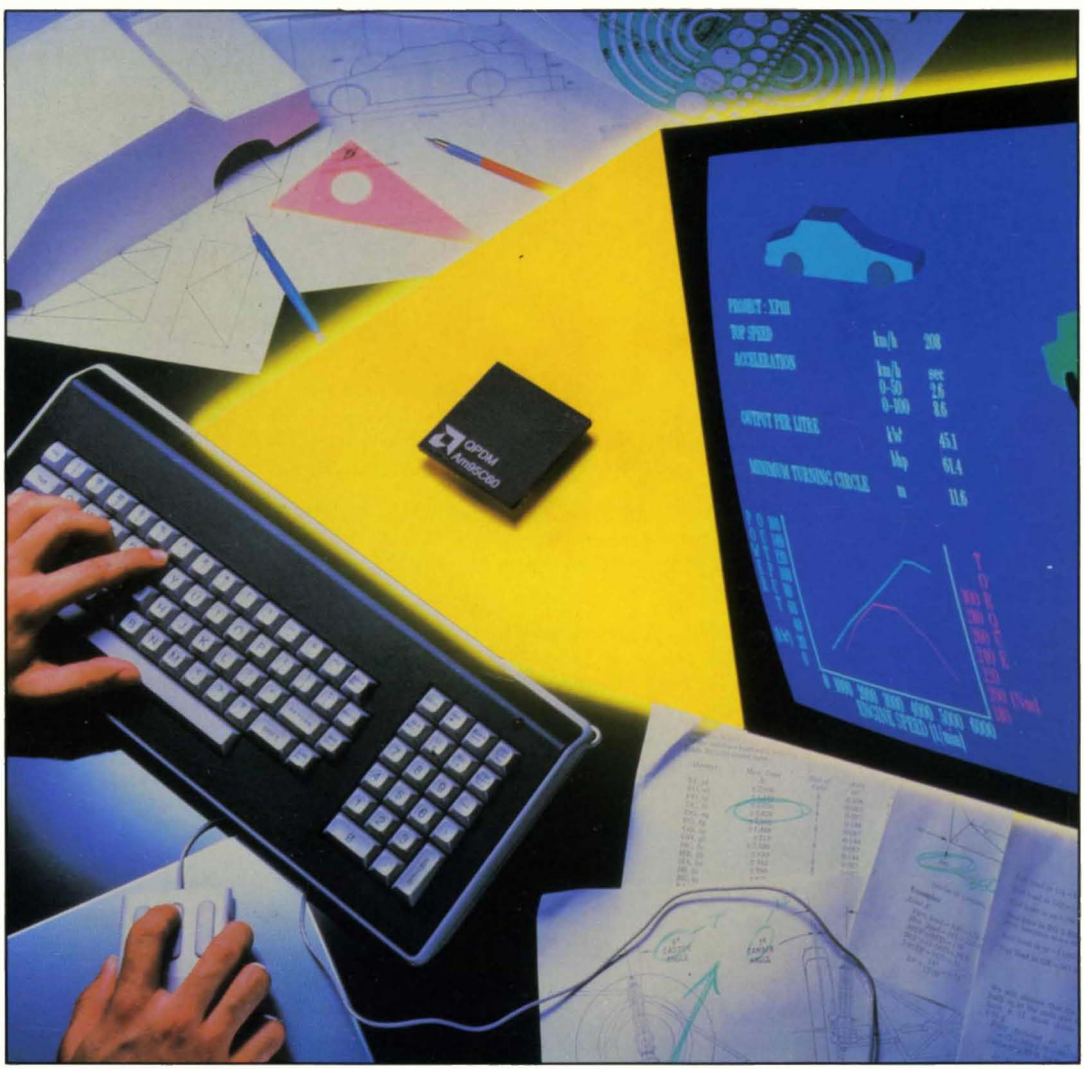
Am95C60
Quad Pixel Dataflow Manager

Applications
Handbook

Advanced
Micro
Devices



Advanced Micro Devices

Am95C60
QPDM Quad Pixel Dataflow Manager

Applications Handbook

901 Thompson Place, P.O. Box 3453, Sunnyvale, California 94088-3000
(408)732-2400 TWX: 910-339-9280 TELEX: 34-6306

Authors:

Tom Crawford	Chapters 3.0, 3.2, 3.3, 4, 5, and 6.
Stuart Tindall	Chapters 1 and 3.1.
Ed Dupuis	Chapter 2.1.
Wolfgang Reis	Chapter 2.2.
Achim Strupat	Chapter 2.3.

Introduction

This QPDM Applications Handbook is the third in a series of documents describing the Am95C60 Quad Pixel Data Manager (QPDM) device and its use in graphics systems.

The first, most basic document is the QPDM data sheet (Order Number 07013B) which gives a terse functional description plus a very detailed listing of the electrical and timing parameters, as well as package, pin-out, and ordering information. This data sheet will be updated for any parametric changes, e.g. speed enhancements, made as the device matures.

The second document is the QPDM Technical Manual (Order Number 07785B). It provides a more complete functional description and explains each of the 61 instructions in detail.

The third document, this QPDM Applications Handbook, describes a wide variety of interfaces to the QPDM. The System Bus is covered in Chapters 2 and 5, the Display Memory Bus is covered in Chapters 3 and 5, the Memory Bus in Chapters 4 and 5.

Chapter 6 contains some programming hints and a complete initialization program.

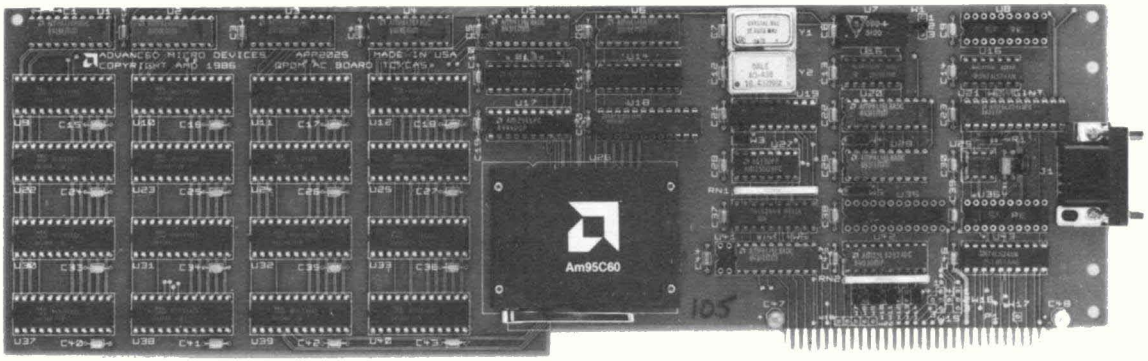


Table of Contents

CHAPTER 1

Overview

CHAPTER 1
Overview

DESIGN APPLICATIONS

Interface helps controller boost graphics performance

Stuart Tindall and Achim Strupat
Advanced Micro Devices Inc., 901 Thompson Pl., P.O. Box 3453, Sunnyvale, CA 94088; (408) 732-2400.

A new generation of graphics processors is hiking the performance of graphic systems by more than an order of magnitude. These devices work their wonders by taking over tasks formerly performed by the system's CPU: frame updating, video refreshing, and memory refreshing. The dedicated processors offload the system CPU of unnecessary tasks while they manipulate image data faster than the original controllers ever could.

A dedicated graphics processor speeds image handling while making the CPU's job easier. Moreover, its bus interface is a snap to implement.

The price to pay for the extra speed and the easier overall system design is the added complexity of connecting one or more graphics processors to the system bus. The task need not be daunting, however. In fact, with the Am95C60 Quad Pixel Dataflow Manager, the connection to the system bus is very straightforward.

The Am95C60 CMOS device is aimed at minicomputers and workstations built around the 68020 microprocessor. The device manages bit maps of up to 4096 by 4096 pixels and pixel rates of up to 400 MHz, which translates into screen sizes of up to 2000 by 2000 pixels. For reference purposes, today's high-resolution CAD and desktop publishing workstations have 1280-by-1024-pixel displays.

The graphics processor can draw up to 110,000 lines, averaging 10 pixels long, per second; place text at 50,000 characters per second; fill polygons at 20 ns per pixel; and perform bit-block transfers at 60 ns per pixel. One device manages and drives up to four bit-mapped memory planes, and designers can cascade up to 64 devices without slowing performance. As a result, a system based on the Am95C60 processor can support 256 display memory planes.

Moreover, the graphics processor connects directly to video dynamic RAMs and supplies all the signals to drive them. Video dynamic RAMs are dual-port memories that make possible simultaneous display refresh from a serial port and display update through a random-access port. In a high-resolution system with video dynamic RAMs, the update bandwidth exceeds 90%, almost triple that with conventional dynamic RAMs.

In a typical graphics subsystem, one or more graphics processors connect to the system bus. Other major components include video dynamic RAMs, one serializer per memory plane, a high-frequency dot clock generator, and a color palette (Fig. 1).

Note that the 68020 is a bus master, and the Am95C60 is addressed as a bus slave. If the system did not have a DMA controller, the CPU would be the only bus master, and it would never have to perform bus-arbitration cycles or give away the system bus. The optional DMA controller helps the CPU load instructions into or exchange data with the graphics processor, but the controller must request the bus and use the standard bus-arbitration handshake.

Because the graphics processor is always a system bus slave, the transactions on its interface do not have to be synchronized to a clock. As a result, bus-interface connections are relatively simple, and the graphics processor needs only a small amount of additional logic to work with all common 8-, 16-, or 32-bit microprocessors.

8- OR 16-BIT MODES POSSIBLE

After a reset, the designer can configure the data bus to work in an 8-bit mode with programmable byte order or in a 16-bit mode. In a 68020 system, the 16-bit mode offers the highest throughput. In this case, the device's 16 data lines connect to bits 16 through 31 of the processor's 32-bit data bus.

The 68020 accommodates both virtual and direct addressing. Because the processor does not distinguish between memory and I/O addresses, peripherals are memory mapped. Virtual memory management is better when the 68020 is the kernel CPU running a high-level operating system. Then

any execution process can access the CPU's total address space—4 Gbytes for the 68020.

Direct addressing is preferred when the 68020 controls peripheral devices or when the peripherals have unique addresses because only one user can access the quad pixel data-flow manager in an interactive graphics system. Even if this user displays results from a multitasking process, the I/O accesses run sequentially through an operating system driver. Ideally, a PAL device contains the address decoding logic needed to generate the relevant Chip Enable signals to the graphics processor. As a result, the following discussion assumes a direct-addressing scheme.

To interface a processor to a peripheral with an independent system bus cycle, as the graphics processor has, several control and response signals must be translated. Also, each device must operate at its own highest clock rate, and therefore, asynchronously.

The interface to the Am95C60 includes a 16-bit bidirectional, three-state data bus (lines D_0 to D_{15}), Read and Write strobe inputs (RD and WR), a Chip Select input (CS), two address line inputs (A_0 and A_1), an interrupt output, three DMA handshake signals, an output that enables an external driver, a reset, and a system clock input. The system clock, which runs at up to 20 MHz, times the internal microengine and controls the display-memory timing, but not the system-bus and video timing. The two address lines connect to four ports within the device.

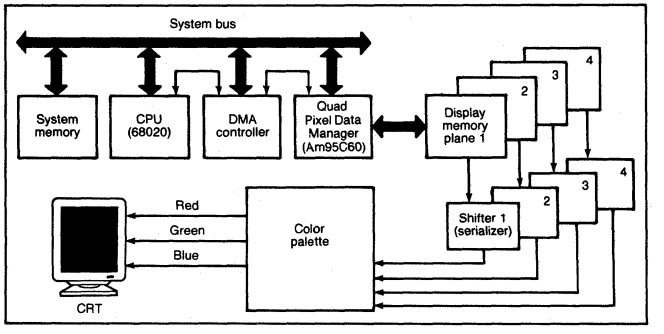
A typical application has a 68020 connected to two Am95C60s that form an eight-plane system (Fig. 2). A PAL device decodes the 68020 address and outputs two Chip Select (CS_0 and CS_1) signals to the graphics processors. A third signal, CSQPDM, which shows an access to either graphics processor, combines with the 68020's Read/Write signal to form the read and write inputs for the Am95C60. The timing of the Chip Select signal and the read and write inputs follows the timing of the 68020's Address Strobe, with the addition of the decode logic's propagation delay.

Because all resources within the graphics processor are 16 bits wide, any write cycle to the device results in the 16-bit quantity on the bus being loaded into the appropriate register. But the data word must be aligned, because the least significant address line is not used in addressing the resources within the device. Connecting the 68020's address lines, A_1 and A_2, to the graphics processor's address pins, A_0 and A_1, allows data to be transferred one word at a time. Consequently, the quad pixel data-flow manager does not need any transfer-size information; the requested 16 bits are always fulfilled. The 68020's address bits A_1 and A_2 connect to the graphics processor's two address inputs, A_0 and A_1, to select the internal resource for a bus access.

Depending on the speed difference between the two processors, none, one, or more wait states extend the processor's bus cycle. Two lines—Data Transfer and Size Acknowledge (DSACK)—cause wait states in the bus cycle if they are not asserted. To avoid this, the designer can generate DSACK responses that are synchronous to the 68020 clock by using a fixed-delay logic sequence that defines the length of any access to the graphics processor after Address Strobe is asserted.

This delay can be modulated by a registered PAL device, timed by the processor's clock. The PAL device's DSACK outputs are put in a three-state mode because a multiperipheral system may have several sources of DSACK signals. A pull-up resistor on the three-stated outputs defines the logical state when the PAL device is not driving the lines.

The 68020 communicates with the graphics processor over two types of bus cycles: word read and word write. At the beginning of a system bus cycle, the 68020 as the bus master asserts the R/W signal to indicate the direction of data flow. A word read cycle moves data from the graphics processor to the CPU, then presents the address and function code to distinguish between user and supervisory address space.



1. The AM95C60 Quad Pixel Dataflow Manager can accommodate four bit-mapped memory planes. A graphics subsystem can include up to 64 devices, for 256 memory planes.

To transfer 16 bits with a word-transfer instruction, the CPU's transfer size outputs, SIZ_1 and SIZ_0, are set to two bytes (that is, 10H). As noted, however, the Am95C60 does not need this information, if the CPU's address lines and the graphics processor's address pins are properly connected.

Because all transactions are word aligned and word wide, address bit A_0 is always low,

and the information is read in on the data-bus lines D_{16} through D_{31}. The 68020 then asserts Address Strobe to show that the address is valid, and activates Data Strobe to indicate that the graphics processor should drive the data bus.

After the CPU produces Chip Select and asserts the read input, the graphics processor needs a specified minimum time to complete the word read bus cycle, depending on the device's speed version. For a 20-MHz device, for instance, the read data on the system bus is valid after a maximum of 110 ns.

Depending on the 68020's clock speed, this time may or may not be fast enough to ensure access without a wait state. To cover the general case, the example assumes that after a certain delay, the graphics processor creates a wait state by asserting the DSACK lines to the CPU. The delay, produced by external logic, is a multiple of the 68020's clock cycle.

Once valid data is on the data bus and the DSACK signals are asserted, the CPU latches the data and terminates the bus cycle by deasserting Data Strobe and Address Strobe. This negates the graphics processor's Read Strobe and lets the device enter the three-state mode on its data bus. To conclude the bus cycle, the PAL device that generates the DSACK signals goes into a three-state mode also.

The pull-up resistor brings the outputs to an inactive state. As soon as the CPU receives the deasserted DSACK signals, it knows the word read cycle is complete and starts another cycle.

The transactions are similar for a write cycle, except that the R/W signal shows a transfer from the CPU to the graphics processor. In this case the CPU places valid data on the system bus before activating the Data Strobe. Because the Write Strobe to the graphics processor may be as short as 70 ns, fewer wait states are needed than in a Word Read cycle. If any are inserted, the PAL device's DSACK signals again handle them.

Assertion of the two DSACK lines tells the CPU that the graphics processor is ready to latch the data. The 68020 then negates its Address and Data strobes and removes the data from the bus. The graphics processor latches the data with the rising edge of its Write pulse, which is formed from the 68020's Data Strobe pulse. Negation of the DSACK signals by the PAL device after the rising edge of Address Strobe concludes the bus cycle.

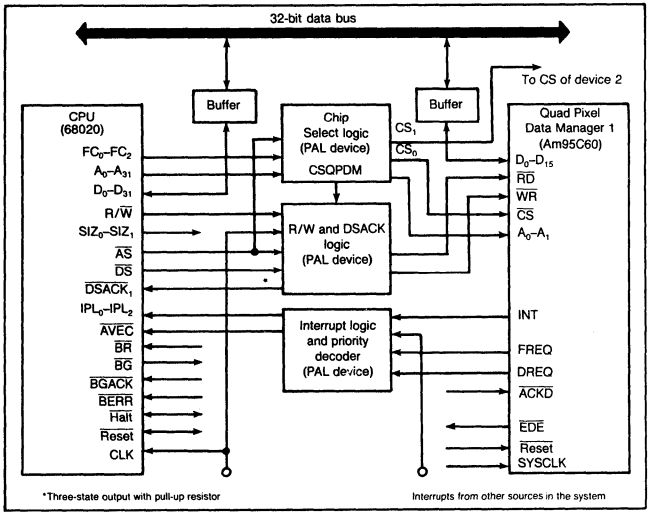
ADDRESS LINES CONTROL I/O PORTS

The Am95C60 has four I/O ports addressed by lines A_0 and A_1. Through these ports, the CPU, using its own A_1 and A_2 lines, directly addresses the instruction FIFO buffer and status register, the block I/O FIFO buffer, the I/O-pointer register, and the data registers indicated by that register (see the table, p.100).

Access to the other registers within the device employs a two-tier process. The operator first loads the address of the desired resource into the I/O-pointer register, through which the resource can be accessed. Then, any subsequent access to the I/O-data register transfers data between the bus master and the register.

In a multiple data-manager system, the processors can be addressed either individually or as a single peripheral. The broadcast mode, in which the CPU transmits data to all quad pixel data managers simultaneously, offers the fastest overall system speed.

In broadcast mode, a global address enables all Chip Select lines. The CPU sends most of the register data during the initialization phase and all but one drawing instruction to the quad pixel data managers. All the graphics processors then execute the same instruction



2. In the straightforward system bus connection between the Am95C60 graphics processor and the 68020 microprocessor, three PAL devices create the logic needed for the connection. The CPU's clock and the graphics controller's clock are asynchronous, easing the interface.

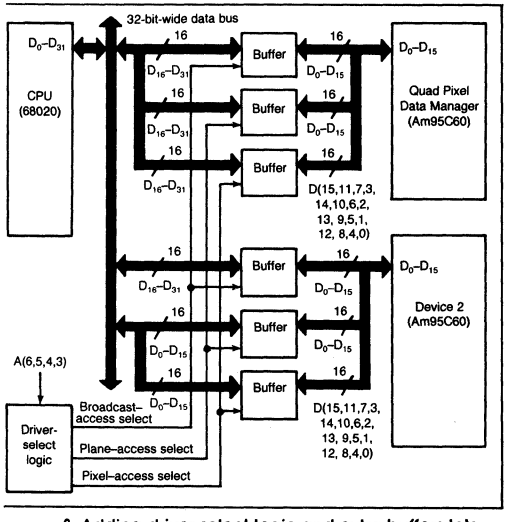
simultaneously. The 68020's address bits A_3 and A_4 create the required Chip Select signals in a two-data-manager system (see the table again).

The one instruction executed can affect different display memory planes differently, depending on what data is in the display memory and on certain parameters in the quad pixel data manager. Those parameters include activity bits, which define the active planes; color bits, which pick the color the graphics processor draws with; and search bits and listen bits, which show what color is needed in certain planes for fill area instructions.

The instructions that set these parameters include a field denoting which quad pixel data manager is being addressed. Each device compares its plane position with this field to determine whether the device is a target. Thus instructions that change only one quad pixel data manager within an array can be broadcast to all devices.

Initialization of individual Am95C60s is important because the multiple graphics processors in a system may be initialized differently. One graphics processor may be the video master, while the others are video slaves. Each device's position in an array is determined by the Set QPDM Position instruction, which must be sent to each chip individually.

A user activates the Am95C60 graphics engine by initializing several registers that define its environment. For instance, these registers specify the type and size of the connected display memory, the video timing, the dynamic memory refresh frequency, and the screen and window size and position within the display memory.



3. Adding driver select logic and extra buffers lets the operator of this two-controller system choose a data-transfer scheme from among a 16-bit broadcast technique, a 32-bit by-plane transfer, and a 32-bit by-pixel transfer.

After initialization, the CPU transmits drawing instructions to the graphics processor. The 60 instructions in the set include drawing lines, moving rectangular blocks, filling triangles, and writing character strings. The selection allows users to create many different types of drawings and to mix graphics and text efficiently.

The graphics processor accepts instructions in three ways: programmed I/O loading, fast loading with an external DMA controller, and program mode. Programmed I/O is the most straightforward method. The host processor directly addresses the instruction FIFO buffer, supplying instructions as long as the buffer has space.

The CPU checks whether the buffer needs service by polling the internal status register or the open-drain FIFO Request (FREQ) output pins, because the FREQ signal is asserted whenever the buffer has room for at least one instruction word.

In a system with multiple graphics processors, all the FREQ pins are tied together and their open-drain structure is connected to an external pull-up resistor that performs a logical AND function. Therefore, only when every device has room in its instruction FIFO buffer will the FREQ node be asserted. The FREQ signal can also be tied into the interrupt structure to request immediate service from the CPU.

Or a DMA channel can load the instruction FIFO buffer directly from system memory. This method also employs the FREQ signal for handshaking with the DMA controller. In effect, the graphics processor requests additional instructions under control of the previously initialized DMA channel.

Finally, the program mode uses the Call instruction to cause the graphics processor to read instructions from the nonvisible part of the display memory, instead of accessing the instruction FIFO buffer. Basically, the program mode switches the device from a Harvard to a von Neumann architecture because the display memory bus both delivers instructions and transports manipulated data.

Essentially, the CPU writes a group of instructions into the display memory. Then a Call instruction is executed, meaning that subsequent instructions are fetched from the display memory. Embedded Calls allow nesting of subroutines within the display memory. A Return instruction restores control to the instruction FIFO buffer.

Users employ Block input and Output instructions to transfer data between the display and system memories. These commands move image, or font, and control data over the Data Input and Output FIFO buffer. Several access methods exist.

Programmed I/O loading is the simplest technique. The data request, DREQ, bit in the status register or the DREQ pin signals when the BLOCK FIFO buffer needs

service. As with the FREQ bit, this condition can be tested by polling, by letting it generate an interrupt, or by the AND structure of the open-drain DREQ node.

In addition, a dedicated DMA channel can service the Block FIFO buffer. The Acknowledge Data, ACKD, line allows the DMA channel to accommodate a two-bus-cycle DMA transfer (Flow-Thru Mode) or a single-bus-cycle transfer (Fly-By Mode).

When transferring data between system bus and display memory, the user can access the data by plane or pixel. A by-plane access transfers 16 bits from one plane. On the other hand, a by-pixel access transfers a complete pixel, meaning one-bit from each plane.

For best efficiency, a designer should choose a transfer scheme that fills the 68020's 32-bit-wide data word. For example, in a two-graphics processor system, a by-plane access transfers 16 bits from each of two planes to the 32-bit bus. Or a by-pixel access allows four 8-bit deep pixels to be transferred to the CPU's bus.

The example application of a two-graphics processor system needs additional data buffers between the system bus and the graphic processors' data lines. These buffers multiplex the relevant data lines to the correct data bits on the bus (Fig. 3).

The 68020 uses the Chip Select lines to enable the buffers and chooses between the additional or the standard access buffers. The choice is implemented by the CPU's address bits A_5 and A_6, which enable the relevant data bus driver. They select either a 16-bit-wide broadcast access using bits 16 through 31, a 32-bit by-plane access of two planes, or a 32-bit by-pixel access of four 8-bit pixels.

Addressing the Am95C60's internal resources

Function desired	68020 address lines A_6	A_5	A_4	A_3	A_2	A_1
Access instruction FIFO register for write access and access status register for read access	X	X	X	X	0	0
Access block in/out FIFO register	X	X	X	X	0	1
Access I/O pointer	X	X	X	X	1	0
Access register pointed to by I/O pointer	X	X	X	X	1	1
Both Quad Pixel Data Managers are accessed (broadcast)	X	X	0	0	X	X
Device 1 is accessed	X	X	0	1	X	X
Device 2 is accessed	X	X	1	0	X	X
Reserved	X	X	1	1	X	X
16-bit wide broadcast	0	0	X	X	X	X
Double 16-bit data transfer	0	1	X	X	X	X
Four pixel with 8 bits each	1	0	X	X	X	X
Reserved	1	1	X	X	X	X

*A_3 through A_6 are undefinable.

Ten maskable conditions in the Am95C60 can signal interrupts to the CPU over the INT output. Typically, this signal connects to a priority encoder that arranges the interrupts for servicing in preferred order. The encoder then asserts the relevant interrupt levels on the CPU's Interrupt Level Priority lines, ILP_0 to ILP_2.

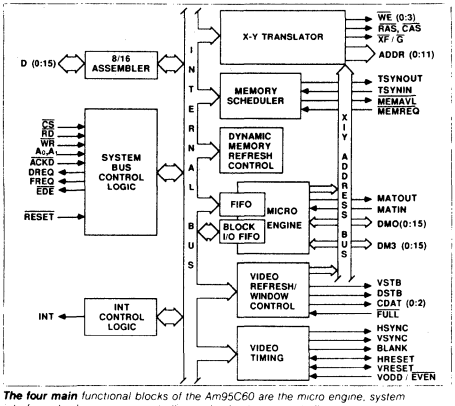
When the CPU detects an interrupt level greater than the current one, it waits until the end of the current instruction, saves its state, and generates an interrupt acknowledge bus cycle to find out which device has raised the interrupt. The device responds with either a vector number or by asserting AVEC, which requests an internally generated vector. The Am95C60 employs the autovector method to handle interrupt acknowledge. Both methods point to an interrupt service routine.

On entering the interrupt service routine, the CPU software reads the graphics processor's status register to find out which interrupts are outstanding. The CPU clears the bits for the interrupt it will service by writing to the graphics controller's interrupt acknowledge register and then it re-enables its interrupt system. Writing the register not only tells the graphics chip that the CPU has serviced the interrupt, but it also clears the relevant interrupt bits, which, when set, assert the interrupt line.

Because all Am95C60 controllers in a multi-unit system execute the same instruction simultaneously, any interrupt will be detected by all the devices, and flagged in their status registers. The 68020 reads the status register of one quad pixel data-flow manager, using its individual Chip Select address, to avoid having several chips drive the data bus at the same time. A Write signal to the interrupt acknowledge register of all the Am95C60s clears the interrupt on all the chips. □

Stuart Tindall is a field applications engineer specializing in graphics products. He works out of AMD's UK office in Warrington. Tindall received his electronic engineering degree from Liverpool University, UK.

Achim Strupat, a field application engineer in AMD's Southern California office, previously was a member of the Quad Pixel Dataflow Manager product-planning group in Sunnyvale, Calif. Strupat earned his MSEE at the Rheinisch Westfaelisch Technische Hochscule in Aachen, West Germany.



***The four main** functional blocks of the Am95C60 are the micro engine, system interface, display memory controller, and video timing controller*

Graphics controller draws 110,000 lines/sec

Controls four bit-mapped memory planes

KA WAI LEUNG
Advanced Micro Devices

To meet the drawing-speed demands of bit-mapped graphics systems, AMD's Am95C60 quad pixel dataflow manager (QPDM) draws as many as 110,000 vectors/sec. The chip also places text at 45,000 cps and fills polygons at a rate of 20 nsec/pixel.

This 20-MHz CMOS processor has four 16-bit-wide data-path units and a 16-bit-wide address-generation unit. These units equip the chip to control one to four bit-mapped memory planes for a screen resolution as fine as 2000×2000 pixels. Each display-memory plane contains as many as 4k×4k bits.

The chip's four 16-bit data units work in parallel, obeying instructions that are decoded and executed in a 16-bit micro engine with a 50-nsec instruction time. In addition, parallel architecture lets you cascade as many as 64 95C60s to support 256-memory-plane systems with no degradation in performance.

Graphics primitives

To speed execution, the device uses hard-wired graphics algorithms to reduce the number of instructions associated with each operation. In contrast, a programmable graphics processor requires the user to have a detailed knowledge of its internal architecture and to spend time developing software.

The 95C60's instruction set supports the implementation of such graphics standards as Computer Graphics Interface (CGI), Graphical Kernel System (GKS), and Graphics Device Interface (GDI). A micro engine handles instruction execution.

Functional blocks

The micro engine, one of four main functional blocks on the chip, uses a ROM with microcode wide enough for parallel control of all the execution units. In addition, the engine has a branch sequencer.

The second functional block is a system interface that links the system bus with the various control blocks. Always a slave to the bus, the interface provides a 16-bit bidirectional data path that can be reconfigured to 8 bits for connection to an 8- or 16-bit host processor. To minimize the load on the host, a dedicated DMA controller can be used to manage data movement.

During DMA operations, the 95C60 uses a 64-word-deep instruction FIFO buffer operating in flow-through mode to minimize CPU waits during instruction transfers. In addition, block-in and block-out buffers speed data transfer from system memory to the display memory. These block buffers use either flow-through or fly-by DMA.

The third functional block—the display-memory controller—generates display-memory timing and arbitrates video refreshes, memory refreshes, and update accesses. To avoid video and update contentions for the display memories, the 95C60 supports dual-port video RAMs.

Video RAMs

Video RAMs improve the updating of graphics memory by more than five times over standard dynamic RAMs. Using video RAMs lets the chip refresh a 1280×1024-pixel screen at 60 Hz noninterlaced, which translates into a 120-MHz pixel rate with more than 90% of the time available for display update.

Another part of the display-memory controller is a translator that lets the CPU use X/Y coordinates for background and window locations. This frees the CPU from having to convert X/Y screen coordinates to display-memory locations. An additional feature that reduces CPU intervention is program mode, which lets you store program data, pointer, and stack values alongside the displayable screen.

The display-memory controller also has a data-plane controller, which contains four 16-bit data-logic units and four 16-bit bidirectional barrel-shifters.

The last main functional block is the video-timing controller, which generates timing signals to control the video monitor and data transmission on the video bus. Twelve video-control registers define horizontal timings, vertical timings, and operating mode. You can program the 95C60 to be the horizontal master or slave(s) and the vertical master or slave(s) of another video source in the system.

Block copy

One of the 95C60's most useful capabilities is block copy. Operating at 50-nsec/pixel, the chip moves large blocks of data within the bit map, allowing source and destination overlaps without contention or loss of data.

During block copy, source data can be rotated in 90° increments, mirrored, and zoomed independently in X and Y directions. You can perform logical operations to the source pixel before it is written to the destination. Because the 95C60 supports mask write in video RAMs, the user can preserve data integrity in selected memory planes during memory accesses.

The block-copy feature also supports one hardware window and many software windows. Unlike software windows, the hardware window does not overwrite the image it replaces.

When using multiple windows, you can designate the most frequently used window as the hardware window and all remaining windows as software windows. The 95C60 responds rapidly to window movements by altering pointers instead of bit-map contents.

In addition to windowing, the chip supports panning, scrolling, and zooming of graphics primitives drawn in various line styles. The 95C60 uses an antialiasing scheme that smooths out the jagged edges of lines, arcs, and circles by illuminating adjacent pixels.

Other 95C60 capabilities include support of proportional spacing and fonts as large as 63×60 pixels—30 times larger than the 9×14-pixel character font of an IBM PC. This large font capability allows the chip to support such foreign-language character sets as Kanji, which requires 24×24 pixels to produce Japanese characters.

The 95C60 comes in a 144-pin pin-grid-array package. Prices are $198.57 for the 12-MHz version, $250 for the 16-MHz unit, and $278.57 for a 20-MHz device (100). Production quantities are available now; delivery, four to six weeks ARO. □

For more information on the Am95C60 graphics controller,

Ka Wai Leung is senior strategic-development engineer at Advanced Micro Devices Inc, Box 3453, Sunnyvale, CA 94088. Phone (408) 749-3412.

BEHIND THE DESIGN

Bit-map design called for video RAMs

AMD started designing the Am95C60 at the beginning of 1982 in response to increasing demand for bit-mapped, high-resolution text and graphics displays. The objective was to build a high-performance graphics controller based on the company's bit-slice architecture.

The bit-slice approach lets the designer increase data width by cascading multiple bit-slice processors. In the 95C60, the pixel width (color resolution) can be increased by cascading 95C60s—one for every four bit planes—without sacrificing drawing performance.

One of the biggest design challenges was keeping the die size <200k mil². This task was complicated by the chip's amount of parallelism and its high degree of integration. Because the initial design called for a display memory that supported dynamic RAMs, the chip needed to incorporate a large video-stream FIFO buffer. However, the emergence of video RAMs as the preferred type of bit-map memory lead to a decision to drop dynamic-RAM support and substitute on-chip support for video RAMs.

CHAPTER 2

System Bus Interface

CHAPTER 2

System Bus Interface

In this section , detailed descriptions of the System Bus Connections for three popular buses are presented. These are the 80186 microprocessor bus, the VME bus, and the 68020 microprocessor bus. We also include schematic diagrams and PAL device equations. While these designs have not actually been built and tested, they have undergone substantial "paper testing".

For a detailed analysis of a demonstration/evaluation board that has already been built and tested, please refer to Section 5.

2.1 Am9560 - 80186 INTERFACE DESIGN

In this section, two designs employing the Am95C60 are discussed in detail. A single Am95C60 system (i.e., four bit planes) will first be discussed and then a four Am95C60 system (i.e., 16 bit planes) will conclude this section. The major theme in these designs is to keep chip count and cost to a minimum. The designs use a 10 MHz 80186 and a 20 MHz Am95C60. The features of both the Am95C60 and the 80186 help to achieve these goals.

2.1.1 Single QPDM Design

Figure 2.1-1, "Single QPDM Schematic ", shows schematically how to connect a single QPDM.

Read Cycle

Figure 2.1-2a, "Single QPDM Read Cycle Timing", shows the QPDM read cycle when using a 10 MHz 80186. In this design the $\overline{PCS_5}$ and $\overline{PCS_6}$ output pins have been programmed to reflect the state of the A_1 and A_2 address lines and $\overline{PCS_0}$ is used to select the QPDM.

These outputs (A_1 and A_2) are latched throughout the cycle. The address bits and $\overline{PCS_0}$ become valid simultaneously so that the address lines are not guaranteed stable at the Am95C60 when chip select goes active. This means that the QPDM chip select must be delayed

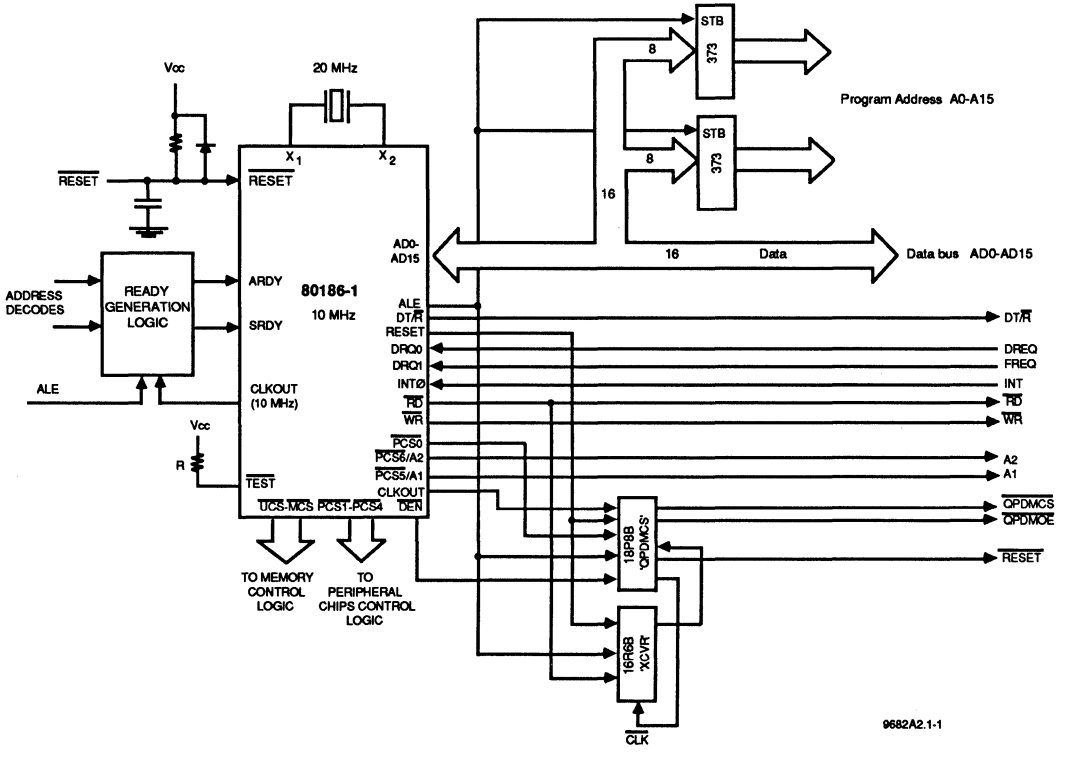


Figure 2.1-1 Single QPDM Schematic

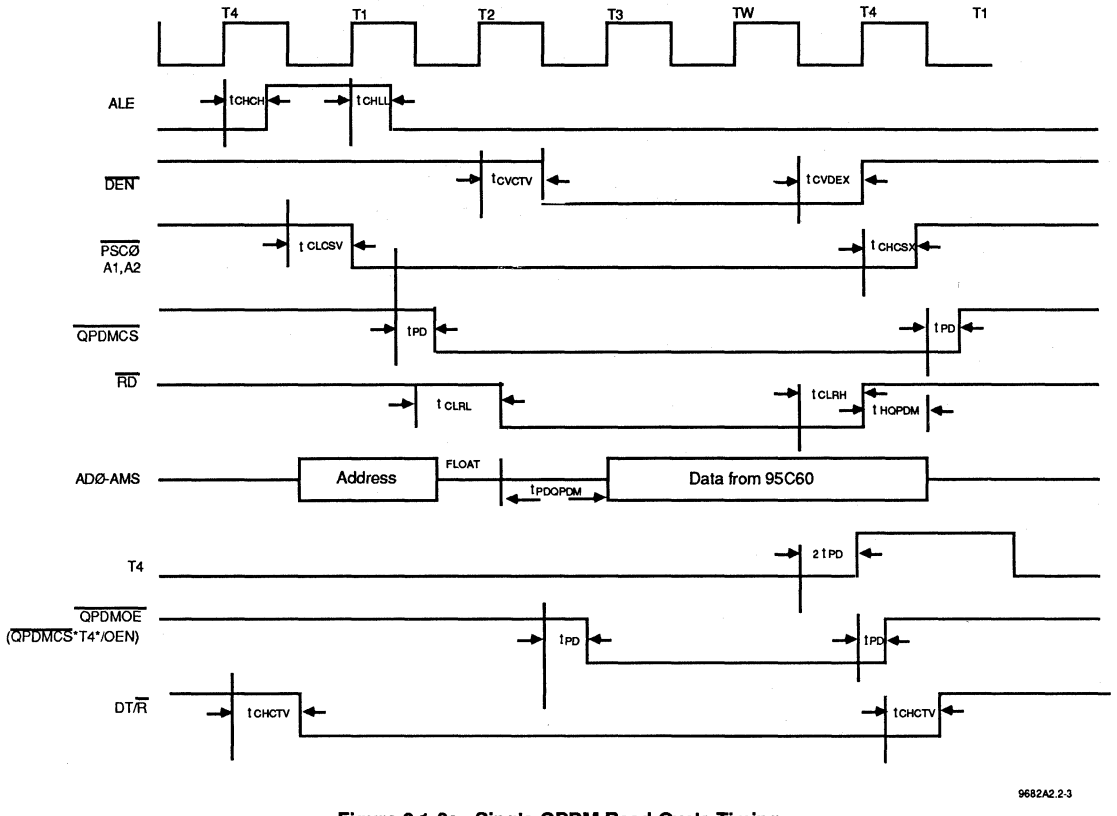


Figure 2.1-2a Single QPDM Read Cycle Timing

as a result of having to be qualified by ALE. This can be made more apparent by examining the PAL equations. The PAL device "QPDMCS" solves two problems: the set-up time of addresses to chip select, and the qualification of addresses with ALE. Figure 2.1-2a, "Single QPDM Read Cycle Timing", illustrates that after $\overline{PCS_0}$ goes active and addresses A_1 and A_2 are latched (via $\overline{PCS5}$ and $\overline{PCS6}$), the $\overline{PCS_0}$ to the QPDM is delayed by waiting for the falling edge of ALE. The $\overline{RD}$ signal generated by the 80186 is guaranteed active t_{CLRL} ns after the falling edge of T2. In the worst case, this is 56 ns . The data from the QPDM is guaranteed valid 80 ns after $\overline{RD}$ becomes active. The allowable read access time is

$$200 \text{ ns} - (t_{CLRLmax} + t_{DVCLmin})$$
$$= 200 - (55 + 15)$$
$$= 130 \text{ ns.}$$

The 80186 samples data on the falling edge of T_4. This means that data from the QPDM is presented to the 80186 in plenty of time to meet the 80186 set-up times of tDVCLmin (15 ns). The QPDM also guarantees that the read data will be held a minimum of 10 ns from the rising edge of $\overline{RD}$. This provides more than adequate hold time ($t_{CLDXmin}$ = 3 ns) for the processor. Figure 2.1-2a, "Single QPDM Read Cycle Timing", illustrates this quite clearly.

The only other parameter of concern during a read cycle is t_{RHAV} (not shown in the diagrams). This parameter is the minimum time from $\overline{RD}$ inactive until addresses are active for the next bus cycle. If memory or peripheral devices cannot disable their output drivers in this time, data buffers will be required to prevent both the 80186 and the peripheral or memory device from driving the data/address lines concurrently. In most designs a data transceiver is required due to the dc characteristics of the QPDM. This can be attributed to the CMOS I/O structures of the QPDM. To guarantee the design, a data transceiver is used. This will be the case in a multiple Am95C60 design as well.

With this in mind, we must now examine the implications of using a data transceiver. The parameter of interest here is the minimum time from $\overline{RD}$ inactive until the addresses become active for the next cycle, which has a minimum value of 60 ns for a 10 MHz 80186. This means

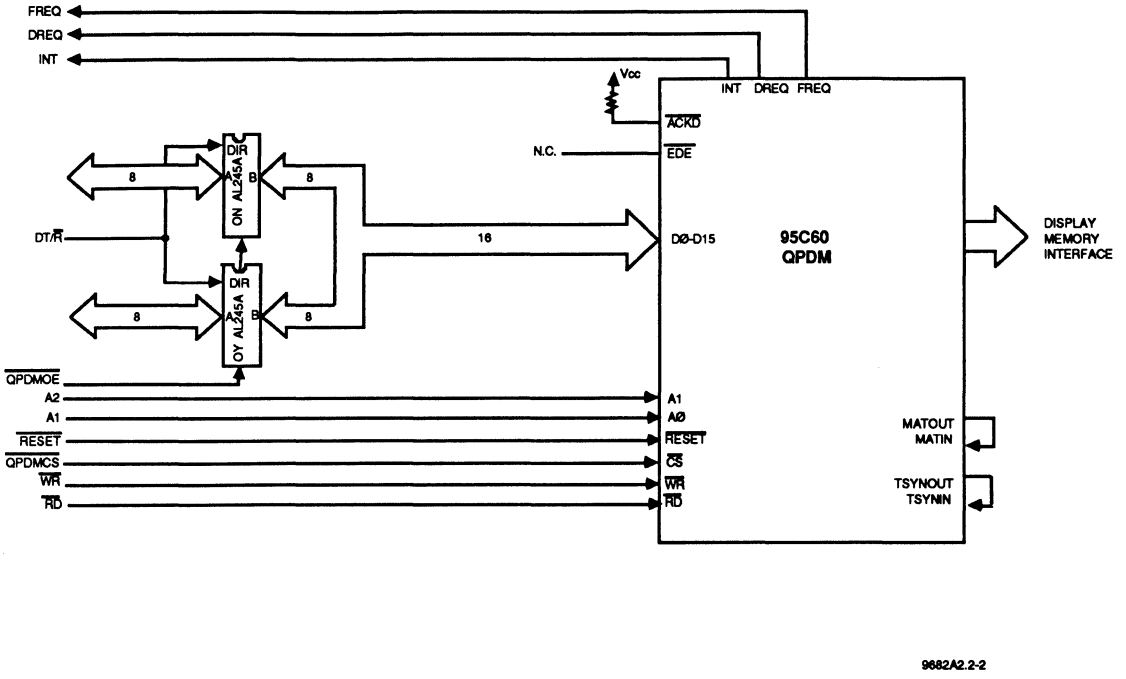


Figure 2.1-2b Single 95C60 Schematic

Single QPDM Read Cycle Timing Parameter List

	MIN (ns)	MAX (ns)
t_{CHCH} (80186-1)		30
t_{CHLL} (80186-1)		30
t_{CVCTV} (80186-1)	5	56
t_{CVDEX} (80186-1)	10	56
t_{CLCSV} (80186-1)		45
t_{CHCSX} (80186-1)	5	32
t_{PD} (B-Speed PAL)		15
t_{CLRL} (80186-1)	10	56
t_{CLRH} (80186-1)	10	44
t_{HQPDM} (95C60-20)	10	
t_{PQPDM} (95C60-20)		80
t_{CHCTV} (80186-1)	10	44

that the data and the associated driving data transceiver must be off the bus 60 ns after $\overline{RD}$ goes inactive. The $\overline{EDE}$ pin provided on the Am95C60 to control the output enable pin of the data transceivers goes inactive far too late in the read cycle to disable the transceiver and meet the tRHAV specification. The $\overline{DEN}$ signal of the 80186 can go inactive a t_{CVDEX} maximum of 56 ns after the falling edge of T_4. The minimum $t_{CLRH/RD}$ inactive delay is 10 ns.

If we add to this set of parameters the maximum t_{PLZ}

$$(t_{CVDEXmax} - t_{CLRHmin}) + t_{PLZ}$$
$$= 56 - 10 + 15$$
$$= 61 \text{ ns};$$

this already exceeds the t_{RHAV} spec of 60 ns.

The solution is to synthesize a signal from the existing processor signals that will allow us to turn off the transceivers after the falling edge of T_4 more quickly. The small state machine PAL device "XCVR" accomplishes this goal. Figure 2.1-3, "Transceiver Enable/Disable Timing", illustrates how the circuit works. An inverted $CLK_{(out)}$ clocks the entire state machine. The purpose of this state machine is to output a signal T_4 at the beginning of state T_4 of the 80186. ALE informs the circuit when state T_1 has occurred. This signal counts through until we get to state T_w. This is a wait state that is automatically inserted by the 80186 when we read from or write to the Am95C60. The reason for a wait state will become clear when the topic of DMA is discussed.

When the cycle reaches T_w, T_4 is also qualified by the signal $\overline{RD}$. If we are not reading the Am95C60, we do not generate signal T_4. In the case of a write cycle, we have no t_{RHAV} specification, and the CPU signal $\overline{DEN}$ is allowed to disable the data transceivers. More on this later.

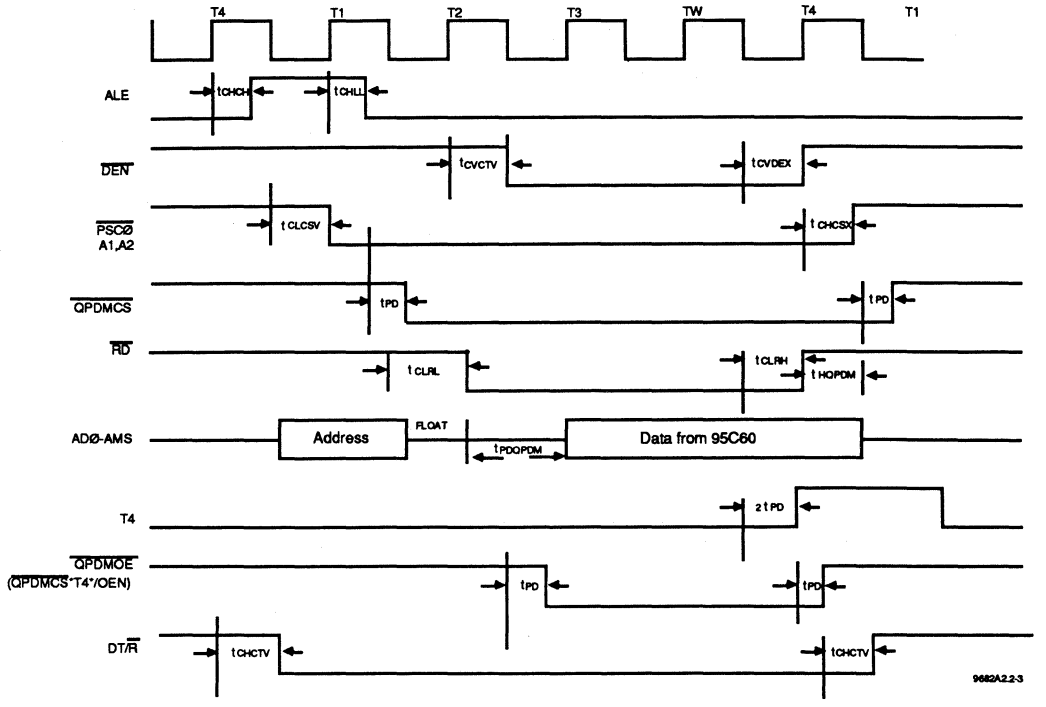


Figure 2.1-3 Transceiver Enable/Disable Timing

Transceiver Enable/Disable Timing

	MIN (ns)	MAX (ns)
t_{PD} (B-Speed PAL)		15
t_{CHLH} (80186-1)		30
t_{CHLL} (80186-1)		30
t_{CLRL} (80186-1)	10	56

The bottom line is that if a read cycle is occurring, the data transceivers are disabled within

$$3 \cdot t_{PDmax} + t_{PLZmax} = 3(15) + 15 = 60 \text{ ns.}$$

This meets the t_{RHAV} specification. The PAL equations are given in the listing XCVR for closer examination. Since the CPU data hold time t_{CLDX} (8 ns min) and the T_4 transceiver turn off delay are relative to the same clock edge (falling edge of T_4), and if we factor in the transceiver delay, the hold time at the processor is guaranteed. The $\overline{DEN}$ turn-on delay allows:

$$2 \cdot t_{CLCL} + t_{CHLLmin} - t_{CVCTVmax} - t_{DVCL}$$
$$= 200 + 44 - 44 - 15$$
$$= 185 \text{ ns}$$

transceiver enable time prior to valid data required at the CPU. The PAL outputs, $\overline{QPDMCS}$ and $\overline{QPDMOE}$, use 15 ns maximum of this time to enable the transceiver, and since the Am95C60 places data in the bus a maximum of 80 ns from the active edge of $\overline{RD}$, the data will be present in plenty of time to meet the processor set-up time.

The DT/$\overline{R}$ signal is used to control the direction of the flow of the transceiver. The timing of this signal is no cause for concern.

Write Cycle

The write cycle of the 80186 is very similar. The timing is shown in Figure 2.1-4, "Single QPDM Write Cycle Timing". The $\overline{WR}$ signal is guaranteed active t_{CVCTV} ns from the falling edge of T_2 and inactivated t_{CVCTX} ns from the falling edge of T_4. The QPDM requires a minimum $\overline{WR}$ pulse width of 70 ns, the data written to the QPDM must be valid at least 50 ns from the rising edge of $\overline{WR}$, and the data must have a finite hold time. The chip select timing is identical to a $\overline{RD}$ bus cycle. The worst case pulse width of the $\overline{WR}$ is

$$3 \cdot t_{CLCL} - t_{CVCTXmin}$$
$$= 300 - (44 + 5)$$
$$= 251 \text{ ns}$$

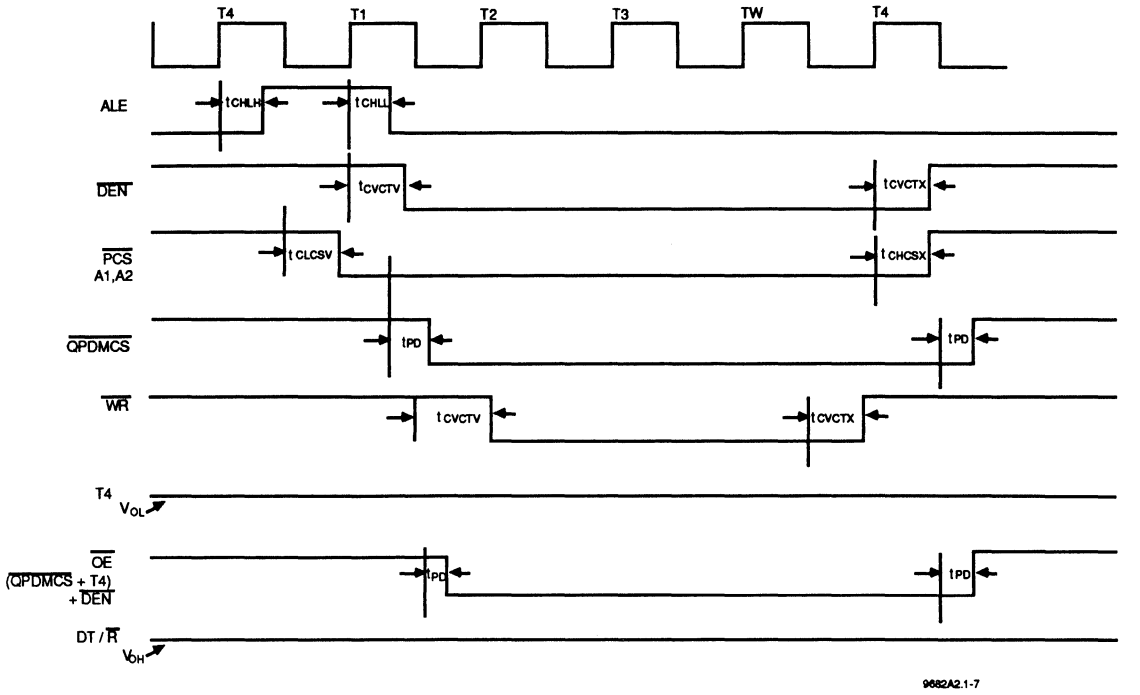


Figure 2.1-4 Single QPDM Write Cycle Timing

Single QPDM Write Cycle Timing Parameter List

	MIN (ns)	MAX (ns)
t_{CHLH} (80186-1)		30
t_{CHLL} (80186-1)		30
t_{CVCTV} (80186-1)	10	44
t_{CVCTX} (80186-1)	5	44
t_{CLCSV} (80186-1)		45
t_{CHCSX} (80186-1)	5	32
t_{PD} (B-Speed PAL)		15
t_{CVCTV} (80186-1)	5	56
t_{CVCTX} (80186-1)	5	44
t_{WQPDM} (95C60-20)	70	
t_{HQPDM} (95C60-20)	0	
t_{SQPDM} (95C60-20)	50	

This substantially exceeds the minimal 70 ns required by the QPDM. The write data is driven by the 80186 a maximum of 40 ns after the falling edge of T_2. The data transceivers are enabled a maximum of

$$t_{CHLLmax} + 2t_{PDmax}$$
$$= 30 + 2(15)$$
$$= 60 \text{ ns}$$

from the rising edge of T_1. Therefore, the data will be presented to the QPDM in plenty of time to meet the set-up time of 50 ns to the rising edge of $\overline{WR}$ (which occurs t_{CVCTX} ns from the falling edge of T_4). The 80186 guarantees a data hold time of $t_{WHDXmin}$ after the rising edge of

$$\overline{WR}\ t_{CLCL} - 34$$
$$= 100 - 34$$
$$= 66 \text{ ns}.$$

Another point to examine is that in this case we allow the $\overline{DEN}$ signal to disable the data transceivers by itself. This is because we don't have the tight t_{RHAV} specification present in the read cycles. Examining the PAL equations in QPDMCS makes this point clearer. $\overline{DEN}$ is disabled a minimum of

$$t_{CLCHmin} + t_{CVCTXmin} - t_{CVCTXmax}$$
$$= 44 + 5 - 44$$
$$= 5 \text{ ns}$$

after $\overline{WR}$ inactive. This, in combination with the delay to turn off the transceivers, ensures that we meet the hold time of the QPDM in relation to the rising edge of $\overline{WR}$. Also, since this last equation uses a minimum t_{CVCTX} with a maximum t_{CVCTX}, the hold time will be longer.

DMA

QPDM DMA requests pose no special problem to the 80186. In fact, since the 80186 built-in DMA controller looks to the QPDM as a flow-through type, the interfacing is quite simple. The 80186 DMA cycles appear as normal processor read or write cycles to the QPDM. These types of read and write cycles have been covered in the preceding paragraphs. The only issue left to be considered is the choice of source or destination synchronized DMA transfers.

When the QPDM requires that large quantities of data be down loaded, destination synchronized transfers must be issued. In destination synchronized DMA transfers, the destination of the DMA data requests the DMA transfer. In this type of transfer, the QPDM is written to during the deposit cycle of the DMA transfer. The only parameter requiring special concern is the DMA request signal inactive time. To prevent unwanted DMA transfer cycles, the DMA requesting device must drop its DMA request at least two clock cycles before the end of the deposit cycle, regardless of the number of wait states inserted into the bus cycle. With a 10 MHz processor clock, the value for DRQ inactive from the start of T_2 (assuming no wait states) is

$$t_{CLCL} - t_{INVCLmin}$$
$$= 100 - 20$$
$$= 80 \text{ ns.}$$

Examining the QPDM specifications, DREQ and FREQ become inactive 50 ns maximum after $\overline{WR}$ to the QPDM. We have seen previously that the $\overline{WR}$ goes inactive 56 ns maximum after the falling edge of T_2. This is a total of 56 + 50 = 106 ns maximum after the falling edge of T_2, which means that in order to avoid unwanted DMA cycles, we must insert a single wait state into the cycle. The wait state provides an additional 100 ns so that the DRQ inactive time becomes 100 + 80 = 180 ns. Since DREQ or FREQ goes inactive a maximum of 106 ns, the 180 ns DRQ inactive time is more than adequate, and a single wait state is all that is required. No extra circuitry is required to insert this wait state, as we shall cover more fully in a later section.

When the QPDM has data to be transferred out via DMA, the DMA can be programmed to source synchronized mode. In a source-synchronized DMA transfer, the QPDM requests DMA transfer, and the QPDM is read during the fetch cycle of the DMA transfer. Please note that the source or destination synchronized transfer modes are selected by programming bits in the peripheral control register block internal to the 80186. This allows the user to change the mode of the DMA controller via software or on the fly. This means that we can edit the appropriate transfer mode for the QPDM depending on the transfer direction required. To ensure that DMA transfers do not occur when it is not desired, the DRQ signal must be driven inactive before the falling edge of T_1 in the deposit cycle. This does not pose a problem because the QPDM will de-activate DREQ or FREQ 50 ns maximum after $\overline{RD}$ to the QPDM. This occurs

$$t_{CLRLmax} + 50 \text{ ns}$$
$$= 56 + 50$$
$$= 106 \text{ ns}$$

after the falling edge of T_2, well before the falling edge of T_1 in the deposit cycle.

There are three other considerations regarding DMA in general. First, the DREQ and FREQ DMA request pins are open-drain and must be pulled up to V_{cc} with resistors. Second, $\overline{ACKD}$ is not used in this design and must also be pulled up. Third, the 20-bit source and destination pointers allow access to the complete 1M byte address space of the 80186, but when addressing I/O space, the upper four bits of the DMA pointer registers should be programmed to be 0. Otherwise, the programmed value (greater than 64K in I/O space) will be driven onto the address bus (an area of I/O space not seen by the processor). This could cause chip selection problems in any external logic that the user may wish to add to the design.

Interrupts

The 80186 contains an integrated interrupt controller. Four external interrupt pins are available for use. If no more than four external interrupt sources are required, no external interrupt controller is needed. When using the internal interrupt controller, the interrupt types are fixed and cannot be changed. In response to an interrupt, the processor will jump to the vector address associated with the interrupt type. The addresses of the interrupt routines are stored in the interrupt vector table in low memory. These addresses are user supplied and controlled. On the 80186, the interrupt vector address is the interrupt type (or number) multiplied by four. This speeds up the interrupt response greatly, because no external bus cycles are required to fetch the interrupt types. Consult the 80186 data sheet for the vector types associated with the four external interrupt pins. The user can connect the QPDM INT pin to any of the four external INT pins of the 80186 according to the design requirements; INT_0 was chosen arbitrarily in this design. Please note that the execution of writes to the Interrupt Acknowledge register of the QPDM is used to clear interrupt requests. These steps should be an integral part of all QPDM interrupt service routines. A "1" must be programmed in the word for each interrupt that is to be cleared. A "0" bit has no effect. When all enabled interrupt requests have been acknowledged and cleared, the INT signal goes inactive. Consult the QPDM Technical Manual for further details.

Miscellaneous

The peripheral chip select lines $\overline{PCS}_5$ and $\overline{PCS}_6$ have been programmed to provide latched address lines A_1 and A_2. This is accomplished by programming the PACS and MPCS registers in the peripheral control block. These two latched address pins are connected to pins A_0 and A_1 of the QPDM and are used to access the internal registers of the QPDM. All the internal QPDM registers will appear at even addresses to the 80186.

In this design, $\overline{PCS}_0$ is used to control $\overline{CS}$ of the QPDM. Each $\overline{PSC}_x$ line is active for one of seven contiguous 128 byte areas in memory space or I/O space above a programmed base address. Consult the PCS Address Ranges Table in the 80186 data sheet for the details regarding address partitioning. As stated earlier, the peripheral chip selects are controlled by two registers in the internal-peripheral control block of the 80186. These registers allow the base address of the peripherals to be set and allow the user to determine whether the addresses will be in memory space or I/O space. Both registers must be programmed by the user before the chip selects become active.

The 80186 includes a ready generation unit. This unit generates an integral ready signal for all accesses to memory or I/O addresses to which the internal chip select circuitry responds. For each chip select, 0-3 wait states may be inserted by the internal unit. Also, the ready generation circuits can be programmed to ignore the state of the external ready pins. In this case, only the internal ready state will be used by the processor. The ready generation circuit can also be programmed to respond to the external ready signal. This means that the ready circuitry will perform a logical AND function of the external and internal ready states and a ready will be provided only after *both* are true. In this QPDM design, the user may program the R_0-R_2 bits in the PACS register for one wait state with no external ready required. Bits R_0-R_2 of the MPCS register control the ready generation for $\overline{PCS}_{4\text{-}6}$. Bits R_0-R_2 of the PACS register specify the ready mode for $\overline{PCS}_{0\text{-}3}$. Bit 7 of the MPCS register is used to select whether the peripheral chip select lines are mapped into memory or I/O space. After reset, the contents of both the MPCS and PACS registers are undefined; however, none of the $\overline{PCS}$ lines will be active until *both* the MPCS and PACS registers are accessed. Also on reset, only $\overline{UCS}$ (upper chip select) is active. It is programmed by reset to be active for the top 1K memory block, to insert 3 wait states to all memory fetches, and to factor external ready for every memory access. Therefore, some kind of circuit must be included to generate an external ready until the ready generation logic is reprogrammed not to factor in external ready.

In this design, the lower 16 address lines are latched. This is done because the integrated chip selects perform the selection between the various memories and peripherals. Therefore, the upper four address bits can be ignored. The usage of these upper four bits will probably vary from design to design.

2.1.2 Multiple QPDM Design

In this section, an extension of the previous design is discussed. A 16 bit plane, four QPDM system design is illustrated (Figure 2.1-5). First, a discussion on multiple QPDM operation is in order.

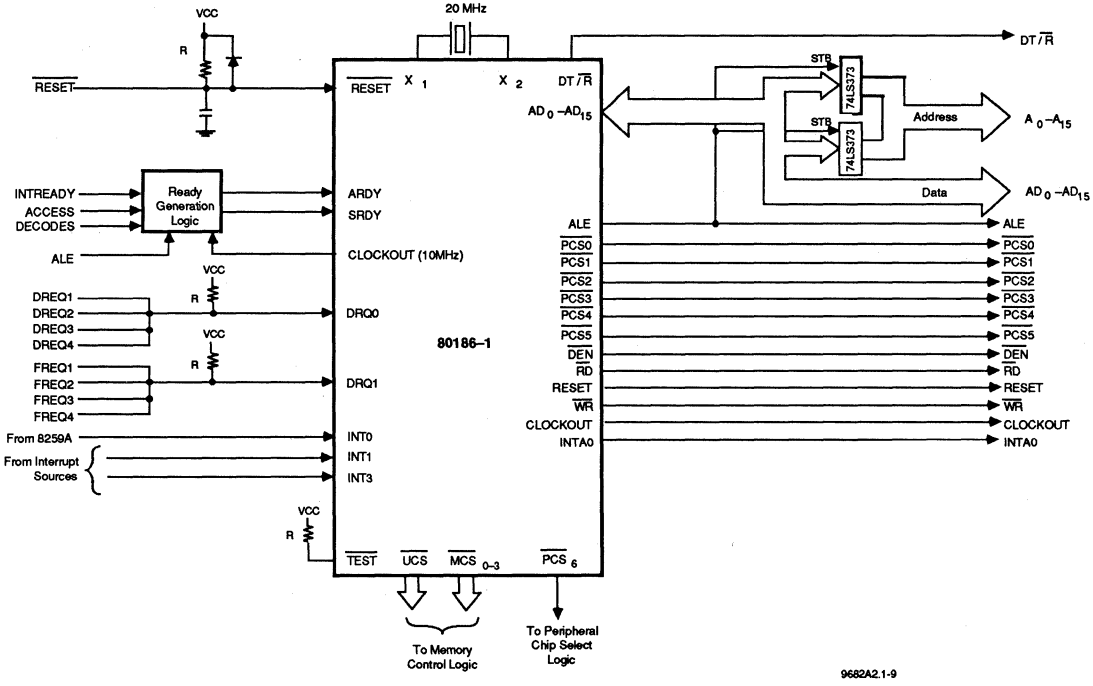
Multiple QPDM Design Considerations

Initialization and Broadcast

Since each QPDM handles up to four bit planes, a 16 plane system will require four QPDM devices. In general, all QPDMs are given each instruction simultaneously with identical parameters, so that the instruction can update each plane. In some cases, however, a means to differentiate between QPDMs and some planes within a single QPDM is necessary. To facilitate individual QPDM plane operation, each QPDM is assigned a QPDM number. This number is loaded into each QPDM using the "Set QPDM Position" instruction. Each QPDM must be assigned a unique number, via four separate set QPDM position instructions; one instruction is executed per QPDM. This means that it is necessary to provide chip select ($\overline{CS}$) decoding for *each* individual QPDM and for *all* QPDMs as a group. In this design we would need five separate $\overline{CS}$ signals. The "Quad QPDM Chip Selects" shows the relationship between $\overline{PCS}_x$ and the QPDM table chip selects in this example.

Quad QPDM Chip Selects

Peripheral Chip Select	QPDM(s) Selected
PCS_0	QPDM1
PCS_1	QPDM2
PCS_2	QPDM3
PCS_3	QPDM4
PCS_4	ALL (QPDM1-QPDM4)



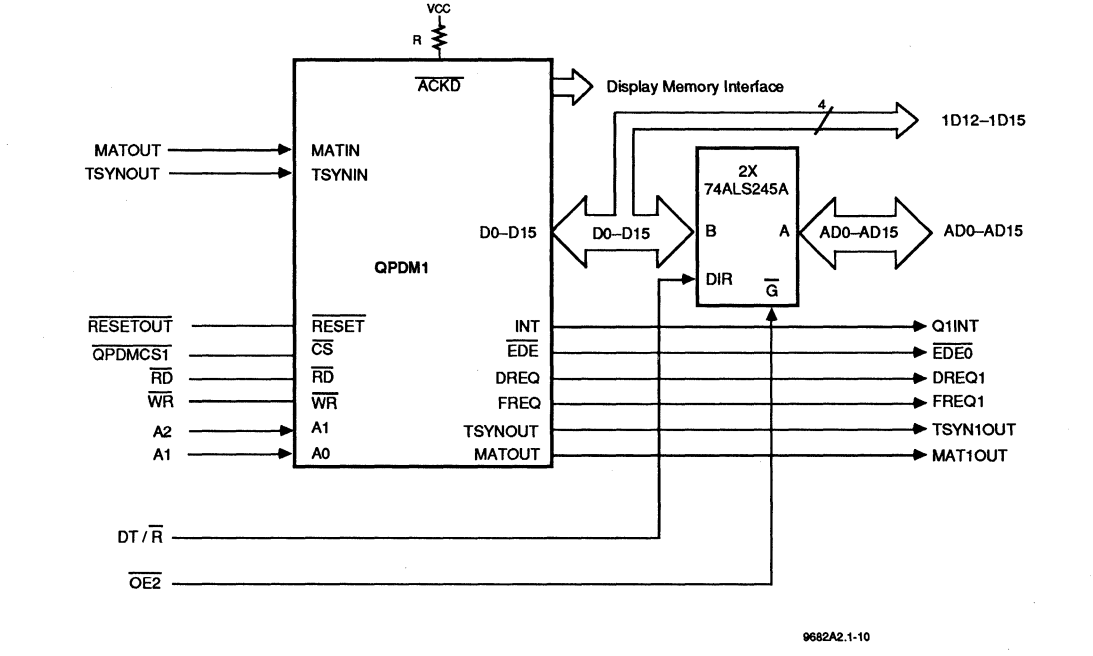
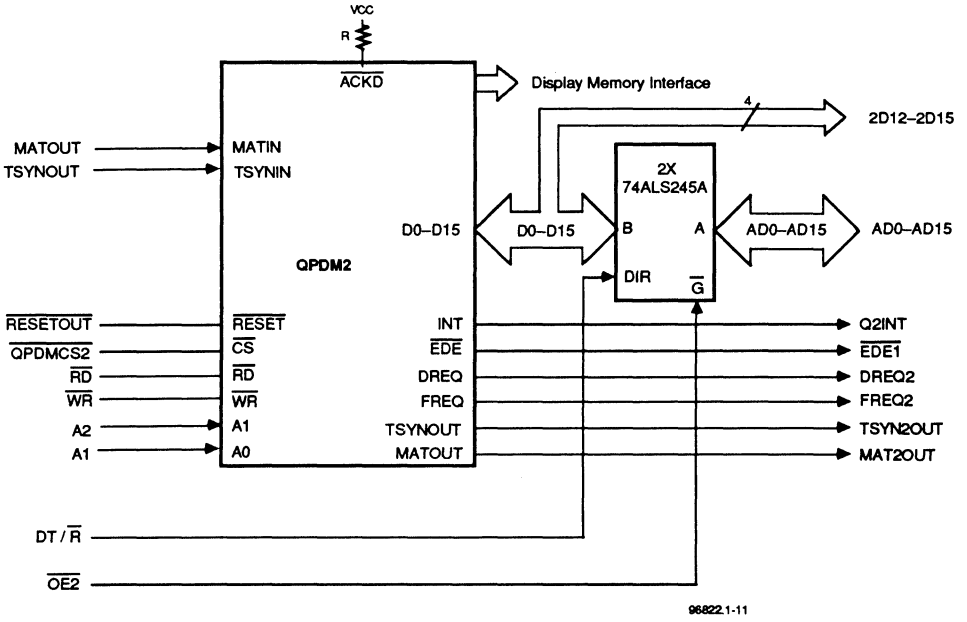


Figure 2.1-5 Quad QPDM Schematic



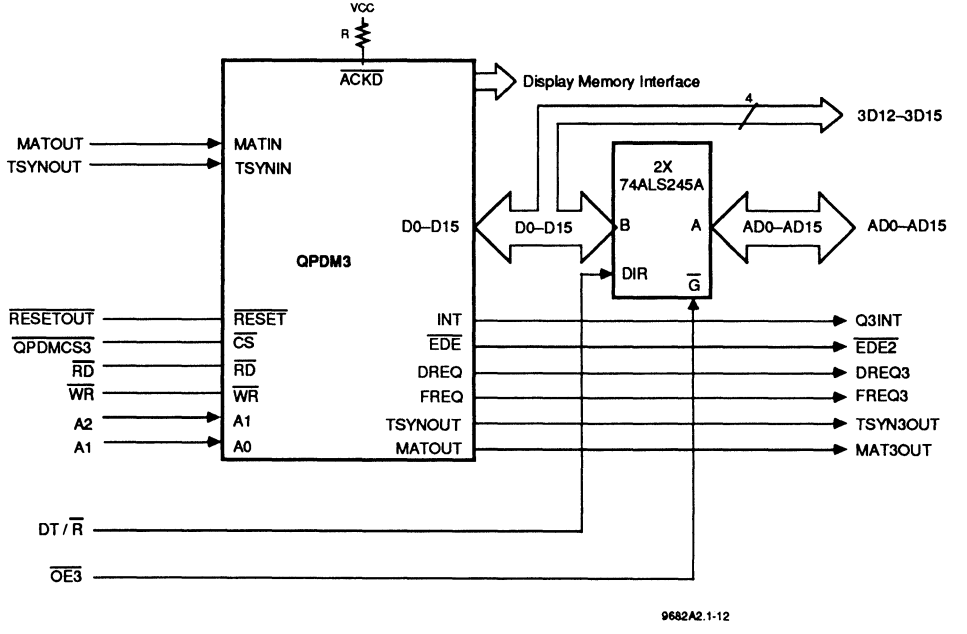
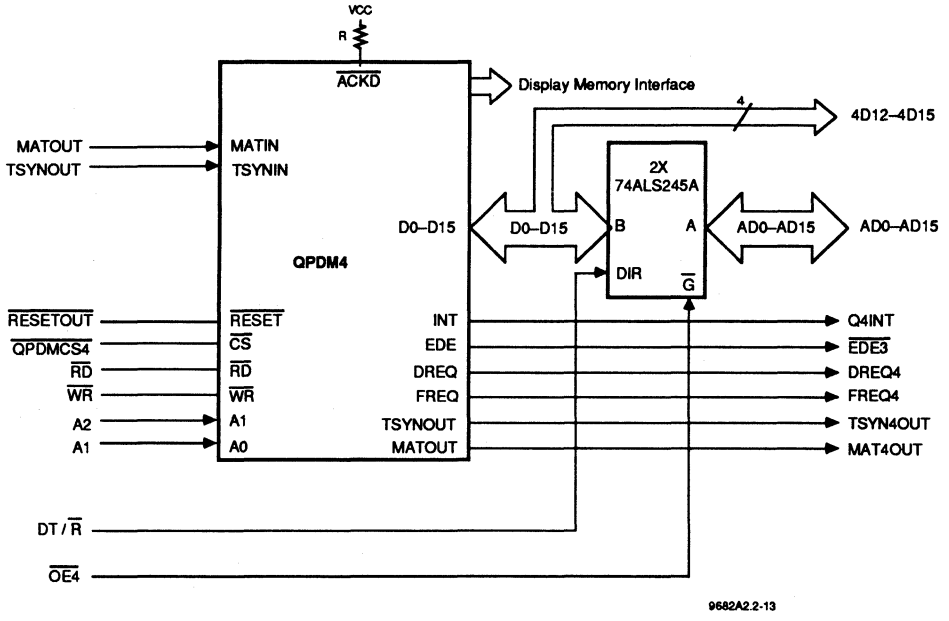


Figure 2.1-5 Quad QPDM Schematic (continued)



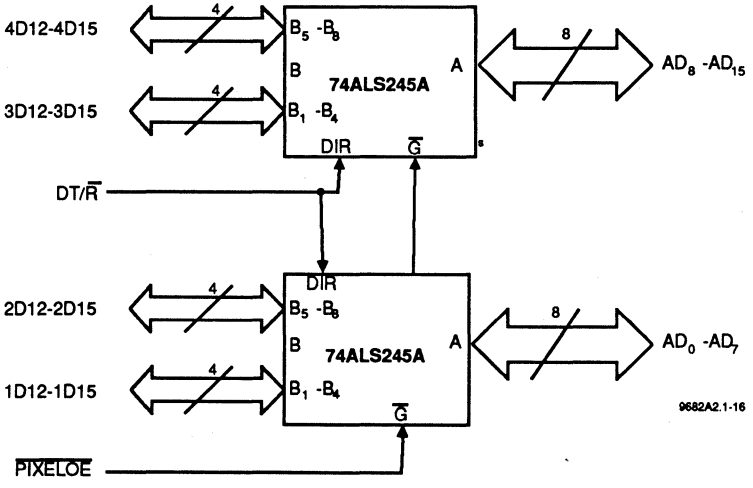


Figure 2.1-5 Quad QPDM Schematic (continued)

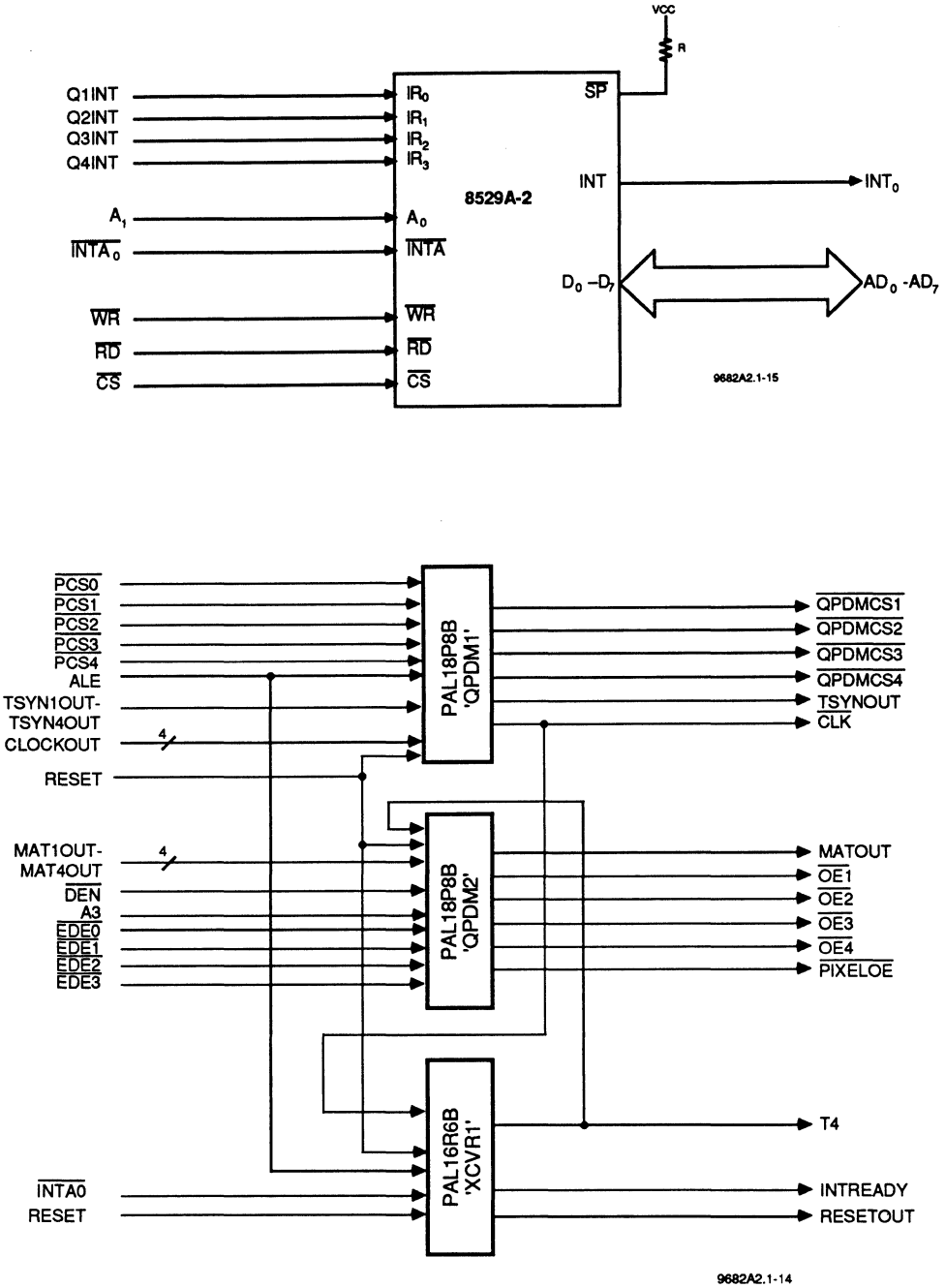


Figure 2.1-5 Quad QPDM Schematic (continued)

QPDM instructions are sent to each QPDM simultaneously. This is called broadcasting. Broadcasting of instructions is accomplished by chip selecting all QPDMs and writing to Port 0. Further details can be found in the QPDM Technical Manual. The hardware requirements for QPDM initialization and for broadcasting are a mechanism for individual chip selection as well as a global chip select.

DMA Requests

DMA with a multiple QPDM system is fairly straightforward. Both FREQ and DREQ are open-drain outputs. Each QPDM in a system will release its FREQ or DREQ when it is ready so that a DMA request will be presented to the 80186. The FREQ controls DMA to the Instruction FIFO and DREQ controls DMA to and from the Data FIFO of the QPDM. In a multiple QPDM system, all the FREQ lines of the QPDMs are tied together through a pull-up resistor to DRQ_1 of the 80186. Similarly, all the DREQ lines are tied together through a pull-up resistor to the DRQ0 input of the CPU. In the case of a data transfer (DREQ), as long as any QPDM is not ready to continue with the transfer, the node will be pulled LOW. When the last QPDM becomes ready, the node goes HIGH and a DMA request will be seen at the 80186. The same is true of the instruction FIFO and the FREQ pins. As long as any QPDM is not ready to request more instructions, the node goes LOW. When the last QPDM becomes ready, the node will be pulled HIGH through the pull-up resistor and an instruction DMA request will be seen at the DRQ_1 pin of the CPU.

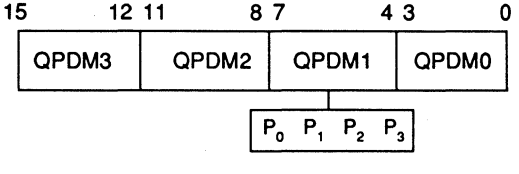
To summarize, one can see that all DMA to and from multiple QPDMs are synchronized by the open-drain AND connection. The open-drain AND ensures that all DREQ and FREQ requests of the QPDMs are active before the DMA request is seen at the processor. Above and beyond the previous discussion, the DMA cycles are the same as outlines in the single QPDM design.

Block Transfer to and from Display Memory

In the multiple QPDM design, provisions have been made for display memory accesses by plane (horizontal on a selected bit plane) and by pixel (reading all bits in all planes per pixel). In a by plane Block I/O instruction, only one plane in a single QPDM is selected for a read or write. In the read case, a provision must be made to keep all QPDMs in synchronization. To do this, the processor fakes a read operation on all the QPDMs. The inactive QPDMs (do not contain the bit plane accessed) leave $\overline{EDE}$ not active, so as not to cause contention with the data of the only active QPDM. The active QPDM, the one with a plane active for the instruction, places data on the bus as in a normal read cycle. In large systems, however, we need external data transceivers. This is where the $\overline{EDE}$ (External Driver Enable) pin comes in handy. For the inactive QPDMs, the $\overline{EDE}$ pin does not go active to their corresponding bus transceivers. Only the active QPDM drives its $\overline{EDE}$ signal valid to its bus transceiver to allow its data onto the bus. Therefore, by adding a little more intelligence to the QPDM interface, synchronization is achieved and maintained even when only one bit plane in a single QPDM is to be accessed.

In a by pixel Block I/O, the user wishes to access all bits in all planes per pixel. This implies that all QPDMs contribute the bits of the planes that they control for that particular pixel. Synchronization, therefore, is not a problem, as all QPDMs will be active. In a multiple QPDM system, an extra set of transceivers must be provided to route the pixels from each QPDM to a single 16-bit data bus. These are shown on Sheet 8 of the Quad QPDM Schematic. For example, in this design each QPDM contributes four bits for each individual pixel. A mechanism has been provided in the QPDM to program the number of shifts necessary for the proper assembly of pixels. This is done by specifying the correct number (1, 2 or 4) dependent on the number of QPDMs in the system in the BOS field of the Input or Output Block instructions. To see how this works examine Figure 2.1-6, "By Pixel Read". On the first pixel's read cycle, each QPDM places 16 bits on its respective data bus, only four of which will be used. A 16-bit transceiver concatenates four bits from each QPDM to form a 16-bit data value. Following each cycle, each QPDM shifts its data four bits to the left. In this way the next pixel's four bits of data are positioned correctly in the data bus to be assembled into the 16-bit value at the transceiver. This process continues until all 16-bit values for the selected number of pixels have been read. From a hardware standpoint, all that is required is an extra transceiver and extra decoding logic to selectively enable or disable the set of transceivers depending upon the type of access (by plane or by pixel).

From a software point of view it is best if planes appear in consistent bit positions. What this means in hardware terms is that some consideration must be given to how the data bits of the "by pixel" transceiver are connected. The relationship between data bits and planes is fixed by the organization of operands in the Set Activity Bits instruction. Each of four QPDMs extract four bits as shown below:



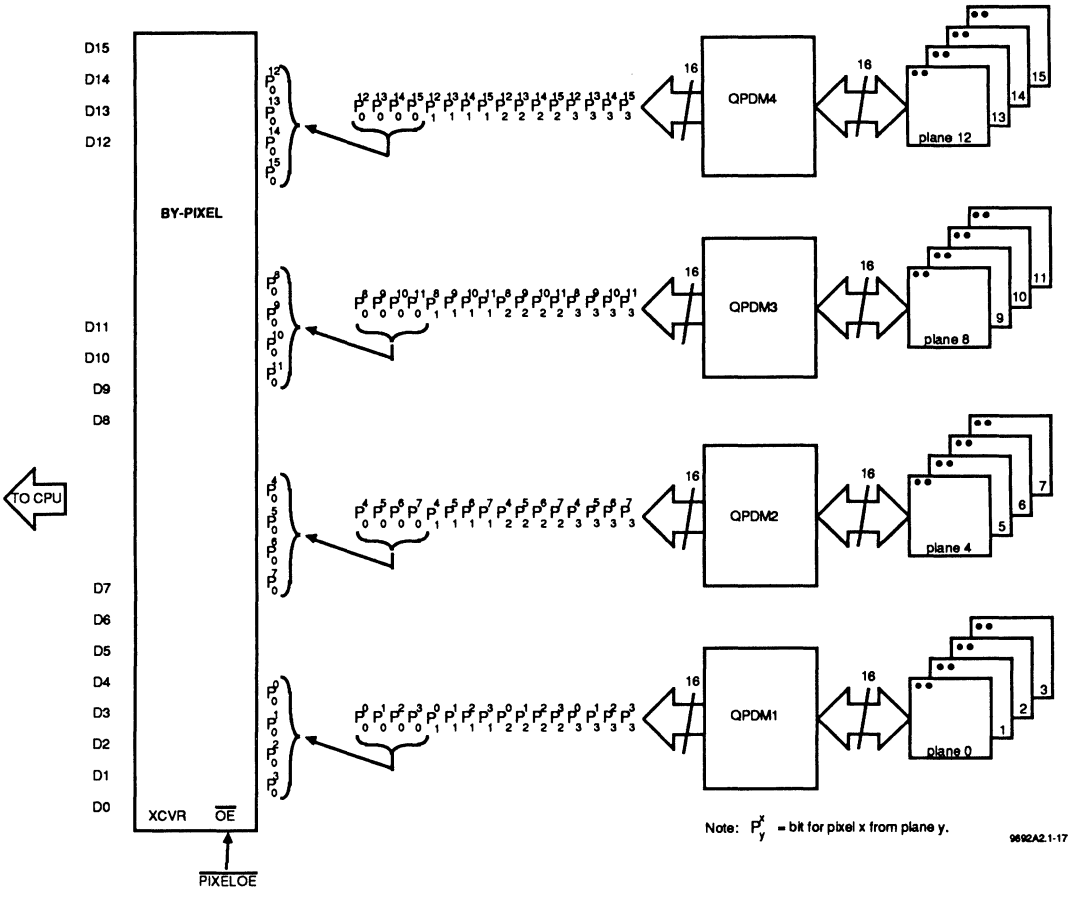


Figure 2.1-6a By Pixel (Output Block) Read - First Cycle

There is no reasonable way to change this relationship; everything else will have to match. To make I/O by-pixel conform, one must wire the by-pixel data transceiver as shown in Figure 2.1-6, "By Pixel Read". Also see Chapter 13 of the QPDM Technical Manual, "The Relationship Between Data Bits and Pixels." It should be noted here that only pixels that are 16 planes or less can be processed in one pass; deeper bit planes would require multiple passes.

Other Synchronization Concerns

The synchronization of getting new words out of the instruction FIFO and reading and writing the data exchange FIFOs use the MAT_{IN} and MAT_{OUT} pins. The MAT_{OUT} pins of all the QPDMs are ANDed together. This composite signal goes HIGH only when all the QPDMs are ready to begin the next instruction. All QPDMs sample the composite MAT_{OUT} signal at the MAT_{IN} input. From an interface standpoint, all that is required is to AND the MAT_{OUT} pins and connect the composite signal to the MAT_{IN} pins. The AND is done in PAL device QPDM2.

In a similar fashion, the $TSYN_{OUT}$ and $TSYN_{IN}$ pin combination is used to synchronize the bit map display memory bus activities. The same hardware considerations discussed in the preceding paragraphs apply. The AND is done in PAL device QPDM1. Please note that even in a single QPDM system MAT_{IN} and MAT_{OUT} must be tied together, as are $TSYN_{OUT}$ and $TSYN_{IN}$.

2.1.3 Hardware Overview

Read Cycles

The multiple QPDM read is shown in Figure 2.1-7, "Quad QPDM Read Timing". The situation is sightly more

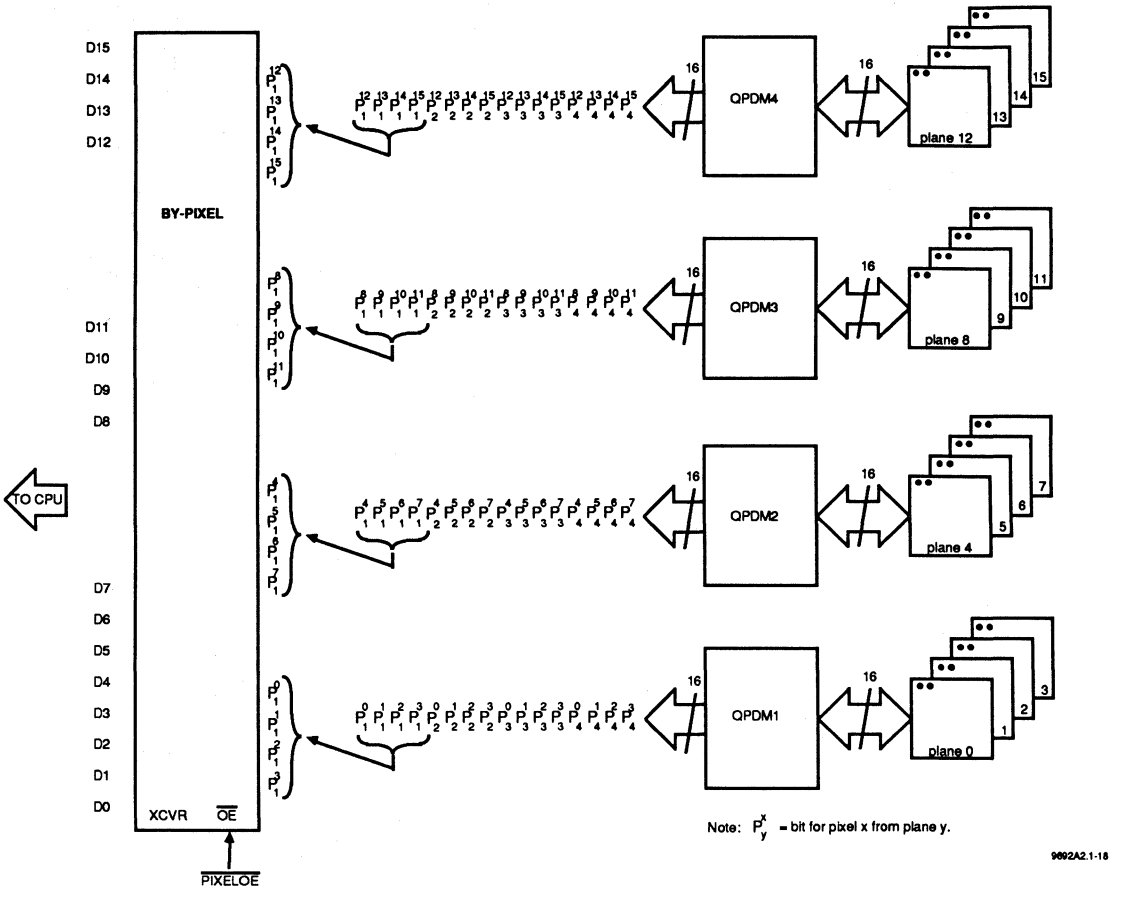


Figure 2.1-6b By Pixel (Output Block) Read - Second Cycle

complex than in the single QPDM case. We must modify the transceiver PAL device logic to factor in the $\overline{EDE}$ signal discussed previously. The $\overline{EDE}$ signal from the selected QPDM is guaranteed active a maximum of 50 ns after $\overline{CS}$ to the selected QPDM goes active. This $\overline{EDE}$ signal can be active a maximum of

$$t_{CHLLMAX} + t_{PDMAX} + t_{PHLMAX}$$
$$= 70 + 15 + 50$$
$$= 135 \text{ ns}$$

from the rising edge of T_1. $\overline{DEN}$ can go active a minimum of 5 ns from the rising edge of T_2. This means that factoring in the $\overline{EDE}$ signal into the PAL equations can delay $\overline{OE}$ to the transceiver a maximum of 30 ns. Note that again we are mixing minimum and maximum parameters of the 80186. In practice, the delay will be less than the calculated 30 ns . Remember from the single QPDM design that we had 185 ns transceiver enable time prior to valid data required at the CPU. With the factoring in of the QPDM $\overline{EDE}$ signal, we have reduced this figure to a minimum of 155 ns . The PAL device circuit uses 15 ns maximum of this time, so that 140 ns still remains to enable the transceivers. Since the QPDM places data on its bus a maximum of 80 ns from the falling edge of read, we can see that the factoring in of the $\overline{EDE}$ signal really does not change things all that much. We still use the T_4 signal to disable the transceivers and meet the t_{RHAN} specification of 60 ns. Examination of the PAL equations, QPDM1 and QPDM2, will clarify these points.

Write Cycles

The write cycles remain the same as in the single QPDM design. Once the QPDM(s) have been chip selected, the write cycles are identical to the single QPDM case.

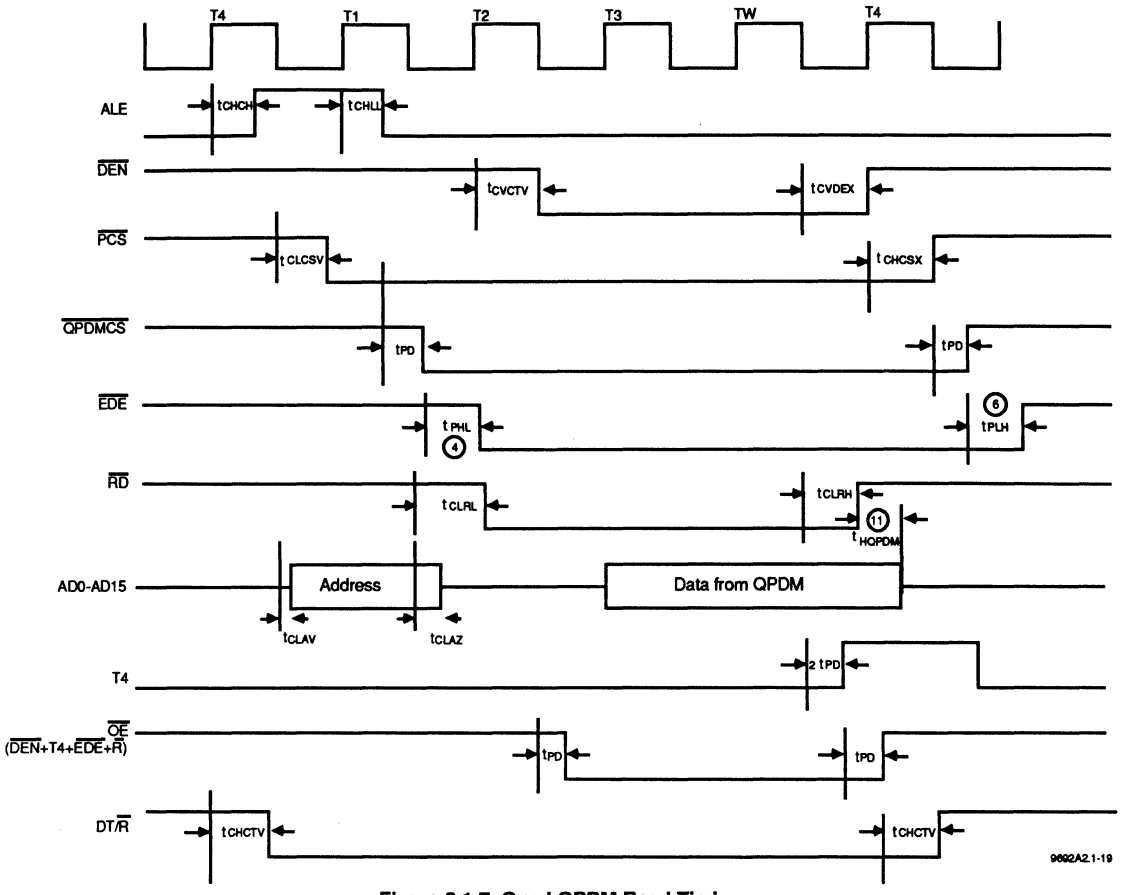


Figure 2.1-7 Quad QPDM Read Timing

Quad QPDM Read Cycle Parameter List

	MIN (ns)	MAX (ns)
t_{CHLH} (80186-1)		30
t_{CHLL} (80186-1)		30
t_{CVCTV} (80186-1)	5	56
t_{CVDEX} (80186-1)	10	56
t_{CLCSV} (80186-1)		45
t_{CHCSX} (80186-1)	5	32
$t_{PHLQPDM}$ (95C60-20)		50
$t_{PLHQPDM}$ (95C60-20)		65
t_{CLRL} (80186-1)	10	56
t_{CLRH} (80186-1)	10	44
t_{CLAV} (80186-1)	5	50
t_{CLAZ} (80186-1)	t_{CLAX}	30
t_{PDQPDM} (95C60-20)		80
t_{HQPDM} (95C60-20)	10	
t_{PD} (B-Speed PAL)		15
t_{CHCTV} (80186-1)	10	44
t_{CHCTV} (80186-1)	10	44

Chip Select Logic

This design uses $\overline{PCS}_{0-4}$ to enable one or all of the chip selects of the individual QPDMs. Note that in this design the address pins A_1 and A_2 are not latched via $\overline{PCS}_5$ and $\overline{PCS}_6$. This means that $\overline{PCS}_5$ and $\overline{PCS}_6$ are free to select other peripheral devices. The address information required by the QPDM is now latched in the Am74LS373 latches and connected directly to the A_0 and A_1 inputs. Note again that the A_2 and A_1 CPU address pins have been connected to the A_0 and A_1 pins of the QPDM such that the internal registers appear at even address multiples.

The $\overline{PCS}_x$ pins still need to be qualified by ALE to ensure that valid address data is present at the A_0 and A_1 pins of the QPDMs before a chip select goes active. ALE is inactive 30 ns maximum from the rising edge of T_1, which means that the qualified QPDM chip selects are active a t_{PD} maximum of 15 ns later. This ensures valid ad-

dresses at the QPDM, as addresses are guaranteed valid by the CPU at least 20 ns before the falling edge of ALE.

To implement a global chip select to the QPDMs, $\overline{PCS_4}$ was chosen. Whenever the user's code wishes to broadcast to all QPDMs, the I/O or memory addresses used will correspond to the fifth 128-byte area above the programmed base address (PBA) in the PACS register of the 80186. The logic to generate both single and global chip selects is easily implemented in a PAL device. Examine the QPDM1 PAL device listing. Whenever $\overline{PCS_4}$ and $\overline{ALE}$ are active, all the chip selects from the QPDMs will be active.

When the user wishes to access only a single QPDM, the address block assigned to the chip select for that particular QPDM is used.

Block I/O "By Pixel" Control

In this design, two extra transceivers are included to allow the user to read all 16-bit planes controlled by all four QPDMs on a pixel-by-pixel basis. This means that some differentiation is necessary between the regular I/O transceiver and the "By Pixel" transceivers. This is accomplished by using address line A_3 to enable the extra transceivers. To read data on a "By Pixel" basis using a block I/O QPDM instruction, the user addresses the global bank of addresses ($\overline{PCS_4}$), with address line A_3 active HIGH. This chip select enables all the QPDMs so that each can contribute up to four bits to the 16-bit word to be read by the CPU. The assembly of these 4-bit nibbles occurs at the "By Pixel" data transceivers. These two transceivers are only enabled when address line A_3 is active.

The same holds true for CPU writes on a "By Pixel" basis. In this case the 16 bits written by the CPU are disassembled by the "By Pixel" transceivers and four bits are presented to each QPDM to be inserted into the bit-map for the selected pixel.

Individual QPDM accesses with address line A_3 LOW only enable the transceivers associated with the selected QPDM. Global QPDM accesses with address line A_3 LOW enable only the eight transceivers associated with regular QPDM accesses. The "By Pixel" transceivers are not enabled. Therefore, with a little extra decoding logic, the user is given the capability to examine all the bit planes of one pixel in a single CPU access cycle.

One further point must be made clear. The Input and Output Block instructions of the QPDM are provided to allow the CPU to access directly into display memory. As discussed earlier, the CPU may access the display memory "By Plane" or "By Pixel". This brings up the point of activity bits. Each QPDM has four activity bits, one for each plane it controls. With the Set Activity Bits instruction the user can set or clear these bits in each QPDM in the design. If the activity bit for a plane is set to "1", the plane will operate normally. Write operations will conditionally write into the plane. If the activity bit for a plane is set to a "0", the plane will not be written to. The activity bits of a QPDM also affect the generation of the $\overline{EDE}$ (external driver enable) signal. During accesses to the Block I/O FIFO of the QPDM, the following equation determines whether or not an $\overline{EDE}$ signal is generated:

$$\overline{EDE} = \overline{CS} \cdot A_1 \cdot A_0 \cdot (AB_0 + AB_1 + AB_2 + AB_3)$$

where AB_0 - AB_3 are the activity bits for each plane the QPDM controls. What are the ramifications of the activity bits? During an Input Block instruction, data from the CPU will only be written to the planes whose activity bits are set. During an Output Block instruction, where the QPDM outputs display map data to the CPU, two cases arise. In the case of a by-plane read of display data, only *one* activity bit in *one* QPDM may be set. To repeat, in a system with multiple QPDMS, when executing a by-plane Output Block instruction, make sure that only *one* QPDM has *one* activity bit set. If multiple QPDMs have their activity bits set, then their $\overline{EDE}$ pins will go active, enabling their data transceivers and bus contention will result. In the case of an output block by pixel, the software must be consistent to interpret the data bits from all the QPDMS. For a by pixel output, "1"s will be returned for all inactive planes. Therefore, the software must be consistent and keep track of all active planes so that it can interpret the data for the individual pixels.

DMA

The DMA issues remain the same as in the single QPDM design.

Interrupts

In this design an external interrupt controller is added. In the case of an external interrupt controller, several of the internal 80186 peripheral control registers must be reprogrammed. The internal interrupt controller must be in master mode. The cascade bit of the INT_0 control register must also be set. In this mode, whenever the interrupt presented to the INT_0 is acknowledged, the integrated interrupt controller will not provide the interrupt type for the interrupt. Instead, two interrupt-acknowledge bus cycles will be run, with the INT_2 pin of the 80186 now providing the interrupt-acknowledge pulses for the INT_0 requests. The 80186 will read the interrupt type from the lower eight bits of the address/data bus on the second interrupt-acknowledge cycle. In this design the INT_0 and INT_2 pins have been reconfigured to hook up to the external 8259A interrupt controller. The INT_1 and INT_3

lines are still used as direct interrupt inputs, identical to the single QPDM design. The 8259A can handle up to eight external interrupt requests. These eight, combined with the two direct input interrupt pins provide a total of 10 external interrupt requests. Also note that an interrupt-ready signal must be returned to the CPU to prevent wait state generation during the interrupt-acknowledge cycles. This is provided via the PAL devices. We have also used PCS_5 to provide the chip-select logic to the external interrupt controller. The four INT pins of the QPDMs have been connected to the external 8259A. Software must be written to initialize the external 8259A and to set up the actual external interrupt vector numbers. More details can be found in the 8259A data sheet.

Miscellaneous

The comments on peripheral chip selects in the single QPDM case also apply in this case. Remember to program the PACS and MPC_5 registers for the correct (one) number of wait states with no external ready. Consult the 80186 data sheet for bit patterns. The base address for the $\overline{PCS}_X$ pins are programmed into the PACS register and each $\overline{PCS}_X$ pin is assigned a 128 byte block relative to this base address. The $\overline{PCS}_X$ pins respond only to the addresses in their individually assigned blocks. This means that each QPDM in the multiple QPDM design is assigned a 128-byte block of addresses. The global chip select takes up another 128-byte block.

One last statement: in any QPDM design, a few extra moments ensuring that the hardware bit organizations are consistent with the software interpretation is time well spent.

The PAL Equations are listed on the next ten pages.

```
CUPL Version 2.10a Serial# 2-00001-066
Copyright© 1983,84,85,86 Personal CAD Systems, Inc.
CREATED Fri Dec 12 12:45:19 1986

LISTING FOR LOGIC DESCRIPTION FILE: XCVR.pld

 1:Name       XCVR ;
 2:Partno     95C60-2 ;
 3:Date       12/12/86;
 4:Revision   02;
 5:Designer   Ed Dupuis ;
 6:Company    Advanced Micro Devices Canada
 7:Device     P16R6;
 8:
 9:/*************************************************************************/
10:/* This PAL generates a signal T4 which signifies to the data            */
11:/* transceivers when the CPU enters T state 4.  During a QPDM            */
12:/* read cycle, this signal T4 turns off the transceivers so that         */
13:/* the CPU specification tHRAV is not violated.  During a QPDM           */
14:/* write cycle, this T4 signal is not generated, and /DEN is             */
15:/* allowed to turn off the data transceivers.                            */
16:
17:/*************************************************************************/
18:/* Allowable Target Device Types:  16R6, 16RB, anything with at          */
19:/* least five free registers.                                            */
20:/*************************************************************************/
21:
22:/**  Inputs  **/
23:
24:Pin 1 =     CLOCK          ;    /* Inverted CPU clock from chip select PAL */
25:Pin 2 =     ALE            ;    /* Address Latch Enable from CPU.          */
26:                                 /* this signal tells us when CPU reaches   */
27:                                 /* state T1                                */
28:Pin 3 =     !READ          ;    /* CPU read strobe. Only when this         */
29:                                 /* signal is active do we generate         */
30:                                 /* output signal T4.                       */
31:Pin 4 =     RESET          ;    /* CPU reset signal out                    */
32:
33:Pin11 =     !OE            ;    /* Output Enable for PAL. Grounded         */
34:                                 /* permanently                             */
35:
36:/**  Outputs  **/
37:
38:Pin18 =     T4             ;    /* This signal is generated when           */
39:                                 /* the CPU reaches state T4 during         */
40:                                 /* a QPDM read.                            */
41:Pin17 =     TW             ;    /* Wait State                              */
42:Pin16 =     T3             ;    /* CPU State T3                            */
43:Pin15 =     T2             ;    /* CPU State T2                            */
44:Pin14 =     T1             ;    /* Appearance of ALE signifies the         */
45:                                 /* onset of CPU state T1                   */
46:
47:
48:/**  Declarations and Intermediate Variable Definitions  **/
49:
50:
```

```
51:
52:
53:
54:
55:/**  Logic Equations  **/
56:
57:T1.D = ALE 7 !RESET ;
58:T2.D = T1 & !RESET  ;
59:T3.D = T2 & !RESET  ;
60:TW.D = T3 & READ & !RESET  ;
61:T4,D = TW & !RESET  ;
62:
63:
64:
65:
66:
[0022ca]  Please note: missing header item(s)

Jedec Fuse Checksum          (2AAD)
Jedec Transmit Checksum      (AF91)
```

```
CUPL Version 2.10a Serial# 2-00001-066
Copyright© 1983,84,85,86 Personal CAD Systems, Inc.
CREATED Fri Dec 12 16:59:56 1986

LISTING FOR LOGIC DESCRIPTION FILE: cup1\qpdmcs.pld

 1:Name       QPDMCS ;
 2:Partno     95C60-1 ;
 3:Date       12/11/86;
 4:Revision   01;
 5:Designer   Ed Dupuis ;
 6:Company    Advanced Micro Devices Canada
 7:Device     P18P8;
 8:
 9:
10:/*************************************************************************/
11:/* This PAL generates a qualified chip select to the Am95C60.            */
12:/* It also provides an inverted CLKOUT from the 80186 to the             */
13:/* processor state monitor PAL.                                          */
14:/* This PAL also generates the output enable signal to the               */
15:/* data transceivers.                                                    */
16:/*************************************************************************/
17:/* Allowable Target Device Types: Am16L8B, Am18P8B                       */
18:/*************************************************************************/
19:
20:/**  Inputs  **/
21:
22:Pin 1 =     CLKOUT        ;    /* CPU clock signal from 80186            */
23:Pin 2 =     RESET         ;    /* RESET signal from 80186                */
24:Pin 3 =     ALE           ;    /* Address Latch Enable from 80186        */
25:Pin 4 =     !PCS0         ;    /* Peripheral chip select from CPU        */
26:Pin 5 =     !DEN          ;    /* Data ENable from CPU                   */
27:Pin 6 =     T4            ;    /* CPU state T4 from state monitor PAL    */
28:
29:/**  Outputs  **/
30:
31:Pin19 =     !QPDMCS       ;    /* Qualified chip select for QPDM         */
32:Pin18 =     !QPDMOE       ;    /* Output enable for QPDM data transceivers*/
33:Pin17 =     !RESETOUT     ;    /* Inverted CPU reset to system           */
34:Pin16 =     !CLK          ;    /* Inverted CPU clock                     */
35:
36:/** Declarations and Intermediate Variable Definitions **/
37:
38:
39:
40:
41:
42:
43:/**  Logic Equations  **/
44:CLK = CLKOUT  ;                 /* Inverted clock to state monitor PAL  */
45:RESETOUT = RESET  ;             /* Inverted CPU reset to system         */
46:
47:QPDMCS = PSC0 & !ALE & !RESET;  /* Qualify address to QPDM by           */
48:                               /* delaying chip select until ALE       */
49:                               /* ensures valid addresses.             */
50:
```

```
51:QPDMOE = DEN & !T4 & QPDMCS ;     /* Turn off data transceivers when      */
52:                                  /* either DEN goes inactive or          */
53:                                  /* T4 goes active, whichever event      */
54:                                  /* occurs first.                        */
55:
56:
[0022ca]  Please note: missing header item(s)

Jedec Fuse Checksum          (2362)
Jedec Transmit Checksum      (B64F)
```

```
CUPL Version 2.10a Serial# 2-00001-066
Copyright© 1983,84,85,86 Personal CAD Systems, Inc.
CREATED Wed Dec 17 11:06:28 1986

LISTING FOR LOGIC DESCRIPTION FILE: XCVR1.pld

 1:Name       XCVR1 ;
 2:Partno     95C60-3 ;
 3:Date       12/16/86;
 4:Revision   02;
 5:Designer   Ed Dupuis ;
 6:Company    Advanced Micro Devices Canada
 7:Device     P16R6;
 8:
 9:/*************************************************************************/
10:/* This PAL generates a signal T4 which signifies to the data            */
11:/* transceivers when the CPU enters T state 4.  During a QPDM            */
12:/* read cycle, this signal T4 turns off the transceivers so that         */
13:/* the CPU specification tHRAV is not violated.  During a QPDM           */
14:/* write cycle, this T4 signal is not generated, and /DEN is             */
15:/* allowed to turn off the data transceivers.                            */
16:/* This PAL also generated QPDM and system reset, and also               */
17:/* provides the correct polarity ready signal to the CPU from the        */
18:/* interrupt acknowledge pulse output from the CPU.                      */
19:
20:*************************************************************************/
21:/* Allowable Target Device Types: 16R6, 16R8, anything with at           */
22:/* least five free registers.                                            */
23:/*************************************************************************/
24:
25:/**  Inputs  **/
26:
27:Pin 1 =     CLOCK          ;    /* Inverted CPU clock from chip select PAL */
28:Pin 2 =     ALE            ;    /* Address Latch Enable from CPU.          */
29:                                /* This signal tells us when CPU reaches   */
30:                                /* state T1.                               */
31:Pin 3 =     !READ          ;    /* CPU read strobe.  Only when this        */
32:                                /* signal is active do we generate         */
33:                                /* output signal T4.                       */
34:Pin 4 =     RESET          ;    /* CPU reset signal out                    */
35:
36:Pin11 =     !OE            ;    /* Output Enable for PAL.  Grounded        */
37:                                /* permanently.                            */
38:
39:Pin 5 =     !INTAO         ;    /* Interrupt acknowledge pulses            */
40:                                /* from CPU to external 8259A-2            */
41:                                /* these pulses are necessary to           */
42:                                /* the CPU to finish the interrupt         */
43:                                /* vector fetching process.                */
44:/**  Outputs  **/
45:
46:Pin19 =     !RESETOUT      ;    /* System and QPDM reset from CPU          */
47:Pin18 =     T4             ;    /* This signal is generated when           */
48:                                /* the CPU reaches state T4 during         */
49:                                /* a QPDM read.                            */
50:Pin17 =     TW             ;    /* Wait State                              */
```

```
51:Pin16 =     T3            ;    /* CPU State T3                          */
52:Pin15 =     T2            ;    /* CPU State T2                          */
53:Pin14 =     T1            ;    /* Appearance of ALE signifies the       */
54:                               /* onset of CPU state T1.                */
55:Pin12 =     INTREADY      ;    /* Inverted interrupt acknowledge        */
56:                               /* pulse to the ARDY input of the        */
57:                               /* CPU.                                  */
58:
59:/**  Declarations and Intermediate Variable Definitions  **/
60:
61:
62:
63:
64:
65:
66:/**  Logic Equations  **/
67:
68:T1.D = ALE & !RESET  ;
69:T2.D = T1 & !RESET   ;
70:T3.D = T2 & !RESET   ;
71:TW.D = T3 & READ & !RESET  ;
72:T4.D = TW & !RESET   ;
73:
74:INTREADY = INTAO     ;
75:RESETOUT = RESET     ;
76:
77:
78:
[0022ca]  Please note: missing header item(s)


Jedec Fuse Checksum          (3A8C)
Jedec Transmit Checksum      (CDF6)
```

```
CUPL Version 2.10a Serial# 2-00001-066
Copyright© 1983,84,85,86 Personal CAD Systems, Inc.
CREATED Wed Dec 17 11:02:24  1986

LISTING FOR LOGIC DESCRIPTION FILE: QPDM1.pld

 1:Name       QPDM1 ;
 2:Partno     18P8-1 ;
 3:Date       12/16/86;
 4:Revision   03;
 5:Designer   Ed Dupuis ;
 6:Company    Advanced Micro Devices Canada
 7:Device     P18P8;
 8:
 9:
10:/*************************************************************************/
11:/* This PAL generates qualified chip selects to the 4 Am95C60s.          */
12:/* It also provides an inverted CLKOUT from the 80186 to the             */
13:/* processor state monitor PAL.                                          */
14:/* This PAL also generates the signal TSYNOUT.  This signal is used      */
15:/* by the QPDMs to synchronize display memory activities.                */
16:/*************************************************************************/
17:/* Allowable Target Device Types:  Am16L8B, Am8P8B                       */
18:/*************************************************************************/
19:
20:/**  Inputs  **/
21:
22:Pin 1 =      CLKOUT        ;    /* CPU clock signal from 80186            */
23:Pin 2 =      RESET         ;    /* RESET signal from 80186                */
24:Pin 3 =      ALE           ;    /* Address Latch Enable from 80186        */
25:Pin 4 =      !PSC0         ;    /* QPDM #1 chip select from CPU           */
26:Pin 5 =      !PCS1         ;    /* QPDM #2 chip select from CPU           */
27:Pin 6 =      !PCS2         ;    /* QPDM #3 chip select from CPU           */
28:Pin 7 =      !PCS3         ;    /* QPDM #4 chip select from CPU           */
29:Pin 8 =      !PCS4         ;    /* Global chip select which selects       */
30:                                /* all QPDMs for broadcasting             */
31:Pin 9 =      TSYN1OUT      ;    /* These are the four signals output      */
32:Pin11 =      TSYN2OUT      ;    /* by the individual QPDMs and are used   */
33:Pin12 =      TSYN3OUT      ;    /* to synchronize display memory          */
34:Pin13 =      TSYN4OUT      ;    /* activities.                            */
35:
36:/**  Outputs  **/
37:
38:Pin19 =      !QPDMCS1      ;    /* Qualified chip select for QPDM1        */
39:Pin18 =      !QPDMCS2      ;    /* Qualified chip select for QPDM2        */
40:Pin17 =      !QPDMCS3      ;    /* Qualified chip select for QPDM3        */
41:Pin16 =      !QPDMCS4      ;    /* Qualified chip select for QPDM4        */
42:Pin15 =      !CLK          ;    /* Inverted CPU clock to state            */
43:                                /* monitor PAL.                           */
44:
45:Pin14 =      TSYNOUT       ;    /* Global signal to TSYNIN inputs         */
46:                                /* of all the QPDMs.                      */
47:
48:/**  Delcarations and Intermediate Variable Definitions  **/
49:
50:
```

```
51:
52:
53:
54:
55:/**  Logic Equations  **/
56:CLK = CLKOUT   ;                  /* Inverted clock to state monitor PAL  */
57:RESETOUT = RESET ;                /* Inverted CPU reset to system         */
58:
59:QPDMCS1 = PCS0 & !ALE & !RESET    /* Qualify addresses to QPDM1 by       */
60:         # PCS4 & !ALE & !RESET;  /* delaying chip select until ALE       */
61:                                  /* ensures valid addresses              */
62:
63:QPDMCS2 = PCS1 & !ALE & !RESET    /* Qualify addresses to QPDM2 by        */
64:         # PCS4 & !ALE & !RESET;  /* delaying chip select until ALE       */
65:                                  /* ensures valid addresses              */
66:
67:QPDMCS3 = PCS2 & !ALE & !RESET    /* Qualify addresses to QPDM3 by        */
68:         # PCS4 & !ALE & !RESET;  /* delaying chip select until ALE       */
69:                                  /* ensures valid addresses              */
70:
71:QPDMCS4 = PCS3 & !ALE & !RESET    /* Qualify addresses to QPDM4 by        */
72:         # PCS4 & !ALE & !RESET;  /* delaying chip select until ALE       */
73:                                  /* ensures valid addresses              */
74:
75:TSYNOUT = TSYN1OUT & TSY2OUT & TSYN3OUT & TSYN4OUT ;    /* All QPDMs     */
76:                                  /* must be in synch for this signal     */
77:                                  /* to be active                         */
78:
79:
[0022ca]  Please note: missing header item(s)

Jedec Fuse Checksum          (456C)
Jedec Transmit Checksum      (04E7)
```

```
CUPL Version 2.10a Serial# 2-00001-066
Copyright© 1983,84,85,86 Personal CAD Systems, Inc.
CREATED Wed Dec 17 11:04:33 1986

LISTING FOR LOGIC DESCRIPTION FILE: QPDM2.pld

 1:Name       QPDM2 ;
 2:Partno     18P8-2 ;
 3:Date       12/16/86;
 4:Revision   03;
 5:Designer   Ed Dupuis ;
 6:Company    Advanced Micro Devices Canada
 7:Device     P18P8;
 8:
 9:
10:/*************************************************************************/
11:/* This PAL generates qualified output enables to the data xcvrs         */
12:/* associated with each individual QPDM.  When the selected QPDM's       */
13:/* chip select goes active this will cause the QPDM's /EDE output        */
14:/* pin to go active.  This signal is then factored in to generate        */
15:/* an output enable signal.  As in the single QPDM design, the           */
16:/* signal T4 from the state monitor PAL will disable the xcvrs on         */
17:/* a read of the selected QPDM by the CPU.  In the event of a CPU         */
18:/* write cycle, T4 is not generated by the state PAL and /DEN             */
19:/* going inactive disables the xcvrs.                                     */
20:/* This PAL also generates the output enable for the "by pixel"           */
21:/* data transceiver.                                                      */
22:/* This PAL also generates the MATOUT signal to all the QPDMs.            */
23:
24:
25:/*************************************************************************/
26:/* Allowable Target Device Types: Am16L8B, Am18P8B                        */
27:/*************************************************************************/
28:
29:/**  Inputs  **/
30:
31:Pin 1 =     T4            ;    /* CPU state T4 signal from state          */
32:                               /* PAL.                                    */
33:Pin 2 =     RESET         ;    /* RESET signal from 80186                 */
34:Pin 3 =     !DEN          ;    /* Data ENable from CPU                    */
35:Pin 4 =     !EDE0         ;    /* QPDM #1 external driver enable          */
36:Pin 5 =     !EDE1         ;    /* QPDM #2 external driver enable          */
37:Pin 6 =     !EDE2         ;    /* QPDM #3 external driver enable          */
38:Pin 7 =     !EDE3         ;    /* QPDM #4 external driver enable          */
39:Pin 8 =     A3            ;    /* Address bit A3 which differentiates     */
40:                               /* between the regular data xcvrs and      */
41:                               /* the "by pixel" data xcvr.               */
42:Pin 9 =     MAT1OUT       ;    /* Color match/syncronization signals      */
43:Pin11 =     MAT2OUT       ;    /* from the four QPDMs.  These signals     */
44:Pin12 =     MAT3OUT       ;    /* must be ANDed together to ensure        */
45:Pin13 =     MAT4OUT       ;    /* that all QPDMs execute together         */
46:
47:/**  Outputs  **/
48:
49:Pin19 =     !OE1          ;    /* Qualified output enable for             */
50:                               /* QPDM1's xcvrs                           */
```

```
51:Pin18 =      !OE2          ;    /* Qualified output enable for          */
52:                                /* QPDM2's xcvrs                        */
53:Pin17 =      !OE3          ;    /* Qualified output enable for          */
54:                                /* QPDM3's xcvrs                        */
55:Pin16 =      !OE4          ;    /* Qualified output enable for          */
56:                                /* QPDM4's xcvrs                        */
57:Pin15 =      !PIXELOE      ;    /* Qualified output enable for          */
58:                                /* "by pixel" xcvr                      */
59:
60:Pin14 =      MATOUT        ;    /* Synchronization signal for QPDMs     */
61:
62:/**  Declarations and Intermediate Variable Definitions  **/
63:
64:
65:
66:
67:/**  Logic Equations  **/
68:
69:OE1 = EDE0 & !A3 & DEN & !T4 & !RESET  ;     /* enable qpdm1's xcvr     */
70:
71:0E2 = EDE1 & !A3 & DEN & !TA & !RESET  ;     /* enable qpdm2's xcvr     */
72:
73:OE3 = EDE2 & !A3 & DEN & !TA & !RESET  ;     /* enable qpdm3's xcvr     */
74:
75:OE4 = EDE3 & !A3 & DEN & !T4 & !RESET  ;     /* enable qpdm4's xcvr     */
76:
77:PIXELOE = EDE0 & A3 & DEN & !T4 & !RESET     /* enable the "by pixel"   */
78:         # EDE1 & A3 & DEN & !T4 & !RESET     /* data xcvr.              */
79:         # EDE2 & A3 & DEN & !T4 & !RESET
80:         # EDE3 7 A3 & DEN & !T4 & !RESET  ;
81:
82:
83:
84:
85:MATOUT = MAT1OUT & MAT2OUT & MAT3OUT & MAT4OUT; /* Determine if all QPDMs*/
86:                                              /* are matched and in synch */
87:
88:
[0022ca]  Please note: missing header item(s)

Jedec Fuse Checksum          (3DFE)
Jedec Transmit Checksum      (04CE)
```

2.2 VME BUS

The QPDM, as a fast graphic display controller, can be adapted to nearly any system bus structure.

This application note describes the adaption to the VME Bus. The term VME Bus, in this case, does not include the other busses such as VMX and VMS Bus. Only a small amount of hardware must be developed to adapt the QPDM to the VME Bus in a simple way.

In the following chapters, only the system bus of the QPDM is described. The other busses are not of interest when discussing the system bus interface.

The QPDM has a normal operation and a DMA-driven operation. Two approaches have thus been made for an adaption logic. The first approach uses only the CPU to do all tasks for the QPDM. The second approach uses a DMA controller to do high-speed transfers between the QPDM and the Main Memory.

2.2.1 Simple Approach

Circuit diagram 2.2-1 shows the simple approach. The circuit only needs an address decoder, a QPDM, Data drivers, and a PAL device that does the interfacing between the VME Bus and the QPDM.

Address Decoder

The address decoder contains an 8-input NAND gate and an Am25LS521 Comparator. So, the QPDM address space begins at an address with the eight most-significant address bits = 1. The next eight bits are selected by the comparator. The lowest bits are not fully decoded.

Interface PAL

The VME Bus is an asynchronous bus. The QPDM is a synchronous device, so the PAL device has to generate an asynchronous signal $\overline{\text{DTACK}}$ for the VME bus. All

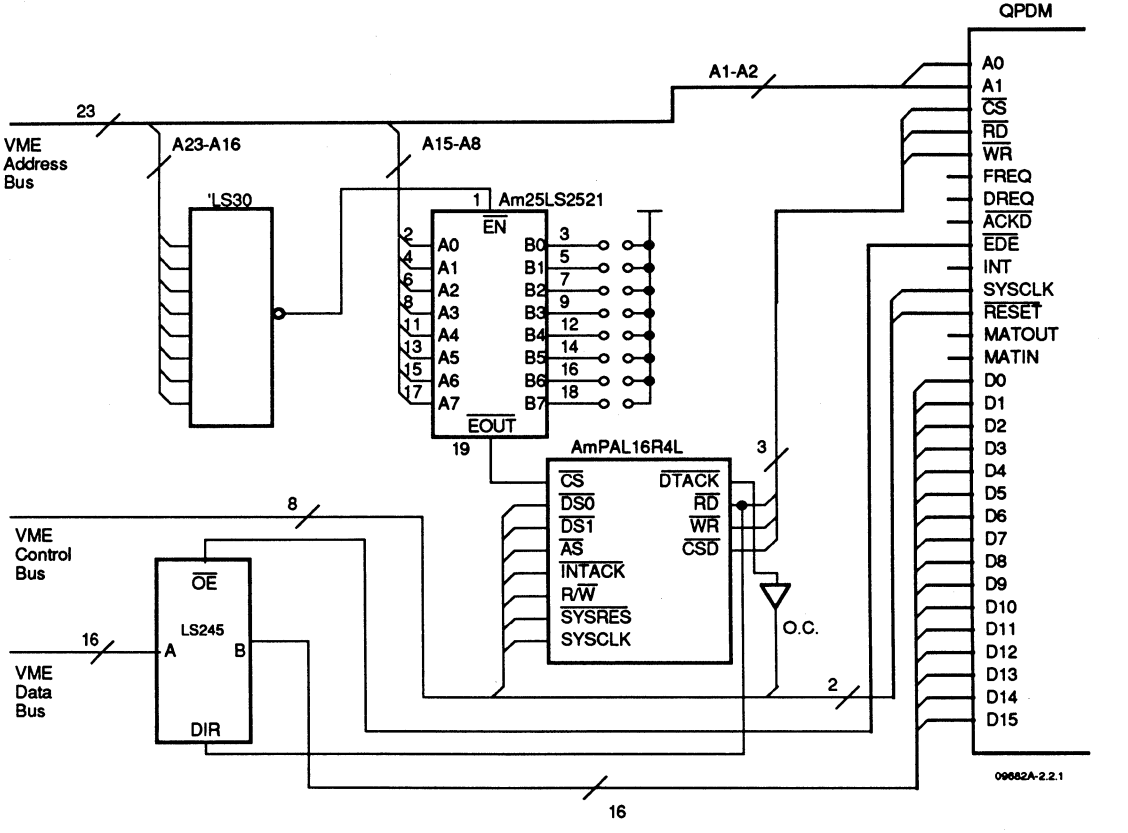


Figure 2.2-1 Circuit Diagram: Design without DMA

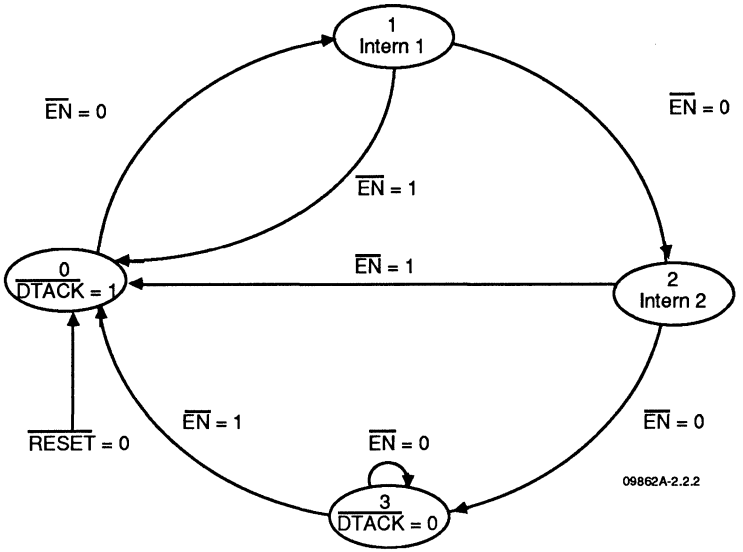


Figure 2.2-2 States of Machine

terms concerning the VME Bus are not described in detail, therefore, the VME Bus specification has to be studied. Another task of the PAL device is to generate the necessary control signals for the QPDM.

The following describes the generation of all signals in detail.

Generation of DTACK

The DTACK signal is generated to show the Bus Master that the QPDM is ready with the Data transfer. The QPDM itself is not generating this signal. The signal is generated in the following way.

When a normal Read or Write to the QPDM is done, the PAL device generates three cycles after the DTACK signal. The clock is the normal system clock. The DTACK signal is generated until the Read or Write cycle is finished.

The criterion for a normal Read or Write cycle at the VME Bus is:

1. CS for the device is LOW.
2. Either DS_0 or DS_1 is LOW.
3. INTACK is HIGH. (The cycle is not an interrupt acknowledge cycle.)

The logical form is:

$$\overline{EN} = \overline{CS} + (\overline{DS}_0 \cdot \overline{DS}_1) + INTACK$$

To generate DTACK, a little state machine has to be developed. The states of the machine are shown in Figure 2.2-2.

Because the machine has four states, two flip-flops are needed to realize the state machine. Figure 2.2-3 shows the state change of the machine. With this form, the combinational logic could be easily computed. The simplification of the logic is shown in the two KV diagrams (Figure 2.2-4)

Figure 2.2-3 Change Table

EN	D1	D0	I1	I0
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

As another point the state machine has to be set to state 0 after a system reset. The Input formula for flip-flop 1 is:

$$I_0 = EN \cdot \overline{D_0} \cdot \overline{RESET} + EN \cdot D_1 \cdot \overline{RESET}$$

with the equation of $\overline{EN}$:

$$\begin{aligned} I_0 &= \overline{(\overline{CS} + \overline{DS_0} \cdot \overline{DS_1} + INTACK)} \cdot \overline{D_0} \cdot \overline{RESET} \\ &\quad + \overline{(\overline{CS} + \overline{DS_0} \cdot \overline{DS_1} + INTACK)} \cdot D_1 \cdot \overline{RESET} \\ &= DS_0 \cdot CS \cdot \overline{INTACK} \cdot \overline{D_0} \cdot \overline{RESET} \\ &\quad + DS_1 \cdot CS \cdot \overline{INTACK} \cdot \overline{D_0} \cdot \overline{RESET} \\ &\quad + DS_0 \cdot CS \cdot \overline{INTACK} \cdot D_1 \cdot \overline{RESET} \\ &\quad + DS_1 \cdot CS \cdot \overline{INTACK} \cdot D_1 \cdot \overline{RESET} \end{aligned}$$

The input formula for flip-flop 2 is:

$$I_1 = EN \cdot D_0 \cdot \overline{RESET} + EN \cdot D_1 \cdot \overline{RESET}$$

with the equation of $\overline{EN}$:

$$\begin{aligned} I_1 &= \overline{(\overline{CS} + \overline{DS_0} \cdot \overline{DS_1} + INTACK)} \cdot D_0 \cdot \overline{RESET} \\ &\quad + \overline{(\overline{CS} + \overline{DS_0} \cdot \overline{DS_1} + INTACK)} \cdot D_1 \cdot \overline{RESET} \\ &= DS_0 \cdot CS \cdot \overline{INTACK} \cdot D_0 \cdot \overline{RESET} \\ &\quad + DS_1 \cdot CS \cdot \overline{INTACK} \cdot D_0 \cdot \overline{RESET} \\ &\quad + DS_0 \cdot CS \cdot \overline{INTACK} \cdot D_1 \cdot \overline{RESET} \\ &\quad + DS_1 \cdot CS \cdot \overline{INTACK} \cdot D_1 \cdot \overline{RESET} \end{aligned}$$

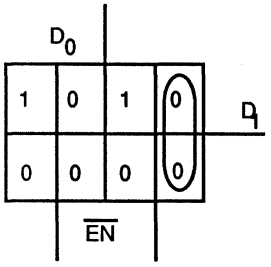
$\overline{DTACK}$ signal is generated in the following way:

$$\overline{DTACK} = \overline{D_0} \cdot D_1$$

Other Control Signals

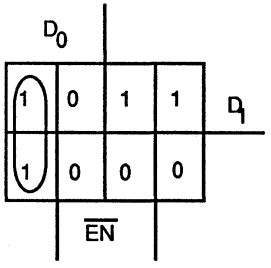
The other control signals for the QPDM are:

- **$\overline{RD}$:** $\overline{RD}$ is generated when either $\overline{DS_0}$ or $\overline{DS_1}$ is LOW and R/$\overline{W}$ is HIGH.
- **$\overline{WR}$:** $\overline{WR}$ is generated when either $\overline{DS_0}$ or $\overline{DS_1}$ is LOW and R/$\overline{W}$ is LOW.
- **$\overline{CSD}$:** QPDM Chip Select, $\overline{CSD}$, is generated when $\overline{AS}$ is LOW and $\overline{CS}$ is LOW and no interrupt acknowledge cycle is performed ($\overline{INTACK}$ = High).



$$I_0 = \overline{EN} \cdot \overline{D_0} + \overline{EN} \cdot D_0 \cdot D_1$$

KV-Diagram for I_0



$$I_1 = \overline{EN} \cdot D_0 + \overline{EN} \cdot \overline{D_0} \cdot D_1$$

KV-Diagram for I_1

Figure 2.2-4 KV-Diagrams

The formulas are:

$$\overline{RD} = \overline{DS_0} \cdot \overline{DS_1} + \overline{R/W}$$

$$\overline{WR} = \overline{DS_0} \cdot \overline{DS_1} + R/\overline{W}$$

$$\overline{CSD} = \overline{AS} + \overline{CS} + INTACK$$

The timing of the PAL device is shown in Figure 2.2-5.

2.2.2 DMA Approach

To use all the features of the QPDM, the QPDM must couple with a DMA controller; thus an Am9516 two-channel DMA controller was used. The interface for that device to the VME-Bus is described in the Am9516 DMA Contoller Technical Manual starting on page 6-20. The QPDM is adapted to the VME-Bus with the same PAL device discussed in Section 2.2.1 of this handbook. This PAL device also performs bus-driver steering. Address decoding is discussed in the same section. Only the lower addresses are decoded with a multiplexer.

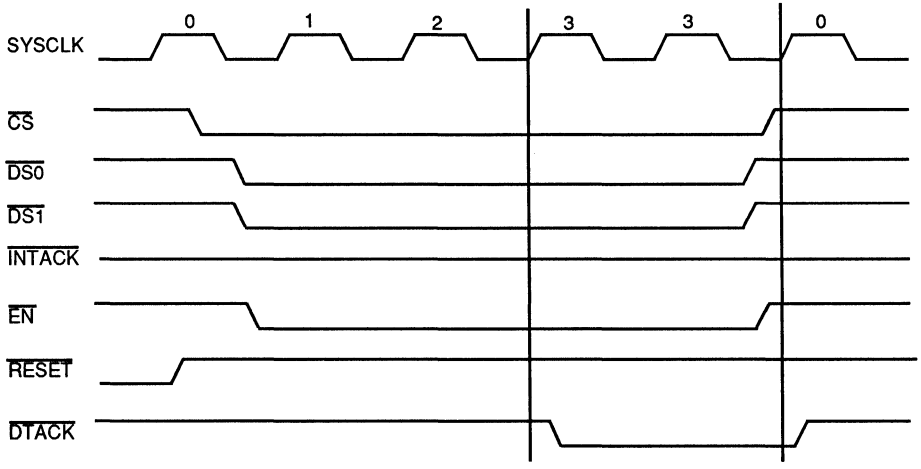


Figure 2.2-5a Timing of DTACK

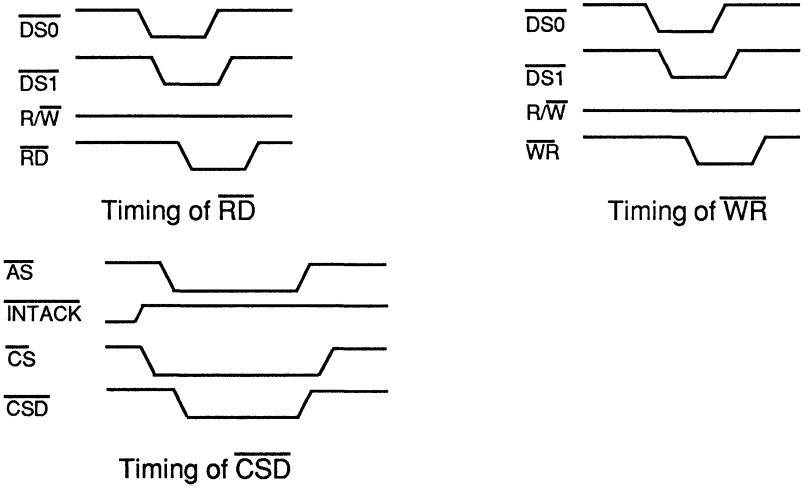


Figure 2.2-5b Other Timings

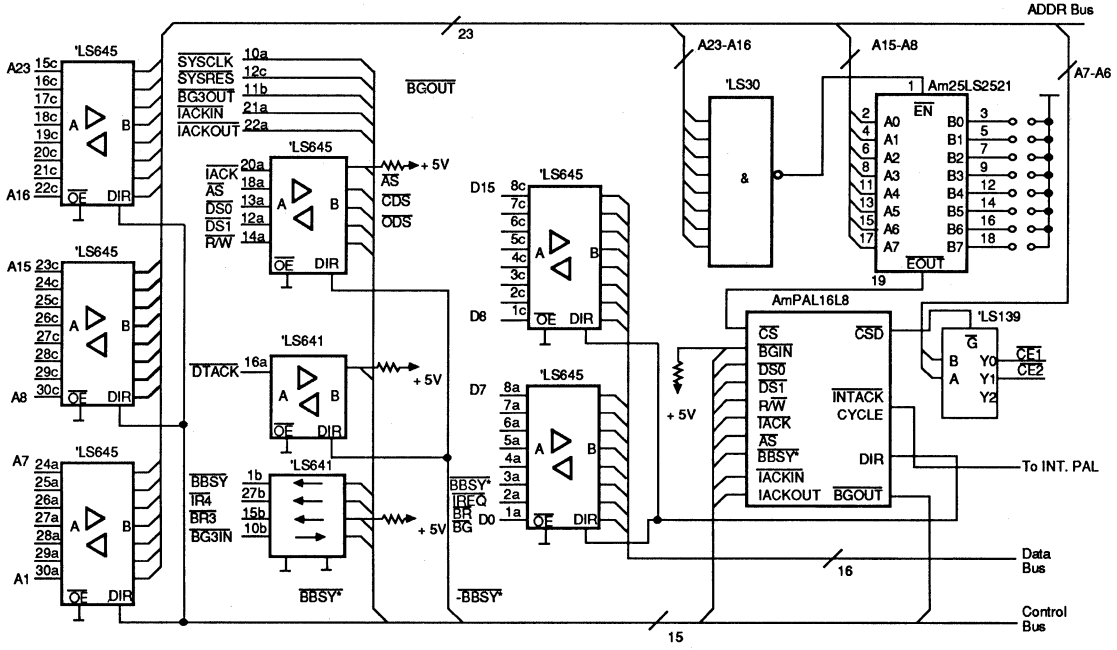


Figure 2.2-6 Circuit Diagram: VME - Bus - Interface

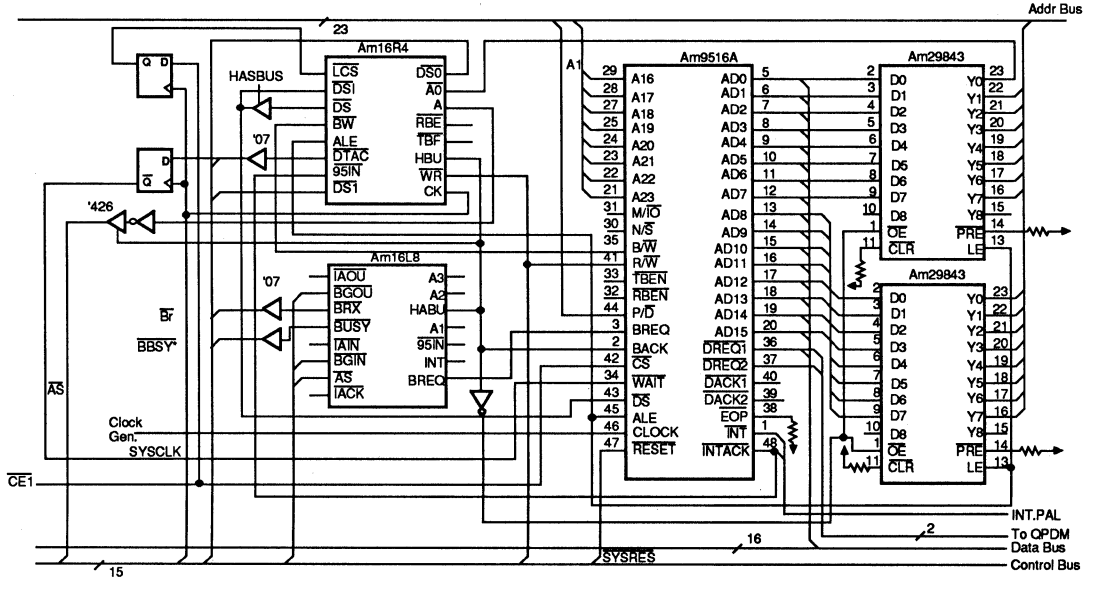


Figure 2.2-7 Circuit Diagram: DMA Section

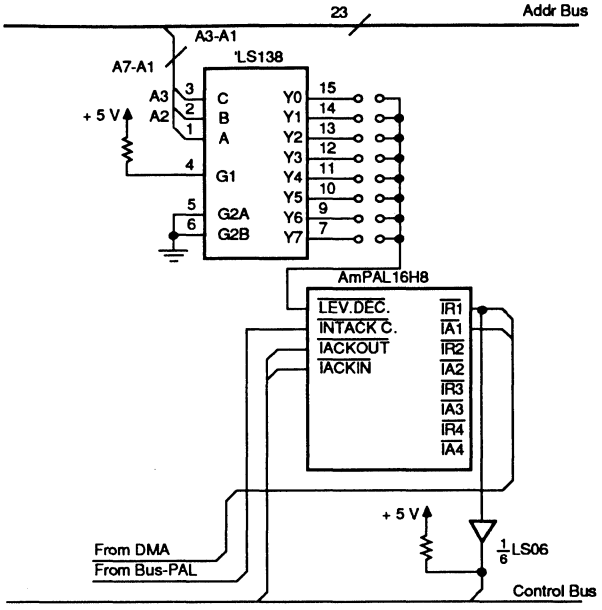


Figure 2.2-8 Circuit Diagram: Interrupt Logic

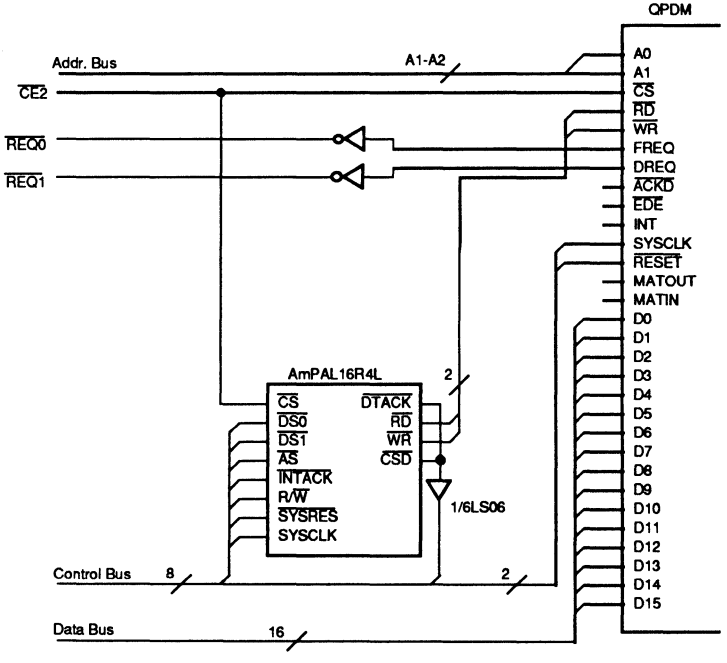


Figure 2.2-9 Circuit Diagram: QPDM Interface

2.3 68020 BUS

A graphics engine utilizing a new graphics controller can provide a real boost to minicomputers based on the 68020. The Am95C60 Quad Pixel Dataflow Manager (QPDM) combines four major functions previously requiring many integrated circuits: video refresh of the display; memory refresh; update of the bit map; and arbitration between the memory update, video refresh, and dynamic memory refresh.

Not only can the Am95C60 support a bit map size of up to 4K x 4K pixels and screen size of up to 2K x 2K, but it also has the high- performance drawing capabilities that are necessary in today's advanced graphic systems.

e.g. Block Move - 60 ns/pixel,
Vector Draw - 300 ns/pixel
Polygon Fill - 280 ns/pixel

Advanced features such as Anti-Aliasing, Hardware and Software Windowing, Clipping, Picking, Text and Polygon Fill are supported directly on chip.

A minimum system consists of one Am95C60 plus Video DRAM frame buffer, one serializer per plane (Am8172 or Am8177) and clock generator (Am8158), together with a color palette (Am8159 OR Am8151). Such a system is easily expandable to support up to 256 color planes.

A small amount of "glue" logic allows the Am95C60 to interface with all of the common 8-, 16- and 32-bit microprocessors.

2.3.1 Overview of Graphics Engine and Display System

The Tasks Required of a Graphics Engine

Key features in today's workstation are the ability to offer a high-resolution graphic display and a rapid response to prompts by the user to manipulate images on the screen.

As many of the applications of graphic workstations involve the manipulation of "visual information" stored within the system, there is an obvious need not only for a significant amount of memory to hold this visual information, but also for dedicated hardware to manipulate such data and supply it at the required data rate and in the correct format to the workstation display system (typically a high-resolution color CRT).

The Advantages of Single Chip, Dedicated Graphics Controller

The Am95C60 is a graphics processing chip capable of handling all the necessary tasks for supporting a bit-mapped display memory graphics scheme, where the display memory is constructed from Video DRAMs. As such, much of the work load in supporting the graphics sub-system is taken from the host CPU and handled directly by this dedicated graphics processor. These features include dynamic memory refresh control, video display refresh control, line drawing and other graphic function support, and arbitration to allow other parts of the system access to the video display memory.

The Frame Buffer

The frame buffer consists of a number of memory devices which hold the current picture information to be supplied to the video display device (CRT).

Video DRAMs are used as the memory device for such a frame buffer. This special type of DRAM is similar to standard DRAMs, but has additional features which include a second port, ideal for supporting the interface to a video system.

In a special access cycle, called a Transfer Cycle, 1024 pixels of data can be read from the DRAM array into an on-chip shift mechanism. This shift mechanism can then be driven independently of, and concurrently with, further accesses to the DRAM array from the normal (host) port, providing serial pixel data at rates of up to 100 MHz. For example, by banking four Video DRAMs in parallel to provide a 16-bit data path, pixel rates of up to 400 MHz are achievable, with the real limiting factor being the maximum clock rate of the shift registers being used (Am8177 and Am8172 - 200 MHz max).

The advantage of this scheme over a system with video memory constructed from standard DRAMs is that typically the host or Am95C60 can access the video memory of Video DRAMs for update in excess of 95% of the time, compared with less than 40% of the time for a video memory of standard DRAMs. This will offer a greater drawing and BLITing capability, an essential feature in supporting animation and quick drawing and data transfer responses, key features in today's workstations.

The Am95C60 generates all the necessary signals to control such a Video DRAM array.

2.3.2 How to Address Peripheral Chips in a 68020 System

Virtual Memory - How to Address Hardware Resources

A virtual memory scheme allows a process in execution to have access to the total address space of the CPU, which for the MC68020 is 4 Gbytes (32 address lines). The address generated by the host processor will index

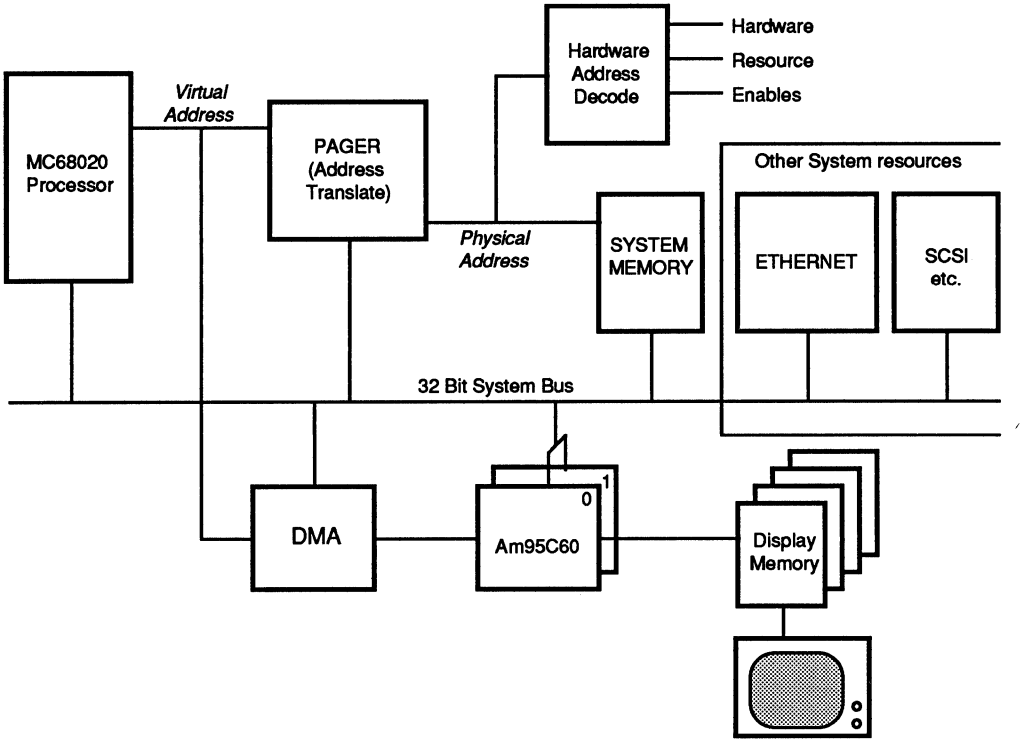


Figure 2.3-1 Typical Workstation Block Diagram

into a page table mechanism, or pager, which maps this 4 Gbytes of virtual address space to the physical memory within the system.

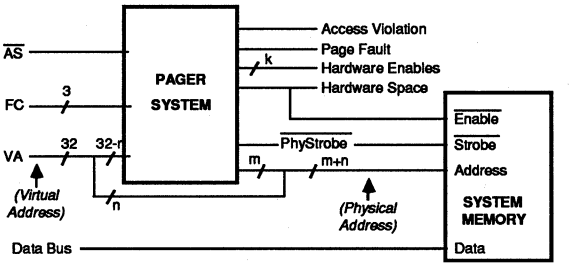
The pager system needs not only to generate the required address lines to the physical memory, translated from the virtual address, but must be able to select the hardware resources within the system. A simple method of implementing this could be to configure the pager system to generate one more physical address line than is required to address the system memory. The additional line is used to indicate that the access is to a hardware resource, not system memory. The page table entries can hence be set up to map a virtual address to any hardware resource within the system, offering maximum flexibility. To address a number of hardware resources within this "hardware address space", the additional address line can be used to enable decode logic of the lower address lines, thus disabling any access to the system memory to select the desired hardware resource.

Using this type of scheme, the enable to any peripheral device, such as the graphics engine, can be generated from the pager hardware system, which would include the additional address decode logic to generate individual chip select lines to each hardware resource within the system. For hardware resources containing a number of registers, some of the low order address lines can be used to address such registers, together with the chip select to that device.

The pager would also need to indicate to memory if and when the presented physical address is valid (PhyStrobe). This would be true only if a mapping of virtual address to physical address was found.

Other lines may be available on the system bus to qualify the address space (the function code lines FC0-2 for the MC68020), or to define whether the access is for code or data within user or supervisor address space. This information may also be used by the pager system to implement a memory protection scheme, but this is beyond the scope of this application note. (See Figure 2-3.2)

Typically the address decode can most simply be implemented using a combinatorial PAL device. (See Figures 2-3.3 and 2-3.4)



Where : n defines the No. of bytes per page;
m defines the No. of pages of system memory;
k defines the No. of Hardware Resources within system.

Figure 2.3-2 Simplified Pager System Block Diagram

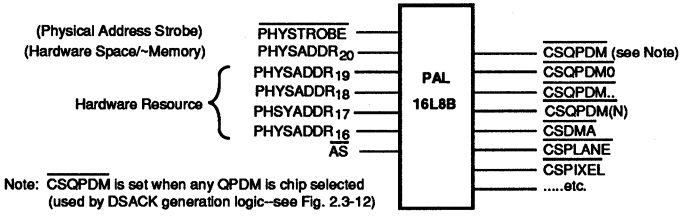


Figure 2.3-3 Hardware Resource Address Decoding

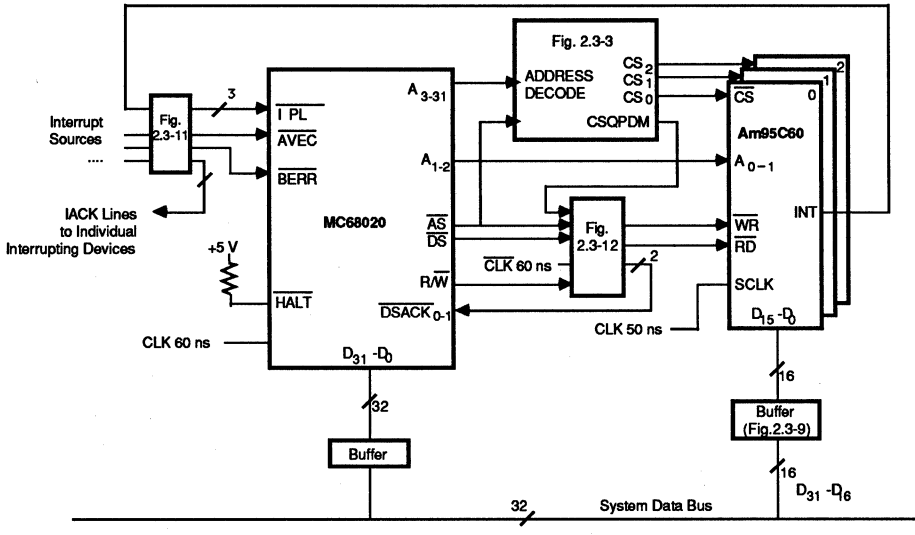


Figure 2.3-4 MC68020-Am95C60 System Block Diagram

The addresses used for individual hardware resources within any particular system can be totally arbitrary and will be defined by the designer as desired.

$\overline{AS}$ is used in the equation to turn off the CS device quickly at the end of a bus cycle.

$\overline{PhyStrobe}$ is only relevant when a "Pager" or similar system is implemented.

$\overline{CSPIXEL}$ is used to set the I/O mode register defining subsequent accesses between the system bus and display memory planes to be "By Pixel".

$\overline{CSPLANE}$ is used to reset the I/O mode register defining subsequent cases between the system bus and display memory planes to be "By Plane".

Direct Addressing of Peripheral Devices

In many applications where the MC68020 is the kernel CPU in a mini- or microcomputer running a high-level operation system, a virtual memory management scheme will be most applicable. However, in applications where the MC68020 is a controlling processor for peripheral devices, it is reasonable to use a direct addressing scheme where unique addresses are permanently allocated to peripheral devices within the system.

In such a system, the address decode logic needed to generate the relevant chip selects to the peripheral devices need only be a simple PAL device to decode the address lines together with Address Strobe ($\overline{AS}$). (See Figure 2.3-3)

Appendix 1 contains an example of the source code for such a PAL device.

2.3.3 Signal Definitions

When a common bus is connected to multiple components that do not have an identical bus cycle structure, such as the Am95C60 and the MC68020, a number of control and response signals require translation into the appropriate form. (See Figure 2.3-5)

Initiating the Bus Cycle

The MC68020 or similar bus master initiates a bus cycle by first requesting bus mastership via the arbitration signals "Bus Request", "Bus Grant" and "Bus Grant Acknowledge" ($\overline{BREQ}$, $\overline{BGRNT}$ and $\overline{BGACK}$). Once bus control is gained, the bus master drives an address and function code onto the address and function code bus lines and drives the Read/Write line defining the direction of transfer on the bus. The bus master then generates an address strobe, $\overline{AS}$, to define when the address lines are valid.

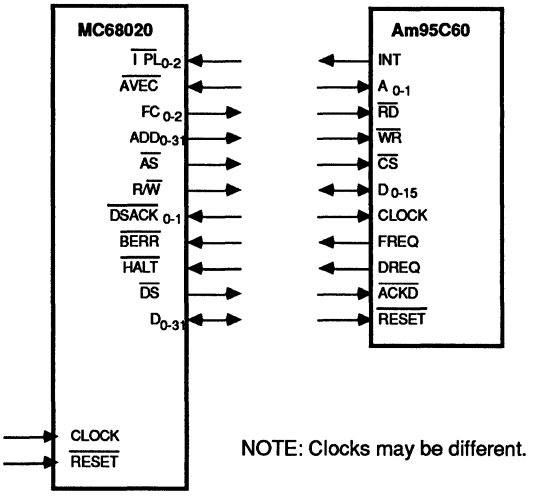


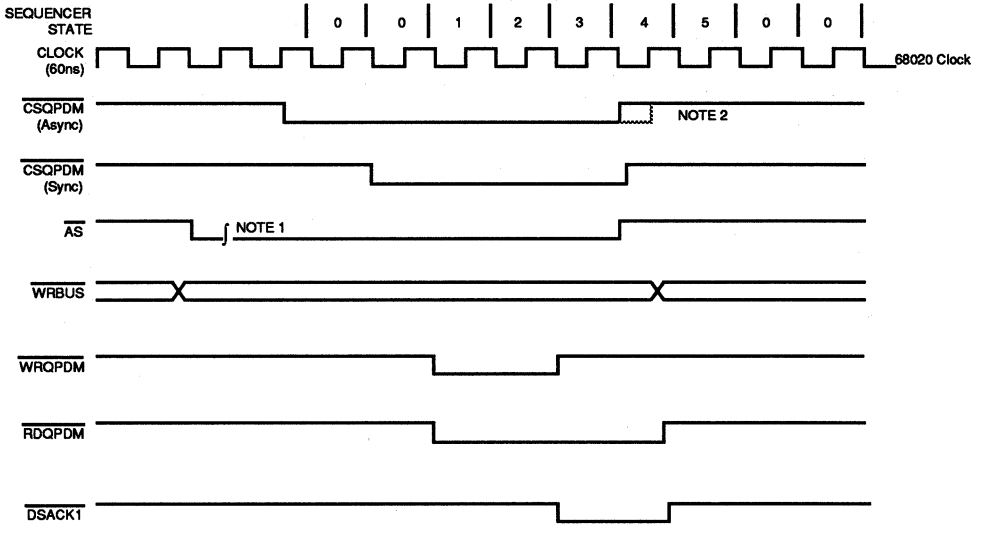
Figure 2.3-5 QPDM to 68020 Interface Signals

Data Strobe, $\overline{DS}$, is then generated at the appropriate time within the bus cycle on a Write indicating to the slave device that data is valid, or on a Read defining when the slave should send data to the bus master.

The 32-bit address is further qualified by the function code lines, FC_{0-2}, to define the address space within which this address resides. Logic (probably within the pager system) decodes all these lines to generate either an enable to the memory (if a mapping of virtual address to physical memory address exists within the pager system), an enable to the appropriate hardware resource (if a mapping exists to that hardware physical address), or neither if an access violation or page fault has occurred (no mapping exists).

If the bus cycle is aimed at the Am95C60, the $\overline{CSQPDM(N)}$ should be asserted. (See Figures 2.3-2 and 2.3-3)

The MC68020 qualifies the direction of transfer of the bus cycle using the $\overline{WR}$ signal (here called $\overline{WRBUS}$ to distinguish it from the write strobe signal to the QPDM), which has been set prior to $\overline{AS}$, and hence prior to $\overline{CSQPDM}$. This $\overline{WRBUS}$ signal can then be used to generate the correctly timed $\overline{RDQPDM}$ or $\overline{WRQPDM}$ signals to qualify the CSQPDM, as required by the Am95C60 to define the direction of transfer of information between the MC68020 and the Am95C60 (Figure 2.3-6).



NOTE 1: This delay is dependent on the Address Decode Logic.

NOTE 2: This delay in clearing the signal depends on the Address Decode PAL propagation delay from $\overline{AS}$ negating.

Critical Timing Parameters	Required	Guaranteed
Min WRQPDM width	70/90/100 ns	120 ns
Set up CSQPDM to RDQPDM WRQPDM asserting	0 ns	60 ns
Set up Write Data to WRQPDM negating	50/75/100 ns	>180 ns
Read Data valid from DSACKxx asserting	60 ns max	-40/-20/0 ns
DSACKxx asserting with respect to negative clock edge	18.5 ns	12 ns

Where more than one figure appears in a column, the different requirements for different speeds (12, 16 and 20 MHz) of the Am95C60 are reflected. The MC68020 timing requirements are for a 16.67 MHz device. Refer to the Am95C60 Technical Manual and the MC68020 specifications for bus cycle timing details.

Figure 2.3-6 Bus Cycle Timing Diagram

Responding to a Bus Cycle

Having initiated a busy cycle to the Am95C60, logic is required to generate sufficient WAIT states to the MC68020 to generate a bus cycle of acceptable length to the Am95C60. The response lines needing to be set are the DSACK0 and DSACK1 lines. When not asserted, these lines cause WAIT states to be inserted in the bus cycle. At the appropriate times, these lines can be asserted, and the code supplied to the MC68020 indicating the width of the device responding to the bus cycle (1, 2 or 4 bytes wide). The Am95C60 is capable of operating as either an 8- bit or 16-bit wide port, but in this instance, for optimum performance it should be configured as a 16-bit port and hence should respond with a code of 01H. (See Figure 2.3-7)

$\overline{DSACK1}$	$\overline{DSACK0}$	Function
H	H	Insert WAIT states
H	L	Cycle complete - 8-bit wide data port
L	H	Cycle complete - 16-bit wide data port
L	L	Cycle complete - 32-bit wide data port

Figure 2.3-7 DSACK Code Definition

Port No. A1 (ADDR2)	A0 (ADDR1)	Operation	Action (Address pins of QPDM) (System Bus address lines)
0	0	Read	Read Am95C60 Status Register
0	0	Write	Write Instruction FIFO
0	1	Read	Read Block Output I/O FIFO
0	1	Write	Write Block Input I/O FIFO
1	0	Read/Write	Access to I/O Pointer Register
1	1	Read/Write	Access to Internal Register pointed to by the I/O Pointer Register

Figure 2.3-8 Am95C60 Internal Register Address Decode

Once these signals are asserted, the bus cycle will terminate, allowing the MC68020 to read the presented 16 bits of data on a Read cycle or to write 16 bits of data on a Write cycle using the most significant 16 bits of the 32-bit data bus.

Note: The Am95C60 must be configured on the most significant 16 bits of the 32-bit data bus.

Addressing the Am95C60's Internal Resources

The Am95C60 recognizes four addresses in conjunction with CSQPDM, as defined in Figure 2.3-8.

The method of accessing resources within the Am95C60 requires that the I/O Pointer Register be first loaded with the address of the resource to be accessed. Having loaded the I/O Pointer Register with the appropriate value, any subsequent access to the I/O Data Register of the Am95C60 will transfer data between the Bus Master and this resource.

It is important to note that each Am95C60 within a system must be individually addressable to initialize each device (using the Set QPDM Position instruction) to define the position of each Am95C60 within the array of display memory planes.

2.3.4 Dynamic Bus Sizing

The size of any bus cycle is dynamically defined by the SIZO-1 lines generated by the MC68020. Any bus cycle is capable of accessing 1, 2, 3 or 4 bytes on any byte boundary provided that the access does not cross a longword boundary (a longword = 4 bytes). Hence a 32-bit wide memory system would need to decode the SIZO-1 and address lines A0 and A1 together with $\overline{AS}$ and $\overline{DS}$ to determine which of the byte select lines should be asserted on Write cycles to the memory, so as to modify only those bytes defined within any longword.

All resources within the Am95C60 are 16 bits wide. Since any Write cycle (under the control of CSQPDM and WRQPDM) to the Am95C60 will take the 16-bit quantity presented on the bus and load it into the appropriate register, it is essential that data is word aligned. The least significant address line is not used in addressing the resources within the device.

The simplest way of organizing data to be loaded into the Am95C60, either directly from the MC68020 or from system memory under the control of a DMA channel, is to ensure that the data is word aligned and that all bus cycles are word transfers, i.e., avoid instructions that will generate byte accesses to the Am95C60. For word accesses to the IOP register, null data should be used in the most significant byte position on loading, and undefined data will be returned in this byte position when reading.

Note also that as data is transferred a word at a time between the QPDM and the MC68020, the address lines A_1 and A_2 of the MC68020 connect to address pins A_0 and A_1 respectively of the QPDM.

2.3.5 Halt and Bus Error Control

Other response lines which need to be controlled on bus cycles generated by the MC68020 or similar bus master are "Halt" and "Bus Error" (HALT and BERR). These two signals inform the bus master whether or not the bus cycle has terminated successfully, and if not, whether a repeat bus cycle should immediately be attempted (once HALT has been negated), or whether a Bus Error Exception should be taken causing a bus error handling routine, similar to an interrupt routine, to be executed.

In general, the repeat option is used when the pager mechanism or cache control system finds that a virtual address presented does not immediately map to available local memory, but an update mechanism exists which does not require the CPU to execute specific code to update that local memory. The bus master is inhibited from using the bus until HALT is negated, allowing the update mechanism time to do whatever is necessary before negating HALT.

The bus error exception is usually taken if the host processor is required to execute code to correct the fault

that caused the bus cycle to terminate unsuccessfully, such as "Bring in new data from secondary memory or backing store into local memory".

Bus error may be set for other reasons than page fault (the page of memory required is not currently resident in available local memory) such as memory parity error or access protection violation. In general, the pager mechanism will be responsible for detecting these exception conditions and will normally contain the logic to generate these signals.

A special case of when Bus Error can be set is when an Interrupt Acknowledge cycle is generated by the MC68020 but no device is requesting service. If this condition occurs and the Bus Error is asserted in such an Interrupt Acknowledge bus cycle, it is interpreted that Spurious Interrupt has occurred. The Bus Error exception is not taken under this condition. The Interrupt Handling logic of Figure 2.3-11 further describes this condition.

In general, when accessing hardware resources within a device such as the Am95C60 where the resource should always be available, there should never be a need to unsuccessfully terminate the bus cycle, and hence additional logic should not be required beyond what is included in Figure 2.3-11 or would already be present within such a pager system.

If a particular system requires that Bus Error or Halt be driven under specific conditions, it would be a simple task to generate control logic to set these signals as appropriate. (See Figure 2.3-6)

2.3.6 System Bus Arbitration

The Am95C60 can only act as a bus slave, never as a bus master, and hence does not have direct involvement with system bus arbitration to become the bus master. However, the task of loading the Am95C60 with data and instructions can be taken from the host processor and given to a suitable DMA controller to reduce the load on the CPU, thus resulting in greater system performance. Such a DMA controller must interface to the bus arbitration scheme.

2.3.7 Initializing the Am95C60

Two main tasks are involved in controlling the Am95C60. In order to initiate any activity within the graphics engine following power-up reset, the device needs to be initialized with a number of parameters defining the environment in which it resides (such as the size of the Video DRAMs constituting the display memory, whether an 8- or 16-bit bus interface is being used to the system bus, etc.). As stated previously, it is essential that each Am95C60 within a system can individually be chip selected when executing the Set QPDM Position instruction. Once having loaded each Am95C60 position register, most accesses to the array of Am95C60s should set all chip select lines (CSQPDM(0...N), where there are N devices within the system) as all Am95C60s execute the same instruction simultaneously.

Execution of these instruction may have different effects on different display memory planes. This depends on the data already present in display memory or on the contents of certain registers within each Am95C60. The following are some examples: defining which planes are active (activity bits), what color lines should be drawn when executing drawing instructions (color bits), what color is being searched for and on which planes, and when using Area Fill instructions (color search bits and listen bits).

When the appropriate instruction is used to set the desired value in these registers, the instruction has within it a field defining which Am95C60 is being accessed. Each Am95C60 compares this field with the contents of its plane position register to determine whether it is the target for this operation. See Section 13.2.4 of the Technical Manual.

Hence when defining the addresses with the hardware space for the Am95C60s within a system, individual addresses should be allocated for each Am95C60 for use when initializing the devices. A further address should also be allocated causing all CSQPDM(0- N) lines to be asserted for use when accessing all Am95C60s simultaneously (once all the Am95C60s are initialized). Refer to Figures 2.3-3 and 2.3-4.

Once the CPU has initialized the Am95C60, the device is ready to begin executing drawing or data manipulation instructions. Over fifty different instructions are available which can be loaded into the Am95C60 in a number of different ways.

2.3.8 Initiating Am95C60 Activity

Loading Instructions from the Host Processor

The most straightforward method of loading instructions is for the host processor to generate a Write cycle and directly address the Instruction FIFO within the Am95C60 by writing to Port 0 in "hardware space" (see Figure 2.3-8). This method is commonly known as Programmed I/O.

When servicing the Instruction FIFO by the host processor, the FREQ signal may not be directly connected. However, when this is true, the FREQI interrupt, i.e., the Instruction FIFO is half empty, can be "mask-controlled"

to set a bit within the interrupt register of the Am95C60, thereby causing the host to service the Instruction FIFO. Alternatively, the host may poll the status register (Read Port 0) bit 14 to determine whether or not the Instruction FIFO requires service.

Using a DMA Channel to Load Instructions

The Instruction FIFO may be loaded using a dedicated DMA channel, although no Acknowledge signal is available for the DMA channel supporting the instruction FIFO; therefore, "flow-through", not "fly-by", transfers must be supported for this channel.

Flow-through mode means that the DMA channel reads system memory in one bus cycle using the address reloaded into the Source Address Register of the DMA to obtain the instruction to be loaded into the Am95C60. The data is stored in a temporary data register. In the next available bus cycle the DMA writes this data into the Instruction FIFO using the address preloaded into the Destination Address Register of the DMA which should incorporate the port number (Port 0) of the Instruction FIFO of the Am95C60.

The signal FREQ is generated by the Am95C60 indicating that the Instruction FIFO is not full, and hence can be used to request further instructions from system memory under control of the previously initialized DMA channel to keep the Instruction FIFO full.

Using Program Mode to Load Instructions

The third method of loading instructions into the Am95C60 is to use a special instruction that causes the Am95C60 to read instruction from an area in video memory instead of accessing the Instruction FIFO. Thus, once having written a string of instructions into video memory, the Am95C60 can be loaded with the "Call" instruction. When executed, this instruction will start to take subsequent instructions from an area of video memory pointed to by the following operand address pair after the Call instruction. Subsequent Call instructions allow the use of nested subroutines within display memory. Execution control is switched back to the Instruction FIFO either by executing a "Return" instruction (when not in a nested subroutine), or by a reset of the device (hence, the device always initially executes from the Instruction FIFO).

Moving Data Between System and Display Memory

Certain instructions may require data to be written to the Block Input FIFO, or data to be read from the Block Output FIFO when data is being transferred between video memory and system memory, or another resource on the system bus.

Programmed I/O

These FIFOs may be serviced directly by the host processor either by interrupting the host processor on such a condition or by the host processor polling the status register to determine whether the Data FIFOs require service, although this would impose a heavy workload on the host. Alternatively, the Data FIFOs may be serviced using a dedicated DMA channel from a suitable device, such as the Am9516A two-channel DMA controller.

Request and Acknowledge lines are available on the Am95C60 to allow such a DMA channel to support the Data Input and Output FIFOs, thus relieving the host processor of this task.

Using a DMA Channel to Service the Data Input/ Output FIFOs

When data is required on a write-to-display memory, or is ready on a ready-from-display memory, a request is raised (DREQ) by the Am95C60 to request service of the appropriate data FIFO. The FIFO may be serviced directly by the host CPU by reading or writing the appropriate port on the Am95C60 (Port 1) or by using a suitably initialized DMA channel.

The DMA channel's request input may be linked via an inverting gate to the DREQ signal. Using the ACKD associated with this DMA channel, "fly-by" transfer can be achieved between Data FIFOs and system bus. Hence, whenever the Data FIFOs require service, no further host processor intervention will be required, provided the DMA channel has been initialized with the start address of the area of system memory to be used and the number of words to be transferred to/from system memory.

On completion of each data transfer instruction or on initiation of the next data transfer instruction, the host processor needs to be informed so that it can initialize the Data DMA channel with the relevant parameters for the next data transfer instruction to be executed.

Note: The port number value on address lines ADDR2 and ADDR1 (QPDM pins A1 and A0) need not be valid during DMA transfers using the DREQ and ACKD lines, since these form part of the address to system memory. Port 1 is assumed by the QPDM.

Using a DMA Channel to Service Multiple Am95C60s

Since all Am95C60s begin execution of the same instructions at approximately the same time, they will require their Instruction and Data FIFOs to be serviced at the same time.

As the instruction stream to each Am95C60 is held in an on-board FIFO, the ripple-through delay of each FIFO may be sufficiently different to cause different Am95C60s to detect and begin execution of an instruction on different clock edges. Hence, for a system containing multiple Am95C60s, their instruction execution may initially be skewed by one clock cycle. This problem is resolved by using the MATIN and MATOUT lines between QPDMs to re-synchronize and ensure all devices are in step.

To ensure that all Am95C60s are ready for the DMA transfer to begin, all FREQ and DREQ lines are connected together, effectively implementing a "Wire-AND" function for each signal. Until all devices are ready, the resultant line will not be asserted. This is possible as these signals are of "open drain" construction (active HIGH), and as such require a pull-up resistor to +5 V. These are then inverted to generate active LOW DMA channel requests.

2.3.9 Bus Interface Control

Six instructions are provided within the Am95C60s instruction set to facilitate transfer of data between the system bus and display memory. These are the Output Block, Input Block and Store Immediate instructions for reading and writing display memory. Each can either use the current pen position or use the address specified within the instruction as the target area within display memory.

2.3.10 Data Transfers by Plane or Pixel

When transferring between the system bus and memory, two options are available under the control of the Z bit within the instruction field to define whether data should be accessed by plane or by pixel (Z=0 transfer by plane; Z=1 transfer by pixel).

Display Memory Access by Plane

When reading display memory by plane, the activity bits associated with each display memory plane must be set, using the Set Activity Bit instruction, so that only the plane involved with the data transfer is active. All other plane activity bits must be reset. Hence the only Am95C60 to generate an enable (EDE) to control the bidirectional buffer linking the Am95C60's 16-bit data port to the system bus will be the device with an activity bit set for one of the planes for that it has control. When writing display memory by plane, multiple activity bits may be set if identical data is to be written to more than one plane.

Display Memory Access by Pixel

When accessing the display memory by pixel using the Input and Output Block instructions, more than one plane will be accessed concurrently. Any number of activity bits may be set during the execution of the instruction. When executing such an Input or Output Block (by Pixel) instruction, the Block Input Step (BIS) field defines the number of pixels contained in each 16-bit data word.

Bidirectional Buffer Enable Control

The control of the enables of the bidirectional buffers will be more complex when using Input or Output Block Transfer instructions by pixel within a Multi-QPDM system, since each Am95C60 within the system will need to transfer four bits (relating to their four planes) to be assembled into the 16-bit value to be presented to the system bus.

In a multi-QPDM system, additional bidirectional buffers will be required that are only enabled when using this mode to interface these four bits of data from each Am95C60 to the 16-bit data bus. Figure 2.3-9 shows an example for a 2-QPDM system. The Input Block Section in Chapter 14 of the Technical Manual shows the recommended connections for all possible system sizes.

By Plane or By Pixel

Control of the bidirectional buffer enables can be achieved by using a PAL16R4 to decode the EDE lines from each Am95C60, and by using a register (I/O mode register) within the PAL device that can be set by the host processor prior to the loading and execution of a Block Input or Output instruction. This register will define whether the Block I/O transfer is to be done by pixel (register set) or by plane (register reset). Responsibility lies with the software to ensure that the register is set to the appropriate state to match subsequent Block I/O instructions; for example, if the register defines, "Transfer by Pixel", then the subsequent Block I/O instruction should also define "Transfer by Pixel" using the "Z" field within the instruction. The appropriate enables to each buffer can be generated by the PAL device.

A simple way to set and reset this "Pixel/Plane" I/O mode register could be to allocate two addresses within "hardware space", one to define "setting" the register and the other to define "resetting" the register. The host would then only need to generate a bus cycle to the appropriate address to set the I/O mode register to the desired value. This is only one of many different ways of implementing this function.

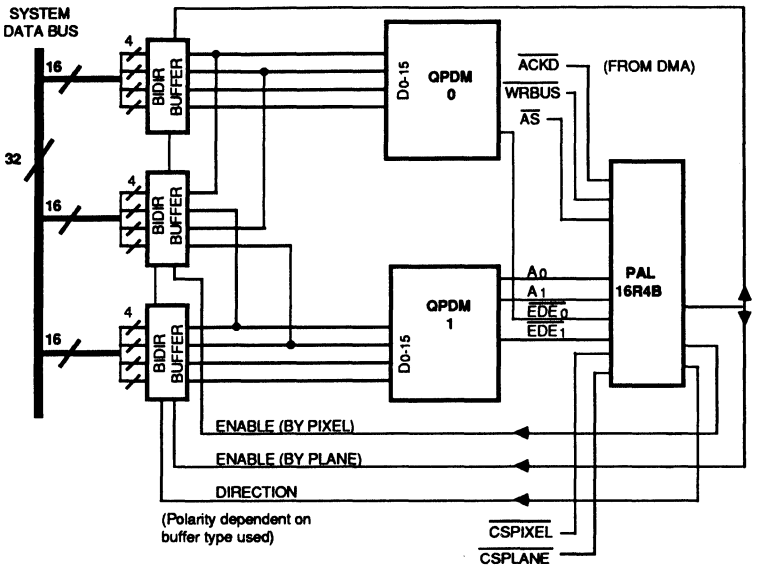


Figure 2.3-9 Transceiver Configuration to System Bus (2-QPDM System)

Note: The DSACK logic would need to respond to an access to these addresses, otherwise the system bus would lock up with infinite WAIT states inserted.

Bidirectional Buffer Direction Control

The direction control of the buffers is a simple decode of the system write (WRBUS) line and the acknowledge (ACKD) line from the DMA channel associated with the Data FIFOs. The ACKD line is required to indicate that the sense of the WRBUS line is inverted when transferring data between the system bus and the Data FIFOs in "fly-by" mode. An example of the PAL code to achieve this function for a 2 QPDM system is shown in Appendix 2.

2.3.11 Interrupt Handling

There are a number of conditions that cause the Am95C60 to raise interrupts to the host processor to inform it of some specific event or an illegal condition. The different types of interrupts that the Am95C60 can generate are listed in Figure 2.3-10 All interrupts are maskable.

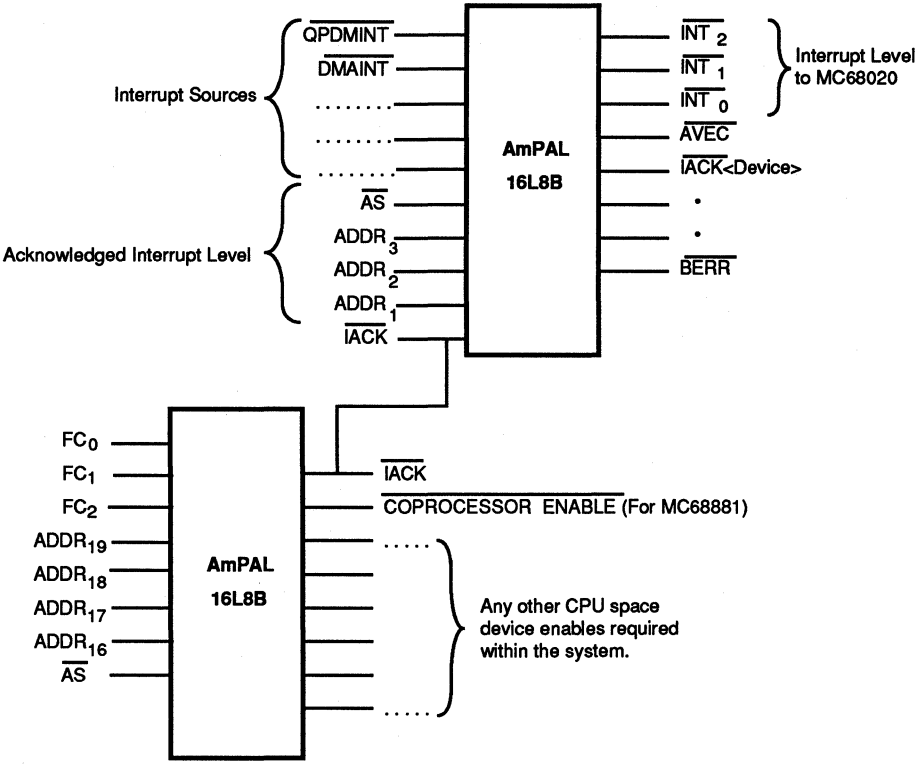
The interrupt signal from the Am95C60 typically will be connected to some priority encode scheme, so that all the sources of interrupt within the system can be arranged by priority. (See Figure 2.3-11)

— Idle
— Stack Overflow
— Display Memory Boundary Crossed
— Clipping Boundary Crossed
— Frame
— FREQ (Instruction FIFO DMA Control)
— DREQ (Data FIFO DMA Control)
-- Vertical Blank
— Software (The SIGNAL Instruction)* or
Picking Detect (Non-maskable)*

*Note: "Software" and "Picking Detect" conditions set the same interrupt bit in the interrupt register. However, "Picking Detect" is only enabled when picking is enabled.

Figure 2.3-10 Am95C60 Interrupt Sources

Decode Logic Schematic:



Note – An IACK cycle uses a Function Code of 7H (defining CPU space access), and address lines 16 to 19 contain a code of OFH.

The interrupt level being acknowledged is asserted by the MC68020 on address lines 1 to 3.

$\overline{\text{AVEC}}$ and $\overline{\text{IACK}}$ <Device> are mutually exclusive.

No DSACK response should be generated on bus cycles when $\overline{\text{AVEC}}$ is asserted.

Figure 2.3-11 Interrupt Handling Logic

The MC68020 Interrupt Sequence

The interrupt sequence of the MC68020 is as follows:

The MC68020 monitors the level of the $\overline{IPL}_{0-2}$ lines. When a non-zero level is detected for at least two consecutive system clocks, the MC68020 internally flags that a genuine external interrupt condition exists. Note that the $\overline{IPL}_{0-2}$ lines are active LOW.

If this interrupt level present on the the $\overline{IPL}_{0-2}$ lines is greater than the current interrupt level, this will cause the host processor to "stack" the state of the machine on completion of the current instruction. The MC68020 will then generate an Interrupt Acknowledge (Read) Bus Cycle (IACK cycle) to determine which interrupting device for any particular interrupt level has raised the interrupt. The device with an interrupt pending can respond to this bus cycle by supplying a vector number that is used to index into an Interrupt Address table to point to a unique program subroutine to service that particular interrupt.

Using Autovectors

It is not essential that the interrupting device respond to this IACK cycle by providing a vector number and generating a DSACK response as previously described. Instead, the device can cause the AVEC line to be asserted to the MC68020, indicating that the Autovector for this particular interrupt level should be used. Within the Vector Address table, this causes a specific entry unique for each interrupt level, to be used as the source of the interrupt service routine start address, instead of using the returned vector number to index into the Vector Address table to provide this start address.

Using the Autovector system to respond to the IACK cycle, caused by the Am95C60 interrupt, makes the hardware support more simple, as the Am95C60 does not have a specific on-board register to hold an interrupt vector number. However, should a particular system require it, it would be relatively simple to use an external register to hold the vector number enabled by a suitable signal from the "Hardware Space" decode logic. When using the Autovector feature, the Am95C60 does not need to be informed that the IACK cycle has occurred at this time, but decode logic can set the AVEC line to the MC68020 in response to the IACK cycle. (See Figure 2.3-11)

On entering the interrupt service routine for the Am95C60, the software should read the Status Register of the Am95C60. From this register the software can determine that interrupt conditions currently require service. To clear the relevant bits within the Am95C60, a Write to the Interrupt Acknowledge Register should be issued defining that bits of the Interrupt Register are to be reset. If no further interrupt conditions have become set since reading the Status Register, this Write to the Interrupt Acknowledge Register will cause the interrupt line from the Am95C60 to be negated.

If, after the Status Register has been read to determine outstanding interrupts, another interrupt condition occurs before the Write Interrupt Register occurs, then this newly set bit will not be cleared by the Write to the Interrupt Acknowledge Register, and hence the interrupt line will not be reset. This does not cause a problem, however, since when the interrupt routine completes and tries to return to the previous interrupt level, the MC68020 will again be interrupted due to this new interrupt condition. The interrupt routine will again be entered, allowing this new bit to be read, cleared and serviced. This mechanism guarantees that no interrupt will be lost.

Note that if an IACK cycle occurs but the interrupt condition causing the IACK cycle to be generated by the MC68020 is no longer set, then a Spurious Interrupt has occurred. Under this circumstance, the IACK cycle should be responded to with Bus Error, not DSACKxx or AVEC. The MC68020 will interpret the Bus Error signal to indicate that the Spurious Interrupt vector should be used as the entry address into an interrupt routine. The MC68020 will not take the Bus Error exception.

Reading the Status Register

When reading the Status Register of an Am95C60, the device must be explicitly addressed using the unique address associated with that Am95C60, as used when setting the Plane Position Register of each Am95C60. If the global address is used, multiple devices will attempt to drive the system bus simultaneously. All QPDMs will, however, contain the same value in their Status Registers.

System Interrupt Priority

To prioritize a number of interrupts, some of which may cause the same interrupt level, a PAL device may be used simply to look at the currently outstanding interrupts. The relevant interrupt level can than be asserted on the $\overline{IPL}_{0-2}$ lines to the MC68020. All devices must hold their interrupt lines asserted until the device receives an appropriate Acknowledge to their interrupt. Those devices that can return an Interrupt Vector should have an individual IACK line asserted to them during the IACK cycle, informing the device to return the vector number. Again, a PAL device can be used to generate the individual IACK lines, assuring that only one device responds to any IACK cycle. (See Figure 2.3-11).

Appendix 3 shows an example of the Source Code for such a PAL device.

For all devices that use the Autovector facility, no individual IACK line need be set to that device, only the AVEC line asserted to the MC68020. As described above, the device detects that the interrupt is being serviced (interrupt acknowledged) when the Interrupt Acknowledge Register is written. which will clear the relevant interrupt bits that, when set, cause the interrupt line to be asserted.

Interrupt Handling within a Multi-Am95C60 System

As all Am95C60s within a system execute the same instruction in synchronism, any interrupt conditions detected by one Am95C60 will also be detected by all other Am95C60s.

By reading the Status Register of any Am95C60, any outstanding interrupt condition across all Am95C60s can be detected. To clear such an interrupt condition across all Am95C60s, a Write to the Interrupt Acknowledge Register of all Am95C60s can be achieved simultaneously, thus causing the desired interrupt condition to be acknowledged and cleared.

Using the Arbitration and Bus Cycle Response schemes implemented by the MC68020, it is simple to interface two devices on the same bus, each running asynchronously from their own clock source.

To gain the most performance from the MC68020, the device should be operated at the highest clock rate defined within the specification of the part (currently 16.67 MHz). However, the Am95C60 is capable of running on a 20 MHz clock for maximum performance in drawing and data transfer operations. If maximum performance is desired from each device, then each will run from asynchronous clock sources.

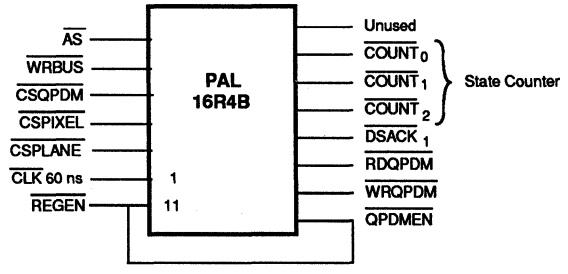
Since the definition of the Bus Cycle for accesses to the Am95C60 does not define a relationship to the Am95C60 clock, the $\overline{\text{CSQPDM}}$(N), $\overline{\text{WRQPDM}}$ and $\overline{\text{RDQPDM}}$ signals may be asserted in synchronism with the MC68020 clock. Any asynchronicity will be handled by the Am95C60 provided that the maximum and minimum specified figures are complied with. Conversely, in responding to the bus cycle, it is feasible to generate the DSACK response signals from the Am95C60 clock, as the MC68020 has internal logic to resynchronize these signals to the MC68020 clock. Depending on whether the response is synchronous or asynchronous, different timings are given in the MC68020 timing definitions, defining the specification of the DSACK response signals.

DSACK Response Generation

Dependent upon other system constraints, it will probably be more simple to generate the DSACK responses synchronously to the MC68020 clock using a fixed delay logic sequence to define the length of any access to the Am95C60. On detecting an access to an Am95C60, sampling using the MC68020 clock, a timing sequence can be initiated that at some programmed delay after detecting the access can generate a synchronous DSACK response. An example of how this may be implemented is shown in Figure 2.3-12.

Example of How DSACK Response May be Generated

Using a PAL device with pull-up resistors on each registered output, the registers can be clocked with the



NOTE 1: 1 k Ohm pull-up resistors are required on the register three-state outputs.

NOTE 2: The $\overline{\text{CSQPDM}}$ signal must be synchronous to the clock.

Figure 2.3-12 DSACK Response Generation using a PAL

synchronous MC68020 clock (clocking on the negative edge of the MC68020 clock).

The signal $\overline{\text{CSQPDM}}$ indicates that an access has occurred to at least one Am95C60. The delay from address strobe to this signal being asserted is totally dependent on the speed of the pager system plus the delay of the logic that decodes the Hardware Space address, and so may be totally asynchronous to the MC68020 clock. However, WAIT states will be inserted in the bus cycle during this period, since no DSACK response has yet been given.

Depending upon how the pager system is designed, the enables to either memory or hardware resources may or may not be synchronous. For example, if a state machine is used to search look-up tables, then it may be as simple to generate a synchronous enable of the hardware decode logic.

If a direct addressing scheme is used, then $\overline{\text{CSQPDM}}$ will not suffer the delay of the pager system, just the delay of the address decode PAL device.

With the advent of "B" speed PALs, a solution is now feasible using a PAL16R4B, as shown in Figure 2.3-12. This solution offers a simple method of generating the Read and Write strobe signals to the QPDM and DSACKxx response signals to the MC68020 within the framework of a MC68020 bus cycle. Appendix 4 shows an example of the Source Code for such a PAL device.

How to Solve Signal Asynchronicity

If the synchronicity of these enables with respect to the MC68020 clock cannot be guaranteed, then logic will be necessary to re-synchronize such an enable. The enable would be used to initiate the DSACK response logic for accesses to the Am95C60 to ensure that metastability problems cannot cause the logic to function erroneously.

A simple way to re-synchronize an asynchronous signal to a clock of at least 60 ns is shown in Figure 2.3-13. Even if the first 74S74 (or 74F74) goes metastable, the output can be guaranteed (within reason) to settle to either a HIGH or LOW within 60 ns. Hence the second 74S74 will be guaranteed not to go metastable, as its data set-up time with respect to the next clock cycle will be met. The output of the second 74S74 will, therefore, be synchronous with the MC68020 clock.

Using a suitable synchronous signal, further control of the Set or Clear pin (dependent on whether the input signal is LOW or HIGH true) can allow the clearing of the synchronous output without having to suffer the two clock delay of the register pipeline. Such a suitable signal in the case of $\overline{\text{CSQPDM}}$ is AS, that runs synchronously to the negative edge of the MC68020 clock.

DSACK Sequence Logic

Using a B-speed PAL guarantees that the DSACK response is set within the requirement of 18.5 ns(worst case) of the negative edge of the MC68020 clock. The maximum "B" speed PAL register outputs from clock delay is 12 ns. This meets the asynchronous set-up time of the DSACK response with respect to the MC68020 clock, hence defining exactly on that clock edge the MC68020 will detect DSACK asserted. The MC68020 will negate $\overline{\text{AS}}$ on the next negative edge of the processor clock. This will help define the minimum possible bus cycle time to maximize bus throughput for maximum efficiency of data transfer instruction between system bus and display memory.

The PAL is designed to use three outputs as a state counter initiated by a synchronous $\overline{\text{CSQPDM}}$. Depending on the level of $\overline{\text{WRBUS}}$ from the MC68020, either $\overline{\text{RDQPDM}}$ or $\overline{\text{WRQPDM}}$ is asserted. The timing is controlled by the state counter. (See timing diagram in Figure 2.3-6) The PAL code in Appendix 2 is annotated to explain the operation of the equations.

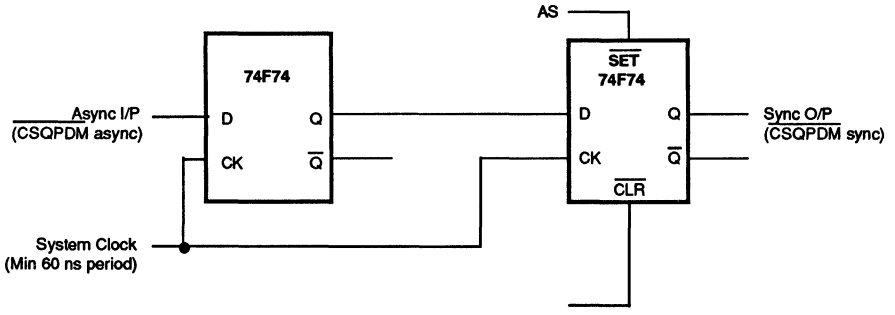


Figure 2.3-13 Resynchronization Logic

2.3.12 Conclusions

This Application Note offers an example of a solution to the problem of interfacing a QPDM Graphics Engine to a MC68020 processor. While offering a solution to this interface problem, the author realizes that many other system constraints may exist that may require this example solution to be modified to fit within system requirements.

In discussing the areas of design that need careful consideration, a comprehensive description is included of the way in which both the MC68020 and QPDM fit within a system and interface to one another. This should help the designer to quickly understand the operation of both devices' interface requirements and modify this design appropriately.

APPENDIX 1 — Address Decode PAL Source Code

```
DEVICE      CPU_SPACE_ADDRESS_DECODE (pal1618)

PIN         FC[0:2] = 1:3       " THESE ARE FUNTION CODE LINES"
                                " FROM THE Mc68020 DEFINING SUPERVISOR OR"
                                " USER, CODE OR DATA SPACE"

            ADDR[16:19]         = 4:7     " THESE ADDRESS LINES DEFINE"
                                          " THE TYPE OF CPU SPACE ACCESS"
                                          " WHEN THE FC VALUE IS 7."
            /AS                 = 8       " ADDRESS STROBE FROM THE 68020"

            IACK                = 12      " O/P INDICATING AN INTERRUPT"
                                          " ACKNOWLEDGE BUS CYCLE IS OCCURRING"

            EN881               = 13;     "ENABLE FOR MC68881"

            " ANY OTHER CPU SPACE ENABLES REQUIRED"

DEFINE

" THE FOLLOWING VALUES ARE DEFINED IN THE 68020 SPEC."

            CPUSPACE = FC[0] * FC[1] * FC[2];

            IACKEN = ADDR[16] * ADDR[17] * ADDR[18] * ADDR[19];

            COPROCEN = /ADDR[16] * ADDR[17] * /ADDR[18] * /ADDR[19];

BEGIN

            " OUTPUT DEFINITION EQUATIONS"

            /IACK = CPUSPACE * IACKEN * AS;

            /EN881 = CPUSPACE * COPROCEN * AS;

END.

            Listing sum-of-products equations for CPU_SPACE_ADDRESS_DECODER

/IACK = FC[0]*FC[1]*FC[2]*ADDR[16]*ADDR[17]*ADDR[18]*ADDR[19]*AS;

/EN881 = FC[0]*FC[1]*FC[2]*/ADDR[16]*ADDR[17]*/ADDR[18]*/ADDR[19]*AS;
```

APPENDIX 2 — I/O Mode PAL Source Code

```
DEVICE BUS_XCEIVER_CONTROLLER (PAL16R4)

PIN

        /AS         = 1     "This is the clock for the I/O Mode reg"
        /WRBUS      = 3     "Write signal for system bus (68020)"
        ADDR[1:0]   = 4:5   "The system address lines defining which
                             port of the QPDM is being accessed. -

                            Note: These pins are connnected to System
                            Address lines A2 & A1.  Refer to App Note."

        /CSPIXEL    = 6     "Line from Address Decode PAL defining access
                             to the I/O Mode reg, setting the register."

        /CSPLANE    = 7     "Line from Address Decode PAL defining access
                             to the I/O Mode reg, resetting the reg."

        ACKD        = 8     "Acknowledge line from the DMA controller."
                            "Ensure correct polarity for DMA in use!"

        /EDE0       = 9     "Buffer enable line from QPDM0."
        /EDE1       = 11    "Buffer enable line from QPDM1."
        DIR         = 12    "Buffer Direction Control Output."
                            "Ensure correct polarity for Bi-Di buffers
                             in use!"

        OEPIX       = 13    "Output Enable for BY PIXEL Buffer."
        PIXMOD      = 14    "The I/O Mode Register - Set = Pixel Mode
                                                     Reset = Plane mode."
        OE0         = 18    "Output Enable of the BY PLANE buffer (QPDM0)"
        OE1         = 19;   "Output Enable of the BY PLANE buffer (QPDM1)"

BEGIN

                "The sense of the WRBUS:L signal is inverted when
                 the DMA is controlling the Bus Cycle."

        /DIR        = WRBUS * /ACKD
                    + /WRBUS * ACKD;

        "Note this Register clocks on the trailing (rising) edge of Address Strobe"

        /PIXMOD := PIXMOD * /CSPIXEL * /CSPLANE     "Leave the I/O Mode register
                                                     contents unchanged (clock back
                                                     in current contents), if access
                                                     not to I/O mode register."

                    + CSPIXEL;                      "Set reg if CSPIXEL is set when
                                                     Address Strobe negates."

        /OE0        = /PIXMOD * EDE0                "Enable PLANE Buffer (QPDM0) if
                                                     EDE0:L set & NOT in pixel mode."
```

```
                    + EDE0 * /(/ADDR[1] * ADDR[0]);  "This last term enables the PLANE
                                                      Buffer if EDE0 is set AND the
                                                      access is not to the Data FIFO's -
                                                      PORT 1."

          /OE1      = /PIXMOD * EDE1                 "Enable PLANE buffer (QPDM1) if
                                                      EDE1:L set & NOT in pixel mode."

                    + EDE1 * /(/ADDR[1] * ADDR[0]);  "Comment as for OE0 above."

          "Enable the PIXEL buffer if in Pixel Mode AND the relevant EDEx is asserted
           AND the access is to PORT 1 or the acknowledge line from the DMA controller
           is set.

           i.e., don't enable the Pixel Buffer if the access is to any resource other
           than the Data FIFOs."

/OEPIX = (PIXMOD * (EDE0 + EDE1)) * ((/ADDR[1] * ADDR[0]) + ACKD);
END.

          Listing sum-of-products equations for BUS_XCEIVER_CONTROLLER

/DIR     = WRBUS*/ACKD
         + /WRBUS*ACKD;

/PIXMOD  := PIXMOD*/CSPIXEL*/CSPLANE
          + CSPIXEL;

/OE0     = /PIXMOD*EDE0
         + EDE0*ADDR[1]
         + EDE0*/ADDR[0];

/OE1     = /PIXMOD*EDE1
         + EDE1*ADDR[1]
         + EDE1*/ADDR[0];

/OEPIX   = PIXMOD*EDE0*ACKD
         + /ADDR[1]*ADDR[0]*PIXMOD*EDE1
         = PIXMOD*EDE0*/ADDR[1]*ADDR[0]
         + PIXMOD*EDE1*ACKD;
```

APPENDIX 3 — Interrupt PAL Source Code

```
DEVICE INTERRUPT_CONTROLLER_PAL (PAL16L8)
PIN

          /AS        = 1              "ADDRESS STROBE FROM THE 68020"
          /IACK      = 2              "SIGNAL INDICATING AN INTERRUPT ACK."
                                      "CYCLE IS OCCURRING (FROM CPUADEC PAL)"

          ADDR[1:3}          = 3:5    "THESE ADDRESS LINES FROM"
                                      "68020 DURING AN IACK CYCLE"
                                      "DEFINE THE INTERRUPT LEVEL"
                                      "BEING ACKNOWLEDGED."

          QPDMINT            = 6      "INTERRUPT SIGNAL FROM THE
                                      "QPDM GRAPHICS SYSTEM."

          DMAINT             = 7      "ENSURE CORRECT POLARITY"
                                      "FOR THE DMA TO BE USED!!"

          "ANY OTHER INTERRUPT INPUTS SHOULD BE DEFINED HERE"

          IPL[0:2]           = 13:15  "INTERRUPT PRIORITY LEVEL"
                                      "SIGNALS TO THE Mc68020."

          AVEC               = 16     "AUTOVECTOR SIGNAL TO 68020"

          IACKDMA            = 17     "INTERRUPT ACKNOWLEDGE LINE TO"
                                      "THE DMA CONTROLLER."

          BERR               = 18;    "BUS ERROR signal to 68020 to
                                       warn of spurious interrupt cond."

          "ANY INDIVIDUAL INTERRUPT ACKNOWLEDGE LINES SHOULD
          "BE DEFINED HERE & ALLOCATED O/P PINS APPROPRIATELY."

DEFINE

          "Let us assume QPDMINT relates to interrupt level 1, and DMAINT relates to
           interrupt level 2 (Clearly these are totally user definable).  Hence QPDMIACK
           will relate to an Interrupt Acknowledge cycle to a level 1 interrupt, and  as
           such we require to set AVEC:L to the Mc68020.  The DMA may, however, require an
           IACK line directly, hence won't set AVEC:L but will set the IACK line to be
           connected to the DMA device."

          QPDMIACK = ADDR[1] * /ADDR[2] * /ADDR[3];

          DMAIACK = /ADDR[1] * ADDR[2] * /ADDR[3];

          "DEFINE ANY OTHER INTERRUPT LEVELS RELATING TO DEVICES"
          "WITHIN THE SYSTEM, HERE. I.E. THE DMA, SCSI, etc."

          "Note : As this is a priority encoder, ensure that if an interrupt, say level 5,
          becomes set, then all the equations relating to lower interrupt levels (levels 1
          to 4) must be inhibited.
```

```
          Hence all the equations must have terms within them which only allow the equation
          to take effect if this is the highest current interrupt level"

          IF (QPDMINT * /DMAINT) THEN /IPL[2:0] = 1;

          IF (DMAINT) THEN /IPL[2:0} = 2;

          "The PAL should only respond to an interrupt acknowledge cycle if the interrupt
          causing the IACK cycle to occur is still asserted.  If it is no longer set then
          BERR should be asserted."

/AVEC = QPDMIACK * IACK * AS * QPDMINT;                  "AN INTERRUPT ACKNOWLEDGE"
                                                         "BUS CYCLE IS OCCURRING, AND"
                                                         "THE INTERRUPT LEVEL BEING"
                                                         "ACKNOWLEDGED BY THE 68020"
                                                         "IS QPDMIACK (DEFINED AS 1"
                                                         "ABOVE)."

/IACKDMA = DMAIACK * IACK * AS * DMAINT;                 "THE DIRECT IACK LINE TO THE"
                                                         "DMA DEVICE."

"DEFINE HERE THE BOOLEAN EQUATION FOR ANY INDIVIDUAL INTERRUPT"
"ACKNOWLEDGE LINES TO INDIVIDUAL DEVICES WITHIN THE SYSTEM."

/BERR = IACK * AS * /((QPDMIACK * /QPDMINT) + (DMAIACK * /DMAINT));

          "Note: If an IACK cycle occurs and no interrupt is outstanding, then don't set
           DSACK or AVEC, but set BERR.  The Mc68020 will take this to mean that a spurious
           interrupt has occurred, and will not take the BUS ERROR trap, but will use the
           SPURIOUS INTERRUPT vector.

           Other reasons for setting BERR are discussed in the Applications Note, but are
           dependent on other system constraints, and as such can't be included in this
           example.

           In many applications, no other condition may need to set BERR."

          Listing sum-of-products for INTERRUPT_CONTROLLER_PAL

/IPL[2] = 0;

/IPL[1] = DMAINT;

/IPL[0] = QPDMINT*/DMAINT;

/AVEC = ADDR[1]*/ADDR[2]*/ADDR[3]*IACK*AS*QPDMINT;

/IACKDMA = /ADDR[1]}*ADDR[2]*/ADDR[3]*IACK*AS*DMAINT;

/BERR = IACK*AS*ADDR[3]
      + IACK*AS*/ADDR[2]*/ADDR[1]
      + IACK*AS*ADDR[2]*ADDR[1]
      + IACK*AS*DMAINT*QPDMINT
      + IACK*AS*ADDR[1]*QPDMINT
      + IACK*AS*DMAINT*/ADDR[1];
```

APPENDIX 4 — DSACK PAL Source Code

```
DEVICE QPDM_68020_INTERFACE_CONTROLLER (PAL16R4)

PIN

          CLOCK    = 1              "This is the inverse of the clock to the Mc68020,
                                     i.e. register clock on the falling edge of the
                                     processor clock."
          /WRBUS   = 2
          /AS      = 3
          /CSQPDM  = 4
          /CSPIXEL = 5
          /CSPLANE = 6
          /ENQPDM  = 11
          QPDMEN   = 12
          COUNT[0:2] = 16:14
          DSACK1   = 17
          RDQPDM   = 19
          WRQPDM   = 18;

DEFINE
          "Start sequence counter if the current state is 0 and CSQPDM:L (or CSPIXEL:L
           or CSPLANE:L) are true, i.e. on the next clock edge step into state 1.

           Once the sequence is started, the sequencer should run through states 1 to 5
           before returning to state 0, and remaining there (Idle) until the start
           conditions are again detected.  The sequencer will stick in state 5 until
           Address Strobe is negated.  On the next clock it will return to Idle ready to
           start the sequence again when the starting conditions again become true."

ACCESS = CSQPDM + CSPIXEL + CSPLANE;                    "Defines which accesses DSACK
                                                         should be generated for"

START = /COUNT[0] * /COUNT[1] * COUNT[2] * ACCESS * AS;

IDLE = /(ACCESS * AS);

"These expressions define values to the labels STATEx"

          STATE0 = /COUNT[0] * /COUNT[1] * /COUNT[2];
          STATE1 =  COUNT[0] * /COUNT[1] * /COUNT[2];
          STATE2 = /COUNT[0] *  COUNT[1] * /COUNT[2];
          STATE3 =  COUNT[0] *  COUNT[1] * /COUNT[2];
          STATE4 = /COUNT[0] * /COUNT[1] *  COUNT[2];
          STATE5 =  COUNT[0] * /COUNT[1] *  COUNT[2];
          STATE6 = /COUNT[0] *  COUNT[1] *  COUNT[2];
          STATE7 =  COUNT[0] *  COUNT[1] *  COUNT[2];

          HOLD5 = STATE5 *  AS;
          END5  = STATE5 * /AS;

BEGIN
```

```
"These equations define the progression of the states, once the sequencer has been initiated."

          IF (START)  THEN /COUNT[2:0] := 1;
          IF (IDLE)   THEN /COUNT[2:0] := 0;
          IF (STATE1) THEN /COUNT[2:0] := 2;
          IF (STATE2) THEN /COUNT[2:0] := 3;
          IF (STATE3) THEN /COUNT[2:0] := 4;
          IF (STATE4) THEN /COUNT[2:0] := 5;
                                                    "Default reset states"
          IF (STATE6) THEN /COUNT[2:0] := 0;
          IF (STATE7) THEN /COUNT[2:0] := 0;

                          "Return to Idle (state 0) if in state 5 AND Address Strobe
                           has been negated.  If in state 5 and AS:L is still asserted,
                           then wait in state 5."

          IF (HOLD5) THEN /COUNT[2:0] := 5;
          IF (END5)  THEN /COUNT[2:0] := 0;

          "Only enable the DSACK line if the current bus cycle is to a QPDM or the
           I/O Mode register."

/QPDMEN = ACCESS * AS;        "This is connected to pin 11, the register enable, so that
                               the DSACK lines are only driven by the PAL when the QPDM's
                               or I/O Mode register are being accessed."

/DSACK1 := STATE2 + STATE3;

          "Set DSACK1 on entering state 3 and hold on during state 4.

           Note the clock to o/p delay must be less than 18.5 ns to meet the DSACKxx
           synchronous timing of the Mc68020."

          "Set WRQPDM if the current cycle is a write (WRQPDM is true) and the sequencer
           is in states 1 or 2.  This guarantees a Write Strobe to QPDM of 120 ns - QPDM
           requires a min Write Strobe of 70 to 110 ns depending on which speed QPDM is
           being used.

           Write data is guaranteed stable for 60 ns after negating WRQPDM, as Address
           Strobe will not be negating WRQPDM, as AS:L will not be negated by the
           68020 until effectively state 4.  Required data hold time is 0 ns.

/WRQPDM = CSQPDM * WRBUS * (STATE1 + STATE2);

          "The timing of when the 68020 samples read data, is associated with the timing
           of the assertion of the DSACKxx lines.  Data must be stable from the QPDM to the
           68020 within 50 ns of DSACK being asserted.

           This PAL ensures that the DSACK1 line will be asserted at least 120 ns after
           RDQPDM.

           The QPDM will guarantee stable data after a max of 80, 100 or 120 ns (depending
           on which speed QPDM) from RDQPDM being asserted, hence the data from the QPDM
           will be stable within the requirements of the 50 ns of DSACK being asserted
           required by the 68020.
```

```
            Data is sampled on the next negative edge of the 68020 clock after DSACK is
            asserted (in the synchronous case), i.e., at the end of state 3.  The data will
            be held stable by the QPDM well beyond this point under the control of RDQPDM
            negating, and hence meets the Data Hold time requirements of the 68020 (0 ns in
            the synchronous mode)."

/RDQPDM = CSQPDM * /WRBUS * (STATE 1 + STATE2 + STATE 3 + STATE4);
END.

          Listing sum-of-products equations for QPDM_68020_INTERFACE_CONTROLLER

/COUNT[2]  := /COUNT[1]*COUNT[2]*AS
            + COUNT[0]*COUNT[1]*/COUNT[2]
            + /COUNT[0]*/COUNT[1]*COUNT[2];

/COUNT[1]  := COUNT[0]*/COUNT[1}*/COUNT[2]
            + /COUNT[0]*COUNT[1]*/COUNT[2];

/COUNT[0]  := /COUNT[1]*COUNT[2]*AS
            + /COUNT[0]*COUNT[1]*/COUNT[2]
            + /COUNT[0]*/COUNT[1]*COUNT[2]
            + /COUNT[0]*/COUNT[1]*AS*CSPLANE
            + /COUNT[0]*/COUNT[1]*AS*CSPIXEL
            + /COUNT[0]*/COUNT[1]*AS*CSQPDM;

/QPDMEN     = AS*CSPIXEL
            + AS*CSPLANE
            + AS*CSQPDM;

/DSACK1    := COUNT[1]*/COUNT[2];

/WRQPDM     = CSQPDM*WRBUS*COUNT[0]*/COUNT[1]*/COUNT[2]
            = CSQPDM+WRBUS*/COUNT[0]*COUNT[1]*/COUNT[2];

/RDQPDM     = CSQPDM*/WRBUS*COUNT[0]*/COUNT[2]
            = CSQPDM*/WRBUS*COUNT[1]*/COUNT[2]
            + CSQPDM*/WRBUS*/COUNT[0]*/COUNT[1]*COUNT[2];
```

CHAPTER 3

Display Memory Bus

CHAPTER 3
Display Memory Bus

In this section we cover the Display Memory Bus. In Section 3.1, we present a detailed description of a multi-bank display memory based on 64K * 4 VRAMs. This design can be easily extended to 256K * 4 devices. In Section 3.3, we present a method of connecting relatively slow ROMs to the Display Memory Bus to store very large fonts. While these designs have not been built and tested, they have undergone rigorous "paper testing". In Section 3.2, we present the listing of a program that perform a numerical analysis of the timing margins for a QPDM Display Memory Bus.

For a detailed analysis of a demonstration /evaluation board that was built and tested, refer to Section 5.

3.1 DISPLAY MEMORY CONNECTIONS OF THE QPDM

This section describes the connections between the Quad Pixel Dataflow Manager (QPDM), the display memory, and the specialized video shift register, Video Data Assembler FIFO (VDAF).

3.1.1 System Configuration and Block Diagram

The system contains one QPDM and therefore interfaces to four display memory planes. Each plane in this system is 2048 pixels by 2048 pixels.

The block diagram (Figure 3.1-1) of this system shows the display-memory bus of the QPDM connected via drivers and a small amount of interface logic to the four planes of display memory. The serial-data outputs of the dual ported video memory connect to one Video Data Assembly FIFO (VDAF) Am8172 for each plane. This shifter provides an ECL output for the HIGH video speed. The control signals for the VDAFs are provided by the QPDM.

All the circuitry for the display memory connection is synchronous to SYSCLK. This main clock signal for the QPDM runs the internal micro-engine and determines all display memory timing. The signals VSTB and DSTB, synchronous to the SYSCLK signal, also strobe serial data from the dual-ported video memory into the VDAF's internal FIFO. The VIDCLK signal, derived from the DOTCLK and *asynchronous* to the SYSCLK, is described in Chapter 4. The QPDM produces the video synchronization signals HSYNC, VSYNC, and BLANK with this VIDCLK signal and also reads data out of the video side of the FIFO of the VDAF.

3.1.2 What You Can Do With The System

This 2K by 2K display memory is a common size for graphic terminals, personal computers, high-performance desktop publishing systems, and CAE/CAD workstations.

This size allows easily for a 1280 pixel by 1024 pixel screen and leaves enough room in the display memory to scroll the screen vertically and pan it horizontally. Furthermore there is room to store one or more images of real windows that can be displayed alternately over any rectangular area of the screen. Figure 3.1-2 shows the size and the use of the 2K by 2K display memory. The user may elect to lay out display memory differently.

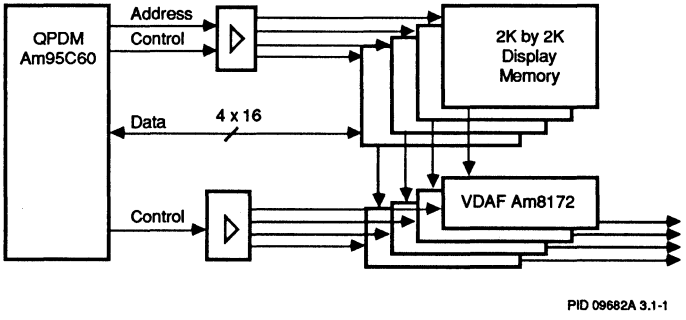


Figure 3.1-1 Block Diagram of a 4-Plane 2K by 2K Display Memory System

3.1.3 Circuit Diagram

Figures 3.1-4a and 3.1-4b show the complete circuit diagram for the connection of the QPDM to four planes of display memory and four VDAFs. Figure 3.1-4a starts with the QPDM and the buffer and logic section. Figure 3.1-4b shows the memory array and the VDAF section for planes 0. The memory for planes 1 through 3 is organized similarly. The total memory array consists of 4 banks. Each bank is organized as 64K by 16 bit per plane. The total amount of memory is:

4 planes with 2K by 2K pixels =
4 planes by 4 banks with each 64K by 16 bit = 16 MBit

Figure 3.1-3 shows the logical to physical address mapping for a 2K wide display memory. It illustrates that four banks are required for a 2K deep memory. The bank boundaries are defined by the Y-address bit Y_{10} and Y_9 which are output via the address pins $ADDR_{9-8}$, respectively. The multiplexed address bits $ADDR_{7-0}$ address all display memory words within one bank. Address outputs $ADDR_{11-10}$ are not utilized in this application. See Chapter 12 of the technical manual for other memory organizations.

The QPDM and the Drivers

Figure 3.1-4a shows the QPDM and the buffers to the display memory. All display memory bus signals except for the 64 bit wide data bus and some signals to and from the VDAFs are buffered. Many of the signals just pass through a buffer from the QPDM to the display memory array. Some others pass through logic for decoding and then get distributed within the memory array.

The eight least significant bits of the address bus $ADDR_{7-0}$ of the QPDM are fed into a buffer Am29827A. The outputs of this driver $ADDR^*_{7-0}$ are connected to the multiplexed address inputs A_{7-0} of every single 64K by 4 bit video memory chip (there are 64 chips). For consistent nomenclature all names of *amplified* signals into the memory array have a "*" at the end. Although not explicitly shown in Figure 3.1-4a, all those amplified signals have a serial resistor of 25 Ω in the signal line to prevent undershoot.

Figure 3.1-4a lists on the bottom right side of the Am29827A block for the address lines some additional information about these signal lines. The information {256 pF, 3-16 ns, to 64} indicates that the $ADDR^*_{7-0}$ lines go to 64 chips altogether, that these chips represent 256 pF (64 chips with 4 pF each) input capacitance, and that the best and worst case delay for the Am29827A for this capacitance is 3 ns and 16 ns respectively.

$ADDR_{9-8}$ are employed to select one of the four banks. The decoding logic is described later. The address lines $ADDR_{11-10}$ are not used in this example. In another application these address lines would help to decode bank addresses for a larger display memory array, for example 4K by 4K.

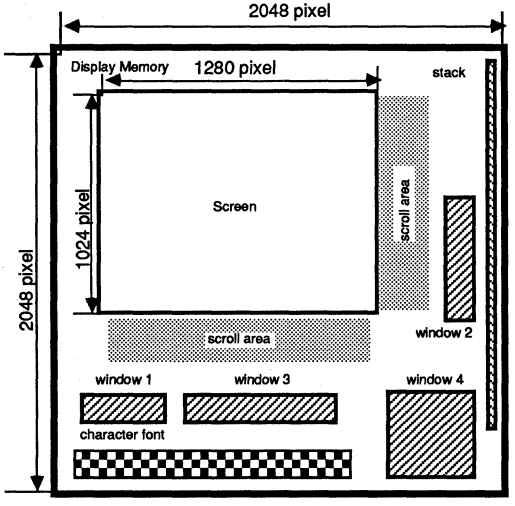


Figure 3.1-2 2048 by 2048 Display Memory with Several Real Window Locations and Allocation for One or Several Character Fonts

To minimize the CAS delay time the CAS signal is amplified by eight buffers of an Am29827A driver. The amplified CAS*(7:0) signal is distributed to all 64 chips of the display memory array, thus each $CAS*_{7-0}$ output drives 8 memory chips.

The XF/G signal is buffered by an Am29827A driver. The amplified signal XF/G* is transmitted, just like the amplified address lines $ADDR*_{7-0}$ and the amplified and distributed CAS*(7:0) signals to all 64 memory chips. The unbuffered XF/G signal is also used in subsequently discussed logic.

The four Write Enable signals WE(3:0) for the four display memory planes are buffered by an Am29827A buffer. The buffered signals WE*(3:0) are distributed to 16 chips each, i.e. to all memory chips in one plane.

The addressing of the four banks within the display memory is accomplished by generating four RAS signals. These four RAS signals are latched with the falling edge of RAS by four negative edge triggered flip-flops F114. A D-speed PAL AmPAL18P8 decodes the two address lines $ADDR_{9-8}$ and provides the J-K-inputs for the F114 flip-flops. The RASbk*(3:0) signals must be latched in order to keep them stable during the complete memory cycle. The individual RASbk*(3:0) signals stay LOW for as long as RAS is LOW. The rising edge of RASbk*(3:0) is generated by clearing the flip-flops with the asynchronous reset function at the end of RAs from the QPDM.

During a display memory read, write, or transfer cycle, only one bank of the memory is involved, thus only one of the four possible RASbk*(3:0) signals is activated, and only one of the four banks of memories comes out of LOW power mode and switches to normal power consumption. This feature allows the display memory to be operated with minimal power consumption. During a display memory refresh cycle, the RASbk*(3:0) of all four banks are activated simultaneously to refresh the complete memory with the smallest number of refresh cycles.

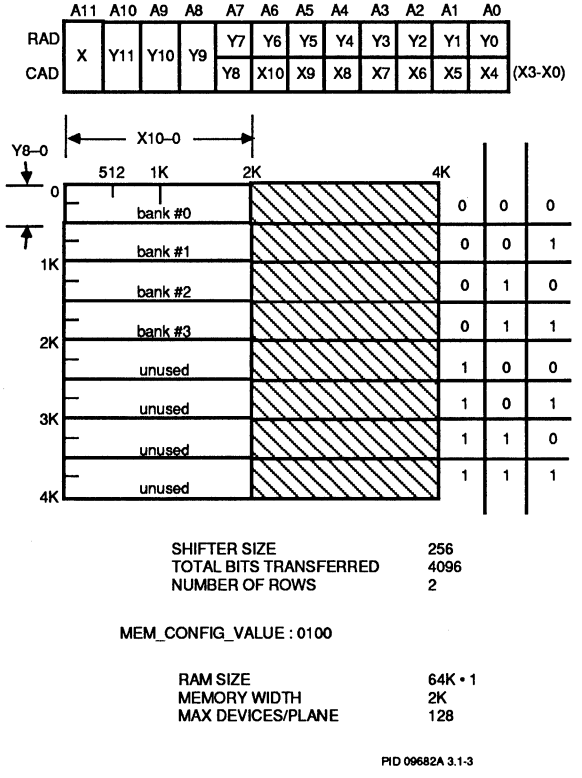


Figure 3.1-3 Logical to Physical Address Mapping for 64K by 4-bit Memory Chips for a Display Memory Width of 2K Pixels

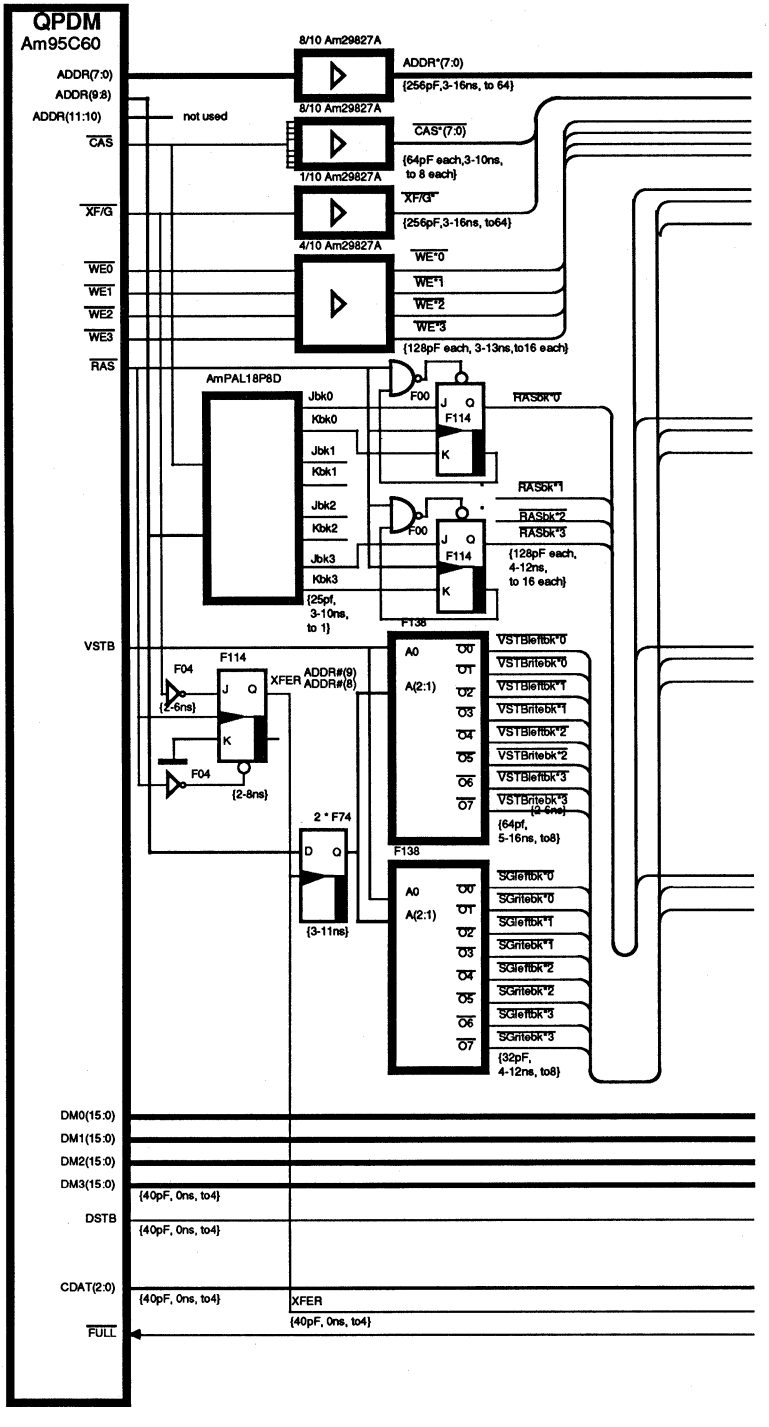


Figure 3.1-4a Circuit Diagram of the QPDM, the Buffer, and the Interface Logic

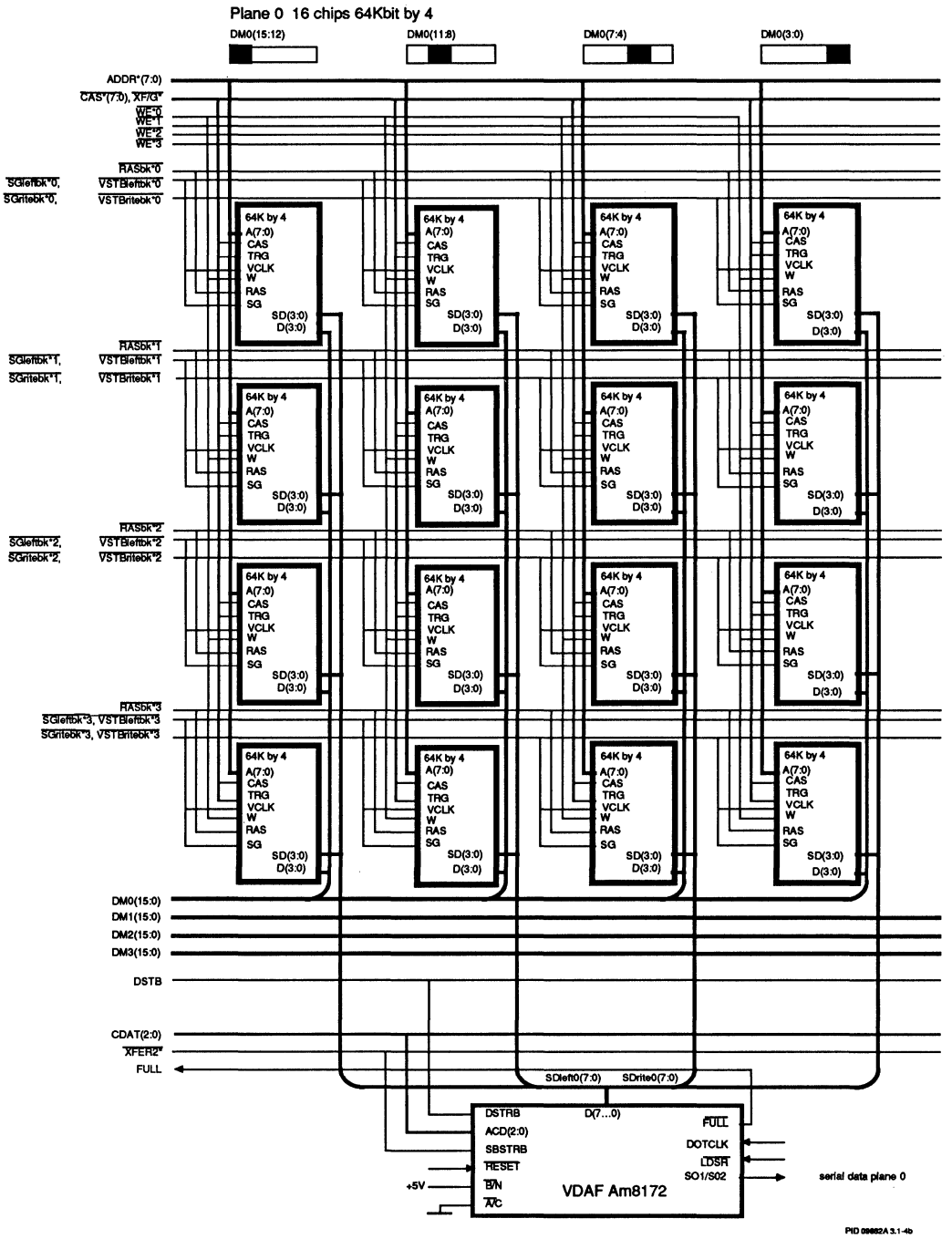


Figure 3.1-4b Circuit Diagram of Memory Plane #0 and VDAF #0

Table 3.1-1 Generation of J-K Inputs by AmPAL18P8D for RASbk*(3:0)

CAS	Jbk_{3-0} of the bank selected by $ADDR_{9-8}$	Kbk_{3-0}	Jbk_{3-0} of the bank *not* selected by $ADDR_{9-8}$	Kbk_{3-0}	number of active RASbk*(3:0)
1	L	H	H	L	one
0	L	H	L	H	all

The logic in the PAL uses the fact that the CAS signal is already LOW before the falling edge of RAS during a dynamic memory refresh cycle whereas it is still HIGH during all other display memory cycles. Table 3.1-1 shows the relationship between the $ADDR_{9-8}$, RAS, and CAS inputs and the Jbk_{3-0} and Kbk_{3-0} outputs of the PAL. Each of the RASbk*(3-0) signals is distributed to 4 chips in each of the four planes, that is to all chips within one bank.

The VSTB signal provides timing for the two F138 logic blocks to generate the signals VSTBsidebk*(3:0) and SGsidebk*(3:0), respectively. In these signal names the term *side* stands for either *right* or *left*, indicating a VSTB- or SG- signal to the right or the left half of a 16-bit display memory word, respectively. The term *bk* means bank.

The two F138 1-of-8 decoders for the generation of the signals VSTBsidebk*(3:0) and SGsidebk*(3:0) require a latched bank-select address for the time in between two transfer cycles. The bank address $ADDR_{9-8}$ is stored in two flip-flops F74. These flip-flops get strobed by the XFER pulse. This active HIGH pulse is generated by a negative edge triggered flip-flop F114. The XFER signal becomes active with the falling edge of RAS whenever the XF/G signal of the QPDM was already LOW during the RAS-transition. The flip-flop F114 gets asynchronously reset when the XF/G pin of the QPDM is no longer active, thus when it goes HIGH. If the XF/G signal is HIGH during the falling edge of RAS, no XFER pulse is generated.

Each VSTBsidebk*(3:0) signal is distributed to eight video memory chips, i.e. to all chips in one bank in all four planes that contain either the left byte or the right byte. These signals clock data out of the serial video memory shifter. In this application example we chose to activate the QPDM's VSTB signal to the minimum number of chips possible (to only one bank) at any given time in order to reduce the power requirements for the video memory chips. The logic in the F138 1-of-8 decoder provides the correct phase shift for this serial clock signal.

When activated by register programming, the additional VSTB pulse is generated by the QPDM during a transfer cycle. Thus at the end of a transfer cycle a complete 16-bit wide word is available on the output of the serial port of four video memory chips. The subsequently discussed SGsidebk*(3:0) signals will then output enable the left and the right byte in sequence after the transfer cycle. Depending on whether the first DSTB pulse (see below) occurs while VSTB is LOW or HIGH, one or both bytes, respectively, of this first word are strobed into the VDAF.

The VSTBritebk*(3:0) signal is at all times in phase with the VSTB output signal from the QPDM, whereas the VSTBleftbk*(3:0) signal is inverted to VSTBritebk*(3:0).

The following Table 3.1-2 shows the relationship between the QPDM's VSTB output, the latched $ADDR_{9-8}$ signals and the VSTBsidebk*(3:0) output signals of the PAL.

The video memory chips have an output enable signal for the serial port. This signal is active in exactly one bank during the entire time video data is shifted out of the serial port. The active bank during this time is kept in the CF74s.

Table 3.1-2 VSTB Outputs and Address Bits 8 and 9

ADDR #9	ADDR #8	VSTB	VSTB-leftbk0	VSTB-ritebk0	VSTB-leftbk1	VSTB-ritebk1	VSTB-leftbk2	VSTB-ritebk2	VSTB-leftbk3	VSTB-ritebk3
0	0	0	H	L	H	H	H	H	H	H
0	0	1	L	H	H	H	H	H	H	H
0	1	0	H	H	H	L	H	H	H	H
0	1	1	H	H	L	H	H	H	H	H
1	0	0	H	H	H	H	H	L	H	H
1	0	1	H	H	H	H	L	H	H	H
1	1	0	H	H	H	H	H	H	H	L
1	1	1	H	H	H	H	H	H	L	H

Furthermore, the output enable signal on the video memory chips must perform a multiplexing function. The VDAF has only an 8-bit-wide input whereas the memory outputs a 16-bit wide word per plane. Thus, during the time the VDAF latches in the left half of a display memory word, only the two "left" video memory chips must be enabled, and during the time the VDAF strobes in the right half of the display memory word, the other two video memory chips of one bank in each plane must be enabled. The task of enabling and disabling the serial outputs is accomplished by the signals SGleftbk*(3:0) and SGritebk*(3:0) for the left and right sides, respectively.

These output enable signals for the serial port of the video memory chips are generated by an F138 chip. This F138 is always enabled, thus at any given time there is one active signal to the D_{7-0} inputs of the VDAF. The VSTB signals controls the left and right side of the display memory word, the latched $ADDR_{9-8}$ lines control the addressing of the bank. The following Table 3.1-3 shows the generation of the output enable signals.

The XFER pulse is also connected to the SBSTRB input of the VDAFs. Here this signal strobes in the position of the first valid bit within the first byte of video data in the scan line. This position is presented to the VDAFs during the rising edge of the SBSTRB input on the ACD_{2-0} lines.

The four 16 bit wide data busses $DM_{3-015-0}$ are not buffered and connect directly to the video memory array. Since there are four banks in the system, each data pin of the QPDM is connected to four common data input/output pins of the video memory chips. The bank select encoding ensures that at any given time only one bank of memory chips interchanges data with the QPDM.

The DSTB output of the QPDM is not buffered and supplies the clock to strobe data into the VDAF. The DSTB signal is distributed only to DSTRB inputs of the four VDAF chips.

The $CDAT_{2-0}$ outputs of the QPDM are not buffered and supply control data for the VDAF. During a transfer cycle the $CDAT_{2-0}$ lines carry the information of the first valid bit position within the first byte after a transfer cycle. With every DSTB cycle the $CDAT_{2-0}$ lines inform the VDAF about the number of valid bits within the current byte from the video memories. The CDAT(2:0) signals are distributed only to the ACD_{2-0} inputs of the four VDAF chips.

The FULL signal is send by the VDAF and is an input to the QPDM. This input indicates when the VDAF has its FIFO nearly full and cannot accept any more data. Only the output of one VDAF – in this case from plane 0 – is connected to the QPDM. Since all VDAFs receive the same control signal and therefore the same number of data bytes, the status of the FIFO is the same for all planes. Thus, when plane 0 indicates that its FIFO is full, the FIFOs of all planes are full, and the QPDM will not strobe data into the VDAF's.

The Memory Array

Figure 3.1-4b shows the memory arrays for plane 0. The memory arrays for plane 1 through 3 are not shown; their connections are similar to plane 0. The signals on the right side of Figure 3.1-4b are connections to the other three display memory planes. Each plane consists of 16 memory chips. The top four chips in the figure are bank #0, the next row is bank #1, and so on. On the top of Figure 3.1-4b the position of each chip within the display memory word is indicated. The left column of four chips supplies the four leftmost bits within a display memory word. For plane 0 the data lines of these memory chips are connected to the display memory bus data lines DM_{015-12} with DM_{015} connecting to the leftmost bit within each 16-bit word. The column to the right of the left column connects to DM_{011-8}, and so on.

The $ADDR*_{7-0}$, CAS*, and XF/G* lines are distributed to all chips in all planes. The WE*0 goes to all chips in plane #0, the WE*1 signal goes to all chips in plane #1, and so on. The RASbk*0 goes to bank #0 in all four planes, the RASbk*1 goes to bank #1 in all four planes, and so on. The VSTBleftbk*0 and SGleftbk*0 signals connect to the two left columns of chips in bank #0 in all four planes, the

Table 3.1-3 Truth Table for SGsidebk*(3:0) Generation

Addr 9	Addr 8	VSTB	SG leftbk0	SG ritebk0	SG leftbk1	SG ritebk1	SG leftbk2	SG ritebk2	SG leftbk3	SG ritebk3
0	0	0	H	L	H	H	H	H	H	H
0	0	1	L	H	H	H	H	H	H	H
0	1	0	H	H	H	L	H	H	H	H
0	1	1	H	H	L	H	H	H	H	H
1	0	0	H	H	H	H	H	L	H	H
1	0	1	H	H	H	H	L	H	H	H
1	1	0	H	H	H	H	H	H	H	L
1	1	1	H	H	H	H	H	H	L	H

VSTBritebk*0 and SGritebk*0 go to the two right columns of chips in bank #0 in all four planes, and so on.

The data pins of all 4 chips within one bank of each plane form a 16-bit-wide data bus. The data from plane #0 are connected to the $DM0_{15-0}$ lines of the QPDM, the data from plane #1 are connected to the DM_{115-0} lines of the QPDM, and so on. The serial data outputs of both the two left chips SDleft(3:0)(7:0) and the two right chips SDrite(3:0)(7:0) within one bank of each plane form an 8-bit-wide data bus. This 8-bit-wide data bus from plane #0 is connected to the D_{7-0} inputs of the VDAF for plane #0, the 8-bit-wide data bus from plane #1 is connected to the D_{7-0} inputs of the VDAF for plane #1, and so on.

The VDAF

Figure 3.1-4b also shows the VDAF serializer for plane 0. The 8-bit-wide data input to the shifter is obtained from the serial data output of the video memory array. At any time, only the left side or the right side of a video memory plane supplies data to the VDAF. DSTB, $CDAT_{2-0}$, and the XFER pulse are supplied in parallel to the DSTRB, ACD_{2-0}, and SBSTRB inputs of the VDAFs in all four planes. The RESET signal is supplied to all VDAFs to initialize the internal logic. The B/N pin the VDAF is set to accept byte wide data rather than nibble wide data. The A/C input specifies that the VDAF interprets the ACD_{0-2} input as the number of valid bits rather than the bit position of the first unusable bit within a byte from the memory array.

The DOTCLK and LDSR signals are also supplied to the VDAF. The generation and distribution of these signals, however, is analyzed and described in Chapter 4 of this manual. The SO_1 pin outputs the HIGH speed serial data stream to the color palette or directly to the monitor. The SO_2 output pin is not used.

3.1.4 The Timing Analysis

The timing analysis considers the propagation delay of each signal. This insures that the suggested system will work under worst case conditions. To drive the highly capacitive load of a memory array, it is especially important to use drivers that can drive high capacitances. The propagation delay for the Am29800A family is specified for an unloaded output. The AMD Bus Interface Products Handbook (publication number #07175B) specifies some additional guidelines. The switching speed increases by 0.5 ns for each additional 50 pF of load and by 0.3 ns for each additional output switching at the same time. An output whose unloaded switching time, for example, is specified to be 9 ns, will switch in reality in 14.2 ns with a 250 pF load when all 10 outputs in the package switch at the same time. (2.5 ns slower for added load plus 2.7 ns slower for simultaneous switching.) Since the exact load capacitance of the circuit varies with every signal, some conservative interpolations have been performed to calculate the actual propagation delay for the capacitance in the circuit.

The QPDM has a maximum SYSCLK frequency of 20 MHz. The SYSCLK not only determines the display memory timing, but also clocks the internal microengine. The drawing performance is directly proportional to the SYSCLK speed. This design implements the full 20 MHz (50 ns) SYSCLK speed for highest performance. If, in a similar application one cannot fulfill all timing parameters of the display memory interface, the speed of the SYSCLK must be decreased until all parameters are met.

Timing Parameters

	Not to Exceed	
	Set-up Time	**Hold Time**
Row Address	4 ns	26 ns
Column Address	0 ns	73 ns
Masked Write Strobe wrt RAS	2 ns	51 ns
Write Mask Data wrt RAS	6 ns	48 ns
Write Command wrt CAS	3 ns	50 ns
Write Data wrt CAS	5 ns	50 ns
XF/G wrt RAS	0 ns	129 ns

	Not to Exceed
Row Access Time	128 ns
Column Access Time	50 ns
Output Enable Time on Random Port	74 ns
Clock to Output Time on Serial Port	69 ns
Output Enable Time on Serial Port	23 ns

To run the display memory interface with 20 MHz a relatively fast video memory chip must be selected. For this application a "-10" (100 ns) 64K by 4-bit video-memory chip has been chosen. In particular this memory chip must fulfill the following timing parameters.

Most of these times are easily fulfilled. The only two tight parameters are the column access time and the serial output enable time.

Display Memory Read Cycle

Figure 3.1-5 shows the timing for a Display Memory Read Cycle.

For a 20 MHz system clock the QPDM outputs its bank select address $ADDR_{9-8}$ and row address $ADDR_{7-0}$ at least 15 ns (parameter #30 t_S of the QPDM) prior to the falling edge of RAS and holds the row address valid for at least 35 ns (parameter #31 t_H of the QPDM). The address lines are amplified in the Am29827A with a best and worst case delay for a 256 pF load from 3 ns to 16 ns. The resulting signal is ADDR*(7:0). The bank select address $ADDR_{9-8}$ is decoded in a D-speed AmPAL18P8 with a best and worst case propagation delay of 3 ns to 10 ns. When the falling edge of RAS arrives at the F114 flip-flop, the Jbk(3:0) and Kbk(3:0) bank select signals have been steady for at least 5 ns (parameter #30 t_S of the QPDM minus maximum propagation delay through the PAL), and will remain steady for the rest of the read cycle. Thus the 5 ns set-up and 0 ns hold time (F114 parameter) for the F114 flip-flop is fulfilled.

The propagation delay time to generate the RASbk*(3:0) signals from the bank select signals Jbk(3:0) and Kbk(3:0) is 4 ns to 12 ns. Figure 3.1-5 shows that this leaves a set-up and hold time for the row address on the video memory chips of 4 ns and 26 ns respectively, which is easily fulfilled with most video memory chips.

The RASbk*(3:0) signals are negated with the rising edge of RAS. As soon as the inverted RAS signal goes LOW, the direct-set inputs of the F114 flip-flops that are asserted go LOW and de-assert the RASbk*(3:0) signals with a total propagation delay of 6 ns to 18 ns (addition of propagation delays of NAND-gate F00 and flip-flop F114). The NAND-gate logic in this path provides the necessary recovery time for the F114 flip-flop.

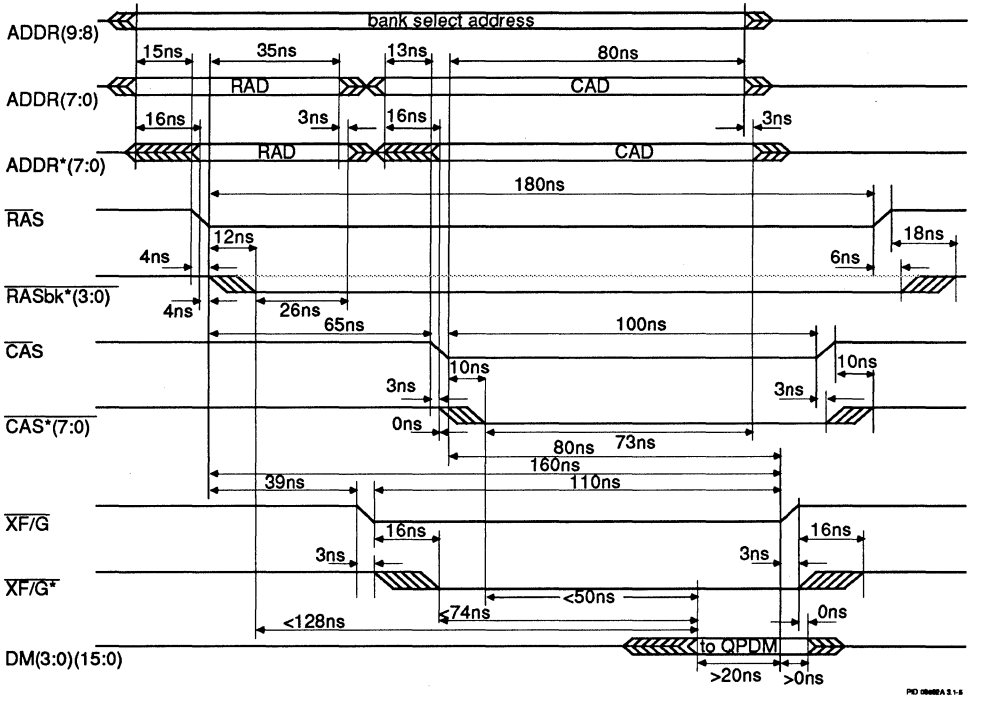


Figure 3.1-5 Display Memory Read Cycle

The QPDM outputs its column address $ADDR_{7-0}$ at least 13 ns (parameter #37 t_S of the QPDM) prior to the falling edge of CAS and holds the column address and bank select address valid for at least 80 ns (parameter #38 t_H of the QPDM). The amplified signal is ADDR*(7:0). The CAS signal is fed into eight drivers Am29827A, each of which supplies eight video memory chips. The propagation delay to generate the four CAS* signals is 3 ns to 10 ns (interpolation of Am29827A timing parameter). Figure 3.1-5 shows that this leaves a set-up and hold time for the column address on the video memory chips of 0 ns and 73 ns respectively, which fulfill the requirement of video memory chips. The normal requirement for video memories is 0 ns and 20 ns for the set-up and hold time, respectively.

The XF/G signals goes active 39 ns (parameter #42 t_{PD} of the QPDM) or later after the falling edge of RAS. It stays valid for at least 110 ns (parameter #44 t_W of the QPDM). This signal passes through the driver within 3 ns to 16 ns (interpolation of the Am29827A timing specification). The video memory chips will enable their outputs soon after they see XF/G LOW.

The XF/G signal will stay LOW for at least 80 ns (parameter #41 t_{PD} of the QPDM) after CAS has gone LOW and 160 ns (parameter #32 t_{PD} of the QPDM) after RAS has gone LOW. The data coming from the memory array into the QPDM must be valid 20 ns (parameter #45 t_S of the QPDM) prior and 0 ns (parameter #46 t_H of the QPDM) after the rising edge of XF/G on the QPDM. Figure 3.1-5 shows that, in order to fulfill these requirements, the video memory chips must have an access time of less than 128 ns after the falling edge of RAS, less than 50 ns after the falling edge of CAS, and less than 74 ns after XF/G has been asserted. The speed of 100 ns video memory just fulfills these parameters, with the access time from CAS being the most critical parameter.

The signals WE(3:0), VSTB, DSTB, and CDAT(2:0) are not employed during a display memory read cycle.

Display Memory Write Cycle

The timing for a Display Memory Write Cycle is shown in Figure 3.1-6.

The timing to supply addresses and address strobe signals to the video memory is identical to the read cycle. This includes the parameters #30 and #37 t_S and the parameters #31 and #38 t_H of the QPDM. Thus all set-up and hold time requirements for the row, column, and bank-select addresses are fulfilled. The XF/G signal is inactive during the complete write cycle. In the write cycle the delay from RAS to CAS is 90 ns (parameter #56 t_{PD} of the QPDM) rather than 65 ns (parameter #36 t_{PD} of the

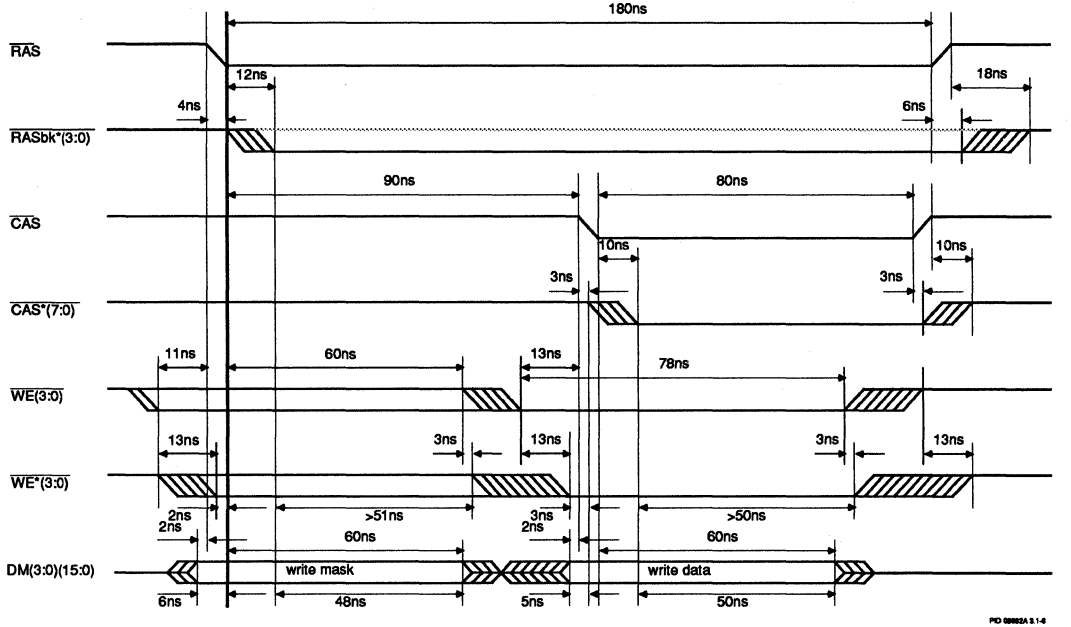


Figure 3.1-6 Display Memory Write Cycle

QPDM) as in the read cycle. The CAS signal stays active for 80 ns (parameter #57 t_w of the QPDM).

During the first half of the write cycle the write mask may be strobed into the video memory chips. A write mask, i.e. the data on the DM(3:0)(15:0) bus, is loaded into the video memory chips whenever its write input is LOW during the falling edge of RAS. If WE is HIGH when RAS falls, all four bits will be written.

The WE(3:0) signal is stable 11 ns (parameter #59 t_S of the QPDM) before the falling edge of RAS and remains stable for approximately 60 ns (interpolation of the QPDM spec) after the falling edge of RAS. The WE(3:0) signals run through an Am29827A driver with a best and worst case propagation delay of 3 ns to 13 ns, respectively (interpolation of Am29827A timing specification). Figure 3.1-6 shows that this leaves a set-up and hold time for the write enable pulse W on the video memory chip of 2 ns and more than 51 ns respectively, which is easily fulfilled by video memory chips.

The data is supplied to the memory chips without passing through any driver. The QPDM provides valid output data at least 2 ns (parameter #62 t_S of the QPDM) prior and 60 ns (parameter #63 t_H of the QPDM) after the falling edge of RAS. Figure 3.1-6 shows that this leaves a set-up and hold time for the write mask data on the video memory chips of 6 ns and 48 ns respectively, which is easily fulfilled by video memory chips.

During the second half of the write cycle the actual data get strobed into the memory chips. The QPDM asserts the write command at least 13 ns (parameter #60 t_S of the QPDM) prior to the falling edge of CAS and has a valid WE(3:0) pulse width of at least 78 ns (parameter #61 t_w of the QPDM). The propagation delay from CAS to CAS*(7:0) is 3 ns to 10 ns. The video memory chips see a write command set-up and hold time of 3 ns and more than 50 ns, respectively, which satisfies video memory chips.

The actual data is output at least 2 ns (parameter #64 t_S of the QPDM) prior to the falling edge of CAS and stay valid at least 60 ns (parameter #65 t_H of the QPDM) after this falling edge. Figure 3.1-6 shows that this leaves a set-up and hold time for the data on the video memory chip of 5 ns and 50 ns respectively, which again is satisfied by video memory chips.

Display Memory Transfer Cycle

The timing of the display memory transfer cycle involves both the display memory and the VDAFs.

The strobing of the row and shifter start address (column address in the read cycle) into the video memory chips works with the same timing as in the read cycle. Thus all set-up and hold times for the row, column, and bank-select address are fulfilled.

During a transfer cycle, however, the XF/G signal is valid before the falling edge of RAS. The QPDM outputs the XF/G signal 12 ns (parameter #49 t_S of the QPDM) before the falling edge of RAS. It stays active 140 ns (parameter #69 t_{PD} of the QPDM) after the falling edge of RAS and 55 ns (parameter #67 t_{PD} of the QPDM) after the falling edge of CAS. The XF/G signal becomes inactive at least 39 ns (parameter #70 t_H of the QPDM) before the rising edge of RAS and at least 40 ns (parameter #68 t_H of the QPDM) before the rising edge of CAS. The XF/G signal passes through an Am29827A buffer, and XF/G* is available 3 ns to 16 ns later on the video memory chips. Figure 3.1-7 shows that this leaves (with the propagation delay from the RAS to the RASbk*(3:0) signal) a set-up and hold time for the XF/G signal with respect to the RAS signal on the video memory chips of 0 ns and 131 ns, respectively. This parameter, again, is fulfilled by video memory chips.

The WE(3:0) signal of the QPDM is HIGH during the complete transfer cycle. This indicates a transfer direction from the memory array to the shifter. Since this is always the case, so-called "dummy" transfer cycles are never required.

The falling edge of the RAS signal clocks the flip-flop F114 that generates the transfer signal XFER. The inverter F04 delays the XF/G signal by 2 ns to 6 ns (F04 data). This leaves a set-up and hold time for the XF/G signal on the flip-flops F114 of 6 ns and more than 100 ns respectively, which is more than the required 5 ns and 0 ns (F114 data). The propagation delay within the flip-flop is 2 ns to 8 ns (F114 data). The XFER pulse is deactivated 2 ns to 8 ns after the rising edge of RAS.

The rising edge of the XFER pulse clocks the F74 D-register to store the bank-select address for the time between two transfer cycles. This address is valid 15 ns (parameter #30 t_S of the QPDM) before and more than 130 ns after the falling edge of RAS. This leaves a set-up and hold time with respect to the address for the F74 of 17 ns and more than 118 ns, respectively, which is more than the required 3 ns and 1 ns (F74 data). The propagation delay for the ADDR(9:8) signal through the flip-flop is 3 ns to 11 ns (F74 data).

During a transfer cycle start offset control information is strobed into the VDAFs. These control data is presented to the VDAFs on the ACD_{2-0} inputs and are strobed into the VDAFs by their $\overline{SBSTRB}$ inputs. The control data is transferred without buffering from the QPDM $CDAT_{2-0}$ outputs to the four VDAFs. The strobe signal is the XFER signal generated by the flip-flop F114. The data on

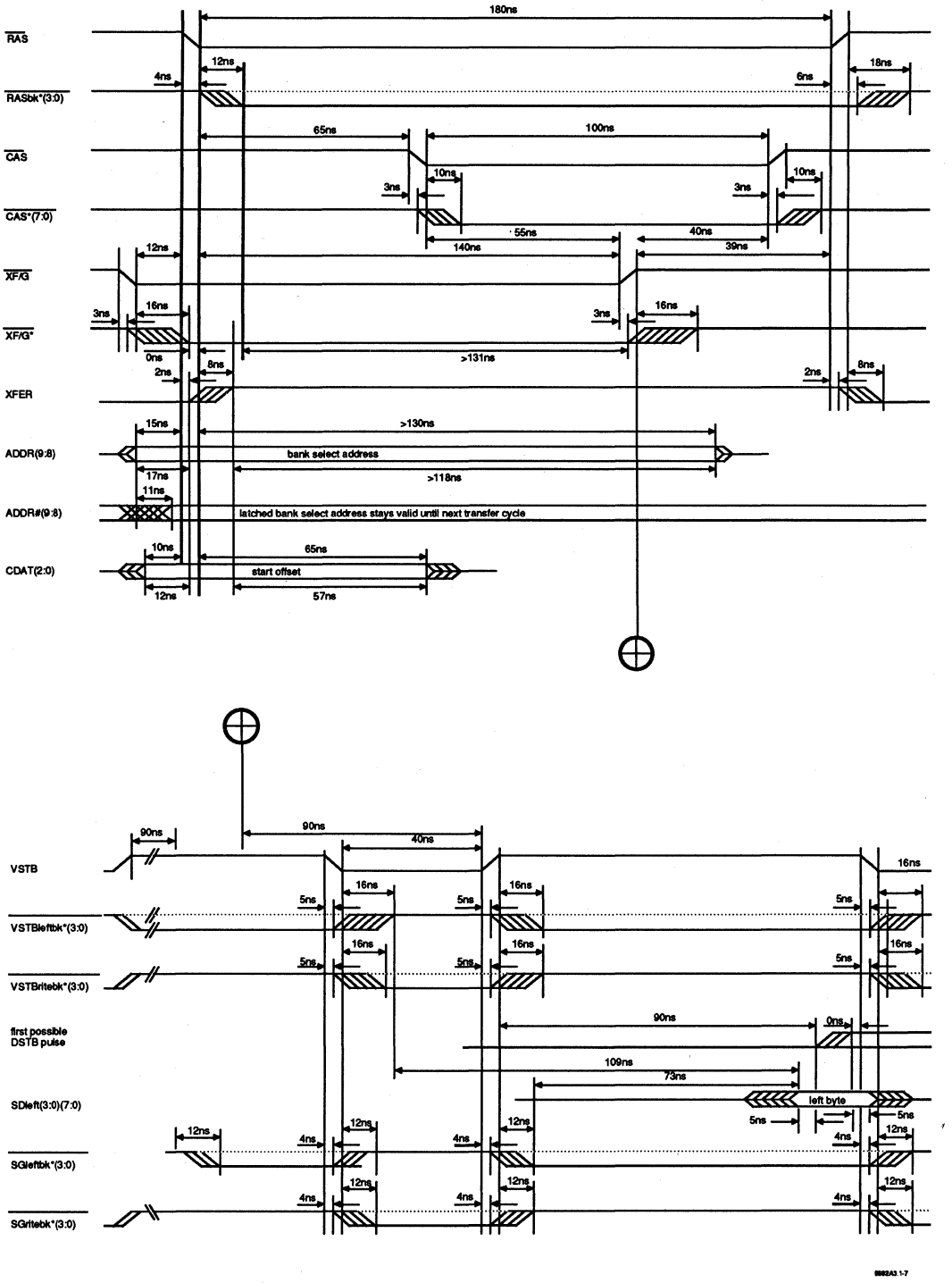


Figure 3.1-7 Display Memory Transfer Cycle Timing

$CDAT_{2-0}$ are valid at least 10 ns prior to and 65 ns after the falling edge of RAS (parameters #50 t_S and #51 t_H of the QPDM). In a multi QPDM system, each QPDM delivers this control information to the four connected VDAFs. Figure 3.1-7 shows that the ACD_{2-0} data on the VDAF have a set-up and hold time with respect to XFER of 12 ns and 57 ns, which fulfills the 10 ns and 15 ns (parameter #9 t_S and #10 t_H of the VDAF). The set-up and hold time for the strobing of the valid bit count data on the the $CDAT_{2-0}$ lines during the first possible DSTB pulse follows the timing of the general DSTB pulse and is discussed under the video clock cycle timing section for the VDAF.

Figure 3.1-7 shows that the VSTB signal of the QPDM is in a HIGH state at the beginning and the end of the transfer cycle. A VSTB LOW pulse can be activated by register programming. The QPDM's VSTB signal is HIGH 90 ns (parameter #52 t_S of the QPDM) before the falling edge of RAS and stays HIGH until after the first possible rising DSTB pulse. The 40 ns VSTB LOW pulse (parameter #71 t_W of the QPDM) ends 90 ns (parameter #73 t_S of the QPDM) after XF/G has become inactive. Furthermore, this pulse ends at least 90 ns (parameter #72 t_{PD} of the QPDM) before the first possible DSTB pulse, the pulse that strobes data into the VDAF.

The VSTBsidebk*(3:0) signals are produced by an F138 1-of-8 decoder with a best and worst case propagation delay of 5 to 16 ns. The VSTBritebk*(3:0) signals follow the waveform, and the phase of the VSTB signal from the QPDM, the VSTBleftbk*(3:0) is inverted. This is necessary to give both the left half and the right half of the word sufficient clock to output time inside the video memory chip and time to travel from the video memory to the VDAF.

The $ADDR_{9-8}$ is also fed into the F138 1-of-8 decoder. This decoder outputs the SGsidebk*(3:0) signals with a propagation delay of 4 ns to 12 ns (interpolation from F138 data). At the end of the transfer cycle, before the first possible DSTB pulse, the SGleftbk*(3:0) signal is asserted and output enables the left byte of the selected bank. If the first possible rising DSTB edge occurs the left byte is strobed into the VDAFs. Next, the 1-of-8 decoder selects the right byte by enabling SGritebk*(3:0) of the selected bank when the QPDM's VSTB signal goes LOW after the first possible DSTB pulse, that is at the end of the transfer cycle. SGritebk*(3:0) of the enabled bank follows the waveform of the QPDM's VSTB signal, the SGleftbk*(3:0) signal of that bank is inverted to its SGritebk*(3:0) counterpart.

Valid data from the serial output of the video memory chips must be present at the VDAFs at least 5 ns (parameter #5 t_S of the VDAF) before the first possible DSTB pulse. This is accomplished by fulfilling two access time parameters of the video memory. First, after the rising edge of VSTBleftbk*(3:0) the first word of new background or window data is transferred to the video memories serial output pin. This leaves a clock to output time of 109 ns (40 ns for the VSTB pulse width plus 90 ns to the first possible DSTB pulse minus 16 ns for the VSTBsidebk*(3:0) propagation delay minus 5 ns for the data set-up time for the VDAF) for the video memories, which is fulfilled by any video memory chip. Second, after asserting SGleftbk*(3:0) and enabling the serial output driver of the video memory chips the data travel from the video memory chips to the VDAFs. This must be accomplished in 73 ns, which again is fulfilled by all video memory chips.

Display Memory Refresh Cycle

The timing for the refresh cycle is fairly simple. The QPDM outputs a CAS before RAS refresh cycle. The set-up and hold times for the refresh address with respect to the falling edge of RAS are identical to the row address set-up and hold times during a read cycle and therefore are fulfilled.

The QPDM activates CAS 37 ns (parameter #47 t_S of the QPDM) before the falling edge of RAS, and CAS stays active for at least 185 ns (parameter #48 t_H of the QPDM) after the falling edge of RAS. The AmPAL18P8D generates the appropriate J-K-inputs for the flip-flops F114. See Table 3.1-1 for a truth table of the PAL function. Since CAS is LOW during the falling edge of RAS *all four* flip-flops are set by the J-K-inputs to activate their $RASbk*_{3-0}$ outputs. Figure 3.1-8 shows that the video memory chips see a set-up and hold time of 31 ns and 176 ns of the CAS*(7:0) signal with respect to the falling edge of RASbk*(3:0).

Video Clock Cycle for VDAF

With every rising edge of DSTB (DSTRB input on the VDAF) a new byte of video data and a new 3-bit control word on the ACD_{2-0} inputs are strobed into the VDAF. The verification of the set-up and hold times for the control data and the video data is is looked at independently.

First, the $CDAT_{2-0}$ data from the QPDM are valid at least 8 ns prior and 15 ns after the rising edge of DSTB (parameter #81 t_S and #82 t_H of the QPDM). The set-up and hold time requirement of the ACD_{2-0} inputs with respect to the DSTRB input is 5 ns and 10 ns respectively (parameter #7 t_S and #8 t_H of the VDAF), thus the set-up and hold time requirements are fulfilled.

Second, video data is strobed into the VDAFs by the DSTB signal. The correct video data is selected by a combination of phase shifted clocking with the

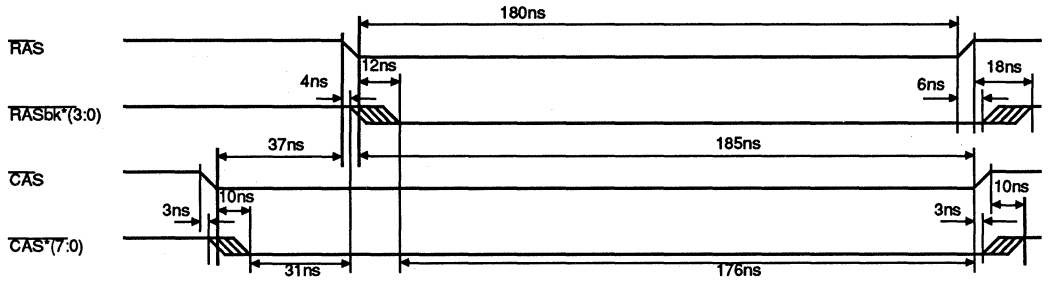


Figure 3.1-8 Display Memory Refresh Cycle

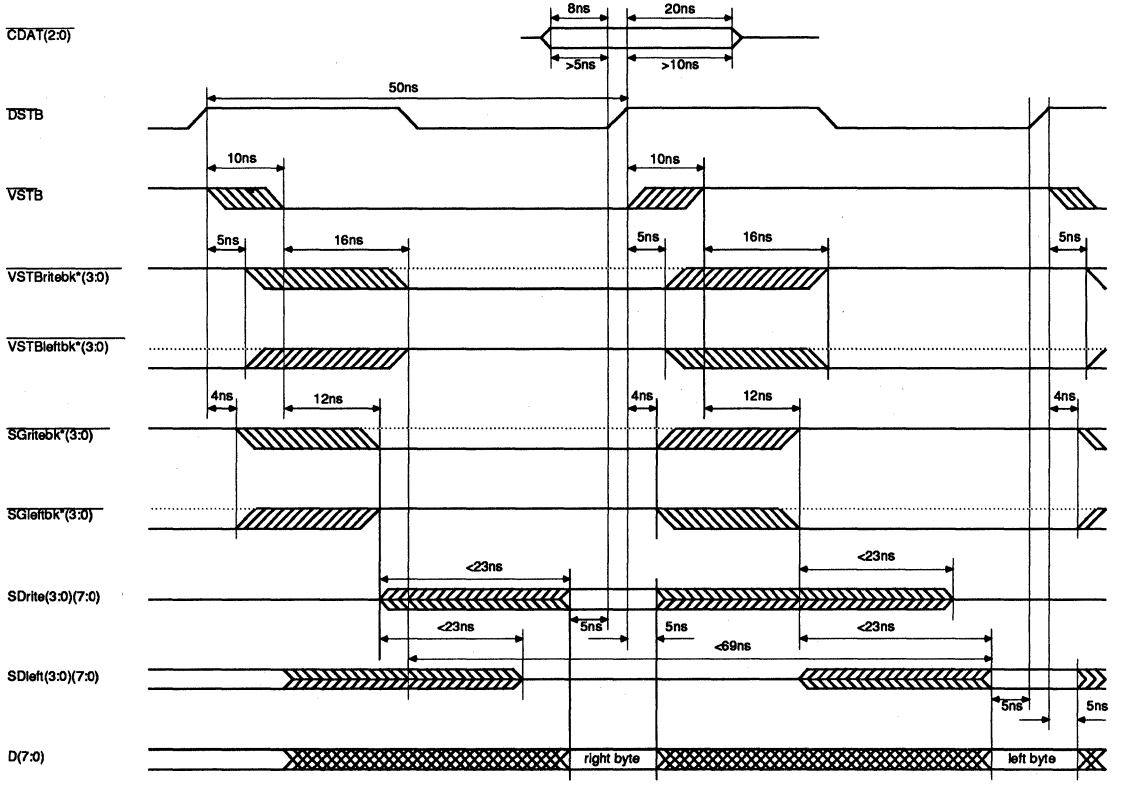


Figure 3.1-9 Video Clock Timing for VDAF

VSTBsidebk*(3:0) signals and phase shifted output enabling with the SGsidebk*(3:0) signals.

The first byte of a 16-bit video word, i. e. the byte to appear first on the screen, is called the left byte. It is strobed out of one of the four banks of the video memory chips by the rising edge of one of the four possible VSTBleftbk*(3:0) clocks. This byte is, however, not transmitted to the VDAFs immediately;SGleftbk*(3:0), the output enable signal of this video memory chip is not enabled yet. With the next rising edge of the DSTB clock the output enable signal SGleftbk*(3:0) becomes active and the left byte of the video word appears on the SDleft(7:0) lines and is brought to the D_{7-0} inputs of the VDAF. While the left byte is strobed into the VDAF, the right byte is shifted inside the video memory chip to the

output buffer. It appears on the $SDrite_{7-0}$ outputs with the next rising edge of DSTB, at the same time when the next left byte is being shifted inside the video memory.

Every rising edge of DSTB from the QPDM changes the polarity of the VSTB signal. The changing of the polarity of the VSTB signal may take anywhere from 0 ns to 10 ns (parameter #80 t_{PD} of the QPDM) after the rising edge of DSTB. This VSTB signal in conjunction with the latched bank address $ADDR_{9-8}$ create the signals VSTBsidebk*(3:0) in the B-speed AmPAL18P8 and and SGsidebk*(3:0) in a F138 multiplexer. The propagation delay through the PAL is 5 ns to 16 ns for a 64 pF (8 devices with 8 pF each) load. The propagation delay through the F138 is 4 ns to 12 ns for a 40 pF (8 devices with 5 pF each) load. This is illustrated in Figure 3.1-9.

To obtain a valid data input signal on the VDAFs, two conditions must be met. First, the clock-to-output delay, and second, the output enable to output delay times of the video memory chips must be fulfilled. For a required data set-up and hold time for the video data with respect to DSTRB of 5 ns (parameter #5 t_S of the VDAF) and 5 ns (parameter #6 t_H of the VDAF), the following maximum propagation delays inside the video memory are required. The maximum output enable to output delay is 23 ns. (One 50 ns DSTB clock cycle minus the 10 ns VSTB propagation delay minus the 12 ns SGsidebk*(3:0) propagation delay minus the 5 ns data set-up time.) The maximum clock to output delay for all normal video clock cycles is 69 ns. (Two 50 ns DSTB clock cycle minus the 10 ns VSTB propagation delay minus the 16 ns VSTBsidebk*(3:0) propagation delay minus the 5 ns data set-up time.) These times are easily fulfilled by video memory chips. After the rising edge of DSTB there is at least a 5 ns delay until SGsidebk*(3:0) becomes inactive. This time satisfies the required hold time of 5 ns for the VDAF.

3.1.5 Physical Board Dimensions

This design represents a typical graphic terminal memory design for the QPDM. It uses, in addition to the QPDM Am95C60, 64 memory ICs of the organization 64K by 4, four VDAF ICs, three driver ICs Am28827A, one flip-flop F74 chip, three flip-flops F114 chips, one decoder F138, two PALs 18P8, and one inverter F04. The required area for these ICs is illustrated in the following Table 3.1-4. The table assumes that all chips are conventional non-surface mount ICs.

The size of the memory board is at least 7.5" by 7". Approximately three-quarters of the board space is occupied by the memory chips and decoupling capacitors.

3.1.6 Considerations for Modification of the Design

This design can be modified to accommodate smaller or larger display memory areas. This may require decreasing or increasing the number of banks for decoding the address.

Table 3.1-4 Required Area for ICs on Memory Board

1	Am95C60	144 pin		2.25 $inch^2$	2.25 $inch^2$
64	Video Memory	24 pin	400 mil	0.60 $inch^2$	38.40 $inch^2$
4	Am8171	24 pin	600 mil	0.84 $inch^2$	3.36 $inch^2$
3	Am29827A	24 pin	300 mil	0.48 $inch^2$	1.44 $inch^2$
1	F74	14 pin	300 mil	0.28 $inch^2$	1.44 $inch^2$
3	F114	14 pin	300 mil	0.28 $inch^2$	0.28 $inch^2$
1	F138	16 pin	300 mil	0.32 $inch^2$	0.32 $inch^2$
2	PAL 18P8	20 pin	300 mil	0.40 $inch^2$	0.80 $inch^2$
1	F04	14 pin	300 mil	0.28 $inch^2$	0.28 $inch^2$
1	F00	14 pin	300 mil	0.28 $inch^2$	0.28 $inch^2$
75	100 nF Caps	2 pin	300 mil	0.03 $inch^2$	2.25 $inch^2$
					50.40 $inch^2$

For a slower display system, the complete decoding hardware may be implemented in PALs. Simultaneously the buffers Am29827A can be replaced by Am2966 or Am2976 dynamic memory drivers which include the serial resistor and provide balanced driving to a highly capacitive load.

If the system allows access to the display memory by another processor, all address and control lines to the display memory must be three-statable. All Am29827A have three-state outputs. They must be controlled by the handshake lines MEMREQ and MEMAVL.

3.2 DISPLAY MEMORY PROGRAM

In Section 3.2 we present the source code for a program which examines the timing parameters for a Display Memory Bus. This program is written for MS Basic for a Macintosh Plus. It runs with either the compiler or interpreter.

For each VRAM parameter, the program calculates the worst-case and the best-case timing margins. These are displayed on the screen and may be optionally written to a file.

The program models a QPDM driving an array of VRAM chips. There are certain delays between the QPDM and VRAMs as follows:

RAS Decode Time to decode RAS in a multi-bank system

RAS Delay Time to buffer and distribute the decoded RAS

CAS Decode Time to decode CAS in a multi-bank system

CAS Delay Time to buffer and distribute the decoded CAS

XFG Decode Time to decode XFG in a multi-bank system

XFG Delay Time to buffer and distribute the decoded XFG

WE Decode Time to decode WE in a multi-bank system

WE Delay Time to buffer and distribute the decoded WE

Address Delay Time to buffer and distribute the multiplexed address

The program uses three values for each delay: minimum, nominal, and maximum. These numbers are typically taken from data sheets. In any case, the rules at your design center should be followed.

Each value for each delay can be built into the program or it can be set at run time. Unfortunately, there is no way to permanently change default values short of changing the program itself. For the interpreted version, this is simple. For the compiled version, it means doing a new compilation.

We calculate three sets of numbers for each parameter. In the first set, we use minimum delays for paths which must be subtracted, and maximum delays for paths which must be added. In the second set, we use nominal delays for all paths (obtaining a sort of "typical" number). In the third case, we use maximum delays for paths which are subtracted and minimum delays for paths which are added. The truth is guaranteed to lie somewhere between the first and third set of numbers.

Consider VRAM parameter 17, tASR. This is the time that the row address is required to be valid at the VRAM before RAS can fall (typically it is 0 ns). This is basically the same as QPDM parameter 30, but we have to correct for decoding and buffer delays. In particular, for this particular parameter, we have to subtract delays in the address paths and add delays in the RAS path.

Here is the program's output for parameter 17.

17, tASR	**Address Setup to RAS (ns)**		
+QPDM Para 35	15.0	15.0	15.0
-Adrs Delay	-10.0	-13.0	-17.0
+RAS Decode	0.0	0.0	0.0
+RAS Delay	12.0	10.0	7.0
Total Time:	17.0	12.0	5.0
VRAM	0.0	0.0	0.0
Margins	17.0	12.0	5.0

The three columns of numbers are for the three cases described above. We can be sure that the truth lies between the two extremes.

Generally, these begin with one or more QPDM parameters. This is followed by an appropriate number of decodes and delays, depending on the VRAM parameter being calculated.

On the Total Time line, the numbers are all summed. This provides the time available at the VRAM. We subtract the VRAM requirements and the remainder is the margin.

```
DIM qpar(4,115)                         'here go the qpdm parameters
ram.vend.count=9                        'number of ram number sets
ram.para.count=98                       'number of parameters to worry about
DIM ram(ram.vend.count,ram.para.count)
DIM ram.vend$(ram.vend.count)           'who built them
DIM ram.text$(ram.para.count)           'keep the ram parameter strings
DIM ram.desc$(ram.para.count)           'ram parameter descriptors
CALL TEXTFONT(4)
PRINT FRE(1)
'we will be using the NEC ram parameter numbers (1..70)
clock(2)=50/2      '20mhz clock
clock(3)=62/2      '16 MHZ clock
clock(4)=83/2      '12 MHz clock
st1$="#####.#"     'one decimal point
st2$="\            \ ########.# ########.# #########.#"
begin:
INPUT "Do you want a file copy of the results (y/n)";a$
IF a$ <> "y" AND a$<>"Y" THEN GOTO begin1
            file=1
            INPUT "Specify Filename";file$
            OPEN file$ FOR OUTPUT AS #1
            GOTO getspeed
begin1:
IF a$<>"n" AND a$<>"N" THEN GOTO begin2
            file=0      'will not write an output file
            GOTO getspeed
begin2:
PRINT "You must respond y or n."
GOTO begin

getspeed:
INPUT "what speed part are you designing with (20,16,12)";a$
a=VAL(a$)          'make it a number
devicespeed=a      'save to compare with clockrate
IF a<>20 THEN GOTO not20
            sp=2   'speed parameter
            GOTO getclock
not20:
IF a<>16 THEN GOTO not16
            sp=3   'speed parameter
            GOTO getclock
not16:
IF a <>12 THEN GOTO not12
            sp=4
            GOTO getclock
not12:
PRINT"you need to enter a number:20, 16, or 12.":GOTO getspeed


getclock:
INPUT "what clock rate (SYSCLK in MHz) are you designing to";a$
a=VAL(a$)
IF a <=devicespeed THEN GOTO speedok
PRINT "I won't let you design with SYSCLK out of spec for part." :GOTO getclock
```

```
speedok:
'force clock(n) to c/2 according to selected clock rate
clock(sp)=500/a    'reciprocal over 2

get.fake:
INPUT "Do we use the stored delays?(y/n)";a$
      IF a$="y" OR a$="Y"  THEN
      fake.numbers=1 :GOTO got.fake
      END IF
      IF a$="n" OR a$="N" THEN
      fake.numbers=0 :GOTO got.fake
      END IF
      PRINT "You must respond 'y' or      'n'"
      GOTO get.fake


got.fake:
IF fake.numbers=1 THEN GOTO read.fakes     'otherwise get from keyboard
INPUT "Specify /RAS Decode min,nom,max ";ras.decode(1),ras.decode(2),ras.decode(3)
INPUT "Specify /RAS Delay min,nom,max";ras.delay(1),ras.delay(2),ras.delay(3)
INPUT "Specify /CAS Decode min,nom,max";cas.decode(1),cas.decode(2),cas.decode(3)
INPUT "Specify /CAS Delay min,nom,max";cas.delay(1),cas.delay(2),cas.delay(3)
INPUT "Specify /XFG Decode min,nom,max";xfg.decode(1),xfg.decode(2),xfg.decode(3)
INPUT "Specify /XFG Delay min,nom,max";xfg.delay(1),xfg.delay(2),xfg.delay(3)
INPUT "specify /WE Decode min,nom,max ";we.decode(1),we.decode(2),we.decode(3)
INPUT "Specify /WE Delay min,nom,max";we.delay(1),we.delay(2),we.delay(3)
INPUT "Specify Address delay min, nom,max ";ad.delay(1),ad.delay(2),ad.delay(3)

get.cas:
INPUT "Are you going to make early /CAS from /XF-G  (y/n)";a$
IF a$<>"n" AND a$<>"N" THEN GOTO notn
early.cas(0)=0     'no early cas
GOTO ram.para
notn:
IF a$<>"y" AND a$<>"Y" THEN GOTO noty
early.cas(0)=1     'we will be doing early cas
INPUT "Specify /XF-G to early /CAS delay min,nom,max
";early.cas(1),early.cas(2),early.cas(3)
GOTO ram.para
noty:
PRINT "you must respond with 'n' or 'y'":GOTO get.cas


read.fakes:
ras.decode(1)=0:      ras.decode(2)=0:      ras.decode(3)=0
ras.delay(1)=7:       ras.delay(2)=10:      ras.delay(3)=12
cas.decode(1)=0:      cas.decode(2)=0:      cas.decode(3)=0
cas.delay(1)=5:       cas.delay(2)=10:      cas.delay(3)=15
xfg.decode(1)=0:      xfg.decode(2)=0:      xfg.decode(3)=0
xfg.delay(1)=6:       xfg.delay(2)=9:       xfg.delay(3)=11
we.decode(1)=0:       we.decode(2)=0:       we.decode(3)=0
we.delay(1)=6:        we.delay(2)=9:        we.delay(3)=11
ad.delay(1)=10:       ad.delay(2)=13:       ad.delay(3)=17
early.cas(0)=0:       early.cas(1)=8:       early.cas(2)=10:  early.cas(3)=12
```

```
ram.para:
FOR i=1 TO ram.para.count
      FOR j=1 TO ram.vend.count
      READ ram(j,i)      'read ram parameters
'PRINT USING "######";ram(j,i),
      NEXT j
      READ ram.text$(i) 'and the text
'PRINT ,ram.text$(i)
      READ ram.desc$(i) 'and the description
'PRINT ,ram.desc$(i)
      NEXT i
GOTO ram.vendors

'these are the data which are moved into table RAM

'it is ordered first by NEC parameter number and secondly by device

'the second ordering is the same as RAM.VEND$

'a value of 99 says that parameter is not defined for that vendor

DATA 220,270,220,260,190,220,260,230,260,tRC,Read Write Transfer
DATA 300,365,295,345,260,300,355,300,345,tRWC,RMW Cycle
DATA 120,145,120,145,70,85,105,120,145,tPC,Page Mode Cycle
DATA 120,150,120,150,100,120,150,120,150,tRAC,Row Access
DATA 60,75,60,75,50,60,75,60,75,tCAC,Column Access                    :REM 5

DATA 0,0,0,0,0,0,0,0,0,tOFF,Output Disable from CAS HI
DATA 3,3,3,3,3,3,3,3,3, tT,Transition
DATA 90,100,90,100,80,90,100,100,100,tRP,RAS Precharge
DATA 120,150,120,150,100,120,150,120,150,tRAS,RAS Pulse Width
DATA 60,75,60,75,50,60,75,60,75,tRSH,CAS Falls to RAS Rises           :REM 10

DATA 25,30,25,30,99,99,99,50,60,tCPN,CAS Precharge (Not PM)
DATA 50,60,50,60,10,15,20,50,60,tCP,CAS Precharge (PM)
DATA 60,75,10,75,50,60,75 ,60,75,tCAS,CAS Pulse Width
DATA 120,150,120,150,100,120,150,120,150,tCSH,CAS Hold From RAS Falls
DATA 60,75,60,75,50,60,75,60,75,tRCD,RAS to CAS Delay                 :REM 15

DATA 10,10,10,10,10,10,10,0,0,tCRP,CAS Hi to RAS Low Precharge
DATA 0,0,0,0,0,0,0,0,0, tASR,Address Setup to RAS
DATA 15,20,15,20,15,15,20,15,15,tRAH,Row Address Hold
DATA 0,0,0,0,0,0,0,0,0,tASC,Address Setup to CAS
DATA 20,25,20,25,20,20,25,20,25,tCAH,Column Address Hold              :REM 20

DATA 80,100,80,100,99,99,99,80,100,tAR,Column Address Hold from RAS
DATA 0,0,0,0,0,0,0,0,0,tRCS,Read Command Setup to CAS
DATA 20,20,20,20,10,10,10,10,10,tRRH,Read Command Hold from RAS Hi
DATA 0,0,0,0,0,0,0,0,0,tRCH,Read Command Hold from CAS Hi
DATA 0,0,0,0,0,0,0,-5,-5,tWCS,Write Command Setup to CAS               :REM 25

DATA 35,45,35,45,25,25,35,35,45,tWCH,Write Command Hold
DATA 95,120,95,120,99,99,99,95,120,tWCR,Write CMND Hold from RAS Falls
DATA 35,45,35,45,15,20,25,35,45,tWP,Write Pulse Width
DATA 40,45,40,45,35,40,45,35,45, tRWL,Write Command to RAS
DATA 40,45,40,45,30,40,45,35,45,tCWL,Write Command to CAS Lead Time   :REM 30
```

```
DATA 0,0,0,0,0,0,0,0,0,tDS,Data Setup to CAS
DATA 35,45,35,45,25,25,30,35,45,tDH,Data Hold from CAS
DATA 95,120,95,120,99,99,99,95,120,DHR,Data Hold from RAS
DATA 60,75,100,120,85,100,125,90,110,tCWD,CAS to WE Delay
DATA 120,150,160,195,99,99,99,150,185,tRWD,RAS to WE Delay              :REM 35

DATA 30,40,35,40,30,35,40,40,45,tOEA,Access from OE
DATA 35,40,35,40,25,30,40,99,99,tOED,OE High to Data in Setup
DATA 30,40,30,40,10,15,20,0,0,tOEH,OE Hi hold from WE Low
DATA 30,40,30,40,0,0,0,25,30,tOEZ,Output Disable from OE Hi
DATA 10,10,10,10,10,10,10,20,25,tCSR,CAS to RAS Setup for Refresh       :REM 40

DATA 25,30,25,30,20,25,30,20,25,tCHR,CAS before RAS Refresh Hold
DATA 0,0,0,0,10,10,10,99,99,tRPC,RAS Hi to CAS Lo Precharge
DATA 4,4,4,4,4,4,4,4,4,tREF,Refresh Interval
DATA 0,0,0,0,0,0,0,0,0,tDLS,DT to RAS Setup for Xfer
DATA 100,130,90,130,80,90,110,99,99,tRDH,DT Hold from RAS for Xfer      :REM 45

DATA 40,55,40,55,99,99,99,99,99,tCDH,DT Hold After CAS LO
DATA 10,20,20,25,99,99,99,10,15,tSDD,SC Hi to DT Hi Delay
DATA 10,20,10,20,99,99,99,10,15,tSDH,SC Low Hold after DT Hi
DATA 35,40,35,40,99,99,99,0,0,tOE,OE Pulse Width
DATA 30,40,99,99,25,25,30 ,20,25,tSOZ,Serial Output Disable             :REM 50

DATA 40,60,40,60,40,40,60,40,50,tSCC,Serial Clock Cycle
DATA 10,20,10,20,10,10,20,10,10,tSCH,Serial Clock Hi
DATA 10,20,15,20,10,10,10,10,10,tSCL,Serial Clock Precharge (LOW)
DATA 5,5,5,5,99,99,99,99,99,tSOO,SOE LOW to Serial Out Setup
DATA 35,50,35,50,25,30,40,20,25,tSOA,Serial Access from SOE             :REM 55

DATA 10,10,10,10,10,10,10,8,8,tSOH,Serial Out Hold after SC Lo
DATA 40,60,40,50,40,40,60,40,50,tSCA,Serial Access from SC
DATA 0,0,0,0,0,0,0,0,0,tDHS,DT Hi Setup to RAS (no XFER)
DATA 20,25,20,25,99,99,99,15,15,tDHH,DT Hold from RAS
DATA 10,10,10,10,10,10,10,-10,-10,tDTR,DT Hi to RAS Hi Delay            :REM 60

DATA 10,10,10,10,10,10,10,99,99,tDTC,DT Hi to CAS Hi Delay
DATA 10,10,30,40,99,99,99,99,99,tOES,OE Setup to RAS Hi
DATA 0,0,0,0,0,0,0,0,0,tCOD,unused parameter
DATA 0,0,0,0,0,0,0,0,0,tWBS,Masked Write Command Setup
DATA 20,25,20,25,15,15,20,15,15,tWBH,Masked Write Command Hold           :REM 65

DATA 0,0,0,0,0,0,0,0,0,tWS,Write Mask Setup
DATA 20,25,20,25,15,15,20,15,15,tWH,Write Mask Hold
DATA 15,20,10,20,99,99,99,99,99,tSOE,SOE Pulse Width(Lo)
DATA 15,20,10,20,99,99,99,99,99,tSOP,SOE Precharge (Hi)
DATA 20,25,20,25,15,15,20,99,99,tDTH,DT HI Hold after RAS Hi            :REM 70

DATA 99,99,30,40,99,99,99,99,99,th(OECH),CAS hold after OE low
DATA 99,99,99,99,99,99,99,99,99,th(OERH),Unused
DATA 99,99,120,150,99,99,99,99,99,th(RLOE),OE hold after RAS low
DATA 99,99,0,0,99,99,99,99,99,tDOEL,Delay Data to OE low
DATA 99,99,0,0,99,99,99,99,99,tDCL,Delay data to CAS low               :REM 75
```

```
DATA 99,99,10,20,99,99,99,99,99,tw(SEL),SOE low pulse width
DATA 99,99,10,20,99,99,99,99,99,tw(SEH), SOE high pulse width
DATA 99,99,0,0,99,99,99,0,0,tsu(WE),WE setup to RAS low
DATA 99,99,20,25,99,99,99,15,15,th(WE),WE hold after RAS low
DATA 99,99,0,0,99,99,99,0,0,tsu(SE),SE setup to RAS low                    :REM 80

DATA 99,99,15,20,99,99,99,15,15,sh(SE),SE hold after RAS low
DATA 99,99,0,0,0,0,0,0,0,tsu(SD),Serial in setup to SC high
DATA 99,99,10,15,15,20,25,15,15,th(SD),Serial in hold after SC high
DATA 99,99,20,30,99,99,99,99,99,tsu(SCRL),SC setup to RAS low
DATA 99,99,10,15,99,99,99,99,99,tsu(SEH),SE disable setup to SC high     :REM 85

DATA 99,99,20,30,99,99,99,99,99,th(SEH),SE disable hold from SC high
DATA 99,99,10,15,99,99,99,25,30,tsu(SEL),SE enable setup before SC high
DATA 99,99,20,30,99,99,99,99,99,th(SEL),SE enable hold from SC high
DATA 99,99,0,0,99,99,99,99,99,tDDTH,Delay data to DT high
DATA 99,99,20,30,99,99,99,25,30,tDTHD,Delay DT high to data              :REM 90

DATA 99,99,99,99,99,99,99,40,45,tw(TRG),tRG Pulse width
DATA 99,99,99,99,99,99,99,60,75,tCLGH,CAS low to TRG high
DATA 99,99,99,99,99,99,99,100,120,tRLSH,RAS low to SC high after TRG hi
DATA 99,99,99,99,99,99,99,100,100,tTHRL,tRG high to RAS low after xfer
DATA 99,99,99,99,99,99,99,40,45,tCLSH,CAS low to SC after TRG            :REM 95

DATA 99,99,99,99,99,99,99,40,45,tSHRL,SC high to RAS low (w/xfer)
DATA 99,99,99,99,99,99,99,30,45,tRHSH,RAS high to SC high
DATA 99,99,99,99,99,99,99,10,15,tTHSH,tRG high to SC high

ram.vendors:
ram.vend$(1)="NEC uPD41264-12"
ram.vend$(2)="NEC uPD41264-15"
ram.vend$(3)="Mitsubishi M5M4C264P-12"
ram.vend$(4)="Mitsubishi M5M4C264-15"
ram.vend$(5)="Hitachi HM53461-10"
ram.vend$(6)="Hitachi HM53461-12"
ram.vend$(7)="Hitachi HM53461-15"
ram.vend$(8)="T.I. TMS4461-12"
ram.vend$(9)="T.I. TMS4461-15"
GOTO qpdm.para

qpdm.para:
FOR i=1 TO 115
      FOR j=1 TO 4
      READ qpar(j,i)
      NEXT j
NEXT i
GOTO ramkind

FOR i=1 TO 115
      PRINT USING "#####";i;
      IF qpar(1,i)<>0 THEN GOTO inuse
      PRINT "       parameter is not used."
      GOTO loopend
inuse:
      IF qpar(1,i)>0 THEN GOTO formula
```

```
      PRINT USING st1$;qpar(2,i),qpar(3,i),qpar(4,i)
      GOTO loopend
formula:
      FOR j=2 TO 4
      PRINT USING st1$;(qpar(1,i)*clock(j))+qpar(j,i),
      NEXT j
      PRINT
      loopend:
NEXT i
GOTO ramkind
'now the 95C60 parameters
'this is ordered by parameter number, 1..115
'there are four entries for each parameter number
'the first number:0 ->number unused.-1->normal,>0->uses formulas,value is
'clock half cycles.  1 -> c/2, 7-> 7c/2, etc
'the other three numbers are for -20, -16, -12.  Values if no note 4, else note 4
adders
'system bus timing
DATA -1,0,0 ,0          :REM 1
DATA -1,65,95,125       :REM 2
DATA -1,50,60,70        :REM 3
DATA -1,50,60,70        :REM 4
DATA -1,10,10,10        :REM 5
DATA -1,65,70,75        :REM 6
DATA  -1,0,0,0          :REM 7
DATA -1,10,10,10        :REM 8
DATA -1,0,0,0           :REM 9
DATA -1,110,110,120     :REM 10
DATA -1,10,10,10        :REM 11
DATA -1,35,40,45        :REM 12
DATA 0,0,0,0            :REM 13 unused
DATA -1,10,20,20        :REM 14
DATA -1,10,20,20        :REM 15
DATA -1,0,0,0           :REM 16
DATA -1,70,90,110       :REM 17
DATA -1,50,75,100       :REM 18
DATA -1,15,25,25        :REM 19 byte mode
DATA -1,120,150,180     :REM 20
DATA -1,0,0,0           :REM 21
DATA -1,0,0,0           :REM 22
DATA 0,0,0,0            :REM 23 ususd
DATA 0,0,0,0            :REM 24
DATA 0,0,0,0            :REM 25
DATA 0,0,0,0            :REM 26
DATA 0,0,0,0            :REM 27
DATA 0,0,0,0            :REM 28
DATA 0,0,0,0            :REM 29
DATA 1,-10,-15,-20      :REM 30 (uses formula)
DATA 2,-15,-17,-20      :REM 31
DATA 7,-15,-15,-20      :REM 32
DATA 8,-20,-23,-25      :REM 33
DATA 1,-11,-16,-20      :REM 34
DATA 4,-5,-15,-20       :REM 35
DATA 3,-10,-15,-20      :REM 36
DATA 1,-12,-16,-20      :REM 37
DATA 4,-20,-20,-22      :REM 38
```

```
DATA 5,-25,-25,-27      :REM 39
DATA 2,-10,-15,-20      :REM 40
DATA 4,-20,-24,-26      :REM 41
DATA 2,-10,-15,-20      :REM 42
DATA 1,-12,-15,-20      :REM 43
DATA 5,-15,-17,-20      :REM 44
DATA -1,20,30,45        :REM 45 (no formula)
DATA -1,0,0,0           :REM 46
DATA 2,-13,-17,-20      :REM 47
DATA 8,-15,-23,-25      :REM 48
DATA 1,-13,-18,-20      :REM 49
DATA 1,-15,-20,-20      :REM 50
DATA 3,-10,-15,-20      :REM 51
DATA 4,-10,-15,-20      :REM 52
DATA 2,-10,-15,-20      :REM 53
DATA 4,-10,-15,-20      :REM 54
DATA 2,-10,-15,-20      :REM 55
DATA 4,-10,-15,-20      :REM 56
DATA 4,-20,-24,-26      :REM 57
DATA 8,-15,-18,-20      :REM 58
DATA 1,-14,-17,-20      :REM 59
DATA 1,-12,-17,-20      :REM 60
DATA 4,-22,-24,-26      :REM 61
DATA -1,2,7,15          :REM 62
DATA 3,-15,-20,-20      :REM 63
DATA -1 ,2,7,15         :REM 64
DATA 3,-15,-20,-20      :REM 65
DATA -1,-8,-12,-16      :REM 66 (note negative numbers)
DATA 0,0,0,0            :REM 67 (unused
DATA 0,0,0,0            :REM 68
DATA 0,0,0,0            :REM 69
DATA 0,0,0,0            :REM 70
DATA 0,0,0,0            :REM 71
DATA 0,0,0,0            :REM 72
DATA 0,0,0,0            :REM 73
DATA 0,0,0,0            :REM 74
DATA 0,0,0,0            :REM 75
DATA 0,0,0,0            :REM 76
DATA 0,0,0,0            :REM 77
DATA 0,0,0,0            :REM 78
DATA 0,0,0,0            :REM 79
DATA -1,10,10,10        :REM 80  (also has a min of 0)
DATA -1,0,0,5           :REM 81
DATA -1,20,30,40        :REM 82
DATA 2,0,0,0            :REM 83
DATA -1,25,50,75        :REM 84
DATA 2,0,0,0            :REM 85
DATA 6,0,0,0            :REM 86
DATA 0,0,0 ,0           :REM 87 (unused)
DATA 0,0,0,0            :REM 88
DATA 0,0,0,0            :REM 89
DATA 5,35,35,35         :REM 90
DATA 18,0,0,0           :REM 91
DATA 0,0,0 ,0           :REM 92
DATA 0,0,0 ,0           :REM 93
DATA 0,0,0 ,0           :REM 94
```

```
DATA 0,0,0 ,0           :REM 95
DATA 0,0,0 ,0           :REM 96
DATA 0,0,0,0            :REM 97
DATA 0,0,0,0            :REM 98
DATA 0,0,0 ,0           :REM 99
DATA -1,25,31,41        :REM 100
DATA -1,0,0,0           :REM 101
DATA -1,15,20,25        :REM 102
DATA -1,30,40,50        :REM 103
DATA -1,15,20,25        :REM 104
DATA -1,15,20,25        :REM 105
DATA -1,50,62,83        :REM 106
DATA -1,5,5,5           :REM 107
DATA -1,18,23,32        :REM 108
DATA -1,18,23,32        :REM 109
DATA -1,66,72,83        :REM 110
DATA -1,5,5,5           :REM 111
DATA -1,25,27,32        :REM 112
DATA -1,25,27,32        :REM 113
DATA 8,0,0,0            :REM 114
DATA -1 ,20,25,30       :REM 115


ramkind:
PRINT "Please specify the RAM chips you are designing for:'
FOR i=1 TO ram.vend.count
PRINT i,ram.vend$(i)
NEXT i
INPUT ram.point
IF ram.point >0 AND ram.point<ram.vend.count+1 THEN GOTO membus
PRINT "you must specify a number in the range displayed."
GOTO ramkind


membus:
      GOSUB out.top      'print the time and date
param1:
ramp=1: GOSUB out.param
text$="QPDM guarantees 6 SYSCLK cycles = "+STR$(12*clock(sp)) :GOSUB out.string
text$="RAM requires  "+STR$(ram(ram.point,1)) :GOSUB out.string
IF 12*clock(sp) < ram(ram.point,1) THEN GOSUB out.problem

param2:
ramp=2 :GOSUB out.param
text$= "QPDM never does Read/Modify/Write Cycles.": GOSUB out.string

param3:
ramp=3: GOSUB out.param
text$= "QPDM never does Page Mode Cycles.":GOSUB out.string

param4:
ramp=4 : GOSUB out.param
      qpdmp=32 :GOSUB pos.qpdm
GOSUB neg.ras
qpdmp=45: GOSUB neg.qpdm
GOSUB totals
```

```
GOSUB finish

param5:
ramp=5 : GOSUB out.param
IF early.cas(0)=1 THEN GOTO param5.1
qpdmp=41: GOSUB pos.qpdm
GOSUB neg.cas
qpdmp=45: GOSUB neg.qpdm
GOSUB totals
GOSUB finish
GOTO param6
param5.1:
qpdmp=44: GOSUB pos.qpdm
GOSUB neg.e.cas
GOSUB neg.cas
qpdmp=45: GOSUB neg.qpdm
GOSUB totals
GOSUB finish

param6:
ramp=6 :GOSUB out.param
qpdmp=46 :GOSUB neg.qpdm
qpdmp=43 :GOSUB pos.qpdm
GOSUB pos.cas
GOSUB totals
GOSUB finish

param7:
ramp=7 :GOSUB out.param
text$=ram.desc$(ramp): min=ram(ram.point,ramp)
nom=min: max=min :GOSUB out.values
GOTO param8

param8:
ramp=8: GOSUB out.param
qpdmp=35 :GOSUB pos.qpdm
GOSUB totals
GOSUB finish

param9:
ramp=9: GOSUB out.param
qpdmp=33 :GOSUB pos.qpdm
GOSUB totals
GOSUB finish

param10:
ramp=10: GOSUB out.param
qpdmp=39 :GOSUB pos.qpdm
GOSUB totals
GOSUB finish

param11:
ramp=11: GOSUB out.param
qpdmp=40 :GOSUB pos.qpdm
GOSUB totals
```

```
GOSUB finish

param12:
ramp=12: GOSUB out.param
text$="QPDM never does Page Mode Cycles." : GOSUB out.string
GOTO param13

param13:
ramp=13:GOSUB out.param
qpdmp=57 :GOSUB pos.qpdm
GOSUB totals
GOSUB finish

param14:
ramp=14 :GOSUB out.param
qpdmp=33 :GOSUB pos.qpdm
GOSUB neg.ras
GOSUB pos.cas
GOSUB totals
GOSUB finish

param15:
ramp=15: GOSUB out.param
IF early.cas(0)=1 THEN GOTO param15.1
qpdmp=36: GOSUB pos.qpdm
GOSUB neg.ras
GOSUB pos.cas
GOSUB totals
GOSUB finish
GOTO param16
param15.1:
qpdmp=42: GOSUB pos.qpdm
GOSUB neg.ras
GOSUB pos.e.cas
GOSUB pos.cas
GOSUB totals
GOSUB finish
GOTO param16

param16:
ramp=16 :GOSUB out.param
qpdmp=35 :GOSUB pos.qpdm
GOSUB neg.cas
GOSUB pos.ras
GOSUB totals
GOSUB finish

param17:
ramp=17 :GOSUB out.param
qpdmp=30 :GOSUB pos.qpdm
GOSUB neg.adrs
GOSUB pos.ras
GOSUB totals
GOSUB finish

param18:
```

```
ramp=18:GOSUB out.param
qpdmp=31 :GOSUB pos.qpdm
GOSUB pos.adrs
GOSUB neg.ras
GOSUB totals
GOSUB finish

param19:
ramp=19: GOSUB out.param
IF early.cas(0)=1 THEN GOTO param19.1
qpdmp=37: GOSUB pos.qpdm
GOSUB neg.adrs
GOSUB pos.cas
GOSUB totals
GOSUB finish
GOTO param20
param19.1:
qpdmp=66 :GOSUB pos.qpdm
GOSUB neg.adrs
GOSUB pos.e.cas
GOSUB pos.cas
GOSUB totals
GOSUB finish
GOTO param20

param20:
ramp=20: GOSUB out.param
qpdmp=38: GOSUB pos.qpdm
GOSUB pos.adrs
GOSUB neg.cas
GOSUB totals
GOSUB finish

param21:
ramp=21: GOSUB out.param
qpdmp=56: GOSUB pos.qpdm
qpdmp=38: GOSUB pos.qpdm
GOSUB pos.adrs
GOSUB neg.ras
GOSUB totals
GOSUB finish

param22:
ramp=22: GOSUB out.param
qpdmp=43: GOSUB pos.qpdm
qpdmp=40: GOSUB pos.qpdm
GOSUB neg.xfg
GOSUB pos.cas
GOSUB totals
GOSUB finish

param23:
ramp=23: GOSUB out.param
qpdmp=35 :GOSUB pos.qpdm
qpdmp=59 :GOSUB neg.qpdm
GOSUB pos.xfg
```

```
GOSUB neg.ras
GOSUB totals
GOSUB finish

param24:
ramp=24: GOSUB out.param
qpdmp=35 :GOSUB pos.qpdm
qpdmp=59 :GOSUB neg.qpdm
GOSUB pos.xfg
GOSUB neg.cas
GOSUB totals
GOSUB finish

param25:
ramp=25:GOSUB out.param
qpdmp=60 :GOSUB pos.qpdm
GOSUB pos.we
GOSUB neg.cas
GOSUB totals
GOSUB finish

param26:
ramp=26:GOSUB out.param
qpdmp=61 :GOSUB pos.qpdm
qpdmp=60 :GOSUB neg.qpdm
GOSUB pos.we
GOSUB neg.cas
GOSUB totals
GOSUB finish

param27:
ramp=27:GOSUB out.param
qpdmp=61 :GOSUB pos.qpdm
qpdmp=56 :GOSUB pos.qpdm
qpdmp=60 :GOSUB neg.qpdm
GOSUB pos.we
GOSUB neg.ras
GOSUB totals
GOSUB finish

param28:
ramp=28 :GOSUB out.param
qpdmp=61 :GOSUB pos.qpdm
GOSUB totals
GOSUB finish

param29:
ramp=29:GOSUB out.param
qpdmp=33: GOSUB pos.qpdm
qpdmp=56: GOSUB neg.qpdm
qpdmp=60: GOSUB pos.qpdm
GOSUB pos.ras
GOSUB neg.we
GOSUB totals
```

```
GOSUB finish

param30:
ramp=30 :GOSUB out.param
qpdmp=60:GOSUB pos.qpdm
qpdmp=57: GOSUB pos.qpdm
GOSUB pos.cas
GOSUB neg.we
GOSUB totals
GOSUB finish

param31:
ramp=31:GOSUB out.param
qpdmp=64 :GOSUB pos.qpdm
GOSUB pos.cas
GOSUB totals
GOSUB finish

param32:
ramp=32:GOSUB out.param
qpdmp=65 :GOSUB pos.qpdm
GOSUB neg.cas
GOSUB totals
GOSUB finish

param33:
ramp=33:GOSUB out.param
qpdmp=56 :GOSUB pos.qpdm
qpdmp=65 :GOSUB pos.qpdm
GOSUB neg.ras
GOSUB totals
GOSUB finish

param34:
ramp=34 :GOSUB out.param
text$= "QPDM never does Read/Modify/Write Cycles.": GOSUB out.string

param35:
ramp=35:GOSUB out.param
text$= "QPDM never does Read/Modify/Write Cycles.": GOSUB out.string

param36:
ramp=36:GOSUB out.param
qpdmp=44 :GOSUB pos.qpdm
GOSUB neg.xfg
qpdmp=45 :GOSUB  neg.qpdm
GOSUB totals
GOSUB finish

param37:
ramp=37 :GOSUB out.param
text$= "QPDM never does Read/Modify/Write Cycles.": GOSUB out.string

param38:
ramp=38:GOSUB out.param
```

```
text$= "QPDM never does Read/Modify/Write Cycles.": GOSUB out.string

param39:
ramp=39:GOSUB out.param
qpdmp=46 :GOSUB pos.qpdm
GOSUB  pos.xfg
GOSUB totals
GOSUB neg.finish

param40:
ramp=40:GOSUB out.param
qpdmp=47 :GOSUB pos.qpdm
GOSUB pos.ras
GOSUB neg.cas
GOSUB totals
GOSUB finish

param41:
ramp=41:GOSUB out.param
qpdmp=48 :GOSUB pos.qpdm
GOSUB neg.ras
GOSUB pos.cas
GOSUB totals
GOSUB finish

param42:
ramp=42:GOSUB out.param
qpdmp=40 :GOSUB pos.qpdm
GOSUB pos.cas
GOSUB neg.ras
GOSUB totals
GOSUB finish

param43:
ramp=43: GOSUB out.param
intval=ram(ram.point,ramp)*(1000000!/256) 'refresh in nanosec
text$="You may program DMRR to "+STR$(INT(intval/clock(sp)+.99))
GOSUB out.string

param44:
ramp=44:GOSUB out.param
qpdmp=49 :GOSUB pos.qpdm
GOSUB neg.xfg
GOSUB pos.ras
GOSUB totals
GOSUB finish

param45:
ramp=45:GOSUB out.param
qpdmp=32 :GOSUB pos.qpdm
GOSUB neg.ras
GOSUB pos.xfg
GOSUB totals
GOSUB finish

param46:
```

```
ramp=46: GOSUB out.param
qpdmp=41 :GOSUB pos.qpdm
GOSUB neg.cas
GOSUB pos.xfg
GOSUB totals
GOSUB finish

param47:
ramp=47: GOSUB out.param
qpdmp=52: GOSUB pos.qpdm
qpdmp=32: GOSUB pos.qpdm
GOSUB pos.xfg
GOSUB totals
GOSUB finish

param48:
ramp=48: GOSUB out.param
qpdmp=34: GOSUB pos.qpdm
qpdmp=53: GOSUB pos.qpdm
GOSUB totals
GOSUB finish

param49:
ramp=49: GOSUB out.param
qpdmp=44: GOSUB pos.qpdm
GOSUB totals
GOSUB finish

param50:
param51:
param52:
param53:
param54:
param55:
param56:
param57:
param58:
ramp=58: GOSUB out.param
qpdmp=43: GOSUB pos.qpdm
qpdmp=35: GOSUB pos.qpdm
GOSUB pos.ras
GOSUB neg.xfg
GOSUB totals
GOSUB finish

param59:
ramp=59: GOSUB out.param
qpdmp=42: GOSUB pos.qpdm
GOSUB neg.ras
GOSUB pos.xfg
GOSUB totals
GOSUB finish

param60:
ramp=60: GOSUB out.param
qpdmp=34: GOSUB pos.qpdm
GOSUB pos.ras
```

```
GOSUB neg.xfg
GOSUB totals
GOSUB finish

param61:
ramp=61: GOSUB out.param
qpdmp=43:GOSUB pos.qpdm
GOSUB pos.cas
GOSUB neg.xfg
GOSUB totals
GOSUB finish

param62:
ramp=62: GOSUB out.param
qpdmp=44: GOSUB pos.qpdm
qpdmp=34: GOSUB pos.qpdm
GOSUB neg.xfg
GOSUB pos.ras
GOSUB totals
GOSUB finish

param63:
param64:
ramp=64: GOSUB out.param
qpdmp=59: GOSUB pos.qpdm
GOSUB neg.we
GOSUB pos.ras
GOSUB totals
GOSUB finish

param65:
ramp=65: GOSUB out.param
qpdmp=60: GOSUB pos.qpdm
qpdmp=61: GOSUB pos.qpdm
GOSUB pos.ras
GOSUB neg.we
GOSUB totals
GOSUB finish

param66:
ramp=66: GOSUB out.param
qpdmp=52: GOSUB pos.qpdm
GOSUB pos.ras
GOSUB totals
GOSUB finish

param67:
ramp=67: GOSUB out.param
qpdmp=63: GOSUB pos.qpdm
GOSUB neg.ras
GOSUB totals
GOSUB finish

param68:
```

```
param69:

param70:
ramp=70: GOSUB out.param
qpdmp=35: GOSUB pos.qpdm
qpdmp=49: GOSUB neg.qpdm
GOSUB neg.ras
GOSUB pos.xfg
GOSUB totals
GOSUB finish

param71:
ramp=71 :GOSUB out.param
qpdmp=44:GOSUB pos.qpdm
qpdmp=43: GOSUB pos.qpdm
GOSUB neg.xfg
GOSUB pos.cas
GOSUB totals
GOSUB finish
param72:

param73:
ramp=73 :GOSUB out.param
qpdmp=42 :GOSUB pos.qpdm
qpdmp=44 :GOSUB pos.qpdm
GOSUB neg.ras
GOSUB pos.xfg
GOSUB totals
GOSUB finish

param74:
ramp=74: GOSUB out.param
text$="QPDM never does hidden refresh cycles."  :GOSUB out.string

param75:
ramp=75: GOSUB out.param
qpdmp=57 :GOSUB pos.qpdm
qpdmp=40 :GOSUB pos.qpdm
qpdmp=65 :GOSUB pos.qpdm
GOSUB neg.cas
GOSUB totals
GOSUB finish

param76:

param77:

param78:
ramp=78: GOSUB out.param
text$="QPDM never does write transfer cycles.":GOSUB out.string

param79:
ramp=79: GOSUB out.param
text$="QPDM never does write transfer cycles.":GOSUB out.string

param80:
ramp=80: GOSUB out.param
```

```
text$="QPDM never does write transfer cycles.":GOSUB out.string

param81:
ramp=81: GOSUB out.param
text$="QPDM never does write transfer cycles.":GOSUB out.string

param82:
ramp=82 :GOSUB out.param
text$="QPDM never does serializer writes.":GOSUB out.string

param83:
ramp=83 :GOSUB out.param
text$="QPDM never does serializer writes.":GOSUB out.string

param84:
ramp=84 :GOSUB out.param
text$="QPDM never does serializer writes.":GOSUB out.string

param85:
ramp=85 :GOSUB out.param
text$="QPDM never does serializer writes.":GOSUB out.string

param86:
ramp=86 :GOSUB out.param
text$="QPDM never does serializer writes.":GOSUB out.string

param87:
ramp=87 :GOSUB out.param
text$="QPDM never does serializer writes.":GOSUB out.string

param88:
ramp=88 :GOSUB out.param
text$="QPDM never does serializer writes.":GOSUB out.string

param89:
ramp=89 :GOSUB out.param
text$="QPDM never does serializer writes.":GOSUB out.string

param90:
ramp=90 :GOSUB out.param
text$="QPDM never does serializer writes.":GOSUB out.string

param91:
ramp=91 :GOSUB out.param
qpdmp=44 :GOSUB pos.qpdm
GOSUB totals
GOSUB finish

param92:
ramp=92 :GOSUB out.param
qpdmp=41 :GOSUB pos.qpdm
GOSUB totals
GOSUB finish

param93:
```

```
ramp=93 :GOSUB out.param
text$="this will handled only on REV C QPDM silicon.":GOSUB out.string

param94:
ramp=94 :GOSUB out.param
qpdmp=34 :GOSUB pos.qpdm
qpdmp=35 :GOSUB pos.qpdm
GOSUB pos.ras
GOSUB neg.xfg
GOSUB totals
GOSUB finish

param95:
ramp=95 :GOSUB out.param
text$="this will handled only on REV C QPDM silicon.":GOSUB out.string

param96:
ramp=96: GOSUB out.param
text$="QPDM never does write transfer cycles.":GOSUB out.string

param97:
ramp=97: GOSUB out.param
text$="QPDM never does write transfer cycles.":GOSUB out.string

param98:
ramp=98: GOSUB out.param
text$="this will handled only on REV C QPDM silicon.":GOSUB out.string


closeup:
IF file=1 THEN CLOSE #1
IF file=0 THEN GOTO w1
OPEN file$ FOR INPUT AS #1
WHILE NOT EOF(1)
      LINE INPUT #1, a$
      PRINT a$
      WEND
      CLOSE #1
w1:
      GOTO w1

out.top:    'no input, just print time and date
      PRINT DATE$,tIME$
      IF file=1 THEN PRINT#1,DATE$,tIME$
      IF sp=2 THEN text$=STR$(20)    'build up a header
      IF sp=3 THEN text$=STR$(16)
      IF sp=4 THEN text$=STR$(12)
      text$=text$+" MHz QPDM at"
      text$=text$+STR$(500/clock(sp))+"MHz"
      PRINT text$
      IF file=1 THEN PRINT #1, text$
      PRINT ram.vend$(ram.point)
      IF file=1 THEN PRINT #1, ram.vend$(ram.point)
      RETURN
out.string:
```

```
      PRINT text$
      IF file=1 THEN PRINT#1,text$
      RETURN

out.param:  'print ram parameter text index by ramp (RAM Parameter)
      PRINT
      PRINT ramp,ram.text$(ramp),ram.desc$(ramp)
      IF file=1 THEN PRINT#1,"   "
      IF file=1 THEN PRINT#1,ramp,ram.text$(ramp),ram.desc$(ramp)
      mint=0: nomt=0: maxt=0  'clear accumulators
      RETURN

out.problem:
      PRINT "***THERE MAY BE A PROBLEM IN THE ABOVE PARAMETER***"
      IF file=1 THEN PRINT#1, "****THERE MAY BE A PROBLEM IN THE ABOVE PARAME-
TER****"
      RETURN

out.values: 'formatted output
      PRINT USING st2$;text$,min,nom,max
      IF file=1 THEN PRINT#1, USING st2$;text$,min,nom,max
      RETURN

eval.qpdm:  'input is qpdmp.  output is min,nom,max all set to (same) value from
qpar  'qpar of qpdmp.  will use clock (sp) if necessary
      IF qpar(1,qpdmp)<>0 THEN GOTO eval.qpdm1
      PRINT "QPDM Parameter is unused: ";qpdmp
      STOP
eval.qpdm1:
      IF qpar(1,qpdmp)>0 THEN GOTO eval.qpdm2
      min=qpar(sp,qpdmp): nom=min: max=min      'get the para for proper Q speed
      RETURN
eval.qpdm2:
      min=clock(sp)*qpar(1,qpdmp)+qpar(sp,qpdmp)      'n clock/2 + (-) adder
      nom=min: max=min
      RETURN

accumulate:
      mint=mint+min     'add in new min
      nomt=nomt+nom
      maxt=maxt+max
      RETURN
decrement:
      mint=mint-min     'subtract out new min
      nomt=nomt-nom
      maxt=maxt-max
      RETURN
change.sign:
      min=-min
      nom=-nom
      max=-max
      RETURN

get.ras.decode:
      min=ras.decode(1)
      nom=ras.decode(2)
      max=ras.decode(3)
```

```
      RETURN
get.ras.delay:
      min=ras.delay(1)
      nom=ras.delay(2)
      max=ras.delay(3)
      RETURN
get.cas.decode:
      min=cas.decode(1)
      nom=cas.decode(2)
      max=cas.decode(3)
      RETURN
get.cas.delay:
      min=cas.delay(1)
      nom=cas.delay(2)
      max=cas.delay(3)
      RETURN
get.xfg.decode:
      min=xfg.decode(1)
      nom=xfg.decode(2)
      max=xfg.decode(3)
      RETURN
get.xfg.delay:
      min=xfg.delay(1)
      nom=xfg.delay(2)
      max=xfg.delay(3)
      RETURN
get.we.decode:
      min=we.decode(1)
      nom=we.decode(2)
      max=we.decode(3)
      RETURN
get.we.delay:
      min=we.delay(1)
      nom=we.delay(2)
      max=we.delay(3)
      RETURN
get.ad.delay:
      min=ad.delay(1)
      nom=ad.delay(2)
      max=ad.delay(3)
      RETURN
get.early.cas:
      min=early.cas(1)
      nom=early.cas(2)
      max=early.cas(3)
      RETURN

pos.ras:
text$="+RAS Decode":GOSUB get.ras.decode: GOSUB pos.delay
text$="+RAS Delay": GOSUB get.ras.delay: GOSUB pos.delay
RETURN

neg.ras:
text$="-RAS Decode":GOSUB get.ras.decode:GOSUB neg.delay
text$="-RAS Delay":GOSUB get.ras.delay:GOSUB neg.delay
```

```
RETURN

pos.cas:
text$="+CAS Decode":GOSUB get.cas.decode: GOSUB pos.delay
text$="+CAS Delay": GOSUB get.cas.delay: GOSUB pos.delay
RETURN

neg.cas:
text$="-CAS Decode":GOSUB get.cas.decode: GOSUB neg.delay
text$="-CAS Delay": GOSUB get.cas.delay: GOSUB neg.delay
RETURN

neg.e.cas:
text$="-Early CAS": GOSUB get.early.cas: GOSUB neg.delay
RETURN

pos.e.cas:
text$="+Early CAS": GOSUB get.early.cas: GOSUB pos.delay
RETURN

pos.xfg:
text$="+XFG Decode":GOSUB get.xfg.decode: GOSUB pos.delay
text$="+XFG Delay": GOSUB get.xfg.delay: GOSUB pos.delay
RETURN

neg.xfg:
text$="-XFG Decode":GOSUB get.xfg.decode:GOSUB neg.delay
text$="-XFG Delay":GOSUB get.xfg.delay:GOSUB neg.delay
RETURN

pos.we:
text$="+WE Decode":GOSUB get.we.decode: GOSUB pos.delay
text$="+WE Delay": GOSUB get.we.delay: GOSUB pos.delay
RETURN

neg.we:
text$="-WE Decode":GOSUB get.we.decode:GOSUB neg.delay
text$="-WE Delay":GOSUB get.we.delay:GOSUB neg.delay
RETURN

pos.adrs:
text$="+Adrs Delay": GOSUB get.ad.delay: GOSUB pos.delay
RETURN

neg.adrs:
text$="-Adrs Delay": GOSUB get.ad.delay: GOSUB neg.delay
RETURN



finish:
'write the totals line, the ram parameter line, and the margins
'print the error message if necessary
IF ram(ram.point,ramp)<>99 THEN GOTO finish1
```

```
text$="Not a Parameter for this VRAM Vendor": GOSUB out.string: RETURN
finish1:
min=ram(ram.point,ramp): nom=min: max=min: text$="VRAM": GOSUB out.values
text$="Margins:": min=mint-min :nom=nomt-nom: max=maxt-max: GOSUB out.values
IF min=>0 AND nom=>0 AND max=>0 THEN RETURN
GOSUB out.problem
RETURN

neg.finish:
'write the totals line, the ram parameter line, and the margins
'print the error message if necessary
IF ram(ram.point,ramp)<>99 THEN GOTO neg.finish1
text$="Not a Parameter for this VRAM Vendor": GOSUB out.string: RETURN
neg.finish1:
min=ram(ram.point,ramp): nom=min: max=min: text$="VRAM": GOSUB out.values
text$="Margins:": min=min-mint :nom=nom-nomt: max=max-maxt: GOSUB out.values
IF min=>0 AND nom=>0 AND max=>0 THEN RETURN
GOSUB out.problem
RETURN

totals:
text$="Total Time:": min=mint: nom=nomt: max=maxt: GOSUB out.values
RETURN

pos.qpdm:
text$="+QPDM Para "+STR$(qpdmp)
GOSUB eval.qpdm: GOSUB out.values: GOSUB accumulate: RETURN

neg.qpdm:
text$="-QPDM Para "+STR$(qpdmp)
GOSUB eval.qpdm: GOSUB change.sign
GOSUB out.values: GOSUB accumulate
RETURN

pos.delay:
SWAP min,max: GOSUB out.values: GOSUB accumulate: RETURN

neg.delay:
GOSUB change.sign: GOSUB out.values: GOSUB accumulate: RETURN
```

3.3 FONT STORAGE IN *KANJI* ROMS

3.3.1 Introduction

This note describes a method of storing very large fonts in a Am95C60 Quad Pixel Dataflow Manager (QPDM) system. This method avoids the expense and board space that would be required to keep the fonts in VRAM by allowing them to reside in (relatively slow) MOS ROMs.

3.3.2 String Operations on the QPDM

The QPDM provides a powerful text manipulation facility, described in detail in Chapter 10 of the Technical Manual. An overview is provided here.

The Font

A font is stored in display memory (typically with Input Block) and the QPDM is notified of the location of the font using the Set Character Font Base instruction. Each (there may be two) font contains up to 4096 character entries, all beginning at the same X address. See Figure 3.3-1. Each character description begins with a 16-bit attribute word which specifies the size of the character. This is followed by as many pattern words as are required to describe the character. In the font described in this section, each character is 24 pixels by 24 scan lines. With this mechanization, character are separated horizontally by up to eight blank pixels.

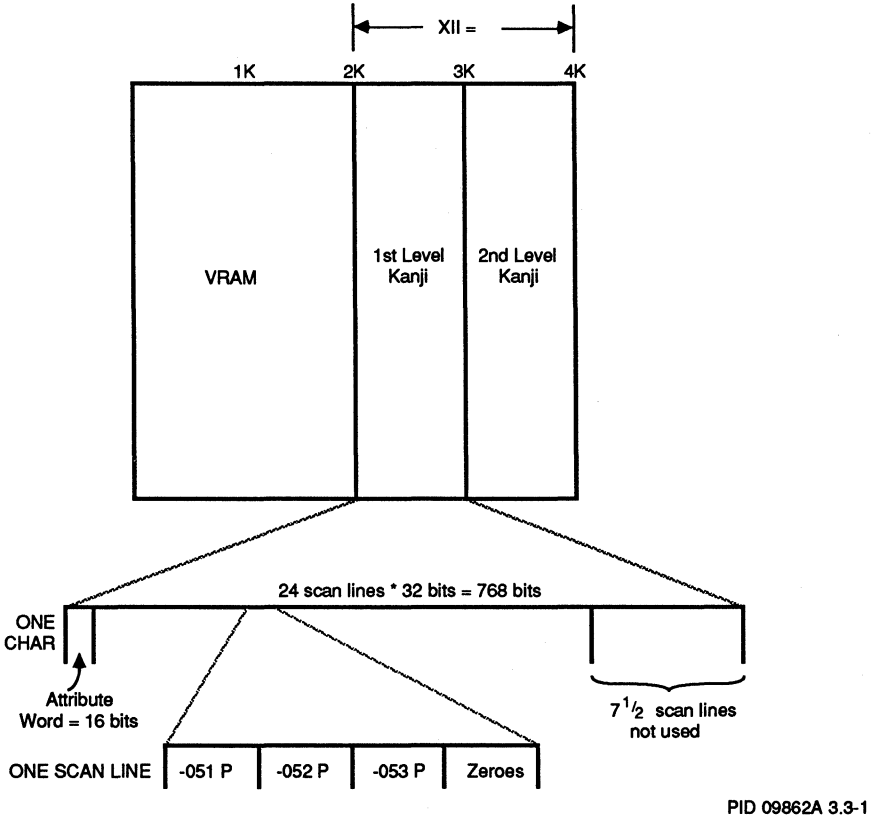
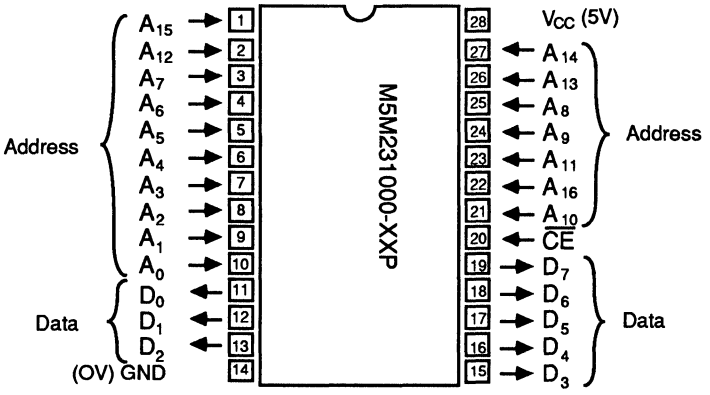
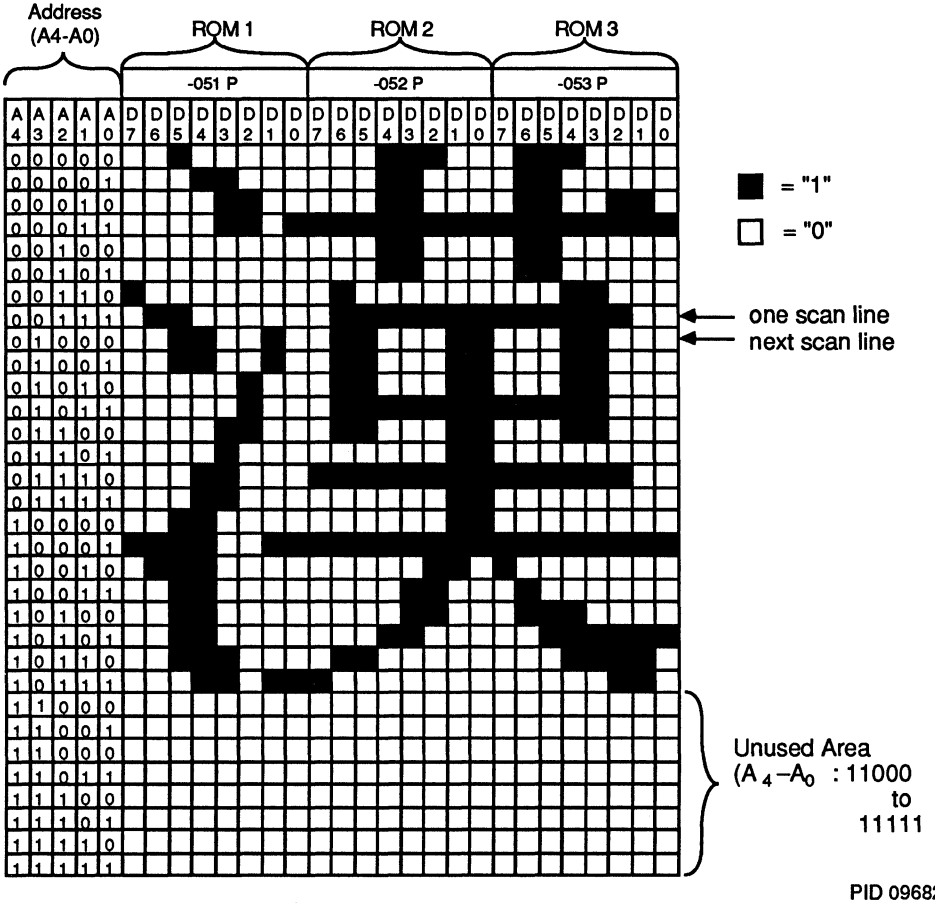


Figure 3.3-1 Display Memory Layout

KANJI ROM PINOUTS



KANJI ROM ORGANIZATION



PID 09682A 3.3-2

Figure 3.3-2 Kanji ROM Pinouts and Organization

The String Instruction

The String instruction is used to move characters from the font area to a (normally) visible area of the display memory. The String instruction specifies the beginning location where the text is to appear. The String instruction is followed by a variable-length list of character codes. Each entry in this list is processed as one character.

The QPDM processes list entries as follows: The entry is used as an index into the font. Since characters in the font are ordered vertically, the index is a Y-offset from the base of the font. The attribute word is fetched first so that the QPDM knows the size of the character.

Then, for each scan line of the character, as many 16-bit pattern words as necessary are fetched and placed into the display memory. Since we are processing 24-bit wide characters, two pattern words per scan line are required (the right-most eight bits are discarded).

The QPDM processes all the scan lines of a character (in this case there are 24) and moves the Current Pen Position to the beginning of the next character space. The high-order bit of the current list entry is tested to determine if it is the last in the list. If not, the process continues with the next entry.

3.3.3 Memory Requirements for Very Large Fonts

This method of storing fonts in the display memory presents a problem when the fonts become very large. Consider a 24 x 24 font containing 8192 entries. Each scan line of each character is stored in 32 bits (eight of which are unused). There are 768 bits for pattern storage plus 16 bits for the attribute word. Each character requires 784 bits; the complete font requires over 6 Mbits. At least in the short run (until 1M x 4 VRAMs become available), this is sufficiently expensive that we would like to find a better way.

3.3.4 The Solution for *KANJI*

Kanji fonts are available in 300 ns ROMs. There are two sets of three ROMs each; 1st level *Kanji* contains 2965 characters and 2nd level *Kanji* contains 3388 characters.

Each ROM in the three-chip set contains an 8-bit slice of each scan line of each character. The ROMs are addressed in parallel; there are 12 address bits to select a character and 5 address bits to select a scan line. Not all of the character codes are used, and only the first 24 scan lines are used. The arrangement of the three ROMs is shown in Figure 3.3-2. The development of the addresses is shown in Table 3.3-1

These ROMs have an access time (both address and Chip Select) of 300 ns, but the QPDM provides the address only 60 ns before the data are required. So there is insufficient memory access time. This problem is solved by pipelining the accesses.

Figures 3.3-3 and 3.3-4 show how this works. When the QPDM executes the display memory read to fetch the attribute word, this is detected in the PAL device since X_{11} is a one and X_9 through X_4 are all zeroes. This is defined as an attribute word. EN_ATTRIB_WORD* enables U6 and U7 onto the DM lines during CAS. This is a "hard-wired" attribute word which specifies a 24 x 24 character. The bits in the attribute word are described in Section 3.3.5.

During this same memory cycle, the character address is clocked into U2 and U3 with CLK_CHAR at the beginning of CAS. Further, the scan line number (which always begins at zero) is clocked with CLK_SCAN_LINE, also at the beginning of CAS. X_{10} is also clocked with CLK_SCAN_LINE. This is used to enable one or the other triplet of ROMs based on the level selected.

The ROMs begin to access the data for the first scan line, i.e., scan-line number zero, as the QPDM completes fetching the attribute word. Even assuming back-to-back cycles, the ROMs have somewhat over 300 ns before the data are required.

When the QPDM executes a memory-read cycle to fetch the first word of the first scan line, the address will have both X_{11} and X_4 as ones. The PAL device U5 will make EN_LEFT_WORD* active while /CAS is active, gating the upper 16 bits onto the DM lines via U8 and U9. At the falling edge of /CAS, CLK_RIGHT_WORD will clock the lower eight bits into U_{10}.

When the QPDM executes a memory-read cycle to fetch the second word of the first scan line, the address will have X_{11} and X_5 ones, and X_4 will be a zero. The PAL device will then make term EN_RIGHT_WORD* active which will enable the lower eight bits of the font onto DM_{15-8} via U10 and zeroes onto DM_{7-0} via U11. The purpose of the zeroes is to guarantee blanks between characters. At the falling edge of /CAS, X_9 through X_5 are clocked into U4 with CLK_SCAN_LINE and the ROMs will begin to access the second scan line (scan line one). Referring to Table 3.3-1, one can observe that X_4 from

Table 3.3-1 Address Bits

ROM ADDRESS BIT	QPDM SOURCE	CONFIG E	CONFIG D
A_{16}	Y_{11}	A_{10} @ CAS	A_9 @ CAS
A_{15}	Y_{10}	A_9 @ CAS	A_8 @ CAS
A_{14}	Y_9	A_8 @ CAS	A_7 @ CAS
A_{13}	Y_8	A_7 @ CAS	A_8 @ RAS
A_{12}	Y_7	A_7 @ RAS	A_7 @ RAS
A_{11}	Y_6	A_6 @ RAS	A_6 @ RAS
A_{10}	Y_5	A_5 @ RAS	A_5 @ RAS
A_9	Y_4	A_4 @ RAS	A_4 @ RAS
A_8	Y_3	A_3 @ RAS	A_3 @ RAS
A_7	Y_2	A_2 @ RAS	A_2 @ RAS
A_6	Y_1	A_1 @ RAS	A_1 @ RAS
A_5	Y_0	A_0 @ RAS	A_0 @ RAS
A_4	X_9	A_5 @ RAS	A_5 @ RAS
A_3	X_8	A_4 @ RAS	A_4 @ RAS
A_2	X_7	A_3 @ RAS	A_3 @ RAS
A_1	X_6	A_2 @ RAS	A_2 @ RAS
A_0	X_5	A_1 @ RAS	A_1 @ RAS

The purpose of this table is to indicate the detailed source of each address bit into the ROM. The column labeled "QPDM SOURCE" is the Y or X address bit internal to the QPDM. The columns labeled "CONFIG E" and "CONFIG D" indicate which address pin of the QPDM presents the address bit and whether it comes out during RAS or CAS time.

X_{11}=1 indicates the access is a Kanji font address. X_{10} selects first level or second level Kanji.

A_{16} through A_5 select a character in the ROM; A_4 through A_0 select a scan line.

the QPDM is not part of the address going into the ROMs, but is used to determine which half of the character should be fetched.

When the QPDM executes a memory read cycle to fetch the first word of the second scan line, X_4 will be a one. The PAL device makes EN_LEFT_WORD* active and the process continues.

3.3.5 Remarks

The X11 Problem

This solution requires that the QPDM emit address X_{11} which is not the case for memory configurations D and E (see Chapter 12 of Tech Manual). The third major revision of silicon (REV. C) supports these two address bits in configurations D and E.

An interim solution is diagramed in Figure 3.3-5. The QPDM is actually programmed for memory configuration C (see Section 12 of the technical manual) so that address X_{11} is available. Then, we externally put a multiplexer in front of A_7 to the VRAMs to make the addresses the same as they would have been in configuration E. The implementer must insure that no RAM timing parameters are violated. Row address set-up and hold times should be checked especially carefully.

The Form of the Attribute Word

The attribute word should be as follows:

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VALUE	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0
FIELD	H				S				D		ICO					

H is set to 1100 (decimal 12). Assuming cell scale is 2, this corresponds to 24 scan lines of active characters. If the cell scale is set to 4, this should be 0110 (decimal 6).

S is set to 0000 because we do not want any space above characters.

D is set to 00 to force left-to-right character positioning.

ICO is set to 01 1100 (decimal 28). Since the character occupies 24 pixels, this leaves four pixels between characters. These pixels are forced to zero by U11. Since eight zeroes are provided, the ICO could be as much as 32, leaving eight pixels between characters.

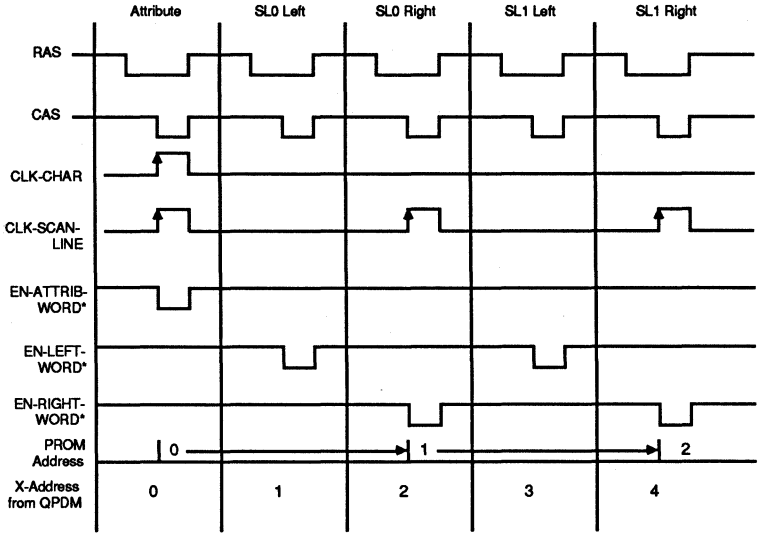


Figure 3.3-3 Timing Diagram

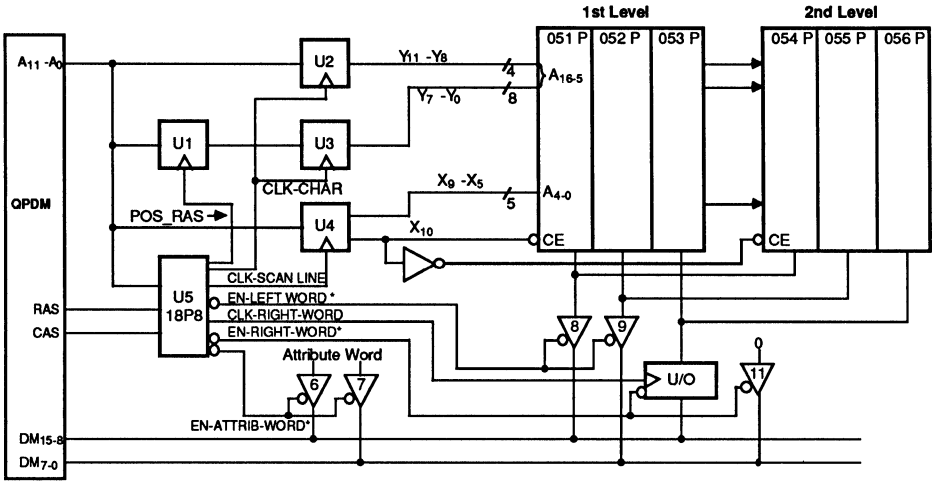


Figure 3.3-4 Logic Diagram

Intervening Memory Cycles

Memory cycles can intervene between accesses to the font memory. There will certainly be write cycles to Visible Display Memory, and there normally will be transfer cycles and refresh cycles. This is no problem so long as none of them make X_{11} a one, because the registers in the ROM logic will remain static. The QPDM will not make X_{11} a one during transfer or refresh cycles. It is up to the user to avoid using addresses that would make X_{11} a one during read or write cycles.

Preventing a Bus Crash

The implementer should use X_{11} to prevent the VRAMs from executing read cycles when the Font ROM is being accessed. Otherwise, a bus contention will result with both the VRAMs and ROMs attempting to put data onto the DM lines.

Logic Minimization

I made no attempt to minimize the logic in this note since I wanted clarity of purpose above all else. It is possible, for example, to combine U6 and U8 into a single PAL device (16H8). Another possibility is to combine U7, U9, and U11 into a single PAL device.

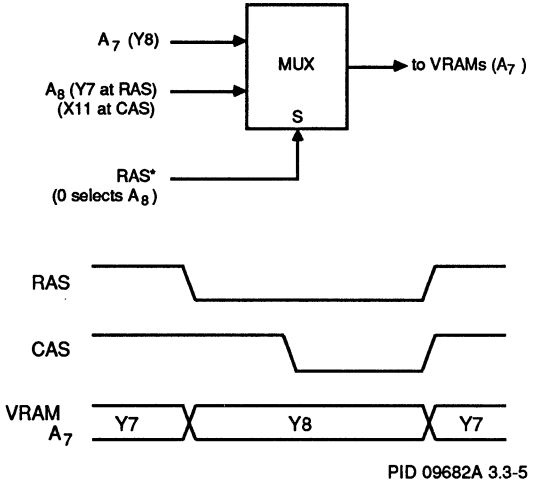


Figure 3.3-5 X_{11} Solution

Table 3.3.2 PAL Equations

```
X11, X9, X8, X7, X6, X5, X4     PIN     1,2,3,4,5,6,7;
RAS, CAS                        PIN     8,9;
POS_RAS                         PIN     19;
CLK_CHAR                        PIN     18;
CLK_SCAN_LINE                   PIN     17;
EN_LEFT_WORD                    PIN     16;
EN_RIGHT_WORD                   PIN     15;
CLK_RIGHT_WORD                  PIN     14;
EN_ATTRIB_WORD                  PIN     13;
SPARE_IO                        PIN     12;
SPARE_IN                        PIN     11;

POS_RAS             =           !RAS;
CLK_CHAR            =           X11 &!X9 & !X8 & !X7 & !X6 & !X5 & !X4 & !ICAS;

CLK_SCAN_LINE       =           X11 & !X4 & !CAS

!IEN_LEFT_WORD      =           X11 & X4 & !CAS
!IEN_RIGHT_WORD     =           X11 & X9 &!X4 & !CAS
                    #           X11 & X8 & !X4 & !CAS
                    #           X11 & X7 & !X4 & !CAS
                    #           X11 & X6 & !X4 & !CAS
                    #           X11 & X5 & !X4 & !CAS;

CLK_RIGHT_WORD      =           X11 & X4 & !CAS:

!IEN_ATTRIB_WORD    =           X11 & !X9 & !X8 & !X7 & !X6 & !X5 & !X4 & !CAS
```

Table 3.3.3 Device Type and Purpose

IC#	Type	Purpose
U1	Octal FF	Save Row Address until CAS time.
U2	Quad FF	Contains $Y_{11,10,9,8}$ ((High Order Char Adrs)
U3	Octal FF	Contains Y_7-Y_0 (Low Order Char Adrs)
U4	Hex FF	Contains X_{10}-X_5 (Select and Scan Line)
U5	PAL Device	Control
U6	Octal Buffer	Emits Left Byte of Attribute
U7	Octal Buffer	Emits Right Byte of Attribute
U8	Octal Buffer	Buffers Top Byte of Kanji Character
U9	Octal Buffer	Buffers Middle Byte of Kanji Character
U10	Octal FF w/3S	Latches, Buffers Low Byte of Kanji Character
U11	Octal Buffer	Emits eight zeros for inter-character space

The purpose of this table is to indicate the device type and purpose of each IC in this design. It really could be part of the block diagram.

CHAPTER 4

Video Bus

CHAPTER 4
Video Bus

4.0 INTRODUCTION

This section covers the Video Bus, defined as the monitor controls and the serializers. The monitor controls (HSYNC and VSYNC) are covered first. Then we present three detailed examples of the video serializers. The first serializer used in the AMD evaluation/demonstration board is suitable for video rates up to 40 MHz. The second serializer uses Am8177s and is suitable for video rates up to 125 MHz. The third serializer uses Am8172s and is suitable for video rates up to 125 MHz. These serializers have all been built and tested.

For a detailed analysis of a demonstration /evaluation board that was built and tested, please refer to Section 5.

4.1 VIDEO BUS

In this section we discuss the ways of designing the video bus portion of a QPDM design. We first talk about the monitor controls and then cover three different video serializers.

4.1.1 Monitor Controls

The three monitor controls from the QPDM (BLANK, HSYNC, and VSYNC) have substantial (greater than one-tenth dot clock) timing uncertainty and will have to be resynchronized prior to use. In general, the timing with which the SYNC signals are synchronized is not critical, so long as it is consistent (does not vary more than a fraction of a dot time from scan line to scan line). If the design provides a timing pulse that is synchronous with VIDCLK and is guaranteed to follow the positive edge by at least timing parameter 103 plus the register set-up time, there will be no problem. The SYNC pulses may be buffered (with or without inversion) and driven directly to the monitor. Alternatively, the SYNCs may go to the color palette to be mixed with the video (usually green).

The time at which BLANK must be synchronized is very much dependent on how the serializers work. We will cover this in the discussion of each of the three serializer methods.

4.2 SERIALIZERS IN GENERAL

The final serialization process is the responsibility of the system designer, not the QPDM. The QPDM will execute the transfer cycles to load the scan line into the VRAM serial port and provide the blank signal to indicate when the actual serialization should take place. The user takes care of the rest.

If the system contains Am8172 VDAF's or the equivalent, the QPDM will take care of loading the INPUT side of the FIFO but taking data out of the FIFO must be done by the user.

4.2.1 Slow-speed Serializers

An example of slow-speed serialization is covered in some detail in Section 5.1. To re-cap, the QPDM display memory is organized in 16-bit words. Thus, the display memory chips are allocated four (64K * 4) to a bit plane. We use the serial output enables on the four chips to multiplex to a 4-bit bus. This bus is loaded into a 4-bit parallel-to-serial shift register with synchronous reset. Every four bit times, the shift register is loaded. During blanking time the output of the shifter is forced to a "0" with the synchronized blanking pulse.

The longest path in this circuit is shown in Figure 4.1-1. The dot clock causes a change in the dot counter value (U_{10}), which makes VRAM output enable active (U_9). This in turn puts serial data from one of the VRAMs onto the nibble bus. This data must be set up before the fourth subsequent dot clock arrives at the serializer(U_{14}). The table below summarizes for various families of PAL devices. In each case, we assume -12 VRAMs. In practice, this should be compared to tmax for the PAL Family.

Table 4.1-1 Summary of PAL devices

Parameter	16XX Typ	16XX Max	16XXA Max	16XXB Max
U_{10} t_{CO}	17	25	15	12
U_9 t_{PD}	23	35	25	15
VRAM t_{SOA}	20	35	35	35
U_{14} t_S	20	30	15	10
Total	80	125	90	72
Total/Bit	20	31	22	18
Freq	50	32	44	56

4.2.2 High-speed Serializer without HW Windows (Am8177)

The following discussion assumes the reader has access to the data sheets for the parts that are used in the examples. These parts and their data-sheet numbers are listed below:

Am8151	Graphics Color Palette (GCP)	04653D
Am8158	Video Timing Controller (VTC)	04659C
Am8172	Video Data Assembly FIFO (VDAF)	07554A
Am8177	Video Data Serializer (VDS)	07080B

Figures 4.1-2 and 4.1-3 are schematic fragments showing a method of serializing video at up to 125 MHz. This method does not provide for a hardware window and requires that the screen be placed on a 16-bit boundary.

Figure 4.1-2 shows an Am8158 Video Timing Controller used to generate the Dot Clock and VIDCLK. Dot Clock is generated on board the Am8158 with a built-in oscillator and 5X frequency multiplier. VIDCLK is generated in the Am8158 with a divider that is programmed for divide-by-16. VIDCLK is buffered to minimize loading in the Am8158 and drives the QPDM VIDCLK, as well as being used to generate the shift clocks for the VRAMs.

DCLKOUT is buffered in three sections of a 10103 to provide two copies of the clock for the 8177 VDSs and both rails for the Am8151 GCPs. The fourth section of the 10103 is used to gate the load pulse to the serializers. Using all the sections of a single chip to generate these signals ensures that the skew will be minimized.

Putting Words into the Serializer

The Am8158 generates LD* during the last dot clock of any VIDCLK during which VC was ever LOW. The relationship amongst these signals is shown in Figure 4.1-4 DOT.CLK and VIDCLK (CCLK) run continuously. QBLANK from the QPDM goes not active during the VIDCLK period before the video will actually begin which causes LD* during the last DOT.CLK of that VIDCLK. This loads the first 16-bit word into the Am8177 serializer (this is shown as word 0 of VIDEO). Eight bit times later, SHIFT.CLK goes high to clock the next 16-bit word into the VRAM serializer outputs. Actually, this can occur almost anywhere during the word; the falling edge of VIDCLK is a convenient time. Then, during the very last DOT.CLK of the first word, LD* goes active again to clock the second word into the Am8177s. This continues until the very last word has been serialized.

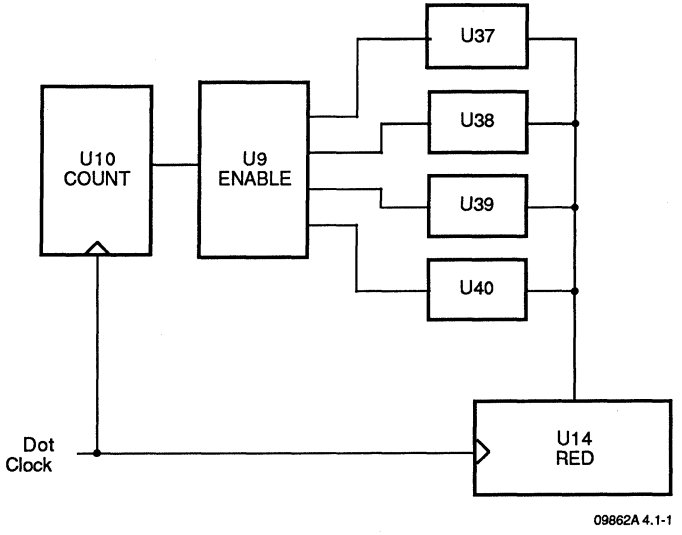


Figure 4.1-1 Slow Serializer

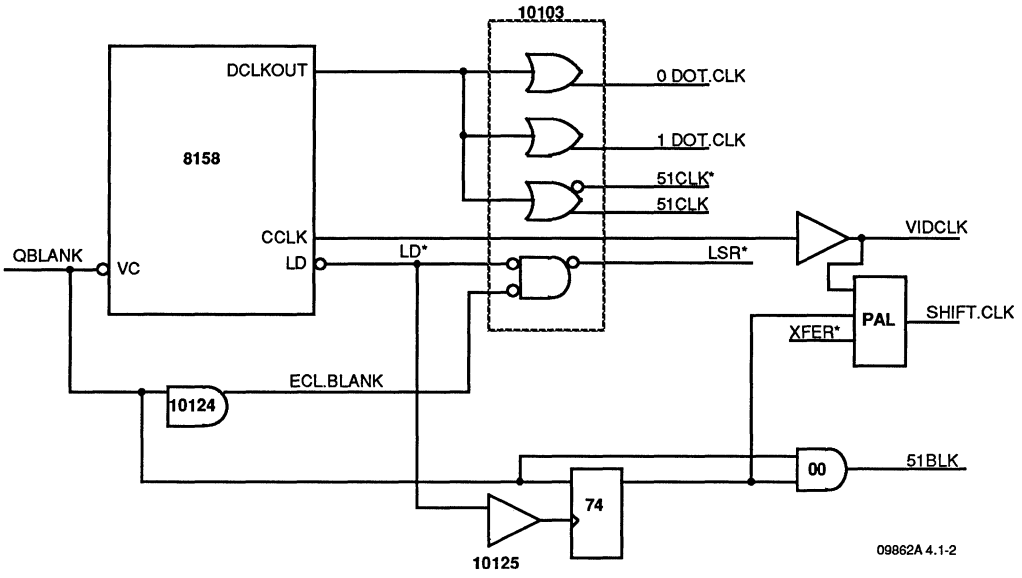


Figure 4.1-2 High Speed Serializer without HW Window - Controls

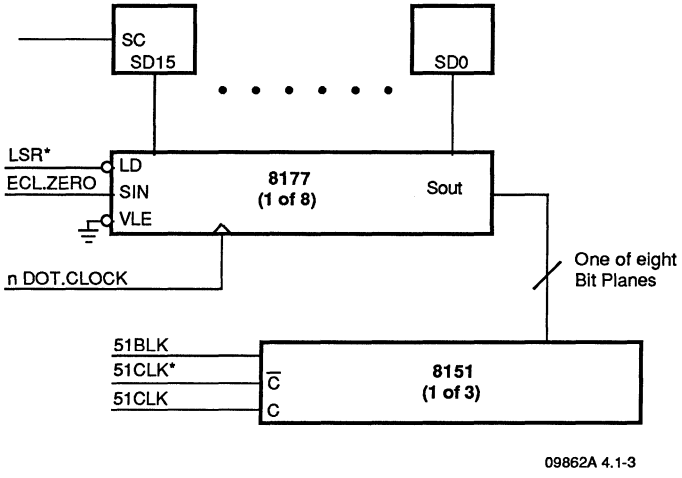


Figure 4.1-3 High Speed Serializer with HW Window - Serializer

Scan Line End Conditions

It is necessary now to examine the beginning and the end of scan lines. As it turns out, the BLANK input of the Am8151 is a TTL signal and cannot be used to start and stop the video at precisely the correct times. Rather, we guarantee that the video outputs of the Am8177 is all zeroes before and after the active portion of the scan line and require that the 0^{th} entry of the Am8151 LUT generate black video. This is accomplished by connecting the serial input of the Am8177 to an ECL zero and suppressing the load signals except during the active portion. The Am8158 suppresses the LD* pulses before the active video. The Am8158 generates one LD* pulse after the active video (because the QPDM doesn't drive QBLANK active in time); we suppress this LD* pulse with ECL.BLANK in the fourth section of the Am101013. Note that ECL.BLANK doesn't need to have very precise timing.

To guarantee that the monitor will accomplish DC restoration correctly, we must drive the video to blank (rather than black) during HSYNC. QBLANK is ANDed with a delayed QBLANK before going to the Am8151. This is shown as 51BLK in Figure 4.1-4 51BLK goes not active before the beginning of the scan line but this doesn't cause any problem because the video is still black. 51BLK does not go active until slightly after the end of the scan line but this doesn't cause any problem because the video is already black (because the Am8177s ran out of data to serialize).

VRAM Serial Shifter Control

We must provide a shift pulse to the VRAMs after the transfer cycle and before the first LSR*. This clocks the first word to be serialized into the VRAM serial outputs. This is done in the PAL device in Figure 4.1-2 which monitors XFER* to generate the initial pulse. It then generates an edge for every VIDCLK after the active portion of the scan line has begun.

4.2.3 High-speed Serializer with HW Window (Am8172)

Figures 4.1-5 and 4.1-6 are schematic fragments showing a method of serializing QPDM video with a hardware window. This uses the Am8172 Video Data Assembly/

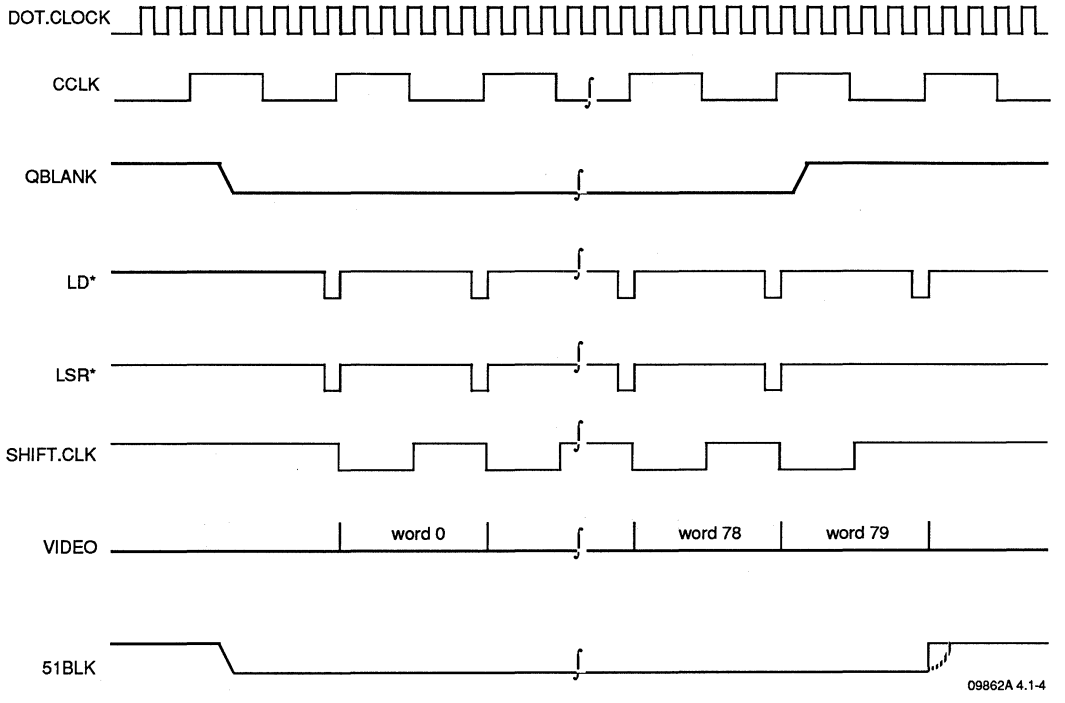


Figure 4.1-4 High Speed Serializer without HW Window - Timing Diagram

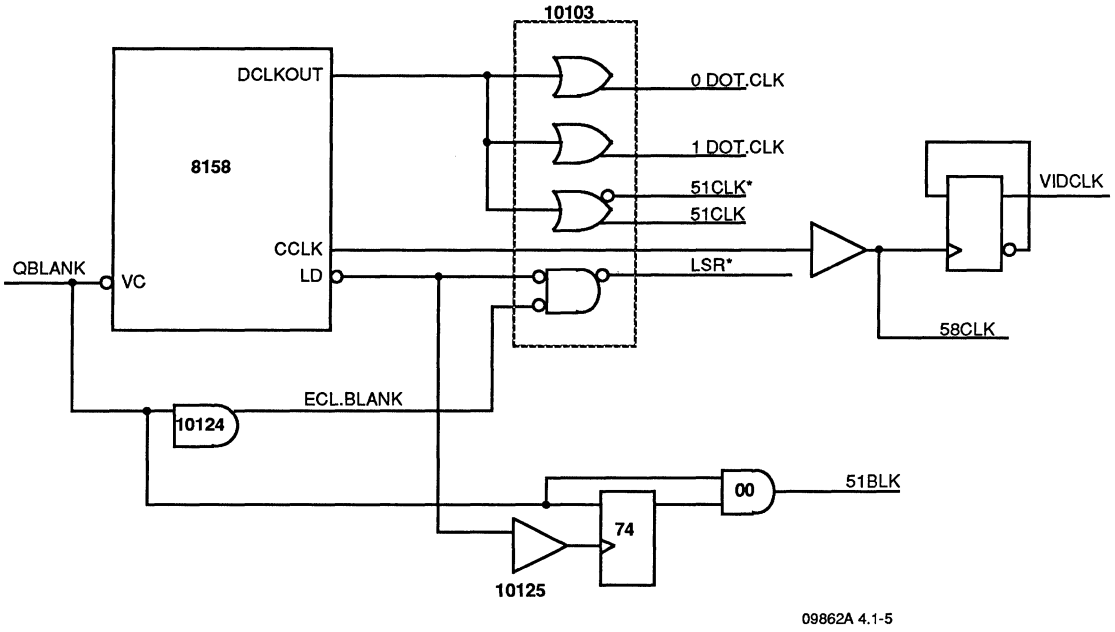


Figure 4.1-5 High Speed Serializer with HW Window - Control

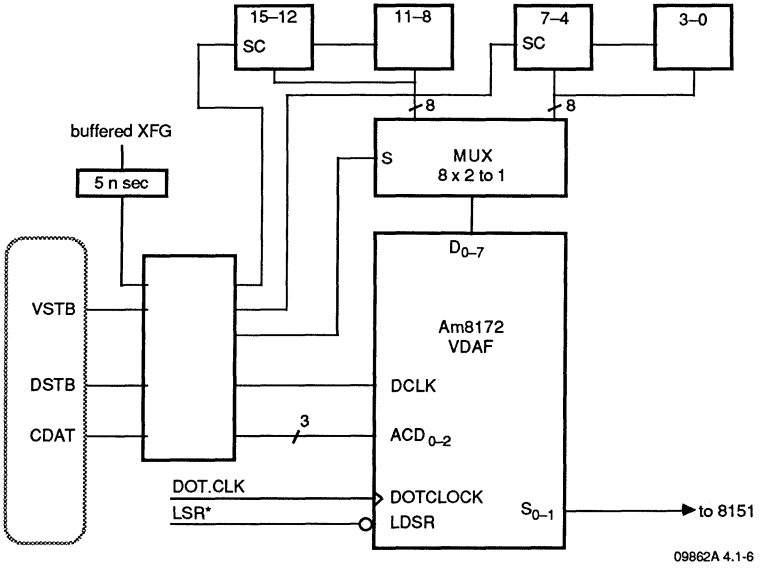


Figure 4.1-6 High Speed Serializer with HW Window - Serializer

FIFO (VDAF). The VDAF provides two functions. The first function is that of a "rubber- band" to provide video when the VRAMs are executing transfer cycles. The second function is to discard unused bits at the edges of windows.

Figure 4.1-5 shows the Am8158 and associated logic. This is similar to the Am8158 schematic shown in Figure 4.1-2 except that we need to load the final serializer every eight pixels rather than every 16. This is done by programming the CCLK divider for eight rather than for 16. In addition, we divide this clock by 2 to generate VIDCLK (to keep VIDCLK below 15 MHz).

Figure 4.1-6 shows the VRAM serializers, the Am8172, and the control logic. We consider first the removal of data from the VDAF and then the loading of the VDAF.

Removing Serial Data from the VDAF

Figure 4.1-7 is a timing diagram showing how the serialization controls are used to drive the Am8172. Observe that the horizontal scale for this diagram is by byte whereas for Figure 4.1-4 it is by word. CCLK is generated in the Am8158 and divided by two to generate VIDCLK. When QBLANK goes not active the first LD* pulse for the scan line is generated which in turn generates the first LSR* pulse to the Am8172s. 51BLK will have already gone not active but the pixels before active video will be black.

Scan Line End Conditions

At the end of the scan line, the final LD* pulse has to be suppressed; this is done by ANDing with ECL.BLANK. As in the case of using the Am8177, the timing of ECL.BLANK is not critical. The serial video from the Am8172s goes to a set of three Am8151 color palettes and thence on to a monitor.

Putting Data into the VDAF

Figure 4.1-8 is a timing diagram showing the video bytes being loaded into the Am8172 VDAF. For purposes of timing analysis, it is easiest to use DSTB from the QPDM as the reference. VSTB from the QPDM changes between 0 and 10 ns following each positive edge of DSTB (this is QPDM parameter 80). CDAT has 8 ns setup (QPDM parameter 81) and 15 ns hold (QPDM parameter 82) from each positive edge of DSTB. All the signals which go between the QPDM and the VDAF pass through a common 22V10 PAL device. Using this common device, even for signals which have no logical requirement, guarantees that the skew will be minimized.

We will generate ACD0-2 directly from CDAT0-2 and DCLK will come from DSTB. The multiplexer select will come from VSTB. The VRAM serial clocks will be generated from DSTB immediately after the data has been clocked into the VDAF. The two clocks (CLK.HI and CLK.LO) will be generated out of phase.

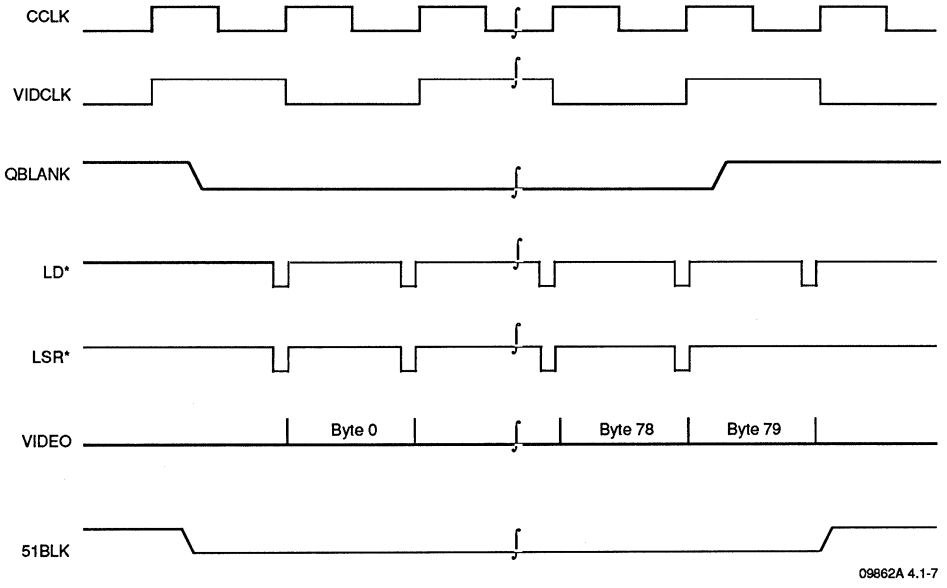


Figure 4.1-7 High Speed Serializer with HW Window - Timing Diagram

ACD Setup and Hold Times

The VDAF has a 5 ns setup from ACD before DCLK can rise. The QPDM provides 8 ns setup from CDAT to DSTB. We get a timing margin of 3 ns.

Parameter	min	typ	max
QPDM Para 81	8	8	8
+ PAL (DSTB - DCLK)	5	10	15
- PAL (CDAT - ACD)	5	10	15
Totals	8	8	8
Required (Para 7)	5	5	5
Margins	3	3	3

The VDAF has a 10 ns hold time after DCLK rises before ACD can change. The QPDM provides 15 ns hold time.

Parameter	min	typ	max
QPDM Para 82	15	15	15
- PAL (DSTB - DCLK)	5	10	15
+ PAL (CDAT - ACD)	5	10	15
Totals	15	15	15
Required (Para 8)	10	10	10
Margins	5	5	5

Data Setup and Hold Times

The VDAF requires that the data (from the VRAM serializers) be valid 5 ns before DCLK can rise. This setup is interesting because it actually begins a full DSTB early. In the worst case (the PAL device is very fast and the multiplexer is very slow), the margins are 22 ns.

Parameter	min	typ	max
DSTB Period	50	50	50
- QPDM Para 80	10	5	0
- PAL (VSTB - mux)	5	10	15
- Mux (S to Q)	21	14	7
+ PAL (DSTB1 - DCLK)	5	10	15
Totals	24	41	58
Required (Para 5)	2	2	2
Margins	22	39	56

The VDAF requires that the data be held on D7-0 for 5 ns after DCLK has risen. In the worst case (the PAL device is very slow, delaying DCLK), the timing margin is 2 ns. This assumes an extremely fast multiplexer as well.

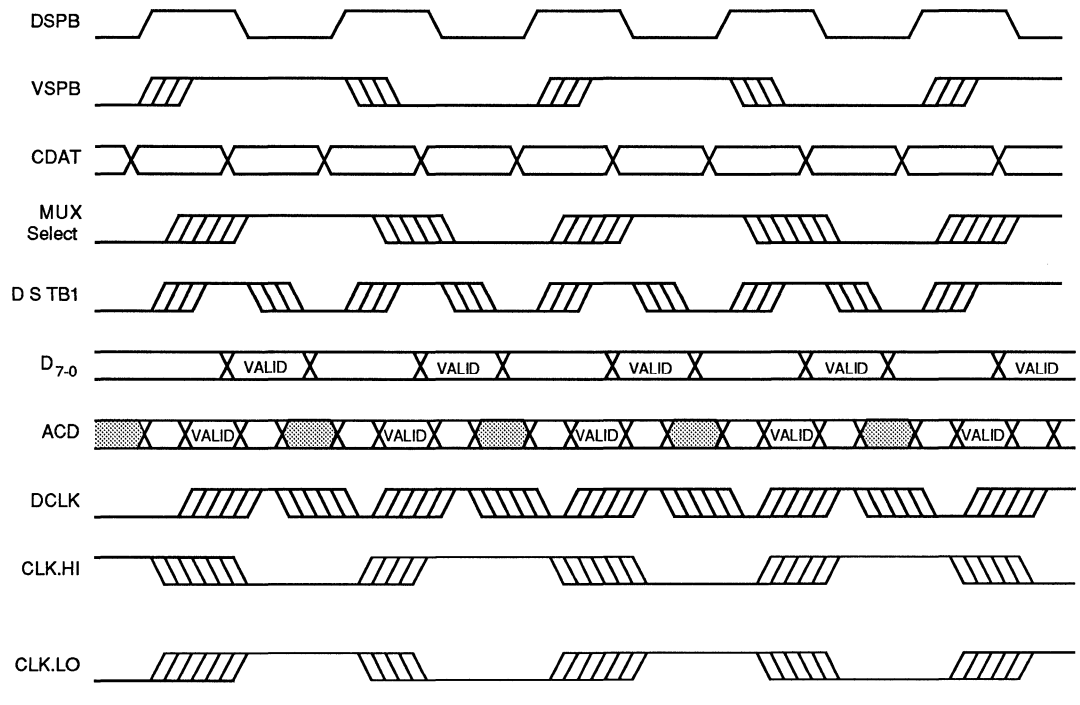


Figure 4.1-8 VDAF Timing Diagram

Parameter	min	typ	max
QPDM Para 80	10	5	0
+ PAL (VSTB - mux)	5	10	15
+ MUX (S to Q)	21	14	7
- PAL (DSTB - DCLK)	5	10	15
Totals	31	19	7
Required (Para 6)	5	5	5
Margins	26	14	2

SBCLK Generation

When the transfer cycle is executed to move the scan line or partial scan line into the VRAM serial registers, the QPDM places data on the CDAT lines to identify the first bit to be serialized. The interface has to recognize this is taking place and generate a positive edge on SBCLK at the correct time.

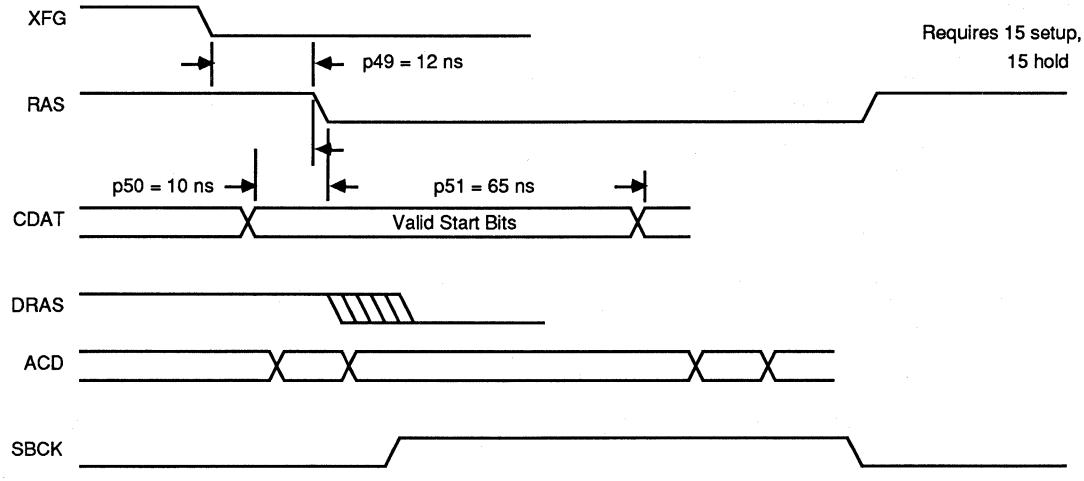
Figure 4.1-9 is a timing diagram that shows when this clock is generated. Two inputs of a PAL device monitor XFG and RAS to determine when a transfer cycle is taking place. This is the case when XFG is low and RAS is high (that is, a transfer cycle is about to begin). This is fed back and latched until RAS goes inactive.

The VDAF requires 15 ns setup time from ACD to the positive edge of SBCK; the QPDM provides only 10 from CDAT valid to RAS falls. We solve this problem by throwing another PAL output at it. That is, we delay RAS once through the PAL and use the result to time SBCK.

TELSC (That Extra Little Shift Clock)

We have saved the best for last. We must provide one shift pulse to the VRAMs after the transfer cycle and before the first data bits are clocked into the VDAF. Figure 4.1-10 shows the timing required to accomplish this.

The bounds on the time at which the edge can occur are; The edge cannot occur too soon after the rising edge of XF-G (at the VRAM) or it will violate VRAM parameter 48.



XFER = !XFG + RAS
# XFER + ! RAS
DRAS = RAS
SBCK = XFER +! DRAS

09862A 4.1-9

Figure 4.1-9 SBCLK Generation

It must occur soon enough to insure the data is available and through the multiplexer to meet the VRAM set-up time to the first DCLK.

The trailing edge of XFG is probably the best edge from which to generate the extra shift clock. It occurs a little earlier than the shift clock should and it is the edge from which the critical timing parameter is measured. XFG at the VRAM should be used; this eliminates timing uncertainties through the buffer. If this goes directly into the PAL device, there is a possibility of the SCLK coming too early (the PAL minimum delay is 5 ns and we require 10 ns). So a 5 ns active delay line must be inserted.

Now we consider whether the data will be available at the VDAF inputs in time. As before, we have a 5 ns setup time. Beginning with XFG rising edge, we have:

Parameter	min	typ	max
QPDM Para 34	14	14	14
+ QPDM Para 55	40	40	40
+ PAL (DSTB - DCLK)	5	10	15
- XFG Buffer Delay	15	12	10
- XFG - XFG_D Delay	5	5	5
- XFG_D - SCLK Delay	5	10	15
- SC Access Time	10	10	10
- Mux D-Q	5	10	14
Total	19	17	15
Required (Para 5)	5	5	5
Margins	14	12	10

For Rev.C and later, this is unnecessary because the QPDM itself will generate the pulse.

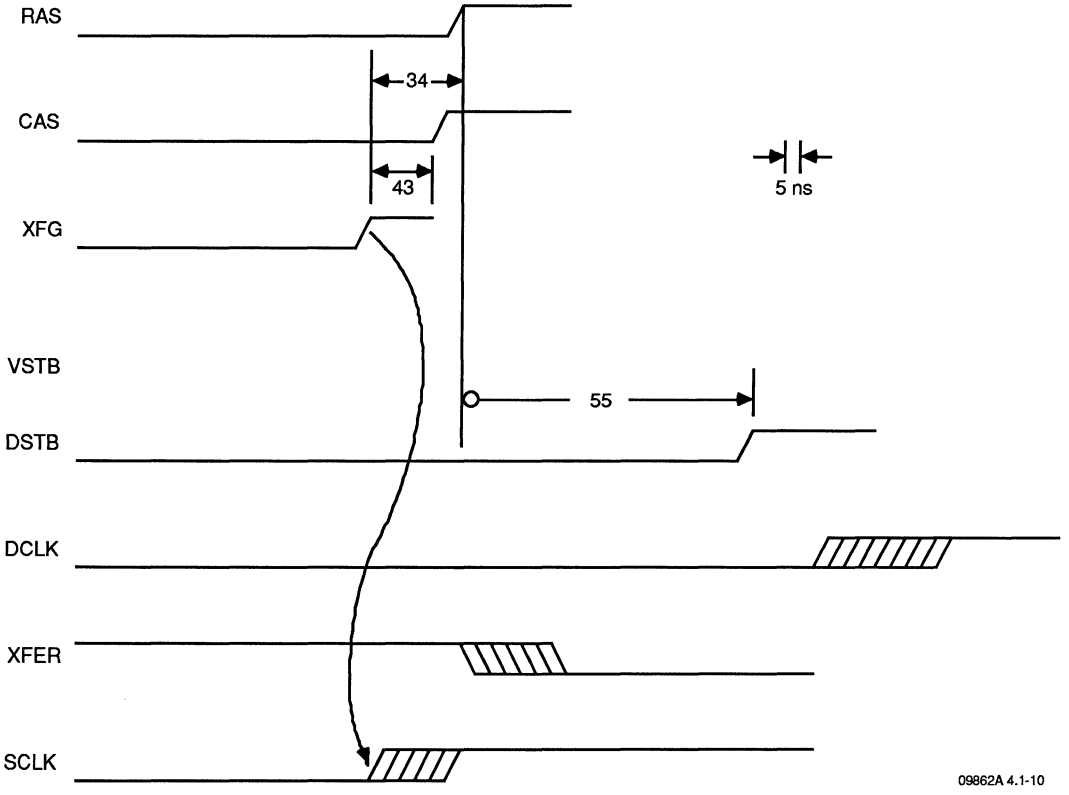


Figure 4.1-10 Extra Shift Clock Timing

CHAPTER 5

Evaluation and Demonstration Board

CHAPTER 5

Evaluation and Demonstration Board

5.0 EVALUATION AND DEMONSTRATION BOARD

In this chapter we describe an evaluation/demonstration board designed and built by AMD. This board has been put into pilot production; the design has been thoroughly tested.

5.1 PC INTERFACE

5.1.1 Address Buffers

We need to buffer 20 address bits onto the board. Each Am29C827 contains 10 bits per device. The two buffers needed are U46 and U47. U47 is shown on Sheet 2 of the schematic diagram; U46 is shown on Sheet 3. Because 29C827s suffered supply shortage in early 1987, some boards may be populated with 29827s. These consume somewhat more power than the CMOS version but otherwise present no problem.

5.1.2 Address Decoder

Address decoding is done in U36, an AmPAL22V10. We must monitor 18 address lines (BA19-BA2) and generate four address match terms. The address match terms are: PROM*, QPDM*, LUT*, and AUX*. This PAL device must be replaced in order to move any function within the address space. The equations for this PAL device are in Section 5.9.1. The PAL device is shown on Sheet 2 of the schematic diagram.

5.1.3 System Bus Control Decoders

The control decoder for the system bus interface is split into two devices, both AmPAL22V10s. BCONT, reference designator U29, controls the data buffers and PROM enables. IOCONT, reference designator U21, controls the I/O devices. These two devices take the four address match terms from U36, and eight command lines and a single address line from the system bus. The outputs are the controls for the three data buffers, RD* and WR* for the QPDM, two control lines for the Am8159, separate enables for the two PROMs, and a control line for the AUX register. The equations for these PAL devices are in Sections 5.9.2 and 5.9.3. U21 is shown on Sheet 2 of the schematic diagram and U29 is shown on Sheet 3.

5.1.4 Data Buffer

Three 8-bit bidirectional buffers are used to get data on to and off of the board. The table below shows the reference designator and bus assignments for the devices:

Device	Enable Term	System Bus	Internal Bus
U_{45}	E_HI_BUF*	$SD_{15}..SD_{08}$	$IDB_{15}..IDB_{8}$
U_{48}	E_LO_BUF*	$SD_{7}..SD_{0}$	$IDB_{7}..IDB_{0}$
U_{34}	E_SW_BUF*	$SD_{7}..SD_{0}$	$IDB_{15}..IDB_{8}$

These buffers are shown on Sheet 3 of the schematic diagram.

When the board is plugged into an 8-bit backplane (PC or XT), buffer U45 is never used. Buffer U48 transfers all data onto and off of the board and U34 is used to transfer bytes from the left EPROM (U6) to the low-order data bus.

When the board is plugged into a 16-bit backplane (AT), all transfers to and from the QPDM and the PROMs take place in 16-bit mode. U48 transfers the low bytes and U45 transfers the high bytes. The software must be compiled differently for each case.

5.1.5 SYSCLK Generator

Y_2 is a standard crystal oscillator with a TTL output. To insure that the SYSCLK input to the Am95C60 is as nearly symmetrical as possible, the oscillator operates at twice the desired SYSCLK frequency. This 2X signal is divided by two in a 74F74, at reference designator U17. The output of the F74 drives only the SYSCLK input and is terminated to minimize undershoot. The terminator is physically placed at the end of the trace farthest from the F_{74}. Y_2 and U17 are shown on Sheet 4 of the schematic diagram.

For a 20 MHz operation, a 40 MHz oscillator is chosen. When the board is set up for a 16 MHz operation, a 32 MHz oscillator is chosen.

5.1.6 INT Jumpers

The INT output of the Am95C60 is active HIGH. It is buffered in a 74S244 (reference designator U4) to make a term called BINT. This can be connected to any of three interrupt input pins at jumper block W4. The interrupt inputs which may be chosen are INT_2, INT_3, and INT_5. These are the interrupts that are least likely to be used in a standard PC. The software available from AMD that uses interrupts is configured for INT_5. U4 is shown on Sheet 8 of the schematic diagram.

5.1.7 System Bus Cycles with QPDM

In general, the timing for system bus cycles with the QPDM all comes from the PC bus timing. This is due to the generous margins provided in the PC.

CPU Read Cycle from QPDM

Figure 5.1-1 shows the timing involved in a CPU read cycle. The cycle begins when the address settles sometime during clock T_1. This makes term QPDM* (which is the QPDM chip select) active. Near the beginning of T_2, IOR* on the bus becomes active. PAL device U_{21} generates QPDMRD* to the QPDM and makes the buffer control terms active. Within 100 ns, the read data are valid at the QPDM and within another 20 ns, are valid on the bus. The data are not required on the bus until just before the beginning of T_4. At the beginning of T_4, IOR* goes inactive, making QPDMRD* at the QPDM, as well as the buffer control terms, inactive. At the end of T_4, the address changes, making CS* at the QPDM inactive. This completes the cycle.

CPU Write Cycle to QPDM

The timing for a CPU Write Cycle is shown in Figure 5.1-2. The cycle begins when the address becomes valid late in T_1. This generates QPDM*, which is the QPDM Chip Select. During T_2, the term IOW* on the bus goes active, which makes QPDMWR* to the QPDM and the buffer control terms active. The data on the bus is valid later in T_2. IOW* on the bus goes inactive at the end of T_w, which allows QPDMWR* and the buffer control terms to go inactive. The cycle completes at the end of T_4, when the address changes, making QPDM* (CS* to the QPDM) inactive.

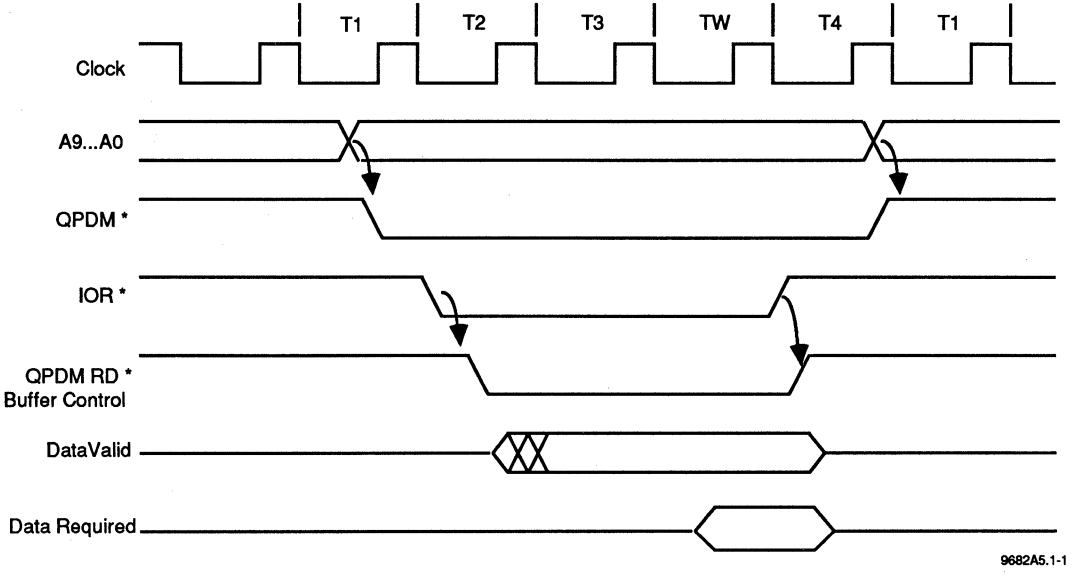


Figure 5.1-1 CPU Read Cycle

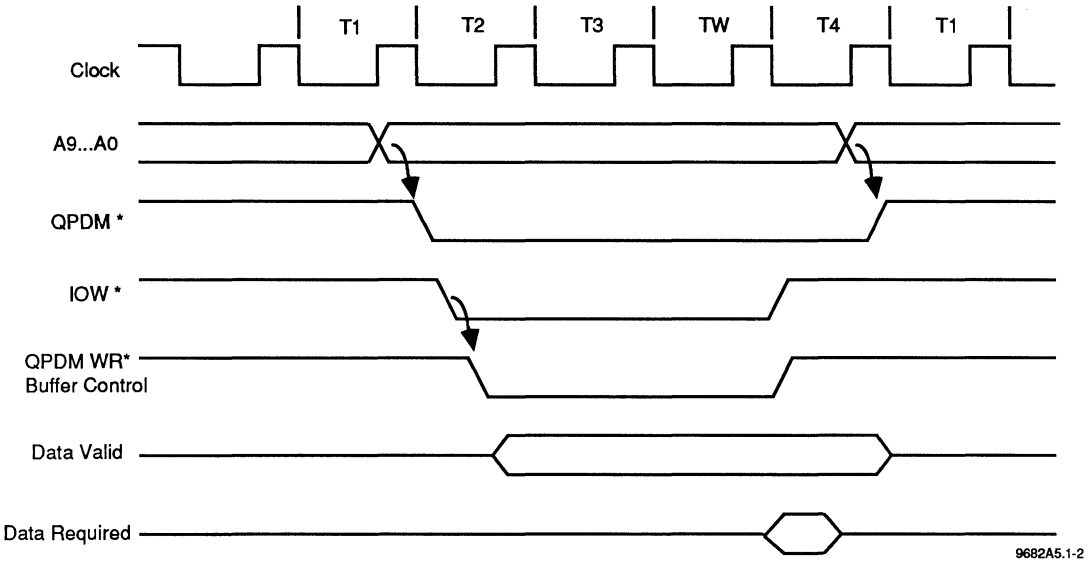


Figure 5.1-2 CPU Write Cycle

5.1.8 DMA Modifications

After this board was put into production, AMD decided that 16-bit DMA transfer to the instruction FIFO was necessary. This would make it possible to drive the QPDM to saturation with a 286-class processor.

The modification is shown in Figure 5.1-3. A DMA Acknowledge "fakes" a write to the instruction FIFO. This is done by forcing QPDMCS*, AEN* and ALE*. In addition, both address inputs to the QPDM are forced low.

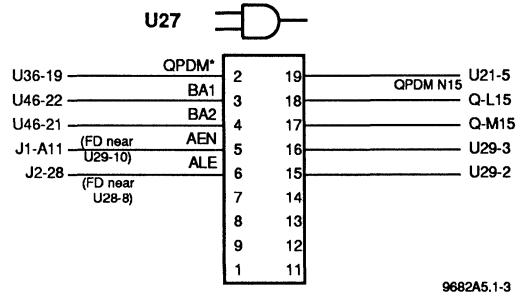


Figure 5.1-3

5.2 DISPLAY MEMORY INTERFACE

The display memory interface is tailored to allow 20 MHZ QPDM operation with 120 ns VRAMs. This involves careful buffering and generating /CAS as early as possible.

The display memory is implemented with 64K x 4 VRAMs. Four devices per bit plane (total of 16 on the board) allow for a bit map of 1K x 1K. Actually, the bit map could be configured as 256 X 4K, 512 x 2K or 1K x 1K by reprogramming the Memory Mode Register (Register 23). The software that comes with the board uses only the 1K x 1K configuration.

5.2.1 Address Buffer

The display memory addresses are buffered from AD_0-AD_7. The buffer used here is an Am2966 (reference designator U_{16} shown on Sheet 5 of the schematic diagram). Since 16 devices are driven with a maximum of 5 pF input capacitance each, we estimate a maximum of 90 pF loading on this 2966. Using the chart in the 2966 data sheet, we estimate the delay through this buffer to be between 10 and 17 ns.

Since the edge rates of the 2966 are relatively slow, due to the internal series resistors, we would not expect any significant undershoot or ringing on these lines.

5.2.2 Write Enable Buffer

The four write enables from the QPDM are buffered in half of an Am2966 (reference designator U5 shown on Sheet 5 of the schematic diagram). Each output drives four VRAMs with a maximum of 5 pF input capacitance each, for a total of 20 pF. Using the chart in the 2966 data sheet, we estimate the delay through this buffer to be between 6 and 11 ns.

5.2.3 XFG Buffer

XF/G from the QPDM drives four inputs of an Am2966 (reference designator U2 shown on Sheet 5 of the schematic diagram). Each of the four outputs of this device drives four VRAM inputs. Each VRAM input has 5 pF input capacitance so that each 2966 is driving 20 pF. From the chart in the 2966 data sheet, we estimate the delay through this buffer will be between 6 and 11 ns.

5.2.4 RAS Buffer

RAS from the QPDM drives four inputs of an Am2966 (reference designator U2 shown on Sheet 5 of the schematic diagram). Each of the four outputs of this device drives four VRAM inputs. Each VRAM input has 7 pF input capacitance so that each 2966 output is driving 28 pF. From the chart in the 2966 data sheet, we estimate the delay through this buffer to be between 7 and 12 ns.

Since the XFG and RAS buffers reside on the same chip, we expect the delays to track. That is, if the XFG buffers are especially slow (due to temperature, VCC, or processing) we expect that the RAS buffers to be also slow.

5.2.5 CAS PAL Device

The equations for the PAL device that generates CAS are given in Section 5.9.4. This is an Am18P8B, reference designator U3. This device is shown on Sheet 8 of the schematic diagram.

A signal called !XFER is generated for use inside the PAL device. This term is active during any transfer cycle from the time XF/G falls until RAS rises. The first min-term detects that XF/G has fallen before RAS (which happens only at the beginning of a transfer cycle) and the second min-term serves to latch the signal until RAS rises at the end of the cycle.

Four separate but identical /CAS terms are generated, one for each four VRAMs. This duplication keeps loading below 50 pF so that the PAL device timing parameters are guaranteed. This also minimizes the length of trace necessary to help reduce undershoot.

There are three min-terms in the CAS equations. The first makes CASn whenever CAS is active and XF/G is not. This occurs during write and refresh cycles. The second min-term makes CASn whenever Delayed XF/G (DXFG) is active and XFER is inactive. This occurs only during read cycles. The purpose is to generate CAS as early as possible during read cycles. We delay XF/G just long enough to guarantee that the Column Address Set-up Time will be met. The third min-term for CASn is whenever CAS is active and XFER is active. This is the case during a transfer cycle.

Delayed Transfer (DLYFER*) is used to force the extra clock pulse required by the VRAMs before serialization begins. This is generated at the very end of the transfer cycle by passing XFER back through the PAL device.

Two additional terms that have absolutely nothing to do with the display memory interface are generated in this device. Active LOW RESET* is generated by inverting RESET from the backplane. A high-frequency filter has been added to this output to minimize noise on the QPDM Reset line. Synchronized Composite Sync (SCS) is generated by combining Synchronized Vertical Sync (SVS) with Synchronized Horizontal Sync (SHS). This is done with an exclusive-OR function.

5.3 TIMING GENERATOR

The timing generator emphasizes simplicity and clearness of thought. Using standard (40 ns) PAL devices, it supports a dot clock of up to 25 MHz. Using -A (25 ns) PAL devices, it will operate at up to 40 MHz.

5.3.1 The Oscillator and Buffers

The Dot Clock oscillator is a standard TTL crystal oscillator, reference designator Y1. In the standard QPDM board configured for the NEC MultiSync (tm) or equivalent, this is a 24 MHz oscillator. It is buffered in four pieces of 74S244 (reference designator U4) making the terms DCLK0-DCLK3. The purpose of the careful clock distribution is to avoid the problems that result from not being careful about clock distribution. The oscillator and buffer are shown on Sheet 8 of the schematic diagram.

5.3.2 The COUNT PAL Devices

This AM16R8 is used to generate the basic timing for the serializers, reference designator U10. The equations for this PAL device are shown in Section 5.9.5. The device is shown on Sheet 2 of the schematic diagram.

Terms Q_0, Q_1, and VIDCLK form a divide-by-8 binary counter that changes state on the positive edge of DCLK. These terms are decoded both internally and in the ENABLE PAL device to allow timing at any required dot clock within a byte. In addition, VIDCLK directly drives VIDCLK of the Am95C60. CFF1* is used to clock an external flip-flop that extends the divide-by-8 to divide-by-16. Basically, it keeps track of which byte of the current word we are serializing.

LSR* is generated once every four dot clocks and is used to load a nibble into the final 4-bit serializer shift registers. ECBLNK* resynchronizes Am95C60 signals BLANK, HSYNC, and VSYNC to a specific dot clock in U18. This is necessary because these signals have substantial timing uncertainty at the QPDM pins.

The timing relationships amongst these signals are shown in Figure 5.1-4. Since this is a registered device, the outputs actually become active during the clock cycle following the one during which the inputs satisfied the equations.

A completely unrelated function of this PAL device is to delay SBLK a single dot clock for use in the Am8159. This delay is necessary to compensate for the delay of SBLK through the serializers.

5.4 SERIALIZERS

The Am95C60 requires that display memory be organized into 16-bit words (for each bit plane). This in turn requires that the serializers be organized to handle 16-bit words. We will discuss only one bit plane; the other three operate identically.

On this board, it was convenient to mechanize the serialization as a 2-step process. First, the 16-bit words are brought out onto a 4-bit bus using the serial output enables of the four VRAMs. Then the contents of the 4-bit bus are serialized in a registered PAL device.

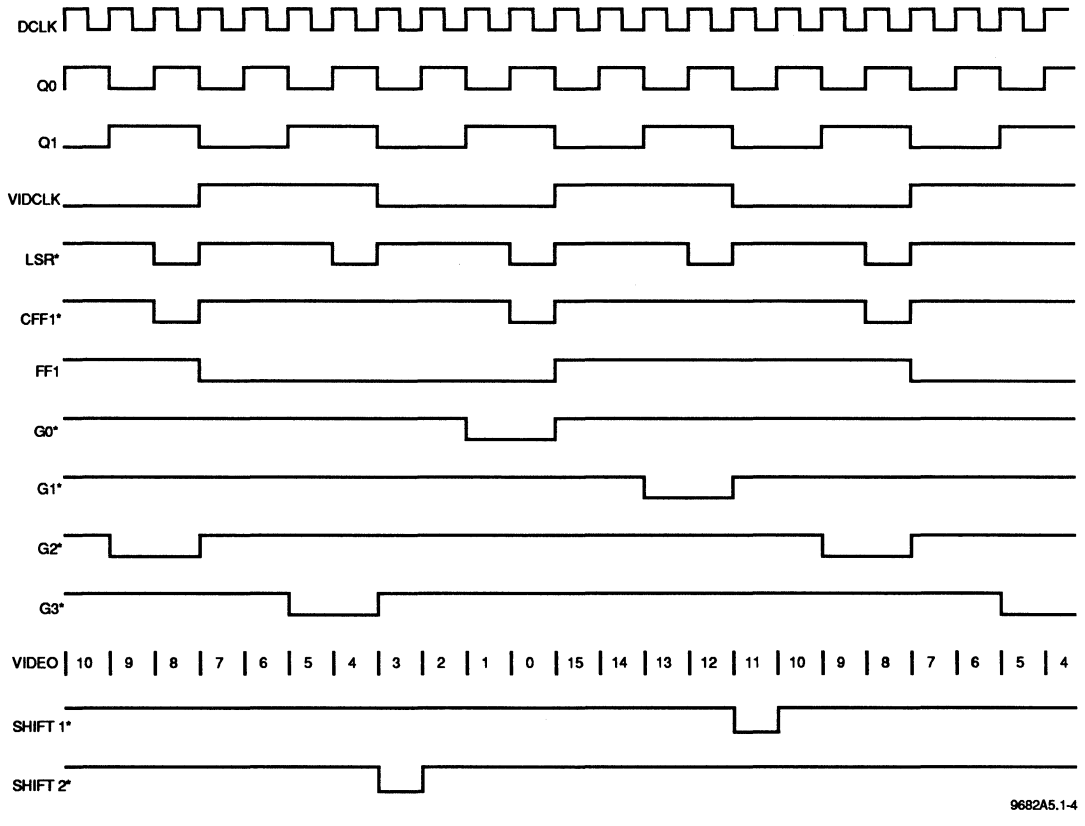


Figure 5.1-4 Serializers

5.4.1 The Enable AMD VCLK PAL Device

PAL device ENABLE is used to handle the first level of serialization, reference designator U9. The equations for this device are given in Section 5.9.6. This device is shown on Sheet 2 of the schematic diagram. Figure 5.1-4 shows the timing relationships amongst the signals into and out of this device.

Output enables G0* through G3 are generated by tracking the VIDCLK and FF1 inputs. The table below indicates which bits are enabled by each of these terms, as well as which VRAMs are enabled.

Enable	Bits	Red	Green	Blue	Intensity
G0*	15-12	U40	U44	U25	U33
G1*	11-8	U39	U43	U24	U32
G2*	7-4	U38	U42	U23	U31
G3*	3-0	U37	U41	U22	U30

Q_1 from COUNT is used to qualify each of the four enables. This precaution avoids a bus contention which could otherwise exist on the 4-bit bus.

The shift terms to the VRAMs are also generated in the PAL device ENABLE. We generate two of these terms to allow the left-most byte of the word to be shifted, independently of the right-most byte. This made the timing somewhat easier. The second min-term is generated at the end of a transfer cycle with DLYFER. This makes the first word of data available at the serializer outputs of the VRAMS.

5.4.2 The Serializer PAL Devices

The final serialization takes place in the SHIFT PAL devices. There are four identical devices, one for each bit plane. The reference designators are U12-U15 and are shown on Sheets 6 and 7 of the schematic diagram. The equations for these devices is given in Section 5.9.7.

The shifters are loaded once every four pixel times with LSR*. Following the rising clock edge during which LSR* is active, the data on D_3 appears at the output (Q_3). If SBLK is active, then the output will be zero regardless of what is loaded. This ensures the TTL outputs are zeroes during blanking.

Figure 5.1-5 shows the timing at the beginning of each scan line (the left margin of the screen). Sometime after the rising edge of VIDCLK, the Am95C60 will make its BLANK output (QBLANK) goes inactive, signaling the beginning of a scan line. At a specific dot clock within each VIDCLK period, ECBLNK will go active allowing QBLANK (and HSYNC and VSYNC) to be sampled. This occurs so that SBLK goes active one dot clock before the first pixel is to be serialized. During the next dot clock period, LSR* is active, allowing the first four pixels to enter the shifters. At the very next positive transition of DCLK, the first pixel is serialized.

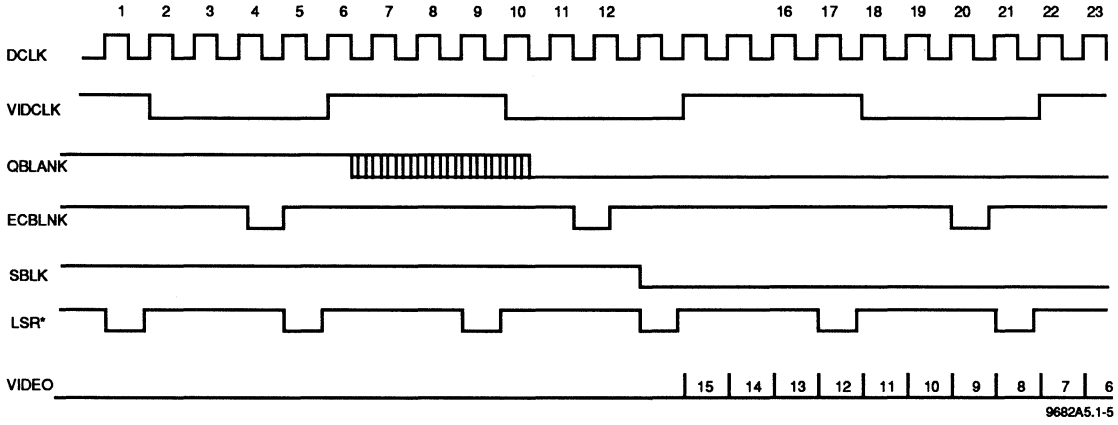


Figure 5.1-5 Left Edge Timing

5.4.3 FF1

FF1 (reference designator U11 on Sheet 2 of the schematic diagram) keeps track of whether we are currently serializing at left byte or right byte. It can be considered a high-order appendage of the counter formed by the Q_0, Q_1, and VIDCLK outputs of the PAL device COUNT.

5.4.4 SYNC Synchronizer

The monitor controls HSYNC, VSYNC, and BLANK must be resynchronized outside the Am95C60. This is because of the substantial uncertainty in their timing at the Am95C60 outputs. This takes place in the 74S379 Quad Register with Clock Enable, reference designator U18 on Sheet 5 of the schematic diagram. The clock input is the dot clock; the clock enable is ECBLNK*. This is generated once every VIDCLK cycle in the PAL device COUNT.

In addition to timing these signals precisely, the '379 also provides both the true and complement output of each of the three terms.

5.5 COLOR LOOKUP TABLE AND DACS

The REV. B board contains an Am8159 Color Palette to provide analog video. The device is shown on Sheet 8 of the schematic diagram. This device contains a 64- entry look-up table as well as three 4-bit DACs. Thus, we can display 16 colors simultaneously from a palette of 4096.

The System Address (SA0-5) and H/L inputs of the Am8159 come directly from the buffered address bus of the PC. The data inputs CD0-7 are tied to the internal data bus IDB0-7. The remaining CD pins are not connected.

The Video Address inputs are connected as indicated below:

VA Input	Source	
0	Intensify Plane Video	(Plane 3)
1	Blue Plane Video	(Plane 2)
2	Green Plane Video	(Plane 1)
3	Red Plane Video	(Plane 0)
4	Output of HILITE Oscillator	
5	Ground	

Thus, the video from the four planes select one of 16 entries in the look-up table and the HILITE oscillator selects between one of two banks of 16 entries. The remaining 32 entries in the look-up table are not used.

The BLANK input of the Am8159 is connected to DBLNK, which is SBLK delayed one bit time. This delay exactly compensates for the 1-bit time SBLK is delayed in the serializers.

HSYNC and VSYNC inputs to the Am8159 can be jumpered to the corresponding synchronized sync from the Am95C60 or can be jumpered to ground. If either or both sync inputs are connected, the corresponding (or composite) sync will appear on the green output of the DAC. If they are both connected to ground, no sync will appear on the green output.

REFOUT is connected to IREF via a nominal 1020 Ω resistor. This provides a current level that is correct for double- terminated 75 Ω video outputs. This node is heavily bypassed to ground.

The R, G, and B outputs are connected to pins 1, 2, and 3 of J2 (the analog output connector). Each is terminated in 75 Ω to ground at the connector. We expect the monitor to have a similar termination at its end of the cable.

5.5.1 DC-DC Convertor

Since the Am8159 requires a substantial amount of current at -5.2 V with respect to ground, we chose to put a dc-dc converter on board. This is shown on Sheet 8 of the schematic diagram, reference designator is U8.

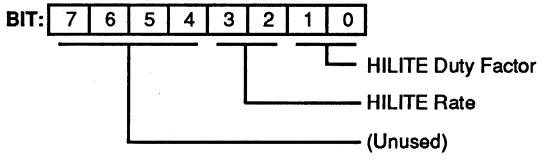
Note: Experiments with a single IBM XT indicated that it could supply adequate -5.0V to power the Am8159. There is no spec for this supply, nor is there any guarantee that some other board in the system is not taking power from the system. We chose to be conservative.

The unit is manufactured by Reliability Incorporated and is rated at 450 mA. The -5.2 V supply is decoupled at the convertor output and at both connections to the Am8159. In addition, each V_{EE} pin on the Am8159 has its own trace to the converter. This appears to provide adequate isolation between the inputs.

5.5.2 HILITE Logic

To provide a highlighting capability (and more closely utilize the functionality of the Am8159) we provided a means of switching between two 16-entry banks of the Am8159. This switching is provided by the hardware at several programmable rates and duty factors.

The control for the oscillator is an Am29845 register at reference designator U35 shown on Sheet 8 of the schematic diagram. It is cleared to all zeroes when RESET* is active. It may then be programmed as required by the host by writing to the AUX port. Bits designation in this register are indicated below:



The four low-order bits control PAL device HI at reference designator U28 (Sheet 8 of the schematic diagram). This Am22V10 is programmed as a variable rate divider that is clocked by SVS (Vertical Sync). The logic equations for this device are given in Section 5.9.8.

The divisors available are:

Bit 1	Bit 0	Divide Ratio	Nominal Rate
0	0	(OFF)	No Hilite
0	1	32	2 Hz
1	0	64	1 Hz
1	1	128	0.5 Hz

The Duty Factors available are:

Bit 3	Bit 2	Duty Factor (ON/OFF)
0	0	0/100 (No Hilite)
0	1	25/75
1	0	50/50
1	1	75/25

5.5.3 Monitor Connections

Two monitor connectors are on the board. They are both shown on Sheet 8 of the schematic diagram. J1 is for a TTL monitor and J2 is for an analog monitor. The software that comes with the board assumes an NEC Multisync (or equivalent) monitor connected to J2 (the analog connector). Each of these connectors is a female DB- 9. The pin assignments are indicated below:

Pin Number	J1(TTL)	J2(Analog)
1	Ground	Red Video
2	Ground	Green Video
3	Red Video	Blue Video
4	Green Video	Horizontal Sync
5	Blue Video	Vertical Sync
6	Intensify Video	Ground
7	(No Connect)	Ground
8	Horizontal Sync	Ground
9	Vertical Sync	Ground

5.6 EPROMS

The two EPROM sockets are U6 and U7, shown on Sheet 3 of the schematic diagram. Each socket is intended for an Am27512; U6 is the left byte and U7 is the right byte. The device enable (chip select) to both devices is made active anytime an address in the range of 0B0000-0BFF7E is on the bus. The output enables are independent. If this is an AT, then both enables go active together. If this is an IBM-XT or PC, then only a single output enable is made active, depending on the low-order address bit.

In the case of an AT, buffers U45 and U48 are both enabled to drive a 16-bit word onto the bus. In the case of a PC or XT, buffer U48 is made active. In addition, buffer U34 will be made active with U6 to drive the left byte onto the low-order bus.

AMD does not, at the time of this writing, supply any software for these sockets. The intention is for user supplied software.

5.7 MEMORY BUS TIMING ANALYSIS

The following table is a complete timing analysis of the display memory bus. This assumes a 20 MHz QPDM running at 20 MHz and -10 VRAM chips (Hitachi HM53461-1-). This file was produced using the program described in 3.2. For eachVRAM parameter, listed by number, acronym and full name, this file explains the best case, nominal and worst-worst case timing analysis. The left column assumes min delays for paths to be subtracted; max delays for paths to be added. The middle column (nominal case) uses nominal delays. The right column assumes max delays for paths to be subtracted, min delays for paths to be added. The truth is guaranteed to lie somewhere between the left and right column.

There is a remote possibility of a timing problem with the Address Setup to CAS. If one assumes a very slow address buffer and a very fast CAS decoder, there is a negative 1 ns margin.

There is a problem with DT HIGH to CAS HIGH after a transfer cycle. This problem will be corrected with REV C QPDMs.

1 t_{RC} Read Write Transfer

QPDM guarantees 6 SYSCLK cycles = 300
RAM requires 190

2 t_{RWC} RMW Cycle

QPDM never does Read/Modify/Write Cycles.

3 t_{PC} Page Mode Cycle

QPDM never does Page Mode Cycles.

4 t_{RAC} Row Access

+QPDM Para 32	160.0	160.0	160.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
-QPDM Para 45	-20.0	-20.0	-20.0
Total Time:	133.0	130.0	128.0
VRAM	100.0	100.0	100.0
Margins:	33.0	30.0	28.0

5 t_{CAC} Column Access

+QPDM Para 41	80.0	80.0	80.0
-CAS Decode	0.0	0.0	0.0
-CAS Delay	-3.0	-5.0	-10.0
-QPDM Para 45	-20.0	-20.0	-20.0
Total Time:	57.0	55.0	50.0
VRAM	50.0	50.0	50.0
Margins:	7.0	5.0	0.0

6 t_{OFF} Output Disable from CAS HI

-QPDM Para 46	0.0	0.0	0.0
+QPDM Para 43	13.0	13.0	13.0
+CAS Decode	0.0	0.0	0.0
+CAS Delay	10.0	5.0	3.0
Total Time:	23.0	18.0	16.0
VRAM	0.0	0.0	0.0
Margins:	23.0	18.0	16.0

7 t_T Transition

Transition	3.0	3.0	3.0

8 t_{RP} RAS Precharge

+QPDM Para 35	95.0	95.0	95.0
Total Time:	95.0	95.0	95.0
VRAM	80.0	80.0	80.0
Margins:	15.0	15.0	15.0

9 t_{RAS} RAS Pulse Width

+QPDM Para 33	180.0	180.0	180.0
Total Time:	180.0	180.0	180.0
VRAM	100.0	100.0	100.0
Margins:	80.0	80.0	80.0

10 t_{RSH} CAS Falls to RAS Rises

+QPDM Para 39	100.0	100.0	100.0
Total Time:	100.0	100.0	100.0
VRAM	50.0	50.0	50.0
Margins:	50.0	50.0	50.0

11 t_{CPN} CAS Precharge (Not PM)

+QPDM Para 40	40.0	40.0	40.0
Total Time:	40.0	40.0	40.0

Not a Parameter for this VRAM Vendor

12 t_{CP} CAS Precharge (PM)

QPDM never does Page Mode Cycles.

13 t_{CAS} CAS Pulse Width

+QPDM Para 57	80.0	80.0	80.0
Total Time:	80.0	80.0	80.0
VRAM	50.0	50.0	50.0
Margins:	30.0	30.0	30.0

14 t_{CSH} CAS Hold From RAS Falls

+QPDM Para 33	180.0	180.0	180.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
+CAS Decode	0.0	0.0	0.0
+CAS Delay	10.0	5.0	3.0
Total Time:	183.0	175.0	171.0
VRAM	100.0	100.0	100.0
Margins:	83.0	75.0	71.0

15 t_{RCD} RAS to CAS Delay

+QPDM Para 36	65.0	65.0	65.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
+CAS Decode	0.0	0.0	0.0
+CAS Delay	10.0	5.0	3.0
Total Time:	68.0	60.0	56.0
VRAM	50.0	50.0	50.0
Margins:	18.0	10.0	6.0

16 t_{CRP} CAS Hi to RAS Low Precharge

+QPDM Para 35	95.0	95.0	95.0
-CAS Decode	0.0	0.0	0.0
-CAS Delay	-3.0	-5.0	-10.0
+RAS Decode	0.0	0.0	0.0
+RAS Delay	12.0	10.0	7.0
Total Time:	104.0	100.0	92.0
VRAM	10.0	10.0	10.0
Margins:	94.0	90.0	82.0

17 t_{ASR} Address Setup to RAS

+QPDM Para 30	15.0	15.0	15.0
-Adrs Delay	-10.0	-13.0	-17.0
+RAS Decode	0.0	0.0	0.0
+RAS Delay	12.0	10.0	7.0
Total Time:	17.0	12.0	5.0
VRAM	0.0	0.0	0.0
Margins:	17.0	12.0	5.0

18 t_{RAH} Row Address Hold

+QPDM Para 31	35.0	35.0	35.0
+Adrs Delay	17.0	13.0	10.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
Total Time:	45.0	38.0	33.0
VRAM	15.0	15.0	15.0
Margins:	30.0	23.0	18.0

19 t_{ASC} Address Setup to CAS

+QPDM Para 37	13.0	13.0	13.0
-Adrs Delay	-10.0	-13.0	-17.0
+CAS Decode	0.0	0.0	0.0
+CAS Delay	10.0	5.0	3.0
Total Time:	13.0	5.0	-1.0
VRAM	0.0	0.0	0.0
Margins:	13.0	5.0	-1.0

****THERE MAY BE A PROBLEM IN THE ABOVE PARAMETER****

20 t_{CAH} Column Address Hold

+QPDM Para 38	80.0	80.0	80.0
+Adrs Delay	17.0	13.0	10.0
-CAS Decode	0.0	0.0	0.0
-CAS Delay	-3.0	-5.0	-10.0
Total Time:	94.0	88.0	80.0
VRAM	20.0	20.0	20.0
Margins:	74.0	68.0	60.0

21 t_{AR} Column Address Hold from RAS

+QPDM Para 56	90.0	90.0	90.0
+QPDM Para 38	80.0	80.0	80.0
+Adrs Delay	17.0	13.0	10.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
Total Time:	180.0	173.0	168.0

Not a Parameter for this VRAM Vendor

22 t_{RCS} Read Command Setup to CAS

+QPDM Para 43	13.0	13.0	13.0
+QPDM Para 40	40.0	40.0	40.0
-XFG Decode	0.0	0.0	0.0
-XFG Delay	-6.0	-9.0	-11.0
+CAS Decode	0.0	0.0	0.0
+CAS Delay	10.0	5.0	3.0
Total Time:	57.0	49.0	45.0
VRAM	0.0	0.0	0.0
Margins:	57.0	49.0	45.0

23 t_{RRH} Read Command Hold from RAS Hi

+QPDM Para 35	95.0	95.0	95.0
-QPDM Para 59	-11.0	-11.0	-11.0
+XFG Decode	0.0	0.0	0.0
+XFG Delay	11.0	9.0	6.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
Total Time:	88.0	83.0	78.0
VRAM	10.0	10.0	10.0
Margins:	78.0	73.0	68.0

24 t_{RCH} Read Command Hold from CAS Hi

+QPDM Para 35	95.0	95.0	95.0
-QPDM Para 59	-11.0	-11.0	-11.0
+XFG Decode	0.0	0.0	0.0
+XFG Delay	11.0	9.0	6.0
-CAS Decode	0.0	0.0	0.0
-CAS Delay	-3.0	-5.0	-10.0
Total Time:	92.0	88.0	80.0
VRAM	0.0	0.0	0.0
Margins:	92.0	88.0	80.0

25 t_{WCS} Write Command Setup to CAS

+QPDM Para 60	13.0	13.0	13.0
+WE Decode	0.0	0.0	0.0
+WE Delay	11.0	9.0	6.0
-CAS Decode	0.0	0.0	0.0
-CAS Delay	-3.0	-5.0	-10.0
Total Time:	21.0	17.0	9.0
VRAM	0.0	0.0	0.0
Margins:	21.0	17.0	9.0

26 t_{WCH} Write Command Hold

+QPDM Para 61	78.0	78.0	78.0
-QPDM Para 60	-13.0	-13.0	-13.0
+WE Decode	0.0	0.0	0.0
+WE Delay	11.0	9.0	6.0
-CAS Decode	0.0	0.0	0.0
-CAS Delay	-3.0	-5.0	-10.0
Total Time:	73.0	69.0	61.0
VRAM	25.0	25.0	25.0
Margins:	48.0	44.0	36.0

27 t_{WCR} Write CMND Hold from RAS Falls

+QPDM Para 61	78.0	78.0	78.0
+QPDM Para 56	90.0	90.0	90.0
-QPDM Para 60	-13.0	-13.0	-13.0
+WE Decode	0.0	0.0	0.0
+WE Delay	11.0	9.0	6.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
Total Time:	159.0	154.0	149.0

Not a Parameter for this VRAM Vendor

28 t_{WP} Write Pulse Width

+QPDM Para 61	78.0	78.0	78.0
Total Time:	78.0	78.0	78.0
VRAM	15.0	15.0	15.0
Margins:	63.0	63.0	63.0

29 t_{RWL} Write Command to RAS

+QPDM Para 33	180.0	180.0	180.0
-QPDM Para 56	-90.0	-90.0	-90.0
+QPDM Para 60	13.0	13.0	13.0
+RAS Decode	0.0	0.0	0.0
+RAS Delay	12.0	10.0	7.0
-WE Decode	0.0	0.0	0.0
-WE Delay	-6.0	-9.0	-11.0
Total Time:	109.0	104.0	99.0
VRAM	35.0	35.0	35.0
Margins:	74.0	69.0	64.0

30 t_{CWL} Write Command to CAS Lead Time

+QPDM Para 60	13.0	13.0	13.0
+QPDM Para 57	80.0	80.0	80.0
+CAS Decode	0.0	0.0	0.0
+CAS Delay	10.0	5.0	3.0
-WE Decode	0.0	0.0	0.0
-WE Delay	-6.0	-9.0	-11.0
Total Time:	97.0	89.0	85.0
VRAM	30.0	30.0	30.0
Margins:	67.0	59.0	55.0

31 t_{DS} Data Setup to CAS

+QPDM Para 64	2.0	2.0	2.0
+CAS Decode	0.0	0.0	0.0
+CAS Delay	10.0	5.0	3.0
Total Time:	12.0	7.0	5.0
VRAM	0.0	0.0	0.0
Margins:	12.0	7.0	5.0

32 t_{DH} Data Hold from CAS

+QPDM Para 65	60.0	60.0	60.0
-CAS Decode	0.0	0.0	0.0
-CAS Delay	-3.0	-5.0	-10.0
Total Time:	57.0	55.0	50.0
VRAM	25.0	25.0	25.0
Margins:	32.0	30.0	25.0

33 D_{HR} Data Hold from RAS

+QPDM Para 56	90.0	90.0	90.0
+QPDM Para 65	60.0	60.0	60.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
Total Time:	143.0	140.0	138.0

Not a Parameter for this VRAM Vendor

34 t_{CWD} CAS to WE Delay

QPDM never does Read/Modify/Write Cycles.

35 t_{RWD} RAS to WE Delay

QPDM never does Read/Modify/Write Cycles.

37 t_{OED} OE High to Data In Setup

QPDM never does Read/Modify/Write Cycles.

38 t_{OEH} OE Hi hold from WE Low

QPDM never does Read/Modify/Write Cycles.

39 t_{OEZ} Output Disable from OE Hi

+QPDM Para 46	0.0	0.0	0.0
+XFG Decode	0.0	0.0	0.0
+XFG Delay	11.0	9.0	6.0
Total Time:	11.0	9.0	6.0
VRAM	0.0	0.0	0.0
Margins:	-11.0	-9.0	-6.0

****THERE MAY BE A PROBLEM IN THE ABOVE PARAMETER****

40 t_{CSR} CAS to RAS Setup for Refresh

+QPDM Para 47	37.0	37.0	37.0
+RAS Decode	0.0	0.0	0.0
+RAS Delay	12.0	10.0	7.0
-CAS Decode	0.0	0.0	0.0
-CAS Delay	-3.0	-5.0	-10.0
Total Time:	46.0	42.0	34.0
VRAM	10.0	10.0	10.0
Margins:	36.0	32.0	24.0

41 t_{CHR} CAS before RAS Refresh Hold

+QPDM Para 48	185.0	185.0	185.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
+CAS Decode	0.0	0.0	0.0
+CAS Delay	10.0	5.0	3.0
Total Time:	188.0	180.0	176.0
VRAM	20.0	20.0	20.0
Margins:	168.0	160.0	156.0

42 t_{RPC} RAS HI to CAS Lo Precharge

+QPDM Para 40	40.0	40.0	40.0
+CAS Decode	0.0	0.0	0.0
+CAS Delay	10.0	5.0	3.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
Total Time:	43.0	35.0	31.0
VRAM	10.0	10.0	10.0
Margins:	33.0	25.0	21.0

43 t_{REF} Refresh Interval

You may program DMRR to 625

44 t_{DLS} DT to RAS Setup for Xfer

+QPDM Para 49	12.0	12.0	12.0
-XFG Decode	0.0	0.0	0.0
-XFG Delay	-6.0	-9.0	-11.0
+RAS Decode	0.0	0.0	0.0
+RAS Delay	12.0	10.0	7.0
Total Time:	18.0	13.0	8.0
VRAM	0.0	0.0	0.0
Margins:	18.0	13.0	8.0

45 t_{RDH} DT Hold from RAS for Xfer

+QPDM Para 32	160.0	160.0	160.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
+XFG Decode	0.0	0.0	0.0
+XFG Delay	11.0	9.0	6.0
Total Time:	164.0	159.0	154.0
VRAM	80.0	80.0	80.0
Margins:	84.0	79.0	74.0

46 t_{CDH} DT Hold After CAS LO

+QPDM Para 41	80.0	80.0	80.0
-CAS Decode	0.0	0.0	0.0
-CAS Delay	-3.0	-5.0	-10.0
+XFG Decode	0.0	0.0	0.0
+XFG Delay	11.0	9.0	6.0
Total Time:	88.0	84.0	76.0

Not a Parameter for this VRAM Vendor

47 t_{SDD} SC HI to DT HI Delay

+QPDM Para 52	90.0	90.0	90.0
+QPDM Para 32	160.0	160.0	160.0
+XFG Decode	0.0	0.0	0.0
+XFG Delay	11.0	9.0	6.0
Total Time:	261.0	259.0	256.0

Not a Parameter for this VRAM Vendor

48 t_{SDH} SC Low Hold after DT Hi

+QPDM Para 34	14.0	14.0	14.0
+QPDM Para 53	40.0	40.0	40.0
Total Time:	54.0	54.0	54.0

Not a Parameter for this VRAM Vendor

49 t_{OE} OE Pulse Width

+QPDM Para 44	110.0	110.0	110.0
Total Time:	110.0	110.0	110.0

Not a Parameter for this VRAM Vendor

58 t_{DHS} DT HI Setup to RAS (no XFER)

+QPDM Para 43	13.0	13.0	13.0
+QPDM Para 35	95.0	95.0	95.0
+RAS Decode	0.0	0.0	0.0
+RAS Delay	12.0	10.0	7.0
-XFG Decode	0.0	0.0	0.0
-XFG Delay	-6.0	-9.0	-11.0
Total Time:	114.0	109.0	104.0
VRAM	0.0	0.0	0.0
Margins:	114.0	109.0	104.0

59 t_{DHH} DT Hold from RAS

+QPDM Para 42	40.0	40.0	40.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
+XFG Decode	0.0	0.0	0.0
+XFG Delay	11.0	9.0	6.0
Total Time:	44.0	39.0	34.0

Not a Parameter for this VRAM Vendor

60 t_{DTR} DT HI to RAS HI Delay

+QPDM Para 34	14.0	14.0	14.0
+RAS Decode	0.0	0.0	0.0
+RAS Delay	12.0	10.0	7.0
-XFG Decode	0.0	0.0	0.0
-XFG Delay	-6.0	-9.0	-11.0
Total Time:	20.0	15.0	10.0
VRAM	10.0	10.0	10.0
Margins:	10.0	5.0	0.0

61 t_{DTC} DT HI to CAS HI Delay

+QPDM Para 43	13.0	13.0	13.0
+CAS Decode	0.0	0.0	0.0
+CAS Delay	10.0	5.0	3.0
-XFG Decode	0.0	0.0	0.0
-XFG Delay	-6.0	-9.0	-11.0
Total Time:	17.0	9.0	5.0
VRAM	10.0	10.0	10.0
Margins:	7.0	-1.0	-5.0

****THERE MAY BE A PROBLEM IN THE ABOVE PARAMETER****

62 t_{OES} OE Setup to RAS HI

+QPDM Para 44	110.0	110.0	110.0
+QPDM Para 34	14.0	14.0	14.0
-XFG Decode	0.0	0.0	0.0
-XFG Delay	-6.0	-9.0	-11.0
+RAS Decode	0.0	0.0	0.0
+RAS Delay	12.0	10.0	7.0
Total Time:	130.0	125.0	120.0

Not a Parameter for this VRAM Vendor

64 t_{WBS} Masked Write Command Setup

+QPDM Para 59	11.0	11.0	11.0
-WE Decode	0.0	0.0	0.0
-WE Delay	-6.0	-9.0	-11.0
+RAS Decode	0.0	0.0	0.0
+RAS Delay	12.0	10.0	7.0
Total Time:	17.0	12.0	7.0
VRAM	0.0	0.0	0.0
Margins:	17.0	12.0	7.0

65 t_{WBH} Masked Write Command Hold

+QPDM Para 60	13.0	13.0	13.0
+QPDM Para 61	78.0	78.0	78.0
+RAS Decode	0.0	0.0	0.0
+RAS Delay	12.0	10.0	7.0
-WE Decode	0.0	0.0	0.0
-WE Delay	-6.0	-9.0	-11.0
Total Time:	97.0	92.0	87.0
VRAM	15.0	15.0	15.0
Margins:	82.0	77.0	72.0

66 t_{WS} Write Mask Setup

+QPDM Para 52	90.0	90.0	90.0
+RAS Decode	0.0	0.0	0.0
+RAS Delay	12.0	10.0	7.0
Total Time:	102.0	100.0	97.0
VRAM	0.0	0.0	0.0
Margins:	102.0	100.0	97.0

67 t_{WH} Write Mask Hold

+QPDM Para 63	60.0	60.0	60.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
Total Time:	53.0	50.0	48.0
VRAM	15.0	15.0	15.0
Margins:	38.0	35.0	33.0

70 t_{DTH} DT HI Hold after RAS HI

+QPDM Para 35	95.0	95.0	95.0
-QPDM Para 49	-12.0	-12.0	-12.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
+XFG Decode	0.0	0.0	0.0
+XFG Delay	11.0	9.0	6.0
Total Time:	87.0	82.0	77.0
VRAM	15.0	15.0	15.0
Margins:	72.0	67.0	62.0

71 $t_{h(OECH)}$ CAS hold after OE low

+QPDM Para 44	110.0	110.0	110.0
+QPDM Para 43	13.0	13.0	13.0
-XFG Decode	0.0	0.0	0.0
-XFG Delay	-6.0	-9.0	-11.0
+CAS Decode	0.0		0.0
0.0			
+CAS Delay	10.0		5.0
3.0			
Total Time:	127.0		119.0
115.0			

Not a Parameter for this VRAM Vendor

73 $t_{h(RLOE)}$ OE hold after RAS low

+QPDM Para 42	40.0	40.0	40.0
+QPDM Para 44	110.0	110.0	110.0
-RAS Decode	0.0	0.0	0.0
-RAS Delay	-7.0	-10.0	-12.0
+XFG Decode	0.0	0.0	0.0
+XFG Delay	11.0	9.0	6.0
Total Time:	154.0	149.0	144.0

Not a Parameter for this VRAM Vendor

74 t_{DOEL} Delay Data to OE low

QPDM never does hidden refresh cycles.

75 t_{DCL} Delay data to CAS low

+QPDM Para 57	80.0	80.0	80.0
+QPDM Para 40	40.0	40.0	40.0
+QPDM Para 65	60.0	60.0	60.0
-CAS Decode	0.0	0.0	0.0
-CAS Delay	-3.0	-5.0	-10.0
Total Time:	177.0	175.0	170.0

Not a Parameter for this VRAM Vendor

78 $t_{su(WE)}$ WE setup to RAS low

QPDM never does write transfer cycles.

79 $t_{h(WE)}$ WE hold after RAS low

QPDM never does write transfer cycles.

80 $t_{su(SE)}$ SE setup to RAS low

QPDM never does write transfer cycles.

81 $s_{h(SE)}$ SE hold after RAS low

QPDM never does write transfer cycles.

82 $t_{su(SD)}$ Serial in setup to SC high

QPDM never does serializer writes.

83 $t_{h(SD)}$ Serial in hold after SC high

QPDM never does serializer writes.

84 $t_{su(SCRL)}$ SC setup to RAS low

QPDM never does serializer writes.

85 $t_{su(SEH)}$ SE disable setup to SC high

QPDM never does serializer writes.

86 $t_{h(SEH)}$ SE disable hold from SC high

QPDM never does serializer writes.

87 $t_{su(SEL)}$ SE enable setup before SC high

QPDM never does serializer writes.

88 $t_{h(SEL)}$ SE enable hold from SC high

QPDM never does serializer writes.

89 t_{DDTH} Delay data to DT high

QPDM never does serializer writes.

90 t_{DTHD} Delay DT high to data

QPDM never does serializer writes.

91 $t_{w(TRG)}$ TRG Pulse width

+QPDM Para 44	110.0	110.0	110.0
Total Time:	110.0	110.0	110.0

Not a Parameter for this VRAM Vendor

92 t_{CLGH} CAS low to TRG high

+QPDM Para 41	80.0	80.0	80.0
Total Time:	80.0	80.0	80.0

Not a Parameter for this VRAM Vendor

93 t_{RLSH} RAS low to SC high after TRG hi

This will handled only on REV C QPDM silicon.

94 t_{THRL} TRG high to RAS low after xfer

+QPDM Para 34	14.0	14.0	14.0
+QPDM Para 35	95.0	95.0	95.0
+RAS Decode	0.0	0.0	0.0
+RAS Delay	12.0	10.0	7.0
-XFG Decode	0.0	0.0	0.0
-XFG Delay	-6.0	-9.0	-11.0
Total Time:	115.0	110.0	105.0

Not a Parameter for this VRAM Vendor

95 t_{CLSH} CAS low to SC after TRG

This will handled only on REV C QPDM silicon.

96 t_{SHRL} SC high to RAS low (w/xfer)

QPDM never does write transfer cycles.

97 t_{RHSH} RAS high to SC high

QPDM never does write transfer cycles.

98 t_{THSH} TRG high to SC high

This will handled only on REV C QPDM silicon.

5.8 SOFTWARE

At this writing, the following software packages are known to run on this board. These are all available from AMD.

5.8.1 QASM

QASM (QPDM Assembler) is a line-at-a-time assembler designed explicitly for the QPDM. The binary code is shipped with the board and the source code is available from AMD for a nominal price.

QASM can be used with the board in two basic ways. In interactive mode, the user can enter an instruction in mnemonic form and watch the results on the screen. This is very useful for experimenting with the QPDM (evaluation). The other way QASM can be used with this board is to assemble from a prepared file, perhaps for a demonstration.

QASM can also be used to translate mnemonic QPDM instructions into "ones and zeroes" for entry into other programs. The "C" array initializer is especially useful for this purpose.

In addition to QASM itself, a number of generally useful source files (QASM input) are shipped with the board.

5.8.2 QDEMO

AMD has prepared an extensive demonstration program for the QPDM that runs on this board. The binary code for this program is shipped with the board and the source code is available from AMD.

The demonstration runs for about 10 minutes and we are making additions to it as time permits. It is intended to show the QPDM to its best advantage. In addition to being a good demonstration of QPDM capability, it also contains many examples of QPDM programming methods.

5.8.3 Other Demos

An additional set of demos has been written for the QPDM. These became available in August of 1987. The binary for these is shipped with the boards. These demos must be run on an AT and require the DMA modification.

5.8.4 G.K.S.

AMD has a "C" binding of the ANSI Graphical Kernel System (G.K.S.). This board was used as the debugging vehicle for this library. This is available from AMD.

5.8.5 Others

AMD has contracted with various third-party vendors for drivers for X-Windows, MS Windows, AutoCAD, and GEM/VP. These will become available during the first half of 1988. In most cases, the board will have been used as the debugging tool.

5.9 PAL DEVICE EQUATIONS

5.9.1 ADECODE

```
MODULE ADECODE
FLAG '-R2'
TITLE 'Generates Address Decodes on REV B QPDM BOARD'

"COPYRIGHT 1987 ADVANCED MICRO DEVICES, INC
"Tom Crawford  Feb 17, 1987

"DECLARATIONS
IC36 DEVICE 'P22V10';
BA2,BA3,BA4,BA5,BA6,BA7 PIN 1,2,3,4,5,6;
BA8,BA9,BA10,BA11,BA12,BA13 PIN 7,8,9,10,11,23;
BA14,BA15,BA16,BA17,BA18,BA19 PIN 22,21,16,15,14,13;
PROM PIN 20;
QPDM PIN 19;
LUT PIN 18;
AUX PIN 17;
VCC PIN 24;
GND PIN 12;

EQUATIONS

!QPDM       =     BA9 & BA8 & BA7 & !BA6 & BA5 & !BA4 & !BA3;

!AUX        =     BA9 & BA8 & BA7 & !BA6 & BA5 & !BA4 & BA3;

!LUT        =     BA19 & !BA18 & BA17 & BA16 & BA15 & BA14 & BA13 & BA12 & BA11 &
                  BA10 & BA9 & BA8 & BA7;

!PROM       =     BA19 & !BA18 & BA17 & BA16 & !BA15
            #     BA19 & !BA18 & BA17 & BA16 & !BA14
            #     BA19 & !BA18 & BA17 & BA16 & !BA13
            #     BA19 & !BA18 & BA17 & BA16 & !BA12
            #     BA19 & !BA18 & BA17 & BA16 & !BA11
            #     BA19 & !BA18 & BA17 & BA16 & !BA10
            #     BA19 & !BA18 & BA17 & BA16 & !BA9
            #     BA19 & !BA18 & BA17 & BA16 & !BA8
            #     BA19 & !BA18 & BA17 & BA16 & !BA7;

END
```

5.9.2 BCONT

```
MODULE BUFFER_CONTROL
FLAG '-R2'
TITLE 'Generates Buffer Ebables and Prom Enables for REVB QPDM BOARD'

"COPYRIGHT 1987 ADVANCED MICRO DEVICES, INC
"Tom Crawford  Feb 17, 1987
"Dave August 27 May 87

"DECLARATIONS
IC29 DEVICE 'P22V10';
```

```
PROM PIN 1;
AEN,SMEMR,SBHE,AT PIN 3,4,5,6;
Q,RD,WR PIN 7,8,9;
SPIN10,SPIN13 PIN 10,13;
BA0 PIN 11;
GND PIN 12;
EN_LO_BUF,EN_SW_BUF,EN_HI_BUF,DRV_PC PIN 14,19,21,15;
SIO16,SIO18,SIO20 PIN 16,18,20;
MEMCS16 PIN 17;
EN_LO_PROM,EN_HI_PROM PIN 22,23;
VCC PIN 24;

EQUATIONS
!EN_HI_PROM     =     !PROM & !AEN & !SMEMR & !AT & !SBHE & !BA0   "Word Access on AT
                #     !PROM & !AEN & !SMEMR & !AT & !SBHE & BA0    "Odd Byte on AT
                #     !PROM & !AEN & !SMEMR & AT &  BA0;           "Odd Byte on PC

!EN_LO_PROM     =     !PROM & !AEN & !SMEMR & !AT & !SBHE & !BA0   "Word Access on AT
                #     !PROM & !AEN & !SMEMR & !AT & SBHE & !BA0    "Even Byte on AT
                #     !PROM & !AEN & !SMEMR & AT & !BA0;           "Even Byte on PC

!EN_HI_BUF      =     !PROM & !AEN & !SMEMR & !AT & !SBHE & !BA0   "Word Access
                #     !PROM & !AEN & !SMEMR & !AT & !SBHE & BA0    "Odd Byte on AT
                #     Q & !AT & RD & !SBHE                         "Word Access to QPDM
                #     Q & !AT & WR & !SBHE;                        "Word Access to QPDM

!EN_SW_BUF      =     !PROM & !AEN & !SMEMR & AT & BA0;            "Odd Byte on PC

!EN_LO_BUF      =     !PROM & !AEN & !SMEMR & !AT & !SBHE & !BA0   "Word Access on AT
                #     !PROM & !AEN & !SMEMR & !AT & SBHE & !BA0    "Even Byte Access on AT
                #     !PROM & !AEN & !SMEMR & AT                   "Memory Access on PC
                #     Q & !AT & RD                                 "QPDM on PC
                #     Q & !AT & WR                                 "QPDM on PC
                #     RD                                           "LUT
                #     WR;                                          "LUT

!DRV_PC         =     RD;

ENABLE MEMCS16  =    !PROM;

!MEMCS16        =    !PROM & !AEN & !SMEMR & !AT & !SBHE & !BA0;
END BUFFER_CONTROL;
```

5.9.3 IOCONT

```
MODULE IOCONTROL
FLAG '-R2'
TITLE 'GENERATES QPDM, LUT, AUX, IOCS16 FOR REVB QPDM BOARD'

"COPYRIGHT 1987 ADVANCED MICRO DEVICES, INC
"Tom Crawford  Feb 17, 1987
"After thought by Dave August 27 May 87

"DECLARATIONS
IC21 DEVICE 'P22V10';

SPIN1, SPIN11, SPIN13 PIN 1,11,13;
DDACK,AEN,SMEMR PIN 2,3,4;
QPDM, AUX, LUT PIN 5,7,6;
IOR, IOW, SMEMW PIN 8,9,10;
GND PIN 12;
VCC PIN 24;
LUTS1, LUTS0 PIN 14,15;
WR_AUX PIN 21;
IOCS16 PIN 20;
QPDM_RD, QPDM_WR PIN 22,23;
Q, RD, WR PIN 18,17,16;

EQUATIONS

!QPDM_RD          =       !QPDM & !AEN & !IOR;

!QPDM_WR          =       !QPDM & !AEN & !IOW;

WR_AUX            =       !AUX & !AEN & !IOW & DDACK;

!LUTS0            =       !LUT & !SMEMW & !AEN
                  #       !LUT & !LUTS1 & SMEMR & !AEN;

!LUTS1            =       !LUT & !SMEMR & !AEN
                  #       !LUT & !LUTS0 & !SMEMW & !AEN;

ENABLE IOCS16     =       !QPDM;

!IOCS16           =       !QPDM;

Q                 =       !QPDM & !AEN;

RD                =       !QPDM & !AEN & !IOR
                  #       !LUT & !AEN & !SMEMR;

WR                =       !QPDM & !AEN & !IOW
                  #       !LUT & !AEN & !SMEMW
                  #       !AUX & !AEN & !IOW;

END IOCONTROL;
```

5.9.4 CAS

```
MODULE CAS
FLAG '-r2'
TITLE 'PAL TO CONTROL CAS ON ONE-BANK BOARD'

"COPYRIGHT 1985 ADVANCED MICRODEVICES, INC
"TOM CRAWFORD          JUNE 20,1985 CHANGED MARCH 10, 1986 RCYCLED FEB 18, 87

"DECLARATIONS

IC3 DEVICE 'P18P8';

IN1,IN8,IN9  PIN 1,8,9;

RESET, INVRESET PIN 11,13;

SVS, SHS, SCS PIN 6,7,12;

RAS, CAS, XFG PIN 2,5,3;
DELAY_XFG PIN 4;

CAS0, CAS1, CAS2, CAS3 PIN 14,15,16,17;
XFER PIN 18;
DLYFER PIN 19;

EQUATIONS

!INVRESET   =     RESET;
SCS         =     SVS & !SHS
            #     !SVS & SHS;
!XFER       =     !XFG & RAS
            #     !XFER & !XFG;
!DLYFER     =     !XFER;
!CAS0       =     !CAS;
!CAS1       =     !CAS;
!CAS2       =     !CAS;
!CAS3       =     !CAS;

END CAS_CONTROL;
```

5.9.5 Count

```
MODULE COUNT
FLAG '-R2'
TITLE 'BIT COUNTER FOR QPDM SMALL BOARD'

"COPYRIGHT 1985 ADVANCED MICRO DEVICES, INC
"TOM CRAWFORD JUNE 17, 1985          PINOUTS MARCH 10, 1986  RECYCLED FEB 17, 87

"DECLARATIONS
IC10 DEVICE 'P16R8';

CLOCK PIN 1;
IN2,IN4,IN5,IN6,IN7,IN8,IN9 PIN 2,4,5,6,7,8,9;
Q0,Q1,VIDCLK PIN 14,13,17;
LSR PIN 18;
CBLANK PIN 16;
CFF1 PIN 15;
SBLK, DBLNK PIN 3,19;
ROUT12 PIN 12;
OUTPUT_ENABLE PIN 11;

EQUATIONS
Q0          :=      !Q0;
Q1          :=      !Q1 & Q0
            #       Q1 & !Q0;
VIDCLK      :=      !VIDCLK & Q1 & Q0
            #       VIDCLK & !Q1
            #       VIDCLK & !Q0;
!LSR        :=      Q1 & !Q0;
!ECBLANK    :=      !VIDCLK & !Q1 & Q0;
!CFF1       :=      !VIDCLK & Q1 & !Q0;
DBLNK       :=      SBLK;
END COUNT;
```

5.9.6 Enables

```
MODULE ENABLE
FLAG '-R2'
TITLE 'PAL TO ENABLE VRAMS AND GENERATE SHIFT PULSES AS WELL'

"COPYRIGHT 1985 ADVANCED MICRODEVICES, INC
"TOM CRAWFORD          JUNE 17,1985                    RECYCLED FEB 18, 1987

"DECLARATIONS
IC9 DEVICE 'P16L8';

DXCYC, VIDCLK, FF1 PIN 1,3,9;
Q0, Q1 PIN 6,7;
IN4, IN5, IN8, IN11 PIN 4,5,8,11;
SBLK PIN 2;
SHIFT1, SHIFT2 PIN 16,15;
G0, G1, G2, G3 PIN 18, 17, 14, 13;
OUT12, OUT19 PIN 12, 19;
```

```
EQUATIONS

!G0          =      !VIDCLK & FF1 & Q1;
!G1          =      VIDCLK & !FF1 & Q1;
!G2          =      !VIDCLK & !FF1 & Q1;
!G3          =      VIDCLK & FF1 & Q1;
!SHIFT1      =      !VIDCLK & !FF1 & !Q1 & !Q0 & !SBLK
             #      !DXCYC;
!SHIFT2      =      !VIDCLK & FF1 & !Q1 & !Q0 & !SBLK
             #      !DLYFER;

END ENABLES;
```

5.9.7 Shift

```
MODULE SHIFT
FLAG '-R2'
TITLE '4-BIT PARALLEL TO SERIAL SHIFT REGISTER WITH SYNCHRONOUS BLANK'

"COPYRIGHT 1986 ADVANCED MICRO DEVICES, INC
"TOM CRAWFORD  JUNE 17,1985  RECYCLED FEB 18, 1987'

"DECLARATIONS
IC12 DEVICE 'P16R8';

CLOCK PIN 1;
D3,D2,D1,D0 PIN 7,6,4,5;
Q3,Q2,Q1,Q0 PIN 19,18,17,16;
BLANK PIN 3;
LSR PIN 2;
IN8, IN9 PIN 8,9;
ROUT15,ROUT14,ROUT13,ROUT12 PIN 15,14,13,12;
OUTPUT_ENABLE PIN 11;

EQUATIONS

Q3           :=     !BLANK & LSR & Q2
             #      !BLANK & !LSR & D3;
Q2           :=     LSR & Q1
             #      !LSR & D2;
Q1           :=     LSR & Q0
             #      !LSR & D1;
Q0           :=     !LSR & D0;

END SHIFT;
```

5.9.8 Hilite

```
MODULE HILITE
FLAG '-R2'
TITLE 'GENERATES BLINK FOR REVB QPDM BOARD'

"COPYRIGHT 1987 ADVANCED MICRO DEVICES, INC
"Tom Crawford  Feb 17, 1987

"DECLARATIONS
IC28 DEVICE 'P22V10';
```

```
S1,S2 PIN 10,5;
P1,P2 PIN 4,3;
BLINK PIN 18;
Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7 PIN 23,15,22,16,21,17,20,19;
PIN14 PIN 14;

EQUATIONS

Q0          :=    !Q0;

Q1          :=    Q0 & !Q1
            #     !Q0 & Q1;

Q2          :=    Q0 & Q1 & !Q2
            #     !Q0 & Q2
            #     !Q1 & Q2;

Q3          :=    Q0 & Q1 & Q2 & !Q3
            #     !Q0 & Q3
            #     !Q1 & Q3
            #     !Q2 & Q3;

Q4          :=    Q0 & Q1 & Q2 & Q3 & !Q4
            #     !Q0 & Q4
            #     !Q1 & Q4
            #     !Q2 & Q4
            #     !Q3 & Q4;

Q5          :=    Q0 & Q1 & Q2 & Q3 & Q4 & !Q5
            #     !Q0 & Q5
            #     !Q1 & Q5
            #     !Q2 & Q5
            #     !Q3 & Q5
            #     !Q4 & Q5;

Q6          :=    Q0 & Q1 & Q2 & Q3 & Q4 & Q5 & !Q6
            #     !Q0 & Q6
            #     !Q1 & Q6
            #     !Q2 & Q6
            #     !Q3 & Q6
            #     !Q4 & Q6
            #     !Q5 & Q6;

Q7          :=    Q0 & Q1 & Q2 & Q3 & Q4 & Q5 & Q6 & !Q7
            #     !Q0 & Q7
            #     !Q1 & Q7
            #     !Q2 & Q7
            #     !Q3 & Q7
            #     !Q4 & Q7
            #     !Q5 & Q7
            #     !Q6 & Q7;
```

```
BLINK       :=    !S1 &  S2 & !P1 &  P2 &  Q4 &  Q5
            #     !S1 &  S2 &  P1 & !P2 &  Q5
            #     !S1 &  S2 &  P1 &  P2 &  Q5
            #     !S1 &  S2 &  P1 &  P2 &  Q4
            #     S1 & !S2 & !P1 &  P2 &  Q5 &  Q6
            #     S1 & !S2 &  P1 & !P2 &  Q6
            #     S1 & !S2 &  P1 &  P2 &  Q6
            #     S1 & !S2 &  P1 &  P2 &  Q5
            #     S1 &  S2 & !P1 &  P2 &  Q6 &  Q7
            #     S1 &  S2 &  P1 & !P2 &  Q7

            #     S1 &  S2 &  P1 &  P2 &  Q7
            #     S1 &  S2 &  P1 &  P2 &  Q6;

PIN14       =     BLINK;

END HILITE;
```

5.10 USERS GUIDE

5.10.1 Addressing

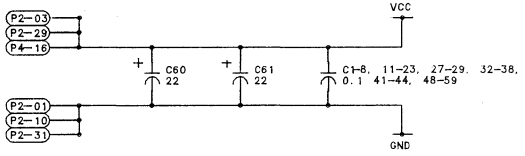
The locations used by the board in the address spaces of the host are controlled by programming U36. The logic equations are shown in 5.9.1. The user may change these addresses if necessary; then the software supplied by AMD will also have to be changed. The standard addresses are shown in the following table.

	Write Access	Read Access
I/O Space:		
03A0	Write QPDM FIFO	Read QPDM Status
03A2	Write QPDM BIF	Read QPDM BOF
03A4	Write QPDM Reg Adrs	Read QPDM Reg Adrs
03A6	Write QPDM Register	Read QPDM Register
03A8	Write Hilite Oscillator	
Memory Space:		
B0000 to -BFF7F	not applicable	Read EPROM
BFF80 to BFFFF	Write 8159 LUT	Read 8159 LUT

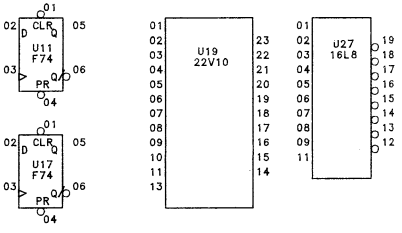
5.10.2 Jumpers

The following table shows the jumper blocks and their use on the board.

Jumper	USE	Case 1	Case 2	Case 3
W1	CAS Delay	5 ns	10 ns	15 ns
W2	HSYNC	Active HI	Active LO	Composite
W3	VSYNC	Active HI	Active LO	Composite
W4	Interrupt	INT2	INT3	INT5
W5	8159 VSYNC	Vsync	Ground	
W6	8159 HSYNC	Hsync	Ground	



SPARES

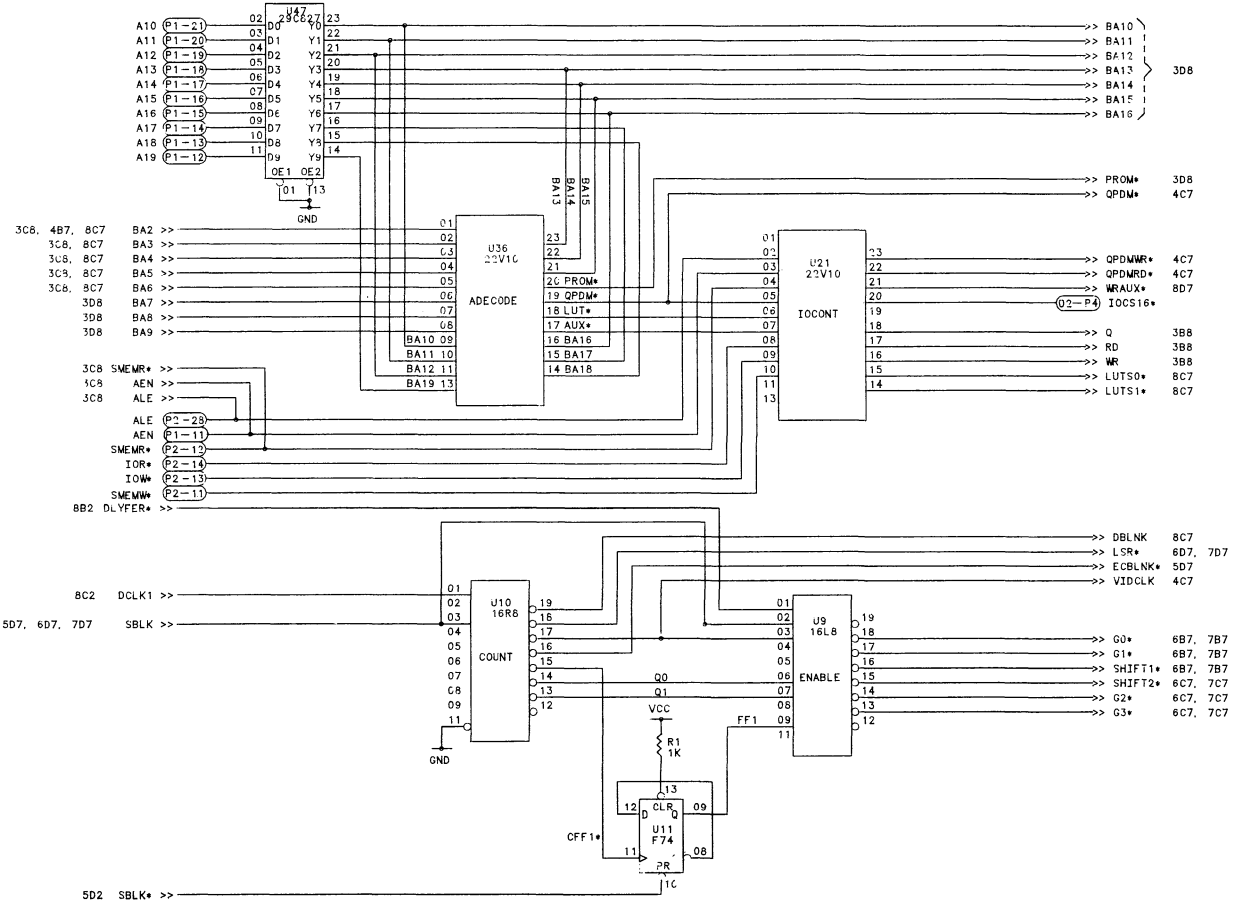


DEVICE TYPE	REF DESIG	# OF PINS	GND	VCC
DELAY LINE	U1	14	7	14
AM2966	U2, 5, 16	20	10	20
AM18P8	U3	20	10	20
74S244	U4	20	10	20
AM27512	U6, 7	28	14	28
DC/DC CONVERTER	U8	12	11	1
AM16L8	U9	20	10	20
AM16R8	U10	20	10	20
74F74	U11, 17	14	7	14
AM16R4	U12, 13, 14, 15	20	10	20
74LS379	U18	16	8	16
SPARE	U19	24	12	24
AM8159	U20	48		
AM22V10	U21, 28, 29, 36	24	12	24
41264-12	U22-25, U30-33, U37-44	24	24	12
AM95C60	U26	169		
SPARE	U27	20	10	20
74LS245	U34, 45, 48	20	10	20
AM29C845	U35	24	12	24
AM29C827	U46, 47	24	12	24

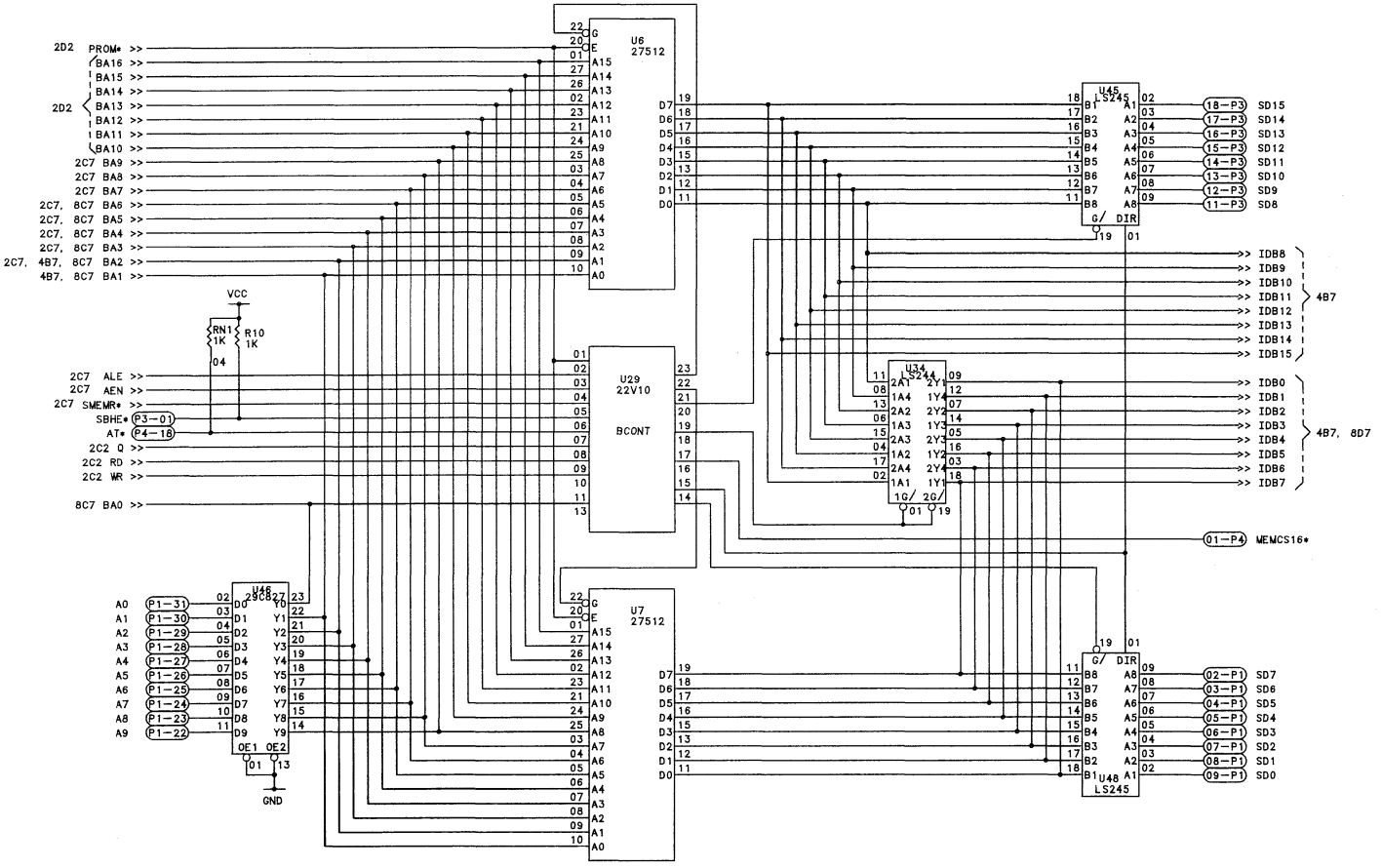
NOTES: UNLESS OTHERWISE SPECIFIED:

1. ALL RESISTOR VALUES ARE IN OHMS, 1/4W, 5%.
2. ALL CAPACITOR VALUES ARE IN MICROFARADS.

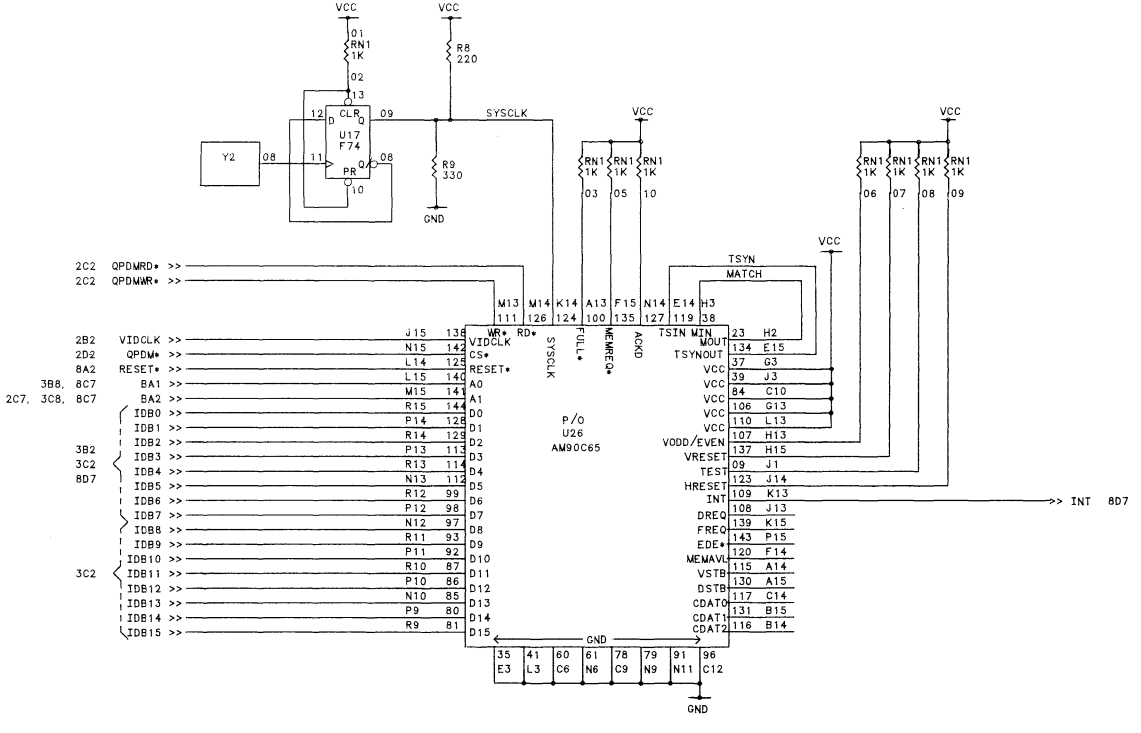
QPDM Plug In Board (Sheet 1 of 8)



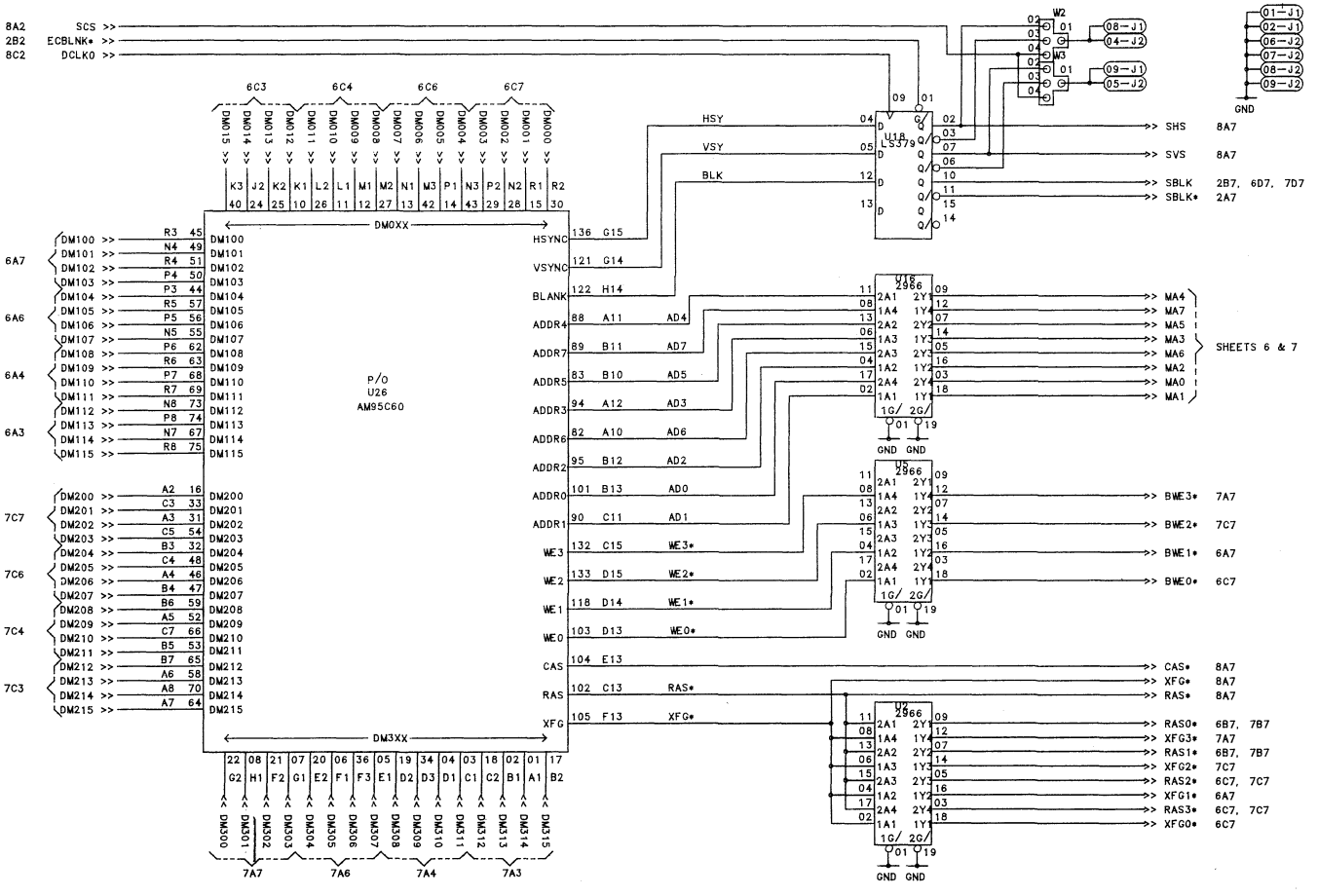
System Bus Controller (Sheet 2 of 8)



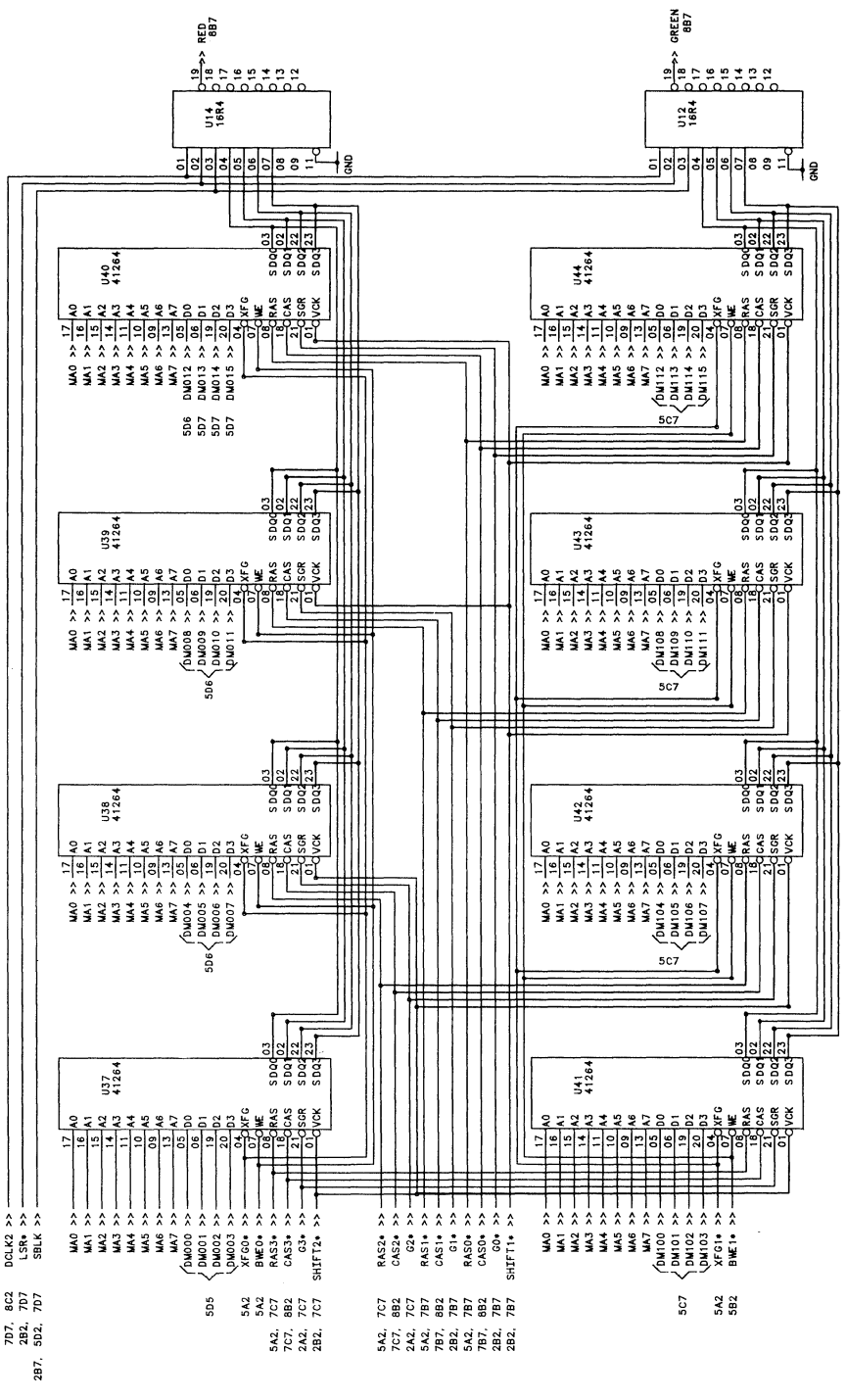
EPROMs and Data Bus (Sheet 3 of 8)



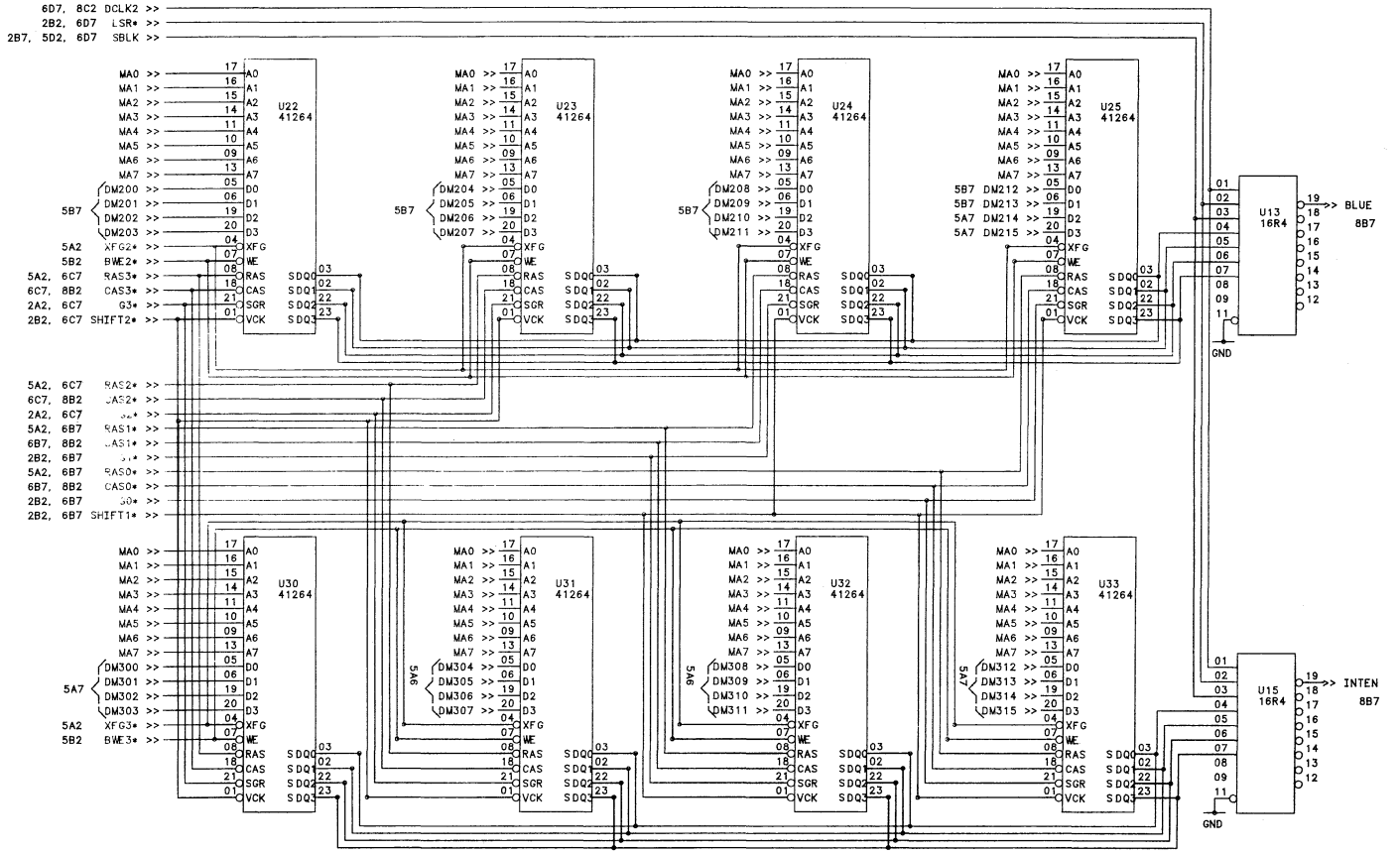
QPDM System Bus (Sheet 4 of 8)



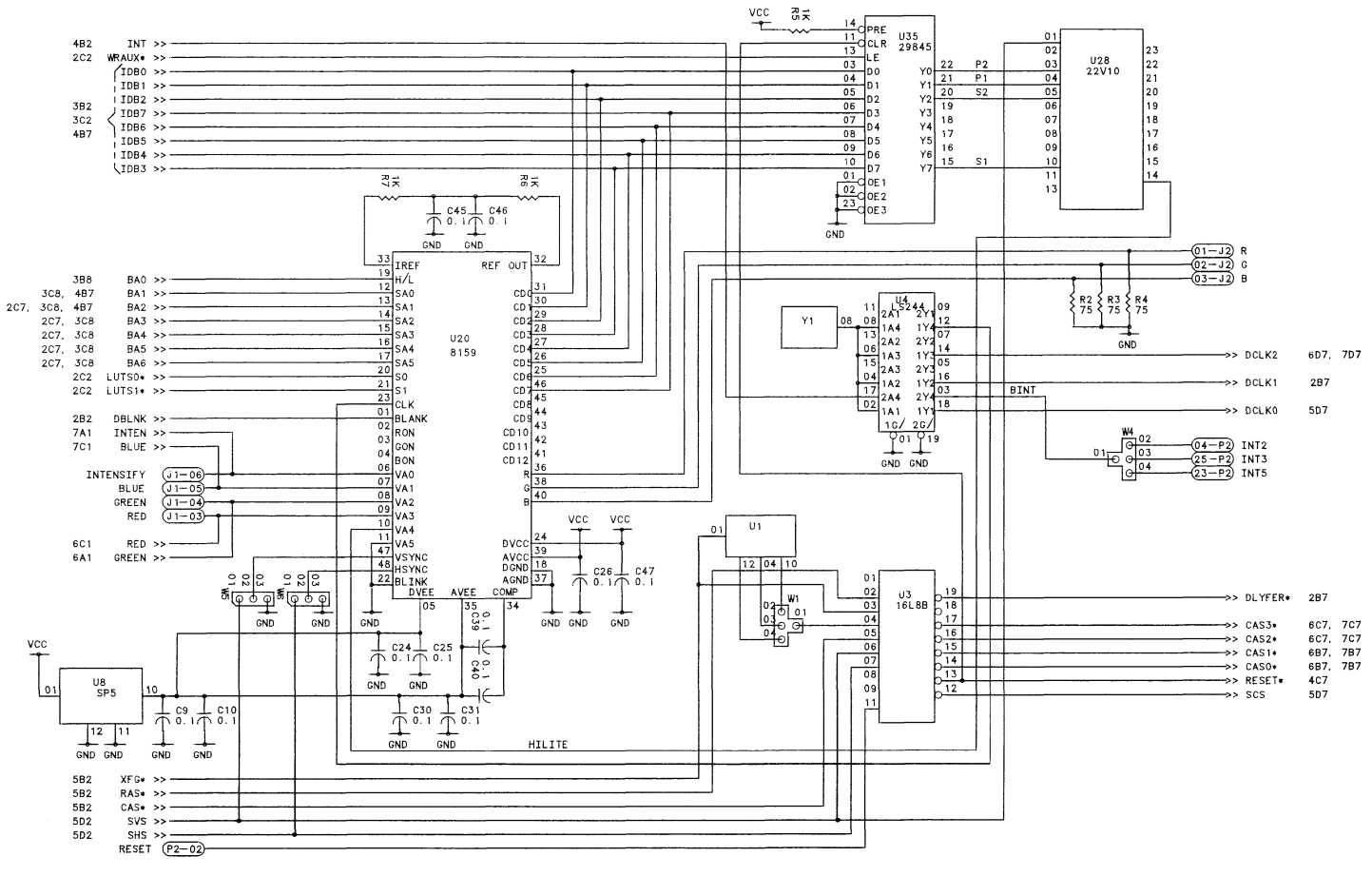
QPDM Memory Bus (Sheet 5 of 8)



Bit Map I (Sheet 6 of 8)



Bit Map II (Sheet 7 of 8)



8159 (Sheet 8 of 8)

CHAPTER 6

Articles/Application Notes

CHAPTER 6
Software

In this section, we present some software application hints for the Am95C60. Section 6.1 is a BASIC program that completely initializes the QPDM (from power on) and draws a very simple message on the screen. This program initializes the registers in the recommended order. It then issues the minimum instructions to get the QPDM started and draws a few wide lines.

In Section 6.2, we discuss the logical and arithmetic operations that can be performed on the source and destination fields during Copy Block operations.

6.1 INITIALIZATION

The following program will completely initialize the add-in board and draw the word "HI!" using stroke characters with a logical PEL (Pixel Element). This program was written in IBM PC BASICA and compiled using the BASIC compiler. Both source code and the compiled binary are shipped with the demonstration board.

The program contains adequate comments; we shall amplify as necessary. The board is described in Chapter 5.

When we execute lines 60-190, we are loading the look-up table of the Am8159. It is programmed as indicated in the table below; the 'Entry' column lists the values used by Set Color Bits.

When we execute lines 210-290, we define the I/O addresses and set the HILITE oscillator to 2 Hz with a 25% duty factor.

When we execute lines 310-420, we call on subroutines that initialize the registers. This is the recommended order. Note that entering 8-bit mode may not be necessary for some applications.

When we execute lines 440-520, we turn on Video Refresh Enable. This is synchronized to top-of-frame by waiting for Vertical Blanking Interrupt. When this program is executed from the interpreter (at least on a PC), the timing is not synchronized properly because it takes several milliseconds to execute lines 500 and 510. The compiled version does not have this problem.

When we execute lines 540-590 we send instructions to the QPDM to initialize it and draw a simple message. These are the minimum instructions required to program the QPDM. Line 560 removes one word from the list of data statements. If it is not the termination word (-1), it is sent to the instruction FIFO. After the last word has been sent, the program stops at line 600. The instructions are in the form of DATA statements occupying lines 630-840.

The subroutine at 1600 ensures there is room in the FIFO by waiting for FREQ (x4000 in Status Register) to be a "1". It then sends the word to the FIFO.

Table 6.1 Programming the Am8159

Entry	HILITE Off	HILITE On
0	Black	Black
1	Black	Grey
2	Dim Blue	Dim Blue
3	Bright Blue	Bright Blue
4	Dim Green	Dim Green
5	Bright Green	Bright Green
6	Dim Cyan	Dim Cyan
7	Bright Cyan	Bright Cyan
8	Dim Red	Dim Red
9	Bright Red	Bright Red
10	Dim Magenta	Dim Magenta
11	Bright Magenta	Bright Magenta
12	Dim Yellow	Dim Yellow
13	Bright Yellow	Bright Yellow
14	Dim White	Dim White
15	Bright White	Bright White

```
10 'This programs the 95C60 on Tom Crawford's demo board
20 ' for the NEC Multi-sync (or equivalent) monitor.  The
30 ' monitor is color, 640 x 480.  The Am8159 Color Palette
40 ' is programmed strictly as one to one (RGBI in, RGBI out).
50 DEFINT A-Z
60 DEF SEG=&HBFF8                                    'point to base of LUT
70 FOR J = 0 TO 126 STEP 2                           '128 locations in lut
80 R=0: G=0: B=0 :I=0    '
90 IF ((J AND 6)=4) THEN B=8                         'dim blue
100 IF ((J AND 6)=6) THEN B=15                       'bight blue
110 IF ((J AND 10)=8) THEN G=8                       'dim green
120 IF ((J AND 10)=10) THEN G=15                     'bright green
130 IF ((J AND 18)=16) THEN R=8                      'dim red
140 IF ((J AND 18)=18) THEN R=15                     'bright red
150 IF ((J AND 62)=34) THEN R=4: G=4: B=4:           'blink intense black
160 POKE J,(G*16+R)                                  'bits 7-0
170 POKE J+1,B                                       'bits 11-8
180 NEXT J
190 DEF SEG                                          'put it back to basic
200 '
210 BSE=&H3A0
220 INST=BSE                                         'instruction FIFO
230 STATUS=BSE                                       'status register
240 BIF=BSE+2                                        'block input FIFO
250 BOF=BSE+2                                        'block output FIFO
260 QADRS=BSE+4                                      'register address pointer
270 QEG=BSE+6                                        'io register
280 AUX=BSE+8                                        'BLINK CONTROL REGISTER
290 OUT AUX,9                                        'SET BLINK TO 1 Hz, 25
300 '
310 'Write the Registers to Initialize the QPDM
320 GOSUB 870                                        'sw reset
330 GOSUB 920                                        '8-bit mode
340 GOSUB 970                                        'interrupts off
350 GOSUB 1010                                        'screen
360 GOSUB 1080                                       'windows
370 GOSUB 1170                                       'horizontal
380 GOSUB 1240                                       'vertical
390 GOSUB 1320                                       'vmode
400 GOSUB 1360                                       'mmode
410 GOSUB 1400                                       'dmrr
420 GOSUB 1440                                       'vte
430 '
440 'now wait for Vertical Blank to enable video timing
450 'First clear all the interrupts (especially VLKBI)
460 QA=30 :V=&H3FF :GOSUB 1480 'clear interrupts
470 'Now read Status Register until VBLKI goes Active
480 B=INP(STATUS) :BL=INP(STATUS) 'always do two byte reads
```

```
490 IF (B AND 1) = 0 THEN GOTO 480 'wait for interrupt to occur
500 OUT QADRS,0 :OUT QADRS,29 'start video timing
510 OUT QEG,0 :OUT QEG,1
520 PRINT "enable VRE"
530  '
540 'Now send some instructions to QPDM
550 'We will initialize it and then write HI! with strokes
560 READ V
570 IF V= -1 THEN GOTO 600
580 GOSUB 1600
590 GOTO 560
600 STOP
610 '
620 'Here are the instructions in the form of DATA statements
630 DATA &H00B8                                    :'set QPDM Position, enable
Masked Writes
640 DATA &H29,0,0                                  :'turn off scaling
650 DATA &H39, &H03f0,0,&H01f0                     :'stack boundary
660 DATA &H34, &H35                                :' turn off clipping and picking
670 DATA &h30,15                                   :' Turn on all activity bits
680 DATA &H31,0                                    :'Set Listen Bits to All Planes
690 DATA &H36,15                                   :'turn off logical pel
700 DATA &H22,10,10                                :'small block size
710 DATA &H20,0                                    :'black color
720 DATA &h0550,0,0,1023,1023                      :'Filled Rectangle to clear
memory
730 DATA &H20,15                                   :'white drawing color
740 DATA &H54A,469,277,474,272                     :'Circle with radius of five
750 DATA &H21,15                                   :'Search Color of all ones
760 DATA &H54,469,277                              :'Fill the Circle with White
770 DATA &HB6, 7441                                :'Logical PEL at 464,272
780 DATA &H54C,100,60,100,280                      :'Left Stroke of 'H'
790 DATA &H54C,220,60,220,280                      :'Right Stroke of 'H'
800 DATA &H54c,100,160,220,160                     :'Crossbar of 'H'
810 DATA &H54c,300,60,340,60                       :'Top of 'I'
820 DATA &H54C,320,60,320,280                      :'Vertical Stroke of 'I'
830 DATA &H54C,300,280,340,280                     :'Bottom of 'I'
840 DATA &H54C,469,60,469,240                      :'Stroke of '!'
850 DATA -1
860 '
870 PRINT "SW reset"
880 OUT QADRS,0 :OUT QADRS,27
890 OUT QEG,0 :OUT QEG,0
900 RETURN
910 '
920 PRINT "8-Bit Mode"
930 OUT QADRS,31 :OUT QEG,0
940 OUT QADRS,59 :OUT QEG,0
```

```
950 RETURN
960 '
970 PRINT "Interrupts Off"
980 QA=26 :V=0 :GOSUB 1480
990 RETURN
1000 '
1010 PRINT "Screen Parameters"
1020 QA=1 :V=0 :GOSUB 1480                    'real start x
1030 QA=2 :V=0 :GOSUB 1480                    'real start y
1040 QA=3 :V=640 :GOSUB 1480                  'real term x
1050 QA=4 :V=480 :GOSUB 1480                  'real tem y
1060 RETURN
1070 '
1080 PRINT "windows"
1090 QA=14 :V=800 :GOSUB 1480                 'Apparent X Start
1100 QA=15 :V=500 :GOSUB 1480                 'Apparent Y Start
1110 QA=16 :V=800 :GOSUB 1480                 'Apparent X Terminate
1120 QA=17 :V=500 :GOSUB 1480                 'Apparent Y Terminate
1130 QA=18 :V=800 :GOSUB 1480                 'Real X Start
1140 QA=19 :V=800 :GOSUB 1480                 'Real Y Start
1150 RETURN
1160 '
1170 PRINT "horizontal"
1180 QA=10 :V=10 :GOSUB 1480                  'HSYNC
1190 QA=11 :V=20 :GOSUB 1480                  'H Scan Delay
1200 QA=12 :V=80 :GOSUB 1480                  'H Active
1210 QA=13 :V=104 :GOSUB 1480                 'H Total
1220 RETURN
1230 '
1240 PRINT "vertical"
1250 QA=5 :V=40 :GOSUB 1480                   'VSYNC
1260 QA=6 :V=50 :GOSUB 1480                   'V Scan Delay Odd
1270 QA=7 :V=50 :GOSUB 1480                   'V Scan Delay Even
1280 QA=8 :V=480 :GOSUB 1480                  'V Active
1290 QA=9 :V=1024 :GOSUB 1480                 'V Total
1300 RETURN
1310 '
1320 PRINT "vmode"
1330 QA=22 :V=11 :GOSUB 1480                  'Non-interlaced, Master, Master
1340 RETURN
1350 '
1360 PRINT "mmode"
1370 QA=23 :V=&H70 :GOSUB 1480                '64K Devices, 1K Display Memory
1380 RETURN
1390 '
1400 PRINT "DMRR"
1410 QA=24 :V=&H200+320 :GOSUB 1480           '320 SYSCLK Cycles and Bit 9
1420 RETURN
```

```
1430 '
1440 PRINT "VTE"
1450 QA=28 :V=1 :GOSUB 1480
1460 RETURN
1470 '
1480 'write 16 bit word to register
1490 'reg adrs in qa, value in v
1500 'most significant byte first
1510 B=INT(QA/256) :OUT QADRS,B
1520 B=QA MOD 256 :OUT QADRS,B
1530 'PRINT B,
1540 B=INT(V/256) :OUT QEG,B
1550 'PRINT B;"  ";
1560 B=V MOD 256 :OUT QEG,B
1570 'PRINT B
1580 RETURN
1590 '
1600 'write 16 bit word to inst FIFO
1610 'word is in v
1620 'most sig byte first
1630 'first make sure there is room in the FIFO
1640 B=INP (STATUS) :B1=INP(STATUS) 'always read two bytes
1650 IF (B AND 64) = 0 THEN GOTO 1640  'wait until FREQ is hi
1660 B=INT(V/256) :OUT INST,B
1670 'PRINT B;"   ";
1680 B=V MOD 256 :OUT INST,B
1690 'PRINT B
1700 RETURN
```

6.2 COPY BLOCK OPERATORS IN THE QPDM

6.2.1 Introduction

This chapter documents how to perform various logical and arithmetic operations with the Copy Block instruction on the QPDM. This study was inspired by Dale Simmonds who took the corresponding operations on the TMS34010 seriously.

6.2.2 Logical Operations

When two bi-modal quantities are logically combined, there are 16 possible results. In other words there are 16 functions of two variables, A and B. It is easy to prove there are exactly 16 ways; simply write down the cases exhaustively.

We can also describe the procedure for each of these. Twelve of the 16 cases can be executed in one operation. The other four require two operations each.

6.3.3 Arithmetic Operators

Overview

People have argued that arithmetic operations are useful when doing graphics. The operations are:

Add	dest = dest plus source
Add with Saturation	Forces all ones rather than overflow
Subtract	dest = dest minus source
Subtract w. Saturation	Forces all zeroes rather than underflow
Maximum	Compare and use the numerically larger
Minimum	Compare and use the numerically smaller

These operations can all be synthesized from the logical operations we have (SOAXZ). The only part that is expecially interesting (and time consuming) is the propagation of carries. As you go over this code, bear in mind that each operation is being done on more than one quantity (pixel) in parallel. This means that optimizing in real time based on the partial results cannot be done. Rather, you have to just go blindly through all the motions.

Add

We shall describe the add routine in great detail. All the others are built to some greater or lesser degree on add. The nomenclature for the four bit planes in a single QPDM is shown below.

Plane Number	0 1 2 3	(for font instruction)
Weight	8 4 2 1	(for act instruction)

Carries propagate from right to left. Plane number 3 contains the LSB of each pixel; plane number 0 contains the MSB of each pixel.

Four blocks in the display memory are defined. The two original operands are "dest" and "source". The operation is defined in a manner consistent with the normal QPDM logical operations.

dest = dest plus source

The other two blocks are "temp1" and "temp2". These are used to contain intermediate results as described below.

Input Values of B,A: (where B is the source and A the destination)

0,0	0,1	1,0	1,1	Name	Equation	Procedure
0	0	0	0	Clear	A = 0	Copy Destination to itself with AND and SI
0	0	0	1	And	A = $\underline{A}$ AND B	Copy Block with Logical AND
0	0	1	0	And Reverse	A = $\overline{A}$ AND B	a) Invert Destination, b) Copy with Logical AND
0	0	1	1	Copy	A = B	Copy Block with Logical SET
0	1	0	0	AndInverted	A = A AND $\overline{B}$	Copy Block with Logical AND, SI
0	1	0	1	NoOp	A = A	(left as an exercise for the reader)
0	1	1	0	Xor	A = A XOR B	Copy Block with Logical XOR
0	1	1	1	Or	$\underline{A}$ = A OR B	Copy Block with Logical OR
1	0	0	0	Nor	$\overline{A}$ = A OR B	a) Copy Block with Logical OR, b)Invert Dest
1	0	0	1	Equivalent	A = $\underline{A}$ XOR $\overline{B}$	Copy Block with Logical XOR, SI
1	0	1	0	Invert	A = $\underline{A}$	Invert Destination
1	0	1	1	OrReverse	A = $\overline{A}$ OR B	a) Invert Destination, b)Copy Block with OR
1	1	0	0	CopyInverted	A = $\overline{B}$	Copy Block with Logical SET, SI
1	1	0	1	OrInverted	$\underline{A}$ = A OR $\overline{B}$	Copy Block with Logical OR, SI
1	1	1	0	NAnd	$\overline{A}$ = A AND B	a) Copy Block with Logical AND, b)Invert Dest
1	1	1	1	Set	A = 1	Copy Destination to itself with OR and SI

" Invert Destination" means: Copy Block Dest to Dest with SI

The program for add is shown in addtemp (which stands for add TEMPlate). The block size of the arrays to be added is set and all the activity bits are set.

The "propagates" are calculated and placed in temp1. The propagate for each bit position of each pixel is the logical OR of the two operands. If the result is set, then a carry into this bit position will result in a carry out. Note that this calculation is a two-step operation. This is because the QPDM requires the destination block to be the same as one of the sources. (2-address machine)

The "generates" are calculated and placed in temp2. The generate for each bit position of each pixel is the logical AND of the two operands. If the result is set, then there will be a carry out from this bit position regardless of any carry in. We will see later how the propagates and generates are combined.

The initial sums (with no carries) are calculated into "dest" by XORing with "source". Recall that XOR is a so-called "half-add". This is all there is to it except for the carries.

For the case of add, there is no carry into the low-order bit, so we can go directly to plane 2. We must add any carry generated from plane 3 into plane 2. We want to affect plane 2 only, so we set only its activity bit. We want to use the generate from plane 3 so we set the single plane source with the fnt 3: instruction. We XOR the generate from plane 3 into the destination of plane 2. This leaves the correct sum in plane 2. Note that the single plane source bit is set so that the source operand (temp2) comes from plane 3.

Now the generate from plane 3 is ANDed with the propagate from plane 2 with the result left in the plane 2 propagate array. Finally, this is ORed with the plane 2 generate and the result is left in plane 2 generate. This can be done without single plane source since both operands are in plane 2. Observe all this has affected only plane 2.

"Generate" from any plane is propagate and carry-in or generate. In a similar manner, we calculate the sum for plane 1 and the carry from plane 1. Finally, we calculate the sum for plane 0. This completes the add routine. There is no need to calculate the carry-out of plane 0. This has all taken 12 Copy Block instructions, five of which use the single plane source option.

Add with Saturation

Add with Saturation is exactly the same as add except that, if the result if larger than the maximum value, the result is forced to the maximum value (there is no overflow). The code is shown in addstemp. It is exactly the same as addtemp with an extra step at the end. We calculate the carry out of plane 0 (this is the overflow). This carry is ORed with all four planes, forcing the maximum for all pixels that have generated an overflow.

Subtract

The classical method of subtracting is "complement and add". The complement is a two's complement; we do a one's complement and force a carry into plane 3. This is shown in subtemp. The block temp3 is used as a source of 1s to force the carry into plane 3. The complement of the subtrahend is the very first "cpy". The carry is forced into plane 3. After that, it is identical to add (down to and including the comments).

Subtract with Saturation

This is identical to subtract except we calculate the carry from plane 0. Everywhere there is no carry, we force the result to 0. The listing is substemp.

Maximum

The two values corresponding to the pixel are compared and the one that is numerically larger is chosen. This is listing maxtemp. Copies of the two operands are saved and the source is subtracted from the destination. The carry-out of plane zero is used to select either the source or destination (from the saved copies). The result is left in the destination.

Minimum

The two values corresponding to the pixel are compared and the one that is numerically smaller is chosen. This is shown in mintemp. This is identical to maxtemp except for the ANDs which select the operands at the end.

Propagation of Carries Saturation between QPDMs

Clearly, in a multi-QPDM system, it is necessary to propagate the carry from the low-order QPDM to the high-order QPDM. It is also necessary to convey the final carry in the operations with saturate as well as maximum and minimum. This is done using the match logic. Observe that while this works, it is not especially fast.

1. Zero the destination plane
2. Set the listen bits for the source plane only.
3. Set the search color to ones in the source plane
4. Copy the destination to itself using source invert and match.

Everywhere else the source plane is a one, the destination will be forced to a one.

6.2.4 Transparency

TI has defined a logical operation called transparency. This involves executing copy block only for pixels where the source is not zero. This buys two things:

I. You can build up your destination in layers with the most recent information on top, providing a type of visual priority. (Using a straight Copy Block obscures the old layers).

II. No new colors are created which might distort the readability or change the intended meaning of color coded information. (Observe that using logical operations to merge information does just that. So does controlling the activity bits to write only selected planes).

The String instruction can do exactly what is called for here. If you use an SOAXZ field of 101 (Graphical Set) and a single plane font, then the character will be written into all planes wherever there is a one in the font. Any place there is a zero in the font, the pixel will not be written, allowing the data that is underneath to come through.

The drawing instructions (Line, Point, etc) also do this very well. Using a SOAXZ field of 101 (Graphical Set), the drawing color will be written into all bit planes everywhere the object exists. This also works properly for single plane PELs.

For Copy Block, the situation may be slightly more complex. If the source is a single plane, the Graphical Set does exactly the correct thing: everywhere the source is a one, all the planes of the destination will be written with the current drawing color.

It is more interesting if the source image contains several colors and the application wants to copy all of them to the destination without overlaying anything where the source is all zeroes. In this case, we cannot use a single plane source because of the multiple color situation. One method is:

1. Make a copy of the source (if it needs to be preserved).
2. Copy the destination to the source matching on a field of all zeroes.
3. Copy the source to the destination.

A third example involves the case where all pixels of a single color are to be copied from the source block, but no others. The solution involves the self-canceling effect of exclusive OR.

1. Copy the source to a temporary region.
2. Copy the source to temporary with logical XOR, matching on the desired color.
3. Copy the source to temporary with logical XOR. This leaves all pixels, except those of the desired color, at zero.
4. Choose a plane whose color bit for the desired color is a one and execute a single plane copy from temporary to the destination. Use Graphical Set with drawing color set to the desired color.

MAXTEMP

```
blk [blksiz]
act 0 {15}
cpy [dest] [temp4]                  ;save copy of original dest
cpy [source] [temp5]                ;and source

cpy ~ [source] [source]             ;invert the subtrahend

cpy [dest] [temp1]                  ;propagates
cpy o [source] [temp1]

cpy [dest] [temp2]                  ;generates
cpy a [source] [temp2]

cpy x [source] [dest]               ;half-adders

;now we have to propagate carries from 3 to 0
act 0 {8}                           ;will set all carries into plane three
cpy ~ o  [temp3] [temp3]            ;

act 0 {1}                           ;force carries into plane three
fnt 0: [nul] 0_[fnt0] 1_[fnt1]      ;carries come from plane zero
cpy 1| x [temp3] [dest]             ;final sum for plane three
cpy 1| a  [temp3] [temp1]           ;lower plane generate and this propagate
cpy o [temp1] [temp2]               ;ORd with this plane generate (final carry)

act 0 {2}                           ;write plane two only
fnt 3: [nul] 0_[fnt0] 1_[fnt1]      ;plane 3 is Source
cpy 1| x [temp2] [dest]             ;final sum for plane two
cpy 1| a  [temp2] [temp1]           ;lower plane generate and this propagate
cpy o [temp1] [temp2]               ;ORd with this plane generate (final carry)

act 0 {4}                           ;write plane one only
fnt 2: [nul] 0_[fnt0] 1_[fnt1]      ;plane 2 is Source
cpy 1| x [temp2] [dest]             ;final sum for plane one
cpy 1| a [temp2] [temp1]            ;lower plane generate and this propagate
cpy o [temp1] [temp2]               ;ORd with this plane generate (final carry)

act 0 {8}                           ;write plane zero only
fnt 1: [nul] 0_[fnt0] 1_[fnt1]      ;plane 1 is Source
cpy 1| x [temp2] [dest]             ;final sum for plane zero

;now we do the saturation part
;look for a carry out of plane zero
```

```
cpy 1| a [temp2] [temp1]          ;lower plane generate and this propagate
cpy o [temp1] [temp2]             ;final carry from plane zero

act 0 {15}                        ;all activity bits
fnt 0: [nul] 0_[fnt0] 1_[fnt1]    ;and will force all zeroes where it isn't
cpy 1| a [temp2] [temp4]          ;keep the maximums
cpy 1| ~ a [temp2] [temp5]        ;from each of the two images

cpy [temp4] [dest]                ;and merge them together
cpy o [temp5] [dest]              ;in the final destination
```

MINTEMP

```
blk [blksiz]
act 0 {15}
cpy [dest] [temp4]                  ;save copy of original dest
cpy [source] [temp5]                ;and source

cpy ~ [source] [source]             ;invert the subtrahend

cpy [dest] [temp1]                  ;propagates
cpy o [source] [temp1]

cpy [dest] [temp2]                  ;generates
cpy a [source] [temp2]

cpy x [source] [dest]               ;half-adders

;now we have to propagate carries from 3 to 0
act 0 {8}                           ;will set all carries into plane three
cpy ~ o  [temp3] [temp3]            ;

act 0 {1}                           ;force carries into plane three
fnt 0: [nul] 0_[fnt0] 1_[fnt1]      ;carries come from plane zero
cpy 1| x [temp3] [dest]             ;final sum for plane three
cpy 1| a  [temp3] [temp1]           ;lower plane generate and this propagate
cpy o [temp1] [temp2]               ;ORd with this plane generate (final carry)

act 0 {2}                           ;write plane two only
fnt 3: [nul] 0_[fnt0] 1_[fnt1]      ;plane 3 is Source
cpy 1| x [temp2] [dest]             ;final sum for plane two
cpy 1| a  [temp2] [temp1]           ;lower plane generate and this propagate
cpy o [temp1] [temp2]               ;ORd with this plane generate (final carry)

act 0 {4}                           ;write plane one only
fnt 2: [nul] 0_[fnt0] 1_[fnt1]      ;plane 2 is Source
cpy 1| x [temp2] [dest]             ;final sum for plane one
cpy 1| a [temp2] [temp1]            ;lower plane generate and this propagate
cpy o [temp1] [temp2]               ;ORd with this plane generate (final carry)

act 0 {8}                           ;write plane zero only
fnt 1: [nul] 0_[fnt0] 1_[fnt1]      ;plane 1 is Source
cpy 1| x [temp2] [dest]             ;final sum for plane zero

;now we do the saturation part
;look for a carry out of plane zero
```

```
cpy 1| a [temp2] [temp1]          ;lower plane generate and this propagate
cpy o [temp1] [temp2]             ;final carry from plane zero

act 0 {15}                        ;all activity bits
fnt 0: [nul] 0_[fnt0] 1_[fnt1]    ;and will force all zeroes where it isn't
cpy 1| ~ a [temp2] [temp4]        ;keep the minimums
cpy 1| a [temp2] [temp5]          ;from each of the two images

cpy [temp4] [dest]                ;and merge them together
cpy o [temp5] [dest]              ;in the final destination
```

ADDSTEMP

```
blk [blksiz]
act 0 {15}

cpy [dest] [temp1]                  ;propagates
cpy o [source] [temp1]

cpy [dest] [temp2]                  ;generates
cpy a [source] [temp2]

cpy x [source] [dest]               ;half-adders

;now we have to propagate carries from 3 to 0

act 0 {2}                           ;write plane two only
fnt 3: [nul] 0_[fnt0] 1_[fnt1]      ;plane 3 is Source
cpy 1| x [temp2] [dest]             ;final sum for plane two
cpy 1| a  [temp2] [temp1]           ;lower plane generate and this propagate
cpy o [temp1] [temp2]               ;ORd with this plane generate (final carry)

act 0 {4}                           ;write plane one only
fnt 2: [nul] 0_[fnt0] 1_[fnt1]      ;plane 2 is Source
cpy 1| x [temp2] [dest]             ;final sum for plane one
cpy 1| a [temp2] [temp1]            ;lower plane generate and this propagate
cpy o [temp1] [temp2]               ;ORd with this plane generate (final carry)

act 0 {8}                           ;write plane zero only
fnt 1: [nul] 0_[fnt0] 1_[fnt1]      ;plane 1 is Source
cpy 1| x [temp2] [dest]             ;final sum for plane zero

;now we look for a carry out of plane zero
cpy 1| a [temp2] [temp1]            ;lower plane generate and this propagate
cpy o [temp1] [temp2]               ;ORd with this plane generate (final carry)

act 0 {15}
fnt 0: [nul] 0_[fnt0] 1_[fnt1]      ;and will force all ones with it
cpy 1| o [temp2] [dest]             ;(forcing result to saturated)
```

ADDTEMP

```
blk [blksiz]
act 0 {15}

cpy [dest] [temp1]                  ;propagates
cpy o [source] [temp1]

cpy [dest] [temp2]                  ;generates
cpy a [source] [temp2]

cpy x [source] [dest]               ;half-adders

;now we have to propagate carries from 3 to 0

act 0 {2}                           ;write plane two only
fnt 3: [nul] 0_[fnt0] 1_[fnt1]      ;plane 3 is Source
cpy 1| x [temp2] [dest]             ;final sum for plane two
cpy 1| a  [temp2] [temp1]           ;lower plane generate and this propagate
cpy o [temp1] [temp2]               ;ORd with this plane generate (final carry)

act 0 {4}                           ;write plane one only
fnt 2: [nul] 0_[fnt0] 1_[fnt1]      ;plane 2 is Source
cpy 1| x [temp2] [dest]             ;final sum for plane one
cpy 1| a [temp2] [temp1]            ;lower plane generate and this propagate
cpy o [temp1] [temp2]               ;ORd with this plane generate (final carry)

act 0 {8}                           ;write plane zero only
fnt 1: [nul] 0_[fnt0] 1_[fnt1]      ;plane 1 is Source
cpy 1| x [temp2] [dest]             ;final sum for plane zero
```

SUBTEMP

```
blk [blksiz]
act 0 {15}

cpy ~ [source] [source]              ;invert the subtrahend

cpy [dest] [temp1]                   ;propagates
cpy o [source] [temp1]

cpy [dest] [temp2]                   ;generates
cpy a [source] [temp2]

cpy x [source] [dest]                ;half-adders

;now we have to propagate carries from 3 to 0
act 0 {8}                            ;will set all carries into plane three
cpy ~ o  [temp3] [temp3]             ;

act 0 {1}                            ;force carries into plane three
fnt 0: [nul] 0_[fnt0] 1_[fnt1]       ;carries come from plane zero
cpy 1| x [temp3] [dest]              ;final sum for plane three
cpy 1| a  [temp3] [temp1]            ;lower plane generate and this propagate
cpy o [temp1] [temp2]                ;ORd with this plane generate (final carry)

act 0 {2}                            ;write plane two only
fnt 3: [nul] 0_[fnt0] 1_[fnt1]       ;plane 3 is Source
cpy 1| x [temp2] [dest]              ;final sum for plane two
cpy 1| a  [temp2] [temp1]            ;lower plane generate and this propagate
cpy o [temp1] [temp2]                ;ORd with this plane generate (final carry)

act 0 {4}                            ;write plane one only
fnt 2: [nul] 0_[fnt0] 1_[fnt1]       ;plane 2 is Source
cpy 1| x [temp2] [dest]              ;final sum for plane one
cpy 1| a [temp2] [temp1]             ;lower plane generate and this propagate
cpy o [temp1] [temp2]                ;ORd with this plane generate (final carry)

act 0 {8}                            ;write plane zero only
fnt 1: [nul] 0_[fnt0] 1_[fnt1]       ;plane 1 is Source
cpy 1| x [temp2] [dest]              ;final sum for plane zero
```

SUBSTEMP

```
blk [blksiz]
act 0 {15}

cpy ~ [source] [source]              ;invert the subtrahend

cpy [dest] [temp1]                   ;propagates
cpy o [source] [temp1]

cpy [dest] [temp2]                   ;generates
cpy a [source] [temp2]

cpy x [source] [dest]                ;half-adders

;now we have to propagate carries from 3 to 0
act 0 {8}                            ;will set all carries into plane three
cpy ~ o  [temp3] [temp3]             ;

act 0 {1}                            ;force carries into plane three
fnt 0: [nul] 0_[fnt0] 1_[fnt1]       ;carries come from plane zero
cpy 1| x [temp3] [dest]              ;final sum for plane three
cpy 1| a  [temp3] [temp1]            ;lower plane generate and this propagate
cpy o [temp1] [temp2]                ;ORd with this plane generate (final carry)

act 0 {2}                            ;write plane two only
fnt 3: [nul] 0_[fnt0] 1_[fnt1]       ;plane 3 is Source
cpy 1| x [temp2] [dest]              ;final sum for plane two
cpy 1| a  [temp2] [temp1]            ;lower plane generate and this propagate
cpy o [temp1] [temp2]                ;ORd with this plane generate (final carry)

act 0 {4}                            ;write plane one only
fnt 2: [nul] 0_[fnt0] 1_[fnt1]       ;plane 2 is Source
cpy 1| x [temp2] [dest]              ;final sum for plane one
cpy 1| a [temp2] [temp1]             ;lower plane generate and this propagate
cpy o [temp1] [temp2]                ;ORd with this plane generate (final carry)

act 0 {8}                            ;write plane zero only
fnt 1: [nul] 0_[fnt0] 1_[fnt1]       ;plane 1 is Source
cpy 1| x [temp2] [dest]              ;final sum for plane zero

;now we do the saturation part
;look for a carry out of plane zero

cpy 1| a [temp2] [temp1]             ;lower plane generate and this propagate
cpy o [temp1] [temp2]                ;final carry from plane zero

act 0 {15}                           ;all activity bits
fnt 0: [nul] 0_[fnt0] 1_[fnt1]       ;and will force all zeroes where it isn't
cpy 1| a [temp2] [dest]
```

ADVANCED MICRO DEVICES' NORTH AMERICAN SALES OFFICES

ALABAMA ..... (205) 882-9122
ARIZONA ..... (602) 242-4400
CALIFORNIA,
Culver City ..... (213) 645-1524
Newport Beach ..... (714) 752-6262
San Diego ..... (619) 560-7030
San Jose ..... (408) 249-7766
Santa Clara ..... (408) 727-3270
Woodland Hills ..... (818) 992-4155
CANADA, Ontario,
Kanata ..... (613) 592-0060
Willowdale ..... (416) 224-5193
COLORADO ..... (303) 741-2900
CONNECTICUT ..... (203) 264-7800
FLORIDA,
Clearwater ..... (813) 530-9971
Ft Lauderdale ..... (305) 776-2001
Melbourne ..... (305) 729-0496
Orlando ..... (305) 859-0831
GEORGIA ..... (404) 449-7920
ILLINOIS,
Chicago ..... (312) 773-4422
Naperville ..... (312) 505-9517
INDIANA ..... (317) 244-7207
KANSAS ..... (913) 451-3115
MARYLAND ..... (301) 796-9310
MASSACHUSETTS ..... (617) 273-3970
MINNESOTA ..... (612) 938-0001
MISSOURI ..... (913) 451-3115
NEW JERSEY ..... (201) 299-0002
NEW YORK,
Liverpool ..... (315) 457-5400
Poughkeepsie ..... (914) 471-8180
Woodbury ..... (516) 364-8020
NORTH CAROLINA ..... (919) 878-8111
OHIO ..... (614) 891-6455
Columbus ..... (614) 891-6455
Dayton ..... (513) 439-0470
OREGON ..... (503) 245-0080
PENNSYLVANIA,
Allentown ..... (215) 398-8006
Willow Grove ..... (215) 657-3101
TEXAS,
Austin ..... (512) 346-7830
Dallas ..... (214) 934-9099
Houston ..... (713) 785-9001
WASHINGTON ..... (206) 455-3600
WISCONSIN ..... (414) 792-0590

ADVANCED MICRO DEVICES' INTERNATIONAL SALES OFFICES

BELGIUM,
Bruxelles ..... TEL ..... (02) 771 91 42
FAX ..... (02) 762 37 12
TLX 61028
FRANCE,
Paris ..... TEL ..... (1) 49-75-10-10
FAX ..... (1) 49-75-10-13
TLX 263282
WEST GERMANY,
Hannover area ..... TEL ..... (05143) 50 55
FAX ..... (05143) 55 53
TLX 925287
München ..... TEL ..... (089) 41 14-0
FAX ..... (089) 406490
TLX 523883
Stuttgart ..... TEL ..... (0711) 62 33 77
FAX ..... (0711) 625187
TLX 721882
HONG KONG,
Kowloon ..... TEL 852-3-695377
FAX 852-123-4276
TLX .. 504260AMDAPHX
ITALY, Milano ..... TEL ..... (02) 3390541
(02) 3533241
FAX ..... (02) 3498000
TLX 315286
JAPAN,
Kanagawa ..... TEL 462-47-2911
FAX 462-47-1729
Tokyo ..... TEL ..... (03) 345-8241
FAX ..... (03) 342-5196
TLX ... J24064AMDTKOJ
Osaka ..... TEL 06-243-3250
FAX 06-243-3253
KOREA, Seoul ..... TEL 82-2-784-7598
FAX 82-2-784-8014
LATIN AMERICA,
Ft. Lauderdale ..... TEL ..... (305) 484-8600
FAX ..... (305) 485-9736
TLX .. 5109554261 AMDFTL
NORWAY,
Hovik ..... TEL ..... (02) 537810
FAX ..... (02) 591959
TLX 79079
SINGAPORE ..... TEL 65-2257544
FAX 2246113
TLX ..... RS55650 MMI RS
SWEDEN, Stockholm ..... TEL ..... (08) 733 03 50
FAX ..... (08) 733 22 85
TLX 11602
TAIWAN ..... TLX 886-2-7122066
FAX 886-2-7122017
UNITED KINGDOM,
Farnborough ..... TEL ..... (0252) 517431
FAX ..... (0252) 521041
TLX 858051
Manchester area ..... TEL ..... (0925) 828008
FAX ..... (0925) 827693
TLX 628524
London area ..... TEL ..... (04862) 22121
FAX ..... (0483) 756196
TLX 859103

NORTH AMERICAN REPRESENTATIVES

CALIFORNIA
I^2 INC ..... OEM (408) 988-3400
DISTI (408) 498-6868
CANADA
Burnaby, B.C.
DAVETEK MARKETING ..... (604) 430-3680
Calgary, Alberta
VITEL ELECTRONICS ..... (403) 278-5833
Kanata, Ontario
VITEL ELECTRONICS ..... (613) 592-0090
Mississauga, Ontario
VITAL ELECTRONICS ..... (416) 676-9720
Quebec
VITEL ELECTRONICS ..... (514) 636-5951
IDAHO
INTERMOUNTAIN TECH MKGT ..... (208) 888-6071
INDIANA
ELECTRONIC MARKETING
CONSULTANTS, INC. ..... (317) 253-1668
IOWA
LORENZ SALES ..... (319) 377-4666
KANSAS
LORENZ SALES ..... (913) 384-6556
KENTUCKY
ELECTRONIC MARKETING
CONSULTANTS, INC. ..... (317) 253-1668
MICHIGAN
SAI MARKETING CORP ..... (313) 750-1922
MISSOURI
LORENZ SALES ..... (314) 997-4558
NEBRASKA
LORENZ SALES ..... (402) 475-4660
NEW MEXICO
THORSON DESERT STATES ..... (505) 293-8555
NEW YORK
NYCOM, INC ..... (315) 437-8343
OHIO
Centerville
DOLFUSS ROOT & CO ..... (513) 433-6776
Columbus
DOLFUSS ROOT & CO ..... (614) 885-4844
Strongsville
DOLFUSS ROOT & CO ..... (216) 238-0300
PENNSYLVANIA
DOLFUSS ROOT & CO ..... (412) 221-4420
UTAH
R^2 MARKETING ..... (801) 595-0631

ADVANCED MICRO DEVICES 901 Thompson Pl., P.O. Box 3453, Sunnyvale, CA 94088, USA
TEL: (408) 732-2400 • TWX: 910-339-9280 • TELEX: 34-6306 • TOLL FREE: (800) 538-8450
APPLICATIONS HOTLINE TOLL FREE: (800) 222-9323
Printed in U.S.A.

PEP-WCP-11M-3/88-1

ADVANCED
MICRO
DEVICES
901 Thompson Place
P.O. Box 3453
Sunnyvale
California 94088-3453
(408) 732-2400
TELEX: 34-6306
TOLL FREE (800) 538-8450
APPLICATIONS HOTLINE
(800) 222-9323

09682B