..00> (shared memory data out), is loaded into this register every VPCLK period by the rising edge of VPCLK1* (end of the first half cycle of the VPCLK period). The data from the register is output to the VPBUS when the signal RDSM* (read shared memory), from the VP source logic, is active.

The write enable into shared memory is controlled by the two F64 devices shown on the bottom right of page 13. There are two write enable pulses—SMWE0* and SMWE1*—one for each bank of the RAM. This allows the VME to execute byte-sized writes to shared memory; the VP always accesses both banks.

The basic write enable pulse shape is determined by the four delay lines shown next to the F64 devices. There are two 17 ns and two 50 ns delay lines. One pair of 17 and 50 ns delay lines combine to produce the write enable pulse for the VP accesses and the other pair of 17 and 50 ns delay lines produce the write enable pulse for the VME accesses. The two F86 gates on the output of the 17 ns delay lines are used as inverters with one of their inputs tied to a pull up resistor.

The VP write enable pulse shape is formed by the PPCLKFR signal going to the top 17 and the 50 ns delay lines. Generation of the VP write enable is started by the PPCLKFR becoming a logic low. 17 ns later the output of the first delay line becomes a logic low and is inverted to be a logic high going into the F64. The output of the other delay line is a logic high since the 50 ns delay has not completed. If the DELLDSMD (delayed load shared memory data), from the VP miscellaneous controls logic, is also a logic high then the three input AND gate in the F64 will be activated and accordingly the output of the F64 will go to a logic low activating the SMWE* (shared memory write enable) signal. This condition will be maintained until the output of the 50 ns delay goes to a logic low deactivating the AND gate in the F64 and thereby deactivating SMWE*. So the write enable pulse is active from 17 to 50 ns in the second half of the PPCLKFR cycle (the first half of the VPCLKFR cycle) if the DELLDSM signal is active.

The VME write enable pulses are shaped in a similar way but with the signal VPCLKFR as the input to the delay lines. The delay lines activate the input to the F64 between 17 and 50 ns of the second half of the VPCLKFR cycle. If during this time the signals SMACCESS*, WEGATE, and the appropriate data strobe, DS0* or DS1*, signals are all active then the write enable to the RAM is activated. If the VME access is a word access then both data strobes will be active and if it is a byte access then only one of them will be active. The output of the F64 devices are put through a series termination resistors before the RAM to dampen the write enable signal.

8

Viewing Processor *(pages 15-17)*

8

Viewing Processor *(pages 15-17)*

These three pages contain the basic elements of the Viewing Processor:

- the microprocessor,
- the microsequencer,
- the instruction register, and
- the VPBUS source/destination decode.

When reading the following section, it will be necessary to refer to the GP Hardware Reference Manual, especially the section which describes the microinstruction format. Also note, in the schematics, the great similarity of pages 15-17 with pages 25-27.

8.1. Instruction Register

The instruction register (referred to as the ''pipeline register'' in the AMD literature) is a 56-bit register holding the microinstruction currently being executed. Almost everything that happens in this cycle is controlled by the contents of this register. As shown on page 15, the instruction register is implemented with seven F374 8-bit registers.

The instruction register input comes directly from the microstore. The output bits are labeled VPIR55..VPIR00. The register is loaded on each rising edge of VPCLK0. This edge thus defines the end of one cycle and the beginning of the next.

8.2. General Field

Bits VPIR00-VPIR15 are called the general field. With this field it is possible to put assembly-time constants onto the VPBUS and route it to a selected VPBUS destination. The ALS244 buffers (page 15) are used to perform this function. VPIR00-VPIR15 are routed to VPBUS00-VPBUS15 (via the ALS244 buffers) if enabled by the control signal VPRDGF*, a VPBUS source control signal.

8.3. VPBUS Source/Destination Decode

The VPBUS source/destination combination is chosen by bits VPIR44-VPIR49. These bits are decoded by the PALs on page 15 to generate control signals that are routed to each source and destination. When a VPBUS source is enabled, the selected subsection places its data word on the VPBUS. Then the selected VPBUS destination strobes this data word into its subsection at the end of the current cycle with the rising edge of VPCLK. The timing constraint is that for each allowable source/destination combination, the source must place the data word on the VPBUS early enough to meet the set-up time of the destination, and keep the data on the VPBUS long enough to meet the hold time. This constraint is further complicated by the time variability of the rising edge of the destination's load strobe.

The VPBUS sources are decoded by the 16L8 PALs at D2 and C3. These are combinatorial PALs that merely decode the source/destination select bits. The possible sources and the enables are listed below.

Table 8-1 *VP Bus Sources and Their Enables*

Source	Enable
General Field	VPRDGF*
Interprocessor flags (PP-to-VP)	VPRDFLG*
FIFO (PP-to-VP)	VPRDFIFO*
Shared memory	RDSM*
Floating-point status	RDFPS*
VP PROM	RDVPPM*
Floating-point data	
register set A	RDFPDA*
register set B	RDFPDB*
Am29116 microprocessor	VPRD116*

Only the floating-point data is not a simple decoding of the instruction register source/destination select bits. In this case a mode flag, FPFLAG, selects whether register set A or B is to be enabled onto the VPBUS. The microinstruction to fetch either register is the same; the microcoder has pre-defined the FPFLAG to select the set. FPFLAG is stored in the JK* flip-flop at D3 of page 15. FPFLAG=1 (reset state) selects register set A, and FPFLAG=0 selects register set B. Note that bit 14 of the VPBUS is inverted before being input to the JK* flip-flop so that flip-flop works like a standard JK.

The VPBUS destinations are decoded by the PAL at C3, and two F138s (3-to-8 demultiplexers). The PAL translates the instruction register encodings into another encoding used by the F138s to generate the individual destination select lines. There are two types of destination control signals, loads and load enables.

- Loads are ANDed with a clock and used directly to strobe the VPBUS into the selected destination (for example, 374 registers).
- Load enables are not ANDed with a clock; their destinations have a direct clock input plus the load enable (for example, 163 counters).

The VPBUS destinations and their controls are listed below. The figure following shows the timing relationship of the destination controls.

Table 8-2 *VPBUS Destinations and Their Controls*

	Destination	Control
Loads		
	Am29116 microprocessor	VPDLE
	VP branch register	VPLDBR*
	FIFO (VP-to-PP)	VPLDFIFO*
	VP PROM pointer	LDVPPM*
	Interprocessor flags (VP-to-PP)	VPLDFLG*
	VP status register	VPLDSTAT*
Load enables		
	Shared-memory data	LDSMD*
	Shared-memory pointer	LDSMP*
	FIFO (VP-to-PP)	VPLDFIFOL*
	Floating-point pointers	
	source A pointer	LDFPSA*
	source B pointer	LDFPSB*
	destination	LDFPDE*
	Floating-point data	LDFPDR*
	VP n register	VPLDN*
	VP status register	VPLDSTL*

Figure 8-1 *Viewing Processor Destination Load Timing Diagram*

```
Destination load signals

          ____      ____      _
2XCLK1  _|    |____|    |____|

        _           _________
VPCLK1*  |_________|         |_

LOAD    _                     _
ENABLE*  |___________________|

        ________________      _
LOAD*                   |____|
```

NOTE *Skew delays not shown.*

Some destinations have both a load and a load enable control—for example, the VP status register. The load (VPLDSTAT*) is used as the clock signal to the JK* flip-flop on page 15. The load enable (VPLDSTL*) is used as the load enable to the ALS163 register (actually a counter used as a register) on page 23. Thus, when the VP status register is chosen as the VPBUS destination, both the

JK* flip-flop and the ALS163 register are updated.

RST0* is input to the destination-generating PAL (at C3) and is used to inhibit the generation of load signals while it is active.

FIFO1NFL* is also input to this PAL because the load FIFO signals, VPLDFIFO* and VPLDFIFOL* should not be generated if the FIFO is already full.

8.4. Microsequencer

As shown on page 16 of the schematics, the microsequencer is based on the Am2910A microprogram controller. Inputs to this chip include a 4-bit instruction, a 12-bit data word, condition code control, and a clock. Output is a 12-bit address used to fetch the next microinstruction. Further information on the microsequencer may be found in the AMD data book.

The Am2910A instruction normally comes from bits VPIR40-VPIR43. The only exception is when the VP is forced to begin execution at microstore location zero. (The VP can be forced to begin execution at location zero with a write to the GP control register via the VME bus.) The four AND gates on page 16 perform this function. If VPJZ* is inactive, VPIR40-VPIR43 are routed to the Am2910A instruction inputs. If VPJZ* *is* asserted, a binary zero (0000) is routed to the Am2910A and a jump zero instruction (which also resets the Am2910A) is executed.

Two possible sources can be routed to the Am2910A ''D'' inputs:

1. the VP branch register, or

2. the VP general field.

The general field provides assembly time constants while the branch register provides runtime values. If bit VPIR54 is a 0, the general field is enabled. If VPIR54 is a 1, the branch register is enabled. The Am2910A uses this input as a possible next microstore address or for internal use.

The branch register (upper left of page 16) is implemented with two F374 registers. It is loaded when it is chosen as the VPBUS destination by the VPLDBR* signal. The general field comes directly from the VP instruction register and the F244 buffers route VPIR00-VPIR11 to the Am2910A when enabled by VPIR54.

The condition code logic selects a condition to test and determine whether or not a jump condition passes or fails. Eight condition code flags are implemented.

Table 8-3 *Condition Code Logic*

Condition	Flag
Logic 1 (used for unconditional branches)	PU2
Floating-point result negative	FPNEG
FIFO (PP-to-VP) not empty	FIFO2NMT
FIFO (VP-to-PP) not full	FIFO1NFL
Am29116 result overflow	VPOVR
Am29116 result carry	VPCAR
Am29116 result negative	VPNEG
Am29116 result zero	VPZERO

The four Am29116 status flags are stored in the F163 register (the count capability is not used) at A5 of page 16. If so enabled by VPIR53, the Am29116 status for the current cycle is stored in this status register for use in a subsequent cycle.

One of the eight conditions is selected by the F151 multiplexer (B4 of page 16) controlled by bits VPIR36-VPIR38. Either polarity of the test condition is then generated by the XOR gate (VPIR39 selects the polarity) and routed to the CC* input of the Am2910A.

NOTE *CCEN* is not used on the Am2910A.*

As explained earlier in the description of the microstore, the basic cycle is split into two halves.

1. The first half is used to compute the address of the next microinstruction.
2. The second half is used to access the memory.

Thus the microsequencer has one half-cycle (60 nsec) to compute the next address. The worst case path is the CC path: 3 nsec for clock skew, 10 nsec for valid VPIR bits, 9 nsec S to Y* through the F151, 8 nsec through the XOR gate, and 30 nsec CC* to Y in the Am2910A for a total of 60 nsec.[†]

The Am2910A provides 12-bit addresses (VPMSA00-VPMSA11) and thus can access 4K words of microinstructions. This was judged not to be enough, so a bank switch capability augments this address space. The PAL at D2 is used to implement this feature.

Inputs to the PAL include a clock, the 4-bit Am2910A instruction, and three high-order "D" bits to provide the larger address from either the branch register or the general field. If the Am2910A instruction is a jump zero, the PAL outputs bank 0. If the Am2910A instruction is a jump map, the PAL outputs the bank selected by the high-order "D" bits. For these two cases, the new bank is updated in the PAL and is used as the selected bank until the next jump zero or jump map instruction. Otherwise, there is no change to the bank selected by the PAL.

[†] Whew! That's tight!

8.5. Microprocessor

On page 17 is the microprocessor, the Am29116. The Am29116 inputs include a clock, a 16-bit instruction, a 16-bit (bidirectional) data bus, and three control signals (OEY*, DLE, and IEN*). Output are a 16-bit (bidirectional) data bus, and four status flags (zero, carry, negative, and overflow). Refer to the AMD literature for further details.

The Am29116 instruction comes from bits VPIR16-VPIR31. However, two fields can be modified at runtime—the n field and the RAM field.

- The n field selects a bit position in a bit-oriented instruction or defines a shift count in a shift instruction.
- The RAM field selects one of 32 on-board registers.

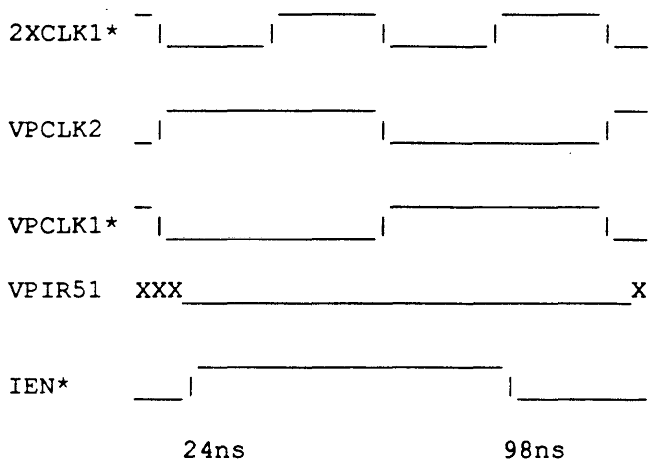
The n field value can be chosen at assembly time or at run time. If VPIR52 is 0 then VPIR25-VPIR28 (assembly time value) is routed to the Am29116 instruction bits 9 to 12. If VPIR52 is a 1, then the n register value (run time value) is routed to the Am29116 instruction. The F241 at D7 performs this multiplexing.

The RAM address field is modified to provide double address accesses—RAM source and RAM destination can be different.

- If not doing a double address access, VPIR51 is high and bits VPIR16-VPIR19 are routed to the Am29116 instruction bits 0-3 through the entire cycle.
- If VPIR51 is low, then VPIR16-VPIR19 are routed to the Am29116 for the first half of the cycle and VPIR32-VPIR35 are routed to the Am29116 for the second half. The F241 at B7 is used for this multiplexing. Because only four of the five RAM address bits are modified, the RAM source and destination must be in the same group of 16, 0 to 15 or 16 to 31.

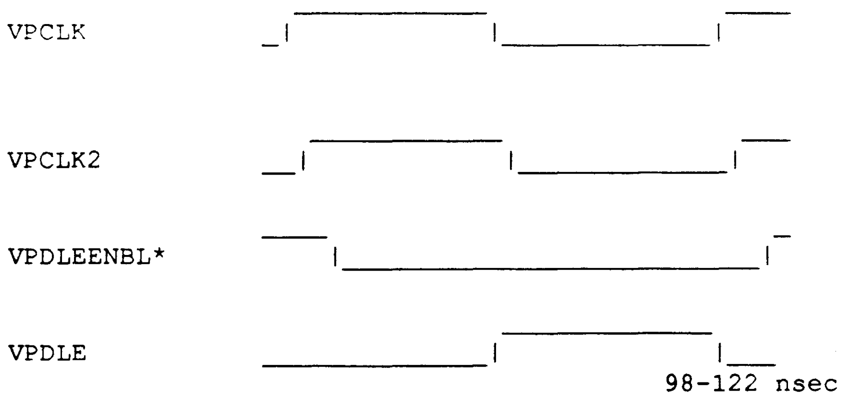
To provide double address accesses, the IEN* line must be managed. If not doing a double address access, VPIR51 is high and the F74 at A7 is held clear so that IEN* is low throughout the cycle. If a double address access is active, then IEN* is as shown in the following two figures. The F32 at A8 sets the IEN* D flip-flop and the proper 2XCLK1* edge resets the signal.

Figure 8-2 *Am29116 Double Address Timing Diagram*



24 nsec easily meets the rising edge spec. 98 nsec exactly meets the worst case falling edge spec.

Figure 8-3 *Am29116 DLE Timing Diagram*



The rising edge easily meets specifications. The falling edge can occur anywhere in the above interval so that data must be valid at 88 (to meet set-up time) and stay until 128 (to meet hold time).

NOTE *All times are referenced to VPCLK2.*

The other two Am29116 control lines involve VPBUS activities. If the Am29116 is chosen as the VPBUS source, VPRD116* is asserted enabling the Am29116 internal Y bus onto the VPBUS. If the Am29116 is to be the destination of a VPBUS transaction, VPDLE is asserted and the VPBUS data is latched in the Am29116 internal D latch. The preceding figure shows the timing of VPDLE. Note that VPDLE occurs early enough so that the Am29116 internal D latch can be considered transparent, and thus the VPBUS data can be used in the current Am29116 cycle except when the Am29116 Y bus or the interprocessor flag register (PP-to-VP) is the VPBUS source.

Also on page 17 are the VPBUS pull-up resistors. They improve the rise times of the VPBUS signals.

8.6. VP Miscellaneous Controls

The miscellaneous controls (page 17) are a collection of unrelated commands. They are controlled by VPIR32-VPIR35 and are enabled only if VPIR51 is high (no double address access). These instruction bits are decoded by the PALS at C1 and D1 to generate various command signals. A complication arises because some of the commands are specified in the current cycle but must executed in a subsequent cycle. The various commands and the control lines are as follows.

Table 8-4 *Miscellaneous Controls*

Command	Control Line
Clear shared-memory pointer	CLRSMP*
Count shared-memory pointer	2CNTSMP*
direction control (up or down)	2DNSMP*
Count up floating-point pointers	
source A	UPFPSA*
source B	UPFPSB*
destination	2UPFPDE*

The ALS374 register holds the bits that indicate that a command in this cycle affects a future cycle. These include the following:

- Load shared-memory — shared-memory write to be done in next cycle;

  LDSMD is the current cycle signal;

  DELLDSMD is the next cycle signal.

- Count shared-memory pointer — count to be done in next cycle if coincident with a load shared-memory;

  DNSMP* and CNTSMP* are the current cycle signals;

  2DNSMP* and 2CNTSMP* are the current or next cycle signals.

- Load floating-point data — floating-point data write to be done in next cycle;

  LDFPDR* is the current cycle signal;

  DELLDFP is the next cycle signal.

- Count up floating-point destination pointer — count up to be done in next cycle if coincident with a load floating-point data;

  UPFPDE* is the current cycle signal;

2UPFPDE* is the current or next cycle signal.

- Floating-point high/low bit — least significant floating-point register address bit must be remembered for one cycle if coincident with a load floating-point data;

  VPIR50 is the current cycle signal;

  DVPIR50 is the next cycle signal.

- Unload floating-point result — an unload most significant half of Weitek result command, data will be valid in two cycles;

  UNLOAD* is the current cycle signal;

  UNLD2DEL* is the 2-cycles hence signal.

The commands CLRSMP*, UPFPSA*, and UPFPSB* are executed in the same cycle they appear. Thus the PALs decode the proper VPIR32-VPIR35 and VPIR51 combinations to generate these control signals. The commands 2CNTSMP*, 2DNSMP*, and 2UPFPDE* may be executed in the current cycle or the next cycle. The corresponding signals CNTSMP*, DNSMP*, and UPFPDE* are decoded in the current cycle and will generate the 2CNTSMP*, 2DNSMP*, and 2UPFPDE* signals, respectively, in the current cycle if there is no concurrent write or in the next cycle if there is a concurrent write.

The PAL outputs LDSMD and DELLDFP are merely inverted from the corresponding inputs. Using these otherwise unused PAL paths saves inverter gates.

The timing constraints on these signals are to meet the set-up times of the count and direction enables. These constraints are easily met.

9

Floating Point *(pages 18-22)*

9

Floating Point *(pages 18-22)*

9.1. Overview

Pages 18-22 contain the floating-point subsection. Pages 18 and 19 show register set A with pointers, and pages 20 and 21 show register set B. (Note the similarity between these two sets of pages.) Page 22 contains the Weitek floating-point chips, the 1032 and 1033.

The floating-point subsection operates in parallel with the Am29116. The general field (VPIR00-VPIR09), VPIR50, and VPIR55 are used to specify the floating point operation. These operations are listed below.

```
1. Start a floating point operation
        a. fp_register(source A) OP fp_register(source B)
        b. fp_result OP fp_register(source B)
        c. fp_register(source A) OP Weitek internal B register
        d. fp_result OP Weitek internal B register
        e. OP fp_register(source A)
        f. OP fp_result
2. fp_result --> fp_register(destination)
```

The floating-point subsection can be involved in a VPBUS transaction. The pertinent operations are listed below.

```
3. VPBUS --> source A pointer
4. VPBUS --> source B pointer
5. VPBUS --> destination pointer
6. fp_register --> VPBUS
        a. fp_register(source A) --> VPBUS
        b. fp_register(source B) --> VPBUS
7. VPBUS --> fp_register(destination)
8. Floating-point status register --> VPBUS
```

The fp_register(pointer) is the contents of the floating-point register addressed by the indicated pointer and fp_result is the Weitek chip result.†

It is possible to do operations 1, 2, and one of operations 3 to 7 in the same cycle. However, 2 and 7 must *not* be executed together (two sets of drivers would be driving the inputs to the floating-point registers simultaneously). Also, the combination of 1 and 6 is unlikely since both use the same pointer.

†For further information, please see the Weitek literature for OP selection.

Two sets of floating-point registers are provided, each implemented with four 4K-by-4 chips. The sets are duplicates and the same data are written into both; the only time they will be different is on power-up, when their contents will be indeterminate.

Four 4K-by-4 chips provide 4K 16-bit words and all accesses are 16-bits wide. However floating-point numbers are 32 bits wide, so access to a floating-point number requires two accesses to the floating-point registers. The three pointers—source A, source B, and destination—point to a floating-point number. Thus another bit is needed to reference each 16-bit half of the floating-point number. This bit is VPIR50, the high/low bit. The high/low bit must be specified for all of the above listed operations.

All floating-point operations use the floating-point registers. The Weitek chip operands come from these registers and Weitek chip results are (eventually) routed back to the registers. And the other VP components (the Am29116 microprocessor, the shared-memory, the FIFO, etc.) access these registers, since the Weitek chips are not directly accessible via the VPBUS.

During each cycle, three accesses are made to the floating-point registers—two reads and an optional write.

During the first half of a cycle, the source A pointer is enabled to floating-point registers set A and the source B pointer is enabled to set B, and the two reads are executed. The fetched values are latched and are available for routing onto the VPBUS or to the Weitek chips during the second half of the cycle. If VPBUS source is the floating-point data register, the FPFLAG (see page 15) determines whether set A or set B will be used as the data source. The Weitek chips typically use two operands, hence the need for floating-point register set A and set B.

During the second half of a cycle, allocation is provided for a write to the floating-point registers. The destination pointer is routed to both set A and set B and the same write enable is used for both sets. Thus the sets are duplicates. Source of the data to be written is either the Weitek result or the VPBUS. If the Weitek chips are the source, the write is done in the current cycle. If the VPBUS is the source, the data word is first latched and then written into the register sets on the next cycle. This is required because the VPBUS-sourced data arrive too late to meet the set-up time of the 4K-by-4 memory chips.

9.2. Detailed Description of the Floating Point Circuitry

Page 18 contains the source A and destination pointers. (There are two destination pointers, one for register set A and one for set B. But the same data, load, and count control signals are used on both pointers so that they are duplicates. Replicates of the circuits are necessary because the set A and set B address buses cannot be tied together.)

Floating Point Addressing: Set A

F569s—4-bit counters with tri-state outputs—are used to implement the pointers. VPBUS00-VPBUS11 are the data inputs, providing 2K address space; (FPAA11 is not used; VPIR50 is the 12th address bit to the 4K memory chips). LDFPSA* is used to load the source A pointer and UPFPSA* is used to count the pointer. LDFPDE* is used to load the destination pointer and 2UPFPDE* is used to count the pointer. Both counters are hardwired to count only up. During the first half of each cycle, VPCLK3* is used to enable the source A pointer to FPAA—the address bus to floating-point register set A. During the second half of each cycle, VPCLK3 is used to route the destination pointer to FPAA.

Floating Point Registers: Set A

Floating-point data register set A is shown on page 19. Four 2169 memory chips, each 4K-by-4, are used to build the 4K-by-16 memory. Eleven of the twelve address lines come from the source A and destination pointers and the least significant address bit, FPALSB, is generated by the F64 at D1. (This signal will be described more fully later.)

The F374s on this page provide temporary data storage to meet timing constraints. The F374s at C7 and C4 are the VPBUS data-in registers. On every cycle, the contents of the VPBUS are strobed into these registers for (possible) subsequent write into the floating-point registers. VPCLK3 delayed by a F32 gate becomes FPDCLK to load these registers. The delay is necessary to ensure that the contents of the these F374s do not change too soon and violate the hold time to the 2169 chips. (It also means that each VPBUS source must hold the data on the VPBUS a little longer.) FPLDENBL* enables these F374s to the 2169s during a write (see page 22 of the schematics).

The F374s at A7 and A4 are used to hold the Weitek result. They are loaded on every VPCLK3* even if there is no valid result, but the F374 output is routed to the 2169s (enabled by FPSTENBL* from page 22) only if a write is active.

The F374s at C5 and C2 and the F374s at A5 and A2 are loaded with the data read from floating-point register set A in the first half of each cycle; VPCLK3* is the load strobe. The floating-point data out register A (C5 and C2) is connected to the VPBUS for possible use as a VPBUS source (selected by the signal RDFPDA* from page 15). The A operand register (A5 and A2) is a possible source of the Weitek chip A operand (the current Weitek result is the other possible source) and is selected by the NOCHAIN* signal from page 22.

The F64 at D1 is used to generate the least significant address bit, FPALSB, to both set A and set B of the floating-point registers. During the first half of each cycle, VPCLK4 is logic 1 so that VPIR50 is selected as the FPALSB. During the second half of each cycle, VPCLK3* is logic 1 so that either VPIR50 or DVPIR50 (VPIR50 from the previous cycle) is selected as FPALSB. DVPIR50 is selected if the floating-point registers were selected as the VPBUS destination in the previous cycle (DELLDFP is logic 1), and VPIR50 is selected otherwise. This complication is necessary because when the floating-point registers are chosen as the VPBUS destination, the actual write is done in the next cycle so that VPIR50 (the high/low select bit) must be remembered and used in the next cycle.

CAUTION **The microcoder must realize this occurs and not attempt to write the Weitek result into the floating-point registers in the cycle immediately after the floating-point registers are chosen as the VPBUS destination.**

Floating Point Addressing: Set B

Page 20 contains the source B and destination pointers. They are similar to those of set A, on page 18.

F569s—4-bit counters with tri-state outputs—are used to implement the pointers. VPBUS00-VPBUS11 are the data inputs, providing 2K address space; (FPBA11 is not used; VPIR50 is the 12th address bit to the 4K memory chips). LDFPSB* is used to load the source B pointer and UPFPSB* is used to count the pointer. LDFPDE* is used to load the destination pointer and 2UPFPDE* is used to count the pointer. Both counters are hardwired to count only up. During the first half of each cycle, VPCLK4* is used to enable the source B pointer to FPBA, the address bus to floating-point register set B. During the second half, VPCLK3 is used to route the destination pointer to FPBA.

Floating Point Registers: Set B

Floating-point data register set B is shown on page 21 (note the similarity between pages 19 and 21). Four 2169 memory chips, each 4K-by-4, are used to build the 4K-by-16 memory. Eleven of the twelve address lines come from the source B and destination pointers and the least significant address bit, FPALSB, is generated on page 19.

The F374s on this page provide temporary data storage to meet timing constraints. The F374s at C7 and C4 are the VPBUS data-in registers. On every cycle, the contents of the VPBUS are strobed into these registers for possible subsequent write into the floating-point registers. FPDCLK (page 19) is used to load these registers. The delay is necessary to ensure that the contents of these F374s do not change too soon and violate the hold time to the 2169 chips. (It also means that each VPBUS source must hold the data on the VPBUS a little longer.) FPLDENBL* enables these F374s to the 2169s during a write (see page 22).

The F374s at A7 and A4 are used to hold the Weitek result. They are loaded on every VPCLK4* even if there is no valid result, but the F374 output is routed to the 2169s (enabled by FPSTENBL* from page 22) only if a write is active.

The F374s at C5 and C2 and the F374s at A5 and A2 are loaded with the data read from floating-point register set B in the first half of each cycle; VPCLK4* is the load strobe. The floating-point data out register B (C5 and C2) is connected to the VPBUS for possible use as a VPBUS source (selected by the signal RDFPDB* from page 15). The B operand register (A5 and A2) is a possible source of the Weitek chip B operand (the internal Weitek B register is the other source) but is always enabled.

Floating Point Chips, Status and Result Registers

Page 22 contains the Weitek floating-point chips, the 1032 Multiplier and 1033 ALU. (See the manufacturer's literature for details.) The two chips are basically wired in parallel and thus can be viewed as a single entity. The exceptions are U1MULT* and U1ALU* which select which chip will enable its output, and F3 and F2 on the multiplier which must be 0 (except on a load mode operation) or it will enter a diagnostic state.

The buses connected to the Weitek chips are as follows:

- □ WTKA00 to WTKA15 — A operand bus from Weitek result or A operand register;

- WTKB00 to WTKB15 — B operand bus from B operand register;
- WTKC00 to WTKC15 — C (result) bus to the 3 Weitek result registers.

The F374s at D2 and C2 store the Weitek result for possible routing back to the Weitek chips as the A operand. This Weitek A operand source is enabled by a chain operation (CHAIN* asserted), which is generated by the PAL at A8.

The 16R6A PALs at B2 and A2 are used to decode the Weitek status bits (S2..S0), store this status, and make the bits available when the floating-point status is chosen as VPBUS source. The B2 PAL is for the non-accrued status and the A2 PAL is for accrued status.

Floating-point status is updated each time the most significant half (so that the sign bit is valid) of the Weitek result is read. This occurs two cycles (because of the Weitek pipelining) after an ''unload the most significant word of the result'' command. Such a command is decoded from VPIR01, VPIR02, and VPIR55 by the PAL at A8 (signal UNLOAD*). The unload is remembered for two cycles on page 17, is returned as the signal UNLD2DEL*, and is input to the floating-point status PALs to enable the update. The signal RDFPS* enables the floating-point status onto the VPBUS and also clears the accrued status (hence RDFPS* is both an input and an enable on the A2 PAL). Since the PAL outputs are active low, the complement of the floating-point status is placed onto the VPBUS (and is easily inverted in the Am29116).

The F163 at A3 is used to hold the sign bit of the last Weitek result. It is routed to the condition code logic thus allowing the microcoder to test on the sign of the floating-point result. This status bit is updated at the same time as the floating-point status PALs; that is, when enabled by the signal UNLD2DEL*.

The PAL at A8 and the F64 at A6 are the control logic for the floating-point subsection. The PAL generates the following signals from the indicated VP instruction register bits.

- WTKL1 — Weitek L1 load command
- WTKL0 — Weitek L0 load command
- MULTF3 — Weitek multiplier F3 function code
- MULTF2 — Weitek multiplier F2 function code
- UNLOAD* — High-order Weitek result will be available in 2 cycles
- CHAIN* — The Weitek A operand is to come from the current Weitek result
- NOCHAIN* — The Weitek A operand is to come from floating-point register set A
- FPSTORE — The Weitek result should be written into floating-point register sets A and B

The F64 and associated F32s generate the write enable to the 2169s (FPREGWR*) and the data enables to the F374s (FPLDENBL* and FPSTENBL*) which select and enable the data to be written to the 2169s.

VPCLK* goes to a logic 1 early in the second half of each cycle. FPDCLK is inverted by the F04 at A7 and goes to a logic 0 just before the end of a cycle. Therefore ANDing these two signals generates a pulse during the second half of a cycle; the F64 performs this AND. Actually this 2-way AND is two 3-way ANDs. FPSTORE, VPCLK*, and FPDCLK inverted are ANDed signifying that the Weitek result is to be written into the floating-point registers, and DELLDFP, VPCLK*, and FPDCLK inverted are ANDed signifying that VPBUS data should be written. In the F64 these two 3-way ANDs are ORed to generate FPREGWR*. The series resistor minimizes ringing on this critical signal. FPREGWR* is then used to generate the data enables selected by which operation is active; DELLDFP* selects FPLDENBL* and FPSTORE inverted selects FPSTENBL*.

10

VP Miscellaneous Logic *(page 23)*

10

VP Miscellaneous Logic *(page 23)*

The Viewing Processor miscellaneous logic includes:

□ the VP PROM,

□ the VP n register,

□ the interprocessor flags #1 (VP-to-PP),

□ and VP status register.

10.1. VP PROM

The Viewing Processor PROM (an identical configuration exists as the PP PROM on the Graphics Buffer board) is shown on page 23. The VP PROM pointer is loaded when chosen as the VPBUS destination. LDVPPM* strobes the VPBUS into the F374s which hold the PROM pointer. The output of this register is routed to the PROM, and after an access time delay,† the PROM data word is valid and routed to the ALS244 buffers. (ALS244 buffers are needed because of the long chip enable time of MOS PROMS.) This data word is routed onto the VPBUS when enabled by RDVPPM*.

On the GP board, two 28-pin sockets are provided for the PROMs. 27128 PROMs are being used to provide 16K words.

10.2. VP n Register

The n register is implemented with a F163 counter (page 23) but the count functions are not used. VPBUS00-VPBUS03 bits are loaded into the counter on the rising edge of VPCLK2 (end of a cycle) if selected as the VPBUS destination by VPLDN*. The n register is multiplexed into the Am29116 instruction stream on page 17.

10.3. Interprocessor Flags #1

This register passes eight flags from the VP to the PP along with three other flags. An ALS374 (page 23) is used to hold the eight flags that are written by the VP. The register is loaded from the VPBUS00-VPBUS07 bus by VPLDFLG*.

There are three other flags which can be read by the PP through an ALS244 (C2 on page 23) device.

□ PPTOVP comes from the FIFO direction flip-flop (page 24) and it indicates the direction of FIFO data flow. When PPTOVP is a logic high the FIFO is

The actual time depends on type of PROM used; the microcoder must account for this time by delaying the proper number of cycles before choosing the VP PROM as the VPBUS source.

passing data from the PP to the VP.

- SFIFOMT* signal comes from the FIFO (page 24) and, when active, indicates that all the words from the actual FIFO devices have been read. Note however that this does not necessarily mean that the processor receiving the FIFO data has read all the data. It is possible for the last FIFO data to be read out and be available on the output of the FIFO devices waiting for the processor to read it. For that situation the SFIFOMT* signal will be active while the data is waiting to be read by the processor.
- The last flag on the ALS244 is always loaded as 1—PU0. This allows the PP microcode to read this register and, seeing the logic 1 in the ninth (PPBUS08) bit position, determine that this is the PP.

These 11 flags are enabled onto the PPBUS with PPRDFLG*.

10.4. VP Status Register

The VP status register is implemented with an ALS163 counter but the count functions are not used. VPBUS00-VPBUS03 are loaded into the counter on the rising edge of VPCLK2 (end of a cycle) if selected as the VPBUS destination (VPLDSTL*). The 4 outputs are routed to the GP status register (page 6), accessible via the VME bus, and to 4 LEDs. Note that the LEDs are configured so that if a counter bit is 0 (for example, after a reset), the corresponding LED is ON.

11

FIFO *(page 24)*

11

FIFO *(page 24)*

The FIFO is a 512-by-16-bit buffer used to pass information bidirectionally between the PP and the VP. The FIFO and its associated circuitry is shown on page 24 of the schematic.

The actual FIFO is implemented using 4501—512-by-9 FIFO—devices. The two FIFO devices are hooked up for width expansion to provide a 512-by-18-bit FIFO. Notice that two of the 18 bits are not used because both the VP and the PP have 16-bit data buses. The 4501 devices are cleared through their RS/ pins by the signal RST1* (reset 1) from the reset circuit on page 5. This will deactivate the two outputs from the devices: EF/ (empty flag) and FF/ (full flag).

The direction of the FIFO is controlled by a signal called PPTOVP (FIFO is passing data from the PP to the VP) and its complement VPTOPP (FIFO is passing data from the VP to the PP.

There are two write registers: one for the VP, and one for the PP. The processors write the data destined for the FIFO into these write registers. The data is then written from the registers to the FIFO devices under control of the FIFO write control. The write registers are implemented with ALS374 devices. The output enables of the two registers are controlled by the signals PPTOVP (and its complement VPTOPP) so that only the output of one of them at a time is enabled onto the FIFO data in bus, FIFOD15..00. When VPTOPP is active (PPTOVP is inactive) the VP write register drives the FIFOD bus. When PPTOVP is active (VPTOPP is inactive) the PP write register drives the FIFOD bus. The VP write register is loaded with the data on the VP bus when the signal VPLDFIFO* from the VP destination logic is activated. The PP write register is loaded with the data on the PP bus when the signal PPLDFIFO* from the PP destination logic is activated.

The direction flag, PPTOVP, is held in an F109 flip-flop which is controlled by the VP. The flip-flop is set or reset when the destination of the VP bus cycle is the VP flag register, indicated by the signal VPLDFLG* being active. When VPLDFLG* is active, PPTOVP (and VPTOPP) is set, reset, or left unchanged—as determined by the VPBUS bits 15 and 14.

There are two FIFO read buffers, one for the VP and one for the PP, that are used to route the data output from the FIFO to the respective processor's data bus. The data output from the FIFO devices is named FIFOQ15..00 and is input to both the read buffers. The read buffers are implemented with ALS244 devices. The FIFO data output is enabled from the input of the VP read buffer to the

VPBUS when the signal VPRDFIFO* from the VP source logic is activated. The FIFO data output is enabled from the input of the PP read buffer to the PPBUS when the signal PPRDFIFO* from the PP source logic is activated.

The FIFO has two output flags which are used by the FIFO control logic.

- EF/ (empty flag), when active, indicates that there is no data in the FIFO.
- FF/ (full flag), when active, indicates that the FIFO is full.

When data is written into the FIFO, the FF/ flag may become active in the amount of time noted in the specification of the 4501 device. Similarly, when data is read from the FIFO, the EF/ flag may become active in the amount of time noted in the specification of the 4501 device. These two flags are seen as asynchronous signals to the FIFO read and write control logic, which works synchronously with 2XCLK. Both of these flags are, therefore, synchronized to 2XCLK, using an F74 shown in section A4 of page 24, before being input to the FIFO control logic. The synchronized version of FF/ and EF/ are SFIFOFULL (synchronized FIFO full) and SFIFOMT (synchronized FIFO empty), respectively.

FIFO write control senses when one of the processors has loaded data into the write register, generates the write pulse (WR*) to the FIFO, and controls the signals that are sent back to the processor to indicate the status of the FIFO. The FIFO write control is implemented with a state machine and logic contained in a 16R4A PAL and an F74 shown in the lower left of page 24. The write control looks at the signal VPTOPP to determine which of the processors can write to the FIFO. When the processor writes data to the FIFO write register, the synchronized full flag, SFIFOFULL, is inspected to determine whether the data can be written to the FIFO devices. If the FIFO is not full then the data is written immediately; if it *is* full the control logic waits until it is not full and then writes data to the FIFO devices. When the processor writes data to the FIFO write register, the FIFO write control deactivates the FIFO not full flag until the data is written to the FIFO devices.

There are two flags, one for the VP and one for the PP, that are used by the processors as status flags when doing writes to the FIFO. The signal FIFO1NFL (FIFO 1 not full) is used by the VP processsor when doing writes to the FIFO. The signal FIFO2NFL (FIFO 2 not full) is used by the PP processor when doing writes to the FIFO. Only one of the processors—VP or PP—can write to the FIFO at any one time.

When one of the processors is writing to the FIFO its appropriate signal (FIFO1NFL or FIFO2NFL) is valid; the signal for the other processor is kept in a constant state in which it signals that the FIFO is not full, as in the following:

- when going from VP to PP the signal PPTOVP keeps FIFO2NFL in its active state;
- when going from PP to VP the signal VPTOPP keeps FIFO1NFL in its active state.

By keeping FIFONFL flag in its active state when the FIFO is turned the other way, the software is kept from hanging if an erroneous write to the FIFO is executed. FIFO1NFL and FIFO2NFL, when they are valid, are controlled by the signal FULL from the write control state machine in the PAL on page 24.

The state machine has four states. The state machine is in state A when the write data register is empty and, therefore, there is no data to write into the FIFO. When data is written to the write data register, the state machine goes to state B and may possibly wait there if the FIFO is full and the data can not be written to the FIFO devices. If the FIFO is not full (or becomes not full) the state machine goes to state C and then D activating the write enable signal to the FIFO, WR*. The state machine goes from state D back to state A which completes the write cycle.

The state diagram for the FIFO write state machine is shown in Figure 15 in the appendix.

In the text below are three timing diagrams for writes to the FIFO.

Figure 11-1 *FIFO Write Timing Diagram— VP to PP*

FIFO passing data from the VP to PP, consecutive writes with SFIFOFULL never becoming active.

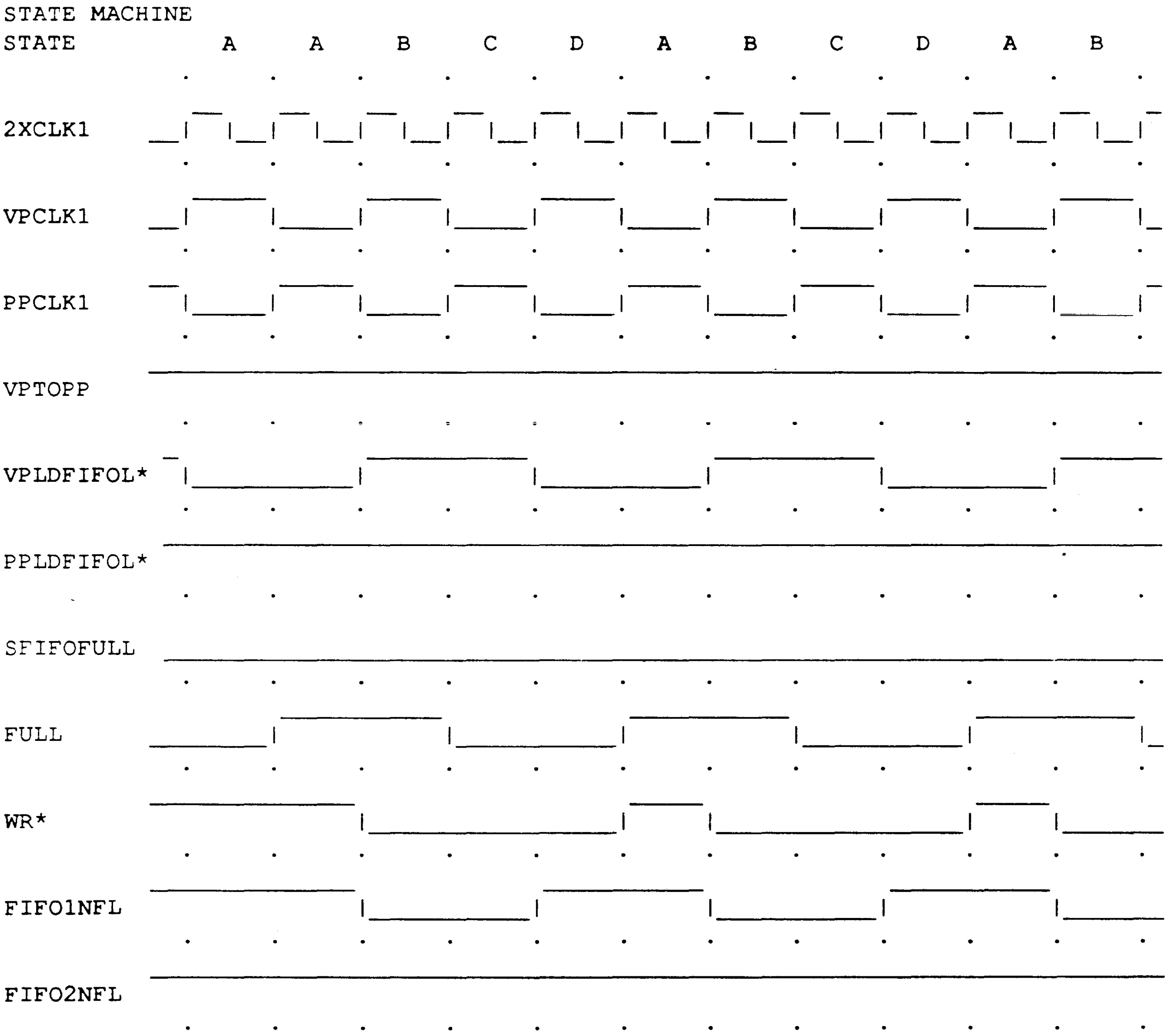
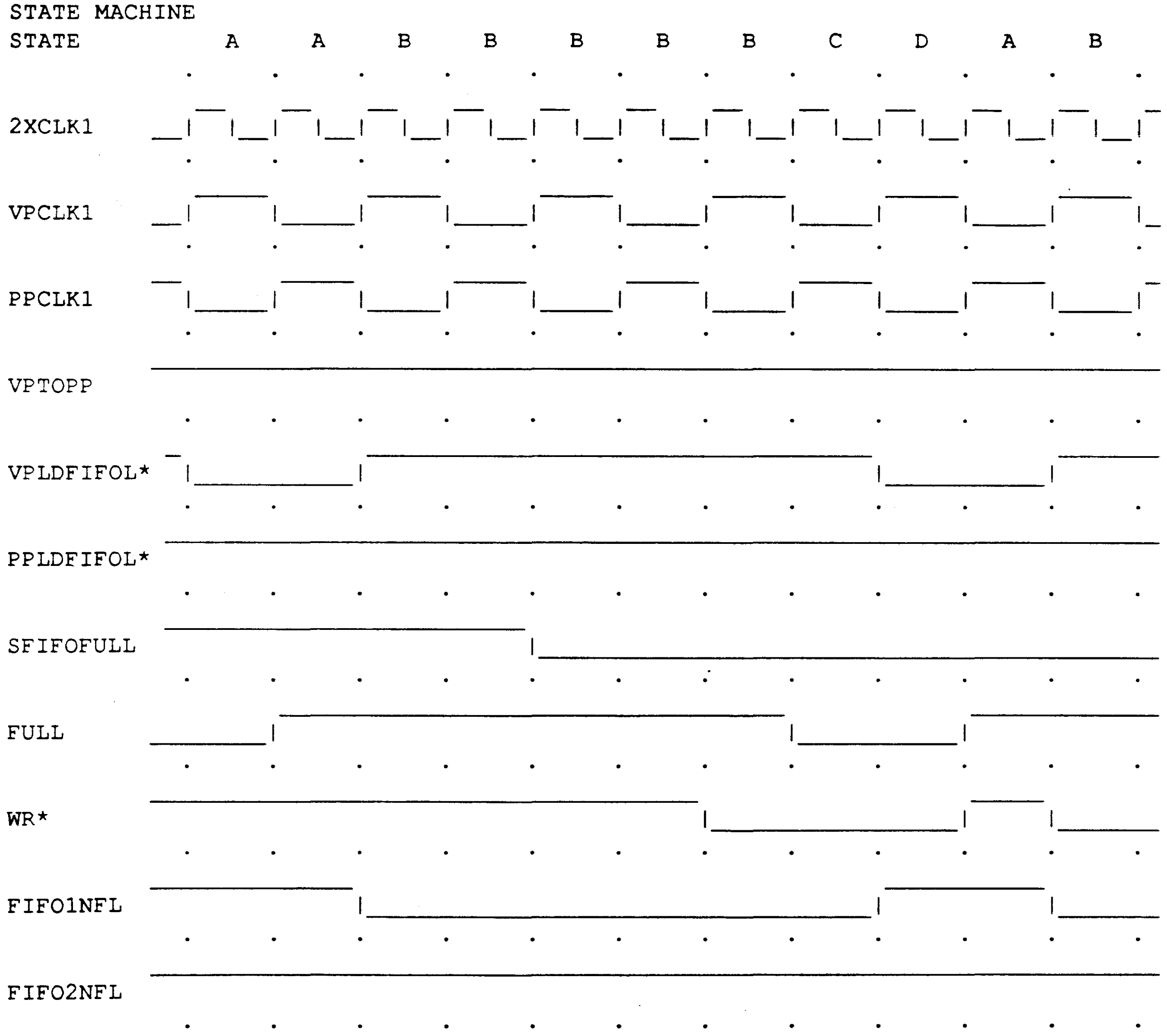


Figure 11-2 *FIFO Write Timing Diagram— VP to PP*

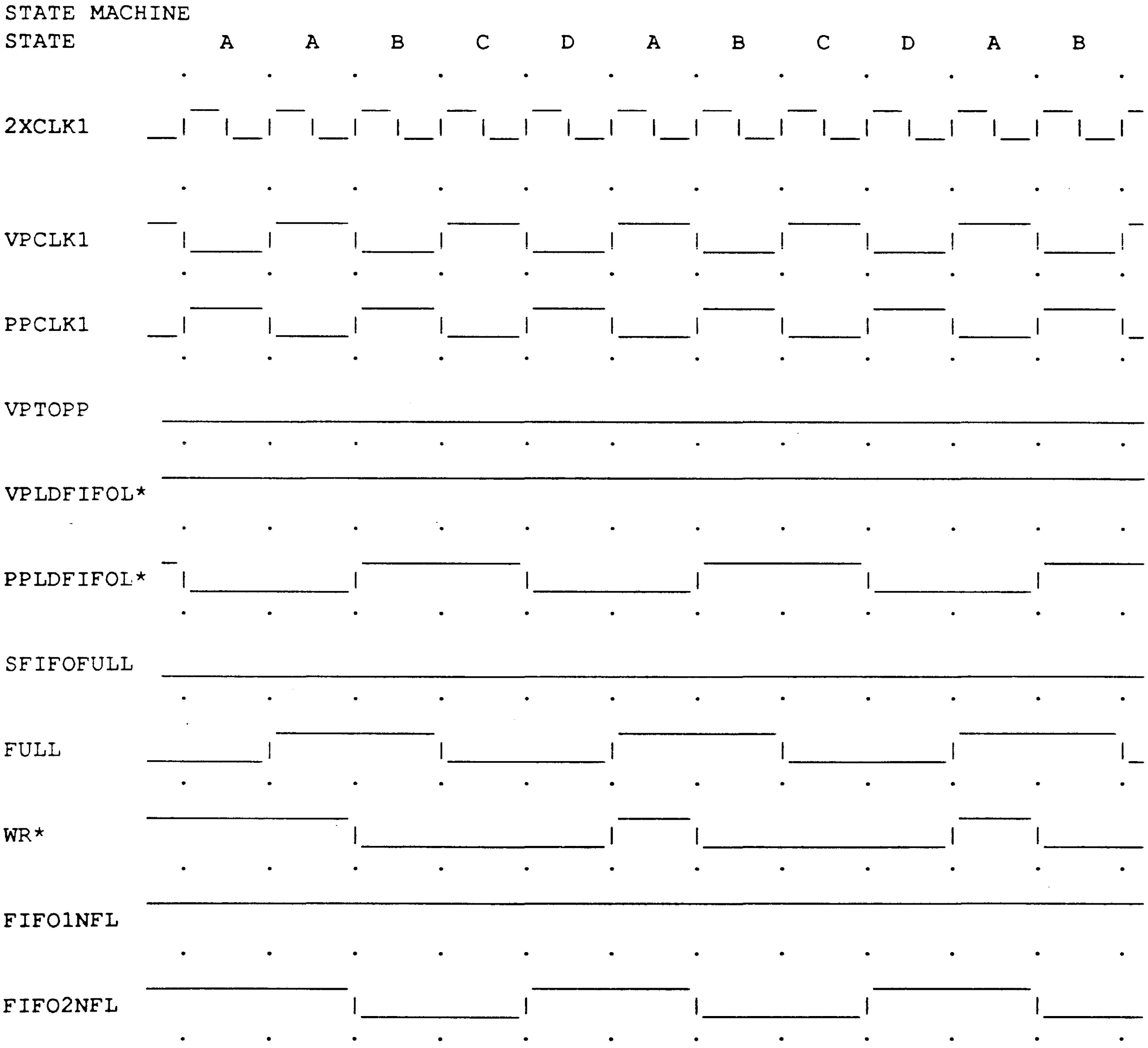
```
FIFO passing data from the VP to PP, consecutive writes with
SFIFOFULL active for a time during the first write.
```



SUN PROPRIETARY

Figure 11-3 *FIFO Write Timing Diagram—PP to VP*

FIFO passing data from the PP to VP, consecutive writes with SFIFOFULL never becoming active.



When the signal RST1* (reset) becomes active the FIFO write state machine goes to state A, waiting for one of the processors to write data to the FIFO write register. The signal VPTOPP is input to the state machine for determining which of the processors can write to the FIFO.

- If VPTOPP is active (FIFO data is going from the VP to the PP), the signal VPLDFIFOL* (VP load FIFO, level) is sampled to determine when the FIFO write register has data in it.
- If VPTOPP is not active (FIFO data is going from the PP to VP), the signal PPLDFIFOL* (PP load FIFO, level) is sampled to determine when the FIFO write register has data in it.

VPLDFIFOL* and PPLDFIFOL* are signals that are active for a whole PPCLK or VPCLK period. Because the data from the VP or the PP bus is not going to be written until the end of the PPCLK or the VPCLK period, the state machine (which operates with 2XCLK) must synchronize its operation to the correct phase of PPCLK or VPCLK. So when PPLDFIFOL* or VPLDFIFOL* becomes active and VPCLK or PPCLK are a logic high, the state machine will stay in state A one more 2XCLK cycle then go to state B.

When going from state A to B, the state machine will activate the signal FULL, which deactivates one of the FIFONFL signals—depending on which processor is writing. Deactivating the FIFONFL signal notifies the processor that a write operation is in process and that it should not try to do any more writes until this one is complete. When going from state A to B the signal WR* may also be activated depending on SFIFOFULL.

- If SFIFOFULL is not active, indicating that the FIFO has room in it to write data, then the WR* signal will be activated.
- If SFIFOFULL is active, indicating that the FIFO is full, the state machine will not activate WR*.

If SFIFOFULL is active the state machine will stay in state B, asserting FULL, until SFIFOFULL becomes inactive. When SFIFOFULL is deactivated, the state machine will synchronize to the correct phase of VPCLK or PPCLK and then activate WR*. VPCLK/PPCLK are synchronized so that the FULL signal is deactivated at the correct time with respect to PPCLK and VPCLK.

When WR* becomes active in state B the state machine goes to state C, and then to state D, while activating WR* for one last 2XCLK period. From state D the state machine goes directly to state A. The extra 2XCLK period delay caused by going through state D to state A is used synchronize the state machine to either PPCLK or VPCLK so that if it should get to state A and find another write operation pending, it can go immediately to state B, starting another write operation.

When data is available from the FIFO, the FIFO read control reads the data out of the FIFO by activating the signal RD (from the PAL at A2 on page 24) then signals the processors that FIFO data is available. It is implemented with a state machine and logic contained in a 16R4A PAL and an F74 shown in the bottom right of page 24. The FIFO read control senses that there is data available from the FIFO when the signal SFIFOMT goes inactive. When RD is activated, data at the top of the FIFO is available at the FIFOQ bus. At this time one of the two flags, FIFO1NMT (FIFO 1 not empty) or FIFO2NMT (FIFO 2 not empty), is activated. The FIFONMT signal is sensed by the processor which reads the data through its read buffer. When the processor reads the data, the state machine activates the appropriate FIFONMT signal (saying that the FIFO does not have any data available now) until it reads more data from the FIFO when it is available.

The two signals FIFO1NMT and FIFO2NMT are used by the PP and the VP, respectively, as indication that data is available to be read from the FIFO. Only one of them is valid at a time, determined by which way the FIFO is turned, while the other one is kept in its active state (indicating that the FIFO is not empty so that the software will not hang in the case of an incorrect read).

- If VPTOPP is active (FIFO is from VP to PP), then FIFO1NMT is valid while FIFO2NMT is kept in its active state.
- If PPTOVP is active (FIFO is from PP to VP), then FIFO2NMT is valid while FIFO1NMT is kept in its active state.

These two signals are kept in their active state (indicating that there is data available from the FIFO) when not valid so that the software will not hang if the processor that should be writing to the FIFO erroneously tries to read from the FIFO. When valid, the FIFONMT signals are controlled by the signal EMPTY from the read control state machine.

The state diagram for the FIFO read control state machine is shown in Figure 17 of the appendix.

In the text below are four FIFO read timing diagrams.

Figure 11-4 *FIFO Read Timing Diagram—VP to PP*

FIFO passing data from the VP to PP, consecutive reads with SFIFOMT never becoming active.

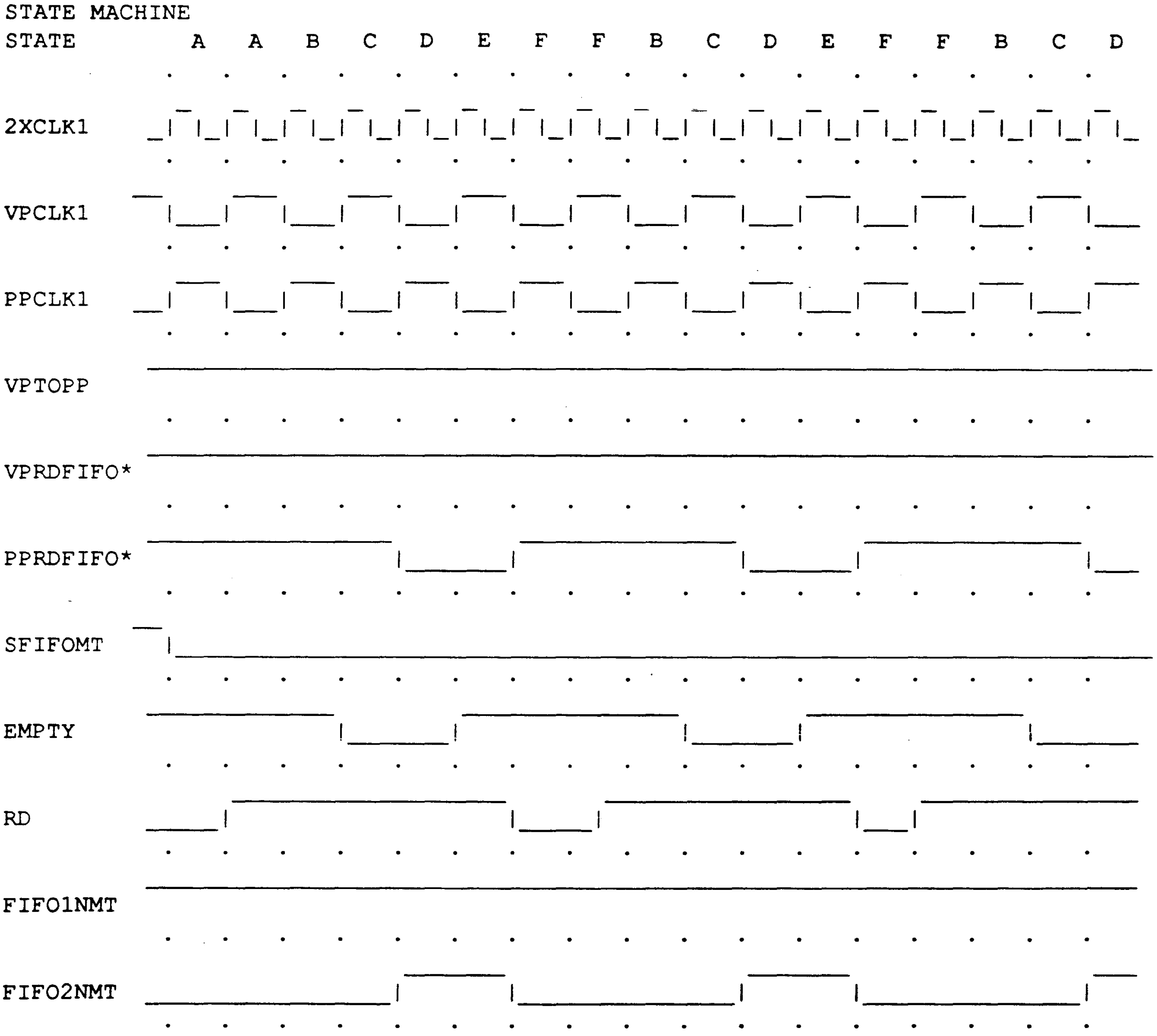


Figure 11-5 *FIFO Read Timing Diagram—PP to VP*

FIFO passing data from the PP to VP, consecutive reads with SFIFOMT never becoming active.

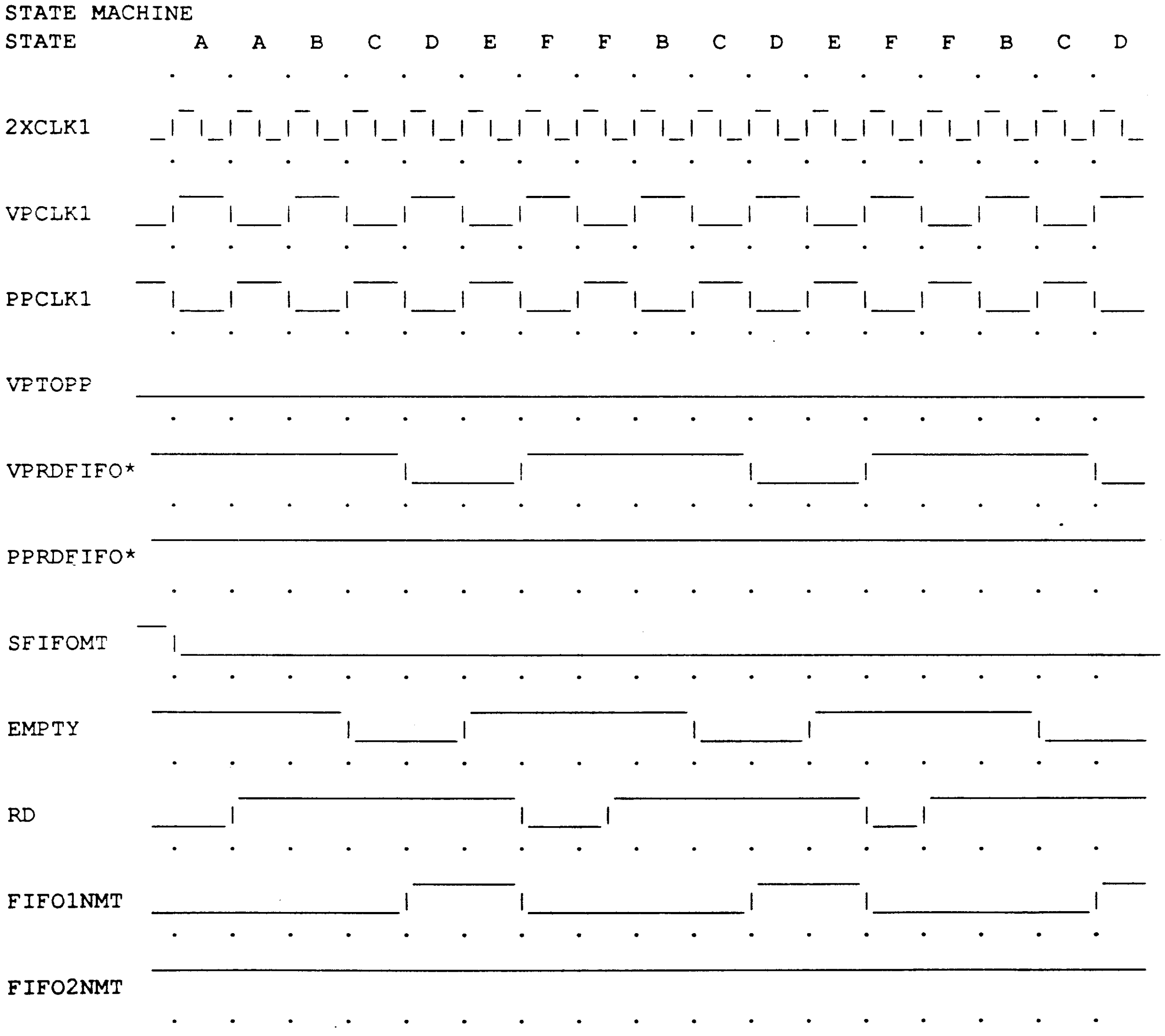


Figure 11-6 *FIFO Read Timing Diagram—PP to VP*

FIFO passing data from the PP to VP, data from FIFO is available but processor does not read it immediately.

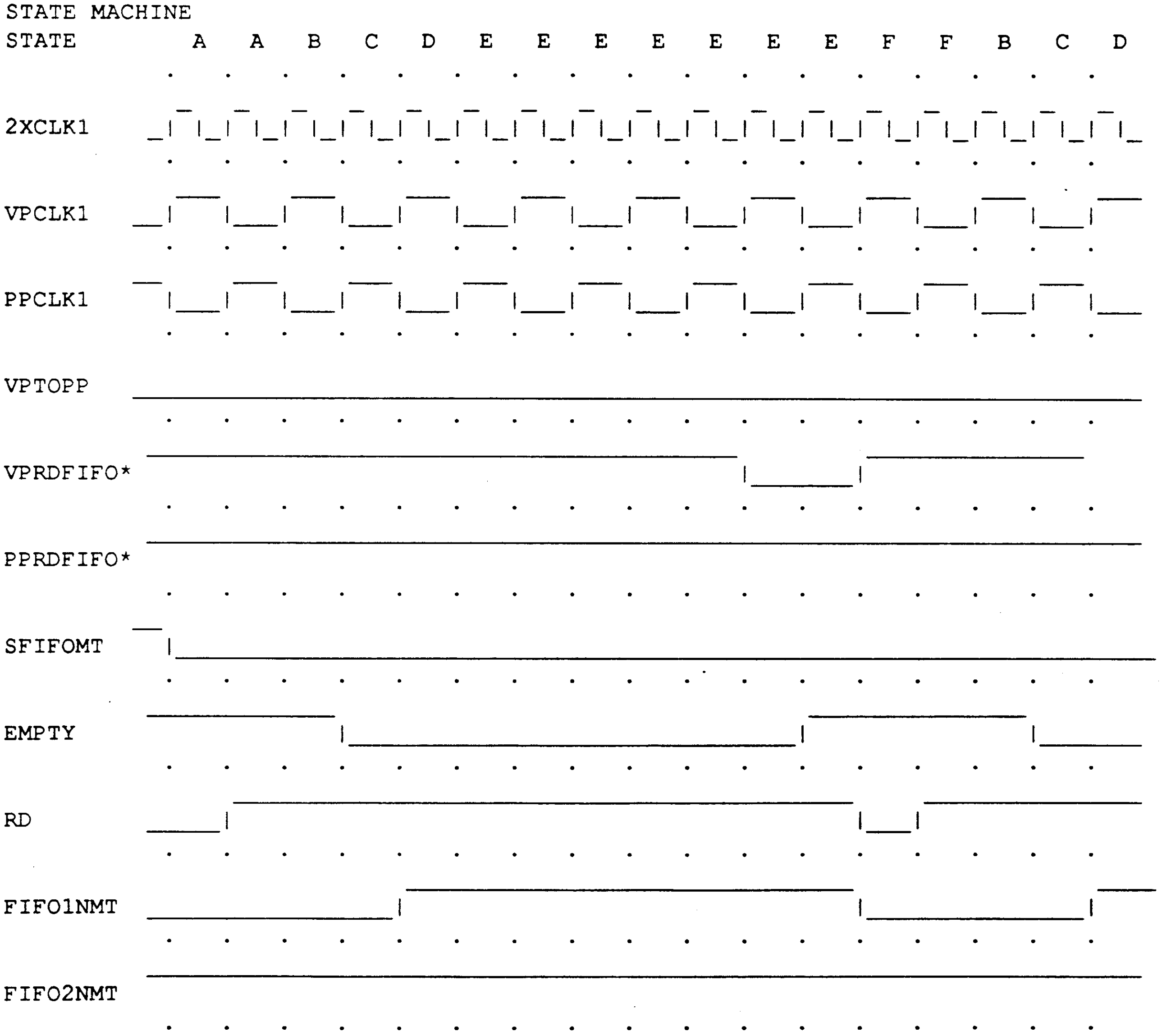
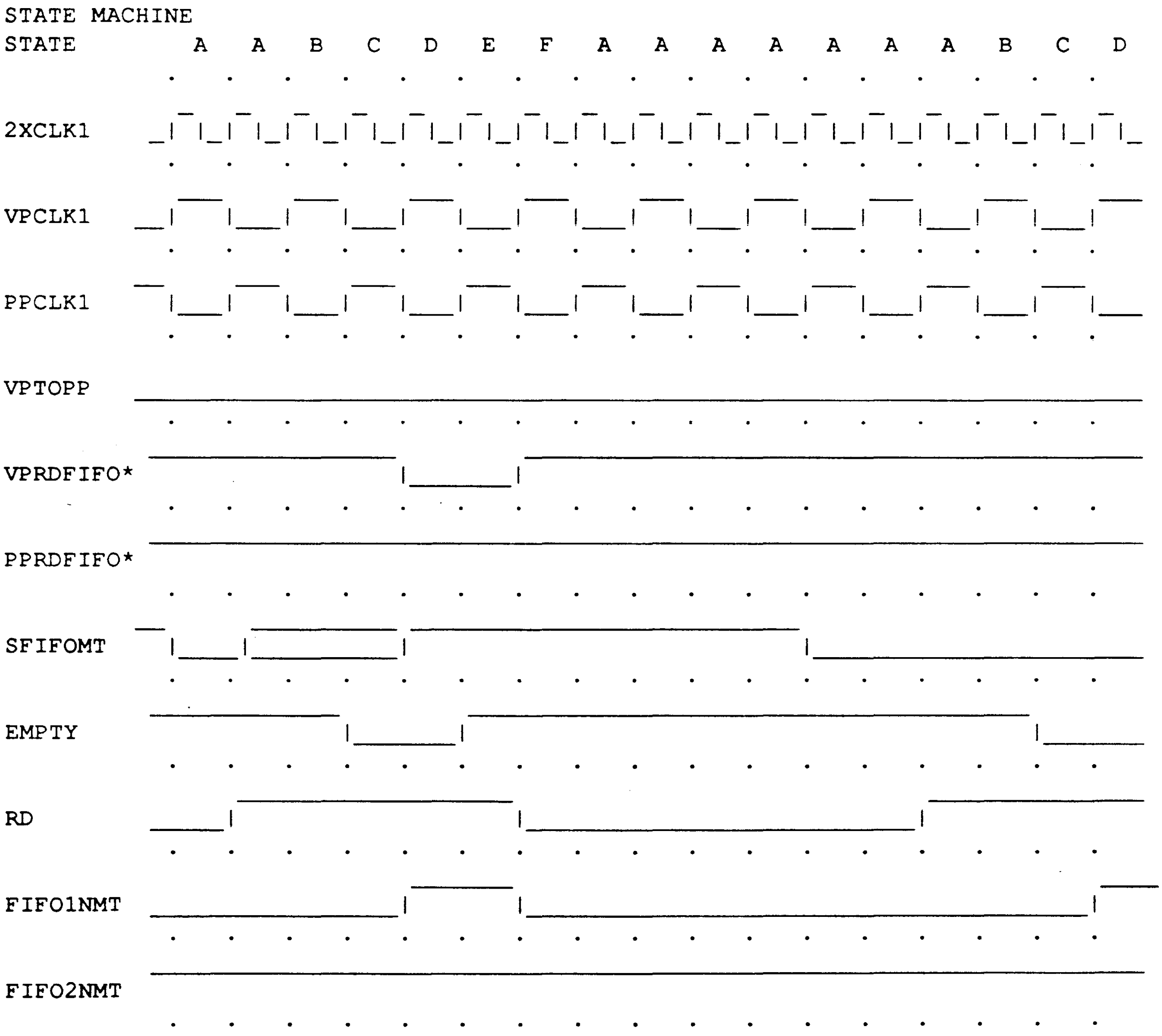


Figure 11-7 *FIFO Read Timing Diagram—PP to VP*

FIFO passing data from the PP to VP, data from FIFO is read causing SFIFOMT to become active, processor reads data, delay until SFIFOMT becomes inactive.



SUN PROPRIETARY

The signal VPTOPP determines whether to look at the input signals from the VP or the PP.

- When VPTOPP is active only the read signal PPRDFIFO* and PPCLK from the PP is valid, while VPRDFIFO* and VPCLK are ignored.
- When VPTOPP is not active only the read signal VPRDFIFO* and VPCLK from the VP is valid while PPRDFIFO* and PPCLK are ignored.

The state machine stays in state A if the FIFO is empty while asserting the EMPTY signal. When the FIFO has data in it, then the signal RD is activated and the state machine goes to state B and then C. In state C, EMPTY is deactivated because the data from the FIFO is available at its output to be read by the processor. The state machine keeps RD activated and goes to state D and then E. It waits in state E for the processor to read the data. When the processor reads the data, it goes to state F. If the FIFO has more data in it, the state machine goes to state B from F to start another read cycle. If the FIFO is empty the state machine goes to state A from F.

A sample process goes something like this:

1. when the signal RST1* (reset) is activated, the state machine goes to state A.
2. The state machine stays in state A as long as the FIFO is empty—keeping EMPTY activated. SFIFOMT, when active, indicates that the FIFO is empty.
3. When SFIFOMT becomes deactivated, the state machine synchronizes to the correct phase of VPCLK or PPCLK—determined by VPTOPP—and activates RD. (This synchronization ensures that EMPTY is deactivated on the correct phase of PPCLK or VPCLK when going from state C to D.)
4. When RD is activated, the state machine goes to state B and then to state C.
5. From state C the state machine goes to state D while deactivating EMPTY.
6. In state E it waits for the processor to read the data.
7. The processor reads the data by activating VPRDFIFO* if VPTOPP is inactive, or PPRDFIFO* if VPTOPP is active.
8. The state machine synchronizes to the correct phase of VPCLK or PPCLK and goes to state F while activating EMPTY again.
9. When in state F, the signal SFIFOMT is again sampled to see if the FIFO has more data to be read.
10. If SFIFOMT is active, it indicates that there is no more data in the FIFO at this time and the state machine goes to state A.
11. If SFIFOMT is not active, indicating that there is more data in the FIFO to be read, the state machine activates RD and stays in state F one more clock cycle and then goes to state B starting another read cycle.

12

Painting Processor *(pages 25-27)*

12

Painting Processor *(pages 25-27)*

Pages 25-27 contain the basic elements of the Painting Processor—the microprocessor, the microsequencer, the instruction register, and the PPBUS source/destination decode. When reading the following section, it will be necessary to refer to the GP Hardware Reference Manual, especially the section that describes the microinstruction format. Also note the great similarity of schematics pages 15-17 with pages 25-27.

12.1. Instruction Register

The instruction register (referred to as the pipeline register in the AMD literature) is a 56-bit register holding the microinstruction currently being executed. Almost everything that happens in this cycle is controlled by the contents of this register.

As shown on page 25, the instruction register is implemented with seven F374 8-bit registers. The instruction register input comes directly from the microstore. The output bits are referred to as PPIR00 to PPIR55. The register is loaded on each rising edge of PPCLK0. This edge thus defines the end of one cycle and the beginning of the next.

12.2. General Field

Bits PPIR00-PPIR15 are called the general field. With this field it is possible to put assembly time constants onto the PPBUS and route it to a selected PPBUS destination. The ALS244 buffers are used to perform this function. Bits PPIR00-PPIR15 are routed to PPBUS00-PPBUS15 if enabled by the control signal PPRDGF*, a PPBUS source control signal.

12.3. PPBUS Source/Destination Decode

The PPBUS source/destination combination is chosen by PPIR44-PPIR50. These bits are decoded by the PALs on page 25 to generate the control signals that are routed to each source and destination. When a PPBUS source is enabled, the selected subsection places its data word on the PPBUS. Then the selected PPBUS destination strobes this data word into its subsection at the end of the current cycle (rising edge of PPCLK). The timing constraint is that for each allowable source/destination combination, the source must place the data word on the PPBUS early enough to meet the set-up time of the destination, and keep the data on the PPBUS long enough to meet the hold time. This constraint is further complicated by the time variability of the rising edge of the destination's load strobe.

The PPBUS sources are decoded by the 16L8 PAL at D2. This is a combinatorial PAL that decodes the source/destination select bits. The possible sources and the enables are listed below.

Table 12-1 *PP Bus Sources and Their Enables*

Source	Enable
Am29116 microprocessor	PPRD116*
Interprocessor flags (VP-to-PP)	PPRDFLG*
General Field	PPRDGF*
FIFO (VP-to-PP)	PPRDFIFO*
Scratchpad memory	RDSPAD*
VME status register	RDVMES*
VME data register	RDVMED*

There are other PPBUS sources on the Graphics Buffer board and they are decoded by the PAL on page 30.

The PPBUS destinations are decoded by the PALs at C3 and B3 and the F138, a 3-to-8 demultiplexer. There are two types of destination control signals—loads, and load enables.

- Loads are ANDed with a clock and are used directly to strobe the PPBUS into the selected destination (for example, 374 registers).
- The load enables are not ANDed with a clock; these destinations have a direct clock input plus the load enable (for example, 163 counters).

The PPBUS destinations and their controls are listed below.

Table 12-2 *PPBUS Destinations and Their Controls*

	Destination	Control
Loads		
	Am29116 microprocessor	PPDLE
	FIFO (PP-to-VP)	PPLDFIFO*
	PP branch register	PPLDBR*
	Interprocessor flags (PP-to-VP)	PPLDFLG*
	VME data register	LDVMED*
	VME interrupt id register	LDIID*
	VME control register	LDVMEC*
Load enables		
	Scratchpad memory pointer	LDSPPTR*
	Scratchpad memory (data)	LDSPAD*
	VME high address register	LDVMEHADR*
	VME low address register	LDVMELADR*
	PP status register	PPLDSTAT*
	PP n register	PPLDN*
	FIFO (PP-to-VP)	PPLDFIFOL*
	VME interrupt id register	LDIIDL*

Other destinations, corresponding to the Graphics Buffer board, are decoded by the PAL on page 30.

Some destinations have both a load and a load enable control—for example, the VME interrupt identification register. The signal LDIID* loads the ALS374 that is used as the VME interrupt vector (page 29), while the signal LDIIDL* enables the setting of a flip-flop on the rising edge of PPCLK2. This flip-flop is used to initiate the VME interrupt cycle (page 31).

RST1* is input to the PALs (at page 25 location B3 and C3) to prevent the generation of load enables when RST1* is active.

Other signals are input to the B3 PAL to prevent overwrites and other such hardware faults:

- VMEBUSY* is used to inhibit writes to the VME data register or VME control register when the VME subsection is still busy with the previous operation;
- FIFO2NFL* is used to prevent writes to the FIFO when it is full.
- The signal ZBUFRDY* is used to inhibit the SETZBBSY* signal. SETZBBSY* is generated if the graphics buffer is the PPBUS destination (encodings of the PPIR PPBUS source/destination bits). This signal is used to set the ZBUFRDY signal signifying that the graphics buffer is busy and another operation should not be initiated until the active one completes (page 30 of the schematics).

The F138 (at A2 of page 25) is used to generate the load-type destination signals. 2XCLK1 and PPCLK1* create the properly-shaped load pulse as shown in the figure below. The PPIR bits are translated into another encoding of the PPBUS destinations by the PAL at B3. Then these signals, PPDEST0* to PPDEST2*, are routed to and decoded by the F138 to generate the load signals.

Figure 12-1 *Painting Processor Destination Load Timing Diagram*

```
Destination load signals

          ____      ____      _
2XCLK1  _|    |____|    |____|

        _           _________
PPCLK1*  |_________|         |_

LOAD    _                     _
ENABLE*  |___________________|

        ________________      _
LOAD*                   |____|
```

NOTE *Skew delays not shown.*

12.4. PP Microsequencer

As shown on page 26 of the schematics, the PP microsequencer is based on the Am2910A microprogram controller. Inputs to this chip include a 4-bit instruction, a 12-bit data word, condition code control, and a clock. Output is a 12-bit address used to fetch the next microinstruction.

The Am2910A instruction normally comes from bits PPIR40-PPIR43. The only exception is when the PP is forced to begin execution at microstore location zero. (The PP can be forced to begin execution at location zero with a write to the GP control register via the VME bus.) The four AND gates on page 26 perform this function.

- □ If no PPJZ* is active, PPIR40-PPIR43 are routed to the Am2910A instruction.
- □ If PPJZ* is asserted, zero is routed to the Am2910A and a jump zero instruction (which also resets the Am2910A) is executed.

Two possible sources can be routed to the Am2910A "D" (data) inputs—the PP branch register or the PP general field. The general field provides assembly time constants while the branch register provides runtime values. If PPIR54 is a 0, the general field is enabled. If PPIR54 is a 1, the branch register is enabled. (See also the 3-way branch described below.) The Am2910A uses this input as a possible next microstore address or for internal use.

- □ The branch register is implemented with two F374 registers. It is loaded when it is chosen as the PPBUS destination by the PPLDBR* signal and is routed to the Am2910A by PPENBR*.
- □ The general field comes directly from the PP instruction register and the F244 buffers route PPIR00 to PPIR11 to the Am2910A when enabled by PPENGF*.

The condition code logic selects a condition to test and determine whether or not a jump condition passes or fails. Eight condition code flags are implemented.

Table 12-3 *Condition Code Logic*

Condition	Flag
VME ready	VMERDY
Graphics buffer ready	ZBUFRDY
FIFO (VP-to-PP) not empty	FIFO1NMT
FIFO (PP-to-VP) not full	FIFO2NFL
Am29116 result overflow	PPOVR
Am29116 result carry	PPCAR
Am29116 result negative	PPNEG
Am29116 result zero	PPZERO

The four Am29116 status flags are stored in the F163 register (whose count capability is not used) at A5. If so enabled by PPIR53, the Am29116 status for the current cycle is stored in this status register for use in a subsequent cycle.

One of the eight conditions is selected by the F151 multiplexer controlled by PPIR36 to PPIR38. Either polarity of the test condition is then generated by the

XOR gate (PPIR39 selects the polarity) and routed to the CC/ input of the Am2910A.

Notice that unlike the VP, there is no logic 1 as one of the branch conditions. However, unconditional branches are still necessary. PPIR55 is used as the Am2910A CCEN* bit. If PPIR55=1, the Am2910A executes the pass condition. If PPIR55=0, the selected condition code determines the pass/fail condition of the Am2910A.

Another difference between the VP and PP is the 3-way branch ability. If a 3-way branch is enabled, the hardware first determines if the VME section is busy; if so, it jumps to the location specified in the general field (usually the current instruction). If the hardware determines that the VME is not busy, it executes the "normal" Am2910A instruction. If this "normal" instruction is a 2-way branch, then effectively the hardware has executed a 3-way branch.

3-way branch works as follows. The microinstruction must conform to:

```
PPIR55 = 0
PPIR54 = 1
PPIR51 = 1
PPIR33 to PPIR35 = 111
```

If the VME is busy (VMEBSY=1), then the AS30 at A2 is asserted (output is equal to zero) and this causes:

1. PPCCEN to be high, and the Am2910A executes a pass,
2. PPENBR* not to be asserted, and
3. PPENGF* to be asserted enabling the PP general field to the Am2910A.

If an Am2910A conditional jump instruction, for example, is being executed, the condition passes and the PP will jump to the address contained in the general field.

If the VME is not busy (VMEBSY=0), then the AS30 is not asserted (output is equal to one) and this causes:

1. PPCCEN to be low (since PPIR55=0) and the Am2910A will pass or fail based on the selected condition code,
2. PPENBR* to be enabled (since PPIR54=1) and the branch register is routed to the Am2910A, and
3. PPENGF* not to be asserted.

If an Am2910A conditional jump instruction, for example, is being executed, the selected condition code will determine the pass/fail state and the PP will either jump to the address contained in the branch register or continue with the next sequential instruction.

As explained earlier in the microstore description, the basic cycle is split into two halves. The first half is used to compute the address of the next microinstruction, and the second half is used to access the memory. Thus the microsequencer has one half-cycle (60 nsec) to compute the next address. The worst case path is the

CC path: 3 nsec for clock skew, 10 nsec for valid PPIR bits, 9 nsec S to Y* through the F151, 8 nsec through the XOR gate, and 30 nsec CC* to Y in the Am2910A for a total of 60 nsec.

The Am2910A provides 12-bit addresses (PPMSA00-PPMSA11) and thus can access 4K words of microinstructions. This was judged not to be enough, so a bank switch capability augments this address space. The PAL at D2 on page 26 is used to implement this feature.

The inputs to the PAL include a clock, the Am2910A instruction, and three high-order ''D'' bits to provide the larger address from either the branch register or the general field.

- If the Am2910A instruction is a jump zero, the PAL outputs bank 0.
- If the Am2910A instruction is a jump map, the PAL outputs the bank selected by the high-order ''D'' bits.

For these two cases, the new bank is updated in the PAL and is used as the selected bank until the next jump zero or jump map instruction. Otherwise, there is no change to the bank selected by the PAL.

12.5. Microprocessor

On page 27 is the microprocessor, the Am29116. The Am29116 inputs include:

- a clock,
- a 16-bit instruction,
- a 16-bit (bidirectional) data bus, and
- three control signals (OEY*, DLE, and IEN*).

Output are

- a 16-bit (bidirectional) data bus, and
- four status flags (zero, carry, negative, and overflow).

The Am29116 instruction comes from PPIR16-PPIR31. However, two fields can be modified at runtime—the n field and the RAM field. The n field selects a bit position in a bit-oriented instruction or defines a shift count in a shift instruction. The RAM field selects one of 32 on-board registers.

The n field value can be chosen at assembly time or at run time.

- If PPIR52 is 0 then PPIR25-PPIR28 (assembly time value) is routed to Am29116 instruction bits 9-12.
- If PPIR52 is a 1, then the n register value (run time value) is routed to the Am29116 instruction.

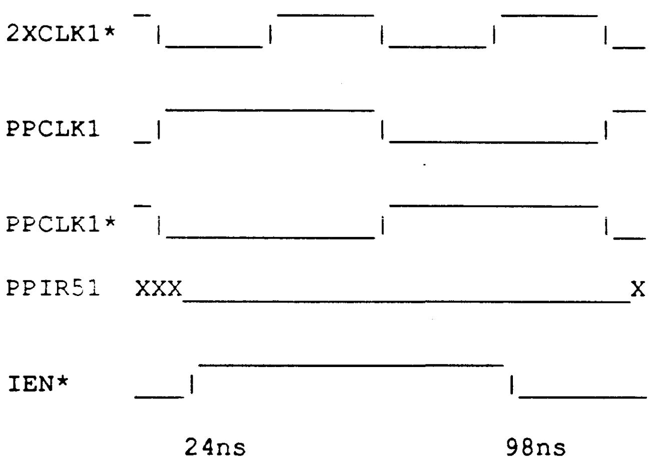
The F241 at D7 of page 27 perfomrs this multiplexing.

The RAM address field is modified to provide double address accesses; that is, the RAM source and destination can be different. If not doing a double address access, PPIR51 is high and bits PPIR16-PPIR19 are routed to the Am29116 instruction bits 0-3 through the entire cycle. If PPIR51 is low, then PPIR16-PPIR19 are routed to the Am29116 for the first half of the cycle and PPIR32-

PPIR35 are routed to the Am29116 for the second half. Because only four of the five RAM address bits are modified, the RAM source and destination must be in the same group of 16, 0-15 or 16-31. The F241 at B7 of page 27 is used for this multiplexing.

To provide double address accesses, the IEN* line must be managed. If not doing a double address access, PPIR51 is high and the F74 at A7 is held clear so that IEN* is low throughout the cycle. If a double address access is active, then IEN* is as shown in the figure below. The F32 at A8 sets the IEN* D flip-flop and the proper 2XCLK1* edge resets the signal.

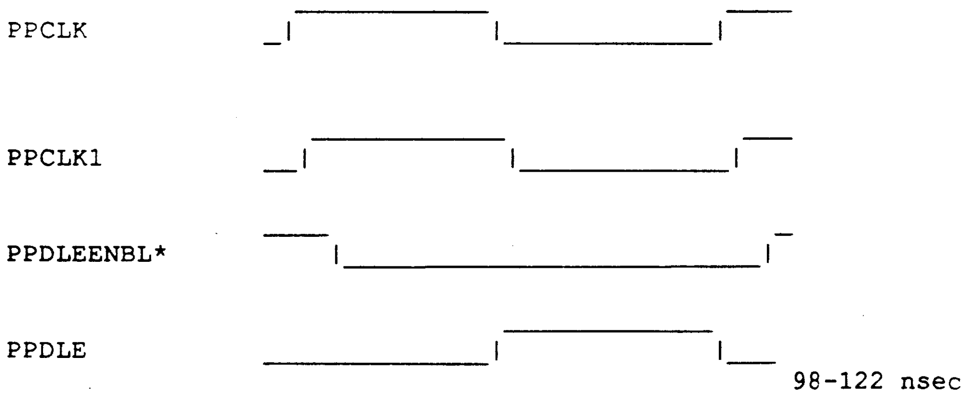
Figure 12-2 *Am29116 Double Address Timing Diagram*



24 nsec easily meets the rising edge spec. 98 nsec exactly meets the worst case falling edge spec.

Figure 12-3 *Am29116 DLE Timing Diagram*

Am29116 DLE timing



The rising edge easily meets specifications. The falling edge can occur anywhere in the above interval so that data must be valid at 88 (to meet set-up time) and stay until 128 (to meet hold time).

NOTE *All times are referenced to PPCLK1.*

The other two Am29116 control lines involve PPBUS activities. If the Am29116 is chosen as the PPBUS source, PPRD116* is asserted enabling the Am29116 internal Y bus onto the PPBUS. If the Am29116 is to be the destination of a PPBUS transaction, PPDLE is asserted and the PPBUS data is latched in the Am29116 internal D latch. The figure above shows the timing of PPDLE. Note that PPDLE occurs early enough so that the Am29116 internal D latch can be considered transparent, and thus the PPBUS data can be used in the current Am29116 cycle except when the Am29116 Y bus or the interprocessor flag register (VP-to-PP) is the PPBUS source.

Also on page 27 are the PPBUS pull-up resistors. They improve the rise times of the PPBUS signals.

12.6. Miscellaneous Controls

The miscellaneous controls are a collection of unrelated commands. They are controlled by PPIR32-PPIR35 and are enabled only if PPIR51 is high (no double address access). These instruction bits are decoded by the PAL at D2 of page 27 to generate various command signals. A complication arises because some of the commands are specified in the current cycle but must executed in a subsequent cycle. The various commands and the control lines are as follows:

Table 12-4 *PP Miscellaneous Controls*

Command	Control Line
Count VME address registers	CNTVME*
direction control (up or down)	DNVME*
Clear scratchpad-memory pointer	CLRSPDP*
Count up scratchpad-memory pointer	2CNTSPDP
Start VME read or write operation	STRTVME
select write	VMEWR*
Start graphics buffer read	ZBFSTRTRD*

The F374 register holds the bits that indicate that a control in this cycle affects a future cycle. These include the following:

□ Load scratchpad-memory — scratchpad-memory write to be done in next cycle;

LDSPAD* is the current cycle signal;

DELLDSPAD* is the next cycle signal.

□ Count scratchpad-memory pointer — count to be done in next cycle if coincident with a load scratchpad-memory;

CNTSPDP* is the current cycle signal;

2CNTSPDP is the current or next cycle signal.

The commands CNTVME*, DNVME*, CLRSPDP*, STRTVME, VMEWR*, and ZBFSTRTRD* are executed in the same cycle in which they appear. The PAL decodes the proper PPIR32-PPIR35 and PPIR51 combinations to generate these control signals. The command 2CNTSPDP may be executed in the current cycle or the next cycle. The corresponding signal CNTSPDP* is decoded in the current cycle and will enable 2CNTSPDP in the current cycle if there is no concurrent write or in the next cycle if there is a concurrent write.

The signal VMEBUSY* is used to inhibit the initiating of a VME operation, read or write, if a VME operation is already active. The signal ZBUFRDY* is used to inhibit the start graphics buffer read if the graphics buffer is already busy.

The signal ZBFSTRTRD* is used to initiate a graphics buffer read and set the graphics buffer flag to busy (see page 30). The signal is routed to the Graphics Buffer board through the P2 connector row A.

The timing constraints on these signals are to meet the set-up times of the count and direction enables. These constraints are easily met.

13

Scratchpad Memory *(page 28)*

13

Scratchpad Memory *(page 28)*

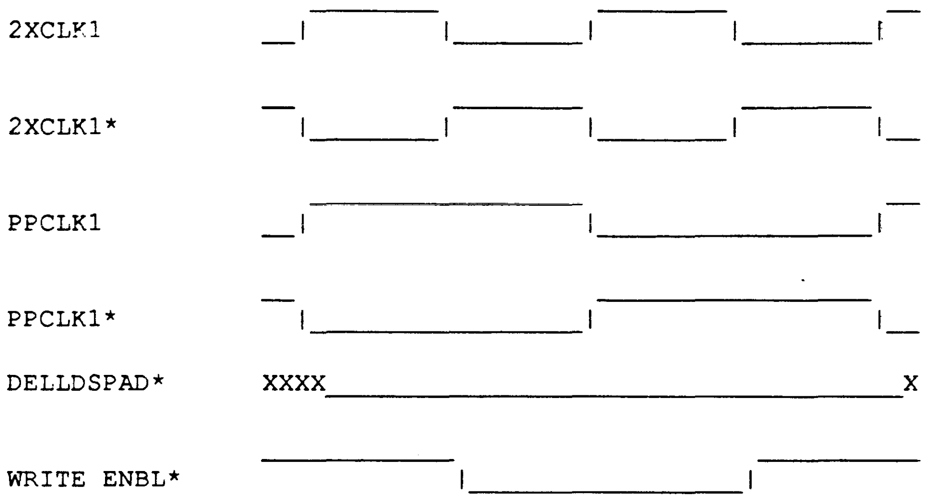
On page 28 of the schematics are the scratchpad pointer, the scratchpad-memory, and the registers necessary to meet the timing constraints of the PPBUS and the 2169 memory chips. Using 4K-by-4 memory chips, the size of the scratchpad is 4K-by-16 (4 Kwords).

The scratchpad pointer is implemented with four F163 counters. The PPBUS is the input to these counter chips and they are clocked on the rising edge of PPCLK1 (end of a cycle) if load-enabled by LDSPPTR*. The counters are incremented on the rising edge of PPCLK1 if 2CNTSPDP is asserted and cleared on the rising edge of PPCLK1 if CLRSPDP* is asserted. The counter outputs are routed to the 2169 memory chips.

During the first half of a cycle (except cycles where a write is active—see below), the scratchpad memory is read and the fetched data word is loaded into the scratchpad data out register, implemented with two F374s at A4 and A2. These registers are loaded on the rising edge of PPCLK1* (the middle of every cycle) and are routed onto the PPBUS if enabled by RDSPAD*.

At the end of every cycle, the contents of the PPBUS are loaded into the scratchpad data in register, implemented with two ALS374s at A5 and A3. If the current cycle selected the scratchpad as the PPBUS destination, then the write is executed in the next cycle. During the next cycle, DELLDSPAD* will be asserted and this signal will set the flip-flop at A7 and trigger the write enable. This write enable is also used to enable the data in register onto the memory chip data bus. Thus the data word captured in the previous cycle is written into the scratchpad. The figure below illustrates the write timing.

Figure 13-1 *PP Scratchpad Memory Write Timing Diagram*



NOTE *Clock skews not accurately shown.*

The scratchpad write cannot be executed in the same cycle the data word is on the PPBUS because PPBUS data cannot meet the set-up times required by the 2169 chips.

CAUTION **Garbage is loaded into the scratchpad data out register during write cycles so that the microcoder should not execute a scratchpad read in the cycle immediately following a scratchpad write.**

14

PP Miscellaneous Logic *(page 30)*

14

PP Miscellaneous Logic *(page 30)*

NOTE *In the remaining sections, the terms "Graphics Buffer" and "G Buffer" are synonymous.*

The Painting Processor miscellaneous logic includes:

- the PP n register,
- the interprocessor flags #2 (PP-to-VP),
- the PP status register, and
- the PP bus extension logic.

14.1. PP n Register

The n register is implemented with a F163 counter but the count functions are suppressed. PPBUS00-PPBUS03 are loaded into the counter on the rising edge of PPCLK2 (end of a cycle) if selected as the PPBUS destination (signal PPLDN*). The n register is multiplexed into the Am29116 instruction stream on page 27.

14.2. Interprocessor Flags #2

This register passes eight flags from the PP (along with three other flags) to the VP. An ALS374 is used to hold the eight flags that are written by the PP. The register is loaded from PPBUS00-PPBUS07 when the signal PPLDFLG* is asserted. There are three other flags which can be read by the VP through the ALS244 device.

- PPTOVP comes from the FIFO direction flip-flop (page 24) and indicates the direction of FIFO data flow. When PPTOVP is a logic high the FIFO is passing data from the PP to the VP.
- SFIFOMT* signal comes from the FIFO (page 24) and, when active, it indicates that all the words from the FIFO devices have been read. Note however that this does not necessarily mean that the processor receiving the FIFO data has *read* all the data. It is possible for the last FIFO data to be read out, and be available on the output of the FIFO devices waiting for the processor to read it. In this case the SFIFOMT* signal will be active while the data is waiting to be read by the processor.
- The last flag on the ALS244 is always loaded as 0. This allows the VP microcode to read this register and, seeing the logic 0 in the ninth bit position, determine that this is the VP.

These 11 flags are enabled onto the VPBUS with VPRDFLG*.

14.3. PP Status Register

The PP status register is implemented with an ALS163 counter but the count functions are not used. PPBUS00-PPBUS03 are loaded into the counter on the rising edge of PPCLK2 (end of a cycle) if selected as the PPBUS destination (by PPLDSTAT*). The 4 outputs are routed to the GP status register (page 6), accessible via the VME bus, and to 4 LEDs. Note that the LEDs are configured so that if a counter bit is 0 (for example, after a reset), the corresponding LED is on.

14.4. PP Bus Extension

The PP bus extension contains the logic necessary to extend the PP bus to the Graphics Buffer. This allows the PP to access the resources of the Graphics Buffer which includes the G buffer memory, integer multiplier, and the PP PROM. The PP bus extension logic includes the G buffer flag, the PP bus extension decode, and the PP bus extension transceivers.

The PP bus extension consists of the PP data bus, some control signals, and the clock signals PPCLKZB and 2XCLKZB. These signals are put through the P2 connector to the Graphics Buffer board and provide the communication channel between the PP and the Graphics Buffer.

The G buffer flag, ZBUFRDY (and its complement ZBUFRDY*), is simply a handshake signal between the PP and the G buffer memory. ZBUFRDY is held in an F109 device shown in the top right of page 30. ZBUFRDY is deactivated, through the J input of the F109, whenever the PP requests an operation to be done to the G buffer memory. ZBUFRDY is deactivated when the signal ZBFSTRTRD*, from the PP miscellaneous controls, or the signal SETZBBSY*, from the PP destination logic, become active. The PP processor looks at ZBUFRDY signal through its condition code logic to determine if the G buffer has completed the present transaction (ZBUFRDY inactive means the G buffer is still busy with the transaction).

When the G buffer memory completes the requested transaction, it activates the signal STZBUFRDY* which activates the ZBUFRDY. Additionally, if the G buffer board is not installed, jumper J7 on page 6 section C8 is not installed, and ZBRDENBL* is a logic high. ZBRDENBL* being a logic high keeps ZBUFRDY always active (this keeps any erroneous accesses to the G buffer memory when the Graphics Buffer board is not installed from hanging the software).

The PP bus extension transceivers are used to buffer PP bus data sent to and received from the Graphics Buffer over the P2 connector. The extension of the PP bus data bits PPBUS15-00 going to the Graphics Buffer is named ZBUS15-00. The F245 devices are used to implement the transceiver. The outputs of the F245s are disabled by keeping the enable pin at a logic high when ZBRDENBL* is at a logic high. This prevents the GP from driving the P2 connector if the Graphics Buffer is not installed. The direction of the transceivers is controlled by the signal ZBRDIR* from the PP bus extension decode PAL. When ZBRDIR* is active the ZBUS is driving the PPBUS.

The PP bus extension decode is implemented with a 16L8A PAL shown in the bottom right of page 30. It decodes the source/destination of the PP bus if it applies to an access to the Graphics Buffer. If the signal ZBRDENBL*, from the jumper J7 on page 6, is not active (that is, jumper is not installed, therefore the Graphics Buffer board is not installed), then the destination signals going to the P2 connector are tri-stated. This prevents the GP from mistakenly driving the P2 connector when the Graphics Buffer is not installed.

The PAL decodes the PPIR source destination field to produce the following Graphics Buffer destination signals.

- ZBRDIR* is used to control the PP bus extension transceivers direction.
- The signal RDMP* (read multiplier) enables the result from the Integer Multiplier onto the ZBUS.
- RDPPPM* (read PP PROM) signal is used to enable the data from the PP PROM onto the ZBUS.
- RDZBUF* (read G buffer) is used to enable the data from the G buffer memory read data register onto the ZBUS.
- The signals ZDEST2* (G buffer destination 2), ZDEST1*, and ZDEST0* are an encoding of the possible destinations of the ZBUS. The ZDEST signals are encoded as follows:

ZDEST 2 1 0	DESTINATION
0 0 0	no destination
0 0 1	G buffer high address pointer
0 1 0	G buffer low address pointer
0 1 1	multiplier mode register
1 0 0	G buffer write data register
1 0 1	PP PROM pointer
1 1 0	multiplier x operand
1 1 1	multiplier y operand

The signals ZDEST2*, ZDEST1*, ZDEST0* and RDZBUF* can only be active if the ZBUFRDY signal is active. This prevents the G buffer from loading the data from the ZBUS or starting another G buffer memory operation until the current one is complete.

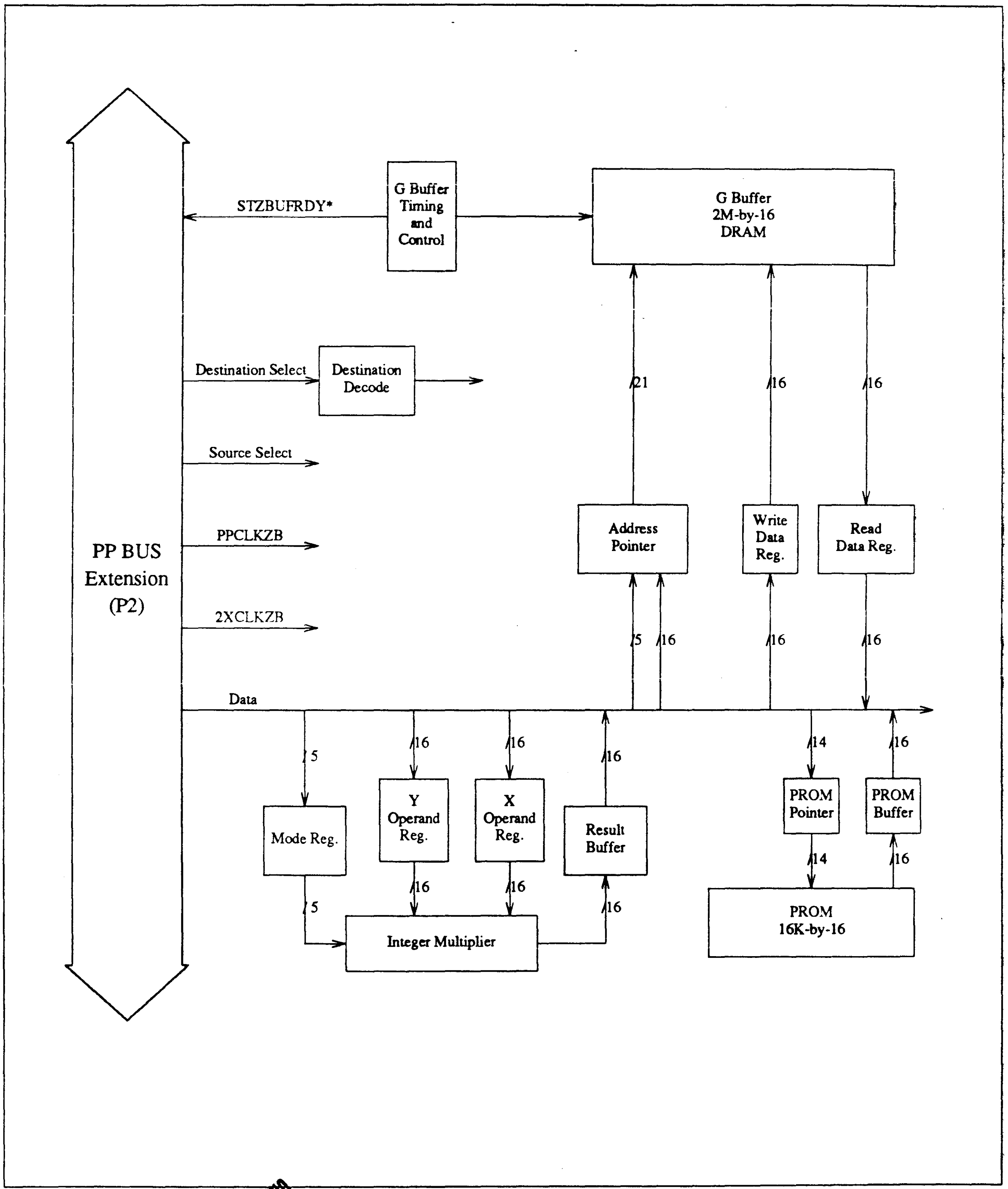
15

Graphics Buffer Board

Graphics Buffer Board

The Graphics Buffer communicates with the GP through signals on the P2 connector. The Graphics Buffer includes the destination decode, G buffer memory registers, G buffer memory, G buffer memory control, integer multiplier, and the PP PROM. The figure below shows the block diagram of the Graphics Buffer board. The following section discusses the Graphics Buffer board and accordingly all the schematic page references apply to the Graphics Buffer schematic.

Figure 15-1 *Graphics Buffer Block Diagram*



15.1. Connectors and Miscellaneous

Page 1 is the title page, listing revisions, spares, PAL and PROM part numbers cross referenced to the IC designations on the schematic and other notes.

Page 2 shows the three 96-pin connectors that interface the Graphics Buffer board to the backplane. The Graphics Buffer uses rows A and C of the P3 connector for power input and uses row A of the P2 connector as the private bus between itself and the GP.

The Graphics Buffer does not use the VME bus and therefore the four bus grant daisy chain lines (BG0-BG3) are merely bused across the Graphics Buffer board. The interrupt acknowledge daisy chain line (IACK) is similarly bused in order not to break the daisy chain.

The bulk and decoupling capacitors are shown on page 3. One decoupling cap is used for approximately each pair of TTL chips (14-, 16-, or 20-pin); the actual number is based on the printed circuit board layout. One cap per memory chip is allocated, and one cap per >20-pin chip is provided.

The ground test points are merely wire loops consisting of jumpers spread out on the PC board to allow for convenient ground connections for a scope probe, etc.

15.2. Destination Decode

Page 4 of the Graphics Buffer schematic contains the:

□ destination decode,

□ reset,

□ pull-up and pull-down resistors, and the

□ free-running PPCLK.

The destination decode is implemented with an F138, 3-to-8 decoder. The destination signals ZDEST2*, ZDEST1*, and ZDEST0* are input to the select lines of the decoder while the enable lines are controlled by PPCLKZB and 2XCLKZB. The decoded outputs are active in the second half of the PPCLKZB cycle when 2XCLKZB is a logic low similar to the load signals in the GP destination logic. The signals that are output from the destination decode are:

LDZPNTRH*	Load G buffer high pointer (MSBs)
LDZPNTRL*	Load G buffer low pointer (LSBs)
LDMMREG*	Load multiplier mode register
LDZBUF*	Load G buffer write data register
LDPPPROM*	Load PP PROM
LDMXOP**	Load multiplier X operand
LDMYOP*	Load multiplier Y operand

The reset for the Graphics Buffer is activated from two sources, the VME bus signal P1.SYSRESET* (system reset) and a jumper tied to a pull-up for a manual reset. Note that a trace has been routed through this jumper so that a header and a jumper is not necessary. The reset circuit is in the top left corner of page 4 and produces the signal RST*.

The Graphics Buffer uses pull-up and pull-down signals (in place of direct connection to +5 and ground) to facilitate ATE testing. The pull-up and pull-downs are shown in the lower left corner of page 4.

A free-running clock is produced on the Graphics Buffer for refresh of the G buffer memory (which is composed of dynamic memory devices). The free-running clock (FRCLK) is generated by dividing the 2XCLKZB signal in two with an F109 flip-flop. FRCLK is the same frequency as PPCLKZB but it may have a 180 degree phase difference from it. Additionally, as the name implies, FRCLK is never halted (unlike PPCLKZB).

15.3. G Buffer Memory Registers

The G buffer memory registers include:

- the write data register,
- the read data register, and
- the G buffer pointer.

These registers are used to interface the G buffer data with the ZBUS.

The G buffer write data register is shown on the top right of page 6 and is implemented with ALS374 devices. The contents of ZBUS is loaded into the register when the signal LDZBUF*, from the destination decode, becomes active. The output of the register is always enabled and goes through a series termination resistor to become the G buffer memory data in bus ZD15..ZD00.

The G buffer read data register is shown on the bottom right of page 6 and is implemented with F374 devices. The input to the register is the G buffer memory data out bus ZQ15..00. The contents of ZQ bus is loaded into the register by the rising edge of the signal LDRDREG* (load read register). LDRDREG* is actually delayed by an F32 gate to meet the set up time of the F374 for worst case timing analysis. The contents of the register are enabled onto the ZBUS when the signal RDZBUF*, from the GP PP bus extension decode, becomes active.

The G buffer address pointer is a 21-bit up counter implemented with F163 devices. The G pointer is shown on the left and top of page 7. Because the 21-bit counter must be loaded from the 16-bit ZBUS, it is broken into two parts, the G buffer low and the G buffer high address pointer. The G buffer low address pointer has the 16 LSBs while the high address pointer has the 5 MSBs of the counter. The low address pointer is loaded by the signal LDZPNTRL* (load G pointer low) and the high address pointer is loaded by the signal LDZPNTRH* (load G pointer high). The G pointer is incremented when the signal ZPNTRCNT* (G pointer count) is activated.

The counter may be incremented by one or by four depending on the state of the signal FILL (fill mode active). In order to implement this counting by four, one extra F163 device was necessary (the 21-bit counter is implemented with seven F163 devices). When the signal FILL is active the G pointer increments by four counts instead of by one count. The first two LSBs of the counter, ZADR00 and ZADR01, are kept in the first F163 device separate from the next high order bits in the next device. If FILL is not active, then ZADR00 and ZADR01 both have to be a logic high for the next device to be enabled to count when ZPNTRCNT is activated. If FILL is active, then the next device is enabled to count every time ZPNTRCNT* is activated. This bypassing of the carry out from the two LSBs, ZADR00 and ZADR01, is done by the F00 and the F32 gates next to the LSB device.

The carry out from the low pointer to the high pointer is generated by two F00 gates because the last device on the low address pointer only has two valid bits.

The outputs of the G address pointer are input to the row/column address multiplexer and also used to decode the bank of RAM being selected in the RAS and CAS generation logic.

- The MSB of the counter is put through a jumper and becomes ZADR20. ZADR20 selects the upper 4 banks (banks 4, 5, 6, or 7) when it is a logic high and the lower four banks (banks 0, 1, 2, or 3) when it is a logic low. The jumper allows ZADR20 to be tied to a logic low if the upper 4 four banks of RAM are not installed.
- The two LSBs of the pointer, ZADR00 and ZADR01, select one of the four banks selected by ZADR20. The two LSBs were chosen to differentiate between the banks so that adjacent address locations would be in different banks. This helps in meeting the RAM address precharge time.

15.4. Graphics Buffer Memory

The G buffer memory array is shown in pages 10 and 11 of the schematics; they consists of 8 banks of dynamic RAM. Each bank is composed of 256K-by-16 bits, implemented with 256 Kbit memory devices, to yield a total memory of 4 Mbytes. Banks 0, 1, 2, and 3 are on page 10 while banks 4, 5, 6, and 7 are on page 11.

Data in and data out of the memory is common to all banks.

- Data in to the memories comes from the G buffer write data register through the series termination resistors and is called ZD15..00.

  Data out from the memory is called ZQ15..00 and goes to the G buffer read data register.

The address, RAS*, CAS*, and the WE* lines to the memory array come from the buffers on pages 8 and 9. The buffers are implemented with ALS244 devices. Output from the buffers go through series termination resistors before they reach the memory devices. Each bank has its own address, WE*, RAS* and CAS* signal. ZAn0..ZAn8 represent the address lines, ZWEn* represent the write enables, RASn* and CASn* represent the control lines—all to the nth bank of RAM. The RAS* and CAS* signals are input to the buffers from the RAS and CAS generation circuit; the address is input from the row/column multiplexer, and the WE* signal is from the G buffer control state machine.

15.5. Graphics Buffer Memory Control

The G buffer memory control is on pages 5, 6, and 7 of the schematic.

The G buffer memory control includes:

- the G Buffer/GP interface signals,
- the G buffer control state machine,
- the RAS and CAS generation,
- the row/column address multiplexer,
- the refresh timer, and
- the refresh address counter.

G buffer access is started by the PP issuing a read or a write through the ZBFSTRTRD* (G buffer start read) signal and the destination signals ZDEST2*..ZDEST0*. Whenever the PP requests an access to the G buffer, it also deactivates the ZBFRDY (G buffer ready) flag on the GP (GP page 30). ZBFRDY being inactive tells the GP software that the G buffer access is not complete.

The PP requesting a G buffer access then activates one of either RD* (read) or WR* (write) signals which is sensed by the G buffer control state machine. The state machine does the required access and then signals the PP by activating STZBUFRDY* (set G buffer ready) which activates the ZBFRDY signal on the GP.

NOTE *Although memory refresh is transparent to the PP, it may cause an access to take longer than normal. Refresh thus makes the time of G buffer access non-constant. To alleviate this inconstancy in the access time of the G buffer, the ZBFRDY signal is used as a handshake between PP and G buffcr memory accesses.*

The G buffer memory has two modes of operation:

1. normal, and
2. read-modify-write (RMW).

In the normal mode the control lines to the memory array—RAS* and CAS*—are left in their inactive state after the access is completed. For RMW mode, the control lines are left in their active state after the access. For example, when a read access is requested, the contents of the memory location are placed into the G buffer read data register and the ZBFRDY flag is activated which completes the read transaction. However, because the control lines are not returned to their inactive state, that same location will continue to be accessed. At some time (within 10 µsec) the PP processor will signal that it wants to write to that same location or that it wants to access the next location.

If the PP wants to access another location, the RMW cycle is completed and another one is started with the new address. If it wants to write that same location then the data is written to the memory by activating the WE* signal but without incurring the delay associated with the strobing of the address into the memory devices. The obvious advantage of course is that writes are done much faster. After the write is completed, the address is incremented and another RMW cycle is started with the new address.

Graphics Buffer/GP Interface Signals

The G buffer mode is determined by the signal NORMAL and its complement NORMAL*. When NORMAL is active the G buffer is in the normal mode. The NORMAL signal is kept in an F109 shown in section B8 of page 5. It is set, reset, or left unchanged as encoded by ZBUS15 and ZBUS14 whenever LDZPNTRH* becomes active. LDZPNTRH* becomes active whenever the G buffer high pointer is being loaded.

The G buffer memory can be operated in *fill mode.* In fill mode, each write to the memory actually writes to four consecutive locations. This allows for the memory to be rapidly cleared or set to a particular value. The value written to the four locations is the contents of the G buffer write data register.

NOTE *Fill mode is only valid for doing writes while in the normal mode. If this condition is not met when a fill mode access is done, the contents of the memory will not be disturbed, however the value of the address pointer is indeterminate after such an access.*

Fill mode is determined by the signal FILL, from an F109 flip-flop in the lower left corner of page 5. This flip-flop is set, reset, or left unchanged as encoded by ZBUS13 and ZBUS12 whenever LDZPNTRH* becomes active.

The FILL signal is used to generate two other signals FILLENH* (fill enable high) and FILLENL* (fill enable low). FILLENL* enables the generation of RAS and CAS for the four lower banks of RAM (banks 0, 1, 2, and 3) simultaneously. FILLENH* enables the generation of RAS and CAS for the four upper banks of RAM (banks 4, 5, 6, and 7) simultaneously. If neither FILLENL* or FILLENH* are active only one of the RAS or CAS lines will be activated for a PP access to the G buffer. FILLENL* and FILLENH* are generated by the F00 and the F74 devices shown in the bottom of page 5. The logic activates one of FILLENL* or FILLENH* whenever a write access is done while being in both the normal and the fill modes. When this condition occurs the MSB of the G buffer address pointer, ZADR20, chooses whether FILLENL* or FILLENH* is activated. If ZADR20 is a logic low, FILLENL* is activated, and when ZADR20 is a logic high, FILLENH* is activated.

The G buffer/GP interface signals are used as the control interface between the G buffer and the GP. When the PP processor issues a request for a G buffer memory access, it is decoded and one of RD* or WR* signals is activated. This is done by a 16R4A PAL device shown on the top left corner of page 5.

The encoded destination signals (ZDEST2*, ZDEST1*, ZDEST0*) and the ZBFSTRTRD* signal are used to determine the type of access requested by the PP. The RD* and WR* signals are input to the G buffer control state machine which accomplishes the access. RD* signal causes the state machine to do a read access while the WR* signal causes the state machine to do a write access.

□ The WR* signal is activated when the ZDEST signals indicate a load of the G buffer write data register (load G buffer). WR* is deactivated when STZBUFRDY* signal becomes active after the access is completed.

- RD* signal is activated when ZBFSTRTRD* is active while the ZDEST signals do not indicate a write to the G buffer write data register (start a read while not starting a write). RD* is deactivated when STZBUFRDY* signal becomes active after the access is completed.

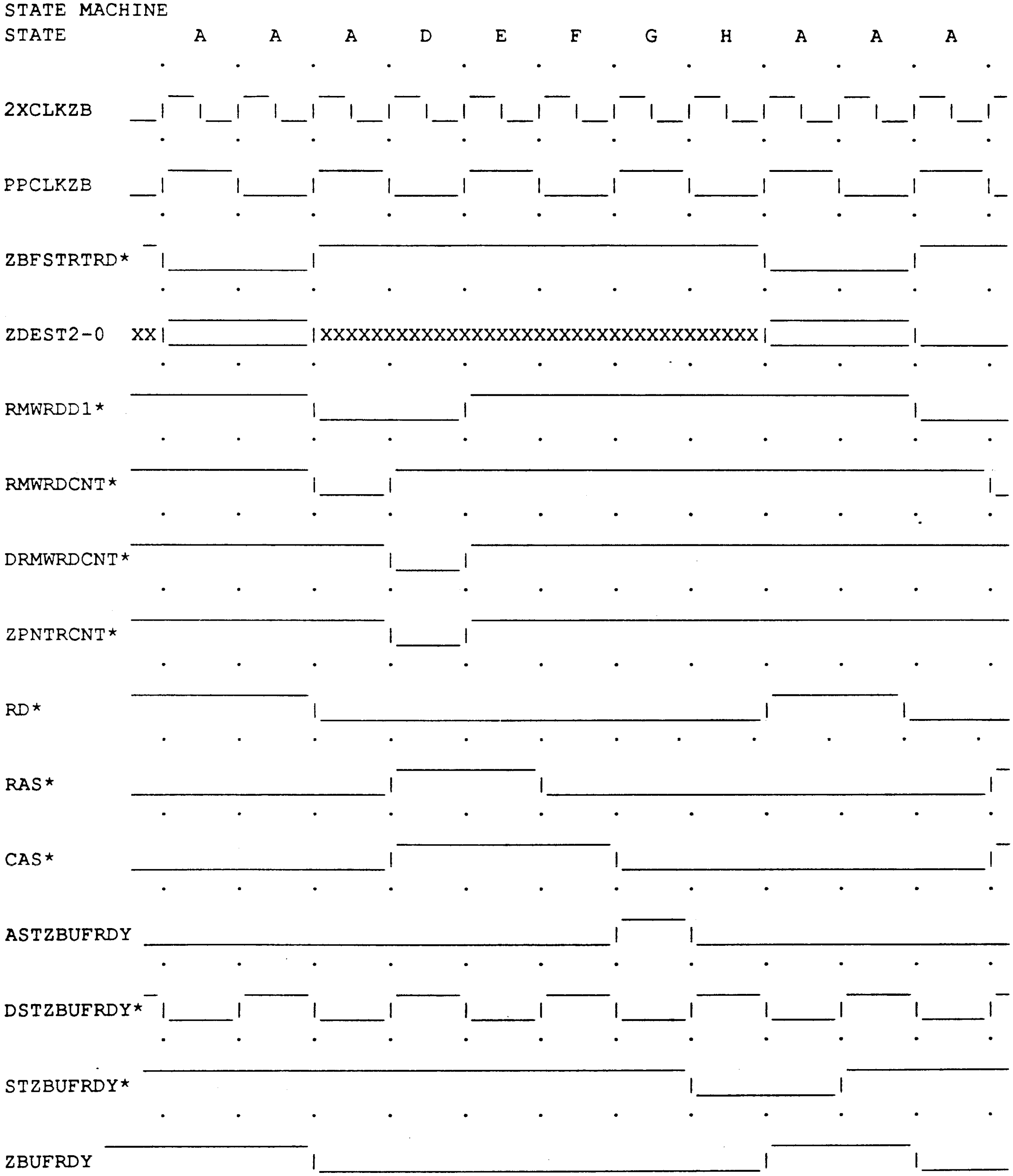
There are three other signals from this PAL, ZPNTRCNT* (G pointer count), RMWRDCNT* (read-modify-write read count), and RMWRDD1* (read-modify-write read delay 1).

- RMWRDD1* is activated when in RMW mode, when ZBFSTRTRD* is active in the second half of the PPCLKZB period, and while the ZDEST signals do not indicate a load of the G buffer low address pointer or a load of the G buffer write data register. It is used to delay incrementing of the G buffer address pointer by one 2XCLKZB period when in RMW mode.
- RMWRDCNT* is used to generate ZPNTRCNT*, which is a combinatorial output from the PAL. RMWRDCNT* is generated whenever the signal input signal SMINCADR* (state machine increment address) is active or when in the RMW mode and RMWRDD1* is active during the first half of the PPCLKZB period. RMWRDCNT* is output from the PAL to an F74 which delays it by one 2XCLKZB period and is input back into the PAL as the signal DRMWRDCNT* (delayed RMW read count). This delay of one period (along with one period delay from RMWRDD1*) is necessary to prevent rampant RAS and CAS pulses from occurring because the address in the G buffer address pointer changes too early.
- ZPNTRCNT* is used to enable the incrementing of the G buffer address pointer. ZPNTRCNT* is a combinatorial output from the PAL and is activated when the signal DRMWRDCNT is active or when in the normal mode and ZBFSTRTRD* is active while the ZDEST signals do not indicate a load of the G buffer low address pointer or the G buffer write data register.

The following timing diagram shows the relationship of RMWRDD1*, RMWRDCNT*, and ZPNTRCNT*.

Figure 15-2 *Graphics Buffer RMW Mode Read Start Timing Diagram*

RMW cyle is ended by a read command (RMW ended without doing a write), and this is followed by another read command (two consecutive reads).



See the chart in the G buffer section of the GP Hardware Reference manual for a better understanding of the above.

The G buffer/GP interface signals PAL also has another output from it called MCHANGE (mode change). MCHANGE* is activated whenever the high portion of the G buffer address pointer is loaded when in the normal mode. When loading the high portion of the address pointer (both in the normal and the RMW modes,, there is the possibility that the mode is changed from normal to RMW or vice-versa. The G buffer control state machine must know when the mode is changed from the normal to RMW and this signal provides that information. MCHANGE* is deactivated by the first active ZBFSTRTRD* signal after the high address pointer is loaded. The discussion below on refresh explains the use of the MCHANGE* signal.

Graphics Buffer Control State Machine

The G buffer control state machine is the central control element of the G buffer memory. It is implemented in a 16R6A PAL shown in section D5 of page 5. The 16R4A PAL in section D3 is called G buffer control signals and is an extension of the state machine in that it holds some of the outputs which did not fit in the other PAL. The state diagram for the G buffer control state machine is shown in Figure 23, contained in the appendix.

The following outputs from the state machines enable the various operations to occur when accessing the G buffer memory.

- RAS* and CAS* are output to the RAS and CAS generation logic on page 6 and are used in the generation of RAS and CAS to the memory array.
- WE* (write enable) is output directly to the memory buffers on pages 8 and 9 and becomes the write enable into the memory array.
- SMINCADR* (state machine increment address) is input to the G buffer/GP interface signals PAL and is used to activate ZPNTRCNT* signal which enables the G buffer address pointer to increment. SMINCADR* is activated by the state machine after each write into the memory (whether in the normal or the RMW mode), which facilitates sequential writes without software having to increment the address.
- LDDOUT* (load data out) is used to clock in the data from the memory array into the G buffer read data register (after it is delayed and becomes LDRDREG* signal).
- REFREQ* (refresh request) is implemented as an S-R flip-flop in the PAL. REFREQ* indicates to the state machine that the refresh timer has reached its terminal count and a refresh should be done. REFREQ* is activated by the signal RTIMERCO* (refresh timer carry out) becoming active and is reset when the REFRESH* signal becomes active.
- REFRESH* is activated by the state machine when it is actually doing a refresh access cycle to the memory array.
- Another output from the state machine is the signal STZBUFRDY* (set G buffer ready). STZBUFRDY* is activated by the state machine when it completes the access requested by the PP. It activates the signal ZBFRDY

on the GP. STZBUFRDY* is output from the F109 device in section D1 of page 5. There are two signals from the G buffer control signals PAL which activate and deactivate STZBUFRDY*, ASTZBUFRDY (activate STZBUFRDY*) and DSTZBUFRDY* (deactivate STZBUFRDY*), respectively. ASTZBUFRDY is activated when the state machine is in state G and the operation being done is not a refresh. DSTZBUFRDY* is activated every time DPPCLKZB is a logic low.

□ DPPCLKZB is just the signal PPCLKZB delayed by one 2XCLKZB period. This allows for STZBUFRDY* to stay active during one—and exactly one—rising edge on PPCLKZB (even when PPCLKZB is halted).

The state machine has 9 states labelled A through I. The state machine operation, for the sake of discussion, can be separated into three operating modes: normal mode, RMW mode (not normal), and refresh. Upon power up or reset the active RST* signal into the state machine causes it to go to state A. State A is labelled as the idle state because if there are no refresh requests or no requests from the PP, the state machine stays in state A.

When in the normal mode of operation, the state machine stays in state A waiting for an access request indicated by RD* or WR* going active. While in state A waiting for RD* or WR* activate, none of the outputs are activated. The figure below shows the timing diagrams for a read, write, and a read or write access with PPCLKZB halted for the normal mode of operation.

Figure 15-3 *Graphics Buffer Normal Mode Read Timing Diagram*

(Two consecutive read operations)

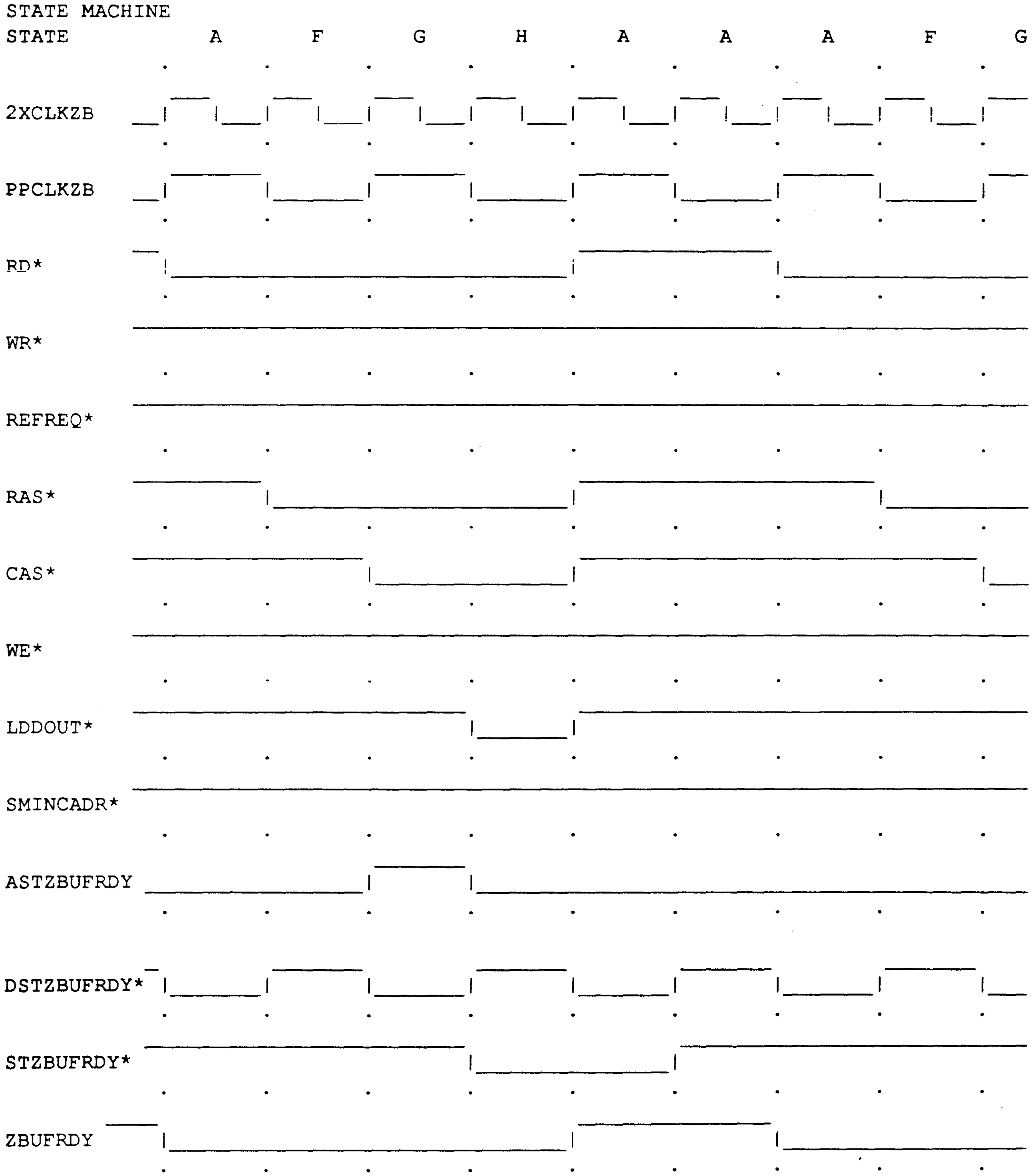
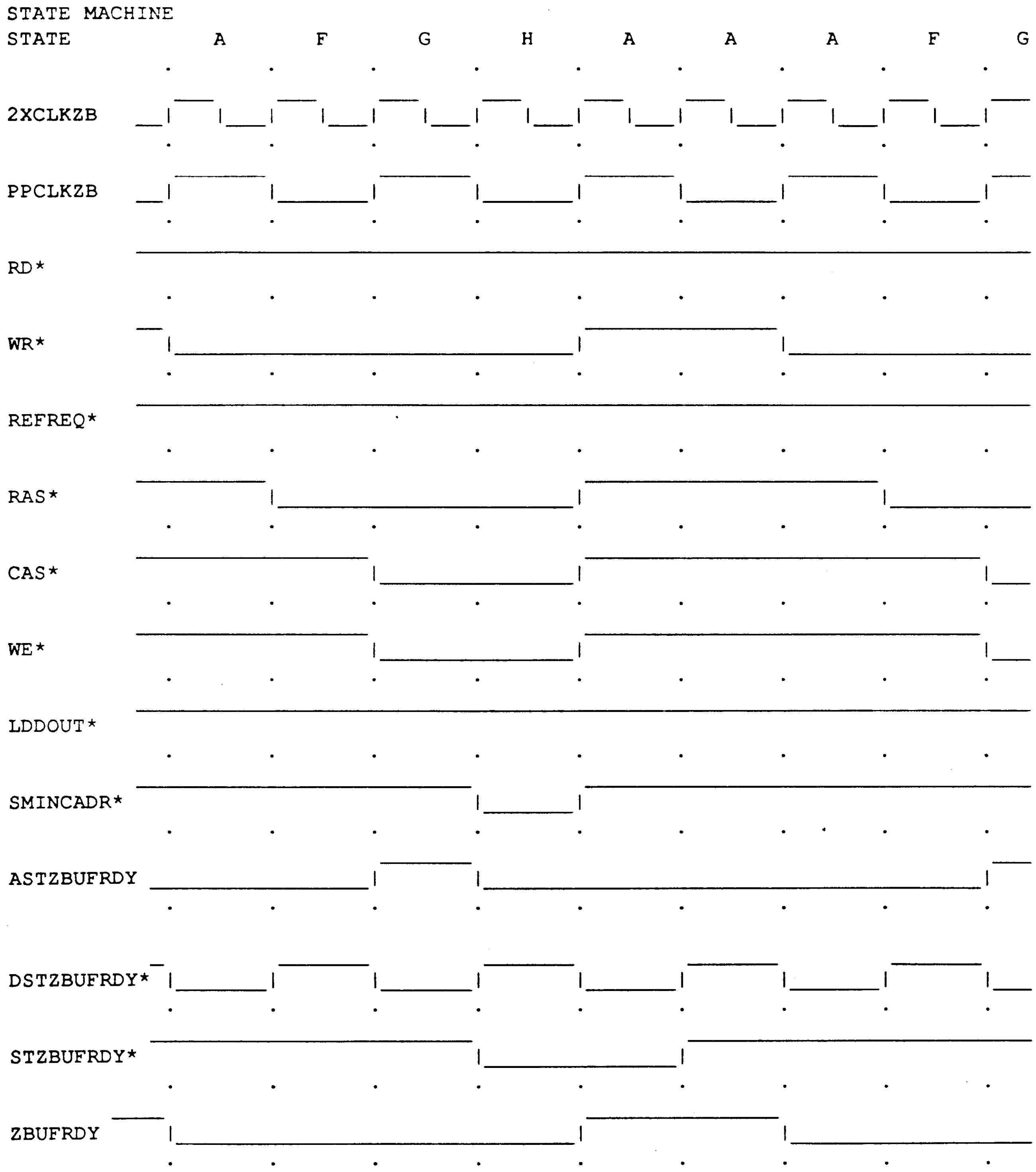


Figure 15-4 *Graphics Buffer Normal Mode Write Timing Diagram*

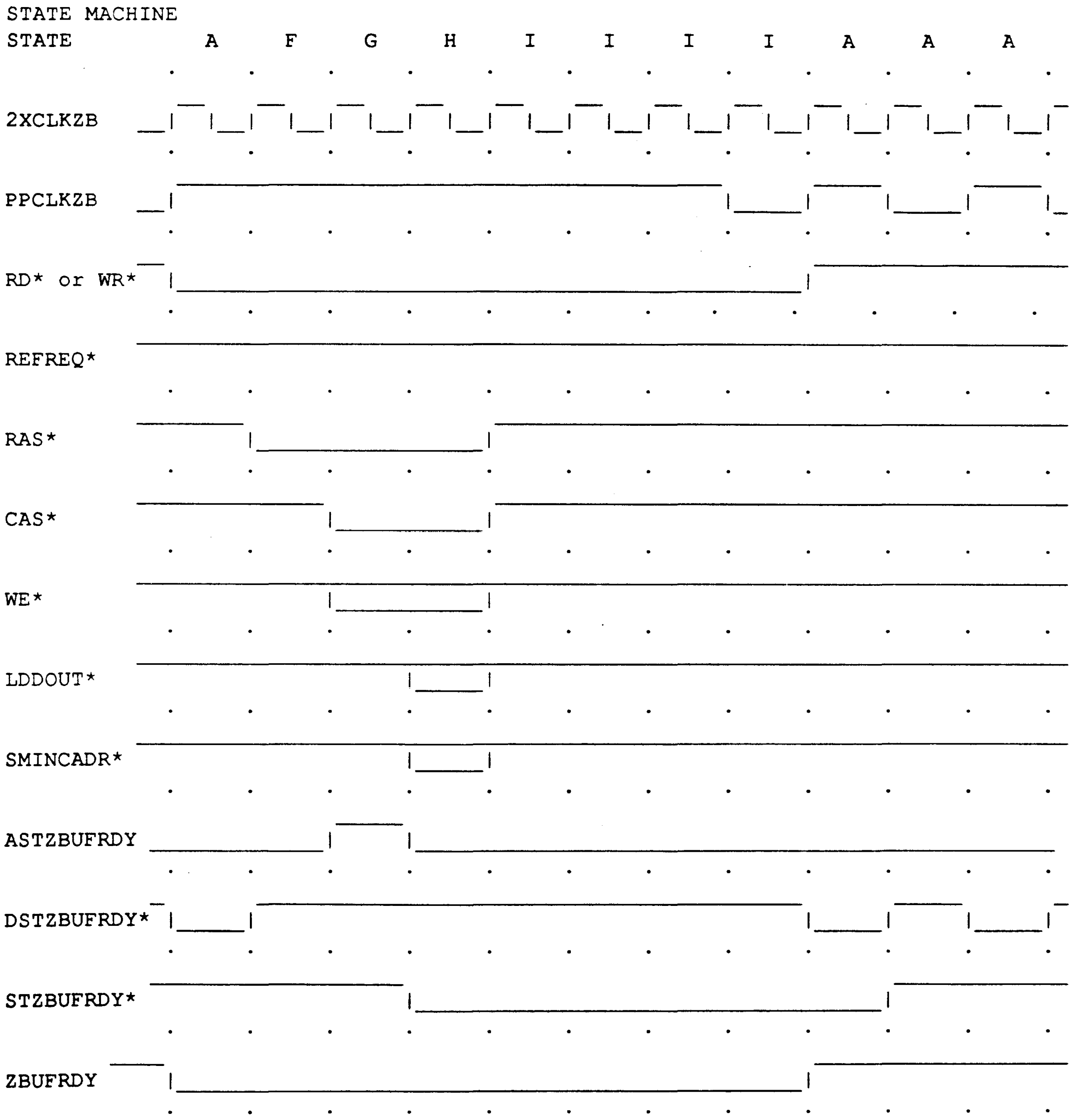
(Two consecutive write operations)



SUN PROPRIETARY

Figure 15-5 *Graphics Buffer Normal Mode Read or Write Timing Diagram*

(PPCLKZB HALTED DURING ACCESS)



When the RD* becomes active (while in state A) the state machine goes to stat F, then G, H, then back to state A. RAS* is activated between states A and F, both RAS* and CAS* are activated between states F and G, and RAS*, CAS*, LDDOUT*, and ASTZBUFRDY are activated between states G and H. From state H the state machine (in normal circumstances where PPCLKZB is not halted) goes back to state A with none of the outputs activated.

If the WR* signal becomes active when in state A, the state machine goes to state F, G, and H, then back to state A (the same path followed as the read access). From state H the state machine, in normal circumstances where PPCLKZB is not halted, goes back to state A with none of the outputs activated.

For both read and write accesses in normal mode, there is the possibility that PPCLKZB will be halted while an access is being done. If so, PPCLKZB will be halted when the state machine is between states A and H. PPCLKZB is known to be halted if it is not a logic low when the state machine gets to state H; if such is the case (PPCLKZB is halted), the state machine will go to state I. It will stay in state I until PPCLKZB is started again (indicated by a logic low on PPCLKZB), then go to state A.

Once in state I, it is possible that a refresh request might occur. In this case the state machine will execute a refresh cycle (by traversing states D, E, F, G, and H) then proceed to state I again. Notice that state I will only be reached when in the normal mode because PPCLKZB should only be halted in the normal mode (in other words, PPCLKZB can not be halted in the RMW mode).

For the RMW mode, the state machine stays in state A waiting for either RD* or WR* to become active. Two conditions may occur while the state machine is waiting for RD* or WR* in state A—RAS* and CAS* will be activated or they will not be activated. The G buffer is reset to normal mode and can only be set to the RMW mode by the PP software. When the PP software changes the mode of the G buffer from normal to RMW, the state machine will be in state A.

When the mode is initially switched to RMW, the state machine will stay in state A and RAS and CAS will not be asserted. From this condition the software will issue a read access request which will start the first RMW access. Other than this initial switch to the RMW mode, the state machine, when in state A, is in the middle of the RMW access (after the read is done) and therefore both RAS* and CAS* will be activated while waiting for RD* or WR* to become active. When RD* becomes active in state A, the current RMW cycle will be completed (without the write being done) and another RMW cycle will be started by the state machine which will proceed to state A after the read portion of this latest cycle is done. When WR* becomes active in state A, the write corresponding to the current RMW cycle will be done and another RMW cycle will be started by the state machine which will proceed to state A after the read portion of that cycle is done.

The two figures below show the timing diagrams for read and write accesses while in the RMW mode of operation. When the RD* signal becomes active in state A, the state machine ends the current RMW cycle and starts another RMW access by going to state D and deasserting RAS* and CAS*. From state D, the state machine traverses states E, F, G, H, and then back to A, in sequence. This accomplishes the read portion of the RMW cycle. The outputs that are activated while going from state to state are:

- RAS* between E and F,
- RAS* and CAS* between F and G,
- RAS*, CAS*, LDDOUT*, and ASTZBUFRDY between G and H, and
- RAS* and CAS* between H and A.

Figure 15-6 *Graphics Buffer RMW Mode Read Timing Diagram*

RMW cyle is ended by a read command (RMW ended without doing a write), and this is followed by another read command (two consecutive reads).

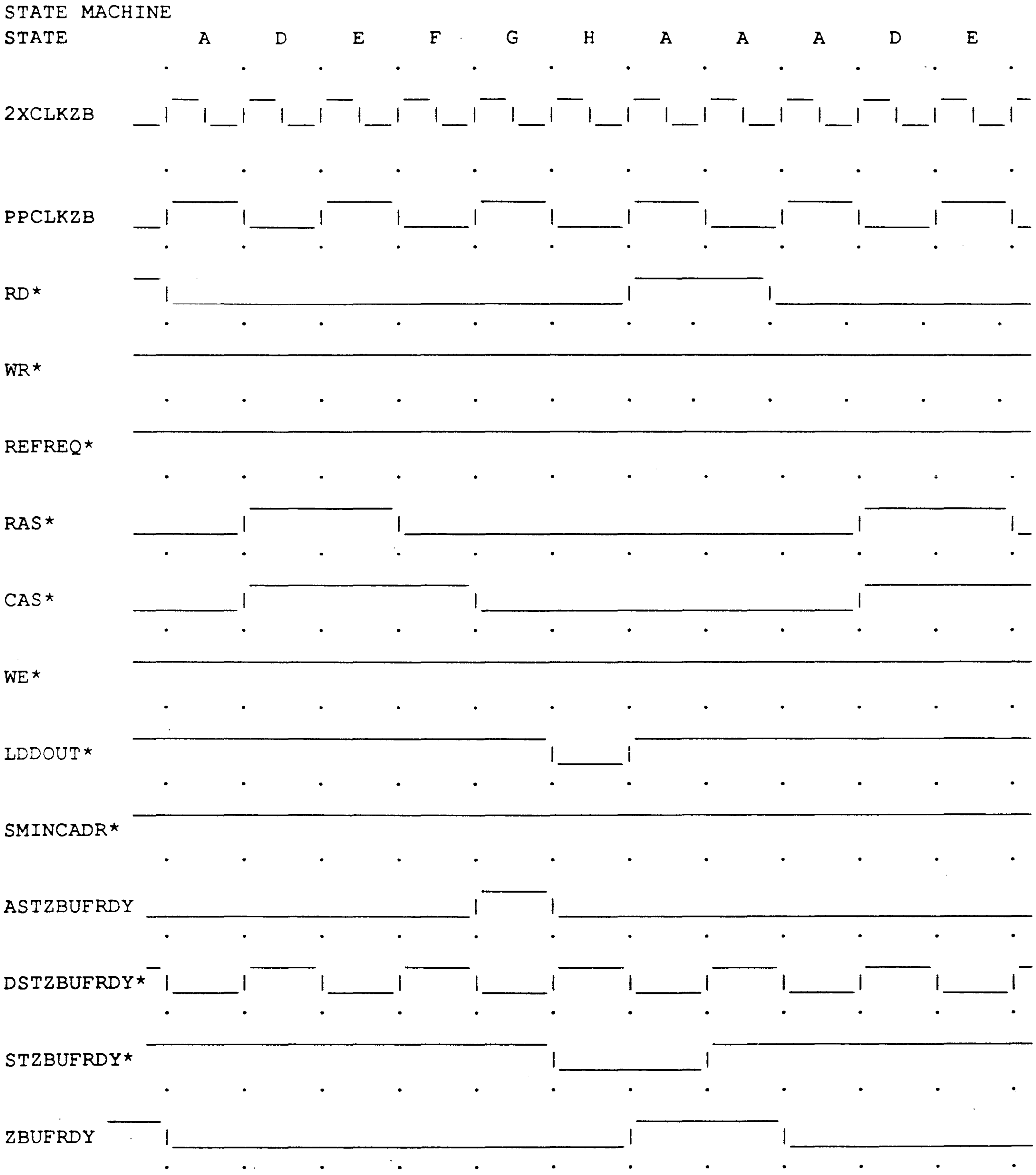
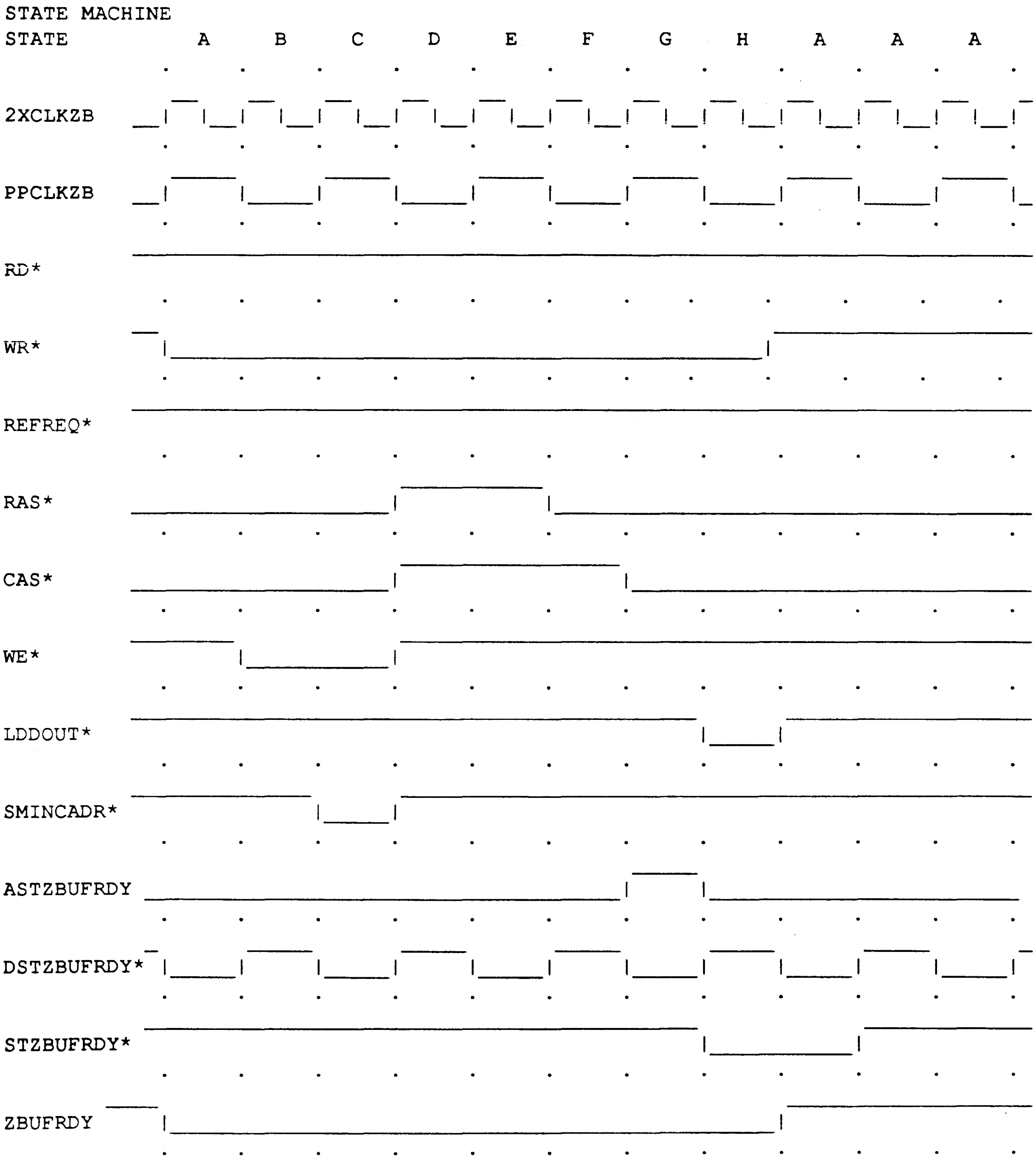


Figure 15-7 *Graphics Buffer RMW Mode Write Timing Diagram*

RMW cyle is ended by a write command and is followed by the read access of the next RMW cycle.



When WR* becomes active in state A, the state machine will begin the write poı tion of the RMW cycle by going to state B with RAS*, CAS*, and WE* active. The state machine will then go to state C with RAS*, CAS* and WE* asserted. From state C the state machine will go to state D with none of the outputs activated which effectively ends the RMW cycle. From state D, the cycle is identical to the read access described above, traversing states E, F, G, H, and A with the outputs being asserted also identical to what is stated above. So it can be said that the read portion of the RMW cycle is done by traversing states D, E, F, G, H, and getting back to state A while the write portion of the RMW is done by traversing states B and C.

Refresh on the G buffer memory array is started by the G buffer control state machine. Normal mode refresh is started differently than the RMW mode. However, although the refresh period is *started* differently, the basic refresh cycle (activation of RAS) is the *same* for all cases. The state machine senses that a refresh is necessary when the signal REFREQ* becomes active. When the state machine is doing a refresh cycle, it activates the REFRESH* signal. The two figures below show the timing diagrams for normal and RMW refresh cycles.

Figure 15-8 *Graphics Buffer Normal Mode Refresh Timing Diagram*

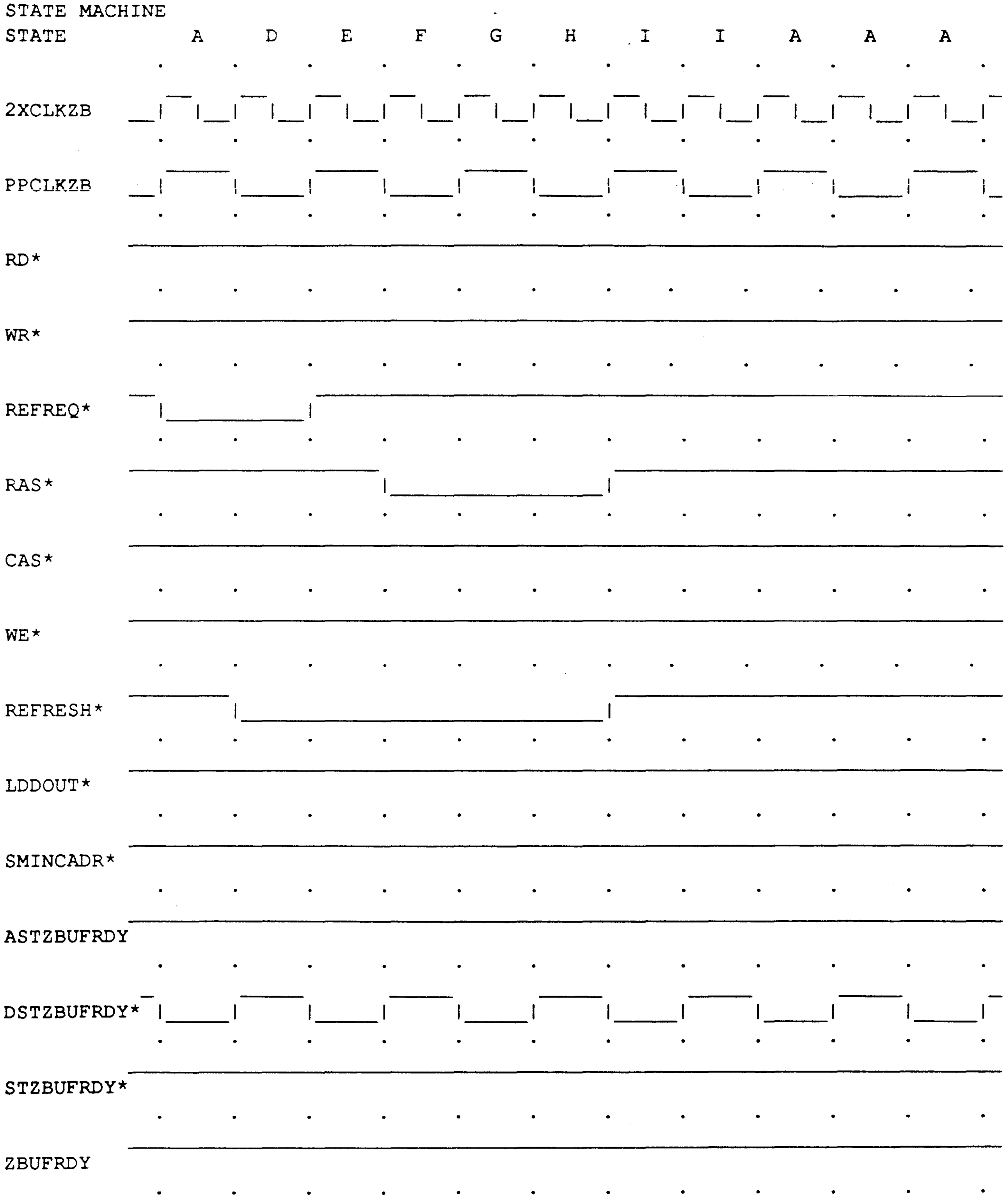
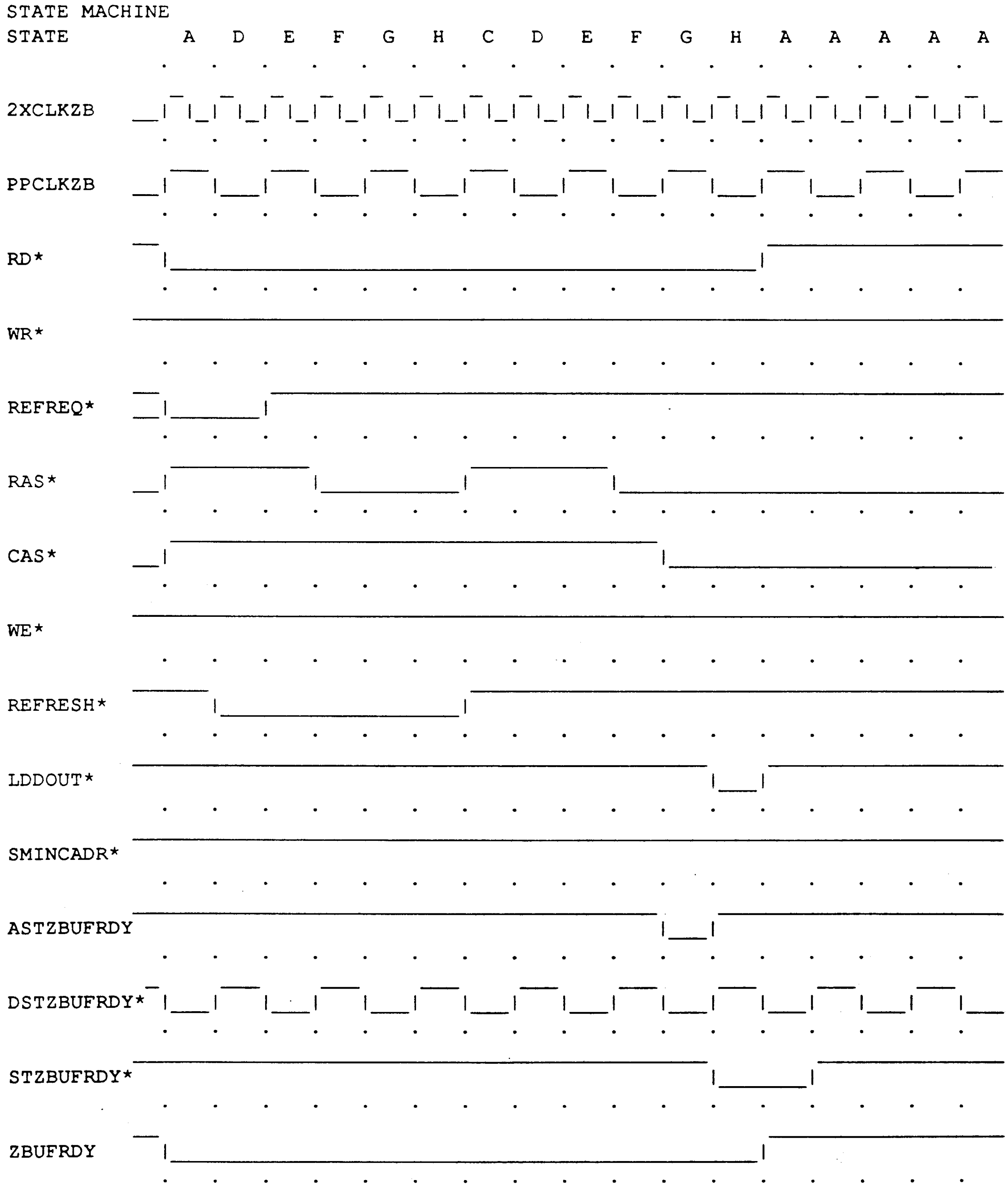


Figure 15-9 *Graphics Buffer RMW Mode Refresh Timing Diagram*



In normal mode, REFREQ* is sensed when in state A. If REFREQ* occurs when in state A, the state machine goes to state D while asserting REFRESH*.

NOTE *If REFREQ* occurs simultaneously with a RD* or WR*, then the refresh cycle will be done before the read or write access.*

REFRESH* deactivates REFREQ* and also signals to other G buffer logic that a refresh operation is being done (as opposed to a regular read or write access). From state D the state machine goes in sequence to states E, F, G, and H while REFRESH* is continuously asserted. RAS* is also activated between states E and F, states F and G, and states G and H. From state H, the state machine goes to state I. It will stay in state I for one more cycle and then return to state A if PPCLKZB is not halted. If PPCLKZB *is* halted, the state machine will stay in state I until PPCLKZB is started again or another REFREQ* occurs. If PPCLKZB is started, the state machine will return to state A.

If REFREQ* becomes active in state I, the state machine starts a refresh cycle by going to state D and asserting REFRESH. From state D the state machine follows the same path as a regular refresh described above. In fact, it can be stated that the refresh cycle is started by going to state D while REFRESH* is activated (from states A, C, and I) and the refresh cycle is done by traversing states D, E, F, G, and H with REFRESH* activated.

One final case: if mode is changed to RMW while a refresh operation is occurring in normal mode, the signal MCHANGE* will become active. This situation is sensed in state H when the refresh cycle is completing (after traversing states D, E, F, and G). If REFRESH is active in state H, MCHANGE* is active, and NORMAL* is not active, the state machine recognizes that the mode has been changed to RMW and goes to state A with none of the outputs activated. This was necessary to guard against RAS or CAS signals being inappropriately activated to a bank of RAM when changing from normal to RMW modes (as a new address was loaded in to the address ponter). Note that MCHANGE* will be deactivated by the first read command (ZBFSTRTRD*) of the RMW mode so that this case only occurs when changing modes.

When in RMW mode, the refresh is done in between the RMW cycles (that is, after one cycle ends and the next one is started). In particular, this means that refresh is started when going from state A to D when a RD* causes the RMW cycle to be ended without doing the write, or when going from state C to D, when a write operation has been done to complete the RMW cycle. In either case the refresh is done, the read portion of the next RMW cycle is done, and the state machine is returned to state A.

The refresh cycle is identical to the refresh cycle described above for normal mode, traversing states D, E, F, G, and H while activating the outputs the same as normal mode. However, when state H is reached after the refresh cycle is completed, the state machine deactivates REFRESH* and goes to state C (provided that MCHANGE* is not active, indicating the mode has been changed from normal to RMW). From state C the read portion of the next RMW cycle is done by traversing the states D, E, F, G, and H and then back to state A. Thus, for the RMW mode, the time between two RMW cycles may be prolonged by a refresh cycle.

RAS and CAS Generation

The RAS and CAS generation circuit is shown on the left of page 6. This logic selects the bank(s) of RAM that is (are) being accessed by activating the RAS and CAS signals to the particular bank(s) of RAM. In addition to selecting the particular bank(s) of RAM, RAS* and CAS* output pulse shapes are determined by this logic. There are eight RAS and eight CAS signals—one for each bank of RAM. The RAS output signals are named RASBANK0* to RASBANK7* and the CAS output signals are named CASBANK0* to CASBANK7*. The outputs from the RAS and CAS generation logic are input to the RAM buffers on pages 8 and 9 which output them directly to the corresponding bank of RAM. The figure below shows the RAS and CAS access timing of the RAM array.

Figure 15-10 *Graphics Buffer RAS/CAS Access Detailed Timing Diagram*

```
STATE MACHINE
STATE              A            F            G            H            A

          _______       _______       _______       _______       _______
2XCLKZB _|       |______|      |______|      |______|      |______|      |______|
         |              |                                         |

          ________________            ________________            ______________
PPCLKZB _|_|            |_|___________|_|           |_|___________|_|
         |              |                                         |

        ________                                                      ___________
WR*             |_____________________________________________________|
         |              |                                         |
FILLENH*     _____________                                                      _
cr FILLENL*               |____________________________________________________|
         |<-29->|       |<-------79.6----->|                      |
         |       |      |<---49--->|       |                      |<-26->|

       ___      __________________         ______________________________
ZAxx      |||||| VALID ROW ADDRESS||||||||| VALID COLUMN ADDRESS         ||||||||
       ___||||||__________________|||||||||______________________________||||||||
                        |<--40-->|                                |<---40-->|
                        |16|     |                                |16|      |

          _______________________                                    ____________
RASx*                      |_____|___________________________________|______|
                        |<----57.6--->|                           |<----46--->|
                        |<--42.5->|   |                           |<19>|      |

          ____________________________                                ___________
ROW/COL                           |___|_______________________________|_______|
                        |<----------117.5---------->|             |<----58----->|
                        |<--------92.5-------->|    |             |<--32->|     |

          __________________________________________                     ________
CASx*                                         |_____|____________________|______|
                        |<-------85------->|                      |<25>|
                        |<----70.5----->|  |                      |10| |

          __________________________________                         ____________
ZWEx*                                   |___|________________________|_|
                        |<--------------------202.5--------------------->|
                        |<------------------177.5------------------>|     |

          _________________________________________________________      ________
ZQxx                                                               ||||||| VALID
          _________________________________________________________|||||||_______
                        |<------------------------219.5------------------------>|
                        |<----------------------206.5---------------------->|   |

          _________________________________________________               _______
READ REG*                                                |_|______________|___|
CLOCK

           ___________________________________________                          _
STZBUFRDY*                                            |________________________|

        _____                                                      ______________
ZBUFRDY      |_____________________________________________________|
```

SUN PROPRIETARY

The pulse shape of the RAS outputs RASBANK0* to RASBANK7* are determined by the RAS* signal from the G buffer control state machine. There are three cases to be considered when generating the RAS signals, regular accesses (normal or RMW), fill mode accesses, and refresh accesses. The eight F11 gates on page 6 enable whichever of the three cases is active to drive the RASBANK* output signals.

- For regular accesses, only one of the banks of RAM is activated at a time as selected by the ZADR20, ZADR00, and ZADR01 signals through an F138 device. The F138 outputs are enabled by RAS* from the state machine.
- For the fill mode accesses, four banks of RAM are activated at the same time. One of the signals FILLENH* or FILLENL* will be active for this mode to select the upper four banks or the lower four banks of RAM, respectively. The RAS* signal is ANDed with the FILLEN* signals to generate the RASBANK* outputs for this case.
- For the refresh accesses, all eight banks of RAM will have their RAS signals activated. The RAS* signal is ANDed with the REFRESH* signal to activate the RASBANK* outputs for all the banks. REFRESH* is passed through a 17 ns delay before going through the F32 gate (section A6-A7 of page 6) to guard against a possible race condition—RAS* deactivated at the same time REFRESH* is activated.

The pulse shape of the CAS outputs CASBANK0* to CASBANK7* are determined by the CAS* signal from the G buffer control state machine after it is delayed 17 ns for timing purposes. There are two cases to be considered when generating the CAS signals, regular accesses and fill mode accesses. The eight F08 gates enable whichever of the two cases that is active to drive the CASBANK* output signals.

- For regular accesses, only one of the banks of RAM is activated at a time as selected by the ZADR20, ZADR00, and ZADR01 signals through an F138 device. The F138 outputs are enabled by the delayed version of the CAS* signal.
- For the fill mode accesses, four banks of RAM are activated at the same time. One of the signals FILLENH* or FILLENL* will be active for this mode to select the upper four banks or the lower four banks of RAM, respectively. The delayed version of the CAS* signal is ANDed with the FILLEN* signals to generate the RASBANK* outputs for this case.

Row/Column Address Multiplexer

The row/column address multiplexer is necessary to input the row and the column addresses into the RAM devices which correspond to the RAS and the CAS signals. It is implemented with F153 devices shown on the right of page 7. The F153s multiplex onto the RC8 to RC0 (row/column 8 through 0) signal lines, the row and column addresses from the G buffer address pointer and also the refresh row address from the refresh address counter. When REFRESH* signal is active the refresh address counter is selected and put on the RC signal lines. If REFRESH* is not active, then the ROW/COL (row/column) signal determines which of the G buffer address pointer bits get routed onto the RC sig nal lines. When ROW/COL is a logic high, it corresponds to an active RAS signal or the row addresses being presented to the RAM devices. The ROW/COL signal is generated by the RAS* signal being input to a chain of delay lines at the bottom of page 6.

The RAS* signal is delayed by 34 ns (typical) and then put through an F32 gate to generate the ROW/COL signal. The F32 gate allows the ROW/COL signal to become a logic high as soon as RAS* becomes deactivated. Thus, ROW/COL is a logic high for 34 ns after RAS* goes active and then becomes a logic low until RAS* becomes inactive.

Refresh Timer and Refresh Address Counter

The refresh counter and the refresh address counter are shown on the bottom right of page 5. The refresh timer is used to signal the G buffer control state machine when a refresh on the RAM array is necessary. The refresh address counter keeps track of the RAM row addresses that should be refreshed. When the refresh counter signals the G buffer control state machine to do a refresh, the state machine, when appropriate, starts the refresh operation by activating REFRESH* signal and the contents of the refresh address counter are presented to the RAM while RAS is also activated to refresh the particular row. A refresh is requested approximately every 15 µsec. The refresh is done by a RAS-only refresh cycle on the RAM array.

The refresh timer is implemented as an 8-bit up counter with F163 devices. The counter is loaded with a constant value and counts up from that value until it gets a carry out of the MSB. This carry out is the signal RTIMERCO (refresh timer carry out) and its complement, RTIMERCO*. RTIMERCO* is input to the G buffer control signal PAL where it activates the REFREQ signal. RTIMERCO* also goes to the load enable of the refresh timer so that the constant value is loaded back into the counter. In addition, RTIMERCO is input to the count enable of the refresh address counter, and every time it is active it enables the refresh address counter to increment. Notice that since REFREQ is activated the refresh timer can be loaded to count the next refresh period without waiting to ensure that the current refresh request is carried out. The G buffer control state machine is guaranteed to do the refresh before the refresh counter counts down.

The value loaded into the refresh counter is the binary "1000xxxx" where xxxx represents a variable as set by the jumpers to the left of the counter. The value loaded into the counter is therefore between "10000000" and "10001111". When FRCLK has a period of 120 ns, this translates to a refresh request between 13.4 µsec and 15.2 µsec. The jumper was put in to provide some flexibility in specifying the refresh request period. The jumpers are hardwired, however, to pull-downs so that no shunts are necessary on them and the value loaded into them is "1000 0000" which corresponds to a refresh period of 15.2 µsec.

The refresh address counter is just an 8-bit up counter implemented with F163 devices. The contents of this counter represents the row of the RAM devices that is being refreshed. The eight bits are just enough for 256K RAM devices which have 256 rows. The refresh address counter is incremented every time that the RTIMERCO signal becomes active. When it reaches its terminal count (255), it rolls over to 0 and starts counting up from there. The output from the counter is the eight bits RA7 to RA0 (row address 7 to 0) which is presented to the row/column address multiplexer.

15.6. Integer Multiplier

The integer multiplier consists of a 16-by-16 multiplier, associated registers and buffers, and the PP PROM.

The integer multiplier used is the Am29517 (or equivalent) on page 12. Inputs include the 16-bit X operand, the 16-bit Y operand, mode bits, clock, clock enables, and output control bits. The output is the 32-bit result, read as two 16-bit words.

The mode register, the ALS374 at D6 of page 12, contains the mode bits which are routed to the multiplier. ZBUS00 to ZBUS04 are loaded into this register by the signal LDMMREG*. The mode bits are:

Table 15-1 *Mode Register Bits*

Bit	Meaning
0	round control
1	format adjust
2	Y mode control
3	X mode control
4	MPSCN—MPSEL* control (see below)

The X and Y operands are latched in ALS374s and routed to the Am29517. The reason for this buffering is that the PPBUS/ZBUS cannot meet the Am29517 set-up times. The ZBUS bits are loaded into the X register with the control LDMXOP* and into the Y operand with the control signal LDMYOP*.

The Am29517 is configured in flow-through mode because ENX and ENY are pulled down. This means that after the X and Y operands are valid, the multiply result is valid some fixed time later. This time is dependent on the type of integer multiplier used and the microcoder must delay the proper number of cycles to guarantee valid results. The minimum delay is one cycle (load X, load Y, delay, read result, read result) for a bipolar multiplier and three delays for a CMOS multiplier.

Since the multiplier has a 16-bit output bus, two reads are necessary to fetch the 32-bit result. If MPSEL* (input to the multiplier from the F74 at A5) is low, the most significant half is read, and if MPSEL* is high, the least significant half is read. The multiplier output is routed through the ALS244s (needed to meet current drive requirements and timing requirements due to the slow MPSEL*-to-output valid time) and onto the ZBUS when enabled by RDMP*.

Rather than have two PPBUS sources to read the most and least significant halves, the integer multiplier has only one PPBUS source—selected by RDMP*. The flip-flop at A5 (MPSEL*) toggles on each read of the multiplier result to enable the reading of the other half. Each time either the X or Y operand is loaded, the output of the OR gate at A6 is asserted. This line will asynchronously set or reset MPSEL* based on the MPSCN mode bit, thus selecting either the most or least significant half on the next integer multiplier read. At the end of the read, RDMP* (ANDed with PPCLKZB to prevent glitching) will toggle MPSEL* enabling the other half on the next read command.

A typical procedure is described below. It assumes that MPSCN=0 and the X operand is loaded first, although X and Y can be loaded in either order.

1. Load the X operand — (LDMXOP* causes MPSEL* to be set to 0);
2. Load the Y operand — (LDXYOP* causes MPSEL* to be set to 0 again);
3. Delay — depends on the flow-through time of the multiplier;
4. Read the multiplier result — the most significant half is enabled, and after the read the trailing edge of RDMP* will toggle MPSEL*;
5. Read the multiplier result — the least significant half is enabled, and after the read the trailing edge of RDMP* will toggle MPSEL*.

The same multiplier result can be read as many times as desired as long as the X and Y operand registers are not altered. (The delay, of course, only needs to be accounted for once.) The reads will continue to toggle between the two halves.

The Painting Processor PROM (an identical configuration exists as the VP PROM on the GP board) is shown on page 13. The PP PROM pointer is loaded when chosen as the PPBUS/ZBUS destination. LDPPPROM* strobes the ZBUS into the F374s which hold the PROM pointer. The output of this register is routed to the PROM, and after an access time delay (actual time depends on type of PROM used; the microcoder must account for this time by delaying the proper number of cycles before choosing the PP PROM as the PPBUS/ZBUS source), the PROM data word is valid and routed to the ALS244 buffers (these buffers are needed because of the long chip enable time of MOS PROMS). This data word is routed onto the ZBUS when enabled by RDPPPM*.

On the Graphics Buffer board, two 28-pin sockets are provided for the PROMs. Currently 27128 types of devices are being used to provide 16K words.

A

State Diagrams, PAL listings, and Schematics

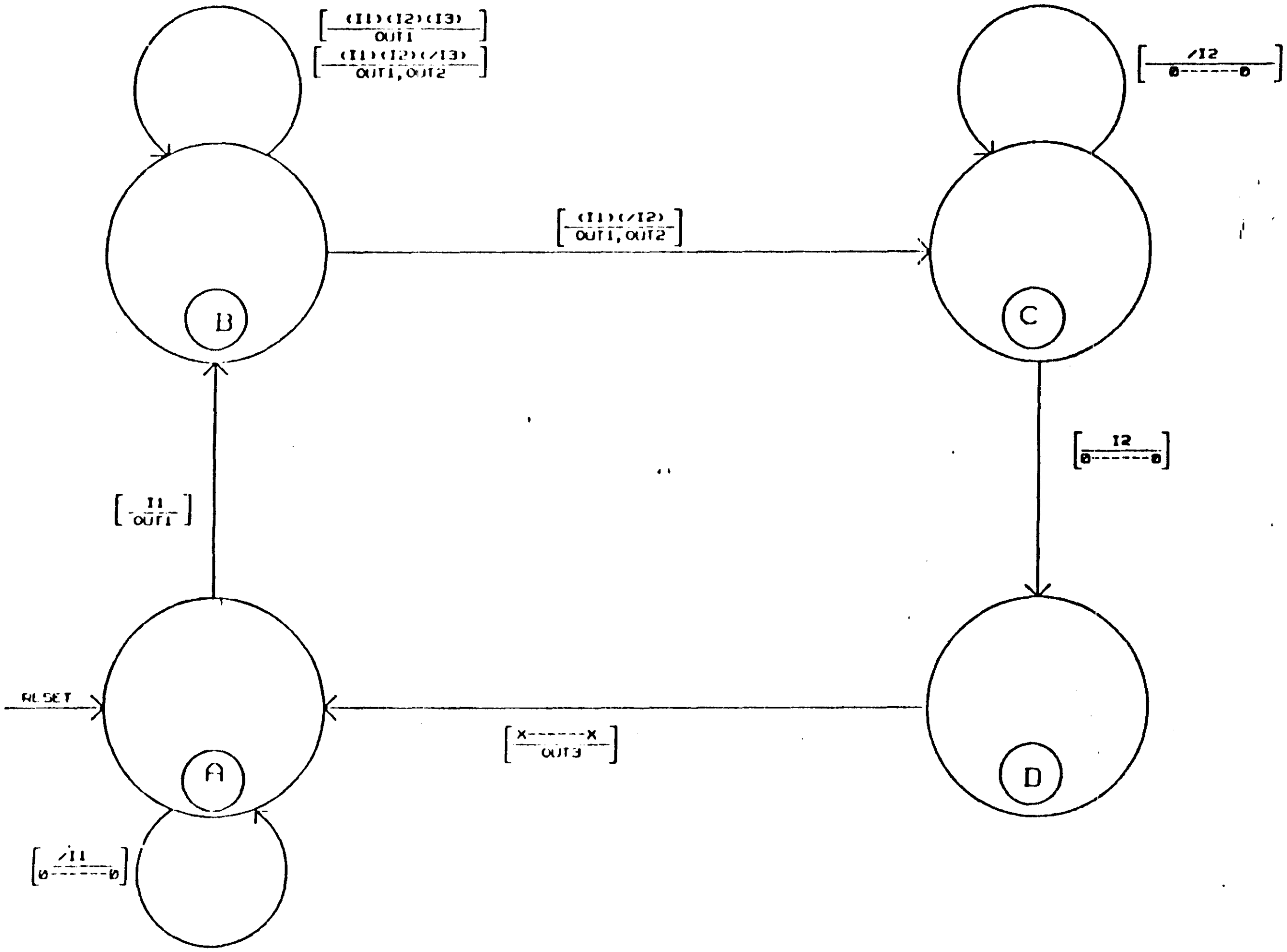
This section contains the state diagrams referred to in this engineering manual.

Since PAL listings and schematics are fairly volatile, they have been ''unbundled'' from this manual. They may be obtained separately through Document Control.

Index

Revision History

Revision	Date	Comments
50	1 July 1985	First release of this Engineering Manual.
A-01	2 February 1986	Added PAL listings and schematics to appendix, made various small revisions.
1-02	1 of 20 October 1986	Changed revision number to comply with Doc Control's new numbering scheme.



STATE ASSIGNMENT		
STATE	FF1	FF0
A	1	0
B	1	1
C	0	1
D	0	0

APPENDIX A
FIGURE 1

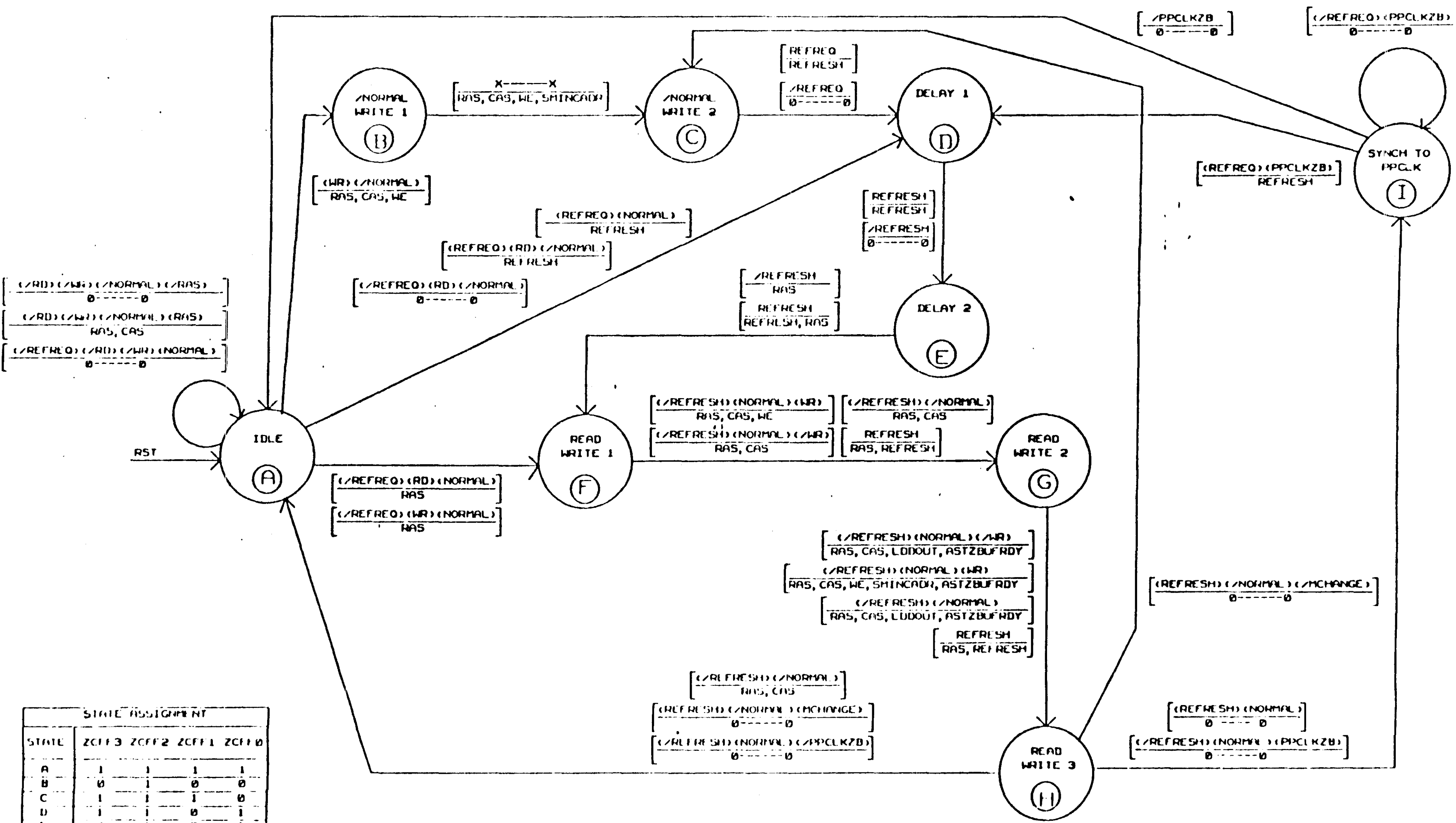
TITLE:
STATE MACHINE EXAMPLE

DATE:
JANUARY 198

ENGINEER:
SERDAR ERGENE

PAGE:
1 OF 1

SUN PROPRIETARY

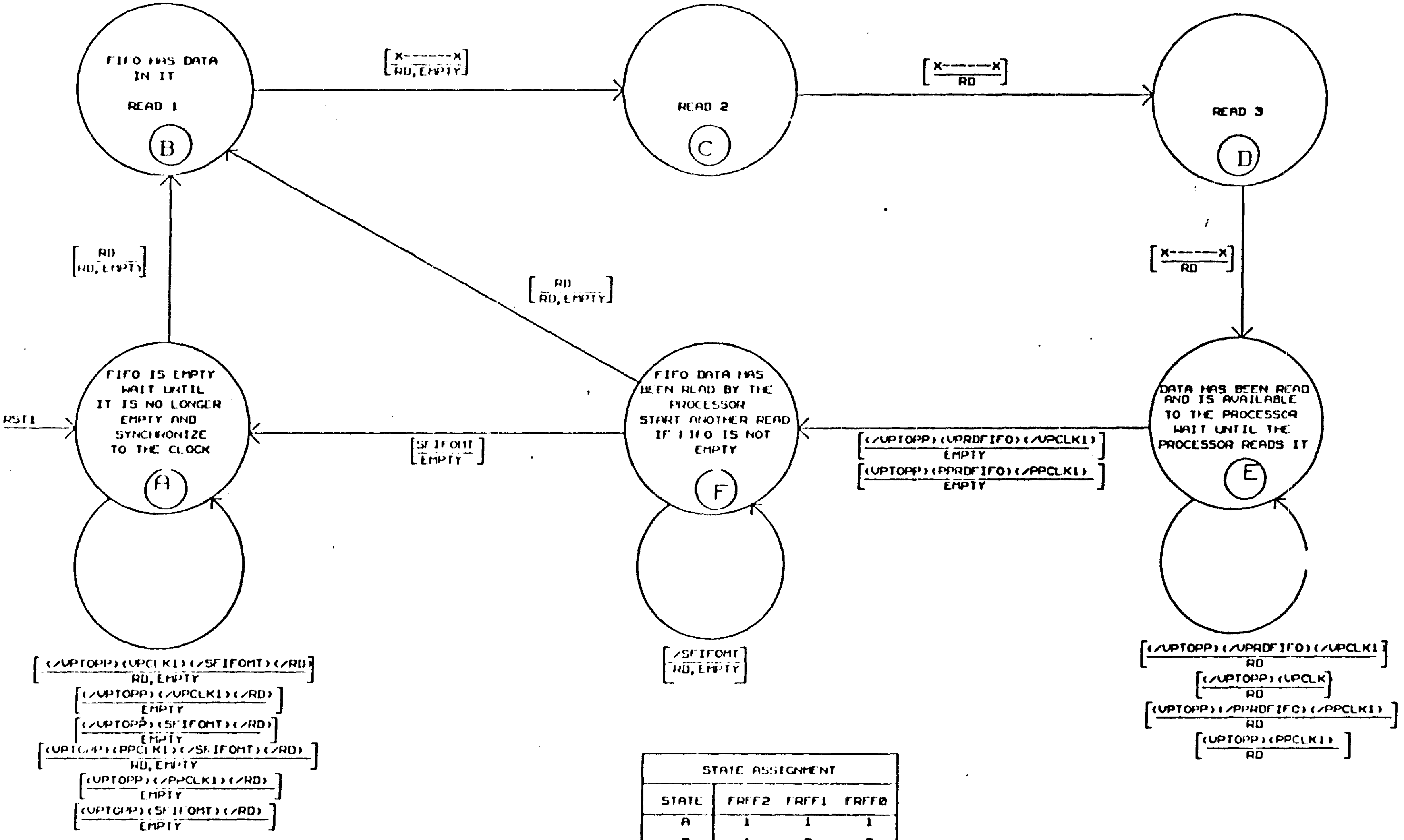


8
7
6
5
4
3
2
1
D
C
B
IDLE
A
RST
/NORMAL WRITE 1
B
/NORMAL WRITE 2
C
DELAY 1
D
DELAY 2
E
READ WRITE 1
F
READ WRITE 2
G
READ WRITE 3
H
SYNCH TO PPCLK
I
(/RD)(/WR)(/NORMAL)(/RAS) / 0------0
(/RD)(/WR)(/NORMAL)(RAS) / RAS, CAS
(/REFREQ)(/RD)(/WR)(NORMAL) / 0------0
(WR)(/NORMAL) / RAS, CAS, WE
X------X / RAS, CAS, WE, SMINCADR
REFREQ / REFRESH
/REFREQ / 0------0
(REFREQ)(NORMAL) / REFRESH
(REFREQ)(RD)(/NORMAL) / REFRESH
(/REFREQ)(RD)(/NORMAL) / 0------0
REFRESH / REFRESH
/REFRESH / 0------0
/REFRESH / RAS
REFRESH / REFRESH, RAS
(/REFREQ)(RD)(NORMAL) / RAS
(/REFREQ)(WR)(NORMAL) / RAS
(/REFRESH)(NORMAL)(WR) / RAS, CAS, WE
(/REFRESH)(NORMAL)(/WR) / RAS, CAS
(/REFRESH)(/NORMAL) / RAS, CAS
REFRESH / RAS, REFRESH
(/REFRESH)(NORMAL)(/WR) / RAS, CAS, LODOUT, ASTZBUFRDY
(/REFRESH)(NORMAL)(WR) / RAS, CAS, WE, SMINCADR, ASTZBUFRDY
(/REFRESH)(/NORMAL) / RAS, CAS, LODOUT, ASTZBUFRDY
REFRESH / RAS, REFRESH
(/REFRESH)(/NORMAL) / RAS, CAS
(REFRESH)(/NORMAL)(MCHANGE) / 0------0
(/REFRESH)(NORMAL)(/PPCLKZB) / 0------0
(REFRESH)(/NORMAL)(/MCHANGE) / 0------0
(REFRESH)(NORMAL) / 0----0
(/REFRESH)(NORMAL)(PPCLKZB) / 0------0
/PPCLKZB / 0------0
(/REFREQ)(PPCLKZB) / 0------0
(REFREQ)(PPCLKZB) / REFRESH

STATE ASSIGNMENT				
STATE	ZCFF3	ZCFF2	ZCFF1	ZCFF0
A	1	1	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	1	0	1
F	0	1	1	0
G	0	1	1	1
H	1	0	1	0
I	1	0	1	1

FIGURE 23

TITLE: Z BUFFER CONTROL STATE [illegible]INE	DATE: AUGUST 1
ENGINEER: SERDAR ERGENE	PAGE: 1 OF

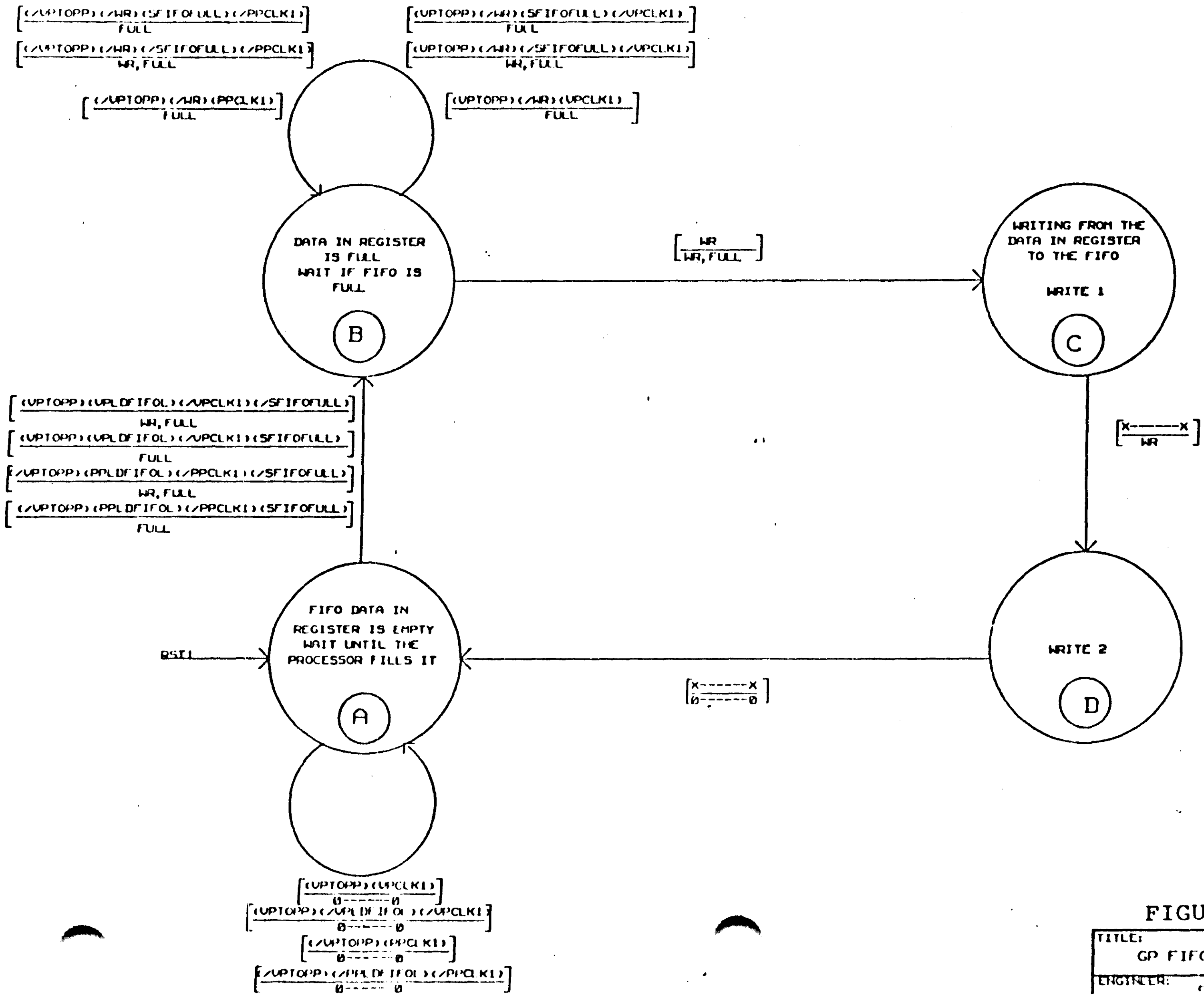

FIFO HAS DATA IN IT
READ 1
B
[X------X / RD, EMPTY]
READ 2
C
[X------X / RD]
READ 3
D
[X------X / RD]
[RD / RD, EMPTY]
[RD / RD, EMPTY]
RST1
FIFO IS EMPTY WAIT UNTIL IT IS NO LONGER EMPTY AND SYNCHRONIZE TO THE CLOCK
A
[SFIFOMT / EMPTY]
FIFO DATA HAS BEEN READ BY THE PROCESSOR START ANOTHER READ IF FIFO IS NOT EMPTY
F
[(/UPTOPP)(UPRDFIFO)(/UPCLK1) / EMPTY]
[(UPTOPP)(PPRDFIFO)(/PPCLK1) / EMPTY]
DATA HAS BEEN READ AND IS AVAILABLE TO THE PROCESSOR WAIT UNTIL THE PROCESSOR READS IT
E
[(/UPTOPP)(UPCLK1)(/SFIFOMT)(/RD) / RD, EMPTY]
[(/UPTOPP)(/UPCLK1)(/RD) / EMPTY]
[(/UPTOPP)(SFIFOMT)(/RD) / EMPTY]
[(UPTOPP)(PPCLK1)(/SFIFOMT)(/RD) / RD, EMPTY]
[(UPTOPP)(/PPCLK1)(/RD) / EMPTY]
[(UPTOPP)(SFIFOMT)(/RD) / EMPTY]
[/SFIFOMT / RD, EMPTY]
[(/UPTOPP)(/UPRDFIFO)(/UPCLK1) / RD]
[(/UPTOPP)(UPCLK) / RD]
[(UPTOPP)(/PPRDFIFC)(/PPCLK1) / RD]
[(UPTOPP)(PPCLK1) / RD]

STATE ASSIGNMENT

STATE	FRFF2	FRFF1	FRFF0
A	1	1	1
B	1	0	0
C	1	0	1
D	0	0	1
E	0	1	1
F	1	1	0

FIGURE 17

TITLE: GP FIFO READ STATE MACHINE	DATE: MAY 1984
ENGINEER: SERDAR ERGENE	PAGE: 1 OF 1


(/VPTOPP)(/WR)(SFIFOFULL)(/PPCLK1)
FULL
(/VPTOPP)(/WR)(/SFIFOFULL)(/PPCLK1)
WR,FULL
(/VPTOPP)(/WR)(PPCLK1)
FULL
(VPTOPP)(/WR)(SFIFOFULL)(/VPCLK1)
FULL
(VPTOPP)(/WR)(/SFIFOFULL)(/VPCLK1)
WR,FULL
(VPTOPP)(/WR)(VPCLK1)
FULL
DATA IN REGISTER IS FULL WAIT IF FIFO IS FULL
B
WR
WR,FULL
WRITING FROM THE DATA IN REGISTER TO THE FIFO
WRITE 1
C
X------X
WR
(VPTOPP)(VPLDFIFOL)(/VPCLK1)(/SFIFOFULL)
WR,FULL
(VPTOPP)(VPLDFIFOL)(/VPCLK1)(SFIFOFULL)
FULL
(/VPTOPP)(PPLDFIFOL)(/PPCLK1)(/SFIFOFULL)
WR,FULL
(/VPTOPP)(PPLDFIFOL)(/PPCLK1)(SFIFOFULL)
FULL
RST1
FIFO DATA IN REGISTER IS EMPTY WAIT UNTIL THE PROCESSOR FILLS IT
A
X------X
0------0
WRITE 2
D
(VPTOPP)(VPCLK1)
0------0
(VPTOPP)(/VPLDFIFOL)(/VPCLK1)
0------0
(/VPTOPP)(PPCLK1)
0------0
(/VPTOPP)(/PPLDFIFOL)(/PPCLK1)
0------0

STATE ASSIGNMENT		
STATE	FWFF1	FWFF0
A	1	0
B	1	1
C	0	1
D	0	0

FIGURE 15

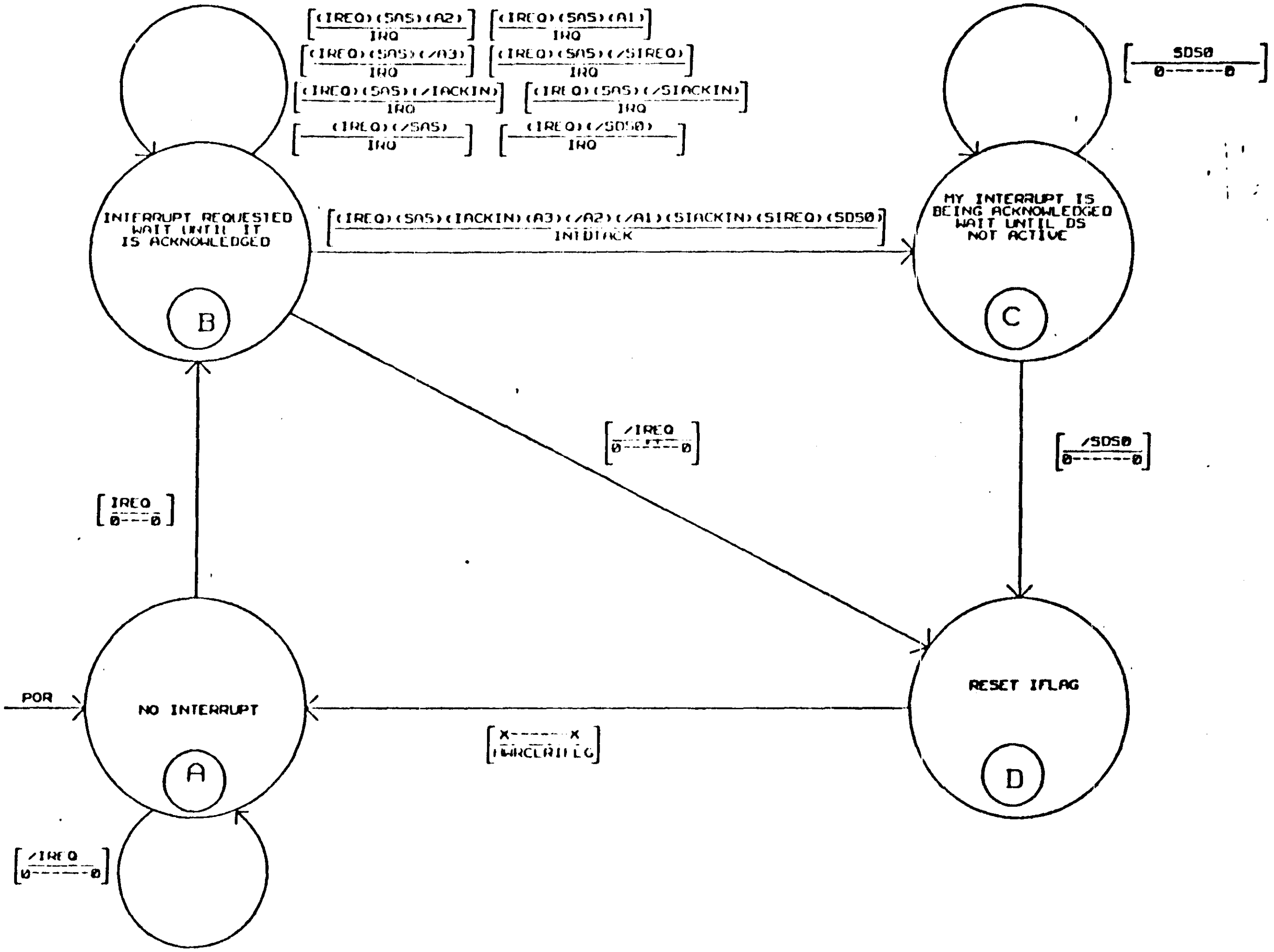
TITLE: GP FIFO WRITE STATE MACH..NE

DATE: MAY 1984

ENGINEER: [illegible]

PAGE:

SUN PROPRIETARY


[(IREQ)(SAS)(A2) / IRQ]
[(IREQ)(SAS)(A1) / IRQ]
[(IREQ)(SAS)(/A3) / IRQ]
[(IREQ)(SAS)(/SIREQ) / IRQ]
[(IREQ)(SAS)(/IACKIN) / IRQ]
[(IREQ)(SAS)(/SIACKIN) / IRQ]
[(IREQ)(/SAS) / IRQ]
[(IREQ)(/SDS0) / IRQ]
[SDS0 / 0------0]
INTERRUPT REQUESTED WAIT UNTIL IT IS ACKNOWLEDGED
B
[(IREQ)(SAS)(IACKIN)(A3)(/A2)(/A1)(SIACKIN)(SIREQ)(SDS0) / INTDTACK]
MY INTERRUPT IS BEING ACKNOWLEDGED WAIT UNTIL DS NOT ACTIVE
C
[/IREQ / 0---IT---0]
[/SDS0 / 0------0]
[IREQ / 0---0]
POR
NO INTERRUPT
A
RESET IFLAG
D
[X------X / HWRCLRIFLG]
[/IREQ / 0------0]

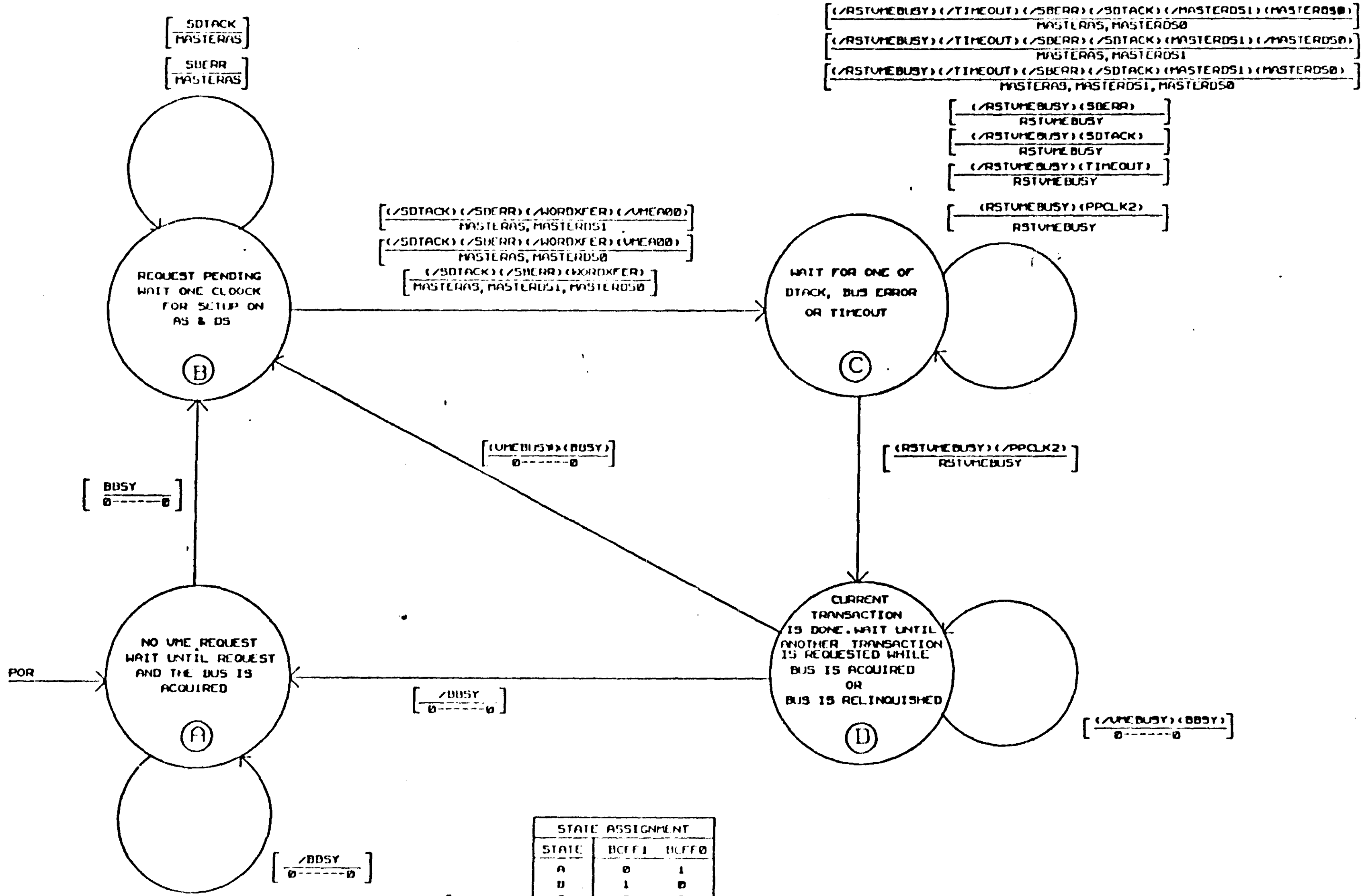
STATE ASSIGNMENT		
STATE	INFF1	INFF0
A	1	0
B	1	1
C	0	1
D	0	0

```
IACKOUT = /POR*IACKIN*VMEAS*(/A3+A2+A1)
          + /POR*IACKIN*VMEAS*(A3*/A2*/A1)*/SIREQ
```

FIGURE 11

TITLE: VME BUS GP INTERRUPTER	DATE: MAY 1984
ENGINEER: SERDAR ERGENE	PAGE: 1 OF 1

SUN PROPRIETARY



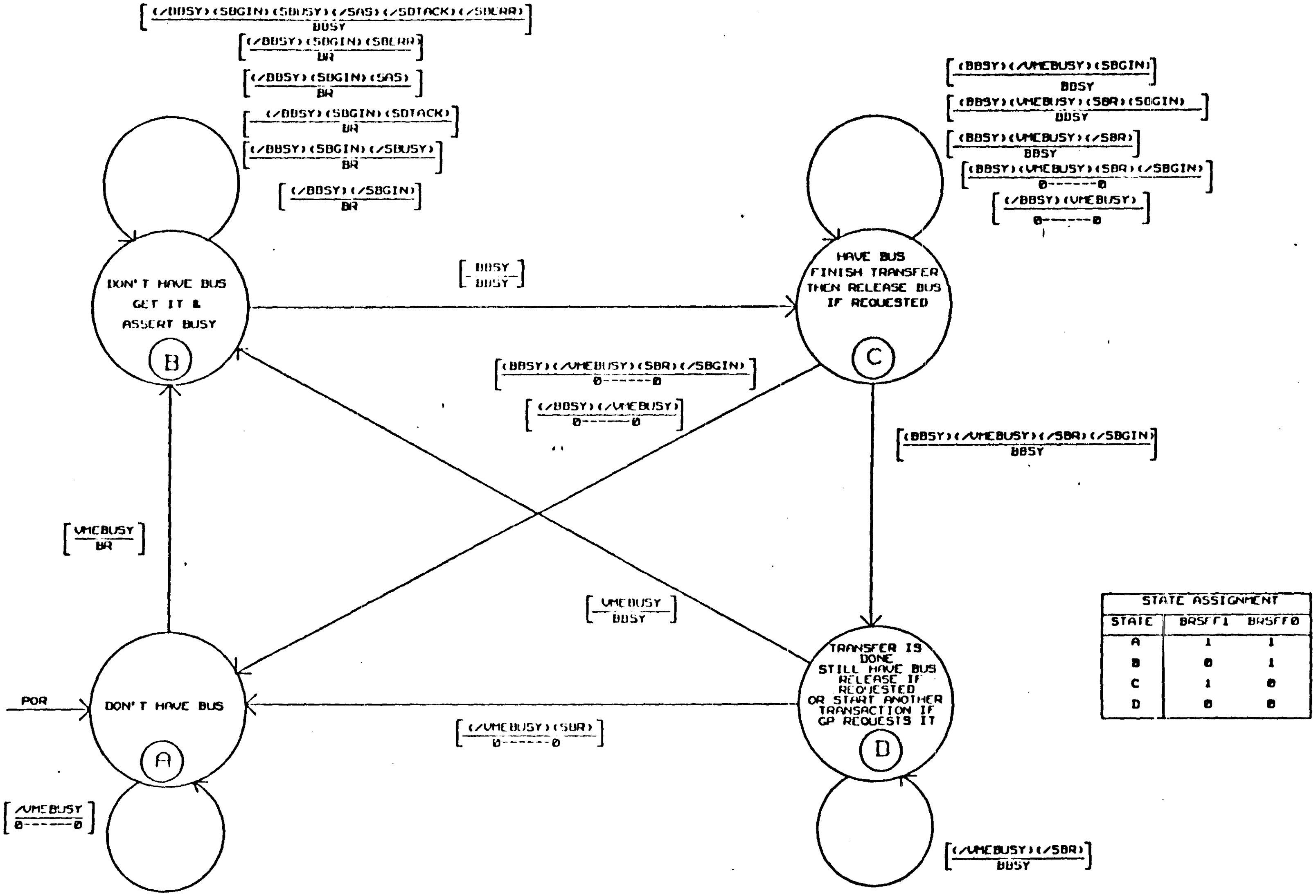
SDTACK / MASTERAS
SBERR / MASTERAS
REQUEST PENDING WAIT ONE CLOOCK FOR SETUP ON AS & DS
B
(/SDTACK)(/SBERR)(/WORDXFER)(/VMEA00) / MASTERAS, MASTERDS1
(/SDTACK)(/SBERR)(/WORDXFER)(VMEA00) / MASTERAS, MASTERDS0
(/SDTACK)(/SBERR)(WORDXFER) / MASTERAS, MASTERDS1, MASTERDS0
(/RSTVMEBUSY)(/TIMEOUT)(/SBERR)(/SDTACK)(/MASTERDS1)(MASTERDS0) / MASTERAS, MASTERDS0
(/RSTVMEBUSY)(/TIMEOUT)(/SBERR)(/SDTACK)(MASTERDS1)(/MASTERDS0) / MASTERAS, MASTERDS1
(/RSTVMEBUSY)(/TIMEOUT)(/SBERR)(/SDTACK)(MASTERDS1)(MASTERDS0) / MASTERAS, MASTERDS1, MASTERDS0
(/RSTVMEBUSY)(SBERR) / RSTVMEBUSY
(/RSTVMEBUSY)(SDTACK) / RSTVMEBUSY
(/RSTVMEBUSY)(TIMCOUT) / RSTVMEBUSY
(RSTVMEBUSY)(PPCLK2) / RSTVMEBUSY
WAIT FOR ONE OF DTACK, BUS ERROR OR TIMEOUT
C
(VMEBUSY)(BUSY) / 0-------0
(RSTVMEBUSY)(/PPCLK2) / RSTVMEBUSY
BUSY / 0------0
POR
NO VME REQUEST WAIT UNTIL REQUEST AND THE BUS IS ACQUIRED
A
/BUSY / 0------0
CURRENT TRANSACTION IS DONE. WAIT UNTIL ANOTHER TRANSACTION IS REQUESTED WHILE BUS IS ACQUIRED OR BUS IS RELINQUISHED
D
(/VMEBUSY)(BBSY) / 0------0
/BBSY / 0------0

STATE ASSIGNMENT		
STATE	BCFF1	BCFF0
A	0	1
B	1	0
C	0	0
D	1	1

FIGURE 10

TITLE: GP VME MASTER DATA TRANSFER CONTR[illegible]R	DATE: MAY 1984
ENGINEER: SERDAR ERGENE	PAGE: 1 OF 1

SUN PROPRIETARY


[(/BBSY)(SBGIN)(SBUSY)(/SAS)(/SDTACK)(/SBERR) / BUSY]
[(/BBSY)(SBGIN)(SBERR) / BR]
[(/BBSY)(SBGIN)(SAS) / BR]
[(/BBSY)(SBGIN)(SDTACK) / BR]
[(/BBSY)(SBGIN)(/SBUSY) / BR]
[(/BBSY)(/SBGIN) / BR]
DON'T HAVE BUS
GET IT &
ASSERT BUSY
B
[BBSY / BUSY]
[(BBSY)(/VMEBUSY)(SBGIN) / BBSY]
[(BBSY)(VMEBUSY)(SBR)(SBGIN) / BUSY]
[(BBSY)(VMEBUSY)(/SBR) / BBSY]
[(BBSY)(VMEBUSY)(SBR)(/SBGIN) / 0------0]
[(/BBSY)(VMEBUSY) / 0------0]
HAVE BUS
FINISH TRANSFER
THEN RELEASE BUS
IF REQUESTED
C
[(BBSY)(/VMEBUSY)(SBR)(/SBGIN) / 0------0]
[(/BBSY)(/VMEBUSY) / 0------0]
[(BBSY)(/VMEBUSY)(/SBR)(/SBGIN) / BBSY]
[VMEBUSY / BR]
[VMEBUSY / BUSY]
POR
DON'T HAVE BUS
A
[(/VMEBUSY)(SBR) / 0------0]
TRANSFER IS
DONE
STILL HAVE BUS
RELEASE IF
REQUESTED
OR START ANOTHER
TRANSACTION IF
GP REQUESTS IT
D
[/VMEBUSY / 0------0]
[(/VMEBUSY)(/SBR) / BUSY]

STATE ASSIGNMENT		
STATE	BRSFF1	BRSFF0
A	1	1
B	0	1
C	1	0
D	0	0

BG3OUT = /POR*/SBUSY*BGIN*/BBSY

FIGURE 9

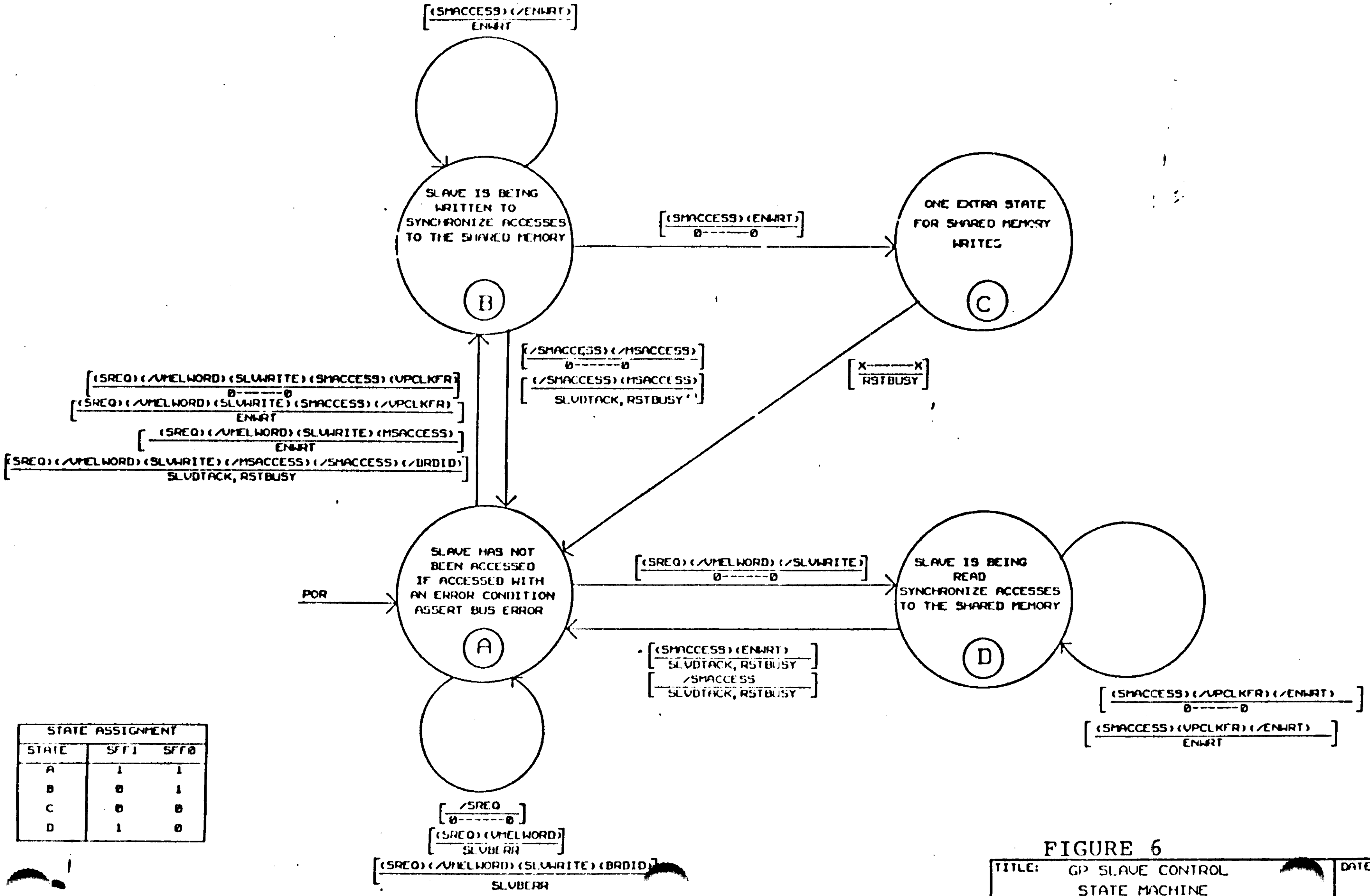
TITLE: GP VME BUS REQUESTER

DATE: MAY 1984

ENGINEER: SERDAR ERGENE

PAGE: 1 OF 1

SUN PROPRIETARY


[(SMACCESS)(/ENWRT) / ENWRT]
SLAVE IS BEING WRITTEN TO SYNCHRONIZE ACCESSES TO THE SHARED MEMORY
B
[(SMACCESS)(ENWRT) / 0------0]
ONE EXTRA STATE FOR SHARED MEMORY WRITES
C
[X------X / RSTBUSY]
[(/SMACCESS)(/MSACCESS) / 0------0]
[(/SMACCESS)(MSACCESS) / SLVDTACK, RSTBUSY]
[(SREQ)(/VMELWORD)(SLVWRITE)(SMACCESS)(VPCLKFR) / 0------0]
[(SREQ)(/VMELWORD)(SLVWRITE)(SMACCESS)(/VPCLKFR) / ENWRT]
[(SREQ)(/VMELWORD)(SLVWRITE)(MSACCESS) / ENWRT]
[(SREQ)(/VMELWORD)(SLVWRITE)(/MSACCESS)(/SMACCESS)(/BRDID) / SLVDTACK, RSTBUSY]
POR
SLAVE HAS NOT BEEN ACCESSED IF ACCESSED WITH AN ERROR CONDITION ASSERT BUS ERROR
A
[(SREQ)(/VMELWORD)(/SLVWRITE) / 0------0]
[(SMACCESS)(ENWRT) / SLVDTACK, RSTBUSY]
[/SMACCESS / SLVDTACK, RSTBUSY]
SLAVE IS BEING READ SYNCHRONIZE ACCESSES TO THE SHARED MEMORY
D
[(SMACCESS)(/VPCLKFR)(/ENWRT) / 0------0]
[(SMACCESS)(VPCLKFR)(/ENWRT) / ENWRT]
[/SREQ / 0------0]
[(SREQ)(VMELWORD) / SLVBERR]
[(SREQ)(/VMELWORD)(SLVWRITE)(BRDID) / SLVBERR]

STATE ASSIGNMENT		
STATE	SFF1	SFF0
A	1	1
B	0	1
C	0	0
D	1	0

FIGURE 6

TITLE:	GP SLAVE CONTROL STATE MACHINE	DATE:	MAY 1984
ENGINEER:	SERDAR ERGENE	PAGE:	1 OF 1