
Using a PCI Bus as the I/O Bus on an Am29030/040^(TM) Microprocessor Design



Application Note

This application note describes how the Peripheral Component Interconnect (PCI) bus can be used as the I/O bus portion on a two-bus microcontroller design. Additionally, one programmable logic part, the MACH®220 device or the MACH®465, is added for the entire control logic for both the memory control and the signal conversion between the Am29030/040TM microprocessor and Revision 2.0 of the PCI I/O bus.

INTRODUCTION

The Peripheral Component Interconnect (PCI) local bus is a high-performance, 32-bit or 64-bit bus with multiplexed address and data lines. It is intended for use as an interconnect mechanism between highly integrated peripheral controller components, peripheral add-in boards, and processor/memory systems. Because the bus is multiplexed, the number of pin contacts is reduced, making the peripheral cheaper to implement. To accomplish this, PCI adds a layer between the CPU and the peripherals, resulting in a processor-independent bus that can be used for a variety of CPUs and processor speeds.

The PCI bus is a well-defined interface that is specified for the signals as well as the physical characteristics of the connector and the loading. Since inexpensive peripherals have become available to meet PC needs, it would therefore be advantageous to extend the use of the PCI bus into the embedded processor market. Since the 29KTM Family is one of the premiere RISC families used in embedded control, this application note shows how easily the PCI bus can be adapted to the 29K Family processor to meet the requirement of an I/O bus.

THE PCI BUS

In order to create an industry standard for PCI, the PCI Special Interest Group (SIG) has defined a specification. The *PCI Local Bus Specification*, Revision 2.0, defines the protocol, electrical, mechanical, and configuration specifications for PCI local bus components and expansion boards. At the writing of this application note Revision 2.1 of the PCI specification is close to adoption. The changes from the Revision 2.0 are relatively minor in the hardware sections and in fact this design is a core subset of the 2.0 and now the 2.1 that there are no foreseen changes to the design to meet the 2.1 specification also. (See page 5 for information on ordering the specification.)

Basics of the PCI Bus

The PCI bus is an address/data multiplexed bus. Additionally, the control signals are multiplexed with the byte control signals. The PCI bus supports a burst protocol that accepts an address with multiple data packets. This protocol is identical to the 29K Family, making the interface design easily accomplished.

Bus Mastering on the PCI bus is considered the normal means of moving large amounts of data so arbitration for the PCI bus can range from a simple design of direct priority to a more complicated design of using arbitration under the existing bus master for the next bus master depending on the performance goal of the design. The bus mastering protocol also supports bus master preemption if desired. Additionally, complex caching cycles for multiprocessors also are defined. With forethought, the PCI SIG specified that these features be optional and not required.

Use of the PCI Bus in this Design

This design focuses on the minimum requirements of the PCI bus, with the exception of parity. Currently, parity is not supported on this design because of the complexity of the chips that would need to be added to the design. The bus master arbitration used in this design is limited to the Am29030/040 processor “parked” on the PCI bus being the main master, and the peripheral chips requesting the bus as necessary. No support for time-out and bus master preemption is included, so it is the responsibility of each master to ensure that each peripheral does not use the bus to the exclusion of the other peripherals.

The memory design is taken directly from the original EZ-030 demonstration board (see the *EZ-030 Demonstration Board Theory of Operation* application note), except that the MACH device replaces all the discrete PAL® devices of the EZ-030 board design. Since the EZ-030 board supports a memory clock (MEMCLK) of 16 MHz, this application note also focuses at 16 MHz for both the memory and the PCI bus clock. The PCI bus clock is defined from 0 to 33 MHz, so this falls within that limit. Also, 33-MHz clock rates, and therefore the bus, require more careful layout than the 16-MHz clock.

The original application note presented a MACH220, 4 PALs and 8 octal buffers to implement the design. This update of that original application note also adds a new MACH part called the MACH465. The MACH465 has enough I/O in one chip to replace both the MACH220 and the 12 octal parts in one convenient surface mount package. Additionally the MACH465 is JTAG programmable so changes can be done without removing the part from the PCB making user design changes for specific system implementations easily done either on the real embedded system or the demo board. This MACH465 design has now been actually developed to a working unit and is available from AMD at a nominal cost.

The rest of this application note will first focus on the MACH220 device and the extra circuitry needed to implement the control portion of the PCI bus. Schematics for this design are shown in Appendix A and the PAL equations are listed in Appendix B. Then this application note will show how to combine the 13 parts into one MACH465 where board space is more critical than cost, speed or power. The MACH465 schematics are then given in Appendix C and the MACH465 design equations are given in Appendix D.

Appendix E contains the equations to a design that uses the internal registers of the MACH465 to implement a pipeline method of operation so the frequency of the PCI bus could be increased to 33 MHz. However the memory system of this design is only good with a 29040-33 and 60 ns DRAMs to 20 MHz but if it were changed to a bank interleaved design 33 MHz system operation could be realized.

THEORY OF OPERATIONS

As discussed in the introduction, this design is a modification of the original EZ-030 board design. The major change is that the three control PAL devices of the EZ-030 board design are merged into one large control PAL device, the MACH220 device, shown in the schematic on page 8. Pages 9 through 12 show the need of multiplexing the address and data bus together, and pages 13 through 15 depict the three separate PCI connectors.

Figures 1 and 2 show block diagrams for the original and modified board designs, respectively.

The PCI bus is a multiplexed address, data, and control bus to conserve on pins. The Am29030/040 processor is not a multiplexed address and data bus part, so in the schematic on page 12, the 12 octal parts required to do the address/data multiplexing are shown. U20 through U23, 74F244s, support placing the initial address in the first clock cycle of the frame. (Note that the upper four bits, A31-A28, are driven onto the PCI bus with 0s. These bits are used on the local Am29030/040 processor side to decode the various PCI bus cycles that are to be accomplished.)

U16 through U19, 74F245s, are the data buffers used to input or output the data off or onto the PCI bus. These parts will be used for Am29030/040 processor PCI bus cycles, as well as when another PCI bus master is using the DRAM memory.

If another bus master takes over the PCI bus and wants to perform a DRAM memory cycle, the parts U24 through U27 capture the address on the first clock cycle of the FRAME signal. The parts are 20L8s used to capture addresses A31-A10, A1, and A0 in a clock-enabled register configuration. The equations for the 20L8s are in Appendix B. The address bits A9-A2 are captured in a 22V10, which is a combination clock-enabled register and a counter that supports the burst feature that PCI has defined and enables the maximum throughput allowed. This PAL device also detects an upper address bit carry if the requesting master tries to burst across a 1K address boundary, which is then used by the PCI controller to signal a STOP condition on the PCI bus. This then is the full address and data support for both bus master and bus slave.

The schematics shown on pages 13 through 15 of this document show the PCI connectors. With the exception of the interrupts and the bus request/grant signals being point to point, all the other signals are bused in common.

Each slot is given its own interrupt to the processor from the INTA of the PCI bus. Interrupts on the PCI bus are negative True and level sensitive so they fit nicely to the Am29030/040 processor interrupt input structure without a need for additional interrupt controllers. Each bus request and grant is given to the MACH220 device to arbitrate the bus. All the power pins of the PCI connector in this design are for 5 V; 3.3 V is not considered.

MACH220 DEVICE

The heart of the design is the MACH220 device, shown in the schematic on page 8. Internally this part can be subdivided into the following categories:

- Bus arbitration
- PCI control signals
- Memory control
- RAS and CAS decode
- Refresh timer

Each of these parts is detailed in the following sections.

Bus Arbitration

The bus arbiter in this design “parks” the Am29030/040 processor on the bus with a PROC_BGRT unless any of the PCI_BREQx's become valid. When a PCI_BREQx becomes valid, then the PROC_BGRT is lowered and the arbiter waits for the REQ (Am29030/040 processor memory request) to be released for at least 2 clock cycles. At that point, the highest level PCI_BREQx is arbitrated and its corresponding PCI_BGRTx is issued. This grant is held until the PCI_BREQx goes away, the PCI_FRAME is released, and the last IRDY and TRDY have been issued. If another PCI_BREQx is active, then the arbiter will move to the highest new PCI_BREQx. If no other PCI_BREQx is active, then the PROC_BGRT is issued again and the whole cycle starts over. No attempt is made to make a rotating-priority arbiter, although that could be accomplished in the buried macros of the MACH220 device. This arbiter simply functions as a fixed-priority level arbiter.

PCI Control Signals

This section has the largest number of equations. Some of them are for support when the Am29030/040 processor is the bus master, while some of them are for the slave memory interface. Address bit A31 is the magic control bit that determines whether the Am29030/040 processor stays local to the memory subsystem or goes to the PCI bus. If A31=0, then the Am29030/040 processor will stay in local memory section; if A31=1, then an external PCI access is started.

When the Am29030/040 processor is a bus master and goes to the PCI bus to access a peripheral, A31=1 and PROC_BGRT is True. PROC_BGRT controls the three-state of the PCI_FRAME, PCI_IRDY, and PCI_C_BEX control lines.

If A31=1 and PROC_BGRT is True, a PCI_FRAME signal is generated on the first line of the equations and is then held True if the Am29030/040 processor is bursting on the bus (BURST True).

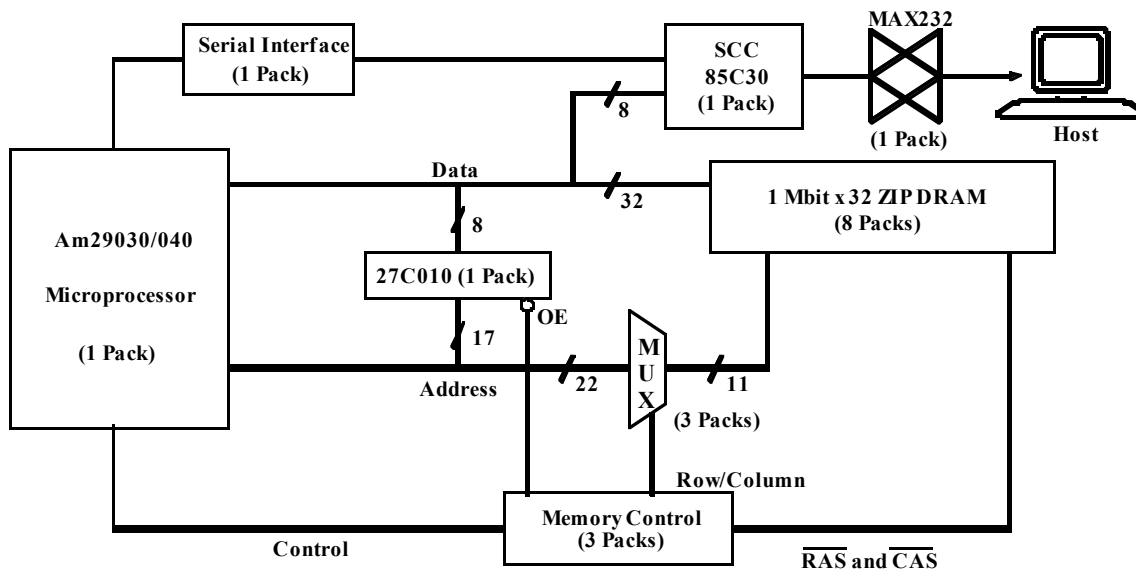


Figure 1. EZ-030 Demonstration Board Block Diagram

18468A-1

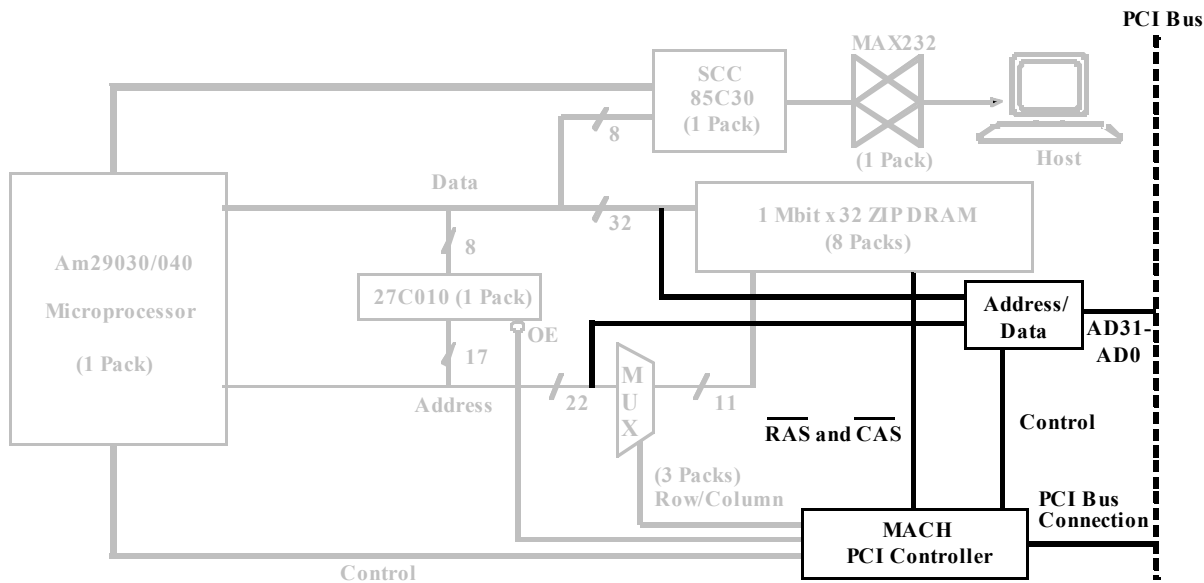


Figure 2. Modified EZ-030 Demonstration Board, With PCI I/O

18468A-2

The Am29030/040 processor address is enabled onto the PCI bus with the term ADD_EN during the time that PCI_FRAME is True while PCI_FRAME_D (delayed by one clock PCI_FRAME) is False. This represents the first cycle on the PCI bus. During this cycle, the PCI_C_BEx lines output the appropriately decoded control information. The only cycles this design supports in master mode are:

- Memory reads and writes
(A31=1, A30=0, A29=0)
- Configuration reads and writes
(A31=1, A30=1, A29=X)
- I/O reads and writes
(A31=1, A30=0, A29=1)

Configuration reads and writes also need a PCI_IDSEL at each slot which is driven with A13 for slot 1, A14 for slot 2 and A15 for slot3.

When PCI_FRAME_D becomes True, control goes to the data portion of the cycle (PCI_DATA_CYCLE is True) and the data bus is enabled with DATA_EN. PCI_FRAME_D stays True until the last PCI_TRDY is generated. The data direction is handled by DATA_DIR. All PCI_C_BEx signals are driven True during the data cycle if the cycle is a read, or they are driven with the WEx off the Am29030/040 processor during a write to the PCI bus. On the PCI bus, a device indicates that it has been selected by asserting a signal PCI_DEVSEL. However, if no device responds, the Am29030/040 processor does not terminate the transaction because no PROC_RDY is asserted and there is a bus lock. To break this lock, a default timer is put into this design in the form of a 3-bit counter (CT0, CT1, CT2) that gives the peripherals 8 clock cycles to respond. If no PCI_DEVSEL is returned, then an Am29030/040 processor PROC_ERR is generated. Additionally, PROC_ERR is generated in normal cycles if the responding peripheral also asserts PCI_STOP. During the data cycle, PCI_IRDY is driven True at all times and PCI_TRDY is passed to the processor on PROC_RDY to signal the end of any given cycle on the PCI bus.

For a PCI master access, a similar transaction takes place in reverse if the incoming PCI address bus bit A31=1. Every address issued on the PCI bus is captured in the address register by the ADD_CLK line being active with PCI_FRAME set to True and PCI_FRAME_D set to False. The address register (ADD_REG_EN) is always enabled onto the Am29030/040 processor address bus if any of the PCI_BGRTx signals are True, which means the Am29030/040 processor does not control its address bus.

If the PCI cycle is for the local memory system, then the internal node DEV_SEL_NODE goes active, thereby activating the three-state of the control for PCI_STOP, PCI_DEVSEL, and PCI_TRDY, and enabling the data bus drivers through the DATA_EN term. Since the PCI bus gives the condition for read and write under the control portion of the PCI bus, and does not keep a write line active through the whole cycle, this fact is captured on an internal node INTERNAL_WRITE. This then controls the direction of the data buffers during a PCI master cycle. INTERNAL_WRITE also drives the WRITE line of the processor for the memory subsystem during this time. PCI_TRDY then becomes MEM_RDY as the state machine for the local memory goes through its paces. If bursting is being done, the ADD_INC signal tells the 22V10 the A9-A2 captured address to increment to the next address. If roll over is detected in the 22V10 by the ADD_STOP signal, then a PCI_STOP is asserted, which says a new bus address is needed.

Memory Control

Memory control becomes almost a direct graft of that described in the *EZ-030 Demonstration Board Theory of Operation* application note. The same state machines are implemented with the exception that accommodation is now made for more than one master to access memory. REF_ACCESS still has top priority to do its job to refresh the memory array. IDLE has more terms added to it to support the processor doing a local memory cycle and the PCI bus doing its master cycle. PCI bus is indicated by the PROC_BGRT set to False and the PCI_DATA_CYCLE set to True. MEM_ACCESS is started in the same manner as before and held as long as burst is True for the Am29030/040 processor.

PCI, though, indicates that it is on its last cycle by deasserting PCI_FRAME, which has a similar control to the Am29030/040 processor BURST signal. However, for a complete cycle to take place, both TRDY and IRDY on the PCI bus must be True so this is taken into account in the equations for MEM_ACCESS, MEM_RDY, and ADD_INC. If at any time, the PCI bus suspends the memory access, then reads and writes occur to the same address and are simply repeated because this is a single-cycle memory unit.

RAS and CAS Decode

The RAS and CAS decode portion of the equations are essentially the same as in the EZ-030 board application note, with the exception that a new equation was added to account for the source of the byte enable control. When a local cycle is performed from the Am29030/040 processor, the byte enables come from the processor on the WEx pins. During a PCI cycle, they are issued on the PCI_C_BEx pins on the PCI bus interface. Internal write is used as the read/write indication

(as was mentioned before) because the PCI bus transfers the state of the transfer on the first cycle and then does not retain a write line for the remainder of the transaction.

Refresh Timer

The refresh timer is exactly as described in the *EZ-030 Demonstration Board Theory of Operation* application note, consisting of a 7-bit timer with the eighth bit serving as a REFRESH_REQ bit, which remembers that a refresh needs to be arbitrated by the main memory system arbiter. The counter will continue counting towards the next refresh interval.

Serial Port

If used, the serial port can be exactly the same as described in the *EZ-030 Demonstration Board Theory of Operation* application note.

TIMING AND WORST-CASE ISSUES

This design assumes that the PGA Am29030/040 processor's Scaleable Clocking™ feature is used, resulting in a 32-MHz processor with a 16-MHz external memory and PCI bus interface. The worst-case timing in this design then occurs in the memory system around a single-cycle read at 16 MHz. The delay must be set at 20 ns for the CAS pulse generation while 12 ns is the best time of a MACH220 device for the combinatorial delay, making CAS fall at 32 ns into the cycle. CAS access time is 20 ns. A set up of 9 ns makes an access time of 61 ns for a 62.5-ns cycle at 16 MHz.

The data path of this design between the processor side and the PCI side is basically a flow-through style of operation. This is chosen for the ease of design but also for the higher amount of data throughput. This does put the 74F245 buffers in the speed path for burst access for the PCI bus master to the DRAM. Based on the above equations the PCI setup is 7 ns instead of 9 ns of the 29030/040 and we had a margin of 1.5 ns for the calculations. This still leaves us 1.5 ns beyond the 62.5 ns. AMD specifies the MACH220 at 12 ns for 24 outputs switching simultaneously while in this case we only have 4 outputs switching simultaneously. Therefore we have a -10 ns instead of a -12 ns part making up the 1.5 ns needed.

If additional setup or CAS access time is needed, then the RAS/CAS decode can be accomplished in a separate, faster decode PAL device. The DELAY signal from the delay line can be used in this PAL device, with the other state lines going to the faster decode PAL device. Additionally, faster address multiplexers can be used, such as 74F or AS parts, instead of the 74LS157s. This reduces the delay from 12 ns to 6 ns and decreases the delay line requirement to 15 ns, gaining an additional 5 ns on CAS access. Use of these faster 74F/AS157s then requires 33-ohm series dampening resistors before the DRAMs.

The PCI bus from the initiator means that the MACH220 side meets the setup times up to 25 MHz, but the CLK to Q delays on the beginning of edges are not to the delay specification. This is really not of consequence to this design because the bus is running slower than the maximum of 33 MHz. This slower speed allows a more relaxed timing as well as board layout in the final analysis. The rest of the paths, including to and from the processor and the surrounding control logic, have wide margins of setup and hold that can be used.

PCI BUS SIGNAL QUALITY

The PCI bus is defined to be a current-driven reflected-wave transmission line. However, all the drivers (74F24x and MACH device) used in this design to drive the PCI bus signals are voltage drivers, and therefore, incident-wave drivers. Reflected-wave drivers have to settle the bus and therefore have two times the transmission line distance to settle, but incident-wave drivers have only one times the transmission line time distance. This makes up for some of the clock-to-Q time of the 74F24x parts. The input threshold levels and the final output drive levels on the PCI bus are TTL-compatible, which means the 74F24x and MACH device drivers are compatible. When laying out the design, the best layout is the MACH device, buffers, and PAL devices on one end of the transmission line, and the sockets for the PCI bus in line towards the opposite end.

The PCI bus allows 10 loads of 10 pF per load maximum at 33 MHz. This design uses six loads for the three slots. The three-way load of the 74F24x and PAL devices then represent about 35 pF, making up the other four loads. At 16 MHz, though, this requirement could be easily relaxed if needed.

THE MACH465 DESIGN DIFFERENCES

When the initial application note was written that MACH family had not been extended in I/O pin count beyond 64 I/O cells which was not sufficient to encompass the whole PCI bus and the full 32 bit address/data bus of the 29030/040 processor. With the advent of the MACH465 that became a reality with now 128 I/O and 14 input pins available to the design. Just the processor address/data pins and the PCI bus AD pins require 96 pins alone. This has allowed what was in the MACH220, the 4 PALs and the 8 octal parts to now be combined into 1 IC. The MACH family has been certified by AMD to be PCI compliant in its members with the MACH465-12 to support the 33 MHz PCI bus. Since only 1 IC is now presenting the load to the PCI bus the previous section on loading has now been done away with. The I/O buffers of the MACH465 also have lower DC loading capability than did the F24X or the PALs which help in the matter of driving the open ended PCI bus smoothly with less undershoot involved in the layout issues.

The combining of all the parts into 1 was a very straight forward process in that the way the parts were interconnected essentially mirrored the internal distribution matrix of a MACH family part. Since some pins were not needed these pins became what is called "STRING" statements in PALASM and then became wider substitutions in the full macrocell equation set. An example of this is the ADD_CLK function to provide a time when the address is captured off the PCI bus under the first clock cycle of a FRAME signal. The tristate enables used by the data buffers and address buffers have been simply incorporated into the .TRST equations for the respective output. Because I/O is limited in this design some un-needed pins for output had to become buried nodes in this design. This does make debug harder but that is why simulation is important. The rest of the equations that were in the MACH220 have been placed in the MACH465 almost unchanged.

TIMING AND WORST-CASE ISSUES FOR THE MACH465

As far as worst case timing of the signals in this design versus the original MACH220 design goes, little changes. The MACH220 was considered to be a -15 because as far as timing at 16 MHz the -15 was sufficient for both clock to Q and setup for easily up to 20 MHz by itself. However the 74F245 parts for the data buffers were chosen for their low propagation delay of 6 ns. However in the expense of pins the MACH465 is now in place of these parts and now the propagation delay has increased to 12 ns. This extra 6 ns makes the worst case path be found in Bus Master accesses from the PCI side in the burst mode. In this mode the MACH465 enters into the equation twice. The equation is given by:

$$62.5 \leq \text{Delay}_{in} + \text{MACH465}(\text{CAS generation}) + \text{CAS access} + \text{MACH465}(\text{Data Buffer}) + \text{PCI setup}$$

This gives the following numbers:

$$20 + 12 + 20 + 12 + 7 = 71 \text{ ns}$$

Even a -10 MACH465 will not solve the problem but a faster DRAM would. 60 ns DRAM can tolerate 10 ns CAS recovery time so the first term could be lowered to 15 ns with no problem and the 60 ns DRAMs give 15 ns CAS access time giving the needed margin. Another consideration is the fact that the MACH family has been specified at 1/2 their outputs switching simultaneously as a worst case and if only one output switches the timing is better by about 2 ns. With this being the case with the CAS outputs gives a little better margin. The other consideration would be to make the frequency of operation 28 MHz for the processor core and 14 MHz for the PCI bus.

It was because of this worst case timing that the Pipelined PCI controller was developed. Here the MACH data path is fully registered across the part eliminating the need for a faster DRAM or slowing down the processor to 28 MHz. However this was not without its cost in performance in some way. The pipeline registers present a problem when the processor is trying to burst write the PCI bus or when a PCI master is trying to burst write the DRAM. In each case the next data value can not be advanced in the bus in a single cycle fashion because it takes an extra clock to get it across the pipeline registers and then a clock cycle to return ready. Hence the burst rate takes 3 clock cycles to complete versus the 1 for the flow through method prior. Reads however do get performed at the bus rate. So the designer must evaluate the relative need for performance in the desired direction and make his choices accordingly.

SUGGESTED REFERENCE

Bank Interleaved Memory System for an Am29030 Microprocessor application note, order# 18478, Advanced Micro Devices

EZ-030 Demonstration Board Theory of Operation application note, order# 17580, Advanced Micro Devices

PCI Local Bus Specification, Revision 2.0

PCI Special Interest Group

M/S HF3-15A

5200 N. E. Elam Young Parkway

Hillsboro, Oregon 97124-6497

(503) 696-2000

Appendix A. Schematics

The schematics for this design are shown as follows.

THIS PAGE AVAILABLE IN PRINT ONLY

THIS PAGE AVAILABLE IN PRINT ONLY

THIS PAGE AVAILABLE IN PRINT ONLY

THIS PAGE AVAILABLE IN PRINT ONLY

THIS PAGE AVAILABLE IN PRINT ONLY

THIS PAGE AVAILABLE IN PRINT ONLY

THIS PAGE AVAILABLE IN PRINT ONLY

THIS PAGE AVAILABLE IN PRINT ONLY

This page intentionally left blank

Appendix B. MACH220 Equations

;PALASM Design Description

;----- Declaration Segment -----

```
TITLE      29030 TO PCI CONVERTOR
PATTERN    030_PCI.PDS
REVISION   A
AUTHOR     DAVID STOENNER
COMPANY    AMD
DATE       02/21/93
```

CHIP Ux MACH220

;----- PIN Declarations -----

; CLOCK PINS

PIN ? MEMCLK

; INPUT PINS ONLY

```
PIN ? /RESET          PAIR  RESET_D
PIN ? /REQ            PAIR  REQ_D
PIN ? /BURST
PIN ? DELAY_IN
PIN ? /WE0
PIN ? /WE1
PIN ? /WE2
PIN ? /WE3
PIN ? /PCI_BREQ0
PIN ? /PCI_BREQ1
PIN ? /PCI_BREQ2
PIN ? A[31]
PIN ? A[30]
PIN ? A[29]
PIN ? /ADD_COMP
PIN ? AD[31]
```

; OUTPUT PINS ONLY

```
PIN ? /PROC_RDY
PIN ? /PROC_ERR

PIN ? /ROM_CS

PIN ? 85C30_CLK

PIN ? /DATA_EN
PIN ? /DATA_DIR
PIN ? /ADD_REG_EN
PIN ? /ADD_EN
PIN ? /ADD_CLK
PIN ? /ADD_INC
```

; INPUT / OUTPUT PINS

```
PIN ? /WRITE

PIN ? /RAS0
PIN ? /MUX
PIN ? /CAS0
PIN ? /CAS1
PIN ? /CAS2
PIN ? /CAS3
```

```

PIN ?          /PROC_BGRT

PIN ?          /PCI_BGRT0
PIN ?          /PCI_BGRT1
PIN ?          /PCI_BGRT2

PIN ?          PCI_C_BE0
PIN ?          PCI_C_BE1
PIN ?          PCI_C_BE2
PIN ?          PCI_C_BE3
PIN ?          /PCI_FRAME
PIN ?          /PCI_IRDY
PIN ?          /PCI_TRDY
PIN ?          /PCI_DEVSEL
PIN ?          /PCI_STOP

PIN ?          /85C30_CS
PIN ?          /85C30_RD
PIN ?          /85C30_WR

; ALL THE NODE DEFINITIONS

NODE ?         IDLE                                REGISTERED
NODE ?         MEM_ACCESS                          REGISTERED
NODE ?         REF_ACCESS                          REGISTERED
NODE ?         MEM_RDY
NODE ?         REQ_D
NODE ?         PCI_SIDE_ON
NODE ?         ABORT_STOP
NODE ?         N_PCI_IRDY          PAIR    PCI_IRDY
NODE ?         N_PCI_TRDY          PAIR    PCI_TRDY
NODE ?         N_PCI_STOP          PAIR    PCI_STOP
NODE ?         CT0
NODE ?         CT1
NODE ?         CT2
NODE ?         ST1
NODE ?         DEV_SEL_NODE
NODE ?         PCI_FRAME_D
NODE ?         PCI_IDLE
NODE ?         REF_REQ                                REGISTERED
NODE ?         Q0                                    REGISTERED
NODE ?         Q1                                    REGISTERED
NODE ?         Q2                                    REGISTERED
NODE ?         Q3                                    REGISTERED
NODE ?         Q4                                    REGISTERED
NODE ?         Q5                                    REGISTERED
NODE ?         Q6                                    REGISTERED
NODE ?         INTERNAL_WRITE
NODE ?         RESET_D
NODE ?         WT_ST0
NODE ?         WT_ST1
NODE ?         85C30_RDY

GROUP MACH_SEG_A 85C30_CS 85C30_RD 85C30_WR 85C30_CLK 85C30_RDY
                  WT_ST0 WT_ST1

GROUP MACH_SEG_B CAS0 CAS1 CAS2 CAS3 RAS0 MUX

GROUP MACH_SEG_C Q0 Q1 Q2 Q3 Q4 Q5 Q6 REF_REQ WRITE INTERNAL_WRITE
                  ROM_CS

GROUP MACH_SEG_D IDLE MEM_ACCESS REF_ACCESS MEM_RDY PROC_RDY
                  ST1

GROUP MACH_SEG_E PROC_BGRT PCI_BGRT0 PCI_BGRT1 PCI_BGRT2 PCI_SIDE_ON
                  PROC_ERR CT0 CT1 CT2

GROUP MACH_SEG_F ADD_INC ADD_EN ADD_REG_EN ADD_CLK DATA_EN DATA_DIR

```

```

GROUP MACH_SEG_G PCI_FRAME PCI_FRAME_D PCI_IRDY PCI_TRDY PCI_IDLE PCI_STOP
ABORT_STOP

GROUP MACH_SEG_H PCI_C_BE0 PCI_C_BE1 PCI_C_BE2 PCI_C_BE3 PCI_DEVSEL
DEV_SEL_NODE

STRING ADD_STOP '( PCI_FRAME_D * PCI_IRDY * ADD_COMP )'

STRING PROC_SIDE_ON '/PCI_BGRT0*/PCI_BGRT1*/PCI_BGRT2'

;----- Boolean Equation Segment -----

EQUATIONS

; THE PCI BUS ADDRESS DATA MUX AND UNMUX

ADD_INC = PCI_SIDE_ON * PCI_FRAME_D * PCI_IRDY * PCI_TRDY

ADD_CLK = PCI_FRAME * /PCI_FRAME_D

PCI_SIDE_ON = PCI_BGRT0 + PCI_BGRT1 + PCI_BGRT2

ADD_EN = PROC_SIDE_ON * /PCI_FRAME_D * /RESET

ADD_REG_EN = PCI_BGRT0 + PCI_BGRT1 + PCI_BGRT2

DATA_EN = PROC_SIDE_ON * REQ * A[31] * PCI_FRAME_D * /RESET
          + PCI_SIDE_ON * DEV_SEL_NODE * /RESET

DATA_DIR = PROC_SIDE_ON * REQ * /WRITE
           + PCI_SIDE_ON * DEV_SEL_NODE * INTERNAL_WRITE

; BUS ARBITOR AND BUS GRANT STATE MACHINES

RESET_D.CLKF = MEMCLK

RESET_D := RESET

REQ_D.CLKF = MEMCLK

REQ_D := REQ

; REQ_D HAS BEEN ADDED TO FIX AN ARBITRATION PROBLEM OF THE 29030 REV B,C AND
; D. TO BE GUARENTEED THAT THE 29030 HAS GIVEN UP THE BUS WHEN BGRT HAS BEEN
; REMOVED REQ MUST BE SAMPLED FALSE FOR 2 CLOCK CYCLES, HENCE THE ADDITION OF
; THE REQ_D TERM AND THE INCLUSION IN THE PCI_BGRTx TERMS. WHEN THIS PROBLEM
; IS FIXED THE REQ_D MAY BE REMOVED FROM ALL EQUATIONS.

PROC_BGRT.CLKF = MEMCLK

PROC_BGRT := /PROC_BGRT * /PCI_BREQ0 * /PCI_BREQ1 * /PCI_BREQ2
            * /PCI_FRAME_D * /RESET_D
            + PROC_BGRT * /PCI_BREQ0 * /PCI_BREQ1 * /PCI_BREQ2 * /RESET_D

PCI_BGRT0.CLKF = MEMCLK

PCI_BGRT0.TRST = /RESET

PCI_BGRT0 := /PCI_BGRT0 * PCI_BREQ0 * /PROC_BGRT * /REQ * /REQ_D
            * /PCI_BGRT1 * /PCI_BGRT2 * /PCI_FRAME_D * /RESET_D
            + PCI_BREQ0 * PCI_BGRT0 * /RESET_D
            + PCI_BGRT0 * PCI_FRAME_D * /RESET_D

PCI_BGRT1.CLKF = MEMCLK

```

```

PCI_BGRT1.TRST = /RESET

PCI_BGRT1 := /PCI_BGRT1 * PCI_BREQ1 * /PCI_BREQ0 * /PROC_BGRT
             * /REQ * /REQ_D * /PCI_BGRT0 * /PCI_BGRT2
             * /PCI_FRAME_D * /RESET_D
             + PCI_BREQ1 * PCI_BGRT1 * /RESET_D
             + PCI_BGRT1 * PCI_FRAME_D * /RESET_D

PCI_BGRT2.CLKF = MEMCLK

PCI_BGRT2.TRST = /RESET

PCI_BGRT2 := /PCI_BGRT2 * PCI_BREQ2 * /PCI_BREQ1 * /PCI_BREQ0
             * /PROC_BGRT * /REQ * /REQ_D * /PCI_BGRT0 * /PCI_BGRT1
             * /PCI_FRAME_D * /RESET_D
             + PCI_BREQ2 * PCI_BGRT2 * /RESET_D
             + PCI_BGRT2 * PCI_FRAME_D * /RESET_D

; PCI BUS CONTROL LOGIC

PCI_FRAME.TRST = PROC_SIDE_ON * /RESET

PCI_FRAME = PROC_SIDE_ON * PCI_IDLE * REQ * A[31] * /PCI_FRAME_D
            + PROC_SIDE_ON * REQ * A[31] * BURST * PCI_FRAME_D * /PROC_ERR

PCI_FRAME_D.CLKF = MEMCLK

; PCI_STOP IS USED IN THE LAST 4 EQUATIONS FOR THE ADVENT AN EARLY TERMINATION
; OCCURS.

PCI_FRAME_D := PROC_SIDE_ON * PCI_IDLE * REQ * A[31] * /PCI_FRAME_D
              + PROC_SIDE_ON * REQ * A[31] * BURST * PCI_FRAME_D * /PCI_STOP
              + PROC_SIDE_ON * REQ * A[31] * /BURST * PCI_FRAME_D
                * /PCI_IRDY * /PCI_STOP
              + PROC_SIDE_ON * REQ * A[31] * /BURST * PCI_FRAME_D
                * /PCI_TRDY * /PCI_STOP
              + PCI_SIDE_ON * PCI_FRAME * /ADD_STOP * /ABORT_STOP
              + PCI_SIDE_ON * /PCI_FRAME * PCI_FRAME_D * /PCI_IRDY
              + PCI_SIDE_ON * /PCI_FRAME * PCI_FRAME_D * /PCI_TRDY

PCI_IDLE.CLKF = MEMCLK

PCI_IDLE := /PCI_FRAME * /PCI_FRAME_D

CT0.CLKF = MEMCLK

CT0 := PCI_FRAME_D * /PCI_DEVSEL * ( /CT0 + CT0 * CT1 * CT2 )

CT1.CLKF = MEMCLK

CT1 := PCI_FRAME_D * /PCI_DEVSEL * (( CT0 :+: CT1 ) + CT0 * CT1 * CT2 )

CT2.CLKF = MEMCLK

CT2 := PCI_FRAME_D * /PCI_DEVSEL * (( CT2 :+: ( CT1*CT0 ) ) + CT0 * CT1 * CT2 )

PROC_ERR = PROC_SIDE_ON * REQ * A[31] * CT2 * CT1 * CT0 * /PCI_DEVSEL
           + PROC_SIDE_ON * REQ * A[31] * PCI_DEVSEL * PCI_STOP
           + PROC_SIDE_ON * REQ * A[31] * PCI_FRAME_D
             * /PCI_DEVSEL * DEV_SEL_NODE

```

```

PCI_STOP.TRST = PCI_SIDE_ON * DEV_SEL_NODE * /RESET

PCI_STOP.CLKF = MEMCLK

PCI_STOP := /PCI_STOP * PCI_FRAME * DEV_SEL_NODE * ADD_STOP * /RESET_D

N_PCI_STOP .CLKF = MEMCLK

N_PCI_STOP := { PCI_STOP }

ABORT_STOP.CLKF = MEMCLK

ABORT_STOP := /ABORT_STOP * PCI_SIDE_ON * PCI_FRAME * PCI_FRAME_D
              * ADD_STOP
              + ABORT_STOP * PCI_SIDE_ON * PCI_FRAME

PCI_DEVSEL.TRST = PCI_SIDE_ON * DEV_SEL_NODE * /RESET

PCI_DEVSEL = PCI_FRAME_D

DEV_SEL_NODE.CLKF = MEMCLK

DEV_SEL_NODE := PCI_SIDE_ON * PCI_FRAME * AD[31] * /PCI_FRAME_D
                * /ABORT_STOP
                + PCI_SIDE_ON * DEV_SEL_NODE * PCI_FRAME_D * /ABORT_STOP
                + PROC_SIDE_ON * PCI_FRAME_D * /DEV_SEL_NODE * PCI_DEVSEL
                * /RESET_D
                + PROC_SIDE_ON * PCI_FRAME_D * DEV_SEL_NODE * /RESET_D

PCI_TRDY.TRST = PCI_SIDE_ON * DEV_SEL_NODE * /RESET

PCI_TRDY.CLKF = MEMCLK

PCI_TRDY := PCI_SIDE_ON * MEM_ACCESS * /N_PCI_TRDY
            + PCI_SIDE_ON * MEM_ACCESS * N_PCI_TRDY * PCI_FRAME * /ADD_STOP
            + PCI_SIDE_ON * MEM_ACCESS * N_PCI_TRDY * /PCI_FRAME * /PCI_IRDY

N_PCI_TRDY.CLKF = MEMCLK

N_PCI_TRDY := { PCI_TRDY }

PCI_IRDY.TRST = PROC_SIDE_ON * /RESET

PCI_IRDY.CLKF = MEMCLK

PCI_IRDY := PROC_SIDE_ON * REQ * A[31] * PCI_IDLE * /N_PCI_IRDY * /RESET_D
            + N_PCI_IRDY * PROC_SIDE_ON * REQ * A[31] * BURST * /RESET_D
            + N_PCI_IRDY * PROC_SIDE_ON * REQ * A[31] * /BURST
              * /PCI_TRDY * /RESET_D

N_PCI_IRDY.CLKF = MEMCLK

N_PCI_IRDY := { PCI_IRDY }

PCI_C_BE0.TRST = PROC_SIDE_ON * /RESET

PCI_C_BE0 = /PCI_FRAME_D * WRITE
            + PCI_FRAME_D * /WE0 * WRITE

PCI_C_BE1.TRST = PROC_SIDE_ON * /RESET

PCI_C_BE1 = /PCI_FRAME_D
            + PCI_FRAME_D * /WE1 * WRITE

```

```

; IF A31 IS 1 WE ARE DOING A PCI ACCESS. THEN A30 DETERMINES WHETHER IT IS A
; MEMORY CYCLE OR A PCI CONFIGURATION CYCLE. IF A30 = 0 THEN IT IS A MEMORY
; AND IF A30 = 1 IT IS A CONFIGURATION CYCLE.

```

```

PCI_C_BE2.TRST = PROC_SIDE_ON * /RESET

```

```

PCI_C_BE2 = /PCI_FRAME_D * A[31] * /A[30]
          + PCI_FRAME_D * /WE2 * WRITE

```

```

PCI_C_BE3.TRST = PROC_SIDE_ON * /RESET

```

```

PCI_C_BE3 = /PCI_FRAME_D * A[31] * A[30]
          + PCI_FRAME_D * /WE3 * WRITE

```

```

WRITE.TRST = PCI_SIDE_ON

```

```

WRITE = INTERNAL_WRITE

```

```

MINIMIZE_OFF

```

```

INTERNAL_WRITE.CLKF = MEMCLK

```

```

INTERNAL_WRITE := PROC_SIDE_ON * WRITE * /RESET_D
                + PCI_SIDE_ON * /INTERNAL_WRITE * PCI_FRAME * /PCI_FRAME_D
                  * PCI_C_BE0 * /RESET_D
                + PCI_SIDE_ON * INTERNAL_WRITE * PCI_FRAME_D * /RESET_D

```

```

MINIMIZE_ON

```

```

; MEMORY STATE MACHINES FOR THE RAS CAS GENERATION

```

```

IDLE.CLKF = MEMCLK

```

```

IDLE := /IDLE * /MEM_ACCESS * /REF_ACCESS
       + RESET_D
       + IDLE * PROC_SIDE_ON * /REF_REQ * /( REQ * /A[31] * /A[30] * A[29] )
       + IDLE * PCI_SIDE_ON * /REF_REQ * /( PCI_FRAME_D * A[31] )
       + /IDLE * REF_ACCESS * /REF_REQ
       + /IDLE * PROC_SIDE_ON * MEM_ACCESS * ST1 * MEM_RDY * /BURST
       + /IDLE * PCI_SIDE_ON * MEM_ACCESS * ST1 * MEM_RDY * /PCI_FRAME
         * PCI_IRDY * PCI_TRDY
       + /IDLE * PCI_SIDE_ON * MEM_ACCESS * ST1 * MEM_RDY * PCI_FRAME
         * ADD_STOP

```

```

REF_ACCESS.CLKF = MEMCLK

```

```

REF_ACCESS := IDLE * REF_REQ * /REF_ACCESS
             + REF_ACCESS * REF_REQ

```

```

MEM_ACCESS.CLKF = MEMCLK

```

```

MEM_ACCESS := IDLE * PROC_SIDE_ON * REQ * /A[31] * /A[30] * A[29]
              * /REF_REQ * /MEM_ACCESS
              + IDLE * PCI_SIDE_ON * PCI_FRAME_D * A[31] * /REF_REQ
                * /MEM_ACCESS
              + MEM_ACCESS * /ST1
              + MEM_ACCESS * PROC_SIDE_ON * BURST
              + MEM_ACCESS * PCI_SIDE_ON * PCI_FRAME * /ADD_STOP
              + MEM_ACCESS * PCI_SIDE_ON * /PCI_FRAME * /PCI_IRDY

```

```

ST1.CLKF = MEMCLK

ST1 := MEM_ACCESS

MEM_RDY.CLKF = MEMCLK

MEM_RDY := PROC_SIDE_ON * REQ * /A[31] * /A[30] * /A[29] * /MEM_RDY
           * /RESET_D ; ROM ACCESS
           + MEM_ACCESS * /MEM_RDY * /RESET_D ; ALL DRAM ACCESSES
           ; EITHER PROCESSOR OR
           ; PCI BUS ACCESS
           + PROC_SIDE_ON * MEM_ACCESS * MEM_RDY * BURST * /RESET_D
           + PCI_SIDE_ON * MEM_ACCESS * MEM_RDY * PCI_FRAME
           * /ADD_STOP * /RESET_D
           + PCI_SIDE_ON * MEM_ACCESS * MEM_RDY * /PCI_FRAME
           * /PCI_IRDY * /RESET_D

PROC_RDY = PROC_SIDE_ON * /A[31] * /A[30] * MEM_RDY ; GOOD FOR BOTH ROM AND
           ; DRAM
           + PROC_SIDE_ON * /A[31] * A[30] * 85C30_RDY
           + PROC_SIDE_ON * A[31] * PCI_FRAME_D * PCI_TRDY * PCI_IRDY

MINIMIZE_OFF

RAS0 = MEM_ACCESS
      + REF_ACCESS * /MEMCLK
      + RAS0 * REF_ACCESS

MUX = MEM_ACCESS * /MEMCLK
     + MUX * MEM_ACCESS

CAS0 = REF_ACCESS
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * MEMCLK * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS0 * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS0 * /MEMCLK
      + MEM_ACCESS * ST1 * PROC_SIDE_ON * INTERNAL_WRITE * WE0 * /MEMCLK
      + MEM_ACCESS * ST1 * PCI_SIDE_ON * INTERNAL_WRITE * PCI_C_BE0
      * /MEMCLK

CAS1 = REF_ACCESS
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * MEMCLK * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS1 * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS1 * /MEMCLK
      + MEM_ACCESS * ST1 * PROC_SIDE_ON * INTERNAL_WRITE * WE1 * /MEMCLK
      + MEM_ACCESS * ST1 * PCI_SIDE_ON * INTERNAL_WRITE * PCI_C_BE1
      * /MEMCLK

CAS2 = REF_ACCESS
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * MEMCLK * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS2 * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS2 * /MEMCLK
      + MEM_ACCESS * ST1 * PROC_SIDE_ON * INTERNAL_WRITE * WE2 * /MEMCLK
      + MEM_ACCESS * ST1 * PCI_SIDE_ON * INTERNAL_WRITE * PCI_C_BE2
      * /MEMCLK

CAS3 = REF_ACCESS
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * MEMCLK * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS3 * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS3 * /MEMCLK
      + MEM_ACCESS * ST1 * PROC_SIDE_ON * INTERNAL_WRITE * WE3 * /MEMCLK

```

```

+ MEM_ACCESS * ST1 * PCI_SIDE_ON * INTERNAL_WRITE * PCI_C_BE3
  * /MEMCLK

```

```
Q0.CLKF = MEMCLK
```

```
Q0.T := VCC
```

```
Q1.CLKF = MEMCLK
```

```
Q1.T := Q0
```

```
Q2.CLKF = MEMCLK
```

```
Q2.T := Q1 * Q0
```

```
Q3.CLKF = MEMCLK
```

```
Q3.T := Q2 * Q1 * Q0
```

```
Q4.CLKF = MEMCLK
```

```
Q4.T := Q3 * Q2 * Q1 * Q0
```

```
Q5.CLKF = MEMCLK
```

```
Q5.T := Q4 * Q3 * Q2 * Q1 * Q0
```

```
Q6.CLKF = MEMCLK
```

```
Q6.T := Q5 * Q4 * Q3 * Q2 * Q1 * Q0
```

```
REF_REQ.CLKF = MEMCLK
```

```
REF_REQ := Q6 * Q5 * Q4 * Q3 * Q2 * Q1 * Q0
          + REF_REQ * /REF_ACCESS
```

```

; ROM_CS ALSO DRIVES THE RDN PIN OF THE 29030 AND NEEDS TO BE 1 IF THE ROM IS
; 8 BITS WIDE AND A ZERO IF IT 16 BITS WIDE DURING A RESET. SO TO DO THIS
; RESET WOULD BE OR WITH THE ROM_CS EQUATION WHEN 16 BIT MEMORY IS NEEDED.
; IT IS CUREENTLY COMMENTED OUT

```

```
ROM_CS = PROC_SIDE_ON * REQ * /A[31] * /A[30] * /A[29]
;       + RESET
```

```

; THIS IS THE SERIAL CONTROLLER SECTION THAT CAN BE REMOVED IF THE SERIAL PORT
; IS NOT USED.

```

```
85C30_CLK.CLKF = MEMCLK
```

```
85C30_CLK.T := VCC
```

```
85C30_CS = PROC_SIDE_ON * REQ * /A[31] * A[30] * /A[29]
```

```
85C30_RD.CLKF = MEMCLK
```

```

85C30_RD := 85C30_CS*/WRITE*/85C30_RD*/85C30_RDY
          + 85C30_RD*/85C30_RDY
          + RESET_D

```

```

85C30_WR.CLKF = MEMCLK

```

```

85C30_WR := 85C30_CS*WRITE*/85C30_WR*/85C30_RDY
          + 85C30_WR*/WT_ST0*/85C30_RDY
          + 85C30_WR*/WT_ST1*/85C30_RDY
          + RESET_D

```

```

WT_ST0.CLKF = MEMCLK

```

```

WT_ST0 := (85C30_RD + 85C30_WR)*/WT_ST0*/85C30_RDY * /RESET_D

```

```

WT_ST1.CLKF = MEMCLK

```

```

WT_ST1 := (85C30_RD + 85C30_WR)*(WT_ST0:+:WT_ST1)*/85C30_RDY * /RESET_D

```

```

85C30_RDY.CLKF = MEMCLK

```

```

85C30_RDY := WT_ST0*WT_ST1*/85C30_RDY * /RESET_D

```

```

;=====

```

Appendix C. MACH465 Equations

;PALASM Design Description

----- Declaration Segment -----

TITLE 29030 TO PCI CONVERTOR
PATTERN PCI46572.PDS
REVISION A
AUTHOR DAVID STOENNER
COMPANY AMD
DATE 01/03/95

CHIP Ux MACH465

----- PIN Declarations -----

; CLOCK PINS

PIN 187 MEMCLK ;

; INPUT PINS ONLY

PIN 188 MEMCLK_IN
PIN 171 /RESET_PAIR RESET_D
PIN 177 /RESET_IN

PIN 173 /REQ_PAIR REQ_D

PIN 72 /BURST

PIN 125 DELAY_IN

PIN 46 /WE0
PIN 31 /WE1
PIN 22 /WE2

PIN 21 /WE3

PIN 85 /PCI_BREQ0
PIN 84 /PCI_BREQ1
PIN 73 /PCI_BREQ2

; OUTPUT PINS ONLY

PIN 42 /PROC_RDY
PIN 49 /PROC_ERR

PIN 165 /ROM_CS
PIN 163 /ROM_OE
PIN 161 /ROM_WE

PIN 159 85C30_CLK

; INPUT / OUTPUT PINS

PIN 153 A[31]
PIN 151 A[30]
PIN 147 A[29]
PIN 150 A[28]
PIN 148 A[27]
PIN 146 A[26]
PIN 152 A[25]
PIN 149 A[24]
PIN 140 A[23]
PIN 138 A[22]
PIN 136 A[21]
PIN 142 A[20]
PIN 139 A[19]

PIN 137 A[18]
PIN 143 A[17]
PIN 141 A[16]
PIN 120 A[15]
PIN 122 A[14]
PIN 124 A[13]
PIN 118 A[12]
PIN 121 A[11]
PIN 123 A[10]
PIN 117 A[9]
PIN 119 A[8]
PIN 110 A[7]
PIN 112 A[6]
PIN 114 A[5]
PIN 108 A[4]
PIN 111 A[3]
PIN 113 A[2]
PIN 107 A[1]
PIN 109 A[0]

PIN 100 D[31]
PIN 98 D[30]
PIN 96 D[29]
PIN 102 D[28]
PIN 99 D[27]
PIN 97 D[26]
PIN 103 D[25]
PIN 101 D[24]
PIN 90 D[23]
PIN 88 D[22]
PIN 86 D[21]
PIN 92 D[20]
PIN 89 D[19]
PIN 87 D[18]
PIN 93 D[17]
PIN 91 D[16]
PIN 67 D[15]
PIN 69 D[14]
PIN 71 D[13]
PIN 65 D[12]
PIN 68 D[11]
PIN 70 D[10]
PIN 64 D[9]
PIN 66 D[8]
PIN 57 D[7]
PIN 59 D[6]
PIN 61 D[5]
PIN 55 D[4]
PIN 58 D[3]
PIN 60 D[2]
PIN 54 D[1]
PIN 56 D[0]

PIN 15 AD[31]
PIN 16 AD[30]
PIN 18 AD[29]
PIN 20 AD[28]
PIN 14 AD[27]
PIN 17 AD[26]
PIN 19 AD[25]
PIN 13 AD[24]
PIN 6 AD[23]
PIN 8 AD[22]
PIN 10 AD[21]
PIN 4 AD[20]
PIN 7 AD[19]
PIN 9 AD[18]
PIN 3 AD[17]

```
PIN 5 AD[16]
PIN 204 AD[15]
PIN 202 AD[14]
PIN 200 AD[13]
PIN 206 AD[12]
PIN 203 AD[11]
PIN 201 AD[10]
PIN 207 AD[9]
PIN 205 AD[8]
PIN 190 AD[7]
PIN 196 AD[6]
PIN 193 AD[5]
PIN 191 AD[4]
PIN 197 AD[3]
PIN 195 AD[2]
PIN 194 AD[1]
PIN 192 AD[0]

PIN 160 /WRITE

PIN 158 /RAS0
PIN 164 MUX

PIN 172 /CAS0
PIN 174 /CAS1
PIN 168 /CAS2
PIN 170 /CAS3

PIN 44 /PROC_BGRT

PIN 48 /PCI_BGRT0
PIN 45 /PCI_BGRT1
PIN 43 /PCI_BGRT2

PIN 35 PCI_C_BE0
PIN 36 PCI_C_BE1
PIN 34 PCI_C_BE2
PIN 32 PCI_C_BE3
PIN 37 /PCI_FRAME
PIN 33 /PCI_IRDY
PIN 39 /PCI_TRDY
PIN 38 /PCI_DEVSEL
PIN 47 /PCI_STOP

PIN 162 /85C30_CS
PIN 175 /85C30_RD
PIN 169 /85C30_WR

; ALL THE NODE DEFINITIONS

NODE ? IDLE REGISTERED

NODE ? MEM_ACCESS REGISTERED

NODE ? REF_ACCESS REGISTERED

NODE ? ST1

NODE ? MEM_RDY

NODE ? REQ_D
NODE ? PCI_SIDE_ON
NODE ? AD_EN
NODE ? D_EN
NODE ? ADD_COMP
NODE ? ABORT_STOP
NODE ? N_PCI_IRDY PAIR PCI_IRDY
```

```

NODE ? N_PCI_TRDY PAIR PCI_TRDY

NODE ? N_PCI_STOP PAIR PCI_STOP

NODE ? CT0
NODE ? CT1
NODE ? NO_DEVSEL

NODE ? DEV_SEL_NODE
NODE ? PCI_FRAME_D
NODE ? PCI_IDLE
NODE ? REF_REQ REGISTERED

NODE ? Q0 REGISTERED
NODE ? Q1 REGISTERED
NODE ? Q2 REGISTERED
NODE ? Q3 REGISTERED
NODE ? Q4 REGISTERED
NODE ? Q5 REGISTERED
NODE ? Q6 REGISTERED

NODE ? INTERNAL_WRITE
NODE ? RESET_D
NODE ? WT_ST0
NODE ? WT_ST1
NODE ? 85C30_RDY
NODE ? N_A[0..31] PAIR A[0..31]

;GROUP MACH_SEG_A AD[0..7]

;GROUP MACH_SEG_B AD[8..15]

;GROUP MACH_SEG_C AD[16..23]

;GROUP MACH_SEG_D AD[24..31]

GROUP MACH_SEG_E DEV_SEL_NODE PCI_FRAME_D PCI_IDLE
; PCI_C_BE0 PCI_C_BE1 PCI_C_BE2 PCI_C_BE3
; PCI_FRAME PCI_IRDY PCI_TRDY PCI_DEVSEL

GROUP MACH_SEG_F CT0 CT1 ABORT_STOP NO_DEVSEL
; PROC_BGRT PCI_BGRT0 PCI_BGRT1 PCI_BGRT2 PCI_STOP
; PCI_SIDE_ON PROC_RDY PROC_ERR

GROUP MACH_SEG_G Q0 Q1 Q2 Q3 Q4 Q5 Q6 REF_REQ
; D[0..7]

GROUP MACH_SEG_H AD_EN D_EN
; D[8..15]

;GROUP MACH_SEG_I D[16..23]

;GROUP MACH_SEG_J D[24..31]

GROUP MACH_SEG_K ADD_COMP
; A[0..7]

;GROUP MACH_SEG_L A[8..15]

;GROUP MACH_SEG_M A[16..23]

;GROUP MACH_SEG_N A[24..31]

GROUP MACH_SEG_O IDLE MEM_ACCESS REF_ACCESS ST1 MEM_RDY INTERNAL_WRITE
; ROM_CS RAS0 MUX WRITE 85C30_CS 85C30_CLK

GROUP MACH_SEG_P 85C30_RDY WT_ST0 WT_ST1

```

```

;          CAS0 CAS1 CAS2 CAS3 85C30_WR 85C30_RD
STRING ADD_INC '( PCI_FRAME_D * PCI_IRDY * PCI_TRDY )'
STRING ADD_STOP '( PCI_FRAME_D * PCI_IRDY * ADD_COMP )'

STRING ADD_HOLD '( PCI_FRAME_D * ( /PCI_IRDY + /PCI_TRDY ) )'
STRING ADD_CLK '( PCI_FRAME * /PCI_FRAME_D )'
STRING PROC_SIDE_ON '/PCI_BGRT0*/PCI_BGRT1*/PCI_BGRT2'

;----- Boolean Equation Segment -----
EQUATIONS
RESET.CLKF = MEMCLK
RESET := RESET_IN
RESET_D.CLKF = MEMCLK
RESET_D := RESET

; THE PCI BUS ADDRESS DATA MUX AND UNMUX
PCI_SIDE_ON = PCI_BGRT0 + PCI_BGRT1 + PCI_BGRT2
AD_EN = PROC_SIDE_ON * ( /PCI_FRAME_D + PCI_FRAME_D * WRITE )
      + PCI_SIDE_ON * PCI_FRAME_D * /INTERNAL_WRITE

AD[0..31].TRST = AD_EN * /RESET
AD[0..27] = A[0..27] * PROC_SIDE_ON * /PCI_FRAME_D * A[31]
          + D[0..27] * PCI_FRAME_D
AD[28..31] = D[28..31] * PCI_FRAME_D

A[0..31].TRST = PCI_SIDE_ON * /RESET
A[0..31].RSTF = GND
A[0..31].CLKF = MEMCLK
A[0..1] := GND
A[6..31] := AD[6..31] * ADD_CLK + N_A[6..31] * PCI_FRAME_D
A[5].T := PCI_FRAME * /PCI_FRAME_D * N_A[5] * /AD[5]
          + PCI_FRAME * /PCI_FRAME_D * /N_A[5] * AD[5]
          + PCI_FRAME_D * PCI_IRDY * PCI_TRDY * N_A[2]
            * N_A[3] * N_A[4]
A[4].T := PCI_FRAME * /PCI_FRAME_D * N_A[4] * /AD[4]
          + PCI_FRAME * /PCI_FRAME_D * /N_A[4] * AD[4]
          + PCI_FRAME_D * PCI_IRDY * PCI_TRDY * N_A[2] * N_A[3]
A[3].T := PCI_FRAME * /PCI_FRAME_D * N_A[3] * /AD[3]
          + PCI_FRAME * /PCI_FRAME_D * /N_A[3] * AD[3]
          + PCI_FRAME_D * PCI_IRDY * PCI_TRDY * N_A[2]

```

```

A[2] := AD[2] * PCI_FRAME * /PCI_FRAME_D
      + PCI_FRAME_D * N_A[2] * /PCI_IRDY
      + PCI_FRAME_D * N_A[2] * /PCI_TRDY
      + PCI_FRAME_D * /N_A[2] * PCI_IRDY * PCI_TRDY

```

```
N_A[0..31].RSTF = GND
```

```
N_A[0..31].CLKF = MEMCLK
```

```

N_A[0] := { A[0] }
N_A[1] := { A[1] }
N_A[2] := { A[2] }
N_A[3].T := { A[3].T }
N_A[4].T := { A[4].T }
N_A[5].T := { A[5].T }
N_A[6] := { A[6] }
N_A[7] := { A[7] }
N_A[8] := { A[8] }
N_A[9] := { A[9] }
N_A[10] := { A[10] }
N_A[11] := { A[11] }
N_A[12] := { A[12] }
N_A[13] := { A[13] }
N_A[14] := { A[14] }
N_A[15] := { A[15] }
N_A[16] := { A[16] }
N_A[17] := { A[17] }
N_A[18] := { A[18] }
N_A[19] := { A[19] }
N_A[20] := { A[20] }
N_A[21] := { A[21] }
N_A[22] := { A[22] }
N_A[23] := { A[23] }
N_A[24] := { A[24] }
N_A[25] := { A[25] }
N_A[26] := { A[26] }
N_A[27] := { A[27] }
N_A[28] := { A[28] }
N_A[29] := { A[29] }
N_A[30] := { A[30] }
N_A[31] := { A[31] }

```

```

D_EN = PROC_SIDE_ON * PCI_FRAME_D * /WRITE
      + PCI_SIDE_ON * PCI_FRAME_D * INTERNAL_WRITE

```

```
D[0..31].TRST = D_EN * /RESET
```

```
D[0..31] = AD[0..31]
```

```
ADD_COMP = N_A[5] * N_A[4] * N_A[3] * N_A[2]
```

```
; BUS ARBITOR AND BUS GRANT STATE MACHINES
```

```
REQ_D.CLKF = MEMCLK
```

```
REQ_D := REQ
```

```
PROC_BGRT.CLKF = MEMCLK
```

```

PROC_BGRT := /PROC_BGRT * /PCI_BREQ0 * /PCI_BREQ1 * /PCI_BREQ2
            * /PCI_FRAME_D * /RESET_D
            + PROC_BGRT * /PCI_BREQ0 * /PCI_BREQ1 * /PCI_BREQ2 * /RESET_D

```

```
PCI_BGRT0.CLKF = MEMCLK
```

```
PCI_BGRT0.TRST = /RESET
```

```

PCI_BGRT0 := /PCI_BGRT0 * PCI_BREQ0 * /PROC_BGRT * /REQ * /REQ_D
            * /PCI_BGRT1 * /PCI_BGRT2 * /PCI_FRAME_D * /RESET_D
            + PCI_BREQ0 * PCI_BGRT0 * /RESET_D
            + PCI_BGRT0 * PCI_FRAME_D * /RESET_D

```

```
PCI_BGRT1.CLKF = MEMCLK
```

```
PCI_BGRT1.TRST = /RESET
```

```

PCI_BGRT1 := /PCI_BGRT1 * PCI_BREQ1 * /PCI_BREQ0 * /PROC_BGRT
            * /REQ * /REQ_D * /PCI_BGRT0 * /PCI_BGRT2
            * /PCI_FRAME_D * /RESET_D
            + PCI_BREQ1 * PCI_BGRT1 * /RESET_D
            + PCI_BGRT1 * PCI_FRAME_D * /RESET_D

```

```
PCI_BGRT2.CLKF = MEMCLK
```

```
PCI_BGRT2.TRST = /RESET
```

```

PCI_BGRT2 := /PCI_BGRT2 * PCI_BREQ2 * /PCI_BREQ1 * /PCI_BREQ0
            * /PROC_BGRT * /REQ * /REQ_D * /PCI_BGRT0 * /PCI_BGRT1
            * /PCI_FRAME_D * /RESET_D
            + PCI_BREQ2 * PCI_BGRT2 * /RESET_D
            + PCI_BGRT2 * PCI_FRAME_D * /RESET_D

```

```
; PCI BUS CONTROL LOGIC
```

```
PCI_FRAME.TRST = PROC_SIDE_ON * /RESET
```

```

PCI_FRAME = PROC_SIDE_ON * PCI_IDLE * REQ * A[31] * /PCI_FRAME_D
            + PROC_SIDE_ON * REQ * A[31] * BURST * PCI_FRAME_D * /PROC_ERR
            * /NO_DEVSEL

```

```
PCI_FRAME_D.CLKF = MEMCLK
```

```
PCI_FRAME_D.RSTF = RESET_D
```

```
; PROC_ERR IS USED IN THE SECOND AND THIRD EQUATIONS FOR THE ADVENT AN
; EARLY TERMINATION OCCURS.
```

```

PCI_FRAME_D := PROC_SIDE_ON * PCI_IDLE * REQ * A[31] * /PCI_FRAME_D
            + PROC_SIDE_ON * REQ * A[31] * BURST * PCI_FRAME_D * /PROC_ERR
            * /NO_DEVSEL
            + PROC_SIDE_ON * REQ * A[31] * /BURST * PCI_FRAME_D
            * /PCI_TRDY * /PROC_ERR * /NO_DEVSEL
            + PCI_SIDE_ON * PCI_FRAME * /ADD_STOP * /ABORT_STOP
            + PCI_SIDE_ON * /PCI_FRAME * PCI_FRAME_D * /PCI_IRDY
            + PCI_SIDE_ON * /PCI_FRAME * PCI_FRAME_D * /PCI_TRDY

```

```
PCI_IDLE.CLKF = MEMCLK
```

```
PCI_IDLE := /PCI_FRAME * /PCI_FRAME_D
```

```

CT0.CLKF = MEMCLK
CT0 := PCI_FRAME_D * /PCI_DEVSEL * ( /CT0 + CT0 * CT1 )

CT1.CLKF = MEMCLK
CT1 := PCI_FRAME_D * /PCI_DEVSEL * (( CT0 :+: CT1 ) + CT0 * CT1 )

NO_DEVSEL.CLKF = MEMCLK
NO_DEVSEL := /NO_DEVSEL * PROC_SIDE_ON * REQ * A[31] * CT1 * CT0 * /PCI_DEVSEL

PROC_ERR.CLKF = MEMCLK
PROC_ERR := PROC_SIDE_ON * REQ * A[31] * PCI_STOP

PCI_STOP.TRST = PCI_SIDE_ON * DEV_SEL_NODE * /RESET
PCI_STOP.CLKF = MEMCLK
PCI_STOP := /PCI_STOP * PCI_FRAME * DEV_SEL_NODE * ADD_STOP

N_PCI_STOP .CLKF = MEMCLK
N_PCI_STOP := { PCI_STOP }

ABORT_STOP.CLKF = MEMCLK
ABORT_STOP := /ABORT_STOP * PCI_SIDE_ON * PCI_FRAME * PCI_FRAME_D
             * ADD_STOP
             + ABORT_STOP * PCI_SIDE_ON * PCI_FRAME

PCI_DEVSEL.TRST = PCI_SIDE_ON * DEV_SEL_NODE * /RESET
PCI_DEVSEL = PCI_FRAME_D

DEV_SEL_NODE.CLKF = MEMCLK
DEV_SEL_NODE.RSTF = RESET_D
DEV_SEL_NODE := PCI_SIDE_ON * PCI_FRAME * AD[31] * /PCI_FRAME_D
               * /ABORT_STOP
               + PCI_SIDE_ON * DEV_SEL_NODE * PCI_FRAME_D * /ABORT_STOP
               + PROC_SIDE_ON * PCI_FRAME_D * /DEV_SEL_NODE * PCI_DEVSEL
               + PROC_SIDE_ON * PCI_FRAME_D * DEV_SEL_NODE

PCI_TRDY.TRST = PCI_SIDE_ON * DEV_SEL_NODE * /RESET
PCI_TRDY.RSTF = RESET_D
PCI_TRDY.CLKF = MEMCLK
PCI_TRDY := PCI_SIDE_ON * MEM_ACCESS * /N_PCI_TRDY
           + PCI_SIDE_ON * MEM_ACCESS * N_PCI_TRDY * PCI_FRAME * /ADD_STOP
           + PCI_SIDE_ON * MEM_ACCESS * N_PCI_TRDY * /PCI_FRAME * /PCI_IRDY

```

```

N_PCI_TRDY.CLKF = MEMCLK
N_PCI_TRDY.RSTF = RESET_D

N_PCI_TRDY := { PCI_TRDY }

PCI_IRDY.TRST = PROC_SIDE_ON * /RESET
PCI_IRDY.CLKF = MEMCLK
PCI_IRDY.RSTF = RESET_D

PCI_IRDY :=  PROC_SIDE_ON * REQ * A[31] * PCI_IDLE * /N_PCI_IRDY
            + N_PCI_IRDY * PROC_SIDE_ON * REQ * A[31] * BURST * /NO_DEVSEL
            + N_PCI_IRDY * PROC_SIDE_ON * REQ * A[31] * /BURST * /PCI_TRDY
              * /NO_DEVSEL

N_PCI_IRDY.CLKF = MEMCLK
N_PCI_IRDY.RSTF = RESET_D
N_PCI_IRDY := { PCI_IRDY }

PCI_C_BE0.TRST = PROC_SIDE_ON * /RESET

PCI_C_BE0 =  /PCI_FRAME_D * WRITE
            + PCI_FRAME_D * /WE0 * WRITE

PCI_C_BE1.TRST = PROC_SIDE_ON * /RESET

PCI_C_BE1 =  /PCI_FRAME_D
            + PCI_FRAME_D * /WE1 * WRITE

; IF A31 IS 1 WE ARE DOING A PCI ACCESS. THEN A30 AND A29 DETERMINES WHETHER
; IT IS A MEMORY CYCLE OR A PCI CONFIGURATION CYCLE. IF A30 = 0 AND A29 = 0
; THEN IT IS A MEMORY AND IF A30 = 0 AND A29 = 1 IT IS AN I/O CYCLE WHILE
; IF A30 = 1 AND A29 = 1 IT IS A CONFIGURATION CYCLE.

PCI_C_BE2.TRST = PROC_SIDE_ON * /RESET

PCI_C_BE2 =  /PCI_FRAME_D * A[31] * /A[30] * /A[29]
            + PCI_FRAME_D * /WE2 * WRITE

PCI_C_BE3.TRST = PROC_SIDE_ON * /RESET

PCI_C_BE3 =  /PCI_FRAME_D * A[31] * A[30]
            + PCI_FRAME_D * /WE3 * WRITE

WRITE.TRST = PCI_SIDE_ON

WRITE = INTERNAL_WRITE

MINIMIZE_OFF

INTERNAL_WRITE.CLKF = MEMCLK

```

```

INTERNAL_WRITE.RSTF = RESET_D

INTERNAL_WRITE :=  PROC_SIDE_ON * WRITE
                  + PCI_SIDE_ON * /INTERNAL_WRITE * PCI_FRAME * /PCI_FRAME_D
                  * PCI_C_BE0
                  + PCI_SIDE_ON * INTERNAL_WRITE * PCI_FRAME_D

MINIMIZE_ON

; MEMORY STATE MACHINES FOR THE RAS CAS GENERATION

IDLE.CLKF = MEMCLK

IDLE :=  /IDLE * /MEM_ACCESS * /REF_ACCESS
        + RESET_D
        + IDLE * PROC_SIDE_ON * /REF_REQ * /( REQ * /A[31] * /A[30] * A[29] )
        + IDLE * PCI_SIDE_ON * /REF_REQ * /( PCI_FRAME_D * A[31] )
        + /IDLE * REF_ACCESS * /REF_REQ
        + /IDLE * PROC_SIDE_ON * MEM_ACCESS * ST1 * MEM_RDY * /BURST
        + /IDLE * PCI_SIDE_ON * MEM_ACCESS * ST1 * MEM_RDY * /PCI_FRAME
          * PCI_IRDY * PCI_TRDY
        + /IDLE * PCI_SIDE_ON * MEM_ACCESS * ST1 * MEM_RDY * PCI_FRAME
          * ADD_STOP

REF_ACCESS.CLKF = MEMCLK

REF_ACCESS :=  IDLE * REF_REQ * /REF_ACCESS
              + REF_ACCESS * REF_REQ

MEM_ACCESS.CLKF = MEMCLK

MEM_ACCESS.RSTF = RESET_D

MEM_ACCESS :=  IDLE * PROC_SIDE_ON * REQ * /A[31] * /A[30] * A[29]
              * /REF_REQ * /MEM_ACCESS
              + IDLE * PCI_SIDE_ON * PCI_FRAME_D * A[31] * /REF_REQ
              * /MEM_ACCESS
              + MEM_ACCESS * /ST1
              + MEM_ACCESS * PROC_SIDE_ON * BURST
              + MEM_ACCESS * PCI_SIDE_ON * PCI_FRAME * /ADD_STOP
              + MEM_ACCESS * PCI_SIDE_ON * /PCI_FRAME * /PCI_IRDY

ST1.CLKF = MEMCLK

ST1 :=  MEM_ACCESS

MEM_RDY.CLKF = MEMCLK

MEM_RDY.RSTF = RESET_D

MEM_RDY :=  PROC_SIDE_ON * REQ * /A[31] * /A[30] * /A[29] * /MEM_RDY
           ; ROM ACCESS
           + MEM_ACCESS * /MEM_RDY
           ; ALL DRAM ACCESSES
           ; EITHER PROCESSOR OR
           ; PCI BUS ACCESS
           + PROC_SIDE_ON * MEM_ACCESS * MEM_RDY * BURST
           + PCI_SIDE_ON * MEM_ACCESS * MEM_RDY * PCI_FRAME * /ADD_STOP
           + PCI_SIDE_ON * MEM_ACCESS * MEM_RDY * /PCI_FRAME * /PCI_IRDY

PROC_RDY =  PROC_SIDE_ON * /A[31] * /A[30] * MEM_RDY ; GOOD FOR BOTH ROM AND

```

```

; DRAM
+ PROC_SIDE_ON * /A[31] * A[30] * 85C30_RDY
+ PROC_SIDE_ON * A[31] * PCI_FRAME_D * PCI_TRDY * PCI_IRDY
+ PROC_SIDE_ON * A[31] * NO_DEVSEL
+ PROC_SIDE_ON * A[31] * PROC_ERR

```

```
MINIMIZE_OFF
```

```

RAS0 = MEM_ACCESS
      + REF_ACCESS * /MEMCLK_IN
      + RAS0 * REF_ACCESS

```

```

MUX = MEM_ACCESS * /MEMCLK_IN
      + MUX * MEM_ACCESS

```

```

CAS0 = REF_ACCESS
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * MEMCLK_IN * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS0 * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS0 * /MEMCLK_IN
      + MEM_ACCESS * ST1 * PROC_SIDE_ON * INTERNAL_WRITE * WE0 * /MEMCLK_IN
      + MEM_ACCESS * ST1 * PCI_SIDE_ON * INTERNAL_WRITE * PCI_C_BE0
        * /MEMCLK_IN

```

```

CAS1 = REF_ACCESS
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * MEMCLK_IN * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS1 * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS1 * /MEMCLK_IN
      + MEM_ACCESS * ST1 * PROC_SIDE_ON * INTERNAL_WRITE * WE1 * /MEMCLK_IN
      + MEM_ACCESS * ST1 * PCI_SIDE_ON * INTERNAL_WRITE * PCI_C_BE1
        * /MEMCLK_IN

```

```

CAS2 = REF_ACCESS
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * MEMCLK_IN * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS2 * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS2 * /MEMCLK_IN
      + MEM_ACCESS * ST1 * PROC_SIDE_ON * INTERNAL_WRITE * WE2 * /MEMCLK_IN
      + MEM_ACCESS * ST1 * PCI_SIDE_ON * INTERNAL_WRITE * PCI_C_BE2
        * /MEMCLK_IN

```

```

CAS3 = REF_ACCESS
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * MEMCLK_IN * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS3 * DELAY_IN
      + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS3 * /MEMCLK_IN
      + MEM_ACCESS * ST1 * PROC_SIDE_ON * INTERNAL_WRITE * WE3 * /MEMCLK_IN
      + MEM_ACCESS * ST1 * PCI_SIDE_ON * INTERNAL_WRITE * PCI_C_BE3
        * /MEMCLK_IN

```

```
MINIMIZE_ON
```

```
Q0.CLKF = MEMCLK
```

```
Q0.RSTF = GND
```

```
Q0.T := 85C30_CLK
```

```
Q1.CLKF = MEMCLK
```

```
Q1.T := Q0 * 85C30_CLK
```

```
Q2.CLKF = MEMCLK
```

```

Q2.T := Q1 * Q0 * 85C30_CLK

Q3.CLKF = MEMCLK
Q3.T := Q2 * Q1 * Q0 * 85C30_CLK

Q4.CLKF = MEMCLK
Q4.T := Q3 * Q2 * Q1 * Q0 * 85C30_CLK

Q5.CLKF = MEMCLK
Q5.T := Q4 * Q3 * Q2 * Q1 * Q0 * 85C30_CLK

Q6.CLKF = MEMCLK
Q6.T := Q5 * Q4 * Q3 * Q2 * Q1 * Q0 * 85C30_CLK

REF_REQ.CLKF = MEMCLK
REF_REQ := Q6 * Q5 * Q4 * Q3 * Q2 * Q1 * Q0 * 85C30_CLK
          + REF_REQ * /REF_ACCESS

; ROM_CS ALSO DRIVES THE RDN PIN OF THE 29030 AND NEEDS TO BE 1 IF THE ROM IS
; 8 BITS WIDE AND A ZERO IF IT 16 BITS WIDE DURING A RESET. SO TO DO THIS
; RESET WOULD BE OR WITH THE ROM_CS EQUATION WHEN 16 BIT MEMORY IS NEEDED.
; IT IS CUREENTLY COMMENTED OUT

ROM_CS = PROC_SIDE_ON * REQ * /A[31] * /A[30] * /A[29]
;      + RESET_D

ROM_OE = PROC_SIDE_ON * REQ * /A[31] * /A[30] * /A[29] * /WRITE

ROM_WE.CLKF = /MEMCLK
ROM_WE.RSTF = RESET_D
ROM_WE := PROC_SIDE_ON * REQ * /A[31] * /A[30] * /A[29] * WRITE * /ROM_WE

; THIS IS THE SERIAL CONTROLLER SECTION THAT CAN BE REMOVED IF THE SERIAL PORT
; IS NOT USED.

85C30_CLK.CLKF = MEMCLK
85C30_CLK.T := VCC

85C30_CS = PROC_SIDE_ON * REQ * /A[31] * A[30] * /A[29]

85C30_RD.CLKF = MEMCLK
85C30_RD := 85C30_CS*/WRITE*/85C30_RD*/85C30_RDY
          + 85C30_RD*/85C30_RDY
          + RESET_D

```

```
85C30_WR.CLKF = MEMCLK

85C30_WR := 85C30_CS*WRITE*/85C30_WR*/85C30_RDY
           + 85C30_WR*/WT_ST0*/85C30_RDY
           + 85C30_WR*/WT_ST1*/85C30_RDY
           + RESET_D

WT_ST0.CLKF = MEMCLK
WT_ST0.RSTF = RESET_D
WT_ST0 := (85C30_RD + 85C30_WR)*/WT_ST0*/85C30_RDY

WT_ST1.CLKF = MEMCLK
WT_ST1.RSTF = RESET_D
WT_ST1 := (85C30_RD + 85C30_WR)*(WT_ST0+:WT_ST1)*/85C30_RDY

85C30_RDY.CLKF = MEMCLK
85C30_RDY.RSTF = RESET_D
85C30_RDY := WT_ST0*WT_ST1*/85C30_RDY

;=====
```

Appendix D. MACH465 Equations for Pipelined Operation

;PALASM Design Description

```

;----- Declaration Segment -----
TITLE      29030 TO PCI CONVERTOR
PATTERN    PIPELINE.PDS
REVISION   A
AUTHOR     DAVID STOENNER
COMPANY    AMD
DATE       06/12/95

CHIP       Ux  MACH465

;----- PIN Declarations -----
; CLOCK PINS

PIN 187 MEMCLK

; INPUT PINS ONLY

PIN 188 MEMCLK_IN
PIN 171 /RESET_PAIR RESET_D
PIN 177 /RESET_IN

PIN 173 /REQ_PAIR REQ_D

PIN 72 /BURST

PIN 125 DELAY_IN

PIN 46 /WE0
PIN 31 /WE1
PIN 22 /WE2
PIN 21 /WE3

PIN 85 /PCI_BREQ0
PIN 84 /PCI_BREQ1
PIN 73 /PCI_BREQ2

; OUTPUT PINS ONLY

PIN 42 /PROC_RDY
PIN 49 /PROC_ERR

PIN 165 /ROM_CS
PIN 163 /ROM_OE
PIN 161 /ROM_WE

PIN 159 85C30_CLK

; INPUT / OUTPUT PINS

PIN 153 A[31]
PIN 151 A[30]
PIN 147 A[29]
PIN 150 A[28]
PIN 148 A[27]
PIN 146 A[26]
PIN 152 A[25]
PIN 149 A[24]
PIN 140 A[23]
PIN 138 A[22]
PIN 136 A[21]
PIN 142 A[20]
PIN 139 A[19]
PIN 137 A[18]

```

PIN 143 A[17]
PIN 141 A[16]
PIN 120 A[15]
PIN 122 A[14]
PIN 124 A[13]
PIN 118 A[12]
PIN 121 A[11]
PIN 123 A[10]
PIN 117 A[9]
PIN 119 A[8]
PIN 110 A[7]
PIN 112 A[6]
PIN 114 A[5]
PIN 108 A[4]
PIN 111 A[3]
PIN 113 A[2]
PIN 107 A[1]
PIN 109 A[0]

PIN 100 D[31]
PIN 98 D[30]
PIN 96 D[29]
PIN 102 D[28]
PIN 99 D[27]
PIN 97 D[26]
PIN 103 D[25]
PIN 101 D[24]
PIN 90 D[23]
PIN 88 D[22]
PIN 86 D[21]
PIN 92 D[20]
PIN 89 D[19]
PIN 87 D[18]
PIN 93 D[17]
PIN 91 D[16]
PIN 67 D[15]
PIN 69 D[14]
PIN 71 D[13]
PIN 65 D[12]
PIN 68 D[11]
PIN 70 D[10]
PIN 64 D[9]
PIN 66 D[8]
PIN 57 D[7]
PIN 59 D[6]
PIN 61 D[5]
PIN 55 D[4]
PIN 58 D[3]
PIN 60 D[2]
PIN 54 D[1]
PIN 56 D[0]

PIN 15 AD[31]
PIN 16 AD[30]
PIN 18 AD[29]
PIN 20 AD[28]
PIN 14 AD[27]
PIN 17 AD[26]
PIN 19 AD[25]
PIN 13 AD[24]
PIN 6 AD[23]
PIN 8 AD[22]
PIN 10 AD[21]
PIN 4 AD[20]
PIN 7 AD[19]
PIN 9 AD[18]
PIN 3 AD[17]
PIN 5 AD[16]

```

PIN 204 AD[15]
PIN 202 AD[14]
PIN 200 AD[13]
PIN 206 AD[12]
PIN 203 AD[11]
PIN 201 AD[10]
PIN 207 AD[9]
PIN 205 AD[8]
PIN 190 AD[7]
PIN 196 AD[6]
PIN 193 AD[5]
PIN 191 AD[4]
PIN 197 AD[3]
PIN 195 AD[2]
PIN 194 AD[1]
PIN 192 AD[0]

```

```

PIN 160 /WRITE

```

```

PIN 158 /RAS0
PIN 164 MUX

```

```

PIN 172 /CAS0
PIN 174 /CAS1
PIN 168 /CAS2
PIN 170 /CAS3

```

```

PIN 44 /PROC_BGRT

```

```

PIN 48 /PCI_BGRT0
PIN 45 /PCI_BGRT1
PIN 43 /PCI_BGRT2

```

```

PIN 35 PCI_C_BE0
PIN 36 PCI_C_BE1
PIN 34 PCI_C_BE2
PIN 32 PCI_C_BE3
PIN 37 /PCI_FRAME
PIN 33 /PCI_IRDY
PIN 39 /PCI_TRDY
PIN 38 /PCI_DEVSEL
PIN 47 /PCI_STOP

```

```

PIN 162 /85C30_CS
PIN 175 /85C30_RD
PIN 169 /85C30_WR

```

```

; ALL THE NODE DEFINITIONS

```

```

NODE ? IDLE REGISTERED
NODE ? MEM_ACCESS REGISTERED
NODE ? REF_ACCESS REGISTERED
NODE ? ST1

```

```

NODE ? REQ_D
NODE ? PCI_SIDE_ON
NODE ? AD_EN
NODE ? D_EN
NODE ? ADD_COMP
NODE ? ABORT_STOP
NODE ? N_PCI_IRDY PAIR PCI_IRDY
NODE ? N_IRDY_D

```

```

NODE ? N_PCI_TRDY PAIR PCI_TRDY

```

```

NODE ? N_PCI_STOP PAIR PCI_STOP

```

```

NODE ? CT0
NODE ? CT1
NODE ? NO_DEVSEL

NODE ? DEV_SEL_NODE
NODE ? PCI_FRAME_D
NODE ? PCI_IDLE
NODE ? REF_REQ REGISTERED

NODE ? Q0 REGISTERED
NODE ? Q1 REGISTERED
NODE ? Q2 REGISTERED
NODE ? Q3 REGISTERED
NODE ? Q4 REGISTERED
NODE ? Q5 REGISTERED
NODE ? Q6 REGISTERED

NODE ? INTERNAL_WRITE
NODE ? RESET_D
NODE ? WT_ST0
NODE ? WT_ST1
NODE ? 85C30_RDY
NODE ? N_AD[0..31]          PAIR    AD[0..31]
NODE ? N_D[0..31]          PAIR    D[0..31]
NODE ? N_A[0..31]          PAIR    A[0..31]
NODE ? N_PCI_FRAME        PAIR    PCI_FRAME
NODE ? N_PCI_C_BE0        PAIR    PCI_C_BE0
NODE ? N_PCI_C_BE1        PAIR    PCI_C_BE1
NODE ? N_PCI_C_BE2        PAIR    PCI_C_BE2
NODE ? N_PCI_C_BE3        PAIR    PCI_C_BE3

;GROUP MACH_SEG_A AD[0..7]

;GROUP MACH_SEG_B AD[8..15]

;GROUP MACH_SEG_C AD[16..23]

;GROUP MACH_SEG_D AD[24..31]

GROUP MACH_SEG_E DEV_SEL_NODE PCI_FRAME_D PCI_IDLE N_IRDY_D
;
;
;          PCI_C_BE0 PCI_C_BE1 PCI_C_BE2 PCI_C_BE3
;          PCI_FRAME PCI_IRDY PCI_TRDY PCI_DEVSEL

GROUP MACH_SEG_F CT0 CT1 ABORT_STOP NO_DEVSEL
;
;          PROC_BGRT PCI_BGRT0 PCI_BGRT1 PCI_BGRT2 PCI_STOP
;          PCI_SIDE_ON PROC_RDY PROC_ERR

GROUP MACH_SEG_G Q0 Q1 Q2 Q3 Q4 Q5 Q6 REF_REQ
;
;          D[0..7]

GROUP MACH_SEG_H AD_EN D_EN
;
;          D[8..15]

;GROUP MACH_SEG_I D[16..23]

;GROUP MACH_SEG_J D[24..31]

GROUP MACH_SEG_K ADD_COMP
;
;          A[0..7] AD_REG[2..7]

;GROUP MACH_SEG_L AD_REG[8..15]
;
;          A[8..15]

;GROUP MACH_SEG_M AD_REG[16..23]
;
;          A[16..23]

;GROUP MACH_SEG_N AD_REG[24..31]
;
;          A[24..31]

```

```

GROUP MACH_SEG_O  IDLE MEM_ACCESS REF_ACCESS ST1 INTERNAL_WRITE
;                 ROM_CS RAS0 MUX  WRITE 85C30_CS 85C30_CLK
;                 MEM_RDY

GROUP MACH_SEG_P  85C30_RDY WT_ST0 WT_ST1
;                 CAS0 CAS1 CAS2 CAS3 85C30_WR 85C30_RD

STRING ADD_INC '( PCI_FRAME_D * PCI_IRDY * PCI_TRDY )'

STRING ADD_STOP '( PCI_FRAME_D * PCI_IRDY * ADD_COMP )'

STRING ADD_HOLD '( PCI_FRAME_D * ( /PCI_IRDY + /PCI_TRDY ) )'

STRING ADD_CLK '( PCI_FRAME * /PCI_FRAME_D )'

STRING PROC_SIDE_ON '/PCI_BGRT0*/PCI_BGRT1*/PCI_BGRT2'

;----- Boolean Equation Segment -----
EQUATIONS

RESET.CLKF = MEMCLK

RESET :=  RESET_IN

RESET_D.CLKF = MEMCLK

RESET_D := RESET

; THE PCI BUS ADDRESS DATA MUX AND UNMUX

PCI_SIDE_ON = PCI_BGRT0 + PCI_BGRT1 + PCI_BGRT2

AD_EN.CLKF = MEMCLK

AD_EN :=  PROC_SIDE_ON * ( /PCI_FRAME * /PCI_FRAME_D
      + PCI_FRAME * /PCI_FRAME_D * WRITE
      + PCI_FRAME_D * WRITE ) * /RESET
      + PCI_SIDE_ON * PCI_FRAME_D * /INTERNAL_WRITE * /RESET

AD[0..31].TRST = AD_EN * /RESET

AD[0..31].RSTF = RESET_D

AD[0..31].CLKF = MEMCLK

AD[0..27] :=  A[0..27] * PROC_SIDE_ON * /PCI_FRAME * /PCI_FRAME_D * A[31]
      + D[0..27] * PCI_FRAME
      + D[0..27] * PCI_FRAME_D

AD[28..31] :=  D[28..31] * PCI_FRAME
      + D[28..31] * PCI_FRAME_D

N_AD[0..31].RSTF = RESET_D

N_AD[0..31].CLKF = MEMCLK

N_AD[0] := { AD[0] }
N_AD[1] := { AD[1] }
N_AD[2] := { AD[2] }
N_AD[3] := { AD[3] }
N_AD[4] := { AD[4] }

```

```

N_AD[5] := { AD[5] }
N_AD[6] := { AD[6] }
N_AD[7] := { AD[7] }
N_AD[8] := { AD[8] }
N_AD[9] := { AD[9] }
N_AD[10] := { AD[10] }
N_AD[11] := { AD[11] }
N_AD[12] := { AD[12] }
N_AD[13] := { AD[13] }
N_AD[14] := { AD[14] }
N_AD[15] := { AD[15] }
N_AD[16] := { AD[16] }
N_AD[17] := { AD[17] }
N_AD[18] := { AD[18] }
N_AD[19] := { AD[19] }
N_AD[20] := { AD[20] }
N_AD[21] := { AD[21] }
N_AD[22] := { AD[22] }
N_AD[23] := { AD[23] }
N_AD[24] := { AD[24] }
N_AD[25] := { AD[25] }
N_AD[26] := { AD[26] }
N_AD[27] := { AD[27] }
N_AD[28] := { AD[28] }
N_AD[29] := { AD[29] }
N_AD[30] := { AD[30] }
N_AD[31] := { AD[31] }

A[0..31].TRST = PCI_SIDE_ON * /RESET

A[0..31].RSTF = RESET_D

A[0..31].CLKF = MEMCLK

A[0..1] := GND

A[6..31] := PCI_SIDE_ON * AD[6..31] * ADD_CLK
           + PCI_SIDE_ON * N_A[6..31] * PCI_FRAME_D

A[5].T := PCI_SIDE_ON * PCI_FRAME * /PCI_FRAME_D * N_A[5] * /AD[5]
          + PCI_SIDE_ON * PCI_FRAME * /PCI_FRAME_D * /N_A[5] * AD[5]
          + PCI_SIDE_ON * PCI_FRAME_D * PCI_IRDY * PCI_TRDY
            * N_A[2] * N_A[3] * N_A[4]

A[4].T := PCI_SIDE_ON * PCI_FRAME * /PCI_FRAME_D * N_A[4] * /AD[4]
          + PCI_SIDE_ON * PCI_FRAME * /PCI_FRAME_D * /N_A[4] * AD[4]
          + PCI_SIDE_ON * PCI_FRAME_D * PCI_IRDY * PCI_TRDY
            * N_A[2] * N_A[3]

A[3].T := PCI_SIDE_ON * PCI_FRAME * /PCI_FRAME_D * N_A[3] * /AD[3]
          + PCI_SIDE_ON * PCI_FRAME * /PCI_FRAME_D * /N_A[3] * AD[3]
          + PCI_SIDE_ON * PCI_FRAME_D * PCI_IRDY * PCI_TRDY
            * N_A[2]

A[2] := PCI_SIDE_ON * AD[2] * PCI_FRAME * /PCI_FRAME_D
        + PCI_SIDE_ON * PCI_FRAME_D * N_A[2] * /PCI_IRDY
        + PCI_SIDE_ON * PCI_FRAME_D * N_A[2] * /PCI_TRDY
        + PCI_FRAME_D * /N_A[2] * PCI_IRDY * PCI_TRDY

N_A[0..31].RSTF = RESET_D

N_A[0..31].CLKF = MEMCLK

```

```

N_A[0] := { A[0] }
N_A[1] := { A[1] }
N_A[2] := { A[2] }
N_A[3].T := { A[3].T }
N_A[4].T := { A[4].T }
N_A[5].T := { A[5].T }
N_A[6] := { A[6] }
N_A[7] := { A[7] }
N_A[8] := { A[8] }
N_A[9] := { A[9] }
N_A[10] := { A[10] }
N_A[11] := { A[11] }
N_A[12] := { A[12] }
N_A[13] := { A[13] }
N_A[14] := { A[14] }
N_A[15] := { A[15] }
N_A[16] := { A[16] }
N_A[17] := { A[17] }
N_A[18] := { A[18] }
N_A[19] := { A[19] }
N_A[20] := { A[20] }
N_A[21] := { A[21] }
N_A[22] := { A[22] }
N_A[23] := { A[23] }
N_A[24] := { A[24] }
N_A[25] := { A[25] }
N_A[26] := { A[26] }
N_A[27] := { A[27] }
N_A[28] := { A[28] }
N_A[29] := { A[29] }
N_A[30] := { A[30] }
N_A[31] := { A[31] }

```

```
D_EN.CLKF = MEMCLK
```

```
D_EN := PROC_SIDE_ON * PCI_FRAME_D * /WRITE * /RESET_D
        + PCI_SIDE_ON * PCI_FRAME_D * INTERNAL_WRITE * /RESET_D
```

```
D[0..31].TRST = D_EN
```

```
D[0..31].RSTF = RESET_D
```

```
D[0..31].CLKF = MEMCLK
```

```
D[0..31] := AD[0..31]
```

```
N_D[0..31].RSTF = RESET_D
```

```
N_D[0..31].CLKF = MEMCLK
```

```

N_D[0] := { D[0] }
N_D[1] := { D[1] }
N_D[2] := { D[2] }
N_D[3] := { D[3] }
N_D[4] := { D[4] }
N_D[5] := { D[5] }
N_D[6] := { D[6] }
N_D[7] := { D[7] }
N_D[8] := { D[8] }
N_D[9] := { D[9] }
N_D[10] := { D[10] }
N_D[11] := { D[11] }
N_D[12] := { D[12] }
N_D[13] := { D[13] }

```

```

N_D[14] := { D[14] }
N_D[15] := { D[15] }
N_D[16] := { D[16] }
N_D[17] := { D[17] }
N_D[18] := { D[18] }
N_D[19] := { D[19] }
N_D[20] := { D[20] }
N_D[21] := { D[21] }
N_D[22] := { D[22] }
N_D[23] := { D[23] }
N_D[24] := { D[24] }
N_D[25] := { D[25] }
N_D[26] := { D[26] }
N_D[27] := { D[27] }
N_D[28] := { D[28] }
N_D[29] := { D[29] }
N_D[30] := { D[30] }
N_D[31] := { D[31] }

```

```
ADD_COMP = N_A[5] * N_A[4] * N_A[3] * N_A[2]
```

```
; BUS ARBITOR AND BUS GRANT STATE MACHINES
```

```
REQ_D.CLKF = MEMCLK
```

```
REQ_D := REQ
```

```
PROC_BGRT.CLKF = MEMCLK
```

```
PROC_BGRT := /PROC_BGRT * /PCI_BREQ0 * /PCI_BREQ1 * /PCI_BREQ2
             * /PCI_FRAME_D * /RESET_D
             + PROC_BGRT * /PCI_BREQ0 * /PCI_BREQ1 * /PCI_BREQ2 * /RESET_D
```

```
PCI_BGRT0.CLKF = MEMCLK
```

```
PCI_BGRT0.TRST = /RESET
```

```
PCI_BGRT0 := /PCI_BGRT0 * PCI_BREQ0 * /PROC_BGRT * /REQ * /REQ_D
             * /PCI_BGRT1 * /PCI_BGRT2 * /PCI_FRAME_D * /RESET_D
             + PCI_BREQ0 * PCI_BGRT0 * /RESET_D
             + PCI_BGRT0 * PCI_FRAME_D * /RESET_D
```

```
PCI_BGRT1.CLKF = MEMCLK
```

```
PCI_BGRT1.TRST = /RESET
```

```
PCI_BGRT1 := /PCI_BGRT1 * PCI_BREQ1 * /PCI_BREQ0 * /PROC_BGRT
             * /REQ * /REQ_D * /PCI_BGRT0 * /PCI_BGRT2
             * /PCI_FRAME_D * /RESET_D
             + PCI_BREQ1 * PCI_BGRT1 * /RESET_D
             + PCI_BGRT1 * PCI_FRAME_D * /RESET_D
```

```
PCI_BGRT2.CLKF = MEMCLK
```

```
PCI_BGRT2.TRST = /RESET
```

```
PCI_BGRT2 := /PCI_BGRT2 * PCI_BREQ2 * /PCI_BREQ1 * /PCI_BREQ0
             * /PROC_BGRT * /REQ * /REQ_D * /PCI_BGRT0 * /PCI_BGRT1
             * /PCI_FRAME_D * /RESET_D
             + PCI_BREQ2 * PCI_BGRT2 * /RESET_D
```

```

+ PCI_BGRT2 * PCI_FRAME_D * /RESET_D

; PCI BUS CONTROL LOGIC

PCI_FRAME.TRST = PROC_SIDE_ON * /RESET

PCI_FRAME.CLKF = MEMCLK

PCI_FRAME.RSTF = RESET_D

PCI_FRAME := PROC_SIDE_ON * PCI_IDLE * REQ * A[31] * /PCI_FRAME
;           + PROC_SIDE_ON * REQ * A[31] * BURST * PCI_FRAME
           + PROC_SIDE_ON * REQ * A[31] * BURST * PCI_FRAME_D * /PROC_ERR
           * /NO_DEVSEL

N_PCI_FRAME.CLKF = MEMCLK

N_PCI_FRAME.RSTF = RESET_D

N_PCI_FRAME := { PCI_FRAME }

PCI_FRAME_D.CLKF = MEMCLK

PCI_FRAME_D.RSTF = RESET_D

; PROC_ERR IS USED IN THE SECOND AND THIRD EQUATIONS FOR THE ADVENT AN
; EARLY TERMINATION OCCURS.

PCI_FRAME_D := PROC_SIDE_ON * PCI_FRAME * /PCI_FRAME_D
+ PROC_SIDE_ON * PCI_FRAME_D * REQ * A[31] * BURST * /PROC_ERR
  * /NO_DEVSEL
+ PROC_SIDE_ON * PCI_FRAME_D * REQ * A[31] * /BURST
  * /PCI_TRDY * /PROC_ERR * /NO_DEVSEL
+ PROC_SIDE_ON * PCI_FRAME_D * REQ * A[31] * /BURST
  * /PCI_IRDY * /PROC_ERR * /NO_DEVSEL
+ PCI_SIDE_ON * PCI_FRAME * / ( PCI_IRDY * ADD_COMP )
  * /ABORT_STOP
+ PCI_SIDE_ON * PCI_FRAME_D * /PCI_FRAME * /PCI_IRDY
+ PCI_SIDE_ON * PCI_FRAME_D * /PCI_FRAME * /PCI_TRDY

PCI_IDLE.CLKF = MEMCLK

PCI_IDLE.RSTF = RESET_D

PCI_IDLE := /PCI_FRAME * /PCI_FRAME_D

CT0.CLKF = MEMCLK

CT0 := PCI_FRAME_D * /PCI_DEVSEL * ( /CT0 + CT0 * CT1 )

CT1.CLKF = MEMCLK

CT1 := PCI_FRAME_D * /PCI_DEVSEL * (( CT0 :+: CT1 ) + CT0 * CT1 )

NO_DEVSEL.CLKF = MEMCLK

NO_DEVSEL := /NO_DEVSEL * PROC_SIDE_ON * REQ * A[31] * CT1 * CT0
  * /PCI_DEVSEL

```

```

PROC_ERR.CLKF = MEMCLK

PROC_ERR :=   PROC_SIDE_ON * REQ * A[31] * PCI_STOP * /PROC_RDY

PCI_STOP.TRST = PCI_SIDE_ON * DEV_SEL_NODE * /RESET

PCI_STOP.CLKF = MEMCLK

PCI_STOP :=   /PCI_STOP * PCI_FRAME * DEV_SEL_NODE * ADD_STOP

N_PCI_STOP .CLKF = MEMCLK

N_PCI_STOP := { PCI_STOP }

ABORT_STOP.CLKF = MEMCLK

ABORT_STOP :=   /ABORT_STOP * PCI_SIDE_ON * PCI_FRAME * PCI_FRAME_D
                * ADD_STOP
                + ABORT_STOP * PCI_SIDE_ON * PCI_FRAME

PCI_DEVSEL.TRST = PCI_SIDE_ON * DEV_SEL_NODE * /RESET

PCI_DEVSEL = PCI_FRAME_D

DEV_SEL_NODE.CLKF = MEMCLK

DEV_SEL_NODE.RSTF = RESET_D

DEV_SEL_NODE :=   PCI_SIDE_ON * PCI_FRAME * AD[31] * /PCI_FRAME_D
                  * /ABORT_STOP
                  + PCI_SIDE_ON * DEV_SEL_NODE * PCI_FRAME_D * /ABORT_STOP
                  + PROC_SIDE_ON * PCI_FRAME_D * /DEV_SEL_NODE * PCI_DEVSEL
                  + PROC_SIDE_ON * PCI_FRAME_D * DEV_SEL_NODE

PCI_TRDY.TRST = PCI_SIDE_ON * DEV_SEL_NODE * /RESET

PCI_TRDY.CLKF = MEMCLK

PCI_TRDY.RSTF = RESET_D

PCI_TRDY :=   PCI_SIDE_ON * MEM_ACCESS * ST1 * /N_PCI_TRDY
              * /INTERNAL_WRITE
              + PCI_SIDE_ON * MEM_ACCESS * N_PCI_TRDY * PCI_FRAME * /ADD_STOP
              * /INTERNAL_WRITE
              + PCI_SIDE_ON * MEM_ACCESS * N_PCI_TRDY * /PCI_FRAME * /PCI_IRDY
              * /INTERNAL_WRITE
              + PCI_SIDE_ON * MEM_ACCESS * ST1 * /N_PCI_TRDY
              * INTERNAL_WRITE
;              + PCI_SIDE_ON * MEM_ACCESS * N_PCI_TRDY * PCI_FRAME * /ADD_STOP
;              * INTERNAL_WRITE
              + PCI_SIDE_ON * MEM_ACCESS * N_PCI_TRDY * /PCI_FRAME * /PCI_IRDY
              * INTERNAL_WRITE

N_PCI_TRDY.CLKF = MEMCLK

N_PCI_TRDY.RSTF = RESET_D

N_PCI_TRDY := { PCI_TRDY }

```

```

PCI_IRDY.TRST = PROC_SIDE_ON * /RESET

PCI_IRDY.CLKF = MEMCLK

PCI_IRDY.RSTF = RESET_D

PCI_IRDY :=  PROC_SIDE_ON * PCI_FRAME * /N_PCI_IRDY * /N_IRDY_D
             + N_PCI_IRDY * PROC_SIDE_ON * REQ * A[31] * BURST * /NO_DEVSEL
               * /WRITE
             + N_PCI_IRDY * PROC_SIDE_ON * REQ * A[31] * /BURST * /PCI_TRDY
               * /NO_DEVSEL * /WRITE
             + N_PCI_IRDY * PROC_SIDE_ON * REQ * A[31] * BURST * /NO_DEVSEL
               * WRITE * /PCI_TRDY
             + N_PCI_IRDY * PROC_SIDE_ON * REQ * A[31] * /BURST * /PCI_TRDY
               * /NO_DEVSEL * WRITE

N_PCI_IRDY.CLKF = MEMCLK

N_PCI_IRDY.RSTF = RESET_D

N_PCI_IRDY :=  { PCI_IRDY }

N_IRDY_D.CLKF = MEMCLK

N_IRDY_D := N_PCI_IRDY

PCI_C_BE0.TRST = PROC_SIDE_ON * /RESET

PCI_C_BE0.CLKF = MEMCLK

PCI_C_BE0 :=  /PCI_FRAME * /PCI_FRAME_D * WRITE
             + PCI_FRAME * /PCI_FRAME_D * /WE0 * WRITE
             + PCI_FRAME_D * /WE0 * WRITE

N_PCI_C_BE0.CLKF = MEMCLK

N_PCI_C_BE0 := { PCI_C_BE0 }

PCI_C_BE1.TRST = PROC_SIDE_ON * /RESET

PCI_C_BE1.CLKF = MEMCLK

PCI_C_BE1 :=  /PCI_FRAME * /PCI_FRAME_D
             + PCI_FRAME * /PCI_FRAME_D * /WE1 * WRITE
             + PCI_FRAME_D * /WE1 * WRITE

N_PCI_C_BE1.CLKF = MEMCLK

N_PCI_C_BE1 := { PCI_C_BE1 }

; IF A31 IS 1 WE ARE DOING A PCI ACCESS. THEN A30 AND A29 DETERMINES WHETHER
; IT IS A MEMORY CYCLE OR A PCI CONFIGURATION CYCLE. IF A30 = 0 AND A29 = 0
; THEN IT IS A MEMORY AND IF A30 = 0 AND A29 = 1 IT IS AN I/O CYCLE WHILE
; IF A30 = 1 AND A29 = 1 IT IS A CONFIGURATION CYCLE.

PCI_C_BE2.TRST = PROC_SIDE_ON * /RESET

```

```

PCI_C_BE2.CLKF = MEMCLK

PCI_C_BE2 := /PCI_FRAME * /PCI_FRAME_D * A[31] * /A[30] * /A[29]
            + PCI_FRAME * /PCI_FRAME * /WE2 * WRITE
            + PCI_FRAME_D * /WE2 * WRITE

N_PCI_C_BE2.CLKF = MEMCLK

N_PCI_C_BE2 := { PCI_C_BE2 }

PCI_C_BE3.TRST = PROC_SIDE_ON * /RESET

PCI_C_BE3.CLKF = MEMCLK

PCI_C_BE3 := /PCI_FRAME * /PCI_FRAME_D * A[31] * A[30]
            + PCI_FRAME * /PCI_FRAME_D * /WE3 * WRITE
            + PCI_FRAME_D * /WE3 * WRITE

N_PCI_C_BE3.CLKF = MEMCLK

N_PCI_C_BE3 := { PCI_C_BE3 }

WRITE.TRST = PCI_SIDE_ON

WRITE = INTERNAL_WRITE

MINIMIZE_OFF

INTERNAL_WRITE.CLKF = MEMCLK

INTERNAL_WRITE.RSTF = RESET_D

INTERNAL_WRITE := PROC_SIDE_ON * WRITE
                 + PCI_SIDE_ON * /INTERNAL_WRITE * PCI_FRAME * /PCI_FRAME_D
                   * PCI_C_BE0
                 + PCI_SIDE_ON * INTERNAL_WRITE * PCI_FRAME_D

MINIMIZE_ON

; MEMORY STATE MACHINES FOR THE RAS CAS GENERATION

IDLE.CLKF = MEMCLK

IDLE := /IDLE * /MEM_ACCESS * /REF_ACCESS
       + RESET_D
       + IDLE * PROC_SIDE_ON * /REF_REQ * /( REQ * /A[31] * /A[30] * A[29] )
       + IDLE * PCI_SIDE_ON * /REF_REQ * /( PCI_FRAME_D * A[31] )
       + /IDLE * REF_ACCESS * /REF_REQ
       + /IDLE * PROC_SIDE_ON * MEM_ACCESS * ST1 * PROC_RDY * /BURST
       + /IDLE * PCI_SIDE_ON * MEM_ACCESS * ST1 * /PCI_FRAME
         * PCI_IRDY * PCI_TRDY
       + /IDLE * PCI_SIDE_ON * MEM_ACCESS * ST1 * PCI_FRAME * ADD_STOP
         * PCI_IRDY * PCI_TRDY

REF_ACCESS.CLKF = MEMCLK

REF_ACCESS := IDLE * REF_REQ * /REF_ACCESS
             + REF_ACCESS * REF_REQ

MEM_ACCESS.CLKF = MEMCLK

```

```

MEM_ACCESS.RSTF = RESET_D

MEM_ACCESS :=  IDLE * PROC_SIDE_ON * REQ * /A[31] * /A[30] * A[29]
               * /REF_REQ * /MEM_ACCESS
               + IDLE * PCI_SIDE_ON * PCI_FRAME_D * A[31] * /REF_REQ
               * /MEM_ACCESS
               + MEM_ACCESS * /ST1
               + MEM_ACCESS * PROC_SIDE_ON * BURST
               + MEM_ACCESS * PCI_SIDE_ON * PCI_FRAME * /ADD_STOP
               + MEM_ACCESS * PCI_SIDE_ON * /PCI_FRAME * /PCI_IRDY
               + MEM_ACCESS * PCI_SIDE_ON * /PCI_FRAME * /PCI_TRDY

ST1.CLKF = MEMCLK

ST1 :=  MEM_ACCESS

PROC_RDY.CLKF = MEMCLK

PROC_RDY.RSTF = RESET_D

PROC_RDY :=  PROC_SIDE_ON * REQ * /A[31] * /A[30] * /A[29] * /PROC_RDY
               ; ROM ACCESS
               + PROC_SIDE_ON * MEM_ACCESS * /PROC_RDY
               ; FIRST DRAM ACCESS
               + PROC_SIDE_ON * MEM_ACCESS * BURST * PROC_RDY
               ; BURST DRAM ACCESS
               + PROC_SIDE_ON * REQ * /A[31] * A[30] * WT_ST0 * WT_ST1 * /PROC_RDY
               ; 85C30 ACCESS
               + PROC_SIDE_ON * REQ * A[31] * PCI_FRAME_D * PCI_TRDY
               * N_PCI_IRDY
               ; PCI TRANSFER
               + PROC_SIDE_ON * REQ * A[31] * CT1 * CT0 * /PCI_DEVSEL * /PROC_RDY
               ; NO DEV SEL
               + PROC_SIDE_ON * REQ * A[31] * PCI_STOP * /PROC_RDY
               ; PCI CYCLE STOP

MINIMIZE_OFF

RAS0 =  MEM_ACCESS
       + REF_ACCESS * /MEMCLK_IN
       + RAS0 * REF_ACCESS

MUX =  MEM_ACCESS * /MEMCLK_IN
      + MUX * MEM_ACCESS

CAS0 =  REF_ACCESS
       + MEM_ACCESS * ST1 * /INTERNAL_WRITE * MEMCLK_IN * DELAY_IN
       + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS0 * DELAY_IN
       + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS0 * /MEMCLK_IN
       + MEM_ACCESS * ST1 * PROC_SIDE_ON * INTERNAL_WRITE * WE0 * /MEMCLK_IN
       + MEM_ACCESS * ST1 * PCI_SIDE_ON * INTERNAL_WRITE * PCI_C_BE0
         * /MEMCLK_IN

CAS1 =  REF_ACCESS
       + MEM_ACCESS * ST1 * /INTERNAL_WRITE * MEMCLK_IN * DELAY_IN
       + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS1 * DELAY_IN
       + MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS1 * /MEMCLK_IN
       + MEM_ACCESS * ST1 * PROC_SIDE_ON * INTERNAL_WRITE * WE1 * /MEMCLK_IN
       + MEM_ACCESS * ST1 * PCI_SIDE_ON * INTERNAL_WRITE * PCI_C_BE1
         * /MEMCLK_IN

```

```

CAS2 = REF_ACCESS
+ MEM_ACCESS * ST1 * /INTERNAL_WRITE * MEMCLK_IN * DELAY_IN
+ MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS2 * DELAY_IN
+ MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS2 * /MEMCLK_IN
+ MEM_ACCESS * ST1 * PROC_SIDE_ON * INTERNAL_WRITE * WE2 * /MEMCLK_IN
+ MEM_ACCESS * ST1 * PCI_SIDE_ON * INTERNAL_WRITE * PCI_C_BE2
  * /MEMCLK_IN

CAS3 = REF_ACCESS
+ MEM_ACCESS * ST1 * /INTERNAL_WRITE * MEMCLK_IN * DELAY_IN
+ MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS3 * DELAY_IN
+ MEM_ACCESS * ST1 * /INTERNAL_WRITE * CAS3 * /MEMCLK_IN
+ MEM_ACCESS * ST1 * PROC_SIDE_ON * INTERNAL_WRITE * WE3 * /MEMCLK_IN
+ MEM_ACCESS * ST1 * PCI_SIDE_ON * INTERNAL_WRITE * PCI_C_BE3
  * /MEMCLK_IN

MINIMIZE_ON

Q0.CLKF = MEMCLK

Q0.RSTF = GND

Q0.T := 85C30_CLK

Q1.CLKF = MEMCLK

Q1.T := Q0 * 85C30_CLK

Q2.CLKF = MEMCLK

Q2.T := Q1 * Q0 * 85C30_CLK

Q3.CLKF = MEMCLK

Q3.T := Q2 * Q1 * Q0 * 85C30_CLK

Q4.CLKF = MEMCLK

Q4.T := Q3 * Q2 * Q1 * Q0 * 85C30_CLK

Q5.CLKF = MEMCLK

Q5.T := Q4 * Q3 * Q2 * Q1 * Q0 * 85C30_CLK

Q6.CLKF = MEMCLK

Q6.T := Q5 * Q4 * Q3 * Q2 * Q1 * Q0 * 85C30_CLK

REF_REQ.CLKF = MEMCLK

REF_REQ := Q6 * Q5 * Q4 * Q3 * Q2 * Q1 * Q0 * 85C30_CLK
  + REF_REQ * /REF_ACCESS

; ROM_CS ALSO DRIVES THE RDN PIN OF THE 29030 AND NEEDS TO BE 1 IF THE ROM IS
; 8 BITS WIDE AND A ZERO IF IT 16 BITS WIDE DURING A RESET. SO TO DO THIS
; RESET WOULD BE OR WITH THE ROM_CS EQUATION WHEN 16 BIT MEMORY IS NEEDED.

```

```

; IT IS CUREENTLY COMMENTED OUT

ROM_CS = PROC_SIDE_ON * REQ * /A[31] * /A[30] * /A[29]
; + RESET_D

ROM_OE = PROC_SIDE_ON * REQ * /A[31] * /A[30] * /A[29] * /WRITE

ROM_WE.CLKF = /MEMCLK
ROM_WE.RSTF = RESET_D

ROM_WE := PROC_SIDE_ON * REQ * /A[31] * /A[30] * /A[29] * WRITE * /ROM_WE

; THIS IS THE SERIAL CONTROLLER SECTION THAT CAN BE REMOVED IF THE SERIAL PORT
; IS NOT USED.

85C30_CLK.CLKF = MEMCLK
85C30_CLK.T := VCC

85C30_CS = PROC_SIDE_ON * REQ * /A[31] * A[30]

85C30_RD.CLKF = MEMCLK

85C30_RD := 85C30_CS*/WRITE*/85C30_RD*/85C30_RDY
+ 85C30_RD*/85C30_RDY
+ RESET_D

85C30_WR.CLKF = MEMCLK

85C30_WR := 85C30_CS*WRITE*/85C30_WR*/85C30_RDY
+ 85C30_WR*/WT_ST0*/85C30_RDY
+ 85C30_WR*/WT_ST1*/85C30_RDY
+ RESET_D

WT_ST0.CLKF = MEMCLK
WT_ST0.RSTF = RESET_D

WT_ST0 := (85C30_RD + 85C30_WR)*/WT_ST0*/85C30_RDY

WT_ST1.CLKF = MEMCLK
WT_ST1.RSTF = RESET_D

WT_ST1 := (85C30_RD + 85C30_WR) * (WT_ST0:+:WT_ST1)*/85C30_RDY

85C30_RDY.CLKF = MEMCLK
85C30_RDY.RSTF = RESET_D

85C30_RDY := WT_ST0*WT_ST1*/85C30_RD

;=====

```

Trademarks

Copyright © 1998 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc.

Am186, Am386, Am486, Am29000, bIMR, eIMR, eIMR+, GigaPHY, HIMIB, ILACC, IMR, IMR+, IMR2, ISA-HUB, MACE, Magic Packet, PCnet, PCnet-FAST, PCnet-FAST+, PCnet-Mobile, QFEX, QFEXr, QuASI, QuEST, QuLET, TAXIchip, TPEX, and TPEX Plus are trademarks of Advanced Micro Devices, Inc.

Microsoft is a registered trademark of Microsoft Corporation.

Product names used in this publication are for identification purposes only and may be trademarks of their respective companies.