AP-726

APPLICATION NOTE

Interfacing the i960® Jx Microprocessor to the NEC µPD98401* Local ATM Segmentation and Reassembly (SAR) Chip

Rick Harris
SPG 80960 Applications Engineer

Intel Corporation
Semiconductor Products Group
Mail Stop CH6-311
5000 W. Chandler Blvd.
Chandler, Arizona 85226

September, 21, 1995

Order Number: 272779-001

AP-726

APPLICATION NOTE

Interfacing the i960® Jx Microprocessor to the NEC µPD98401* Local ATM Segmentation and Reassembly (SAR) Chip

Rick Harris
SPG 80960 Applications Engineer

Intel Corporation
Semiconductor Products Group
Mail Stop CH6-311
5000 W. Chandler Blvd.
Chandler, Arizona 85226

September, 21, 1995

Order Number: 272779-001

1.0 INTRODUCTION 1
1.1 Design Goals 1
1.2 Overview 1
1.3 Page Mode DRAM SIMM 2
1.4 DRAM Refresh 2
1.5 Burst Capabilities for a 32-bit Bus 2
1.6 Bank Interleaving 2
1.7 i960 Jx Processor Address Mapping 3

2.0 CIRCUIT DESCRIPTION 3
2.1 Clock Generation 3
2.2 System Address Latches 3
2.3 DRAM Controller 5
2.3.1 Control Logic 5
2.3.2 Address Flow Logic 5
2.3.3 Data Flow Logic 6
2.3.4 State Machines and Signals 6
2.3.4.1 DRAM Control State Machine 6
2.3.4.2 SELDRAM Signal 6
2.3.4.3 $\overline{\text{REFPEND}}$ Signal 6
2.3.4.4 $\overline{\text{MEMPEND}}$ Signal 8
2.3.4.5 $\overline{\text{MUX}}$ Signal 8
2.3.4.6 $\overline{\text{CASEN}}$ Signal 9
2.3.4.7 $\overline{\text{INCBANK}}$ and $\overline{\text{BANKSEL}}$ Signals 9
2.3.4.8 $\overline{\text{CASA[3:0]}}$ and $\overline{\text{CASB[3:0]}}$ Signals 9
2.3.4.9 MA0B and MA0A Signals 9
2.3.4.10 $\overline{\text{RDYEN}}$ Signal 9
2.3.4.11 $\overline{\text{READY}}$ Signal 9
2.3.4.12 $\overline{\text{DWE}}$ Signal 9
2.4 SAR slave controller 10
2.4.1 Control Logic 10
2.4.2 Address Flow Logic 10
2.4.3 Data Flow Logic 10
2.4.4 State Machine and Signals 10
2.4.4.1 SELC Signal 11
2.4.4.2 $\overline{\text{SELSQ0}}$ Signal 11
2.4.4.3 SAR Access State Machine 11
2.4.4.4 $\overline{\text{READYn}}$ Signal 11
2.4.4.5 $\overline{\text{READYpinn}}$ Signal 11
2.4.4.6 $\overline{\text{RDYENn}}$ Signal 11
2.4.4.7 $\overline{\text{SARSELn}}$ Signal 11
2.4.4.8 $\overline{\text{SARASELn}}$ Signal 11

2.4.4.9 $\overline{\text{SARSWRn}}$ Signal 11
2.4.4.10 $\overline{\text{AOEABn}}$ Signal 11
2.4.4.11 $\overline{\text{DOEABn}}$ Signal 11
2.4.4.12 $\overline{\text{DOEBAn}}$ Signal 12

3.0 DRAM CONTROLLER ACCESS FLOW 12
3.1 Single Word Read and Write Access 12
3.2 Quad Word Read Access 14
3.3 Quad Word Write Access 14
3.4 $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ Refresh 17

4.0 SAR SLAVE ACCESS FLOW 17
4.1 Single Word Read and Write Access 17

5.0 CONCLUSION 20

6.0 RELATED INFORMATION 20

APPENDIX A
PLDshell FILES

APPENDIX B
PLD EQUATIONS

INDEX

FIGURES

Figure 1. Quad-Word Write Request with 2,1,1,1 Wait State Profile..... 2
Figure 2. Two-Way Interleaving 3
Figure 3. 80960Jx Interface Memory Map..... 3
Figure 4. i960®Jx Processor to DRAM / NEC µPD98401 SAR Bus Interface..... 4
Figure 5. System Address Latches 5
Figure 6. DRAM Address Flow Logic 5
Figure 7. DRAM Control State Machine 7
Figure 8. Simplified DRAM State Machine..... 8
Figure 9. SAR Address and Data Buffers..... 10
Figure 10. 80960Jx to SAR Access State Machine..... 11
Figure 11. 80960Jx to DRAM 1 Wait State Single Word Read and Write Timing Diagram ... 13
Figure 12. 80960Jx to DRAM 1,0,0,0 Quad Word Read Timing 15
Figure 13. 80960Jx to DRAM 1,1,1,1 Quad Word Write Timing Diagram..... 16
Figure 14. $\overline{\text{CAS}}$-Before-$\overline{\text{RAS}}$ Refresh Cycle..... 17
Figure 15. 80960Jx to µPD98401 Single Word Read and Write Timing Diagram..... 19

TABLES

Table 1. Related Information 20
Table A-1. DRAM Controller EPX780 PLDshell File..... 1
Table A-2. DRAM Controller PAL20V8 PLDshell File..... 13
Table B-1. SAR Controller MACH210, MACHXL File 1

1.0 INTRODUCTION

This application note describes the interface between Intel's i960® JA/JF/JD microprocessors (referred to herein as the "i960 Jx processor") and NEC's µPD98401® Local ATM Segmentation And Reassembly (SAR) Chip, with 2 Mbytes of DRAM available to the i960 Jx processor. The interface provides for single and burst reads/writes to the DRAM and single reads and writes to the SAR chip, using the i960 Jx processor configured with a 32-bit bus. The interface is controlled by a DRAM controller and a SAR controller.

The DRAM interface features include:

- 32 bit wide bus, allowing 16 and 8 bit accesses
- Two-way interleaving
- 72 pin, 60 ns, 256 Kbyte x 32 bit Fast Page Mode DRAM SIMMs
- 1-0-0-0 wait state burst reads at 25 MHz
- 1-1-1-1 wait state burst writes at 25 MHz

The µPD98401 SAR chip interface features three wait state 32 bit reads and writes.

This document discusses DRAM controller theory with interleaving, and a basic slave read/write controller to the SAR. It also describes the supporting state machines, timing diagrams, and PLD equations. The µPD98401 can operate as a master device, which allows it access to DRAM. However, when the SAR is a master device, it can only request and obtain the i960 Jx processor bus, through the use of the processor's bus arbitration facilities. In this case, upon receiving HOLD from the SAR controller, the processor asserts HOLDA and three-states its signals, which grants bus control to the SAR.

1.1 Design Goals

The goal of this document is to provide a design aid for interfacing an i960 Jx processor to an NEC µPD98401 ATM SAR Chip. The application note defines an interface that was developed and tested on a i960 Jx processor-compliant platform, and can be plugged into a PCI expansion slot.

The platform used is the Intel Cyclone i960® Microprocessor PCI-SDK Evaluation Platform (referred to herein as the "PCI-SDK"), with an ATM Squall module and a i960 Jx processor CPU module.

Benefits of using the PCI-SDK include:

- The PCI-SDK plugs directly into a DOS-based system's PCI slot.
- While the user is working on their hardware design, code (software) for the design can be written concurrently, which saves development time.
- The design can be implemented on the PCI-SDK through a Squall module, which enhances testability.

1.2 Overview

The PCI-SDK consists of a base board with connectors which accept "modules" (daughter boards) for a CPU module and a Squall module. Many CPU and Squall modules are available; refer to Section 6.0, RELATED INFORMATION (pg. 6-20) for access to additional product information.

- The CPU modules are interchangeable daughter boards for Intel's family of i960 processors.
- The Squall modules are interchangeable daughter boards. Users can use these to design custom applications, or use one of PCI-SDK's existing modules, such as an Ethernet Squall II* module.

The PCI-SDK base board features include:

- Two SIMM sockets which support 2, 8, or 32 Mbytes of DRAM
- One Flash ROM socket
- Three 16-bit counter/timers or one 32-bit and one 16-bit counter
- DIP switch-selectable CPU clock frequency, for operation from 16 MHz to 50 MHz
- DRAM controller which automatically optimizes wait states to CPU frequency and memory speed
- One RS-232 serial port
- One Centronics compatible parallel download port
- PCI Bus Interface

The configuration used for development of this application note is:

CPU Module:	i960 Jx processor
CPU Bus Frequency:	25 MHz
DRAM:	2 banks of SIMMs: 60 ns, 2 Mbyte
Squall Module:	NEC µPD98401® - based ATM module

The chosen configuration simplifies the state machine for the DRAM controller (compare Figures 7 and 8).

1.3 Page Mode DRAM SIMM

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 $\overline{\text{CAS}}$ while keeping $\overline{\text{RAS}}$ active, then asserting $\overline{\text{CAS}}$ with the new column address valid to the DRAM. Page mode operation works very well with burst buses, such as the i960 Jx processor bus, in which a single address cycle can be followed by up to four data cycles.

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

1.4 DRAM Refresh

To maintain their data, the DRAM must be refreshed every 16 µs. This is achieved by a 6-bit refresh counter embedded in the DRAM controller, driven by a 4 MHz clock. The 4 MHz clock is created by dividing a 16 MHz clock in half, twice. This corresponds to a refresh every 64 cycles of the 4 MHz clock, or approximately 400 cycles of the 25 MHz clock. The type refresh used is $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ ($\overline{\text{CAS}}$ signal asserted before the $\overline{\text{RAS}}$ signal) for specified lengths of time. Both banks of DRAM are refreshed at the same time.

1.5 Burst Capabilities for a 32-bit Bus

The i960 Jx processor can access up to four data words per request. A request starts when the processor asserts $\overline{\text{ADS}}$ in the address cycle and ends when the processor asserts $\overline{\text{BLAST}}$ in the last data cycle. Figure 1 shows $\overline{\text{ADS}}$ and $\overline{\text{BLAST}}$ timings for a quad-word write request.

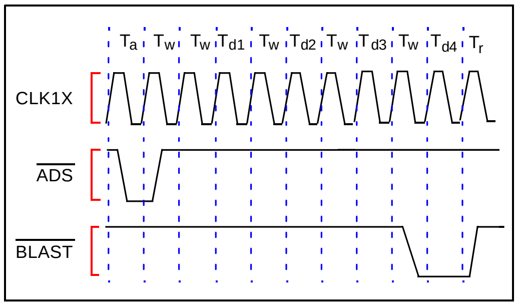


Figure 1. Quad-Word Write Request with 2,1,1,1 Wait State Profile

The processor's burst capabilities on a 32-bit bus include:

- Quad-word and triple-word requests start on quad-word boundaries (A3 = 0, A2 = 0).
- Double-word requests start on double-word boundaries (A3 = X, A2 = 0).
- Single-word requests can start on any word boundary (A3 = X, A2 = X).
- Any request starting on an odd word boundary never bursts (A3 = X, A2 = 1).

1.6 Bank Interleaving

Bank interleaving allows the second, third and fourth accesses of a burst read to occur in zero wait states. The first data access must still “pay” the entire access penalty. Interleaving significantly improves memory system performance by overlapping accesses to consecutive addresses. Two-way interleaving is accomplished by dividing the memory into two 32-bit banks (also referred to as “leaves”):

- one bank for even word addresses (A2=0)
- one bank for odd word addresses (A2=1)

The two banks are read in parallel and the data from the two banks is multiplexed into the processor’s data bus. Multiplexing is implemented via $\overline{\text{CASAx}}$ and $\overline{\text{CASBx}}$. $\overline{\text{CAS}}$ signals, in addition to being the Column Address Strobes, are also output enable signals for the DRAM.

Figure 2 shows DRAM with a 2-1-1-1 quad word burst read wait state profile interleaved to generate a 2-0-0-0 wait state system.

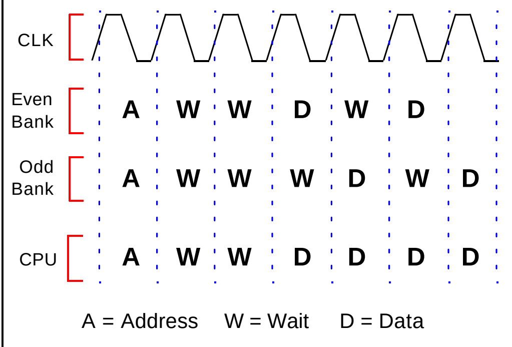


Figure 2. Two-Way Interleaving

1.7 i960 Jx Processor Address Mapping

Figure 3 portrays the memory map for the i960 Jx processor to SAR interface. The DRAM beginning location is set at A0000000H and the SAR beginning location is set at C0000000H. When configuring the processor's memory regions using the PMCON registers, make sure regions PMCON10_11 and PMCON12_13 are set for 32 bit wide bus accesses.

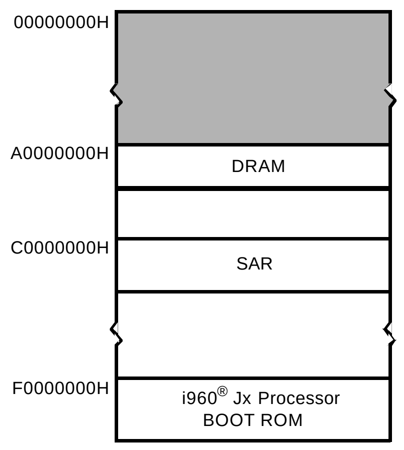


Figure 3. 80960Jx Interface Memory Map

2.0 CIRCUIT DESCRIPTION

Figure 4 shows a block diagram of the i960 Jx processor, interfaced to DRAM and the SAR. It can be separated into four logical blocks: clock generation, address latches, DRAM controller, and SAR slave controller. These blocks are described in the following subsections.

2.1 Clock Generation

The 25 MHz CPU clock, based on the FREQ[2:0] frequency switches, is generated from an AV9155-01. The AV9155-01 feeds the 25 MHz clock into a CY7BB991-7 with an internal PLL. Also, the AV9155-01 directly generates the 16 MHz clock that is later divided down to 4 MHz, used for DRAM refresh generation.

Clock distribution is performed by the CY7BB991-7. Seperate output clocks from this device are distributed back to the CPU module, Squall II Module, and on-board logic. This device guarantees a maximum skew of ±250 ps between outputs, and ±500 ps between the inputs and outputs. Therefore, all clocks on the board are within ±1 ns, making the design work very straightforward. All clock signals are terminated with 22 ohm series resistors.

2.2 System Address Latches

The i960 Jx processor has a multiplexed bus; therefore, address latches are needed to demultiplex the bus for use in accessing the DRAM. The latches, shown in Figure 5, capture the upper 28 address bits of the bus during the processor's address cycle, then holds them until the access is over. ALE provides the signal for latching, and HOLDA controls the output enables of the latches. HOLDA isolates the address latches from the system bus when the SAR uses the bus as a master device. As a master device, the SAR requests the bus with HOLD; the processor grants the bus with HOLDA. When HOLDA is high, the outputs to the latches are three-stated.

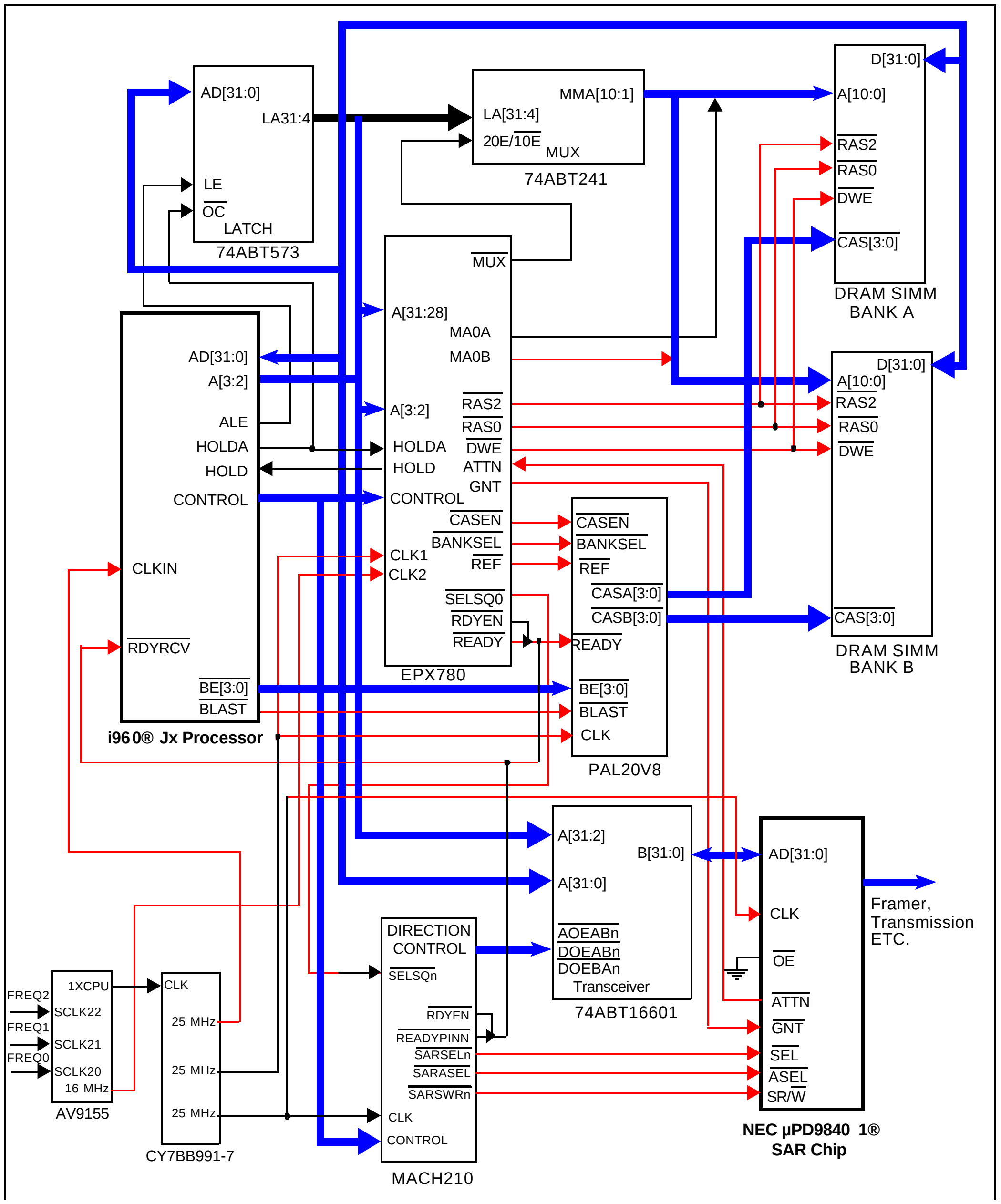


Figure 4. i960® Jx Processor to DRAM / NEC µPD98401 SAR Bus Interface

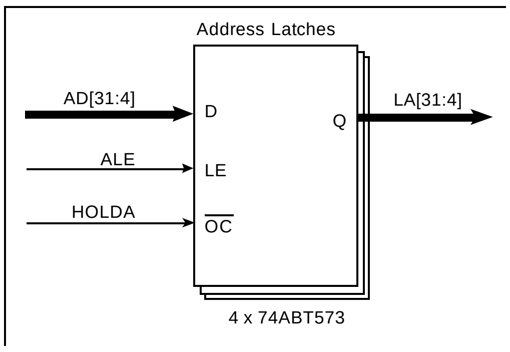


Figure 5. System Address Latches

2.3 DRAM Controller

The DRAM controller can be separated into four sections: control logic, address flow logic, data flow logic, and DRAM controller state machines and signals. See Figure 4.

2.3.1 Control Logic

The control logic is implemented using an EPX780 FPGA and a 20V8 PAL. The FPGA, the main component of the DRAM controller, supplies:

- $\overline{RAS}$ lines and read/write signals directly to the DRAM
- address bits to each DRAM bank that increments the column address
- logic to control the address path to the DRAM

The 20V8 PAL, with direction from the FPGA, provides all the necessary logic to generate the $\overline{CAS}$ signals for the DRAM.

2.3.2 Address Flow Logic

Figure 6 shows the DRAM address flow logic. The address flow logic to the DRAM is controlled by three 74ABT241 octal buffers, configured as a 12-input, two-line to one-line multiplexer. The $\overline{MUX}$ select signal, which controls the multiplexing function, is supplied by the FPGA. The multiplexers divide the latched address bus into two addresses: the row address consisting of the latched address' higher order bits, and the column address consisting of the latched address' lower order bits.

Both row and column addresses are transferred to DRAM via the MMA[10:1] address bus. MA0a connects to A0 of the bank A DRAM, MA0b connects to A0 of the bank B DRAM.

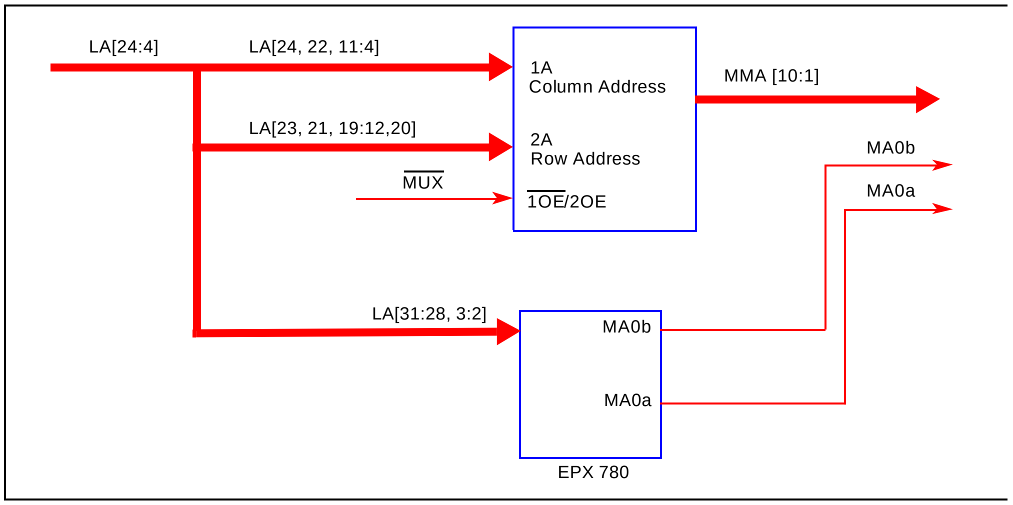


Figure 6. DRAM Address Flow Logic

2.3.3 Data Flow Logic

As indicated in Figure 4, the data path is fairly simple; it's connected directly to the DRAM from the processor, requiring no transceivers or multiplexers to control data flow between the banks. Data flow is controlled by the $\overline{\text{CAS}}$ lines. When one bank is selected ($\overline{\text{CAS}}$ lines asserted) the other bank is de-selected ($\overline{\text{CAS}}$ lines deasserted). Data is output by the DRAM during a read or accepted by the DRAM during a write by the assertion $\overline{\text{CAS}}$.

2.3.4 State Machines and Signals

Figure 7 shows the DRAM control state machine for the PCI-SDK; Figure 8 shows a simplified version. The simplified version is the result of choosing the configuration discussed in Section 1.2, Overview (pg. 1-1). For this configuration, the CPU module Frequency Switches are set to 25 MHz (Positions 1-4 should be OFF, OFF, ON, ON. This maps to a logic 1,1,0,0. Refer to Cyclone documentation for definitions).

The PD3 pin output from the DRAM SIMM, when sampled high, indicates 60 ns DRAM. With these frequency and speed settings, the signal PF0 is set high and all the other PF signals are reset low. The PCI-SDK does not use the EXTEND signal; it is tied high.

The state machine has two major paths: a refresh path and an access path. The refresh path is taken when the refresh counter indicates a refresh is needed. The access path is taken when the processor initiates an access to the address range dedicated to the DRAM.

When a conflict occurs between an access and a refresh, the refresh has priority and proceeds; a memory pending bit is set. When refresh completes, the memory pending bit causes the access to complete. The same process occurs when an access is in progress and a request for a refresh is received. A refresh pending bit is set which causes the refresh to complete when the access completes.

The DRAM state machine is developed with Intel's PLDshell Plus®. Appendix A, PLDshell FILES contains the code which implement the DRAM controller.

The state machine transitions described here use the following PLDshell conventions for logic operators. For all the following, refer to Figure 8.

/	Represents active-low in pin declaration; Boolean NOT elsewhere in file
*	Represents Boolean AND
+	Represents Boolean OR

2.3.4.1 DRAM Control State Machine

The DRAM control state machine, the heart of DRAM controller, resides in the FPGA. It is controlled with inputs from the processor and equations from within the controller. Its outputs are the $\overline{\text{RAS}}$ and $\overline{\text{REF}}$ signals, both are active low. The $\overline{\text{RAS}}$ signals directly drive the $\overline{\text{RAS}}$ inputs of both banks and are used to latch the row addresses into the DRAM. Two identical $\overline{\text{RAS}}$ signals distribute the load between the two banks. $\overline{\text{REF}}$ provides a bit to the $\overline{\text{REFPEND}}$ state machine, indicating a refresh has begun. This resets the $\overline{\text{REFPEND}}$ signal, so it can be set the next time a refresh is required.

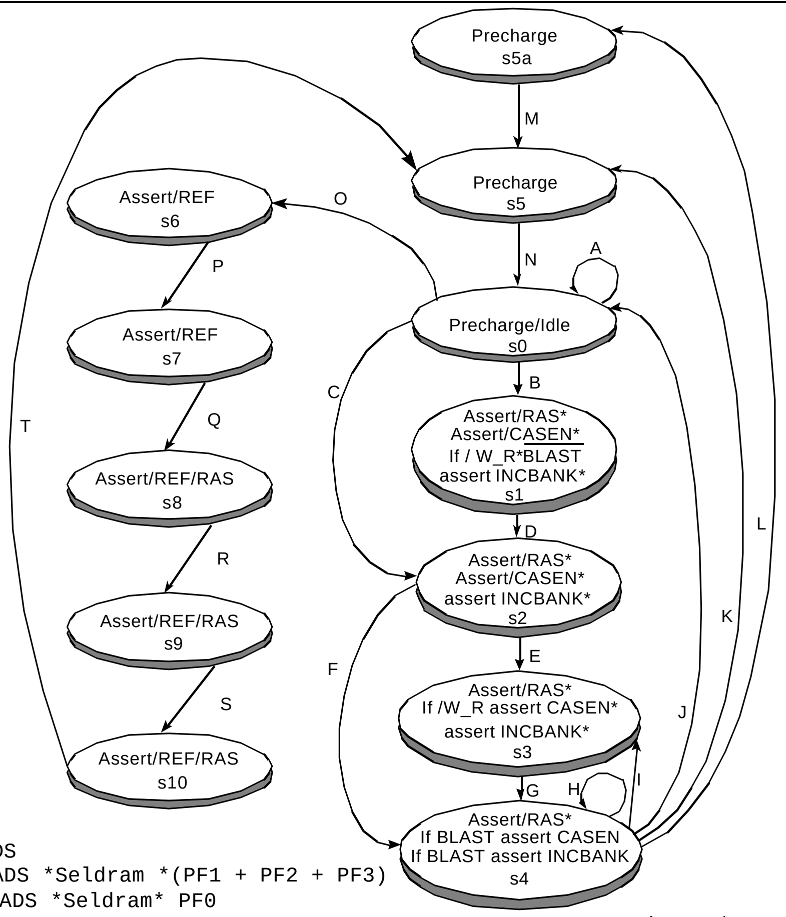
The other output functions of the DRAM controller depend upon which state the DRAM control state machine is in. The following are the signals that make up the DRAM controller.

2.3.4.2 SELDRAM Signal

The active high SELDRAM signal is decoded from the latched address bits LA[31:28]. When these address bits equal A_{16} during the processor's address cycle, SELDRAM is asserted indicating an access to the DRAM memory region.

2.3.4.3 $\overline{\text{REFPEND}}$ Signal

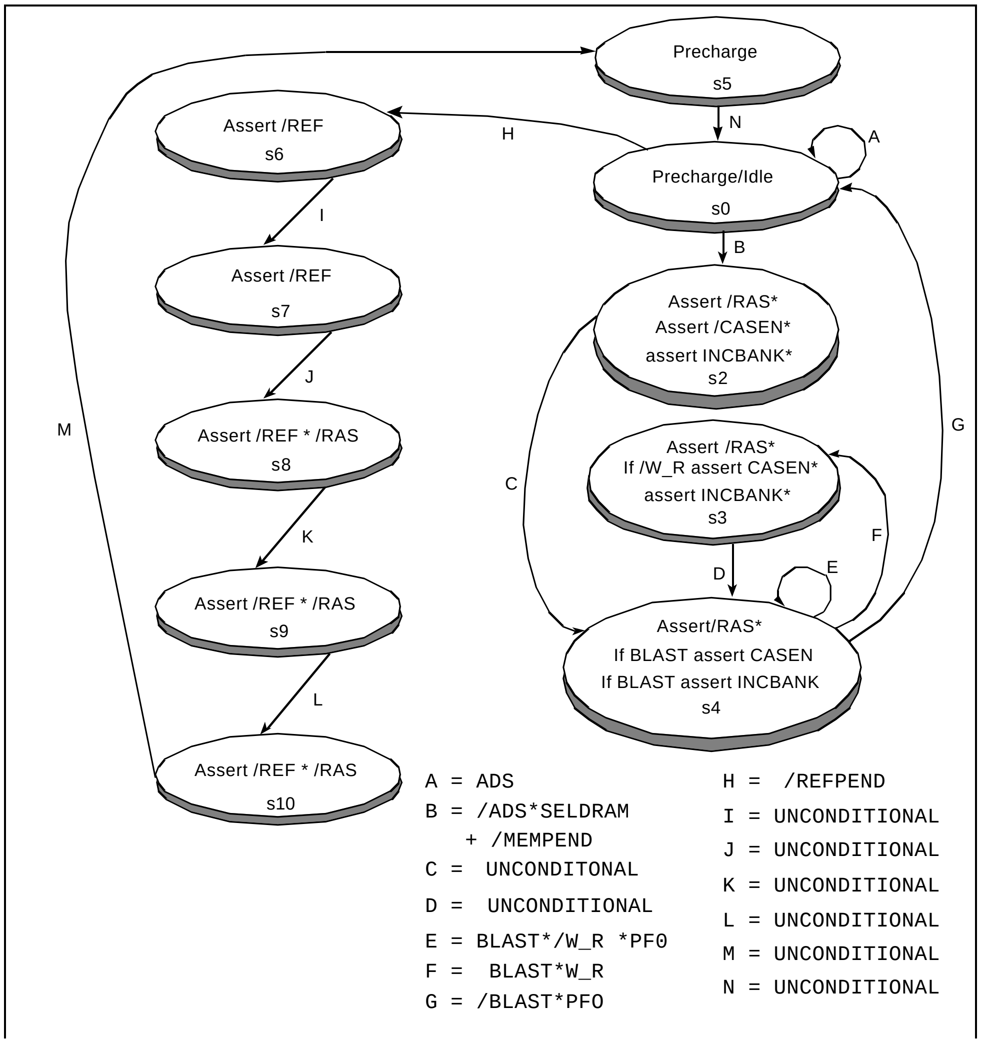
The active low $\overline{\text{REFPEND}}$ signal is asserted when a refresh is requested, either during the idle state or a DRAM access. Its main function is to keep track of when a refresh is required, so the controller will not skip the refresh if it is requested during an access. It is also the bit that causes the refresh to complete first, when in conflict with an access. Since the refresh counter runs at 4 MHz, and the $\overline{\text{REFPEND}}$ signal runs on the 25 MHz, the counter output and the $\overline{\text{REFPEND}}$ signal must be synchronized. The signal used for synchronization is the $\overline{\text{REFSYNC}}$ (see Table A-1 in Appendix A) signal embedded in the controller.



```
A = ADS
B = /ADS *Seldram *(PF1 + PF2 + PF3)
C =  /ADS *Seldram* PF0
  + /MEMPEND * PFO
D =  UNCONDITIONAL
E = /W_R * (PF2 + PF3)
F =  W_R * (PF2 + PF3)
  +PF1
  +PF0
G = UNCONDITIONAL
H =  BLAST*/W_R *PF0
  + BLAST*/W_R *PF1
  + /EXTEND */W_R
I = BLAST* /W_R *PF3
 + BLAST*/W_R*PF2
 + BLAST*W_R
```

```
J = /BLAST*PF0
K = /BLAST*PF3*/W_R
  + /BLAST *PF1
  + /BLAST *PF2
L = /BLAST *PF3*W_R
M = UNCONDITIONAL
N = UNCONDITIONAL
O = /REFPEND
P = UNCONDITIONAL
Q = UNCONDITIONAL
R = UNCONDITIONAL
S = UNCONDITIONAL
T = UNCONDITIONAL
```

Figure 7. DRAM Control State Machine



A = ADS	H = /REFPEND
B = /ADS*SELDRAM + /MEMPEND	I = UNCONDITIONAL
C = UNCONDITONAL	J = UNCONDITIONAL
D = UNCONDITIONAL	K = UNCONDITIONAL
E = BLAST*/W_R *PF0	L = UNCONDITIONAL
F = BLAST*W_R	M = UNCONDITIONAL
G = /BLAST*PFO	N = UNCONDITIONAL

Figure 8. Simplified DRAM State Machine

2.3.4.4 $\overline{\text{MEMPEND}}$ Signal

The active low $\overline{\text{MEMPEND}}$ signal function is similar to the $\overline{\text{REFPEND}}$ signal, except it keeps track of an access when the controller is performing a refresh. It is asserted when an access is requested either during the idle state or a refresh.

2.3.4.5 $\overline{\text{MUX}}$ Signal

The $\overline{\text{MUX}}$ signal, clocked by the falling edge of the 25 MHz clock, controls whether a row or column address is input to DRAM. It is actually a $\overline{\text{RAS}}$ signal delayed by a half clock cycle. The delay allows enough row address hold time (t_{RAH}) which DRAM requires. When $\overline{\text{MUX}}$ is high, the row address to DRAM is valid; when $\overline{\text{MUX}}$ is low, the column address is valid.

2.3.4.6 $\overline{\text{CASEN}}$ Signal

The active low $\overline{\text{CASEN}}$ signal provides an output enable signal to the $\overline{\text{CAS}}$ state machines, located in the PAL20V8. Its function is determined by which state the DRAM Control state machine is in:

- s0: asserted during an address cycle in which one or more are true:
  - access is to the DRAM memory region
  - a refresh is not pending
  - an access is pending
- s2 and s3: $\overline{\text{CASEN}}$ is asserted when the access is a read. (Remember that PF0 is asserted, indicating the system is operating at 25 MHz with 60 ns DRAM.)
- s4: asserted during a burst read or write access EXCEPT for the last access of the burst.

2.3.4.7 $\overline{\text{INCBANK}}$ and $\overline{\text{BANKSEL}}$ Signals

The active low $\overline{\text{BANKSEL}}$ signal determines which bank will have its $\overline{\text{CAS}}$ lines driven:

- When low, selects bank A
- When high, selects bank B

The active low $\overline{\text{INCBANK}}$ signal, in conjunction with the $\overline{\text{BANKSEL}}$ signal, creates wait states by controlling the $\overline{\text{CAS}}$ lines. When asserted low at the end of a clock cycle, the $\overline{\text{CAS}}$ lines are asserted, thereby selecting each bank. The $\overline{\text{INCBANK}}$ signal operates based upon which state the DRAM Control state machine is in.

- s0: asserted during an address cycle, in which one or more are true:
  - access is to the DRAM memory region
  - a refresh is not pending
  - an access is pending
- s2: asserted if the access is a burst read, EXCEPT for the last access of the burst.
- s4: asserted during a burst read or write access, EXCEPT for the last access of the burst.

2.3.4.8 $\overline{\text{CASA[3:0]}}$ and $\overline{\text{CASB[3:0]}}$ Signals

The active low $\overline{\text{CASA}}$ and $\overline{\text{CASB}}$ signals connect to the $\overline{\text{CAS}}$ pins of DRAM banks A and B respectively. They are enabled by the $\overline{\text{CASEN}}$ and $\overline{\text{BANKSEL}}$ signals. With $\overline{\text{BANKSEL}}$ low bank A is selected, with $\overline{\text{BANKSEL}}$ high bank B is selected. These signals serve two purposes: to latch the column addresses into the DRAM, and to provide chip selects for each bank. $\overline{\text{CASA3}}$ and $\overline{\text{CASB3}}$ enables the most significant byte, and $\overline{\text{CASA0}}$ and $\overline{\text{CASB0}}$ enable least significant byte. The byte enables, $\overline{\text{BE[3:0]}}$ control which byte(s) to be transferred within each bank, allowing 8 and 16 bit wide data transfers. $\overline{\text{BE[3:0]}}$ controls bytes 3-0 respectively.

2.3.4.9 MA0B and MA0A Signals

MA0B and MA0A are the least significant column address bits that increment the address to DRAM banks B and A respectively, during a burst access. They are decoded from address bits A3 and A2 of the processor's bus, which increment during a burst.

2.3.4.10 $\overline{\text{RDYEN}}$ Signal

The active low $\overline{\text{RDYEN}}$ signal enables the three-state buffer that controls the $\overline{\text{READY}}$ signal. Since the system has two controllers on it, i.e. a DRAM controller and a SAR controller, the $\overline{\text{READY}}$ signal of the DRAM controller has to be three-stated when the SAR controller drives its $\overline{\text{READY}}$ signal. This prevents contention between the two ready signals. The $\overline{\text{RDYEN}}$ signal is asserted based upon which state the DRAM control state machine is in. It's asserted during states s0, s2, s3 and s4, regardless if $\overline{\text{READY}}$ is asserted or not.

2.3.4.11 $\overline{\text{READY}}$ Signal

The active low $\overline{\text{READY}}$ signal is connected to the $\overline{\text{RDYRCV}}$ pin of the processor. It indicates to the processor that data can be transferred, and that the address can be incremented if the access is a burst. It is asserted based upon which state the DRAM control state machine is in. It is asserted unconditionally in states s2 and s3. It is asserted in s4 if the access is a burst read, provided it's not the last access of the burst.

2.3.4.12 $\overline{\text{DWE}}$ Signal

The active low $\overline{\text{DWE}}$ signal indicates to the DRAM that the access is a write, provided the controller is not in refresh. The write enable signal of the DRAM's must be held high during refresh.

2.4 SAR slave controller

The SAR slave controller controls access to the SAR in slave mode. Reading and writing the SAR in slave mode accesses 32 bit registers that initialize and control the SAR. Referring to Figure 4, the SAR slave controller is discussed in 4 sections: control logic, address flow logic, data flow logic, and the SAR slave controller state machines and signals.

2.4.1 Control Logic

The control logic is implemented using the EPX780 FPGA and a MACH210 PLD. The FPGA decodes the SAR's address space, and provides the select signal to the SAR's access state machine. The MACH210 PLD provides all the logic that makes up the SAR controller, which controls the sequencing of the SAR itself, and the signals that control the SAR's address and data transceivers.

2.4.2 Address Flow Logic

Figure 9 shows the SAR Address Buffers. The address for the SAR is controlled by two 74ABT16601 universal bus transceivers. The $\overline{\text{AOEABn}}$ control signal, output from the SAR controller, enables the latched address bits onto the multiplexed AD bus of the SAR. When inactive, the outputs of the transceivers are three-stated, allowing data to be driven on the data bus without being in contention with the address bus.

2.4.3 Data Flow Logic

Figure 9 shows the SAR Data Buffers. The data for the SAR is controlled by two 74ABT16601 universal bus transceivers. The $\overline{\text{DOEABn}}$ control signal, output from the SAR controller, enables the data onto the multiplexed AD bus of the SAR, when the SAR is being written to. The $\overline{\text{DOEBAn}}$ control signal, output from the SAR controller, enables the data onto the AD bus of the processor, when the SAR is being read. When either control signal is inactive, both sides A and B of the data buffers are three-stated, allowing the bus to be used for other functions.

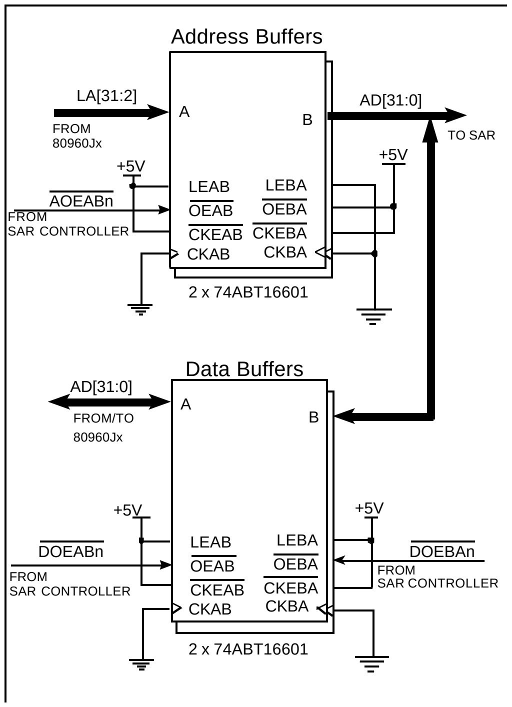


Figure 9. SAR Address and Data Buffers

2.4.4 State Machine and Signals

The SAR slave controller consists of the SAR access state machine, and several equations that implement the control signals. The SAR controller resides in the MACH210 PLD.

The SAR controller is written in MACHXL. Appendix B, PLD EQUATIONS contain the code which implement the SAR controller. The state machine transitions described here follow the MACHXL conventions for logic operators.

/	Represents Boolean NOT
*	Represents Boolean AND
+	Represents Boolean OR
{}	Represents Substitution (OUT1=A*B*C, OUT2=A*B*C*D or OUT2={OUT1}*D)

2.4.4.1 SELC Signal

The active high SELC signal is decoded from the latched address bits LA[31:28]. When these address bits equal C_{16} during the processor's address cycle, SELC is asserted indicating an access to the SAR memory region.

2.4.4.2 $\overline{\text{SELSQ0}}$ Signal

$\overline{\text{SELSQ0}}$ is the select signal (standing for Select Squall module, remembering that the SAR is on the squall module) which initiates the SAR Access State Machine when SELC is asserted.

2.4.4.3 SAR Access State Machine

Shown in Figure 10 is the SAR access state machine. It is initiated when the $\overline{\text{SELSQ0}}$ select signal from the FPGA is asserted, due to a request issued to the SAR's memory region.

The state of the control signals implemented in the controller, depend upon which state the state machine is in. The following are the signals that make up the SAR controller.

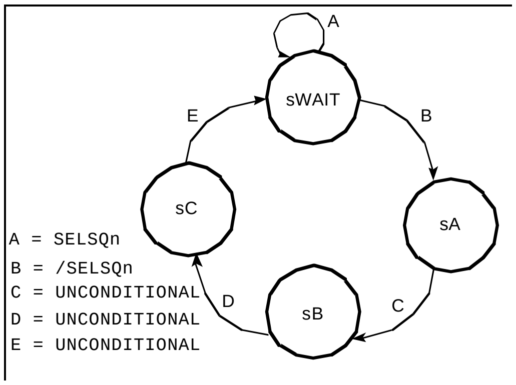


Figure 10. 80960Jx to SAR Access State Machine

2.4.4.4 $\overline{\text{READYn}}$ Signal

The $\overline{\text{READYn}}$ signal is asserted on the rising edge of the clock, at the end of state sB of the SAR access state machine. At the end of state sC, $\overline{\text{READYn}}$ signals to the processor the termination of a data transfer.

2.4.4.5 $\overline{\text{READYpinn}}$ Signal

The $\overline{\text{READYpinn}}$ signal is connected to the $\overline{\text{RDYRCV}}$ pin of the processor. It can be three-stated and is driven by $\overline{\text{READYn}}$. It must be three-stated when an access other than a SAR access occurs. The $\overline{\text{READYpinn}}$ signal is paired with the $\overline{\text{READYn}}$ signal, indicated by the brackets around $\overline{\text{READYn}}$ on the right side of the $\overline{\text{READYpinn}}$ equation. (see the MACHXL PLD file of Appendix B). Pairing connects a node signal like $\overline{\text{READYn}}$, to an output signal like $\overline{\text{READYpinn}}$.

2.4.4.6 $\overline{\text{RDYENn}}$ Signal

$\overline{\text{RDYENn}}$ controls the three-state buffer that three-states the $\overline{\text{READYpinn}}$ signal. It is asserted when $\overline{\text{SELSQ0}}$ is asserted.

2.4.4.7 $\overline{\text{SARSELn}}$ Signal

The $\overline{\text{SARSELn}}$ signal selects the SAR, enabling it to be accessed in slave mode.

2.4.4.8 $\overline{\text{SARASELn}}$ Signal

The $\overline{\text{SARASELn}}$ signal indicates to the SAR that an address is on its AD bus. On the first rising clock edge after $\overline{\text{SARASELn}}$ goes active, the SAR latches the address onto its AD bus for use during an access.

2.4.4.9 $\overline{\text{SARSWRn}}$ Signal

The $\overline{\text{SARSWRn}}$ signal determines the direction of the slave access. 1=read, 0=write.

2.4.4.10 $\overline{\text{AOEABn}}$ Signal

The $\overline{\text{AOEABn}}$ signal enables the B output of the SAR's address buffers, directing the flow of the latched address from the PCI-SDK, onto the AD bus of the SAR.

2.4.4.11 $\overline{\text{DOEABn}}$ Signal

The $\overline{\text{DOEABn}}$ signal enables the flow of data through the data buffers from A to B, to the AD bus of the SAR during a write.

2.4.4.12 $\overline{\text{DOEBAn}}$ Signal

The $\overline{\text{DOEBAn}}$ signal enables the flow of data through the data buffers from B to A, to the AD bus of the processor during a read.

3.0 DRAM CONTROLLER ACCESS FLOW

This section explains how the DRAM controller is sequenced while reading, writing and refreshing the DRAM. Examples discussed are:

- single word read access
- quad word read access
- single word write access
- quad word write access
- refresh

For the following accesses, refer to Figure 8. All accesses to the DRAM are with the i960 Jx processor configured with a 32 bit bus.

3.1 Single Word Read and Write Access

Figure 11 shows a single word read timing diagram. In state s0 of the read when an access has begun, the address bits from the processor are latched, and held latched throughout the access. The $\overline{\text{MUX}}$ signal stays deasserted, selecting the row address via the multiplexer. On the clock edge at the end of state s0, the DRAM control state machine transitions to state s2. In the beginning of state s2, the $\overline{\text{BLAST}}$ signal, $\overline{\text{MEMPEND}}$ bit, $\overline{\text{INCBANK}}$ signal, and all the $\overline{\text{RAS}}$ lines are asserted. $\overline{\text{BLAST}}$ asserting at this time indicates a single access, $\overline{\text{MEMPEND}}$ indicates an access is pending, $\overline{\text{INCBANK}}$ indicates the next state is a data state, while the $\overline{\text{RAS}}$ lines load the row address into all the DRAM's. In the middle of s2 the $\overline{\text{MUX}}$ signal is asserted on the falling edge of CLK1, selecting the column address.

At the end of state s2, the state machine unconditionally transitions to state s4. In s4, $\overline{\text{READY}}$ is asserted and the $\overline{\text{CAS}}$ lines for bank A are asserted, based on the byte enables. Asserting of the $\overline{\text{CAS}}$ lines enables data from the DRAM's onto the AD bus, making state s4 the data state (T_{d0}). On the clock edge at the end of state s4, data is captured by the processor due to sampling $\overline{\text{READY}}$ asserted. It is at this point that data put on the AD bus from the DRAM, must meet the set-up (t_{is1}) and hold (t_{ih1}) times of the processor. At the time $\overline{\text{READY}}$ is sampled low, $\overline{\text{BLAST}}$ is sampled low, causing the state machine to transition to state s0, completing the access and deasserting the $\overline{\text{MEMPEND}}$ bit.

Figure 11 shows a single word write timing diagram. This access sequences in the same manner as the single word read access did, up until the first data state.

In state s0 of the write when an access has begun, the address bits from the processor are latched, and held latched throughout the access. The $\overline{\text{MUX}}$ signal stays deasserted, selecting the row address via the multiplexer. On the clock edge at the end of state s0, the DRAM control state machine transitions to state s2, due to the processor access. In the beginning of state s2, the $\overline{\text{BLAST}}$ signal, $\overline{\text{MEMPEND}}$ bit, $\overline{\text{INCBANK}}$ signal, and all the $\overline{\text{RAS}}$ lines are asserted. $\overline{\text{BLAST}}$ asserting at this time indicates a single access, $\overline{\text{MEMPEND}}$ indicates an access is pending, $\overline{\text{INCBANK}}$ indicates the next state is a data state, while the $\overline{\text{RAS}}$ lines load the row address into all the DRAM's. Also during state s2, data is driven on the AD bus, where it waits to be written into the DRAM's. In the middle of s2 the $\overline{\text{MUX}}$ signal is asserted on the falling edge of CLK1, selecting the column address.

At the end of state s2, the state machine unconditionally transitions to state s4. In s4, $\overline{\text{READY}}$ is asserted and the $\overline{\text{CAS}}$ lines for bank A are asserted, based on the byte enables. Asserting the $\overline{\text{CAS}}$ lines writes data into the DRAM's from the AD bus, making state s4 the data state (T_{d0}). It is at this point that data on the AD bus must meet the set-up (t_{ds}) and hold (t_{dh}) times of the DRAM's. In this design, t_{dh} is most critical, and is met by holding the data on the AD bus until the end of state s4. On the clock edge at the end of state s4, $\overline{\text{READY}}$ and $\overline{\text{BLAST}}$ are sampled asserted, causing the state machine to transition back to state s0, completing the access and deasserting the $\overline{\text{MEMPEND}}$ bit.

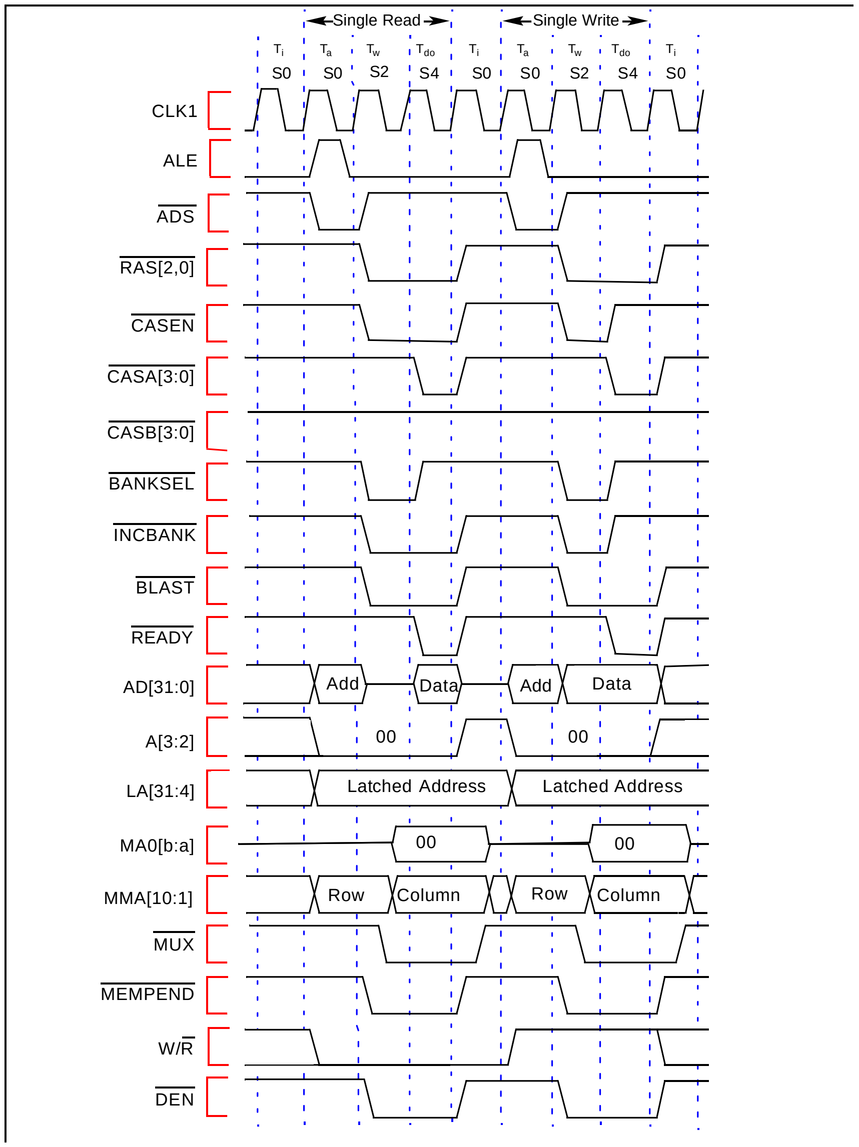


Figure 11. 80960Jx to DRAM 1 Wait State Single Word Read and Write Timing Diagram

3.2 Quad Word Read Access

Figure 12 shows a quad word read timing diagram. The quad word read access begins exactly the same way that the single word read access did, until the clock edge at the end of state s4. At this point, since a quad word read has 1-0-0-0 wait states, the state machine stays in s4, until a $\overline{\text{READY}}$ and $\overline{\text{BLAST}}$ are sampled asserted, on the rising clock edge at the end of state s4. As the state machine spins in s4, the toggling $\overline{\text{BANKSEL}}$ and the byte enables determine which $\overline{\text{CAS}}$ signals activate, thereby controlling which leaf of the DRAM data is transferred from. The toggling of $\overline{\text{BANKSEL}}$ causes $\overline{\text{CASA}}$ and $\overline{\text{CASB}}$ to activate alternately, with $\overline{\text{CASA}}$ enabling bank A first. As $\overline{\text{CASA}}$ and $\overline{\text{CASB}}$ alternate, bits MA0A and MA0B increment, providing two separate addresses, two for each bank. This is evident by observing Figure 12, which shows that bit “a” of MA0[b:a] toggles from 0 in data state T_{d0} to 1 in data state T_{d2}. This is also true with bit “b”, during data states T_{d1} and T_{d3}. During these data states the $\overline{\text{MUX}}$ signal stays asserted, only allowing the $\overline{\text{CAS}}$ address to be changed.

3.3 Quad Word Write Access

Figure 13 shows a quad word write timing diagram. The quad word write access begins exactly the same way that the single word write access did, until the clock edge at the end of state s4. At this point, since a quad word write has 1-1-1-1 wait states, the state machine alternates between states s4 and s3, with s4 being the data state ($\overline{\text{CAS}}$ signals asserted) and s3 being the wait state ($\overline{\text{CAS}}$ signals deasserted). The signals that determine which states are the data and wait states of the DRAM control state machine, are the $\overline{\text{CASEN}}$ and $\overline{\text{INCBANK}}$ signals. If at the end of s4, $\overline{\text{CASEN}}$ and $\overline{\text{INCBANK}}$ are deasserted, the $\overline{\text{CAS}}$ lines deassert and the incrementing address bits MA0[b:a] are incremented, preparing for the next data state. If at the end of s3 $\overline{\text{CASEN}}$ and $\overline{\text{INCBANK}}$ are asserted, the $\overline{\text{CAS}}$ lines are asserted and address bits MA0[b:a] are held constant until the end of the data state. The incrementing of bits MA0A and MA0B provide two separate addresses, two for each bank. This is evident by observing Figure 13, which shows that bit “a” of MA0[b:a] toggles from 0 in data state T_{d0} to 1 in data state T_{d2}. This is also true with bit “b”, during data states T_{d1} and T_{d3}.

If the $\overline{\text{READY}}$ and $\overline{\text{BLAST}}$ signals are sampled asserted on the rising clock edge at the end of state s4, the state machine transitions back to state s0, where the access is complete. During these data states the $\overline{\text{MUX}}$ signal stays asserted, only allowing the $\overline{\text{CAS}}$ address to be changed.

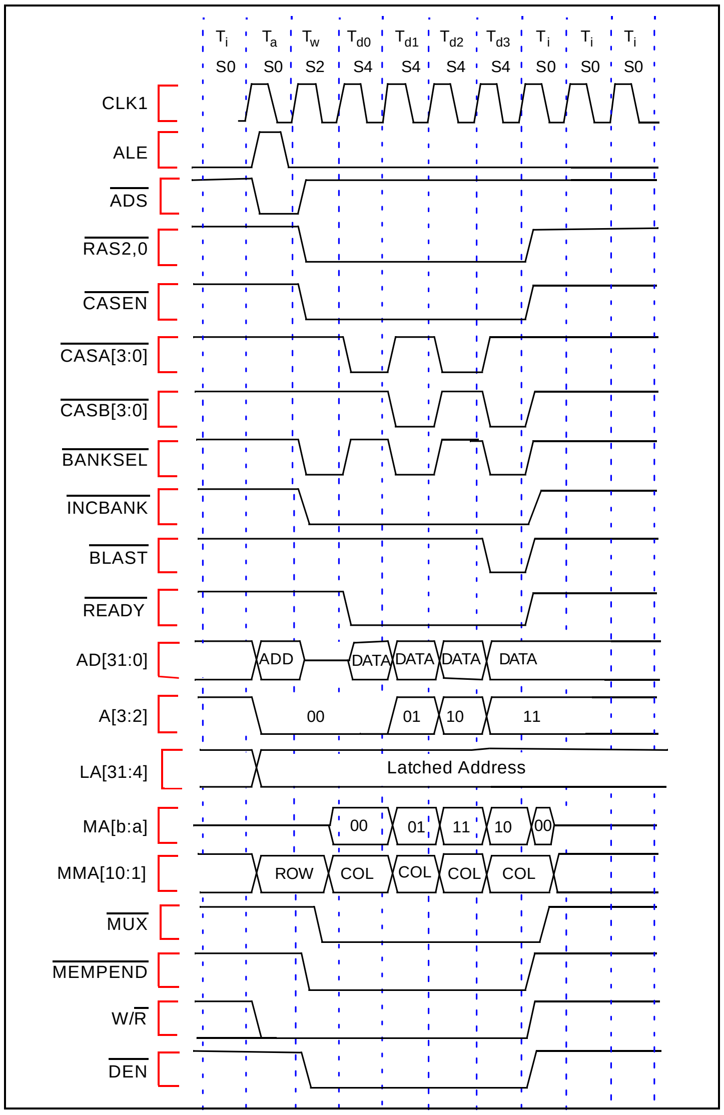


Figure 12. 80960Jx to DRAM 1,0,0,0 Quad Word Read Timing

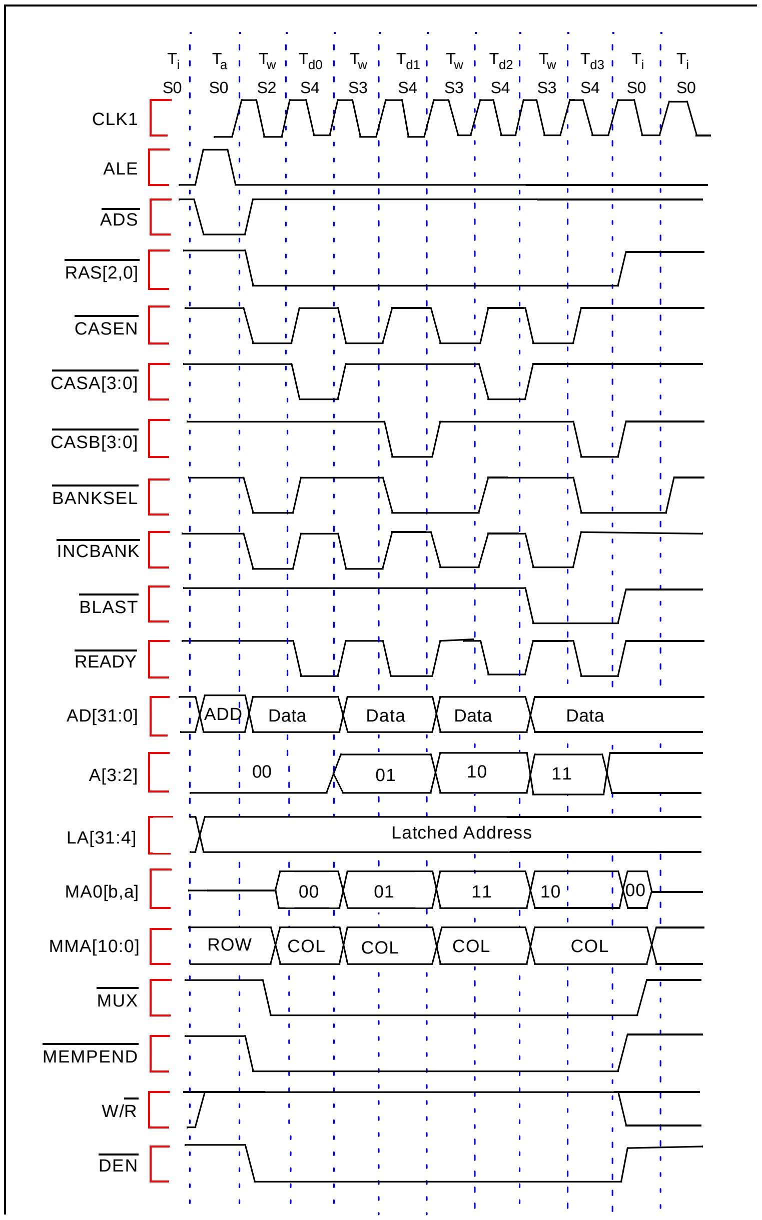


Figure 13. 80960Jx to DRAM 1,1,1,1 Quad Word Write Timing Diagram

3.4 CAS before RAS Refresh

Figure 14 shows a refresh cycle timing diagram. As soon as the PCI-SDK powers up, the refresh counter starts counting, and cannot be stopped by RESET. The counter is a free-running divide-by-64 counter, running on a 4 MHz clock. Its output switches high every 64 cycles of the 4 MHz clock, giving a 16 µs interval for requesting refresh.

On the next 25 MHz clock edge after the output of the counter switches high, the REFSYNC signal is asserted, causing the REFCLK and REFCLKD signals to sequence as shown in Figure 14. When REFCLK is sampled high and REFCLKD is sampled low on the clock edge, REFPEND is asserted. When REFPEND is detected low on the next clock edge, the DRAM control state machine enters state s6 where refresh begins. At the beginning of s6, REF is sserted and stays asserted through state s10. On the clock edge at the end of s6, REF indicates to the REFPEND state machine that the DRAM refresh is going to be satisfied, causing REFPEND to be deasserted.

The REF signal serves another purpose. It enables the CAS state machines during refresh, allowing the CAS signals to be driven. In states s8-s10, the RAS signals are asserted, giving a CAS before RAS refresh, since the CAS signals were asserted back in state s6.

After the state machine transitions out of state s10 completing the refresh, it passes through states s5 and s0, where the required RAS precharge takes place before another access can begin

.

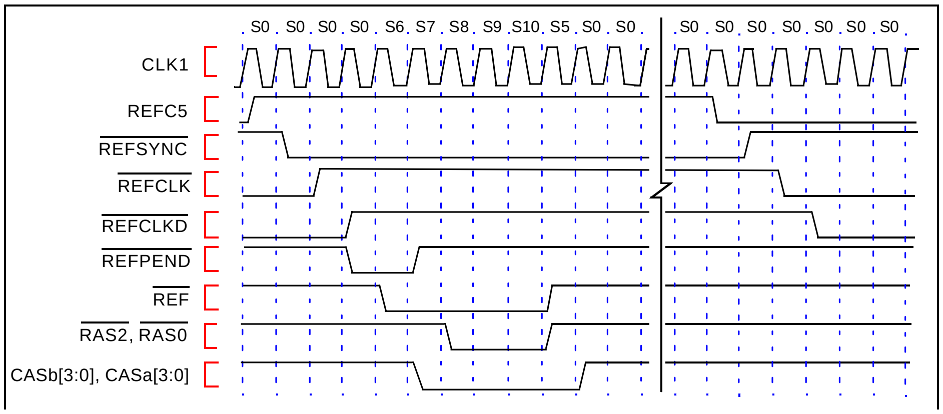


Figure 14. CAS-Before-RAS Refresh Cycle

4.0 SAR SLAVE ACCESS FLOW

This section describes how the SAR controller is sequenced during a single word read and write access.

All accesses to the SAR are with the i960 Jx processor configured with a 32 bit bus.

4.1 Single Word Read and Write Access

Figure 15 shows the Single Word Read Access Timing Diagram. The single word access begins with the processor making a request to the SAR address space. The address is latched and decoded, asserting the SELSQ0 select signal. On the first rising clock edge that SELSQ0 is low, the SAR access state machine initiates and unconditionally transitions from state sA through state sC, back into state sWAIT. The state machine will stay in sWAIT until SELSQ0 asserts, indicating another access.

At the end of state sWAIT when the state machine transitions to sA, the SAR controller drives SARSWRn high for a read, and asserts SARSELn, SARASELn, and AOEABn. As AOEABn asserts, the address buffers place the address on the SAR's AD bus, where it is latched into the SAR on the first rising edge of the clock after SARASELn goes low. On the following clock edge after the address is latched, the SAR places the data on its AD lines, which is at the beginning of state sC, provided that SARSELn is low. At the beginning of state sC with the data placed on the SAR's AD bus, READYpinn is asserted and DOEBAn activates, placing the data onto the processors AD bus, through the data buffers. At the end of state sC, the processor samples $\overline{\text{READY}}$ and $\overline{\text{BLAST}}$ asserted, signaling the end of the data transfer. When $\overline{\text{READY}}$ and $\overline{\text{BLAST}}$ are sampled asserted during the transition from sC to sWAIT, the $\overline{\text{SELSQ0}}$ signal deasserts, keeping the SAR access state machine in state sWAIT.

Figure 15 shows the Single Word Write Access Timing Diagram. The single word access begins with the processor making a request to the SAR address space. The address is latched and decoded, asserting the $\overline{\text{SELSQ0}}$ select signal. On the first rising clock edge that $\overline{\text{SELSQ0}}$ is low, the SAR access state machine initiates and unconditionally transitions from state sA through state sC, back into state sWAIT. The state machine will stay in sWAIT until $\overline{\text{SELSQ0}}$ asserts, indicating another access.

At the end of state sWAIT when the state machine transitions to sA, the SAR controller drives $\overline{\text{SARSWRn}}$ low for a write, and asserts $\overline{\text{SARSELn}}$, $\overline{\text{SARASELn}}$, and $\overline{\text{AOEABn}}$. As $\overline{\text{AOEABn}}$ asserts, the address buffers place the address on the SAR's AD bus, where it is latched into the SAR on the first rising edge of the clock after $\overline{\text{SARASELn}}$ goes low. On the same clock edge that the address is latched, the $\overline{\text{DOEABn}}$ signal is asserted, enabling data onto the SAR's AD bus, through the data buffers. The SAR then completes the access by latching the data present on its AD bus on the leading edge of the clock, just before $\overline{\text{SARSELn}}$ goes high. The clock edge in this case is at the end of state sB.

In the beginning of state sC after the access has completed, the controller asserts $\overline{\text{READYpinn}}$ and deasserts $\overline{\text{DOEABn}}$, removing the data from the SAR's AD bus. At the end of state sC, the processor samples $\overline{\text{READY}}$ and $\overline{\text{BLAST}}$ asserted, signaling the end of the data transfer. When $\overline{\text{READY}}$ and $\overline{\text{BLAST}}$ are sampled asserted during the transition from sC to sWAIT, the $\overline{\text{SELSQ0}}$ signal deasserts, keeping the SAR access state machine in state sWAIT.

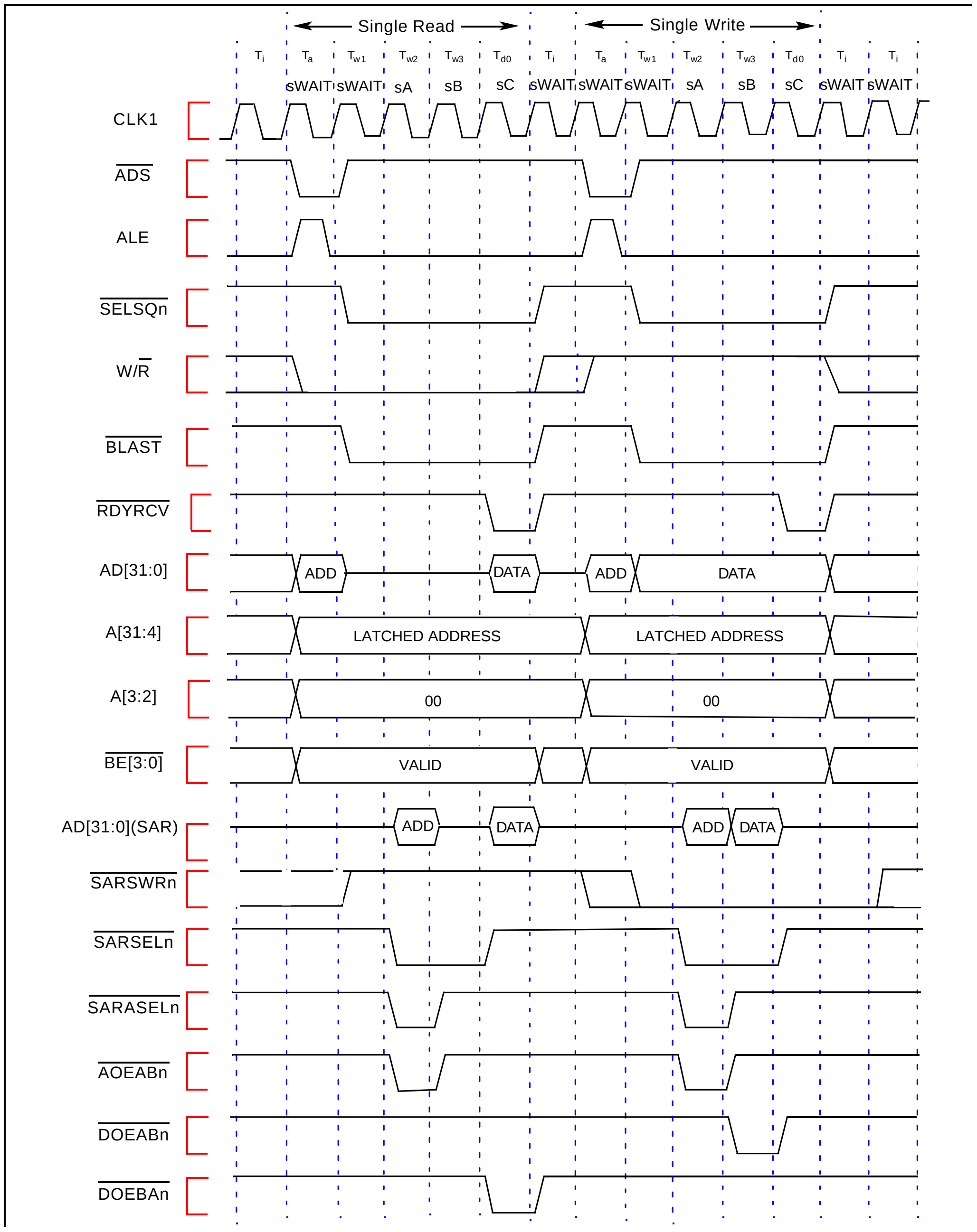


Figure 15. 80960Jx to µPD98401 Single Word Read and Write Timing Diagram

5.0 CONCLUSION

This application note discusses how to interface between an Intel i960 Jx processor and an NEC µPD98401 Local ATM SAR Chip. The platform used was a Cyclone Microsystems PCI-SDK evaluation platform, with a i960 Jx processor module and an ATM Squall module. With the Squall module containing the SAR chip. To support the interface between the i960 Jx processor and the SAR chip are 2, 8, or 32 Mbytes of DRAM, implemented in two 72 pin SIMM sockets. The schematics were generated in Future Net. The PLD equations for the DRAM controller were written in PLDshell, and the SAR controller equations were written in MACHXL. The schematics and PLD files are available through Intel America's Application Support BBS.

6.0 RELATED INFORMATION

To receive Intel literature, contact:

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

Table 1. Related Information

Reference #	Document Name	Order#
1	Intel *Solutions960®* catalog	Intel 270791
2	*i960® Jx Microprocessor Users's Manual*	Intel 272483
3	*80960JA/JF Embedded 32-bit Microprocessor Data Sheet*	Intel 272504
4	*µPD98401 LOCAL ATM SAR CHIP User's Manual*	NEC IEU-1384
5	*µPD98401 LOCAL ATM SAR CHIP Preliminary Data Sheet*	NEC ID-3392
6	*10 ns FLASHlogic FPGA with SRAM Option Data Sheet*	Altera
7	*PLDshell Plus /PLDasm User's Guide* V4.0	Intel
8	*Cyclone i960® Microprocessor User's Guide*	Intel 272577
9	*MACHXL Software Users's Guide*	AMD
10	*MACH 1 and 2 Family Data Book*	AMD

You can use your PC with modem to download schematics and PLD equations from Intel's Bulletin Board Service (BBS).

Intel Technical Support Hotline	North America: 800-628-8686 Europe: 44-793-696-000
Intel's Bulletin Board Service (BBS) for schematics and PLD equations	North America: 916-356-3600 supports up to 14.4 Kbps (n,8,1,p) Europe: 44-793-432-955

To contact Cyclone Microsystems for additional information about their products:

Cyclone Microsystems 25 Science Park New Haven CT 06511	Phone: 203-786-5536
	FAX: 203-786-5025
	e-mail: info@cyclone.com

APPENDIX A
PLDshell FILES

Table A-1 contains the PLD equations used to build the portion of the DRAM controller, implemented in the Altera FLASHlogic EPX780 FPGA. Table A-2 contains the PLD equations used to build the portion of the DRAM controller, implemented in the PAL20V8. The PLD equations were created in PLDshell.

Table A-1. DRAM Controller EPX780 PLDshell File (Sheet 1 of 12)

```
     TITLE          Private DRAM Control
     PATTERN        D601 - Compiled with PLDShell
     REVISION       E
     AUTHOR         J. Smith
     COMPANY        Cyclone Microsystems Inc.
     DATE           5-10-95

     CHIP      DCTRL          iFX780_84


     PIN      3   pclk           ;INP: system clock
     PIN     45   clk16          ;INP: 16MHz clock
     PIN     51   rst            ;INP: reset
     PIN     47   ads            ;INP:
     PIN     81   blast          ;INP:
     PIN     55   a31            ;INP:
     PIN     56   a30            ;INP:
     PIN     58   a29            ;INP:
     PIN     60   a28            ;INP:
     PIN     62   a19            ;INP:
     PIN     64   a18            ;INP:
     PIN     32   a3             ;INP: address to increment
     PIN     13   a2             ;INP: select bank cas
     PIN     20   a1             ;INP: select bank cas
     PIN     75   w_r            ;INP: shared w/r
     PIN     76   holda          ;INP:
     PIN     77   sqbr0          ;INP: Squall 0 bus request
     PIN     78   pcihold        ;INP: PCI 9060 bus request
     PIN     79   lock           ;INP: 960 lock
     PIN     82   extend         ;INP: squall module extend dram reads
     PIN      1   pt2            ;INP: i960 processor type Pulled hi on board
     PIN     35   pt1            ;INP: i960 processor type
     PIN     37   pt0            ;INP: i960 processor type
     PIN      8   freq2          ;INP: i960 processor freq
     PIN      7   freq1          ;INP: i960 processor freq
     PIN     28   freq0          ;INP: i960 processor freq
     PIN     74   pd3            ;INP: SIMM PD3 output 70/60ns
     PIN     73   swaprom        ;INP:
     PIN     21   pciinstall     ;INP: PCI9060 installed = 1
     PIN   IO29   readyin        ;INP: should be given same pin #
                                 ; as ready pin - this is for
                                 ; pin feedback
```

Table A-1. DRAM Controller EPX780 PLDshell File (Sheet 2 of 12)

```
      PIN   IO51   muxin          ;INP: should be given same pin #
                                  ; as mux pin.  muxalt when operating @ 33Mhz
      PIN     22   irquart        ;INP: active high uart interrupt
      PIN     83   breqo          ;INP: PCI9060 indicates deadlock

      PIN     84   boff        REG     ;OUT: BOFF* to Cx processors @ deadlock
      PIN     30   altREADY   REG     ;OUT: ready during deadlock
      PIN     54   irqddlk     REG     ;OUT: deadlock interrupt
      PIN      9   ras0        REG     ;OUT:
      PIN      5   ras2        REG     ;OUT:
      PIN     63   mux         REG     ;OUT: mux output - freq<33 , active low
      PIN     61   muxalt      REG     ;OUT: mux output - freq>=33,   "     "
      PIN     27   casen       REG     ;OUT:
      PIN     72   banksel     REG     ;OUT:
      PIN     70   ma0a        REG     ;OUT:
      PIN     69   ma0b        REG     ;OUT:
      PIN     31   dwe         REG     ;OUT: DRAM write enable
      PIN     12   ready       REG     ;out: i960 ready
      PIN      6   ref         REG     ;out:
      PIN     48   hold        REG     ;OUT: hold processor
      PIN     49   sqbg0       REG     ;OUT: bus grant Squall 0
      PIN     50   pcihlda     REG     ;OUT: bus grant to PLX 9060

      PIN     19   clk4        REG     ;OUT: 4MHz clock
      PIN     18   zrd         REG     ;OUT:
      PIN     16   zwr         REG     ;OUT:
      PIN     15   ior         REG     ;OUT:
      PIN     14   iow         REG     ;OUT:
      PIN     43   selio       REG     ;
      PIN     42   selcio      REG     ;OUT:
      PIN     41   seluart     REG     ;OUT:
      PIN     40   selpp       REG     ;OUT: select parallel port
      PIN     33   selpmrom    REG     ;OUT:
      PIN     34   selfrom0    REG     ;OUT:
      PIN     36   selfrom1    REG     ;OUT:
      PIN     39   selsq0      REG     ;OUT: Select Squall Module
      PIN     57   irquartn    COMB    ;OUT: active low uart interrupt

      NODE   IO46  altRDYEN    REG     ;buried: ready enable during deadlock
      NODE   IO28    rdyen     REG     ;buried: ready enable
      NODE   IO49    st0       REG
      NODE   IO44    st1       REG
      NODE   IO42    st2       REG
      NODE   IO31    bss0      REG     ;buried: bank select state 0
      NODE   IO05    incbank   REG     ;buried:
      NODE   IO07    mempend   REG     ;buried:
      NODE   IO52    sx                ;buried: Sx processor is accessing Mem.
      NODE   IO50    pfv1              ;buried:
      NODE   IO57    pfv0              ;buried:
```

Table A-1. DRAM Controller EPX780 PLDshell File (Sheet 3 of 12)

```
      NODE   IO54    selmux            ;buried:
      NODE   IO76    refsync   REG     ;buried:
      NODE   IO06    refclk    REG     ;buried:
      NODE   IO08    refclkd   REG     ;buried:
      NODE   IO02    refpend   REG     ;buried: refresh pending
      NODE   IO67    BOOTRD0   REG     ;buried: set at reset, clr w/ ADS
      NODE   IO65    BOOTRD1   REG     ;buried: set at ADS, clr w/ blast

      NODE   IO22    clk8      REG     ;buried: 8MHz clock
      NODE   IO10    ios0      REG     ;buried: io control states
      NODE   IO13    ios1      REG     ;buried: io control states
      NODE   IO16    ios2      REG     ;buried: io control states
      NODE   IO15    ios3      REG     ;buried: io control states
      NODE   IO19    ios4      REG     ;buried: io control states
      NODE   IO32    ciotrc    REG     ;buried:
      NODE   IO34    synctrc   REG     ;buried:
      NODE   IO36    count0    REG     ;buried: cio trcv states
      NODE   IO38    count1    REG     ;buried: cio trcv states
      NODE   IO79    refc0     REG     ;buried: refresh counter
      NODE   IO78    refc1     REG     ;buried: refresh counter
      NODE   IO74    refc2     REG     ;buried: refresh counter
      NODE   IO72    refc3     REG     ;buried: refresh counter
      NODE   IO71    refc4     REG     ;buried: refresh counter
      NODE   IO70    refc5     REG     ;buried: refresh counter
      NODE   IO12    t0        REG     ;buried: timeout states
      NODE   IO14    t1        REG     ;buried: timeout states
      NODE   IO55    d0        REG     ;buried: deadlock states
      NODE   IO58    d1        REG     ;buried: deadlock states

      STRING    sel4_7  '/a31 *  a30'                    ;regions 4-7
      STRING    sel8_9  ' a31 * /a30 * /a29'             ;regions 8-9
      STRING    sel0_3  '/a31 * /a30'                    ;region  0-3
      STRING    seldram ' a31 * /a30 *  a29 * /a28'      ;region  A
      STRING    selb    ' a31 * /a30 *  a29 *  a28'      ;region  B
      STRING    selc    ' a31 *  a30 * /a29 * /a28'      ;regions C
      STRING    seld    ' a31 *  a30 * /a29 *  a28'      ;regions D
      STRING    sele    ' a31 *  a30 *  a29 * /a28'      ;region  E
      STRING    self    ' a31 *  a30 *  a29 *  a28'      ;region  F

      STRING    pf0  '/pfv1 * /pfv0'
      STRING    pf1  '/pfv1 *  pfv0'
      STRING    pf2  ' pfv1 * /pfv0'
      STRING    pf3  ' pfv1 *  pfv0'

      STRING    Cx   '/pt2 *  pt1 * /pt0'
      STRING    Jx   ' pt2 * /pt1 * /pt0'
```

Table A-1. DRAM Controller EPX780 PLDshell File (Sheet 4 of 12)

```
      T_TAB (freq2 freq1 freq0 pd3 >> pfv1 pfv0 )
                 0     1     x   x  :    0    0   ; pf0=16 & 20MHz, 60 & 70ns
                 1     0     0   1  :    0    0   ;    =25MHz,      60ns
                                                  ;3111,3222
                 1     0     0   0  :    0    1   ; pf1=25MHz,      70ns
                 1     0     1   x  :    0    1   ;    =33MHz,      60 & 70ns
                 1     1     0   1  :    0    1   ;    =40MHz,      60ns
                                                  ;41111,42221
                 1     1     0   0  :    1    0   ; pf2=40MHz,      70ns
                 1     1     1   1  :    1    0   ;    =50MHz,      60ns
                                                  ;52221,42221
                 1     1     1   0  :    1    0   ; pf3=50MHz,      70ns
                                                  ;52222,42221

              ;*********** I/O Wait & ready generation ***
      STATE MEALY_MACHINE
      DEFAULT_BRANCH io0
      io0  = ios4 *  ios3 *  ios2 *  ios1 *  ios0
      io1  = ios4 *  ios3 *  ios2 *  ios1 * /ios0
      io2  = ios4 *  ios3 *  ios2 * /ios1 * /ios0
      io3  = ios4 *  ios3 *  ios2 * /ios1 *  ios0
      io4  = ios4 *  ios3 * /ios2 * /ios1 *  ios0
      io5  = ios4 *  ios3 * /ios2 * /ios1 * /ios0
      io6  = ios4 *  ios3 * /ios2 *  ios1 * /ios0
      io7  = ios4 *  ios3 * /ios2 *  ios1 *  ios0
      io8  = ios4 * /ios3 * /ios2 *  ios1 *  ios0
      io9  = ios4 * /ios3 * /ios2 * /ios1 *  ios0
      io10 = ios4 * /ios3 * /ios2 * /ios1 * /ios0
      io11 = ios4 * /ios3 * /ios2 *  ios1 * /ios0
      io12 = ios4 * /ios3 *  ios2 *  ios1 * /ios0
      io13 = ios4 * /ios3 *  ios2 *  ios1 *  ios0
      io14 =/ios4 * /ios3 *  ios2 *  ios1 *  ios0
      io15 =/ios4 *  ios3 *  ios2 *  ios1 *  ios0
      io16 =/ios4 *  ios3 *  ios2 *  ios1 * /ios0
      io17 =/ios4 *  ios3 *  ios2 * /ios1 * /ios0
      io18 =/ios4 *  ios3 *  ios2 * /ios1 *  ios0
      io19 =/ios4 *  ios3 * /ios2 * /ios1 *  ios0
      io20 =/ios4 *  ios3 * /ios2 *  ios1 *  ios0
      io21 =/ios4 * /ios3 * /ios2 *  ios1 *  ios0
      io22 =/ios4 *  ios3 * /ios2 *  ios1 * /ios0
      io23 =/ios4 * /ios3 *  ios2 *  ios1 * /ios0
      io24 =/ios4 * /ios3 *  ios2 * /ios1 *  ios0
      io32 =/ios4 * /ios3 * /ios2 *  ios1 * /ios0 ; used only on 1st Cx/Hx read.

      io0  := ( /selio * /selcio * ciotrc) -> io1
            + ( /selio  * selcio ) -> io15
            + ( /selpmrom ) ->io15
            + ( time3 ) ->io15
```

Table A-1. DRAM Controller EPX780 PLDshell File (Sheet 5 of 12)

```
io1  := vcc -> io2
io2  := vcc -> io3
io3  := vcc -> io4
io4  := vcc -> io5
io5  := vcc -> io6
io6  := vcc -> io7
io7  := vcc -> io8
io8  := vcc -> io9
io9  := vcc -> io10
io10 := vcc -> io11
io11 := vcc -> io12
io12 := vcc -> io13
io13 := vcc -> io14
io14 := vcc -> io15
io15 := vcc -> io16
io16 := vcc -> io17
io17 := vcc -> io18
io18 := vcc -> io19
io19 := vcc -> io20
io20 := vcc -> io21
io21 := ( BOOTRD1 ) -> io32    ; Cx first read, loop & hold ready
      + (/BOOTRD1 ) -> io22    ; low until Cx asserts blast.
io22 := ( blast ) -> io15      ; Sx/Kx burst read
      + (/blast ) -> io23
io23 := vcc -> io24
io24 := vcc -> io0
io32 := ( blast ) -> io32
      + (/blast ) -> io23

        ;*********** Local bus Timeout *************
        ;8us minimum timeout on squall cycles.

STATE MEALY_MACHINE
DEFAULT_BRANCH HOLD_STATE
time0     =  t1 *  t0
time1     =  t1 * /t0
time2     = /t1 * /t0
time3     = /t1 *  t0


time0     := (/selsq0)             -> time1
time1     := (/readyin)            -> time0
      + ( refclk * /refclkd) -> time2
time2     := (/readyin)            -> time0
      + ( refclk * /refclkd) -> time3
time3     := (/readyin)            -> time0
```

Table A-1. DRAM Controller EPX780 PLDshell File (Sheet 6 of 12)

```
        ;*********** DRAM Control ******************
STATE MEALY_MACHINE
DEFAULT_BRANCH s0
s0    =  ref *  ras0 *  ras2 * /st2 * /st1  *  st0
s1    =  ref * /ras0 * /ras2 * /st2 * /st1  *  st0
s2    =  ref * /ras0 * /ras2 * /st2 * /st1  * /st0
s3    =  ref * /ras0 * /ras2 * /st2 *  st1  * /st0
s4    =  ref * /ras0 * /ras2 * /st2 *  st1  *  st0
s5    =  ref *  ras0 *  ras2 * /st2 * /st1  * /st0
s5a   =  ref *  ras0 *  ras2 *  st2 * /st1  * /st0
s6    = /ref *  ras0 *  ras2 * /st2 * /st1  *  st0
s7    = /ref *  ras0 *  ras2 * /st2 * /st1  * /st0
s8    = /ref * /ras0 * /ras2 * /st2 * /st1  * /st0
s9    = /ref * /ras0 * /ras2 *  st2 * /st1  * /st0
s10   = /ref * /ras0 * /ras2 *  st2 * /st1  *  st0

s0   := (/refpend)                             -> s6
      + ( refpend * /ads * seldram */(pf0))    -> s1
      + ( refpend * /mempend       */(pf0))    -> s1
      + ( refpend * /ads * seldram * (pf0))    -> s2
      + ( refpend * /mempend       * (pf0))    -> s2
s1   := vcc     -> s2
s2   := (pf2 * /w_r)     -> s3
      + (pf2 *  w_r)     -> s4
      + (pf3 * /w_r)     -> s3
      + (pf3 *  w_r)     -> s4
      + (pf1)            -> s4
      + (pf0)            -> s4
s3   := vcc     -> s4
s4   := ( blast * /w_r * extend * (pf0))      -> s4   ;AP ok
      + ( blast * /w_r * extend * pf1)        -> s4
      + ( blast * /w_r * extend * pf2)        -> s3
      + ( blast * /w_r * extend * pf3)        -> s3
      + ( blast *  w_r )                      -> s3   ;AP ok
      + (/blast           * extend * (pf0))  -> s0   ;AP ok
      + (/blast           * extend * pf1)     -> s5
      + (/blast           * extend * pf2)     -> s5
      + (/blast *  w_r      * pf3)            -> s5a
      + (/blast * /w_r * extend * pf3)        -> s5
      + (/extend * /w_r )                     -> s4
s5a  := vcc     -> s5
s5   := vcc     -> s0
s6   := vcc     -> s7       ;refresh
s7   := vcc     -> s8       ;refresh
s8   := vcc     -> s9       ;refresh
s9   := vcc     -> s10      ;refresh
s10  := vcc     -> s5       ;refresh
```

Table A-1. DRAM Controller EPX780 PLDshell File (Sheet 7 of 12)

```
        ;******* Deadlock State Machine ************
STATE MEALY_MACHINE
DEFAULT_BRANCH sIDLE

;state assignments
 sIDLE =  irqddlk * /d1 * /d0
    sB = /irqddlk * /d1 * /d0
    sC = /irqddlk * /d1 *  d0
    sD = /irqddlk *  d1 * /d0
    sE =  irqddlk *  d1 * /d0
    sF =  irqddlk *  d1 *  d0

;state transitions
 sIDLE := (/breqo ) -> sIDLE              ; waiting for BREQo=1
        + ( breqo * Jx ) -> sB            ; start only if Jx processor
    sB :=       vcc -> sC                 ; \
    sC :=       vcc -> sD                 ;  > IRQDDLK active for 3 clks
    sD :=       vcc -> sE                 ; /
    sE := ( blast ) -> sE                 ; set READY=0, wait for BLAST=0
        + (/blast ) -> sF
    sF := ( breqo ) -> sF                 ; waiting for BREQo=0 before
        + (/breqo ) -> sIDLE              ; rearming state machine

EQUATIONS  ;************ FF Control *******************
ras0.clkf = pclk   ras2.clkf = pclk
casen.clkf=pclk    incbank.clkf= pclk
banksel.clkf=pclk  ma0a.clkf=pclk     ma0b.clkf=pclk
dwe.clkf=pclk      ready.clkf=pclk    hold.clkf=pclk
sqbg0.clkf=pclk    pcihlda.clkf=pclk  rdyen.clkf=pclk st0.clkf=pclk
bss0.clkf=pclk     mempend.clkf=pclk  st1.clkf = pclk st2.clkf = pclk
rdyen.clkf = pclk

ios4.clkf = pclk  ios3.clkf = pclk   ios2.clkf = pclk
ios1.clkf = pclk  ios0.clkf = pclk
zrd.clkf  = pclk  zwr.clkf  = pclk
ior.clkf  = pclk  iow.clkf  = pclk
t1.clkf   = pclk  t0.clkf   = pclk
selio.clkf = pclk selcio.clkf = pclk
seluart.clkf=pclk selpmrom.clkf=pclk selfrom0.clkf=pclk
selpp.clkf=pclk   selsq0.clkf=pclk   selfrom1.clkf=pclk

st0.setf  = /rst  st1.rstf  = /rst   st2.rstf  = /rst
ras0.setf = /rst  ras2.setf = /rst   ref.setf  = /rst

ios4.setf = /rst  ios3.setf = /rst   ios2.setf = /rst
ios1.setf = /rst  ios0.setf = /rst
t1.setf   = /rst  t0.setf   = /rst

irqddlk.clkf = pclk         boff.clkf = pclk
irqddlk.setf = /rst         boff.setf = /rst
```

Table A-1. DRAM Controller EPX780 PLDshell File (Sheet 8 of 12)

```
     d0.clkf = pclk              d1.clkf = pclk
     d0.rstf = /rst              d1.rstf = /rst

     altREADY.clkf = pclk        altRDYEN.clkf = pclk
     altREADY.setf = /rst        altRDYEN.rstf = /rst

      BOOTRD0.clkf = pclk
      BOOTRD1.clkf = pclk

             ;************ Deadlock Equations ***********
    /altREADY := sD
               + sE * blast
     altREADY.trst = altRDYEN
     altRDYEN := sC + sD + sE
        /boff := Cx *  breqo
               + Cx * /breqo * /boff * pcihold

             ;************ Select Signals ***************
     /selio    := /ads * selb                                  ;Bxxxxxxx
                + /ads * self * /swaprom                       ;Fxxxxxxx
                + /ads * sele *  swaprom                       ;Exxxxxxx
                + /ads * seld                                  ;Dxxxxxxx
                + /ads * sel4_7 * /pciinstall
                + /ads * sel8_9 * /pciinstall
                + /selio * /(/blast * /ready) * rst

     /selcio := /ads   * selb */a19 *  a18                       ;b4000000
              + /selio * selb */a19 *  a18 * ready
     /seluart:= /ads   * selb */a19 * /a18                       ;b0000000
              + /selio * selb */a19 * /a18 * ready
     /selpp  := /ads   * selb * a19 * /a18                       ;b8000000
              + /selio * selb * a19 * /a18 * ready

     /selpmrom:= swaprom * /ads      * self           * rst ;f0000000
               + swaprom * /selpmrom * self   * ready * rst
               + swaprom * /ads      * sel0_3         * rst ;00000000
               + swaprom * /selpmrom * sel0_3 * ready * rst
               +/swaprom * /ads      * sele           * rst ;e0000000
               +/selpmrom */(/blast * /ready)         * rst

     /selfrom0:= swaprom * /ads   * sele   */a18            ;e0000000
               + swaprom * /selio * sele   */a18 * ready
               +/swaprom * /ads   * self   */a18            ;f0000000
               +/swaprom * /selio * self   */a18 * ready
               +/swaprom * /ads   * sel0_3 */a18            ;00000000
               +/swaprom * /selio * sel0_3 */a18 * ready
```

Table A-1. DRAM Controller EPX780 PLDshell File (Sheet 9 of 12)

```
/selfrom1:= swaprom * /ads   * sele   * a18             ;e0040000
          + swaprom * /selio * sele   * a18 * ready
          +/swaprom * /ads   * self   * a18             ;f0040000
          +/swaprom * /selio * self   * a18 * ready
          +/swaprom * /ads   * sel0_3 * a18             ;00040000
          +/swaprom * /selio * sel0_3 * a18 * ready
          + /selfrom1 * BOOTRD1 * blast * rst

/selsq0  := /ads * selc * rst                           ;c0000000
          + /selsq0 * /(/blast * /readyin) * rst

BOOTRD0 := /rst                    * /pt2 * pt1       ; set if Cx or Hx
         +  rst * BOOTRD0 *  ads   * /pt2 * pt1       ; hold til 1st ads

BOOTRD1 :=  rst * BOOTRD0 * /ads   * /pt2 * pt1       ; set at ads
         +  rst * BOOTRD1 *  blast * /pt2 * pt1       ; hold til 1st blast

            ;************I/O control equatons *********
   /zrd := /rst
         + /selcio * /w_r * io2 * rst
         + /selcio * /w_r * io3 * rst
         + /selcio * /w_r * /zrd * ready * rst

   /zwr := /rst
         + /selcio *  w_r * io2 * rst
         + /selcio *  w_r * io3 * rst
         + /selcio *  w_r * /zwr * ready * rdyen * rst

   /ior := /selio * selcio * /w_r * ready * rst
         + /selpmrom       * /w_r * ready * rst
         + /selio * selcio * /w_r * /ready * BOOTRD1 * blast * rst
         + /selpmrom       * /w_r * /ready * BOOTRD1 * blast * rst

   /iow := /selio * selcio *  w_r * ready * rdyen * rst
         + /selpmrom       *  w_r * ready * rdyen * rst

/irquartn= irquart
             ;************ Configuration Terms **********
 sx      = /pt2 * /pt1 * /pt0 * /holda                   ;i960Sx
 selmux  =  freq2 */freq1 *  freq0                     ; 33MHz
         +  freq2 * freq1 * /freq0                     ; 40MHz
         +  freq2 * freq1 *  freq0                     ; 50MHz
            ;************DRAM Control *****************
   /mempend  :=  mempend * /ads * seldram * rst
                + /mempend * /(/blast * /ready) * rst ;hold till rdy

 mempend.setf = /rst
/mux         := /ras0 * rst
/muxalt      := /ras0 * rst
 mux.clkf     = /pclk
```

Table A-1. DRAM Controller EPX780 PLDshell File (Sheet 10 of 12)

```
       mux.trst     = /selmux     ;MUX tri-stated when operating @ >=33MHz
       muxalt.clkf  =  pclk
       muxalt.trst  =  selmux

      /casen       := s0 * /ads * seldram * (pf0) * refpend
                    + s0 * /mempend * (pf0)  * refpend
                    + s1
                    + s2 * /w_r * pf2
                    + s2 * /w_r * pf3
                    + s2 * ((pf0) + (pf1)) * /w_r
                    + s3 * /w_r
                    + s4 * blast
                    + s4 * /extend * /w_r * /blast


         /incbank  := s0 * /ads * seldram * (pf0) * refpend * extend ; JX ok
                    + s0 * /mempend * (pf0) * refpend * extend       ; JX ok
                    + s1 * /((pf2) + (pf3)) * /w_r * blast * extend ; JX ok
                    + s1 *  w_r * blast * extend                    ;   “
                    + s2 * /w_r * blast * extend                    ;   “
                    + s3 * /((pf2) + (pf3)) * /w_r * blast * extend ; 1st term
                                                                    ; =1 w/
                                                                    ; pf0=1
                    + s4 * blast      * extend                      ;   “

        /ready  := s2 * (pf0)
                 + s2 * pf1
                 + s2 * pf2 * w_r
                 + s2 * pf3 * w_r
                 + s3
                 + s4 * (pf0) * /w_r * blast
                 + s4 * pf1 * /w_r * blast
                 + s4 * /w_r * /blast * /extend
                 + io21
                 + io32

       ready.trst    = /rdyen

      /rdyen    := s1 + s2 + s3 + s4
                 + io20 + io21 + io22 + io32

      /dwe   :=  w_r * ref

       ma0a.d :=  ras0 * /a3 *  a2   ; a[3:2] = 01
               +  ras0 *  a3 * /a2   ;   “    = 10
               + /ras0 * /ma0b *  banksel * bss0 * /ready
               + /ras0 *  ma0a * /banksel
               + /ras0 *  ma0a * /bss0
               + /ras0 *  ma0a *  ready
```

Table A-1. DRAM Controller EPX780 PLDshell File (Sheet 11 of 12)

```
      ma0b.d :=  ras0 *  a3 * /a2   ; a[3:2] = 10
              +  ras0 *  a3 *  a2        “   = 11
              + /ras0 *  ma0a * /banksel * bss0 * /ready
              + /ras0 *  ma0b *  banksel
              + /ras0 *  ma0b * /bss0
              + /ras0 *  ma0b *  ready

     ma0a.rstf  = /rst      ma0b.rstf  = /rst
     ma0a.trst  = /muxin    ;output enable active with MUX or MUXALT low
     ma0b.trst  = /muxin    ;muxin low when MUX or MUXALT is low

             ;************ DRAM Bank Select *************

  /banksel.d:= /ads *  mempend * /sx * /a2 * rst                  ; JX ok
             + /ads *  mempend *  sx * /a2 * /a1 * rst
             + /ads *  mempend *  sx * /a2 *  a1 * rst
             + /mempend *  incbank * /banksel * rst               ;hold JX ok
             + /mempend * /incbank * /banksel * rst * /extend     ;hold
             + /mempend * /incbank * /banksel * /bss0 * rst * extend
                                                                  ;bs0->bs1
             + /mempend * /incbank *  banksel *  bss0 * rst * extend ; JX ok
                                    ; /\                          ;bs3->bs0
                                      ||
                                  ; oscillates with incbank low
;bs3->bs1 /sx

;bs3->bs1 /sx
  /bss0.d   := /ads * sx * /a2 * /a1 * rst
             + /ads * sx *  a2 * /a1 * rst
             +  ads * sx *  incbank * /bss0 * rst                 ;hold
             +  ads * sx * /incbank * /bss0 * rst * /extend       ;hold
             +  ads * sx * /incbank * /banksel * bss0 * rst * extend
                                                                  ;bs1 ->bs2
             +  ads * sx * /incbank *  banksel * bss0 * rst * extend
                                                                  ;bs3 ->bs0
     banksel.setf = /rst   bss0.setf = /rst

             ;************ DRAM Refresh *****************
     /refsync  := refc5                  ;sync 4MHz count w/ pclock
     /refclk   := refsync
     /refclkd  := /refclk
     /refpend  :=  refclk * /refclkd * rst   ;set
               + /refpend *  ref * rst      ;reset with ref

     refc0.t   := vcc                                   ;refresh counter 16us
     refc1.t   := refc0
     refc2.t   := refc0 * refc1
     refc3.t   := refc0 * refc1 * refc2
     refc4.t   := refc0 * refc1 * refc2 * refc3
```

Table A-1. DRAM Controller EPX780 PLDshell File (Sheet 12 of 12)

```
     refc5.t   := refc0 * refc1 * refc2 * refc3 * refc4
     refc0.aclk = clk4  refc1.aclk = clk4  refc2.aclk = clk4
     refc3.aclk = clk4  refc4.aclk = clk4  refc5.aclk = clk4
     refsync.clkf=pclk  refclk.clkf= pclk  refclkd.clkf=pclk
     refpend.clkf=pclk  ref.clkf   = pclk

             ;************ Arbitration ******************
      hold            := pcihold * /holda * lock  ;hold may not be asserted
               + pcihold *  holda                 ;if lock is active, but
               + /sqbr0  * /holda * lock          ;lock maybe asserted by
               + /sqbr0  *  holda                 ;SQUALL masters once they
                                                  ;are granted the bus.
     /sqbg0           := holda * hold * /sqbr0 * /pcihlda
               + /sqbg0 * /sqbr0 * holda

      pcihlda := holda * hold * pcihold * sqbr0 * sqbg0
               + pcihlda * pcihold * holda
               + pcihold * /boff

             ;************ 4 MHz clock ******************
      clk8.t        := vcc
      clk8.clkf     = clk16
      clk4.t        := vcc
      clk4.aclk     = clk8
             ;************ CIO T recovery ***************
             ;Delay 1000ns for trc on the 8536
     /ciotrc        := /selcio * /ready    * rst       ;set
               + /ciotrc *   synctrc * rst             ;hold/reset

     /synctrc       := /ciotrc *  /count1 *       count0 * rst ;assert

      ciotrc.clkf   = pclk
      synctrc.clkf  = pclk

     /count1        := /ciotrc *  count1 * /count0 * rst ;10 -> 00
                     + /ciotrc * /count1 * /count0 * rst ;00 -> 01

     /count0        := /ciotrc *  count1 *  count0 * rst ;11 -> 10
                     + /ciotrc *  count1 * /count0 * rst ;10 -> 00
      count1.aclk   = clk4
      count0.aclk   = clk4
      count1.setf   = /synctrc *  ciotrc
      count0.setf   = /synctrc *  ciotrc
```

Table A-2. DRAM Controller PAL20V8 PLDshell File (Sheet 1 of 2)

```
TITLE          Private DRAM Control
PATTERN
REVISION       D600A
AUTHOR         J. Smith
COMPANY        Cyclone
DATE           06-14-94
CHIP           CASEN   PAL20V8

PIN     1      pclk           ;INP:       system clock
PIN     2      ready          ;INP:      ;plcc pin 3
PIN     3      casen          ;INP:      ;plcc pin 4
PIN     4      banksel        ;INP:     ;plcc pin 5
PIN     5      blast          ;INP:      ;plcc pin 6
PIN     6      be3            ;INP:      ;plcc pin 7
PIN     7      be2            ;INP:      ;plcc pin 9
PIN     8      be1            ;INP:      ;plcc pin 10
PIN     9      be0            ;INP:      ;plcc pin 11
PIN    10      ref            ;INP:      ;plcc pin 12
PIN    11      nc             ;INP:      ;plcc pin 13
PIN    12      gnd                       ;plcc pin 14
PIN    13      oe                        ;plcc pin 16
PIN    14      nc             ;OUT:      ;plcc pin 17
PIN    15      casb0     REG  ;OUT:      ;plcc pin 18
PIN    16      casb1     REG  ;OUT:      ;plcc pin 19
PIN    17      casb2     REG  ;OUT:      ;plcc pin 20
PIN    18      casb3     REG  ;OUT:      ;plcc pin 21
PIN    19      casa0     REG  ;OUT:      ;plcc pin 23
PIN    20      casa1     REG  ;OUT:      ;plcc pin 24
PIN    21      casa2     REG  ;OUT:      ;plcc pin 25
PIN    22      casa3     REG  ;OUT:      ;plcc pin 26
PIN    23      nc                        ;plcc pin 27
PIN    24      vcc
```

Table A-2. DRAM Controller PAL20V8 PLDshell File (Sheet 2 of 2)

```
        EQUATIONS

        /casa0  := /casen  * /be0 *  /banksel * /(/blast * /ready)
                 + /ref
        /casa1  := /casen  * /be1 *  /banksel * /(/blast * /ready)
                 + /ref
        /casa2  := /casen  * /be2 *  /banksel * /(/blast * /ready)
                 + /ref
        /casa3  := /casen  * /be3 *  /banksel * /(/blast * /ready)
                 + /ref

        /casb0  := /casen  * /be0 *   banksel * /(/blast * /ready)
                 + /ref
        /casb1  := /casen  * /be1 *   banksel * /(/blast * /ready)
                 + /ref
        /casb2  := /casen  * /be2 *   banksel * /(/blast * /ready)
                 + /ref
        /casb3  := /casen  * /be3 *   banksel * /(/blast * /ready)
                 + /ref
```

APPENDIX B
PLD EQUATIONS

Table B-1 contains the PLD equations used to build the SAR controller, implemented in an AMD* MACH210* PLD. The PLD equations were created in MACHXL.

Table B-1. SAR Controller MACH210, MACHXL File (Sheet 1 of 6)

```
     TITLE   NEC-SAR (uPD98401) Control
   PATTERN   H601
  REVISION   A
    AUTHOR   Joe Niedermeyer
   COMPANY   Cyclone
      DATE   6-06-95

      CHIP   H601  MACH210  ;(-12nS tPD req'd for SARRDYn)


  PIN   35  PCLK            ;INP: system clock.
  PIN    7  RESETn          ;INP: system reset.
  PIN   19  SELSQn          ;INP: squall region decode.
  PIN   14  SARATTNn        ;INP: SAR bus request.
  PIN    3  SQBGNTn         ;INP: squall bus grant.
  PIN   13  SIZE2           ;INP: SAR burst length MSB.
  PIN   11  SIZE1           ;INP: SAR burst length bit.
  PIN   10  SIZE0           ;INP: SAR burst length LSB.
  PIN    5  SARDWRn         ;INP: SAR dr/w* (master rd/wr*).
  PIN   33  PHYCSn          ;INP: FRAMER chip select.
  PIN   32  PHYWRn          ;INP: FRAMER read/write*.

  PIN    2  READYpinn  REG  ;OUT: i960 ready*.
  PIN    6  SARASELn   REG  ;OUT: SAR asel*.
  PIN   41  SARSELn    REG  ;OUT: SAR sel*.
  PIN   36  SARSWRn    REG  ;OUT: SAR sr/w* (slave rd/wr*).
  PIN   25  SQBREQn    REG  ;OUT: squall bus request.
  PIN   30  SARGNTn    REG  ;OUT: SAR bus grant.
  PIN    4  BLASTpinn  REG  ;OUT: blast* when SAR is bus master.
  PIN   16  ADSpinn    REG  ;OUT: ads* when SAR is bus master.
  PIN   21  CPUWRpin   REG  ;OUT: w/r* when SAR is bus master.
  PIN   24  SARRDYn    COMB ;OUT: SAR rdy*.
  PIN   18  BE0n       COMB ;OUT: be0* when SAR is bus master.
  PIN   20  BE1n       COMB ;OUT: be1* when SAR is bus master.
  PIN   28  BE2n       COMB ;OUT: be2* when SAR is bus master.
  PIN   38  BE3n       COMB ;OUT: be3* when SAR is bus master.
  PIN   26  AOEBAn     REG  ;OUT: ADDR XCVR control, master direction.
  PIN   40  ACLKENn    REG  ;OUT: ADDR XCVR clock enable, master direction.
  PIN   42  AOEABn     REG  ;OUT: ADDR XCVR control, slave direction.
  PIN   43  DOEABn     REG  ;OUT: DATA XCVR control, slave write direction.
  PIN   17  DOEBAn     REG  ;OUT: DATA XCVR control, slave read direction.
  PIN   15  DCLKENn    COMB ;OUT: DATA XCVR clock enable, master read dir.
  PIN   37  DLEABn     REG  ;OUT: DATA XCVR latch enable, master write dir.
  PIN   39  PHYALE     REG  ;OUT: FRAMER address latch enable.
```

Table B-1. SAR Controller MACH210, MACHXL File (Sheet 2 of 6)

```
    NODE  2  READYn     REG  PAIR  READYpinn
    NODE 28  ADSn       REG  PAIR  ADSpinn
    NODE  6  BLASTn     REG  PAIR  BLASTpinn
    NODE 18  CPUWR      REG  PAIR  CPUWRpin

    NODE 23  RDYENn     REG  ;BRY: ready pin enable.
    NODE 11  CNT3       REG  ;BRY: burst count MSB.
    NODE  7  CNT2       REG  ;BRY: burst count bit.
    NODE  4  CNT1       REG  ;BRY: burst count bit.
    NODE 15  CNT0       REG  ;BRY: burst count LSB.
    NODE  3  SARRDY0    REG  ;BRY: ready*, one clock delayed.

    NODE  14,39  slv[1..0]  REG  ;BRY: slave states.
    NODE  35,22  mst[1..0]  REG  ;BRY: master states.

    ;**********************************;
    ;***  Slave State Assignments  ***;
    ;**********************************;

    STRING   WAIT  ' #b11 '
    STRING      A  ' #b01 '
    STRING      B  ' #b00 '
    STRING      C  ' #b10 '

    STRING  sWAIT  '  slv[1] *  slv[0] '
    STRING     sA  ' /slv[1] *  slv[0] '
    STRING     sB  ' /slv[1] * /slv[0] '
    STRING     sC  '  slv[1] * /slv[0] '

    ;***********************************;
    ;***  Master State Assignments  ***;
    ;***********************************;

    STRING     M0  ' #b11 '
    STRING     M1  ' #b01 '
    STRING     M2  ' #b00 '
    STRING     M3  ' #b10 '

    STRING    sM0  '  mst[1] *  mst[0] '
    STRING    sM1  ' /mst[1] *  mst[0] '
    STRING    sM2  ' /mst[1] * /mst[0] '
    STRING    sM3  '  mst[1] * /mst[0] '

    GROUP  ff  slv[1..0]  mst[1..0]  READYn  RDYENn
               SARASELn  SARSELn  SARSWRn  SQBREQn SARGNTn
               AOEABn AOEBAn ACLKENn DOEABn DOEBAn DLEABn
               ADSn CPUWR SARRDY0 BLASTn PHYALE
               CNT3 CNT2 CNT1 CNT0

    EQUATIONS

       ff.CLKF  = PCLK
       ff.RSTF  = GND
       ff.SETF  = GND
```

Table B-1. SAR Controller MACH210, MACHXL File (Sheet 3 of 6)

```
;**********************************;
;***  Slave State Transitions   ***;
;**********************************;

IF (/RESETn)
THEN  BEGIN  slv[1..0] = WAIT  END
ELSE  BEGIN  CASE  (slv[1..0])
      BEGIN
      WAIT:  BEGIN IF  (SELSQn)
                    THEN BEGIN  slv[1..0] = WAIT  END
                    ELSE BEGIN  slv[1..0] = A     END
             END
         A:  BEGIN  slv[1..0] = B     END
         B:  BEGIN  slv[1..0] = C     END
         C:  BEGIN  slv[1..0] = WAIT  END
      END
      END

;***************************;
;***  Ready Generation  ***;
;***************************;

    READYn := /( sB )

 READYpinn := { READYn }

 READYpinn.TRST = /RDYENn

   /RDYENn := /SELSQn

;******************************;
;***  SAR Slave Equations  ***;
;******************************;

 /SARASELn := sWAIT * /SELSQn

  /SARSELn := sWAIT * /SELSQn
            + sA

  /SARSWRn := sWAIT * /SELSQn * CPUWRpin
```

Table B-1. SAR Controller MACH210, MACHXL File (Sheet 4 of 6)

```
;****************************************;
;***  Address & Data Buffer Control  ***;
;****************************************;

   /AOEABn := sWAIT * /SELSQn                     ; SAR Slave direction

   /AOEBAn := sM1 + sM2 + sM3                     ; SAR Master direction

  /ACLKENn := sM1                                 ; SAR Master direction

   /DOEABn :=  sA *  CPUWRpin                     ; SAR Slave Write
            + sM3 *  SARDWRn                      ; SAR Master Read

   /DOEBAn :=  sB * /CPUWRpin                     ; SAR Slave Read
            + sM3 * /SARDWRn                      ; SAR Master Write

  /DCLKENn  = /READYpinn * /SQBGNTn * SARDWRn     ; SAR Master Read Only

   /DLEABn := sM3 *  SARDWRn                      ; SAR Master Read Only

;**********************************;
;***  Master State Transitions  ***;
;**********************************;

IF (/RESETn)
THEN  BEGIN  mst[1..0] = M0  END
ELSE  BEGIN  CASE  (mst[1..0])
      BEGIN
        M0:  BEGIN IF  (/SQBGNTn * /SARATTNn)
                    THEN BEGIN  mst[1..0] = M1  END
                    ELSE BEGIN  mst[1..0] = M0  END
             END
        M1:  BEGIN  mst[1..0] = M2    END
        M2:  BEGIN  mst[1..0] = M3    END
        M3:  BEGIN IF  (/BLASTn * /READYpinn)
                    THEN BEGIN  mst[1..0] = M0  END
                    ELSE BEGIN  mst[1..0] = M3  END
             END
      END
      END

;*******************************;
;***  Arbitration Equations  ***;
;*******************************;

  /SQBREQn := /SARATTNn * RESETn

  /SARGNTn := /SQBGNTn * /SQBREQn * /SARATTNn * RESETn
```

Table B-1. SAR Controller MACH210, MACHXL File (Sheet 5 of 6)

```
;**************************;
;***  Blast Generation  ***;
;**************************;

; Burst counter needed to generate BLASTn.
; SAR indicates burst length (encoded) on SIZE<2,1,0> pins.
; Counter loaded with (SIZE - 1) at ADS*.
; Counter decrements with each READY*.
; BLASTn asserted when (CNT = 1) and (READY* = 0).

CNT3 := /ADSn *  SIZE2 * /SIZE1 * /SIZE0                         ; LOAD
      +  ADSn *  READYpinn *  CNT3                               ; HOLD
      +  ADSn * /READYpinn *  CNT3 *  CNT2 *  CNT1 *  CNT0       ; DOWN
      +  ADSn * /READYpinn *  CNT3 *  CNT2 *  CNT1 * /CNT0       ; DOWN
      +  ADSn * /READYpinn *  CNT3 *  CNT2 * /CNT1 *  CNT0       ; DOWN
      +  ADSn * /READYpinn *  CNT3 *  CNT2 * /CNT1 * /CNT0       ; DOWN
      +  ADSn * /READYpinn *  CNT3 * /CNT2 *  CNT1 *  CNT0       ; DOWN
      +  ADSn * /READYpinn *  CNT3 * /CNT2 *  CNT1 * /CNT0       ; DOWN
      +  ADSn * /READYpinn *  CNT3 * /CNT2 * /CNT1 *  CNT0       ; DOWN
      +  ADSn * /READYpinn * /CNT3 * /CNT2 * /CNT1 * /CNT0       ; DOWN

CNT2 := /ADSn *  SIZE2 * /SIZE1 * /SIZE0                         ; LOAD
      + /ADSn * /SIZE2 *  SIZE1 *  SIZE0                         ; LOAD
      +  ADSn *  READYpinn *  CNT2                               ; HOLD
      +  ADSn * /READYpinn *  CNT2 *  CNT1 *  CNT0               ; DOWN
      +  ADSn * /READYpinn *  CNT2 *  CNT1 * /CNT0               ; DOWN
      +  ADSn * /READYpinn *  CNT2 * /CNT1 *  CNT0               ; DOWN
      +  ADSn * /READYpinn * /CNT2 * /CNT1 * /CNT0               ; DOWN

CNT1 := /ADSn *  SIZE2 * /SIZE1 * /SIZE0                         ; LOAD
      + /ADSn * /SIZE2 *  SIZE1 *  SIZE0                         ; LOAD
      + /ADSn * /SIZE2 *  SIZE1 * /SIZE0                         ; LOAD
      +  ADSn *  READYpinn *  CNT1                               ; HOLD
      +  ADSn * /READYpinn *  CNT1 *  CNT0                       ; DOWN
      +  ADSn * /READYpinn * /CNT1 * /CNT0                       ; DOWN

CNT0 := /ADSn *  SIZE2 * /SIZE1 * /SIZE0                         ; LOAD
      + /ADSn * /SIZE2 *  SIZE1 *  SIZE0                         ; LOAD
      + /ADSn * /SIZE2 *  SIZE1 * /SIZE0                         ; LOAD
      + /ADSn * /SIZE2 * /SIZE1 *  SIZE0                         ; LOAD
      +  ADSn *  READYpinn *  CNT0                               ; HOLD
      +  ADSn * /READYpinn * /CNT0                               ; DOWN

    BLASTn := /( /ADSn * /SIZE2 * /SIZE1 * /SIZE0 * RESETn
               + /READYpinn * /CNT3 * /CNT2 * /CNT1 * CNT0 * RESETn
               + /BLASTn    * READYpinn              * RESETn )

 BLASTpinn := { BLASTn }
BLASTpinn.TRST = /SQBGNTn
```

Table B-1. SAR Controller MACH210, MACHXL File (Sheet 6 of 6)

```
;******************************;
;***  SAR Master Equations  ***;
;******************************;

      ADSn := /( sM2 )
   ADSpinn := { ADSn }
ADSpinn.TRST = /SQBGNTn

    CPUWR  := /SARDWRn
 CPUWRpin  := { CPUWR }
CPUWRpin.TRST = /SQBGNTn

  /SARRDY0 := /READYpinn

  /SARRDYn  =   /SARRDY0 * /SQBGNTn *  SARDWRn         ; READs pipelined
            + /READYpinn * /SQBGNTn * /SARDWRn         ; WRITEs normal

     /BE0n  = VCC
     /BE1n  = VCC
     /BE2n  = VCC
     /BE3n  = VCC

 BE0n.TRST  = /SQBGNTn
 BE1n.TRST  = /SQBGNTn
 BE2n.TRST  = /SQBGNTn
 BE3n.TRST  = /SQBGNTn

;*******************************;
;***  Physical I/F Equations ***;
;*******************************;

; SAR only holds a valid address for the 1st 6 clocks of a PHY read cycle.
; PHY write cycle works OK, as-is.
; PHY has an active HI ALE input available.
; ALE is held HI for write cycles and pulled LO during reads to latch addr.

    PHYALE := /( /PHYCSn *  PHYWRn )

;*********************;
;***  End Of H601  ***;
;*********************;
```