AP-704

APPLICATION NOTE

A Simple DRAM Controller for 25/16 MHz i960® CA/CF Microprocessors

Rick Schue
i960® Microprocesor Architecture Specialist

Intel Corporation
Embedded Processor Division
Mail Stop CH5-233
5000 W. Chandler Blvd.
Chandler, Arizona 85226

February 20, 1995

Order Number: 272628-001

A SIMPLE DRAM CONTROLLER FOR 25/16 MHZ i960® CA/CF MICROPROCESSORS

FIGURES

TABLES

Additional copies of this document or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

1.0 INTRODUCTION

This application note describes a simple DRAM controller for use with 25 and 16 MHz i960® Cx processors. Other application notes are available which describe DRAM controllers for the i960 Cx and Jx processors; see Section 7.0, RELATED INFORMATION for ordering information.

This document contains general DRAM controller theory, state machine definitions and timing diagrams. It also contains the PLD equations used to build and test the prototype design. All timing analysis was verified with Timing Designer*. PLD equations were implemented in ABEL* as a device-independent design. Design features include:

- a simple low-cost non-interleaved design
- an ability to use standard 70 ns DRAM SIMM*
- 2-1-1-1 wait state burst reads
- 2-1-1-1 wait state burst writes

This design was implemented and tested on real hardware. The timing analysis, schematics and PLD files are available through Intel's America's Application Support BBS, at (916) 356-3600.

1.1 Design Goals

The primary goal of this design is to implement a single bank 32-bit DRAM controller with the minimum number of components, using a conventional 72-pin fast page mode DRAM SIMM. Such a design may be useful in small embedded systems where space is at a premium. Accordingly, the controller design avoids such techniques as bank interleaving, write posting, and parity support. The state machines require only a 1x frequency clock - no frequency multipliers or delay elements are needed. As tested, the controller also generates the basic control signals needed to support flash memory, SRAM, and I/O peripherals residing on an 8-bit buffered bus.

Although an Altera* EPX740 FLEXlogic device was used to design the prototype, none of the advanced features of the FLEXlogic family were used, making the design easy to port to other programmable logic families. Appendix B describes a specific implementation in simple PLDs.

1.2 Page Mode DRAM SIMM Review

Page mode DRAM allows faster memory access by keeping the same row address while selecting random column addresses within that row. A new column address is selected by deasserting $\overline{CAS}$ while keeping $\overline{RAS}$ active and then asserting $\overline{CAS}$ with the new column address valid to the DRAM. Page mode operation works very well with burst buses in which a single address cycle can be followed by multiple data cycles.

The individual DRAM $\overline{WE}$ signals are tied to common $\overline{WE}$ pin on the SIMM; this requires the use of early write cycles. In an early write cycle, write data is referenced to the falling edge of $\overline{CAS}$, not the falling edge of $\overline{WE}$.

Each SIMM also has four $\overline{CAS}$ lines, one for every eight (nine) bits in a 32-bit (36-bit) SIMM module. The four $\overline{CAS}$ lines control the writing to individual bytes within each SIMM.

1.3 Burst Capabilities for 32-Bit Bus

A bus access starts by asserting $\overline{ADS}$ in the address cycle, and ends by asserting $\overline{BLAST}$ in the last data cycle. Figure 1 shows $\overline{ADS}$ and $\overline{BLAST}$ timings for a quad-word access.

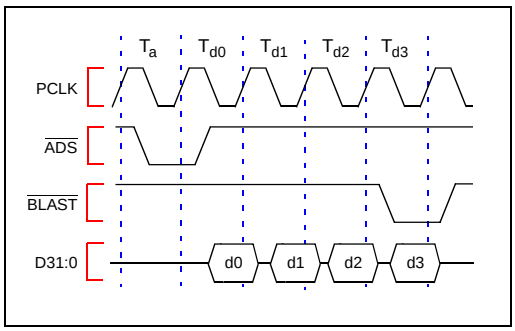


Figure 1. Quad-Word Access Example Showing $\overline{ADS}$ and $\overline{BLAST}$ Timings

2.0 DRAM CONTROLLER OVERVIEW

Figure 2 shows a block diagram of a typical system using this DRAM controller. The design comprises four distinct blocks: control logic, address path, data path, and the DRAM SIMM. This section describes each block.

2.1 Control Logic

The controller consists of several state machines implemented in a programmable logic device (PLD). All state machines are driven by the processor's 1x PCLK.

Design of this controller began by sketching out the desired DRAM control signals on paper, using PCLK as a time base. Individual state machines were then defined by observing the points on the waveforms where the various control signals needed to assert and deassert. These conditions were then recorded as Boolean expressions and transferred to ABEL syntax.

Rather than creating a large state machine to define numerous control outputs, this design uses a large number of simple (two-state) machines — one for each output signal. This technique was chosen to reduce the number of PLD resources required.

The DRAM controller performs address decoding internally. The address bus' upper three bits determine if a processor cycle is to be handled by the DRAM controller. If a finer range is required, additional address bits can be decoded or an external decoder can be added.

Byte and short writes are supported with individual $\overline{CAS}$ signals, one for each byte of the 32-bit memory. Unaccessed bytes during write cycles see a $\overline{RAS}$ only refresh cycle. All write cycles are "early writes", where $\overline{WE}$ is asserted prior to $\overline{CAS}$.

Refresh cycles are generated approximately every 15 µs by dividing down an external 1.8432 MHz baud rate clock. Other refresh clock frequencies can be accommodated with minimal changes. $\overline{CAS}$ before $\overline{RAS}$ refresh cycles are used.

Wait states are inserted by the DRAM controller by providing a $\overline{READY}$ signal to the processor. If additional ready controls are needed for other peripherals, they can be externally OR'ed with the DRAM controller's $\overline{DRDY}$ output.

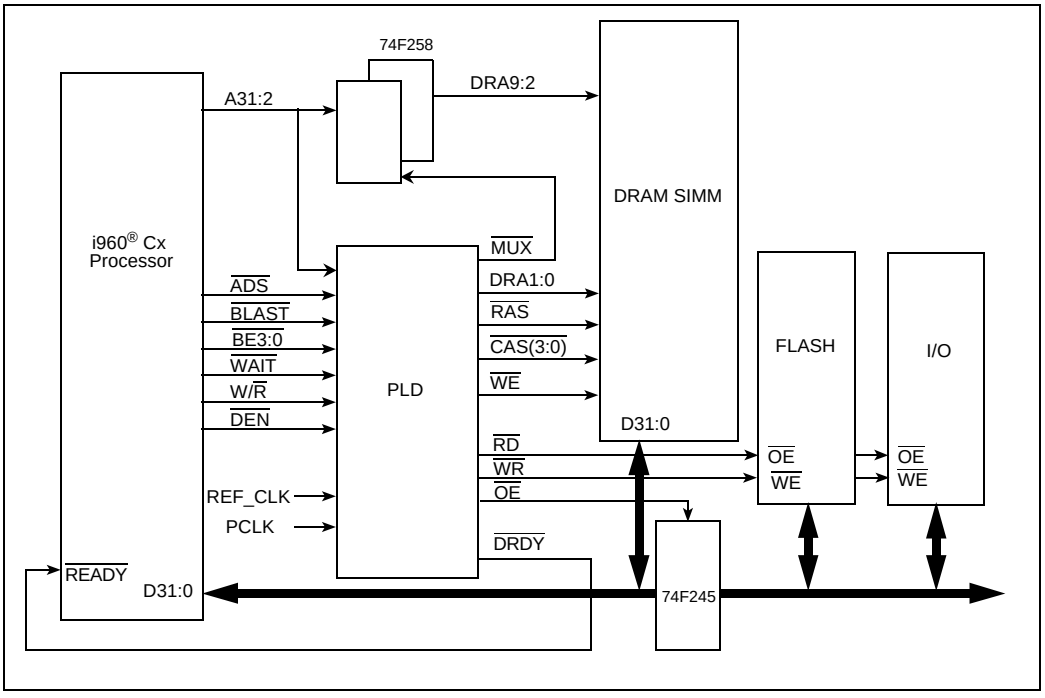


Figure 2. Typical DRAM Controller Design

2.2 Address Path

DRAM address multiplexing is provided by two 74F258 quad 2:1 multiplexers. These devices switch the upper 8 bits of the 10-bit DRAM address bus. The lower two address bits are generated in the PLD to provide faster column address during the page mode bursts. The address multiplexers also provide buffering for the capacitive load of the DRAM SIMM.

Use of larger SIMMs would require an additional 74F258 to switch the extra address lines.

2.3 Data Path

The DRAM data path is unbuffered. All other peripherals in this example are 8-bit devices and are buffered via a 74F245 transceiver. This eliminates the data buffer delay from the DRAM-critical path and places it in the slower (and less critical) I/O path. It also reduces the number of required 74F245 transceivers. Since the DRAM array is the only 32-bit device in the design, the DRAM devices see a very lightly loaded data bus — just the CPU and the 74F245 I/O buffer.

2.4 SIMMS

The SIMM block consists of one standard 72-pin SIMM socket. This design does not use the x36 SIMM parity bits. However, the x36 SIMMs are standard for PCs and workstations; as such, they are readily available. The only penalty is more loading on the address and control lines due to the extra DRAM devices of the x36 SIMM.

All address and control lines use series-damping resistors to control ringing.

3.0 THEORY OF OPERATION

The controller generates the following signals:

- $\overline{RAS}$ — DRAM row address strobe, active low
- $\overline{CAS3:0}$ — Four column address strobes, one for each byte, active low
- $\overline{MUX}$ — Control signal for the 74F258 row/column address multiplexers
- $\overline{DRDY}$ — $\overline{READY}$ control to the i960 Cx processor

The DRAM controller uses the processor's $\overline{READY}$ signal to control wait states. The processor's MCON register is initialized as follows: $N_{XAD} = N_{XDA} = N_{XDD} = 0$, $\overline{READY}$ enable = 1, Burst Enable = 1.

Figure 3 shows non-burst read and write cycles. The DRAM controller begins a processor cycle when it sees $\overline{ADS}$ and a valid address range on the upper address bits A31:29. This occurs on the rising edge of PCLK that ends the address state (T_a) of the bus cycle. On this same edge $\overline{RAS}$ is asserted. One clock later the address multiplexer is switched with the assertion of the $\overline{MUX}$ signal. On the following clock one or more of the $\overline{CAS}$ signals are asserted, based on which byte enables are driven active. $\overline{CAS}$ is low for one clock.

If a burst cycle is requested, $\overline{CAS}$ is deasserted for one clock (hence the one wait state) and reasserted for another clock. This process repeats until $\overline{BLAST}$ is asserted, as shown in Figure 4.

Reads and writes are handled exactly the same way by the RAS and CAS state machines. DRAM $\overline{WE}$ control is provided by the processor's W/$\overline{R}$.

Refresh cycles, as shown in Figure 5, start with the assertion of all four $\overline{CAS}$'s. One clock later $\overline{RAS}$ is asserted and, on the following clock, the $\overline{CAS}$'s are deasserted. On the next clock $\overline{RAS}$ is deasserted. To provide ample row precharge time, a subsequent processor cycle does not start until $\overline{RAS}$ is high for at least two clocks. Refresh requests are recognized during idle bus clocks (T_i) or during the address state (T_a) that starts a processor cycle. Refresh requests are given priority over processor requests. Once a refresh cycle starts, processor requests are delayed until the refresh completes. An example of this can be seen in Figure 8.

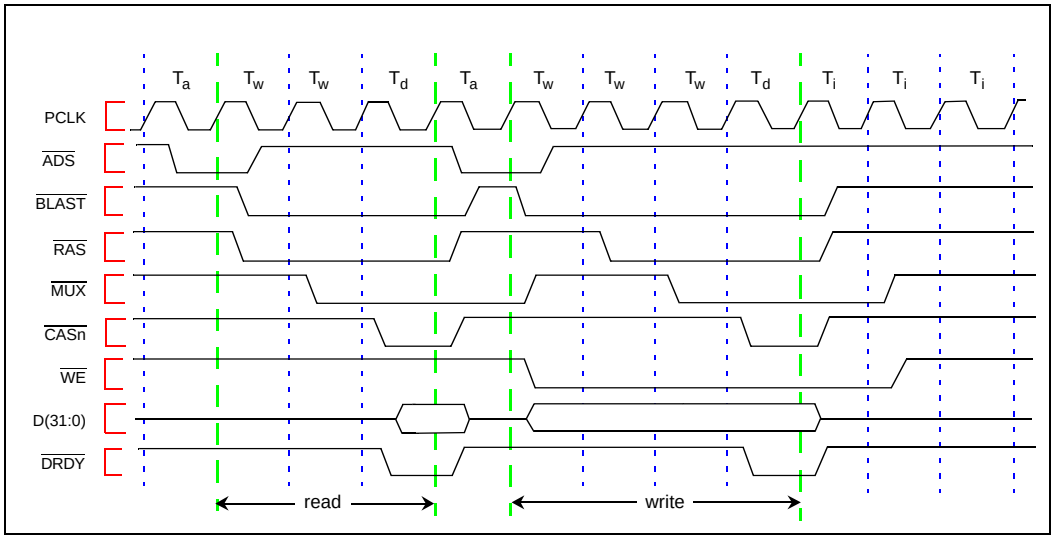


Figure 3. Single Word Read and Write Cycles

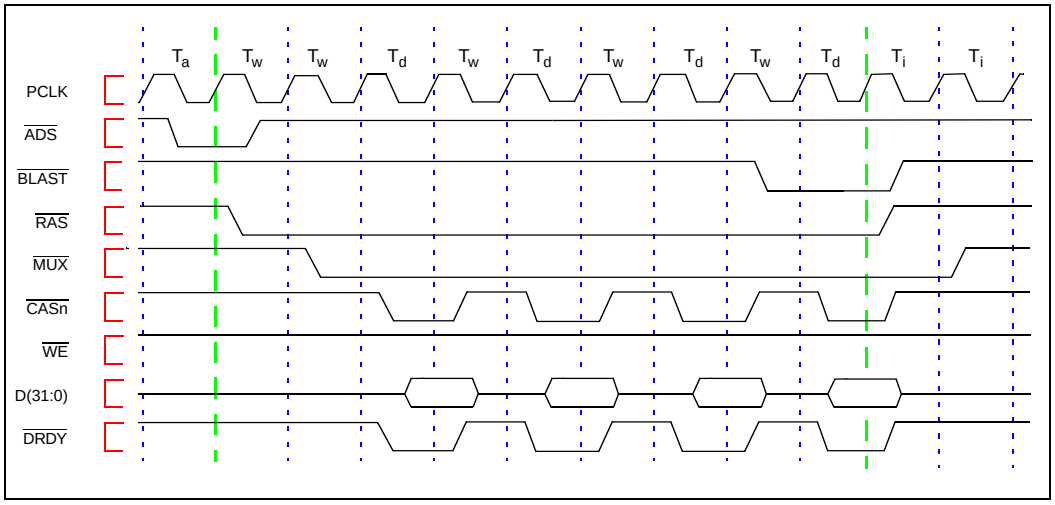


Figure 4. Quad Word Read Cycle

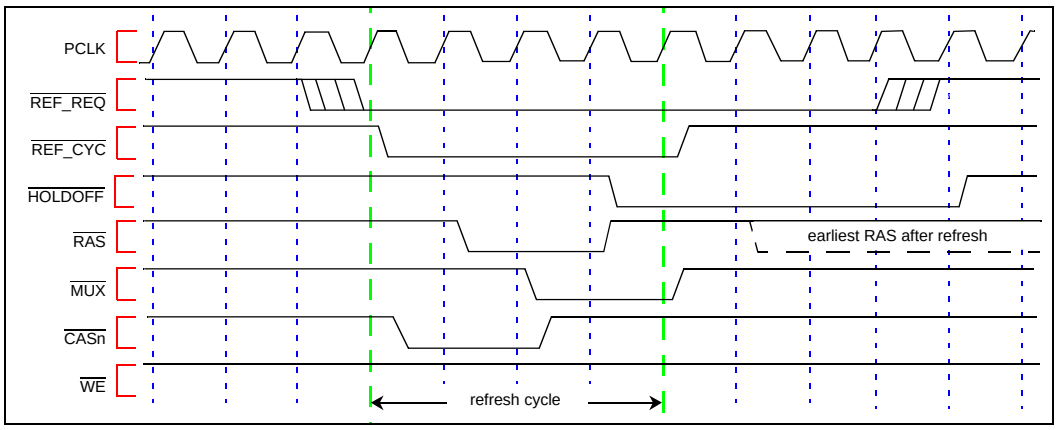


Figure 5. $\overline{\text{CAS}}$-Before-$\overline{\text{RAS}}$ Refresh Cycle

4.0 STATE MACHINE DESCRIPTIONS

This section details the operation of each state machine that this design uses. All machines are "Mealy" machines, in which the current output is a function only of current state. These machines can be easily implemented in the conventional D flip-flop macrocells in common PLDs.

Most state machines in this design have only two states. The machine transitions from one state to another upon satisfying a Boolean expression; otherwise, it remains in the current state. These expressions consist of:

- **assertion** conditions — the machine transitions from the deasserted (off) state to the asserted (on) state, and
- **deassertion** conditions — the machine transitions from the asserted (on) state to the deasserted (off) state

To illustrate these conditions, several timing diagrams are shown included in this section. Each equation is tagged with a number in square brackets "`[]`". Use these numbers to identify the corresponding points in the timing diagrams.

All PLD equations are written in ABEL. APPENDIX A, PLD EQUATIONS, contains a listing of the PLD equations file. The state machine transitions described here follow the ABEL conventions for logic operators.

- ! represents NOT, bit-wise negation
- & represents AND
- # represents OR

All signals appearing in logic equations are assumed to be active high. Many of the PLD outputs are active low; when these appear on timing diagrams, the signal names are denoted with an overline (such as $\overline{\text{RAS}}$).

4.1 RAS State Machine

The RAS state machine operates in one of two modes: one for CPU cycles and another for refresh cycles. Two other state machine inputs are used:

- REF_REQ state machine — indicates that a refresh request is pending
- REF_CYC state machine — indicates that a refresh cycle is actually underway

The RAS state machine is responsible for recognizing the start of processor cycles. When the DRAM controller is idle (and the RAS precharge time has been met), $\overline{\text{RAS}}$ is asserted when $\overline{\text{ADS}}$ is asserted and the upper address bits indicate a DRAM address select:

```
(ADS & DRAM & !MUX & !REF_REQ)[1]
```

The `!MUX` term keeps $\overline{\text{RAS}}$ from being asserted until $\overline{\text{MUX}}$ is deasserted, which provides the minimum 2 PCLK $\overline{\text{RAS}}$ precharge time.

The term `!REF_REQ` keeps a processor cycle from being recognized when a refresh request is pending, which gives priority to refresh cycles. However, if the DRAM controller is running a refresh cycle when this occurs, $\overline{\text{ADS}}$ is no

longer present when the controller needs to start a processor cycle. To this end, the RAS state machine relies upon another state machine, CPU_CYC, to "remember" that a processor cycle has been requested but not yet serviced. To start processor cycles when this occurs, another assertion condition is needed:

```
(CPU_CYC & !REF_CYC)[2]
```

The `!REF_CYC` term prevents false triggering of the RAS state machine during refresh cycles.

$\overline{RAS}$ deasserts after the last data is transferred, which is denoted by the processor's $\overline{BLAST}$ and the DRAM controller's DRDY being both asserted. DRDY is only generated during CPU cycles, so no conditioning with REF_CYC is needed.

```
(BLAST & DRDY) [3]
```

During refresh, the CAS state machines start the cycle and the RAS machine operates as a slave. For a $\overline{CAS}$-before-$\overline{RAS}$ refresh cycle, $\overline{RAS}$ is asserted one clock after $\overline{CAS}$ is asserted and $\overline{RAS}$ is deasserted one clock after $\overline{CAS}$ is deasserted. Since all four $\overline{CAS}$'s are asserted during refresh, only one (in this case $\overline{CAS0}$) is needed to trigger the RAS state machine. The assertion condition during refresh cycles is:

```
(CAS0 & REF_CYC) [4]
```

Deassertion is:

```
(!CAS0 & REF_CYC) [5]
```

At this point the two operating modes of the RAS machine are combined and a single set of assertion/deassertion equations can be written. Assertion conditions are:

```
  (ADS & DRAM & !MUX & !REF_REQ) [1]
# (CPU_CYC & !REF_CYC) [2]
# (CAS0 & REF_CYC) [4]
```

Deassertion conditions are:

```
  (BLAST & DRDY) [3]
# (!CAS0 & REF_CYC) [5]
```

The following figures illustrate these conditions superimposed on timing diagrams. Figure 7 shows the added wait state needed for RAS precharge when two CPU cycles occur back-to-back. Figure 8 shows what happens when a refresh cycle preempts a CPU cycle. In this case four additional wait states are inserted. Figure 9 shows refresh cycles.

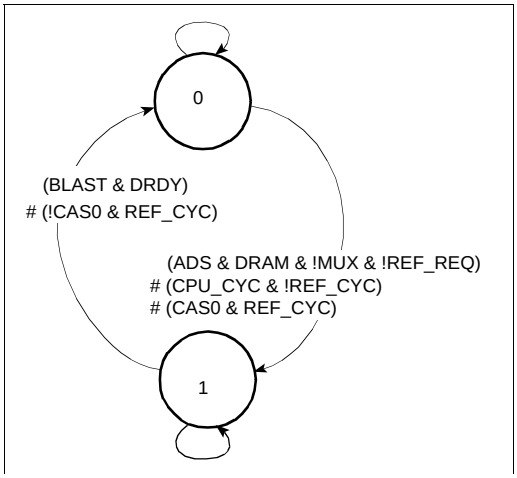


Figure 6. RAS State Machine

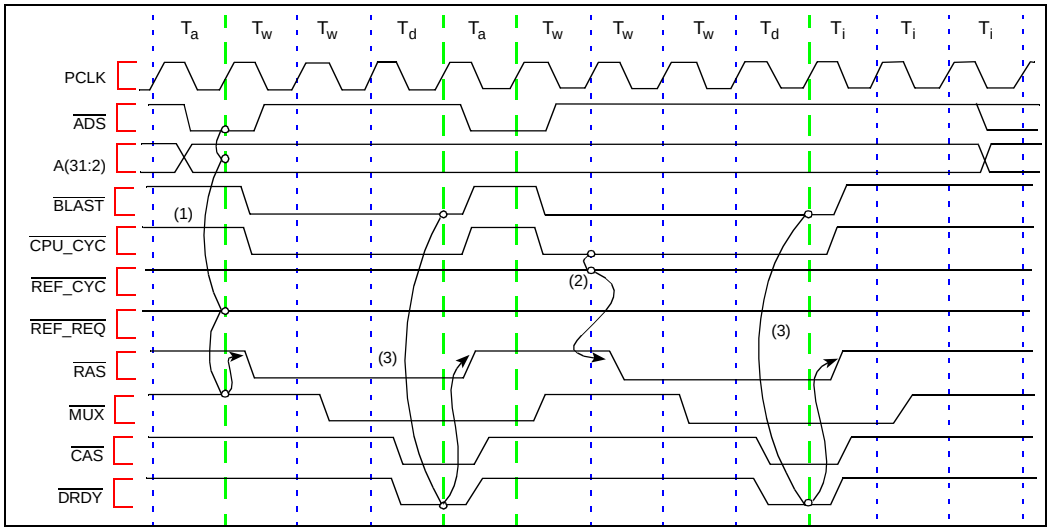


Figure 7. RAS State Machine Transitions - Back-to-Back Processor Cycles

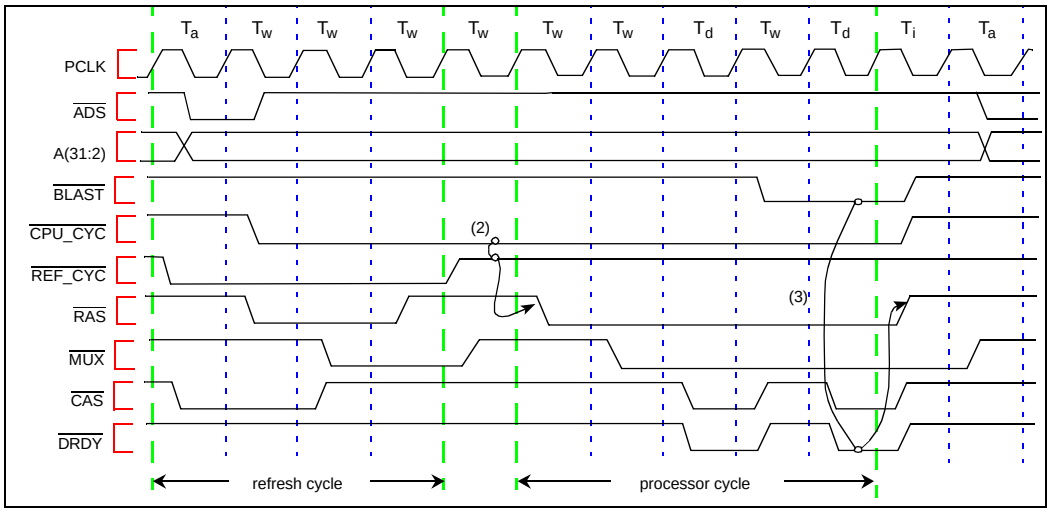


Figure 8. RAS State Machine Transitions - Processor Cycle Delayed By Refresh

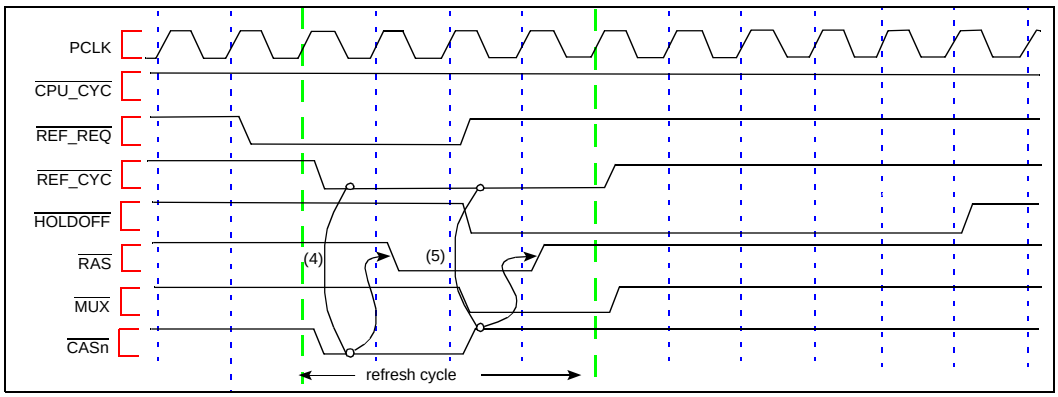


Figure 9. RAS State Machine Transitions - Refresh Cycles

4.2 MUX State Machine

The MUX state machine delays $\overline{RAS}$ by one clock. This switches the address multiplexer from "row" to "column" in time for the column address to stabilize before $\overline{CAS}$ asserts. It also provides an additional timing signal for use in other state machines; e.g., in the CAS state machine, $\overline{MUX}$ helps provide the additional delay clock between $\overline{RAS}$ and the assertion of $\overline{CAS}$. The assertion condition for $\overline{MUX}$ is:

```
(RAS)
```

The deassertion condition is:

```
(!RAS)
```

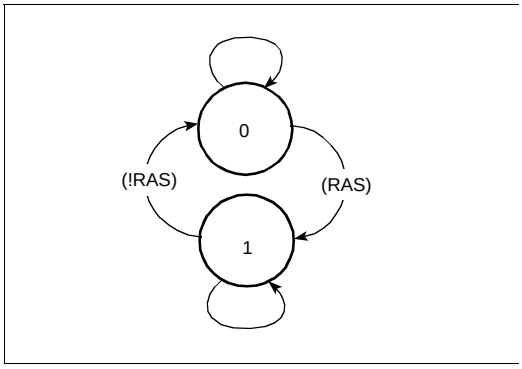


Figure 10. MUX State Machine

4.3 CAS State Machines

Four nearly-identical CAS state machines (CAS0, CAS1, CAS2, and CAS3) provide the column address strobes for the individual bytes in the DRAM array. They differ only in which byte enable signal is used to qualify their output.

Like the RAS state machine, the CAS state machines operate in one of two modes: CPU cycles and refresh cycles. During CPU cycles, the CAS state machines operate as a slave to the RAS machine, asserting $\overline{CAS}$ two clocks after $\overline{RAS}$. The two-clock delay is generated by waiting until both $\overline{RAS}$ and $\overline{MUX}$ are asserted. Thus the assertion condition for a CPU cycle is:

```
(RAS & MUX & BEn & !REF_CYC) [6]
```

where $\overline{BEn}$ stands for one of the four byte enables: $\overline{BE0}$, $\overline{BE1}$, $\overline{BE2}$, or $\overline{BE3}$. Since $\overline{CAS}$ is only asserted for a single clock, $\overline{CAS}$ is deasserted on the next clock edge. The deassertion condition is degenerate:

```
(!REF_CYC) [7]
```

When it is time to do a refresh, the CAS machines begin the cycle, the RAS machine acts as a slave. The refresh arbitration is done in a separate state machine: REF_REQ. Refresh cycles begin when a refresh request is requested and the DRAM controller is not running a CPU cycle. This occurs during the address state (T_a) and idle bus clocks (T_i). The assertion condition for $\overline{CAS}$ during refresh is:

```
(REF_REQ & !CPU_CYC) [8]
```

The deassertion condition is (see **(4)** in Figure 13):

```
(RAS & REF_CYC) [9]
```

$\overline{CAS}$ is deasserted one clock after $\overline{RAS}$ goes low.

The CPU refresh modes can now be combined into a single set to generate the state diagram shown in Figure 11. Assertion conditions are:

```
(RAS & MUX & BEn & !REF_CYC) [6]
# (REF_REQ & !CPU_CYC) [8]
```

Deassertion:

```
(!REF_CYC) [7]
# (RAS & REF_CYC) [9]
```

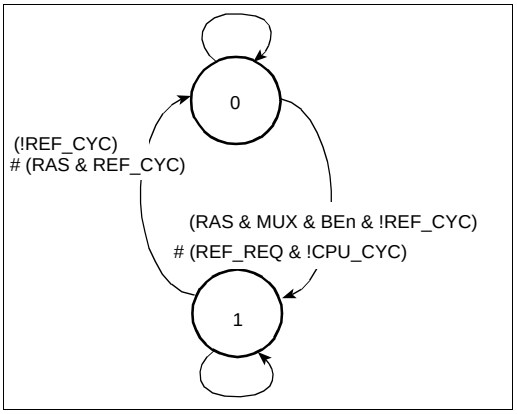


Figure 11. CAS State Machines

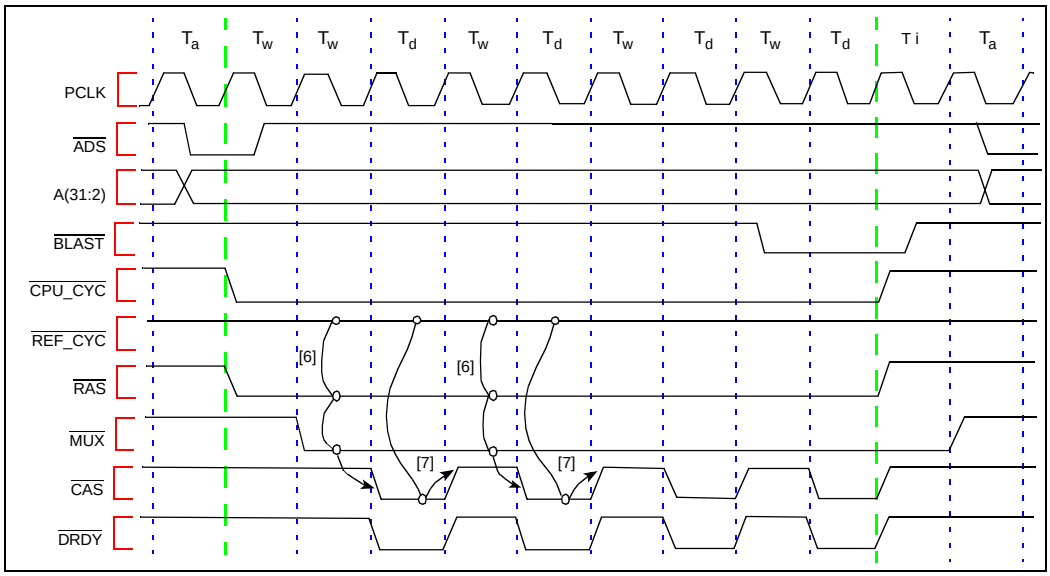


Figure 12. CAS State Machine Transitions - Processor Cycle

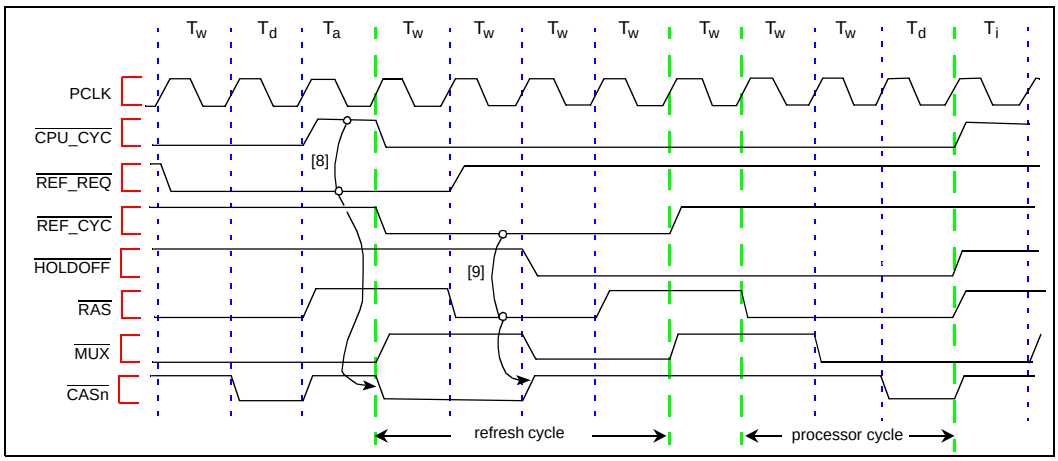


Figure 13. CAS State Machine Transitions - Refresh Cycles

4.4 DRDY State Machine

The DRDY state machine generates $\overline{\text{READY}}$ for the processor. $\overline{\text{DRDY}}$ is asserted simultaneously with $\overline{\text{CAS}}$ during processor cycles. Thus, the DRDY state machine resembles a $\overline{\text{CAS}}$ machine, except for: $\overline{\text{DRDY}}$ is not conditioned with byte enables, and it is not asserted during refresh. Assertion occurs when $\overline{\text{RAS}}$ and $\overline{\text{MUX}}$ are both active during non-refresh cycles:

```
(RAS & MUX & !REF_CYC) [10]
```

Deassertion is one clock later:

```
(!REF_CYC) [11]
```

To provide for DRDY deassertion at power-up, and additional term is added:

```
RESET
```

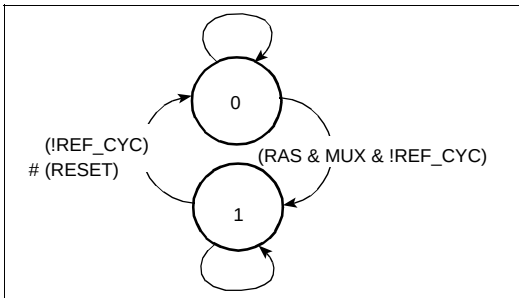


Figure 14. DRDY State Machine

4.5 CPU_CYC State Machine

This machine tracks processor requests to DRAM. CPU_CYC goes active after the address state (T_a) of a CPU cycle and goes inactive after the last data state (T_d). CPU_CYC is inactive during T_a and all idle bus clocks (T_i). Several state machines use CPU_CYC and its "sister", REF_CYC, to control their operation.

CPU_CYC is asserted when $\overline{\text{ADS}}$ is active and the upper three address bits match the DRAM region:

```
(ADS & DRAM) [12]
```

It is deasserted when $\overline{\text{DRDY}}$ is returned in the last data transfer or when $\overline{\text{RESET}}$ is asserted:

```
(BLAST & DRDY) [13]
```

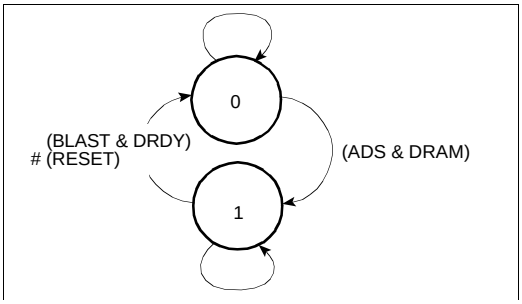


Figure 15. CPU_CYC State Machine

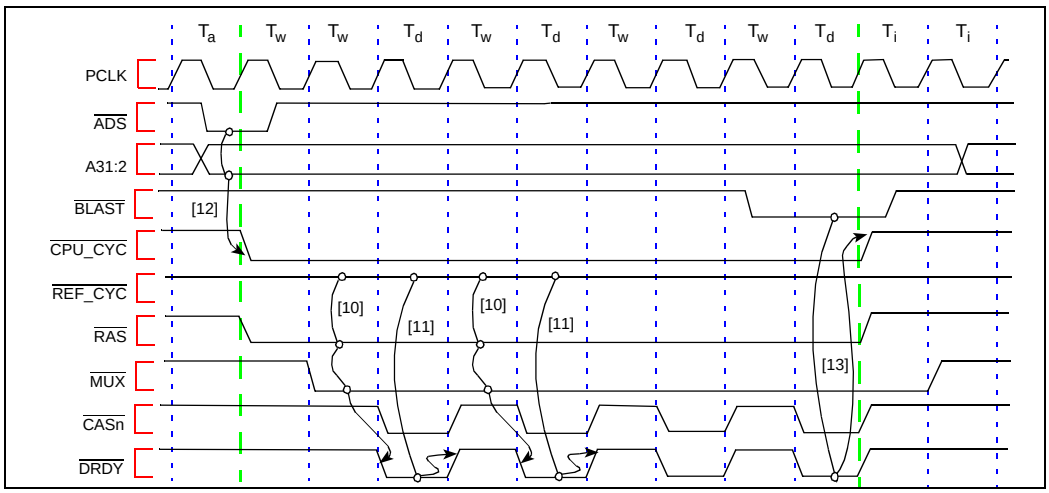


Figure 16. CPU_CYC and DRDY State Machine Transitions

4.6 REF_CYC State Machine

The REF_CYC State Machine indicates when the DRAM controller is performing a refresh cycle. Refresh cycles are requested by another state machine, the REF_REQ state machine, every 15 µs. Refresh cycles may begin during any idle bus clock (T_i) or during the address state (T_a) that starts a processor bus cycle. If a refresh request occurs during a processor cycle, $\overline{\text{REF\_CYC}}$ is not asserted until the next T_i or T_a state.

An idle bus and the address state look the same to the REF_CYC state machine. Both are signified by having $\overline{\text{CPU\_CYC}}$ deasserted.

The assertion conditions for $\overline{\text{REF\_CYC}}$ are:

```
(!CPU_CYCLE & REF_REQ) [14]
```

REF_CYC is not deasserted until $\overline{\text{RAS}}$ has returned high after the refresh cycle. Since $\overline{\text{RAS}}$ is high during the first clock of a refresh cycle, the deassertion condition includes $\overline{\text{MUX}}$ to keep $\overline{\text{REF\_CYC}}$ from going prematurely inactive. The deassertion conditions are:

```
(!RAS & MUX) [15]
```

Figure 18 shows the worst case delay of a processor cycle due to a DRAM refresh. Figure 19 shows the worst case delay of a refresh cycle due to a processor cycle.

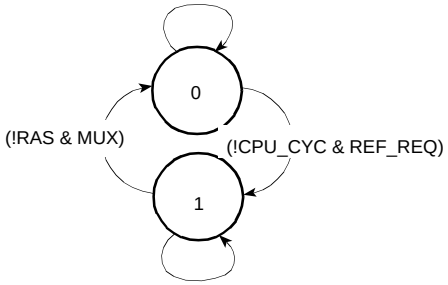


Figure 17. REF_CYC State Machine

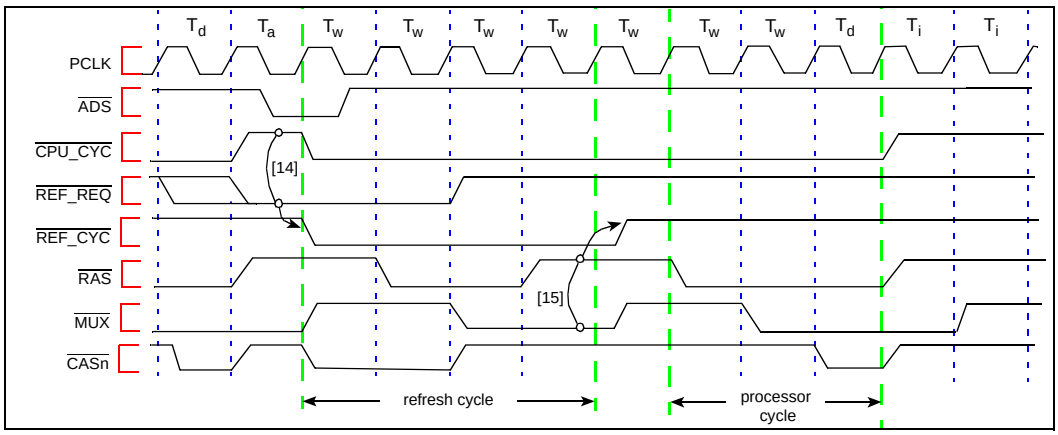


Figure 18. REF_CYC Transitions Where Refresh Cycle Delays Processor Cycle

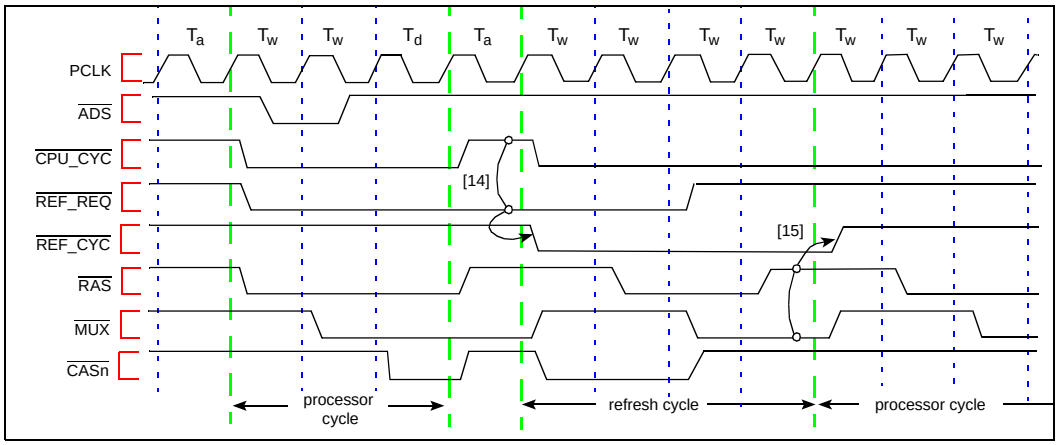


Figure 19. REF_CYC Transitions Where Processor Cycle Delays Refresh

4.7 REF_REQ State Machine

The REF_REQ state machine is an edge-detected version of the refresh counter's most significant bit. It is also synchronized to PCLK. $\overline{\text{REF\_REQ}}$ is asserted as soon as the counter ($\overline{\text{R4}}$) is set and $\overline{\text{HOLDOFF}}$ is deasserted. $\overline{\text{HOLDOFF}}$, as shown below, prevents $\overline{\text{REF\_REQ}}$ from being reasserted until $\overline{\text{R4}}$ changes state. $\overline{\text{REF\_REQ}}$ is deasserted as soon as a refresh cycle is actually started. Assertion conditions are:

```
(R4 & !HOLDOFF) [16]
```

Deassertion conditions are:

```
(REF_CYC) [17]
```

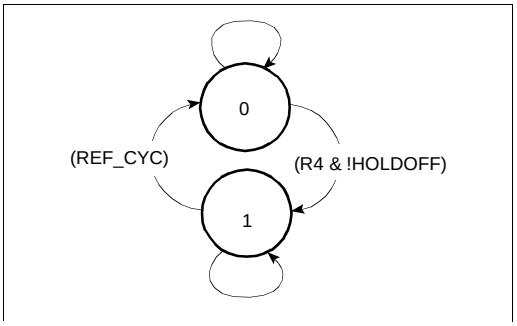


Figure 20. REF_REQ State Machine

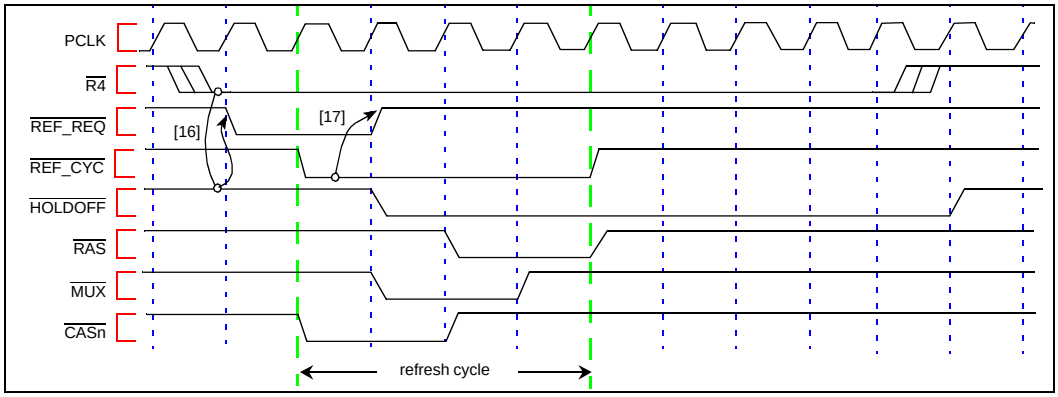


Figure 21. REF_REQ State Machine Transitions

4.8 HOLDOFF State Machine

As noted in Section 4.7, the HOLDOFF state machine provides an edge-detect function for the $\overline{\text{REF\_REQ}}$ input. $\overline{\text{HOLDOFF}}$ is asserted during a refresh cycle at the point when RAS is asserted. $\overline{\text{HOLDOFF}}$ assertion prevents another refresh cycle from being requested by the REF_REQ state machine until the refresh timer $\overline{\text{R4}}$ is deasserted. Assertion conditions are:

```
(REF_CYC) [18]
```

Deassertion conditions are:

```
(!R4) [19]
```

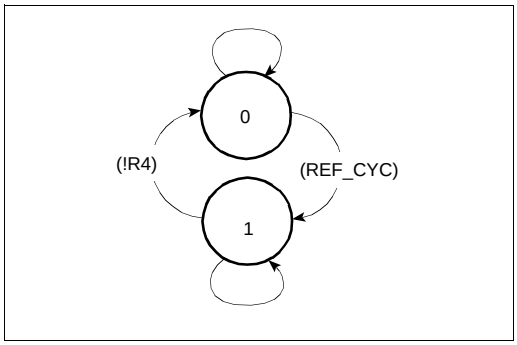


Figure 22. HOLDOFF State Machine

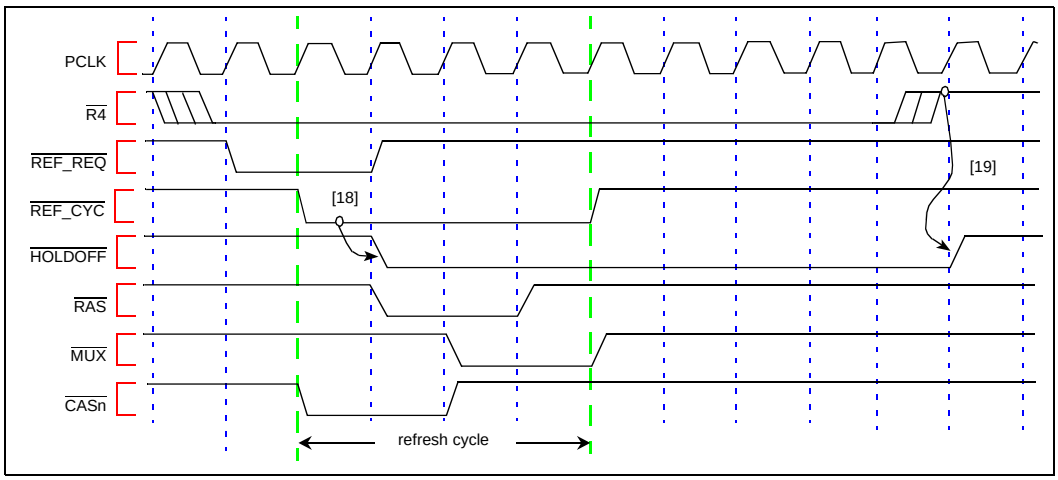


Figure 23. HOLDOFF State Machine Transitions

4.9 Burst Address (DRA0, DRA1) State Machine

The burst address state machine, one of the more complex machines in the DRAM controller, is responsible for generating the two low-order address bits for the DRAM array. It functions as a multiplexer during the first part of the cycle to provide processor address bits A12 and A11 as part of the row address. During the data cycles it provides a synthesized version of processor bits A3 and A2. Generating A3 and A2 in the state machine reduces the delay in the critical path from column address to data out.

The address generation requires a four-state machine to track the bus cycle, plus some combinatorial logic to provide the multiplexer function. This combinatorial logic adds to the normal output valid delay of the state machine, but is quicker than an external multiplexer. The four states are:

1. Multiplexer. In this state pass A12 and A11 or A3 and A2 (based on the $\overline{\text{MUX}}$ signal) to the outputs DRA1 and DRA0.

2. State “1”. Drive DRA1 = 0, DRA0 = 1.

3. State “2”. Drive DRA1 = 1, DRA0 = 0.

4. State “3”: Drive DRA1 = 1, DRA0 = 1.

The “idle” state is the multiplexer state. Transitions between states must track the type of bus cycle run by the processor to produce the proper address bits.

Single accesses (bytes, shorts, and words) stay in the multiplexer state - no burst address generation is needed.

Double word accesses may be to word addresses 0 - 1 or 2 - 3. Double word accesses that start at word address 0 start in the multiplex state (to allow address 0 to flow through), move to state “1” to provide word address 1, then return to the multiplexer state (idle). Accesses that start at word address 2 jump from the multiplexer state to state “3” before returning to idle.

Triple word accesses, by definition, always start with word address 0. They transition from multiplexer state, state “1”, state “2”, and then back to idle.

Quad word accesses always begin with word address 0 and cycle though all four states in order.

The machine stays in the multiplexer state until a multi-word bus cycle is detected ($\overline{\text{DRDY}}$ asserted while $\overline{\text{BLAST}}$ is deasserted). Based on the state of A3 and A2 during the first access, it advances to either state “1” or state “3”. While in state “1” through “3” it returns to the multiplexer state when the bus cycle ends ($\overline{\text{DRDY}}$ and $\overline{\text{BLAST}}$ asserted).

The state machine is defined in a “present state - next state” syntax as follows:

```
STATE_MUX:
  !A3 & !A2 & DRDY & !BLAST-> STATE_1
  A3 & !A2 & DRDY & !BLAST-> STATE_3
  else      -> STATE_MUX

STATE_1:
  DRDY & !BLAST-> STATE_2
       DRDY & BLAST-> STATE_MUX
  else      -> STATE_1

STATE_2:
  DRDY & !BLAST-> STATE_3
  DRDY & BLAST-> STATE_MUX
  else      -> STATE_2

STATE_3:
  DRDY & BLAST-> STATE_MUX
  else      -> STATE_3
```

While in the MUX state, DRA1 and DRA0 are either the row address (A12 and A11) or the column address (A3 and A2) based on the $\overline{MUX}$ signal. When in states 1 through 3, DRA1 and DRA0 are the binary value of those states, as follows:

```
DRA1  = (STATE_MUX & !MUX & A12) #
(STATE_MUX & MUX & A3) #
          STATE_2 # STATE_3

DRA0  = (STATE_MUX & !MUX & A11) #
(STATE_MUX & MUX & A2) #
         STATE_1 # STATE_3
```

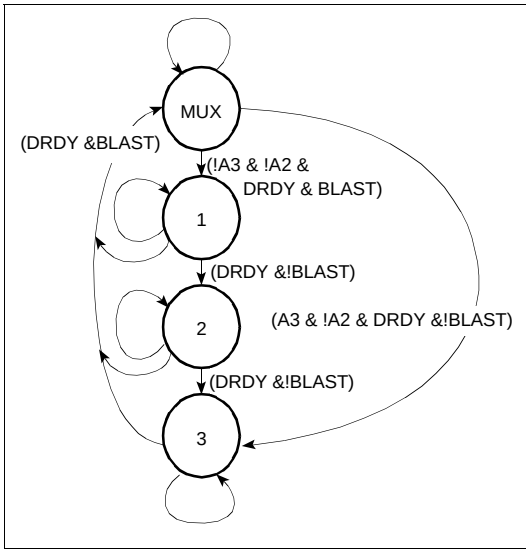


Figure 24. DRA0, DRA1 State Machine

4.10 Refresh Divider State Machine

The 1.8432 MHz baud rate clock is divided by 28 to produce a $\overline{REF\_REQ}$ with a 15 µs period. This is performed by a five-bit binary up counter. $\overline{R4}$ is the most significant bit of the counter. See APPENDIX A for details. Use of a different refresh clock frequency would require adjusting the division ratio to maintain the 15 µs period.

This state machine may be replaced with an external refresh timer. In this case, an asynchronous square wave with a period of 15 µs may be substituted for the $\overline{R4}$ input to the REF_REQ and HOLDOFF state machines.

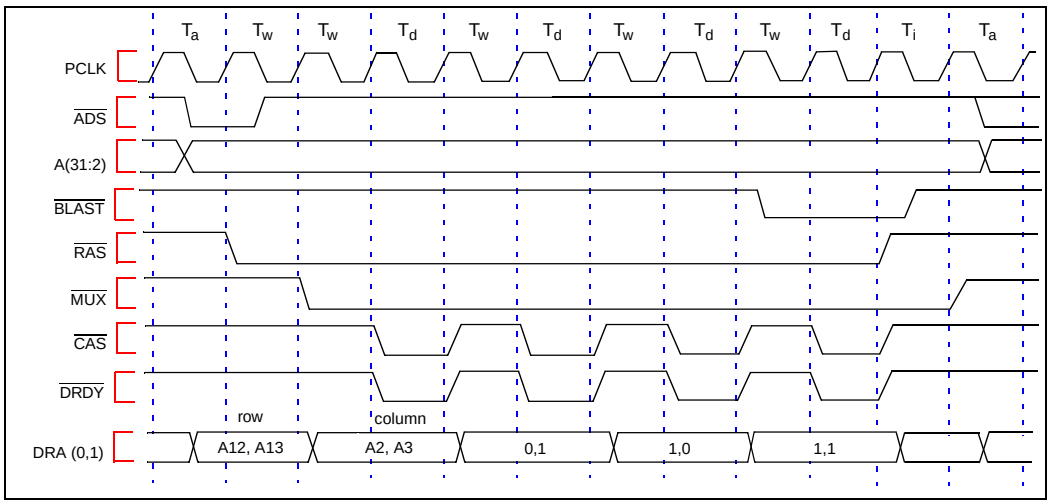


Figure 25. DRA0, 1 Burst Address State Transitions

4.11 Read Strobe State Machine

This state machine produces a general-purpose read strobe ($\overline{RD}$) for FLASH memory, SRAM, and other I/O peripherals. $\overline{RD}$ is asserted at the start of all read cycles without any address qualification. $\overline{RD}$ is deasserted when $\overline{BLAST}$ is issued by the processor. It is assumed that the internal i960 Cx processor's wait state generator is used to control all such bus cycles, since no $\overline{READY}$ is generated by this logic.

$\overline{RD}$ assertion conditions are:

```
(ADS & !W/R) [20]
```

Deassertion conditions are:

```
(BLAST) [21]
```

Additionally, $\overline{RD}$ is deasserted upon power-up by the $\overline{RESET}$ signal.

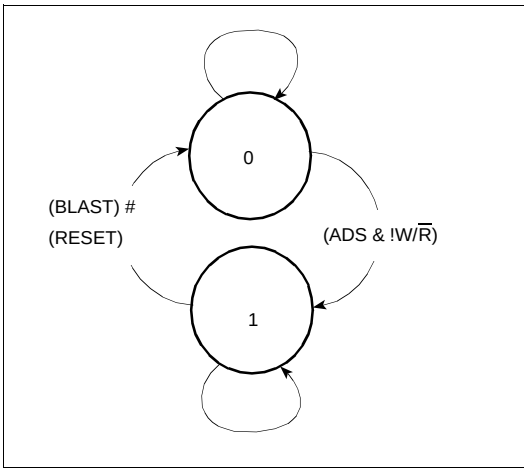


Figure 26. RD (Read Strobe) State Machine

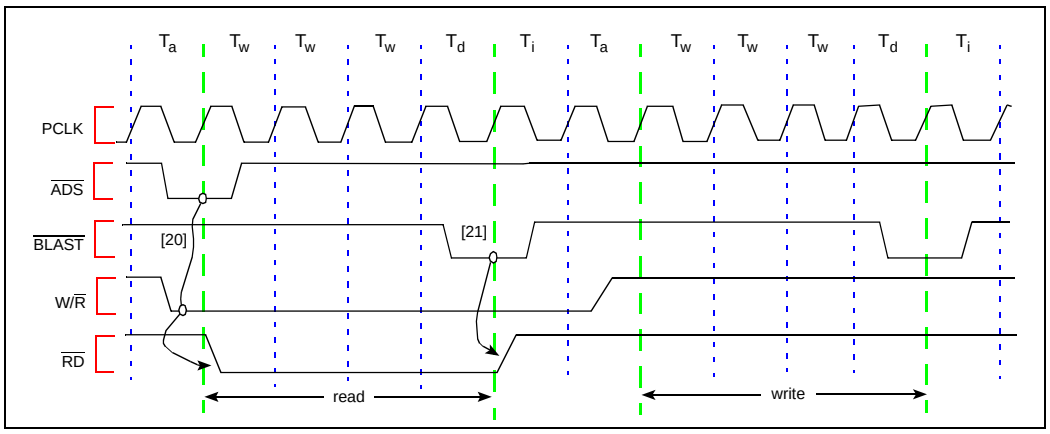


Figure 27. RD State Machine Transitions

5.0 COMBINATORIAL OUTPUTS

The few remaining general-purpose controls can be generated from processor signals with simple combinatorial logic. Among these are the SRAM and I/O write strobe ($\overline{WR}$), the DRAM write enable ($\overline{WE}$), and the transceiver output enable ($\overline{245OE}$).

5.1 I/O Write Strobe

For regions controlled by the internal wait state generator with more than one wait state, the processor's $\overline{WAIT}$ signal can be used to generate a suitable write strobe for I/O peripherals (see Figure 28):

```
WR# = (W/R & WAIT)
```

5.2 DRAM Write Enable

An inverted version of the processor's W/$\overline{R}$ signal can be used for this purpose with one modification. Most DRAMs require that $\overline{WE}$ is high (deasserted) during refresh cycles. Since refresh cycles may occur while the processor is waiting to perform a DRAM access, the state of the W/$\overline{R}$ signal cannot be guaranteed. Accordingly W/$\overline{R}$ needs to be gated with $\overline{REF\_CYC}$ as follows:

```
WE# = (W/R & !REF_CYC)
```

5.3 Transceiver Control

In the tested design, an I/O transceiver isolates the 8-bit I/O bus from the processor and DRAM array. The 74F245 device requires two controls: a direction select and an output enable. To eliminate bus contention, the direction should be changed only when the output enable is deasserted.

Direction select is taken directly from the processor's DT/$\overline{R}$ control. Output enable is generated by a combination of the processor's $\overline{DEN}$ signal and an address decode of the upper address bits. This defines a section of memory that activates the transceiver, keeping it from interfering with DRAM cycles.

```
OE245# = (DEN & IO_DECODE)
```

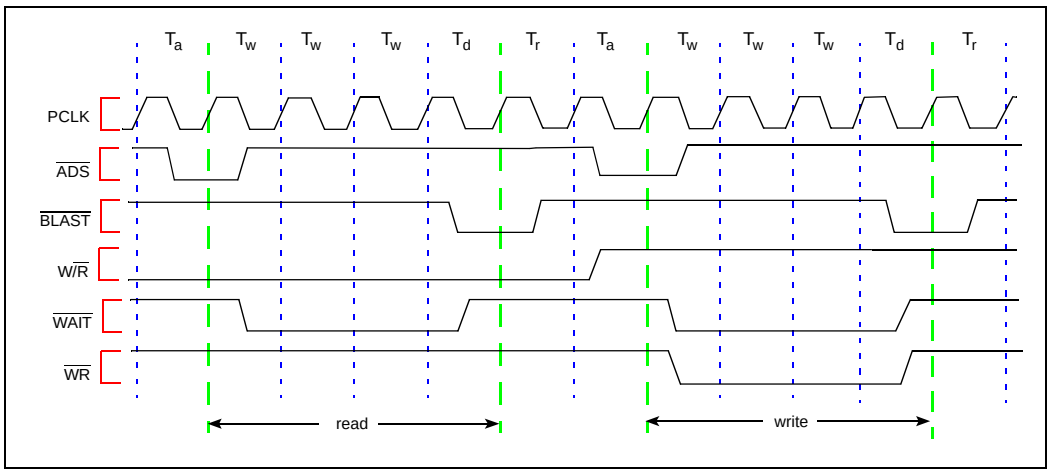


Figure 28. W/$\overline{R}$ Strobe Generation

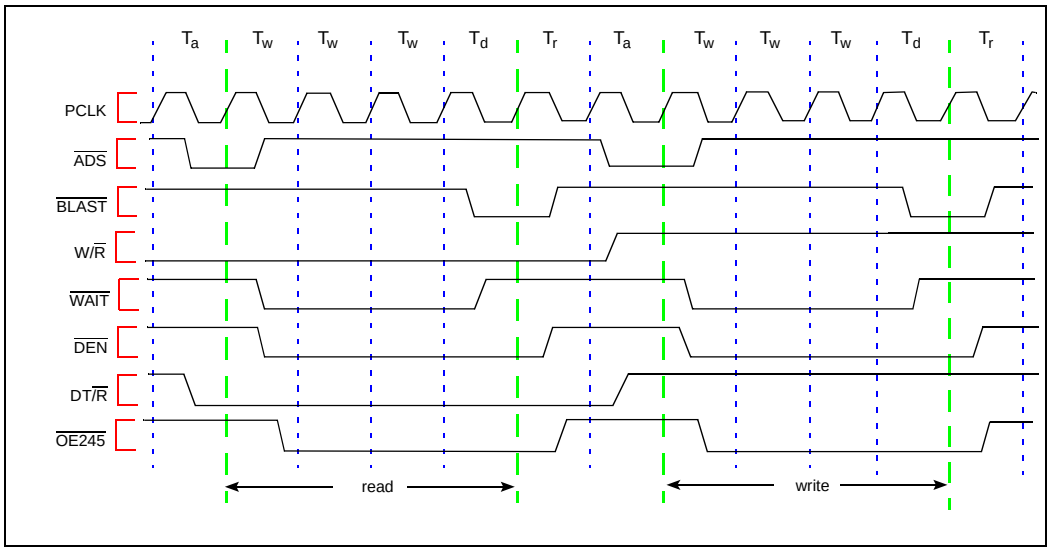


Figure 29. Transceiver $\overline{OE}$ Generation

6.0 CONCLUSION

In conclusion, this application note describes a simple DRAM controller for use with i960 CA/CF 16/25 MHz processors. This DRAM controller was built and tested for validation purposes. The PLD equations used to build and test the prototype design were created in ABEL. All timing analysis was verified with Timing Designer. Schematics were created with OrCAD. The timing analysis, schematics and PLD files are available through Intel's America's Application Support BBS.

7.0 RELATED INFORMATION

This application note is one of four that are related to DRAM controllers for the i960 processors. The following table shows the documents and order numbers:

Document Name	App. Note #	Order #
DRAM Controller for the 40 MHz i960® CF Microprocessors	AP-706	272655
DRAM Controller for the i960® Jx Microprocessors	AP-712	272674
DRAM Controller for 33 MHz i960® CA/CF Microprocessors	AP-703	272627

To receive these documents or any other available Intel literature, contact:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect IL 60056-7641
1-800-879-4683

To receive files that contain the timing analysis, schematics and PLD equations for this and the other DRAM controller application notes, contact:

Intel Corporation
America's Application Support BBS
916-356-3600

APPENDIX A
PLD EQUATIONS

Table A-1 contains the PLD equations which were used to build and test the prototype design. Table A-1 defines signal and product term allocation. The PLD equations were created in ABEL as a device-independent design. Using the ABEL* software, a PDS file was created and subsequently imported into PLDSHELL* software. PLDSHELL was used to fit the design into an Altera EPX780 FLEXlogic* PLD. PLDSHELL was also utilized to create the JEDEC file, and to simulate the design.

In addition, this appendix contains a table listing the number of product terms used by each macrocell.

Table A-1. 33 MHz Simple DRAM Controller PLD Equations (Sheet 1 of 9)

```
module  DRAM
title   '960CA DRAM Controller (21121)
Rick Schue  Intel Corp.  01/25/95'

    H,L,Ck,X = 1, 0, .C., .X.;

" Device independent design

" input pin assignments

    PCLK            pin;                " 25.000 MHz clock for state variables
    REFCLK          pin;                " 1.8432 MHz clock for refresh timer
    !ADS            pin;                " CPU ADS# control
    !BLAST          pin;                " CPU BLAST# control
    !RESET          pin;                " system reset (active low)
    A31             pin;                " CPU address lines
    A30             pin;                "
    A29             pin;                "
    A12             pin;                " address lines used to generate
    A11             pin;                " DRA1,0
    A3              pin;                "
    A2              pin;                "
    !BE3            pin;                " CPU byte enables
    !BE2            pin;                "
    !BE1            pin;                "
    !BE0            pin;                "
    !READ           pin;                " CPU W!R# control
    !WAIT           pin;                " CPU WAIT# control
    !DEN            pin;                " CPU DEN# control

" output pin assignments
    !RAS            pin  ISTYPE 'reg';  " DRAM RAS
    !CAS3           pin  ISTYPE 'reg';  " DRAM CAS
    !CAS2           pin  ISTYPE 'reg';  " DRAM CAS
    !CAS1           pin  ISTYPE 'reg';  " DRAM CAS
    !CAS0           pin  ISTYPE 'reg';  " DRAM CAS
    DRA1            pin  ISTYPE 'com';  " DRAM address outputs (combinatorial)
    DRA0            pin  ISTYPE 'com';  " DRAM address outputs (combinatorial)
    !DRDY           pin  ISTYPE 'reg';  " DRAM ready
    !MUX            pin  ISTYPE 'reg';  " row/column control: 1=row 0=col
    !WE             pin  ISTYPE 'com';  " DRAM WE#
```

Table A-1. 33 MHz Simple DRAM Controller PLD Equations (Sheet 2 of 9)

```
    !RD             pin  ISTYPE 'reg';  " I/O READ
    !WR             pin  ISTYPE 'com';  " I/O WRITE (combinatorial)
    !OE245          pin  ISTYPE 'com';  " 74F245 transceiver control for I!O bus

" internal state variables - these do not have to be assigned to pins unless
" one wishes to use them for debug.
    !CPU_CYC        node ISTYPE 'reg';  " CPU cycle pending
    !REF_REQ        node ISTYPE 'reg';  " refresh request (active low)
    !REF_CYC        node ISTYPE 'reg';  " Refresh cycle pending
    !HOLDOFF        node ISTYPE 'reg';  " Refresh holdoff
    S1,S0           node ISTYPE 'reg';  " DRA1,0 state bits
    R0,R1,R2,R3,R4  node ISTYPE 'reg';  " Refresh timer (divide by 28)

" region decode:
"  to conserve pins in this simple example, addresses are decoded into only
"  eight regions.

    MEMADDR = [A31,A30,A29,X, X,X,X,X, X,X,X,X, X,X,X,X,
                      X,X,X,X, X,X,X,X, X,X,X,X, X,X,X,X];

" state machine vector definitions
    RAS_MACHINE     = [RAS];
    MUX_MACHINE     = [MUX];
    CAS0_MACHINE    = [CAS0];
    CAS1_MACHINE    = [CAS1];
    CAS2_MACHINE    = [CAS2];
    CAS3_MACHINE    = [CAS3];
    DRDY_MACHINE    = [DRDY];
    RD_MACHINE      = [RD];
    CPU_CYC_MACHINE = [CPU_CYC];
    REF_REQ_MACHINE = [REF_REQ];
    REF_CYC_MACHINE = [REF_CYC];
    HOLDOFF_MACHINE = [HOLDOFF];
    BURST_MACHINE   = [S1,S0];
    REFRESH         = [R4,R3,R2,R1,R0];

" literals
    IDLE      = ^b0;
    ACTIVE    = ^b1;
    STATE_MUX = ^b00;
    STATE_1   = ^b01;
    STATE_2   = ^b10;
    STATE_3   = ^b11;

" modify the following address assignments to relocate IO and/or DRAM into
" other memory regions.  In this example:
"          DRAM  0xc0000000 to 0xdfffffff
"          I/O   0x20000000 to 0x3fffffff (UART)
"                0x60000000 to 0x7fffffff (parallel port)
"                0xe0000000 to 0xffffffff (FLASH boot ROM)

    DRAM  = (MEMADDR >= ^hC0000000) & (MEMADDR <= ^hDFFFFFFF);
    IO    = ((MEMADDR >= ^h20000000) & (MEMADDR <= ^h3FFFFFFF)) #
            ((MEMADDR >= ^h60000000) & (MEMADDR <= ^h7FFFFFFF)) #
            ((MEMADDR >= ^hE0000000) & (MEMADDR <= ^hFFFFFFFF));
```

Table A-1. 33 MHz Simple DRAM Controller PLD Equations (Sheet 3 of 9)

```
equations

" DRAM burst address multiplexer
"   state MUX    DRA1  DRA0
"    00    0     A12   A11      pass through ROW address
"    00    1      A3    A2      pass through COL address
"    01    x      0     1       burst address
"    10    x      1     0       burst address
"    11    x      1     1       burst address

    DRA1  = (!S1 & !S0 & !MUX & A12) #
            (!S1 & !S0 & MUX & A3) # (S1 & !S0) # (S1 & S0);
    DRA0  = (!S1 & !S0 & !MUX & A11) #
            (!S1 & !S0 & MUX & A2) # (!S1 & S0) # (S1 & S0);

" simple I/O control signals
    WE    = (!READ & !REF_CYC);
    WR    = (WAIT & !READ);
    OE245 = (DEN & IO);

" the following state machines are synchronous with the CPU
    RAS.clk         = PCLK;
    MUX.clk         = PCLK;
    CAS0.clk        = PCLK;
    CAS1.clk        = PCLK;
    CAS2.clk        = PCLK;
    CAS3.clk        = PCLK;
    DRDY.clk        = PCLK;
    RD.clk          = PCLK;
    CPU_CYC.clk     = PCLK;
    REF_REQ.clk     = PCLK;
    REF_CYC.clk     = PCLK;
    HOLDOFF.clk     = PCLK;
    [S1,S0].clk     = PCLK;

" the refresh divider runs off a second (asynchronous) clock
" if a higher frequency REFCLK is used, more bits may be required here
   [R4,R3,R2,R1,R0].clk = REFCLK;
```

Table A-1. 33 MHz Simple DRAM Controller PLD Equations (Sheet 4 of 9)

```
"-------------------------------------------------------------------------------

state_diagram BURST_MACHINE

" This state machine controls the DRAM A1,A0 multiplexer described
" above in the EQUATIONS section.

    state STATE_MUX:
        if      (!A3 & !A2 & DRDY & !BLAST)   then STATE_1;
        else if ( A3 & !A2 & DRDY & !BLAST)   then STATE_3;
        else    STATE_MUX;

    state STATE_1:
        if      (DRDY & !BLAST)   then STATE_2;
        else if (DRDY &  BLAST)   then STATE_MUX;
        else    STATE_1;

    state STATE_2:
        if      (DRDY & !BLAST)   then STATE_3;
        else if (DRDY &  BLAST)   then STATE_MUX;
        else    STATE_2;

    state STATE_3:
        if      (DRDY &  BLAST)   then STATE_MUX;
        else    STATE_3;

"-------------------------------------------------------------------------------

state_diagram CPU_CYC_MACHINE

" This machine tracks all DRAM cycles requested by the CPU

    state IDLE:
        if (ADS & DRAM) then ACTIVE;
        else IDLE;

    state ACTIVE:
        if (BLAST & DRDY # RESET) then IDLE;
        else ACTIVE;
```

Table A-1. 33 MHz Simple DRAM Controller PLD Equations (Sheet 5 of 9)

```
"-------------------------------------------------------------------------------

state_diagram MUX_MACHINE

" MUX_MACHINE is a delayed version of RAS, used to switch the external row/
" column multiplexer.

    state IDLE:
        if (RAS) then ACTIVE;
        else IDLE;

    state ACTIVE:
        if (!RAS) then IDLE;
        else ACTIVE;

"-------------------------------------------------------------------------------

state_diagram RAS_MACHINE

" This generates RAS for a DRAM bank.  A single RAS controls all bytes in
" that bank.  If additional banks are desired, extra RAS's will have to be
" generated with similar state machines.

    state IDLE:
        if (ADS & DRAM & !MUX & !REF_REQ) #
           (CPU_CYC & !REF_CYC) #
           (REF_CYC & CAS0) then ACTIVE;
        else IDLE;

    state ACTIVE:
        if (BLAST & DRDY) # (REF_CYC & !CAS0) then IDLE;
        else ACTIVE;

"-------------------------------------------------------------------------------

state_diagram CAS0_MACHINE

" The following machines generate individual CAS's for each byte of the
" DRAM array.  Their only differences are the byte enables that enable them

    state IDLE:
        if (RAS & BE0 & !REF_CYC & MUX) # (REF_REQ & !CPU_CYC) then ACTIVE;
        else IDLE;

    state ACTIVE:
        if (!REF_CYC) # (REF_CYC & RAS) then IDLE;
        else ACTIVE;
```

Table A-1. 33 MHz Simple DRAM Controller PLD Equations (Sheet 6 of 9)

```
"-------------------------------------------------------------------------------

state_diagram CAS1_MACHINE

    state IDLE:
        if (RAS & BE1 & !REF_CYC & MUX) # (REF_REQ & !CPU_CYC) then ACTIVE;
        else IDLE;

    state ACTIVE:
        if (!REF_CYC) # (REF_CYC & RAS) then IDLE;
        else ACTIVE;

"-------------------------------------------------------------------------------

state_diagram CAS2_MACHINE

    state IDLE:
        if (RAS & BE2 & !REF_CYC & MUX) # (REF_REQ & !CPU_CYC) then ACTIVE;
        else IDLE;

    state ACTIVE:
        if (!REF_CYC) # (REF_CYC & RAS) then IDLE;
        else ACTIVE;

"-------------------------------------------------------------------------------

state_diagram CAS3_MACHINE

    state IDLE:
        if (RAS & BE3 & !REF_CYC & MUX) # (REF_REQ & !CPU_CYC) then ACTIVE;
        else IDLE;

    state ACTIVE:
        if (!REF_CYC) # (REF_CYC & RAS) then IDLE;
        else ACTIVE;

"-------------------------------------------------------------------------------

state_diagram DRDY_MACHINE

" This machine generates READY for all DRAM accesses.  This output should be
" externally OR'ed with any other external ready type peripherals.

    state IDLE:
        if (RAS & !REF_CYC & MUX) then ACTIVE;
        else IDLE;

    state ACTIVE:
        if (!REF_CYC) # (RESET) then IDLE;
        else ACTIVE;
```

Table A-1. 33 MHz Simple DRAM Controller PLD Equations (Sheet 7 of 9)

```
"-------------------------------------------------------------------------------

state_diagram HOLDOFF_MACHINE

" This machine is used to edge detect the square wave refesh request.  See
" REF_REQ_MACHINE.

    state IDLE:
        if (REF_CYC) then ACTIVE;
        else IDLE;

    state ACTIVE:
        if (!R4) then IDLE;
        else ACTIVE;

"-------------------------------------------------------------------------------

state_diagram REF_REQ_MACHINE

" This machine is used to hold a pending refresh request.  R4 is a 65 KHz
" square wave derived from the refresh timer (or some other source).

    state IDLE:
        if (R4 & !HOLDOFF) then ACTIVE;
        else IDLE;

    state ACTIVE:
        if (REF_CYC) then IDLE;
        else ACTIVE;

"-------------------------------------------------------------------------------

state_diagram REF_CYC_MACHINE

" This machine indicates when the DRAM controller is actually performing
" a refresh cycle.

    state IDLE:
        if (!CPU_CYC * REF_REQ) then ACTIVE;
        else IDLE;

    state ACTIVE:
        if (!RAS & MUX) then IDLE;
        else ACTIVE;
```

Table A-1. 33 MHz Simple DRAM Controller PLD Equations (Sheet 8 of 9)

```
"-------------------------------------------------------------------------------

state_diagram RD_MACHINE

" This machine tracks all READ cycles issued by the CPU.  It assumes that
" the internal wait state generator is used, since it relies on BLAST to
" signify the end of the cycle.

    state IDLE:
        if (ADS & READ) then ACTIVE;
        else IDLE;

    state ACTIVE:
        if (BLAST) # (RESET) then IDLE;
        else ACTIVE;

"-------------------------------------------------------------------------------

state_diagram REFRESH

" This machine divides the 1.8432 MHz REFCLK by 28, producing a 65 KHz
" square wave to request refesh cycles.  If a higher frequency REFCLK is
" used, modify this machine accordingly.  If an external 65KHz source is
" available, this machine may be eliminated.  Convert node R4 into a pin
" and apply the external refresh signal to this pin

    state 0:     goto 1;
    state 1:     goto 2;
    state 2:     goto 3;
    state 3:     goto 4;
    state 4:     goto 5;
    state 5:     goto 6;
    state 6:     goto 7;
    state 7:     goto 8;
    state 8:     goto 9;
    state 9:     goto 10;
    state 10:    goto 11;
    state 11:    goto 12;
    state 12:    goto 13;
    state 13:    goto 14;
    state 14:    goto 15;
    state 15:    goto 16;
    state 16:    goto 17;
    state 17:    goto 18;
    state 18:    goto 19;
    state 19:    goto 20;
    state 20:    goto 21;
    state 21:    goto 22;
    state 22:    goto 23;
    state 23:    goto 24;
    state 24:    goto 25;
    state 25:    goto 26;
    state 26:    goto 27;
    state 27:    goto 0;
    state 28:    goto 29;
```

Table A-1. 33 MHz Simple DRAM Controller PLD Equations (Sheet 9 of 9)

```
    state 29:    goto 30;
    state 30:    goto 32;
    state 31:    goto 0;

"-------------------------------------------------------------------------
end
```

Table A-2. Signal and Product Term Allocation

OUTPUT MACROCELLS		BURIED MACROCELLS	
Signal	**Product Terms**	**Signal**	**Product Terms**
$\overline{\text{RAS}}$	7	$\overline{\text{CPU\_CYC}}$	3
$\overline{\text{CAS3}}$	3	$\overline{\text{REF\_REQ}}$	2
$\overline{\text{CAS2}}$	3	$\overline{\text{REF\_CYC}}$	3
$\overline{\text{CAS1}}$	3	$\overline{\text{HOLDOFF}}$	2
$\overline{\text{CAS0}}$	3	S1	4
DRA1	3	S0	3
DRA0	3	R4	4
$\overline{\text{DRDY}}$	2	R3	5
$\overline{\text{MUX}}$	1	R2	4
$\overline{\text{WE}}$	1	R1	3
$\overline{\text{RD}}$	2	R0	2
$\overline{\text{WR}}$	1		
$\overline{\text{OE245}}$	2		

APPENDIX B
IMPLEMENTATION FOR SPECIFIC PLDs

Appendix A contains device-independent programmable logic equations and fitter data. Appendix B partitions the design into specific PLDs which are readily available and relatively inexpensive.

According to the fitter report (Table A-2, Signal and Product Term Allocation), the maximum number of product terms in any state machine is seven. State machines like these, which need few product terms, can fit into simple PLDs.

Most of the state logic fits into PLD1, including the state machines RAS, MUX, DRDY, CPU_CYC, REF_CYC, RD and CAS. As indicated in Table B-1, these machines share a number of input and feedback signals. In this design, address decoding is performed externally in a 74F138 which feeds the single !DRAM input. PLD1 is a 22V10 device.

PLD2 contains the burst tracking state logic, refresh request flag and combinatorial DRAM address outputs. See Table B-2. PLD2 is a 20V8 device.

PLD3, illustrated in Table B-3, contains the refresh counter and a combinational output enable for a 74F245 bus transceiver. In the example, a 1.8432 MHz clock is divided to generate the refresh timer output R4 for PLD1. PLD3 is a simple 16R6 device.

Partitioning the design into multiple simple PLDs was not tested on actual hardware.

Table B-1. PLD 1 Source Code Table (Sheet 1 of 3)

```
ABEL 5.10  -  Device Utilization Chart

960CA DRAM Controller (21121)
Rick Schue  Intel Corp.  11/21/94



--------------------------------------------------------------------------------

Module                   : ‘pld1’

--------------------------------------------------------------------------------

Input files:

    ABEL PLA file        : pld1.tt3
    Device library       : P22V10.dev

Output files:

    Report file          : pld1.doc
    Programmer load file : pld1.jed
```

Table B-1. PLD 1 Source Code Table (Sheet 2 of 3)

```
P22V10 Programmed Logic:
--------------------------------------------------------------------------------



RAS.D   = (  !ADS & !DRAM & RAS.FB & MUX & REF_REQ
           #   BLAST & !RAS.FB & REF_CYC
           #   DRDY & !RAS.FB & REF_CYC
           #   RAS.FB & !CPU_CYC & REF_CYC
           #   BLAST & !RAS.FB & !CAS0
           #   DRDY & !RAS.FB & !CAS0
           #   RAS.FB & !REF_CYC & !CAS0 ); " ISTYPE 'INVERT'
RAS.C   = (  PCLK );

MUX.D   = (  RAS ); " ISTYPE 'BUFFER'
MUX.C   = (  PCLK );

CAS0.D   = (  !REF_REQ & CPU_CYC & CAS0.FB
            #   RAS & !REF_CYC & !CAS0.FB
            #   !RAS & !MUX & REF_CYC & CAS0.FB & !BE0 ); " ISTYPE 'INVERT'
CAS0.C   = (  PCLK );

CAS1.D   = (  !REF_REQ & CPU_CYC & CAS1.FB
            #   RAS & !REF_CYC & !CAS1.FB
            #   !RAS & !MUX & REF_CYC & CAS1.FB & !BE1 ); " ISTYPE 'INVERT'
CAS1.C   = (  PCLK );

CAS2.D   = (  !REF_REQ & CPU_CYC & CAS2.FB
            #   RAS & !REF_CYC & !CAS2.FB
            #   !RAS & !MUX & REF_CYC & CAS2.FB & !BE2 ); " ISTYPE 'INVERT'
CAS2.C   = (  PCLK );

CAS3.D   = (  !REF_REQ & CPU_CYC & CAS3.FB
            #   RAS & !REF_CYC & !CAS3.FB
            #   !RAS & !MUX & REF_CYC & CAS3.FB & !BE3 ); " ISTYPE 'INVERT'
CAS3.C   = (  PCLK );

DRDY.D   = (  !RAS & !MUX & REF_CYC & DRDY.FB
            #   RESET & !REF_CYC & !DRDY.FB ); " ISTYPE 'INVERT'
DRDY.C   = (  PCLK );

RD.D   = (  BLAST & RESET & !RD.FB
          #   !ADS & RD.FB & !READ ); " ISTYPE 'INVERT'
RD.C   = (  PCLK );

CPU_CYC.D   = (  CPU_CYC.FB & !ADS & !DRAM
               #   !CPU_CYC.FB & BLAST & RESET
               #   !CPU_CYC.FB & DRDY & RESET ); " ISTYPE 'INVERT'
CPU_CYC.C   = (  PCLK );

REF_CYC.D   = (  REF_REQ & REF_CYC.FB
               #   !CPU_CYC & REF_CYC.FB
               #   RAS & !MUX & !REF_CYC.FB ); " ISTYPE 'BUFFER'
REF_CYC.C   = (  PCLK );
```

Table B-1. PLD 1 Source Code Table (Sheet 3 of 3)

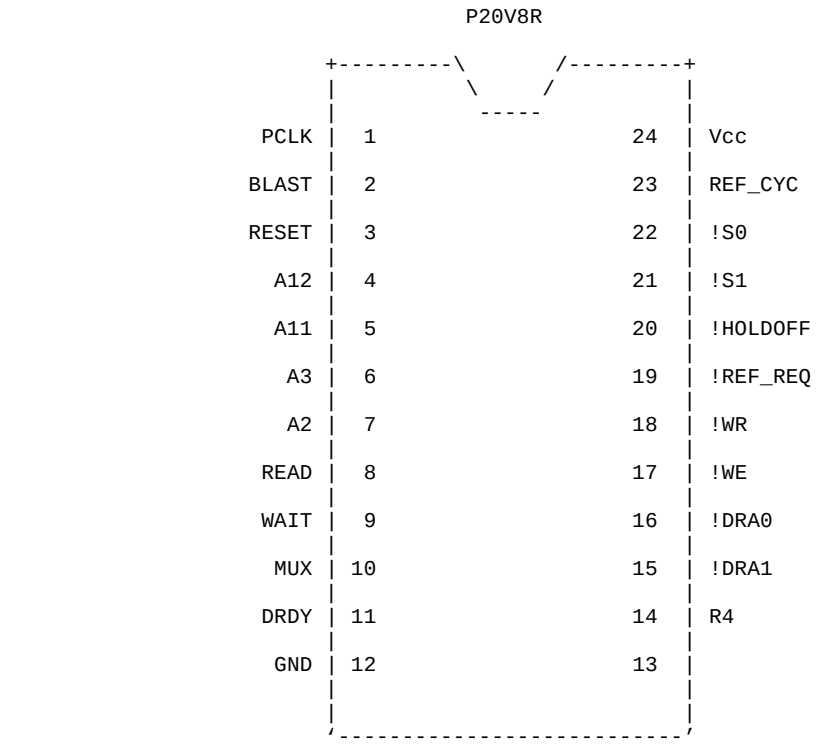
```
-------------------------------------------------------------------------



960CA DRAM Controller (21121)
Rick Schue  Intel Corp.  11/21/94

P22V10 Chip Diagram:
-------------------------------------------------------------------------


                                  P22V10

                          +---------\       /---------+
                          |          \     /          |
                          |           -----           |
                    PCLK  |  1                   24   |  Vcc
                          |                           |
                     ADS  |  2                   23   |  !RAS
                          |                           |
                   BLAST  |  3                   22   |  !CAS3
                          |                           |
                   RESET  |  4                   21   |  !CAS2
                          |                           |
                    DRAM  |  5                   20   |  !CAS1
                          |                           |
                     BE0  |  6                   19   |  !CAS0
                          |                           |
                     BE1  |  7                   18   |  !DRDY
                          |                           |
                     BE2  |  8                   17   |  MUX
                          |                           |
                     BE3  |  9                   16   |  !RD
                          |                           |
                    READ  | 10                   15   |  !CPU_CYC
                          |                           |
                 REF_REQ  | 11                   14   |  REF_CYC
                          |                           |
                     GND  | 12                   13   |
                          |                           |
                          |                           |
                          '---------------------------'

                             SIGNATURE: N/A
```

Table B-2. PLD 2 Source Code Table (Sheet 1 of 2)

```
ABEL 5.10  -  Device Utilization Chart

960CA DRAM Controller (21121)
Rick Schue  Intel Corp.  11/21/94



--------------------------------------------------------------------------------

Module                   : 'pld2'

--------------------------------------------------------------------------------

Input files:

    ABEL PLA file        : pld2.tt3
    Device library       : P20V8R.dev

Output files:

    Report file          : pld2.doc
    Programmer load file : pld2.jed

P20V8R Programmed Logic:
--------------------------------------------------------------------------------



DRA1     = !(  !S1 & S0
           #   !S1 & MUX & !A12
           #   !S1 & !MUX & !A3 );

DRA0     = !(  S1 & !S0
           #   !S0 & MUX & !A11
           #   !S0 & !MUX & !A2 );

WE     = !(  READ & REF_CYC );

WR     = !(  READ & !WAIT );

REF_REQ.D   = (  REF_CYC & !REF_REQ.FB
              #   R4 & REF_REQ.FB & HOLDOFF ); " ISTYPE 'INVERT'
REF_REQ.C   = (  PCLK );

HOLDOFF.D   = (  !REF_CYC & HOLDOFF.FB
              #   !HOLDOFF.FB & R4 ); " ISTYPE 'INVERT'
HOLDOFF.C   = (  PCLK );

S1.D   = (  !A3 & !S1.FB & !S0.FB
         #   A2 & !S1.FB & !S0.FB
```

Table B-2. PLD 2 Source Code Table (Sheet 2 of 2)

```
          #   !S1.FB & DRDY
          #   !DRDY & !BLAST ); " ISTYPE 'INVERT'
S1.C   = (  PCLK );

S0.D   = !(  S0.FB & DRDY
          #   S1.FB & !DRDY & BLAST
          #   !A2 & !S0.FB & !DRDY & BLAST ); " ISTYPE 'INVERT'
S0.C   = (  PCLK );


P20V8R Chip Diagram:
---------------------------------------------------------------------------
```



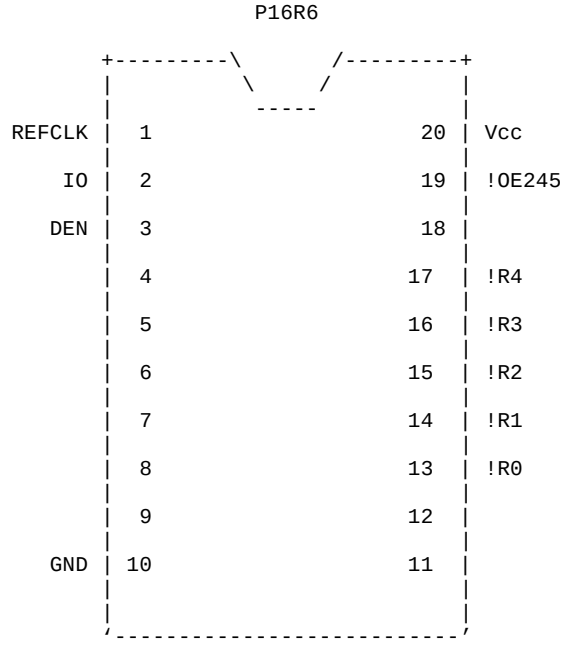

```
   SIGNATURE: N/A
```

Table B-3. PLD 3 Source Code Table (Sheet 1 of 2)

```
ABEL 5.10  -  Device Utilization Chart

960CA DRAM Controller (21121)
Rick Schue  Intel Corp.  11/21/94



--------------------------------------------------------------------------------

Module                   : 'pld3'

--------------------------------------------------------------------------------

Input files:

    ABEL PLA file        : pld3.tt3
    Device library       : P16R6.dev

Output files:

    Report file          : pld3.doc
    Programmer load file : pld3.jed


P16R6 Programmed Logic:
--------------------------------------------------------------------------------



OE245     = !(  !DEN & !IO );

R4.D   = (  !R4.FB & !R3.FB
          #   R4.FB & R3.FB & R2.FB & R1.FB
          #   !R4.FB & !R1.FB
          #   R3.FB & !R2.FB & R1.FB & R0.FB
          #   !R4.FB & !R0.FB ); " ISTYPE 'INVERT'
R4.C   = (  REFCLK );

R3.D   = (  !R3.FB & !R2.FB
          #   !R3.FB & !R1.FB
          #   R4.FB & R3.FB & R1.FB & R0.FB
          #   R3.FB & R2.FB & R1.FB & R0.FB
          #   !R3.FB & !R0.FB
          #   R4.FB & R2.FB & R1.FB & !R0.FB ); " ISTYPE 'INVERT'
R3.C   = (  REFCLK );

R2.D   = (  R4.FB & R3.FB & R1.FB
          #   !R2.FB & !R1.FB
          #   R2.FB & R1.FB & R0.FB
          #   !R2.FB & !R0.FB ); " ISTYPE 'INVERT'
```

Table B-3. PLD 3 Source Code Table (Sheet 2 of 2)

```
R2.C   = (  REFCLK );

R1.D   = (  R1.FB & R0.FB
         #   R4.FB & R3.FB & R2.FB & !R0.FB
         #   !R1.FB & !R0.FB ); " ISTYPE 'INVERT'
R1.C   = (  REFCLK );

R0.D   = (  R4.FB & R3.FB & R2.FB & R1.FB
         #   R0.FB ); " ISTYPE 'INVERT'
R0.C   = (  REFCLK );



P16R6 Chip Diagram:
---------------------------------------------------------------------------------
```



SIGNATURE: N/A