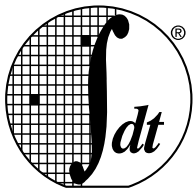


Integrated Device Technology, Inc.

79S465
Evaluation Board

Hardware User's Manual

Version 1.1
January 1997

2975 Stender Way, Santa Clara, California 95054
Telephone: (800) 345-7015 • TWX: 910-338-2070 • FAX: (408) 492-8674
Printed in U.S.A.

Integrated Device Technology, Inc.

Table of Contents

List of Tables and Figures

Integrated Device Technology, Inc.

Description of 79S465 Evaluation Board

Chapter 1

Introduction

The 79S465 Evaluation Board is an example of a complete working MIPS R4700 RISC System. The board is a low cost, few parts count design example that uses the highly integrated R4700 RISController CPU. Primary uses of the board include:

- R4700/R4650 architecture evaluation
- Prototypes and running software

The board is highly configurable and contains hardware options for different memory configurations and bus speed. Figure 1.1 shows a block diagram of the 79S465 board.

Overview Of Features

Major features of the 79S465 RISController Evaluation Board include:

- Complete 50MHz RISC system
- Supports the R4700/R4650/R4640 highly integrated RISController CPUs
- 64MByte DRAM, interleaved, 50nsec for 50MHz bus
- 2MBytes of Flash, non-interleaved, cacheable
- Ethernet interface
- 4 MBytes of SRAM, zero wait-state interleaved
- IDT's System Integration Manager (IDT/sim) included in Flash
- Two serial RS232 channels
- Clock, reset, and interrupt generation logic
- SCSI interface

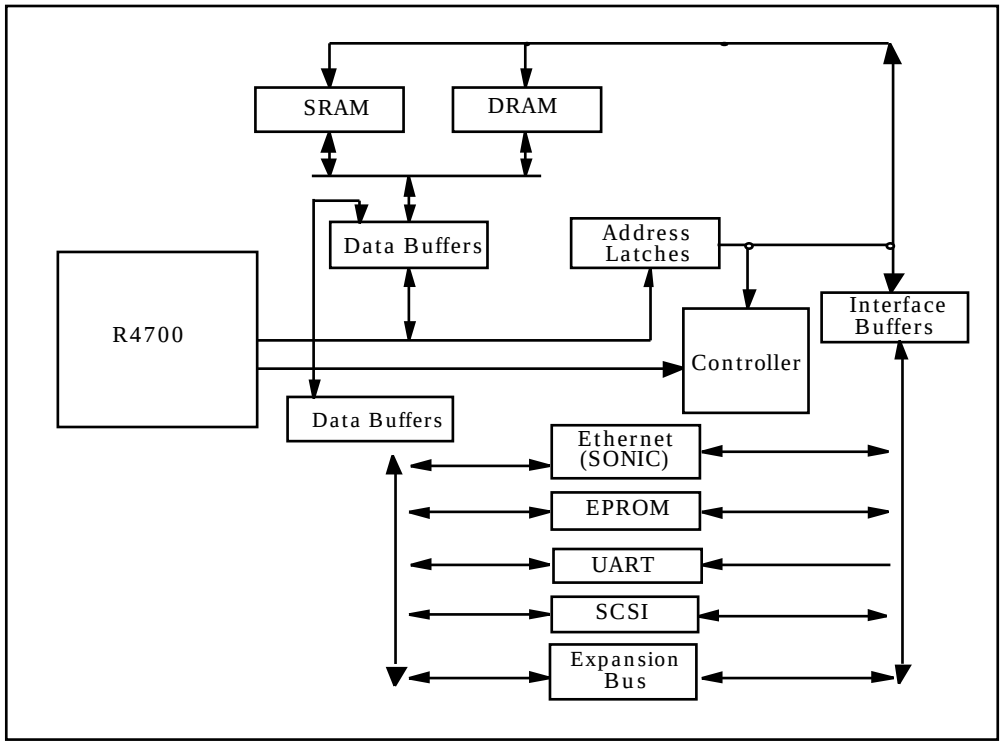


Figure 1.1 79S465 R4700 Evaluation Board Block Diagram

Explanation Of Features

The 79S465 Evaluation Board is configured as a stand-alone board, which requires only a standard RS232-C CRT video terminal and a 5-volt power supply for operation.

The 79S465 board contains 4 MBytes of SRAM SIMMs for zero wait-state systems. Main memory consists of 4 MBytes, using 256Kx32 DRAM SIMMs. The 79S465 is also equipped with two serial ports, one Ethernet port, and one SCSI port.

The boot EPROM contains IDT's System Integration Manager (IDT/sim), a debug monitor that supports download of code from host systems, and remote debug interface. Execution control commands include single stepping and instruction tracing, memory probing, register probing, line-based assembly, and disassembly of code.

A SPARCstation^{tm}, or PC/AT, can be connected to the 79S465 via one of the serial ports, and user developed code (generated using a cross-compiler such as GNU/GCC++, MIPS/C, or the IDT/c compiler) can be downloaded to the board.

The 79S465 board is designed around the IDT79R4700 RISController. The R4700 is a highly integrated version of the ORION family of RISControllers and includes 16KBytes of on-chip instruction and 16KBytes of on-chip data cache. For more information about the R4700 RISController, refer to the *IDT79R4600^{TM} & IDT79R4700^{TM} ORION^{TM}Processor Hardware User's Manual* and the R4700 Preliminary Data Sheet.

The 79S465 board can be used to evaluate the performance of the R4700. In addition, the 79S461, when populated with the R4650, can be plugged into the R4700 CPU socket, also allowing the R4650's system performance to be evaluated. Similarly, the 79S440, when populated with the R4640, can be used to evaluate the performance of the R4640.

The 79S465 board has a daughter card attachment—the 79S466—that uses the IDT79R4650 CPU and supports both 3.3V and 5V systems.

The 79S465 board is constructed with both through-hole and surface-mount devices on a 13" x 9" PCB rectangular form factor laminated board with standoffs, and is intended for use as a stand-alone bench top device. The board's physical layout is illustrated in Figure 1.1 on page 1-3.

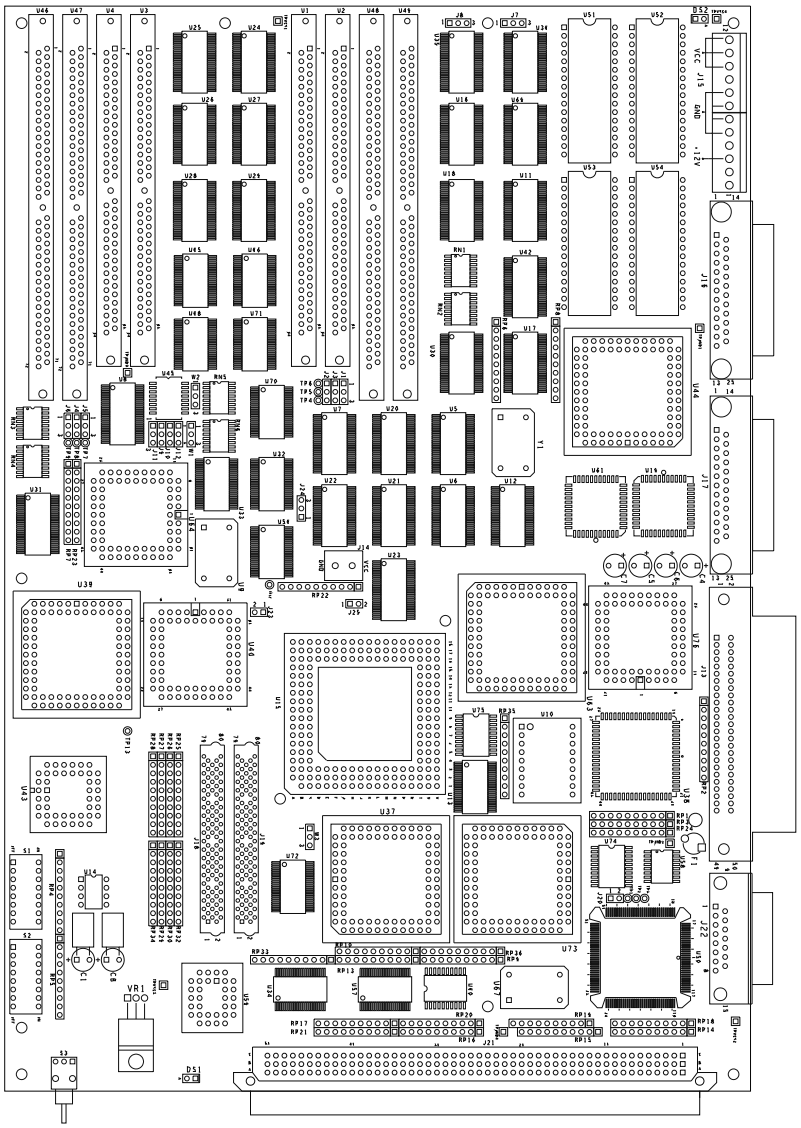


Figure 1.1 Physical Layout of 79S465 Evaluation Board

Specification Summary

Part Number
- IDT79S465

RISController
- IDT79R4700-100

On-Board Memory Capacity
- On-chip
  - Instruction Cache — 16KByte
  - Data Cache — 16KByte
- As shipped
  - DRAM — 4MByte
  - Flash — 2MByte
  - SRAM — 4Mbyte
- Maximum
  - DRAM — 64MByte
  - Flash — 2MByte
  - SRAM — 4Mbyte

Debug Monitor Flash
- 2MByte containing IDT/sim

Serial Ports
- Controlled by 85C30
- CRT Terminal connects to J16
- Software configurable features
- Default state: 9600 Baud, 8 bits, no parity, 1 stop bit

AUX Download Port
- Connects to J17
- Software configurable features
- Default state: 9600 Baud, 8 bits, no parity, 1 stop bit

Serial Port Connectors
- One DTE DB25 (25-pin right angle female) connector (J16)
- One DCE DB25 (25-pin right angle female) connector (J17)

Ethernet Port
- Controlled by National Ethernet Controller
- Sonic DP83932CUF-25

SCSI Port
- Controlled by NCR 53C90A

Timers
- On-chip R4700 timer
- Through Ethernet Controller

Interrupts
- 6 unsynchronized (3 used on-board)

Physical Dimensions
- Rectangular form factor: 13” x 9”

Operating Temperature
- 0-50°C

Relative Humidity
- 5% - 95%

Power Supply
- 5.0V ± 5%, 10 Amps typical

Flash
- Cached/non-cached, single access
  - X8 or X32 support
  - Slow access, slow operation; about 8 cycles (+ 1 release) for 120 ns Flash (one long word); code will run cached or uncached
  - Non-interleaved

DRAM
- Basic structure
  - 72 pin SIMM socketed (4 sockets on board)
  - 60 ns DRAM required for 50 MHz bus operation
- Configurations allowed
  - Only 4 SIMMs allowed (interleaved)
  - All SIMMs must be the same size/configuration
  - In one bank: 256Kx32 = 1 MB
  - Memory size configurable PLD programming

SRAM
- Basic structure
  - 64-pin SIMM socket (4 sockets on board)
  - 15ns SRAM
  - zero wait-state operation for block read and block write
- Configurations allowed
  - SIMM must be same size

Serial Ports
- 85C30 provides two serial UARTs (to 115 kbps)
- RS-232 SCSI port
- SCSI controlled by NCR 53C90A

Ethernet Port
- National DP83932 CVF-25
- DMA based

Expansion Connector
- 150-pin expansion connector for custom use

79S465 Installation

This chapter discusses the steps required to install and boot the 79S465 Evaluation Board.

The primary installation steps are as follows:

1. Connecting a power source
   - This involves connecting an external power supply to the board.
2. Connecting a video display monitor
   - This involves connecting an RS232-C cable from a video monitor to the board, through J16.
3. Configuring jumper options
   - This involves altering the CPU reset initialization mode vector and changing the memory configuration. The board is shipped with the jumpers set to the default configuration.
4. Connecting an Ethernet Port
   - This involves connecting an Ethernet cable from a PC or workstation to the board. This should only be used when downloading execution code through the Ethernet port.
5. Running software
   - No additional software is required.
6. Booting IDT/sim
   - When power to the board is turned on, the board's IDT/sim program boots and displays the start-up message.

Getting Started Quickly

The 79S465 board is shipped ready to run. Before the board is shipped, jumpers are configured to the default settings, and they do not require further modification or setup. The default jumper/switch settings are shown in Table 2.1.

Two basic requirements for the board to run are:

- +5V power supply with at least 10 Amp of current
- CRT video terminal with an RS232-C port connected to J16

The +5V power supply can be a typical PC compatible power supply. The connector on the evaluation board uses the two 6-pin power supply connectors that mate with the connectors on a standard PC compatible power supply.

The CRT video terminal can be a typical VT100 type/ANSI terminal or emulator running with 9600 baud, 8 data bits, no parity, and 1 stop bit. Typically, a video terminal will have a male 25-pin DTE connector. On the evaluation board, the RS232-C connector uses a female 25-pin DTE connector (J16) of which only the RX, TX, and GND pins are necessary.

For the default stand-alone mode, power to the board is provided by using a standard PC/AT power supply, available from a wide variety of computer equipment retailers. Two 6-pin connectors of the PC/AT power supply are used as the power entry connection. On the board, J15 should be connected to P8 and P9, as shown in Figure 2.1.

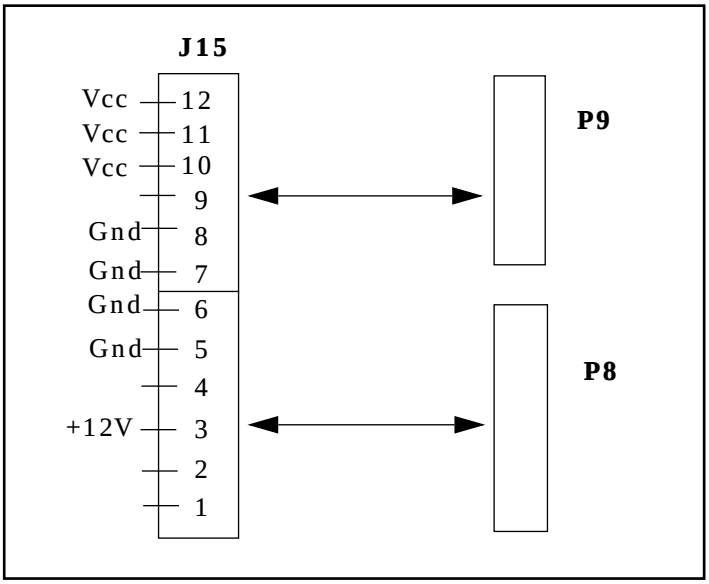


Figure 2.1 Connecting J15 to P8/P9 from PC Power Supply

The default jumper and switch settings for the 79S465 board are listed in Table 2.1 on page 3.

Jumper or Switch	Default	Description
J1	2-3	4MByte SRAM
J2	2-3	4MByte SRAM
J3	2-3	4MByte SRAM
J4	2-3	4MByte SRAM
J5	2-3	4MByte SRAM
J6	2-3	4MByte SRAM
J7	1-2	X32-bit Flash
J8	2-3	X32-bit Flash
J9	1-2	4MByte DRAM, 64-bit mode
J10	1-2	4MByte DRAM, 64-bit mode
J11	1-2	4MByte DRAM, 64-bit mode
J12	1-2	4MByte DRAM, 64-bit mode
J20	Open	Int 5 routed to internal R4700 timer
J23	1-2	SyncIn Routed to CPU
J24	1-2	Clock for R4700
J25	Open	Clock for R4700
W1	1-2	4MByte Dram
W2	1-2	4MByte DRAM
W3	1-2	5V System
S1.1	On	R4K Write compatible mode
S1.2	On	Clock divide by 2
S1.3	On	Clock divide by 2
S1.4	On	N/C
S1.5	Off	DRAM disabled
S1.6	Off	DRAM disabled
S1.7	On	SRAM Present and Enabled
S1.8	Off	4MByte SRAM
S2.1	On	N/C
S2.2	On	N/C
S2.3	On	Clock divide by 2
S2.4	Off	Big Endian
S2.5	On	R4xxx compatible mode
S2.6	Off	64-bit bus mode
S2.7	On	100% CPU output drive strength
S2.8	Off	Int 5 routed to internal R4700 timer
Note: For more information about these jumpers and switches, refer to Chapter 3.		

Table 2.1 Default Jumper/Switch Setting

System Software - IDT/sim

EPROMs on the 79S465 contain IDT's System Integration Manager (IDT/sim). IDT/sim is a software boot PROM debug monitor that provides functions for downloading software and for integrating hardware with software. Using IDT/sim, software can be downloaded onto the board from a SUN SPARCstation™ or a PC/AT personal computer.

Drivers are added easily by using the IDT Cross Development Software IDT/kit, and a user can acquire the IDT/sim source code to support other I/O devices or change I/O addresses to fit their specific application: for example, to change from big endian to little endian addressing.

Serial Port for CRT Video Terminal and Auxiliary Port

The 79S465 system board has two RS232-C serial port connectors. Pin assignments are listed in Table 2.2.

The port for the CRT video terminal is the DB25 (25-pin female) connector designated as J16 (refer to Table 2.1). The terminal must be set for a data rate of 9600 baud with 8 bits of data, no parity bits, and one stop bit. The J17 port is for auxiliary use—such as downloading software from a PC or SPARCstation™—as described later in this chapter. The J17 port is also DB25 (25-pin female).

The J16 is a DB25 connector (female), which uses the Data Communication Equipment (DCE) convention. As a DCE, the board connects to Data Terminal Equipment (DTE) terminals—such as PC/ATs—and most video terminals.

J16 25-Pin Female DCE			J17 25-Pin Female DCE		
Pin	**Board Signal**	**Board I/O**	**Pin**	**Board Signal**	**Board I/O**
3	TXD	Output	2	RXD	Output
2	RXD	Input	3	TXD	Input
5	RTS	Output	4	CTS	Input
4	CTS	Input	5	RTS	Output
7	GND	GND	6	DTR	Output
			7	GND	GND
			20	DSR	Input

Table 2.2 Serial Port Connections and Pin Assignments

Initialization and System Start-Up

System start-up is performed by turning on the power supply. If the power is already on, pressing the reset button will reinitialize the board. The board's default configuration is big endian. Once started, IDT/sim will automatically boot and size the CPU internal cache and main memory, determine which CPU is plugged into the board, and issue a message on the screen. The console is connected via the CRT serial port. A message indicating cache and memory sizes, similar to the one in Figure 2.1, will appear along with the first command line prompt.

Note: Future upgrades will be assigned a different version number and date. The starting address of the free memory space may differ slightly from the example shown in Figure 2.2 on page 5.

```
IDT System Integration Manager Ver. 6.5 beta May, 1995
Copyright 1994, 1995 Integrated Device Technology, Inc.
For help enter'?'
Memory size: 8126464 (0x7c0000) bytes
Icache size: 16384 (0x4000) bytes, 32/line
Dcache size: 16384 (0x4000) bytes, 32/line
User Memory Space 0xa001f6c8-0xa07bfffc
CPU:R4700. default baud rate: 9600 Register: 64-bit ENDIAN:
Big
<IDT>
```

Figure 2.2 Initial Screen Display for the IDT/sim Debug Monitor

Host-to-Target Program Downloading

Information on how to download code from your host to the evaluation board can be found in the IDT/c release notes.

Ethernet Downloading

Performing an Ethernet download consists of the following steps:

1. Setup the environment "netaddr" so that it is in the same subnet as the host-machine. This is done by issuing a "setenv" command under the <IDT> prompt in IDT/sim. For example, in the command
   setenv netaddr 89.0.5.4 for host "89.0.3.62"
   the 89 indicates that they are in the same subnet.
2. To load a Motorola S-record file, issue the "load" command under <IDT> prompt with "-t" option. The path of the file should include the host's internet address. For example:

   l -t 89.0.3.62: /jaguar/user/tftpboot/stanford.sre

3. To successfully complete a download from a Unix machine, the file "inet.conf" under "/etc" path, in the host machine, must be modified so that "tftp" demon is turned on. Activation of "tftp" demon is ensured when "tftp dgram udp root /usr/esc/in.tftpd in.tftpd" is present in "inet.conf". Please note that modifying this file may require superuser access.
4. An ecoff or elf executable file can be downloaded using a boot command. To download either of these files, a boot command can be issued with the "-n" option under the <IDT> prompt. Also, the path of the file name should include the internet address of the host machine. For example:

   boot -n 89.0.3.62:/jaguar/user/tftpboot/stan

Logic Analyzer Connections

There are no specific logic analyzer connectors on this board. However, an HP preprocessor pod E2438A, for the R4000 PC, can be used to connect to the HP logic analyzer. The preprocessor pod can be plugged into the PGA socket of the CPU.

The following sections provide information on the functional operation of the 79S465 R4700/R4650 evaluation board. For detailed schematics, refer to Chapter 4. For an example of PLD output, refer to Chapter 5.

Address Space Decoding

Table 3.1 provides a list of the physical addresses for the resources on the 79S465 board.

Note that access to the physical address space 1F70 0000-1F7F FFFF (expansion board address) should not be attempted unless an expansion board is connected to the expansion connector. If an expansion board is not connected, any attempt to access this address space will result in the board hanging, which requires a hardware reboot to recover.

Description	Physical Address
ROM Bank 0	 1FC0 0000 - 1FDF FFFF
SRAM (64 Bit-Mode) Bank 0 **SRAM** (32 Bit-Mode) Bank 0	 0000 0000 - 003F FFFF 0000 0000 - 001F FFFF
DRAM (64 Bit-Mode) Bank 0 **DRAM** (32 Bit-Mode) Bank 0	 0040 0000 - 03FF FFFF 0020 0000 - 01FF FFFF
I/0 Space Allocation a. Expansion CS b. Ethernet c. NVRAM d. SCSI e. SCC (8530)	 1F70 0000 - 1F7F FFFF 1F60 0000 - 1F6F FFFF 1F50 0000 - 1F5F FFFF 1F40 0000 - 1F4F FFFF 1F30 0000 - 1F3F FFFF

Table 3.1 79S465 Board Physical Addresses

Interrupts

The interrupt lines on the 79S465 board are assigned the values listed in Table 3.2. These interrupts become valid after ~**Reset** is asserted.

Interrupt	Signal and Description
Int0	Reserved
Int1	SCC
Int2	Expansion connector
Int3	Ethernet
Int4	SCSI
Int5	Timer/SCSI-2
NMI	N/C

Table 3.2 79S465 Interrupt Sources

Jumper or Switch Tables

The 79S465 board can be configured to work in different modes. As discussed in the following sections, Main Memory, Flash, Bus Frequency, CPU Voltage, Clocking Scheme, SyncIn Path, and Int5 options can be selected through a combination of jumpers and switches.

Main Memory Option

The main memory can be configured as an SRAM-only system (no DRAM). In this case, for both block read and block write operations, access to the SRAM is zero wait-state from the SRAM system. This option is ideal for benchmarking different routines and is the default setting.

The main memory can also be configured as a DRAM-only system (no SRAM). And although access to the DRAM is much slower than access to the SRAM, this case is a more realistic implementation of an actual system.

Finally, main memory can be configured as a combination of SRAM and DRAM. In this case, the block-read operation from the SRAM is zero wait-state, but the block-write operation from the SRAM is executed at the DRAM block-write speed.

These combinations allow the system user to investigate the effect of the main memory latency on a typical application. Table 3.3 lists the options available for SRAM only size selection, Table 3.4 lists the options for DRAM only size selection, and Table 3.5 lists the SRAM and DRAM options.

SRAM Only Option

SRAM SIZE		
0 Mbytes	**4 Mbytes in 64 bit-mode**	**2 Mbytes in 32 bit-mode**
S1.5 = OFF	S1.5 = OFF	S1.5 = OFF
S1.6 = OFF	S1.6 = OFF	S1.6 = OFF
S1.7 = OFF	S1.7 = ON	S1.7 = ON
S1.8 = don't care	S1.8 = OFF	S1.8 = OFF
J1 = don't care	J1 = 2-3	J1 = 2-3
J2 = don't care	J2 = 2-3	J2 = 2-3
J3 = don't care	J3 = 2-3	J3 = 1-2
J4 = don't care	J4 = 2-3	J4 = 2-3
J5 = don't care	J5 = 2-3	J5 = 2-3
J6 = don't care	J6 = 2-3	J6 = 1-2

Table 3.3 SRAM Size Selections

DRAM Only Option

DRAM SIZE		
0 Mbytes	**4 Mbytes in 64 bit-mode**	**2 Mbytes in 32 bit-mode**
S1.5 = OFF	S1.5 = ON	S1.5 = ON
S1.6 = OFF	S1.6 = ON	S1.6 = ON
S1.7 = OFF	S1.7 = OFF	S1.7 = OFF
S1.8 = don't care	S1.8 = ON	S1.8 = ON
J9 = don't care	J9 = 1-2	J9 = 2-3
J10 = don't care	J10 = 1-2	J10- = 2-3
J11 = don't care	J11 = 1-2	J11 = 2-3
J12 = don't care	J12 = 1-2	J12 = 2-3
W1 = don't care	W1 = 1-2	W1 = 2-3
W2 = don't care	W2 = 1-2	W2 = 2-3

Table 3.4 DRAM Size Selections

SRAM and DRAM Option

SRAM and DRAM SIZE		
0 Mbytes	**4 Mbytes in 64 bit-mode**	**2 Mbytes in 32 bit-mode**
S1.5 = OFF	S1.5 = ON	S1.5 = ON
S1.6 = OFF	S1.6 = ON	S1.6 = ON
S1.7 = OFF	S1.7 = ON	S1.7 = ON
S1.8 = don't care	S1.8 = OFF	S1.8 = OFF
J1 = don't care	J1 = 2-3	J1 = 2-3
J2 = don't care	J2 = 2-3	J2 = 2-3
J3 = don't care	J3 = 2-3	J3 = 1-2
J4 = don't care	J4 = 2-3	J4 = 2-3
J5 = don't care	J5 = 2-3	J5 = 2-3
J6 = don't care	J6 = 2-3	J6 = 1-2
J9 = don't care	J9 = 1-2	J9 = 2-3
J10 = don't care	J10 = 1-2	J10 = 2-3
J11 = don't care	J11 = 1-2	J11 = 2-3
J12 = don't care	J12 = 1-2	J12 = 2-3
W1 = don't care	W1 = 1-2	W1 = 2-3
W2 = don't care	W2 = 1-2	W2 = 2-3

Table 3.5 SRAM and DRAM Size Selections

Flash Option

The Flash system can be configured as either an 8-bit or 32-bit interface, as listed in Table 3.6.

Flash SIZE	
X32 bit (default)	**X8 bit**
J7 = 1-2	J7 = 2-3
J8 = 2-3	J8 = 1-2

Table 3.6 Flash Configuration Selections

Bus Frequency Option

The pipeline-to-bus frequency ratio can be configured as listed in Table 3.7.

Clock Divisor	Jumper/Switch		
	S2.3	S1.2	S1.3
divide by 2 (R4700) multiply by 2 (R4650)	ON	ON	ON
divide by 3 (R4700) multiply by 3 (R4650)	ON	ON	OFF
divide by 4 (R4700) multiply by 4 (R4650)	ON	OFF	ON
divide by 5 (R4700) multiply by 5 (R4650)	ON	OFF	OFF
divide by 6 (R4700) multiply by 6 (R4650)	OFF	ON	ON
divide by 7 (R4700) multiply by 7 (R4650)	OFF	ON	OFF
divide by 8 (R4700) multiply by 8 (R4650)	OFF	OFF	ON
N/A	OFF	OFF	OFF

Table 3.7 Pipeline-to-Bus Frequency Ratio Configuration Selections

CPU Voltage Option

The CPU voltage can be selected as either 3.3V or 5V. The 3.3V option can be easily selected when the S466 daughter card is used (the rest of the board will remain at 5V). Voltage selections are presented in Table 3.8.

CPU Voltage	Jumper/ Switch
	W3
5V	1-2
3.3V (using add-on card only	2-3

Table 3.8 Voltage Selections

Clocking Scheme Option

The 79S465 board's clocking scheme can be selected to accommodate either the R4700/R4650 or the R5000, as listed in Table 3.9.

Clock Selection	Jumper/Switch	
	J24	J25
R4700/R4650	2-3	OPEN
R5000	1-2	ON

Table 3.9 Clocking Scheme Selections

SyncIn Path Option

This option is used as a SyncIn to SyncOut path for the R4700 or as a clock path for the R4650.

SyncIn Path	Jumper/Switch J23
SyncIn routed to CPU for R4700	ON
SyncIn not routed to CPU for R4650	OFF

Table 3.10 SyncIn Path Selections

Int5 Option

The CPU Int5 can be shared by either the CPU internal timer or the second SCSI interrupt, as listed in Table 3.10.

CPU Internal Timer	Jumper/Switch J20	Jumper/Switch S2.8
Int5 used by internal timer	ON	ON
Int5 used by SCSI	OFF	OFF

Table 3.11 CPU Internal Timer Configurations

Miscellaneous Options

Miscellaneous settings are described in Table 3.12.

Jumper/Switch	Default	Description
S1.1	ON OFF	R4K Write compatible mode N/C
S1.4	OFF ON	N/C N/C
S2.1	ON OFF	N/C N/C
S2.2	ON OFF	N/C N/C
S2.4	OFF ON	Big Endian N/C
S2.5	ON OFF	R4xxx compatible N/C
S2.6	OFF ON	64-bit bus mode 32-bit bus mode
S2.7	ON OFF	100% output drive strength 83% output drive strength

Table 3.12 Miscellaneous Settings

Wait State Summary

Table 3.13 lists the transaction values and number of clock cycles required per transaction. For single read, values include the number of clock cycles required for the CPU to read the data. The bus turnaround at the end of the cycle is listed as an additional clock cycle.

For single writes, the number includes the two dead cycles at the end of a write operation in the R4xxx-compatible mode. For block reads, the first value is for the first data; the remaining values are for the three remaining data, respectively. The bus turnaround at the end of the cycle is listed as an additional clock cycle.

State	Clocks (Includes issue cycle and release)
8-bit FLASH	
Single Read	32 + 1*
Single Write	6
32-bit FLASH	
Single Read	8 +1*
Block Read	8/ 8 /8 /8 +1*
Single Write	6
SRAM zero wait-state (No DRAM)	
Single Read	3 + 1*
Block Read	3/1/1/1 + 1*
Single Write (R4xxx compatible)	4
Block Write	2/1/1/1
DRAM (No SRAM)	
Single Read	6 + 1*
Block Read	7/1/1/1 + 1*
Single Write (R4xxx compatible)	6
Block Write	2/1/3/1 +2**
SRAM & DRAM	
SRAM Single Read	3 + 1*
SRAM Block Read	3/1/1/1 + 1*
DRAM Single Read	6 + 1*
DRAM Block Read	7/1 /1 /1
DRAM & SRAM Single Write (R4xxx compatible)	6
DRAM & SRAM Block Write	2/1/3/1 + 2**

Table 3.13 Transaction Values and Number of Cycles

State	Clocks (Includes issue cycle and release)
I/O	
Ethernet Slave Read Single	15 +1*
Ethernet Slave Write Single	15
LED Display Single Read	10 + 1*
LED Display Single Write	10
SCSI Controller Single Read	10 + 1*
SCSI Controller Single Write	10
SCC Controller Single Read	10 + 1*
SCC Controller Single Write	10 + 1*
Note: *extra turnaround cycle at end of read access. **2 extra clock cycles at end of block write for DRAM.	

Table 3.13 (pg 2 of 2) Transaction Values and Number of Cycles

Parity

The 79S465 board does not store or return external parity bits to the CPU.

Power-ON LEDs

Two LEDs are installed on the 79S465 board, to indicate that power has been successfully applied. DJ2 indicates that the 5V power supply is ON; DJ1 indicates that the 3.3V power supply is ON.

R4600 Configuring From Reset

By using the boot-mode sequence, the CPU (R4700) is configured to the following settings:

- Transmit data pattern = 0 (DDDD) (for zero wait-state SRAM)
- SysClkRatio = 0 = divide by 2
- Big-endian
- R4xxx compatible writes
- Timer interrupt enabled
- Output drivers @ 100%

State Machine

The State Machine is implemented by using 9 Altera EPLDs. The state machine is fully synchronous and consists of the following distributed state machines:

- DRAM Controller
- SRAM Controller
- Flash Controller
- I/O Controller
- Sonic Controller
- Reset Controller
- Main Controller
- BusExchanger Controller
- RClock Controller.

The interface to the SRAM and the DRAM systems executes at the CPU bus speed of 50 MHz. while interface to the flash memory, the I/Os, and the Ethernet controller executes at half the CPU bus speed (25 Mhz).

Write Buffer

Between the CPU and the main memory, the state machine implements a two-level distributed write buffer. The first level is implemented by using '823 devices to latch the data from the CPU. The '823 devices are clocked by using the **RClock** from the CPU.

The second level is implemented by using the '16260 bus exchangers. There are two sets of bus exchangers: the first set is for the main memory (SRAM and DRAM); the second set is for the I/O subsystem (Flash, I/O, and Ethernet).

Main Memory Option

The main memory can be configured as an SRAM-only system (no DRAM). In this case, the access to the SRAM is zero wait-state from the SRAM system, both for block read and write operations. This is ideal for benchmarking different routines.

The main memory can also be configured as a DRAM-only system (no SRAM). Although access to the DRAM is much slower than access to the SRAM, this case is a more realistic implementation of an actual system.

Finally, the main memory can be configured as a combination of both SRAM and DRAM. In this case, the block read operation from the SRAM is zero wait-state but the block write operation from the SRAM is executed at the DRAM block write speed.

These various combinations of main memory configurations allow the system user to investigate the effect of the main memory latency on a typical application.

Write Protocols

The S465 supports only the R4xxx-compatible write mode.

Bus Width Option

For the R4650 and the R4600, the main memory (SRAM & DRAM) can be selected to be either 64- or 32-bits wide.

The Reset Controller

The Reset controller is driven by **MasterClock** and generates the 256 mode bits to the CPU. This controller has the option of selecting any pipeline to bus speed ratio. It also generates all of the reset signals to the CPU and the rest of the board.

The Main Controller

The Main controller is driven by **TClock.** By controlling the assertion and deassertion of the **~RdRdy** and **~WrRdy** signals, it interfaces directly to the CPU and manages the data flow between the CPU and the memory system. Similarly, by requesting the bus from the CPU and releasing it at the end of the DMA transfer, the Main Controller is responsible for managing the DMA operation between the Ethernet Controller and the main memory.

Address decoding is distributed among all of the controllers. Address bit 28 is used to differentiate between an I/O operation—including Flash access—and a main memory access such as DRAM or SRAM.

The Main Controller does not track the length of a particular access. Instead, it receives "**Done"** signals from the various memory controllers to determine the end of the access and when to assert the **~ValidIn** signal for a read access. The Main Controller then returns the **SysCmd(8:0)** signals to the CPU at the end of a read or DMA operation.

The Rclock Controller

The Rclock controller is the only controller clocked using the **RClock** signal, and it latches the read data from the memory to the CPU. The Rclock Controller also controls the write operation for the SRAM.

The Bus Exchanger Controller

The Bus Exchanger controller is driven by **TClock** and controls both the operation and the direction of the memory and I/O data buffers. It also controls the flow of the data among the CPU, the main memory, and the I/O subsystem. This controller is involved in any bus transaction such as memory read or memory write, I/O read or I/O write, and DMA read or DMA write.

The SRAM Controller

The SRAM Controller is driven by **TClock** and is responsible for the operation of the SRAM system only. If the SRAM option is enabled, then the SRAM system will be active. When the SRAM system is enabled, it is always located at physical address 0x0000_0000. A read operation to the SRAM (single or block read) is always a zero wait-state. The speed of the write operation to the SRAM depends on whether the DRAM option is engaged or not. If the DRAM is disabled, then the write to the SRAM (single or block) is always zero wait-state. If the DRAM option is on, then the write to the SRAM is executed at the DRAM write speed.

The SRAM access supports the R4xxx-compatible write mode and the DMA operation from the Ethernet port to the SRAM system. The SRAM Controller also supports the option of a 32-bit bus interface. This can be used in the case of the R4640 or the R4650 CPUs when configured in a 32-bit bus option.

The DRAM Controller

The DRAM controller is driven by **TClock** and is only responsible for operation of the DRAM system. The location of the DRAM system is always on top of the SRAM system. Thus, if the SRAM is enabled, the DRAM starts from the end of the SRAM space. If the SRAM is disabled, then the DRAM system will start at location 0x0000_0000. Access to the DRAM is much slower than access to the SRAM.

The DRAM controller supports the R4xxx-compatible write mode, and it supports DMA operation from the Ethernet port to the DRAM system. The DRAM Controller also supports a 32-bit bus interface option, which can be used in the case of the R4640 or the R4650 CPUs when configured as a 32-bit bus option.

The Flash Controller

The Flash controller is responsible for the operation of the Flash system and executes at half the CPU bus frequency. The Flash system can be either 8 or 32 bits wide. The Flash controller performs all of the necessary byte gathering and funneling before returning the data to the CPU. The Flash Controller can interface to either a 64- or 32-bit CPU main bus. The 32-bit wide Flash supports both cached and uncached accesses.

The Sonic Controller

The Sonic controller executes at half the CPU bus frequency and is responsible for the slave access to the Ethernet Controller. It is also responsible for the DMA operation from the Ethernet Controller to either the SRAM system or the DRAM system.

The I/O Controller

The I/O controller executes at half the CPU bus frequency and is responsible for all other I/O accesses, which include access to the serial communication controller, the LED display unit, the SCSI controller, and the expansion bus.

Cycle Types (List of possible transactions)

CPU as Bus Master

SRAM/DRAM from CPU in 64-bit mode

- Read
  - Single word
  - Four doubleword block refill
- Write
  - Single word or 1 - 8 bytes
  - Four doubleword block write

SRAM/DRAM from CPU in 32-bit mode

- Read
  - Single word
  - Eight word block refill
- Write
  - Single word or 1 - 8 bytes
  - Eight word block write

FLASH from CPU in 64-bit mode

- Read (all reads are 64-bit wide)
  - Single word for 8 and 32-bit Flash system
  - Four doubleword block refill in 32-bit Flash only
- Write (for Flash programming only)
  - Single word

FLASH from CPU in 32-bit mode

- Read
  - Single word for 8- and 32-bit Flash system
  - Eight word block refill in 32-bit Flash only
- Write
  - Single word

Ethernet Controller from CPU
(this path is 32 bits wide so block access is not implemented)

Schematics

This chapter contains a net list cross reference chart and 30 schematic drawings for the 79S465 evaluation board.

465 Evaluation Board Net List Cross Reference

Please note that VCC, VCC3.3, VDD and GND nets have not been cross referenced.

Net Name	Page/Coordinates
+12V	3/1E,20/3E
3_6MHZ	2/3C,3/4A,24/4C,28/2D
AD_MUX_LE	1/3D,3/4A,12/2B/2B,13/2D/2B,28/1D
B_RCLK0	1/3A/4E,5/3E,8/2A
B_RCLK1	1/3A/4E,5/3E,10/5A
B_RCLK2	5/3E,8/5A
B_RCLK3	5/4E,27/4C
B_TCLK0_0	1/4E,5/3E,27/1D
B_TCLK0_1	1/4E,5/3E,28/1B
B_TCLK0_2	5/4E,25/3D
B_TCLK1_0	5/3E,27/1A
B_TCLK1_1	1/4E,5/3E,28/1D
B_TCLK1_2	1/4E,5/4E
BSC_ADDR3O	10/1C,11/1B
BSC_ADDR4O	10/1C,11/1C
BYTE[7:0]_N	1/3A,3/4B
BYTE0_N	10/2D,11/2C,27/3D,28/2D
BYTE1_N	10/2D,11/2C,27/3D,28/2D
BYTE2_N	10/2D,11/2C,27/3D,28/3D
BYTE3_N	10/2D,11/2C,27/3E,28/1D
BYTE4_N	10/4D,11/4C,27/2D,28/3E
BYTE5_N	10/4D,11/4C,27/3D,28/3D
BYTE6_N	10/4D,11/4C,27/3D,28/2E
BYTE7_N	10/4D,11/4C,27/3D,28/2D
CAS[7:0]	1/2D,3/4B
CAS0	14/2A,28/3D
CAS1	14/2A,28/2D
CAS2	14/2A 28/2D
CAS3	14/2A,28/2E
CAS4	14/3A,28/2D
CAS5	14/3A,28/2D
CAS6	14/3A,28/2D
CAS7	14/3A,28/2D

COLDRESET_N	1/4D,2/3E,5/3A,6/2D,17/2D
D_ADDR_EV0	12/4B/4B
D_ADDR_EV8	12/3D/5D
D_ADDR_OD0	13/3C/4B
D_ADDR_OD8	13/3D/4D
D_SMST_AS_N	3/4C,25/3E,30/3C
DMA_ACK_N	3/4C,28/2C,30/2B
DMA_RD_N	2/2C,3/4A/4D,27/2D/3A/4D,28/3D,30/2D/3C
DMA_REQ_N	3/4A/4C,28/2A,30/3B/4C
DMA_WR_N	2/2C,3/4A/4D,27/1B/3D/5D,28/3E,29/3D,30/2D/3C
DRAM_ACCESS_N	3/3B,14/4C,27/1B/3D/5B,28/1D
DRAM_PRESENT_N	1/4E,3/4A,6/2C,27/2B/1D/4B
DRAM_SIZE[1:0]	1/4E,3/4A
DRAM_SIZE0	28/2C/2E
DRAM_SIZE1	28/2E/3B
E_BYTE[3:0]_LE_N	1/2A,3/4D
E_BYTE0_LE_N	18/5D,29/2E
E_BYTE1_LE_N	18/4D,29/3D
E_BYTE2_LE_N	18/2D,29/3D
E_BYTE3_LE_N	18/2D,29/3D
E_O_SELECT	1/3A,3/4B,9/5A,27/2D
E8_ADDR[2:0]	1/2A,3/4D
E8_ADDR0	18/1B,29/3D
E8_ADDR1	18/1B,29/2E
E8_ADDR2	18/1B,29/2D
E8_CE_N	1/1A,3/4D,18/2D,19/1A,29/3D
E8_OE_N	1/1A,3/4D,18/2D,19/1A,29/3E
E8W_OE_N	1/1A,3/5D
E8W_OE1_N	18/4E,29/2D
E8W_OE2_N	18/2E,29/3E
E8W_OE3_N	18/1E,29/3D
ED_OE_N	1/4A,3/4B,9/1D,27/1D
ED2RD_LE	1/4A,3/3B,9/1D,27/5C
EDW_ADDR[4:3]	1/1A,3/4D
EDW_ADDR3	2/2D,18/1B/3B,19/1A,25/3B,29/3D
EDW_ADDR4	2/2D,18/1B/3B,19/1A,25/3B,29/3D
EH_OE_N	1/1A,3/4D,18/5E,29/3D
EW_ADDR0	1/1A,2/2D,3/4D,18/3B,19/1A,25/3B, 29/3E
EX_ACK_N	25/1B/2C,26/4E
EX_BE0_N	25/1C,26/4D
EX_BE1_N	25/1C,26/4D
EX_BE2_N	25/2C,26/4D
EX_BE3_N	25/2C,26/4D
EX_BGRD_N	25/5C,26/4E
EX_BREQ_N	3/4C,25/2D/2A,26/4E

EX_CLK0	25/1C,26/3E
EX_CLK1	25/1C,26/3E
EX_DIR_N	3/5C,25/2B,30/3B
EX_EDW_ADDR3	25/3D,26/3E
EX_EDW_ADDR4	25/3D,26/3E
EX_ERR_N	25/2A/2C,26/4E
EX_EW_ADDR0	25/3D,26/3E
EX_EW_CE_N	25/4C,26/3E
EX_EW_OE_N	25/4C,26/3E
EX_INT_N	25/3D,26/4E
EX_NC10	25/2C,26/4E
EX_NC11	25/2C,26/4E
EX_NC12	25/2C,25/2A
EX_NC7	25/2C,26/4E
EX_NC8	25/2C,26/4E
EX_NC9	25/2C,26/4E
EX_RD_N	25/1C,26/3E
EX_RST_N	25/5C,26/4E
EX_TCLK	25/3E,26/2E
EX_UCS_N	25/5C,26/4E
EX_WE_8_N	25/3D,26/2E
EX_WE1_N	25/4C,26/2E
EX_WE2_N	25/4C,26/2E
EX_WE3_N	25/4C,26/3E
EX_WR_N	25/1C,26/3E
EX8_CE_N	3/4D,25/4B,29/2D
EX8_OE_N	3/4D,25/4B,29/3E
EXTRQST_N	1/4D,2/3D,3/5A,5/3C,17/1D,28/3A
FAULT_N	5/2B,6/4B
FLASH_DONE_IN	3/4A,20/4D,27/3E,29/3E
FLASH_DONE_N	3/4B,20/4D,27/1D,28/2A
FLASH8_N	1/1B,3/3C,29/3D
IGNORE_ACCESS_N	3/4B,14/4A,27/2A/2E,28/3E/3B
INT[5:0]_N	1/4D,2/3E
INT0_N	14/4A,17/4C,27/1D
INT1_N	2/3C,6/3B,24/3A
INT2_N	2/2E,6/3B,17/3C,25/3E
INT3_N	2/2C,14/4B,20/4B
INT4_N	2/2B,17/3C,23/3A
INT5_N	2/2B,6/3B
INV_IO_ENABLE_N	30/2B/5D
INVERT_EX_DIR_N	3/5C,25/3B,30/3B
IO_ACCESS_N	3/4A/5C,14/5A,27/2E,29/3B,30/2C
IO_ACK2_N	3/4A,28/2A,30/2B/2D
IO_ACK3_N	2/2D,3/4C,25/1B,30/3C

IO_ADDR2	8/5C,22/4B,28/2A,29/2A/3D,30/2B
IO_ADDR20	29/2D/3B,30/3B/4A
IO_ADDR21	29/3A/3D,30/3B/4A
IO_ADDR22	29/3A/2D,30/2C/4A
IO_ADDR23	29/3A/3D,30/2C/4A
IO_ADDR24	29/3B/3D,30/3B/4A
IO_ADDR25	29/3A/2D,30/3C/4A
IO_ADDR26	8/5C,29/3B/3E,30/2C
IO_ADDR27	8/5C,29/2A/3E,30/3B
IO_ADDR28	8/5C,29/3B/2D,30/3B
IO_ADDR3	8/5C,22/5B,29/2A/3D
IO_ADDR4	8/5C,29/2A/3D
IO_BGRD_N	3/4C,25/5C,30/3B
IO_BREQ_N	2/2E,25/2E,30/2B
IO_CLK0	2/1C/2B,27/1E,29/2D
IO_CLK1	3/3C,23/4A,27/1D,29/2B
IO_CLK2	20/4B,27/2E,30/2B
IO_CLK3	25/1B,27/2E,30/3D/5C
IO_ENABLE_N	8/5D,30/3B/5C/4A/5A
IO_ERR_N	2/2D,3/4C,25/2B,30/2B
IO_OE_N	2/2E,25/2A
IO_RD_N	2/2D,3/4C,25/1A,30/3C
IO_RESET_N	1/4E,2/2D,3/3C,6/2C,25/5B,29/2B/2D, 30/2C
IO_SYSCMD[3:0]	1/1B,3/3C
IO_SYSCMD0	29/3D,30/3B/4A
IO_SYSCMD1	29/3D,30/2B/5A
IO_SYSCMD2	29/3D,30/2C/5A
IO_SYSCMD3	29/2D,30/2B/5A
IO_SYSCMD6	3/5D,30/2C/5A
IO_UCS_N	3/4C,25/5C,30/2B
IO_WR_N	2/2B/2D,3/4C,25/1B,30/3C
IO2RA_CLK	18/3A,20/4E
IO2RA_LE	18/3A,20/4E
IO2RA_OE_N	1/1B,3/4B,18/3A,28/1B
IOENABLE_OUT_N	3/5B,28/3B
IOEX_LE1B	2/2B,3/4C,21/3C,28/1B,30/3B
IOEX_LE2B	2/2B,3/4C,21/2A,28/2A,30/3B
IOEX_LEA1B	2/2B,3/4C/5A,21/2A,29/3D,30/3B
IOEX_LEA2B	2/2B,3/4D/4B,21/2A,29/2D,30/3B
IOEX_OE1B_N	2/2B,3/4C/5A,21/2A,29/2D,30/2C
IOEX_OE2B_N	2/2B,3/4C/5A,21/2A,29/2D,30/2C
IOEX_OEA_N	2/2B,2/2B,21/3A/3B/3D,28/2C,30/3C
IOEX_SEL	2/1B,21/3B/3D/3E,28/3B,30/3C
IOIN	5/2B,6/4B,7/2D

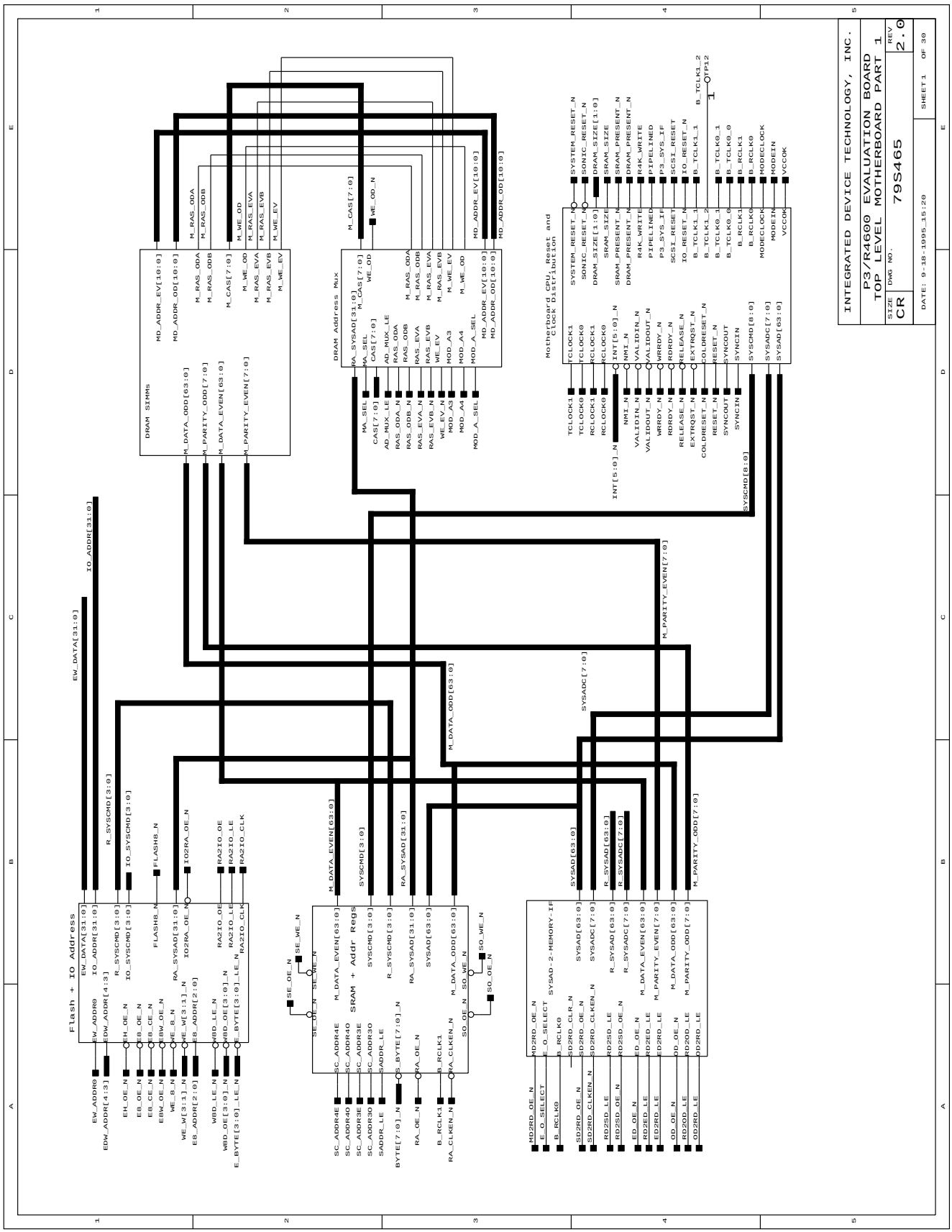
JTCK	5/2B,6/4B,7/2D
JTDI	5/2B,6/4B,7/2D
JTMS	5/2B,6/4B,7/2D
L_RA_SYSAD12	12/3D,14/3C/3E
L_RA_SYSAD13	12/3D/4B,13/4A
L_RA_SYSAD21	12/4C,14/3C
LED_BL_N	2/3C,3/3D,22/4C,29/3B
LED_CE_N	2/3B,22/4C,29/3B
LED_CLR_N	2/3B,3/3D,22/4B,29/3B
LED_CU_N	2/3C,3/3D,22/4B,29/3B
LED_CUE	2/3B,3/3D/3D,22/4B,29/3B
LED_WR_N	2/3C,3/3D,22/4B,29/3B
M_CAS0	14/3B,15/3A,16/3A
M_CAS1	14/3B,15/3A,16/3A
M_CAS2	14/3B,15/3A,16/3A
M_CAS3	14/3B,15/3A,16/3A
M_CAS4	14/3B,15/4A,16/4A
M_CAS5	14/3B,15/4A,16/4A
M_CAS6	14/3B,15/4A,16/4A
M_CAS7	14/3B,15/4A,16/4A
M_RAS_EVA	1/2E,14/2B,16/2B
M_RAS_EVB	1/2E,14/2B,16/4A
M_RAS_ODA	1/2E,14/1B,15/2A
M_RAS_ODB	1/2E,14/1B,15/4A
M_WE_EV	1/2E,14/2B,16/4A/3A
M_WE_OD	1/2E,14/2B,15/4A/3A
MA_SEL	1/2D,3/4A,12/3D,13/3D,28/1D
MASTERCLOCK	2/3E,5/1E,7/3B,17/3D
MASTERCLOCK2	5/1D/2D,6/1C
MASTEROUT	7/3B,17/3D
MD2RD_OE_N	1/3A,3/4B,9/5A,27/1D
MEM_DONE_N	3/3C/3B,20/4D,27/1D/4B,28/3E/2C, 30/3B
MOD_A_SEL	1/3D,3/5B,12/5C/5A,13/5C/4A,28/2E
MOD_A3	1/3D,3/5B,14/3E,28/2E
MOD_A4	1/3D,3/5B,12/4B,13/4A,28/2E
MODECLOCK	1/5E,2/3E,5/2A,6/1C,7/2C,17/2C
MODEIN	1/5E,2/3E,5/2A,6/2D,7/2C,17/2D
MSYSAD12_3	12/4C,13/4C,14/3D
MSYSAD21_12	12/4C,13/4C,14/3D
MSYSAD22_21	12/3D,13/2D,14/2D
MSYSAD23_22	12/3B,13/2B,14/1D
MSYSAD24_23	12/3D,13/2D,14/2D
MSYSAD25_24	12/3B,13/2B,14/1D
NMI_N	1/4D,2/3E,5/3A,6/3E,17/2C
NV_CE_N	2/3B,3/3D,22/3D,29/3B

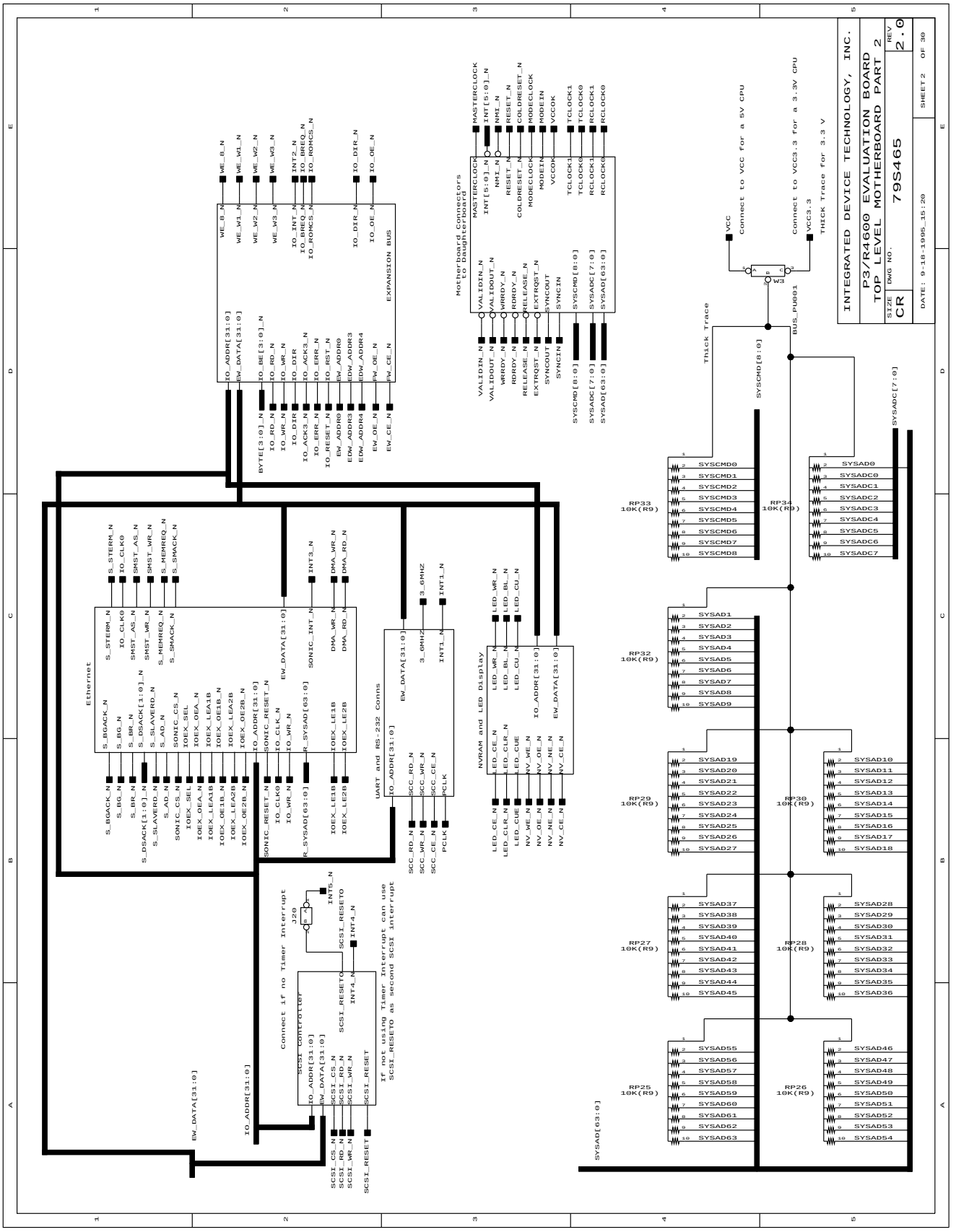
NV_NE_N	2/3B,3/4D,22/3B,29/3B
NV_OE_N	2/3B,3/3D,22/3B,29/3B
NV_WE_N	2/3B,3/4D,22/3B,29/3B
OD_OE_N	1/4A,3/4B,9/2E,27/1D
OD2RD_LE	1/4A,3/3B,9/1B,27/5C
OTHER_IO_DONE_N	29/3B,30/3B/2C
P3_SYS_IF	1/4E,3/3C/4A,27/3B/2E/5C,28/1E/1B, 29/3E,30/2C
PCLK	2/3B,3/3D,24/3A,29/3B
PIPELINED	1/4E,3/4A,28/3B
PUP10	13/2D,14/4B
PUP11	13/2B,14/4B
PUP12	13/2B,14/4B
PUP13	13/2B,14/4B
PUP14	13/2B,14/4B
PUP15	14/4B,21/3C
PUP16	14/4B,21/3C
PUP17	14/4B,21/3C
PUP18	14/4B,21/3C
PUP19	14/5A,21/3C
PUP2	14/4A,20/4A/4A
PUP20	14/5A,21/3C
PUP21	14/5A,21/3C
PUP22	14/5A,21/3C
PUP24	14/5A,21/4D
PUP25	14/5A,21/4D
PUP26	14/5A,21/4D
PUP27	14/5A,21/4D
PUP37	14/2A,14/4C
PUP38	14/2A,14/4C
PUP4	10/1B,14/4A
PUP43	13/2D,14/4C
PUP44	13/2C,14/4C
PUP45	13/3D,14/4C
PUP46	13/3C,14/4C
PUP47	14/4C,20/4A
PUP5	12/2B,14/4A
PUP6	12/2B,14/4A
PUP7	12/2C,14/4A
PUP8	12/2C,14/4A
PUP9	13/2C,14/4A
R_SYSAD[63:0]	1/4B,2/2B
R_SYSCMD3	3/3A,27/5B,28/3D
R4K_WRITE	1/4E,3/4A,28/3A
RA_CLKEN_N	1/3A,3/3B,10/5A,27/4B,28/3B

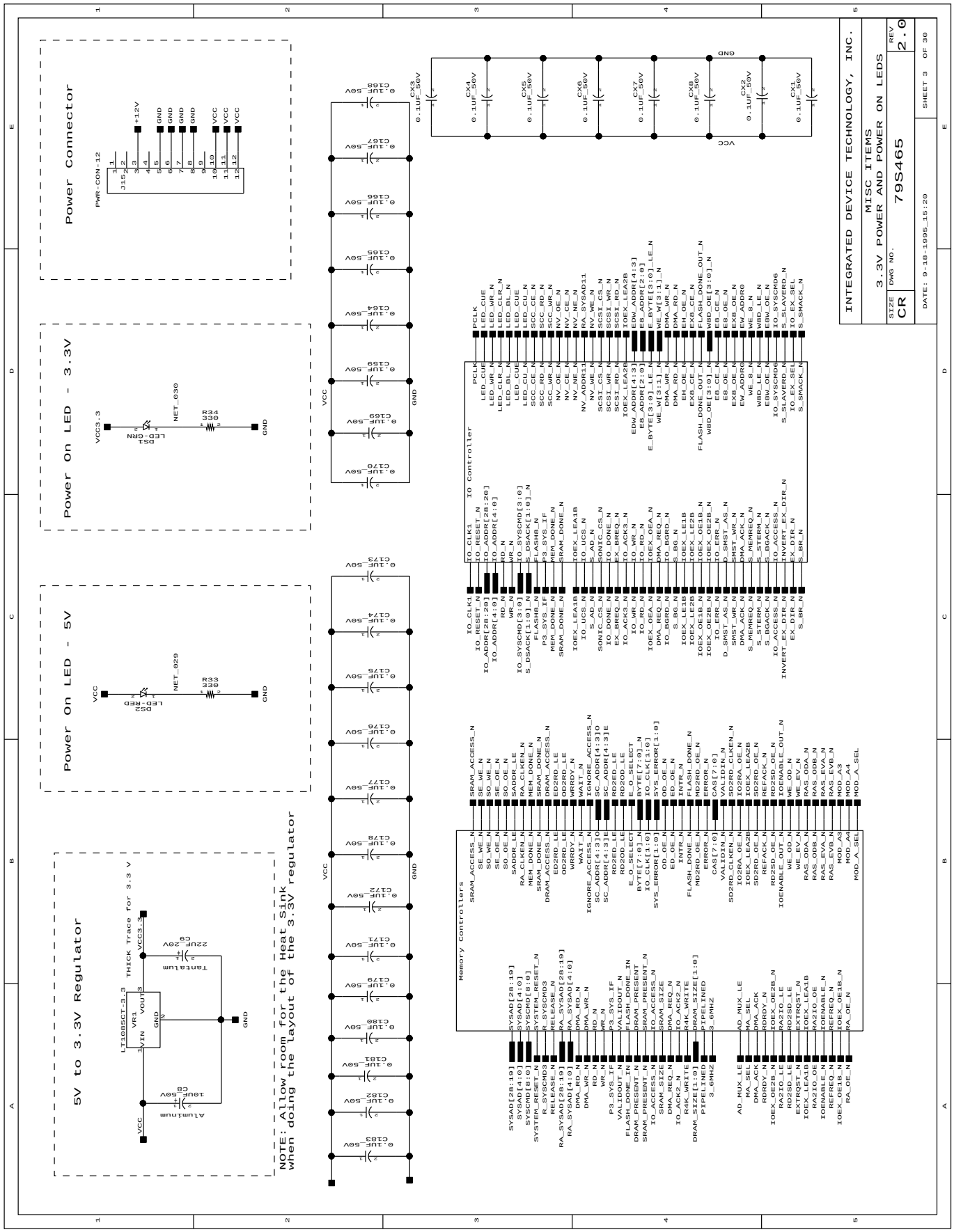
RA_OE_N	1/3A,3/5A,10/1B,28/3B
RA_SYSAD11	3/4D,29/3A
RA_SYSAD2	27/5C,28/2D
RA_SYSAD20	27/2E,28/3D
RA_SYSAD21	12/3C,14/2E,27/3D,28/3E
RA_SYSAD22	14/1E/2C,27/2D,28/3D
RA_SYSAD23	14/1D/2E,27/2D,28/3D
RA_SYSAD24	14/1E/2D,27/2E,28/3E
RA_SYSAD25	14/1D,27/2D,28/3D
RA_SYSAD26	27/2D,28/2E
RA_SYSAD27	27/2E,28/3D
RA_SYSAD28	27/2E/5C,28/3E
RA_SYSAD3	27/4D,28/3D
RA2IO_CLK	1/2B,18/3A,28/2C
RA2IO_LE	1/2B,3/5A,18/3A,28/2C
RA2IO_OE	1/2B,3/5A,18/3A,28/3A
RAS_EVA_N	1/3D,3/5B,14/1A,28/1D
RAS_EVB_N	1/3D,3/5B,14/1A,28/3E
RAS_ODA_N	1/3D,3/5B,14/1A,28/3D
RAS_ODB_N	1/3D,3/5B,14/1A,28/1D
RCLOCK0	1/4D,2/4E,7/3B,17/3D
RCLOCK1	1/4D,2/4E,7/3B,17/3D
RD_N	3/3C/4A,27/1E/4B,28/2E/1B, /2D,30/3B 29/2B
RD2ED_LE	1/4A,3/4B,9/1D,27/2D
RD2OD_LE	1/4A,3/4B,9/1B,27/2D
RD2SD_LE	1/4A,3/5A,8/4E,28/5B/3C
RD2SD_OE_N	1/4A,3/5B,8/4E,28/5D/2A
RDRDY_N	1/4D,2/3D,3/5A,5/3C,17/2D,27/2D, 28/2C
REFACK_N	3/5B,28/2E/2A
REFREQ_N	3/5A,20/4D,28/2D/3B
RELEASE_N	1/4D,2/3D,3/3A,5/3C,17/1D,27/2A/2D, 28/3B
RESET_N	1/4D,2/3E,5/2A,6/2D,17/3D
S_AD_N	2/1B,3/4C,20/3B,30/2B
S_BG_N	2/1B,3/4C,20/3B,30/3B
S_BGACK_N	2/1B,3/5C,20/3A/4E,30/2B
S_BR_N	2/1B,3/5C,20/3B/4D,30/2C
S_DSACK[1:0]_N	2/1B,3/3C
S_DSACK0_N	20/3A/4E,30/4B
S_DSACK1_N	20/3A/4E,30/4B
S_MEMREQ_N	2/1C,3/4C,20/4C,30/2B
S_SLAVERD_N	2/1B,3/5D,20/3B,30/2B
S_SMACK_N	2/1C,3/5D,20/4C,30/3C
S_STERM_N	2/1C,3/4C,20/4A/4E,30/2B
SADDR_LE	1/2A,3/3B,10/4B,11/4A,27/4B
SC_ADDR3E	1/2A,10/1B,27/3B

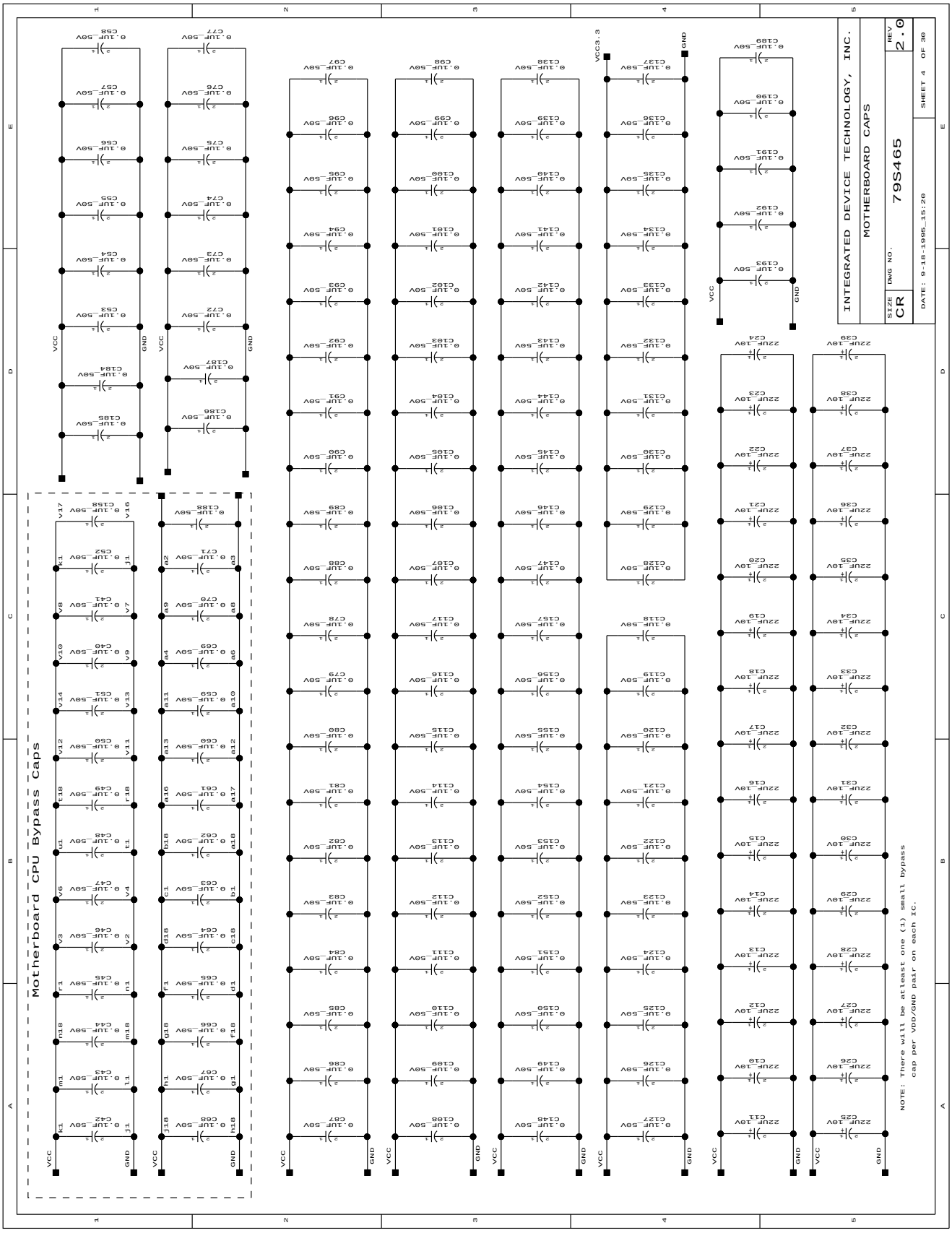
SC_ADDR3O	1/2A,10/1B,27/3B
SC_ADDR4E	1/2A,10/1B,27/3A
SC_ADDR4O	1/2A,10/1B,27/3A
SCC_CE_N	2/3B,3/3D,24/3A,29/3A
SCC_RD_N	2/3B,3/3D,24/3A,29/3B
SCC_WR_N	2/3B,3/3D,24/3A,29/3B
SCSI_CS_N	2/2A,3/4D,23/4C,29/3B
SCSI_RD_N	2/2A,3/4D,23/4C,29/2B
SCSI_RESET	1/4E,2/2A,6/2C,23/3B
SCSI_WR_N	2/2A,3/4D,23/4C,29/2B
SD2RD_CLKEN_N	1/4A,3/4B,8/2A,28/1B
SD2RD_OE_N	1/4A,3/4B,8/1A,28/2A
SE_OE_N	1/2A,3/3B,10/4D,27/3B
SE_WE_N	1/2B,3/3B,10/4D,27/5C
SMST_AS_N	2/1C,20/3C/4E,25/3D
SMST_WR_N	2/1C,3/4C,20/4C/4D,30/2B
SO_OE_N	1/3A,3/3B,11/4C,27/3B
SO_WE_N	1/3B,3/3B,11/4C,27/4D
SONIC_CS_N	2/1B,3/4C,20/3B,30/2B
SONIC_RESET_N	1/4E,2/2B,6/2D,20/4B
SRAM_ACCESS_N	3/3B,20/4D,27/2B/2E/4D,28/1E
SRAM_DONE_N	3/3B/3C,20/4D,27/3B/4D,28/2A,30/3B
SRAM_PRESENT_N	1/4E,3/4A,6/3E,27/2B
SRAM_SIZE	1/4E,3/4A,6/3E,27/3A,28/1D
SYNCIN	1/4D,2/3D,7/2D,17/3D
SYNCOUT	1/4D,2/3D,7/2D,17/3D
SYS_ERROR0	27/2E,28/1B
SYS_ERROR1	27/1E,28/2C
SYSAD[63:0]	1/4B,2/4D
SYSAD0	8/5D,27/2E
SYSAD1	8/5D,27/2E
SYSAD19	27/3B,30/4B
SYSAD2	8/5D,27/2D,28/3B
SYSAD20	27/2B,30/4B
SYSAD21	27/2B,30/4B
SYSAD22	27/3B,30/4B
SYSAD23	27/2A,30/4B
SYSAD24	27/2B,30/4B
SYSAD25	27/2B,30/4B
SYSAD26	8/5D,27/2A
SYSAD27	8/5D,27/2A
SYSAD28	8/5D,27/1A/2D,28/3B
SYSAD3	8/5D,27/2A/3E
SYSAD4	8/5D,27/3B

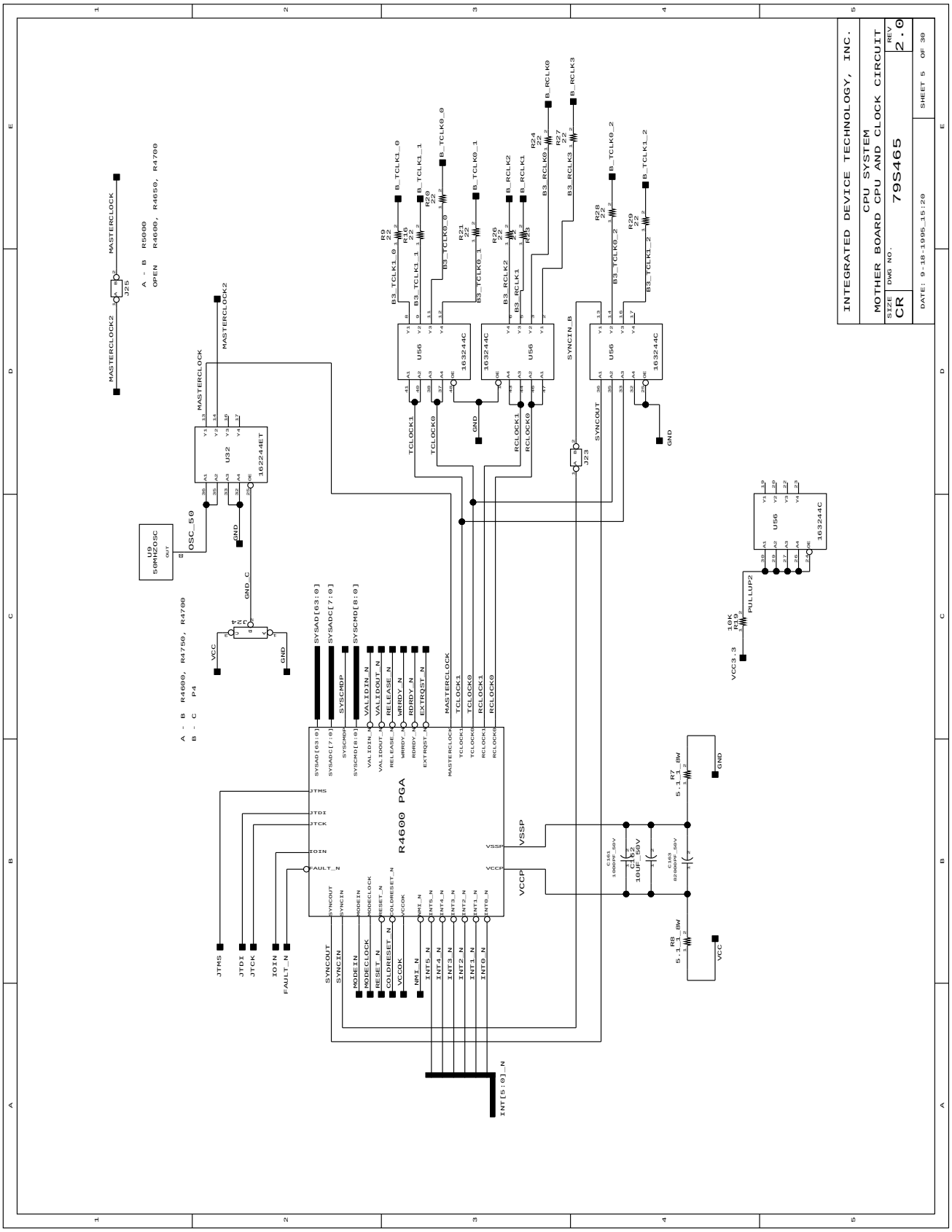
SYSCMD[8:0]	2/4D,3/3A
SYSCMD0	27/2E,28/5C,30/4B
SYSCMD1	27/2D,28/4C,30/5B
SYSCMD2	27/2D,28/4C,30/5B
SYSCMD3	27/1D/2A,28/3C/4C,30/5B
SYSCMD4	28/1B/4C
SYSCMD5	28/1B/4C
SYSCMD6	27/1D/2B/4B,28/2C/4C,30/5B
SYSCMD7	17/1C,28/3B/4C
SYSCMD8	27/3B/3D,28/2A/4C
SYSCMDOUT0	28/3B/5B
SYSCMDOUT1	28/2A/4B
SYSCMDOUT2	28/3B/4B
SYSCMDOUT3	28/3A/4B
SYSCMDOUT4	28/3C/4B
SYSCMDOUT5	28/2C/4B
SYSCMDOUT6	28/3B/4B
SYSCMDOUT7	28/1B/4B
SYSCMDOUT8	28/3C/4B
SYSCMDP	5/2C,6/4B,7/3B
SYSTEM_RESET_N	1/4E,3/3A,6/2D,27/1B/1E/4C,28/1B/1E
TCLOCK0	1/4D,2/4E,7/3B,17/4D
TCLOCK1	1/3D,2/3E,7/3B,17/3D
USR0	20/4E/5B
USR1	20/4E/5B
VALIDIN_N	1/4D,2/3D,3/4B,5/2C,17/1D,28/1B
VALIDOUT_N	1/4D,2/3D,3/4A,5/2C,17/1D,27/2A/2D/5C,28/2A
VCCOK	1/5E,2/3E,5/3A,6/2C,7/2C,17/3C
W8D_LE_N	1/2A,3/4D
W8D_LE0_N	18/4E,29/3D
W8D_LE1_N	18/3E,29/3D
W8D_LE2_N	18/2E,29/3E
W8D_LE3_N	18/1E,29/3D
W8D_OE[3:0]_N	1/2A,3/4D
W8D_OE0_N	18/4D,29/2E
W8D_OE1_N	18/3D,29/3E
W8D_OE2_N	18/2D,29/3E
W8D_OE3_N	18/1D,29/3D
WAIT_N	3/4B,20/4D,27/1D/2A,28/1B/2E
WE_8_N	1/1A,2/2E,3/4D,18/3D,25/3B,29/2E
WE_EV_N	1/3D,3/5B,14/2A,28/2E
WE_OD_N	1/2E,3/5B,14/2A,28/3D
WE_W[3:1]_N	1/1A,3/4D
WE_W1_N	2/2E,19/5B,25/4B,29/2E
WE_W2_N	2/2E,19/4C,25/4B,29/3D

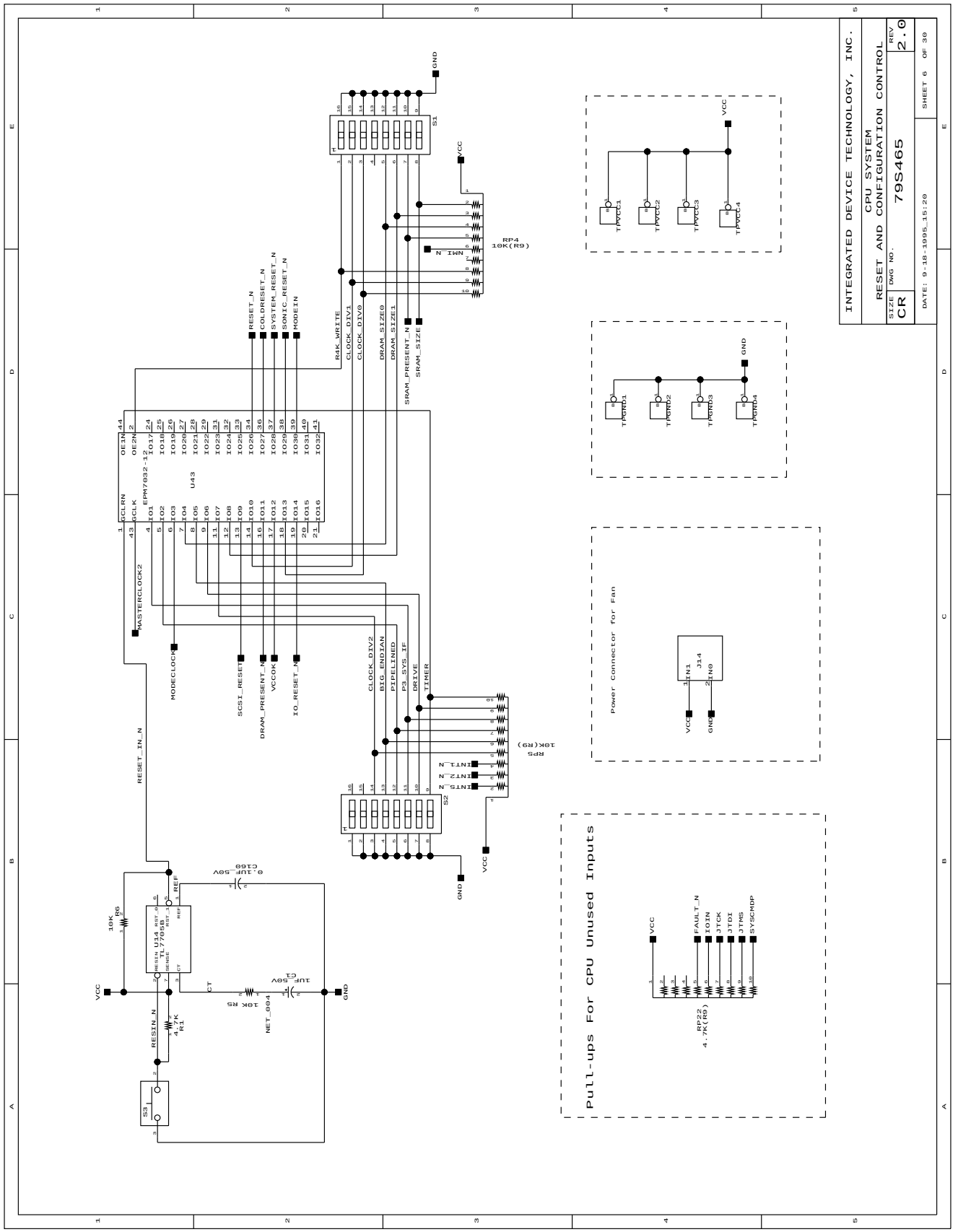

INTEGRATED DEVICE TECHNOLOGY, INC.
P3/R4600 EVALUATION BOARD
TOP LEVEL MOTHERBOARD PART 1
SIZE CR
DWG NO: 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET1 OF 30
DRAM SIMMs
DRAM Address Mux
Motherboard CPU, Reset and Clock Distribution
Flash + IO Address
SRAM + Addr Regs

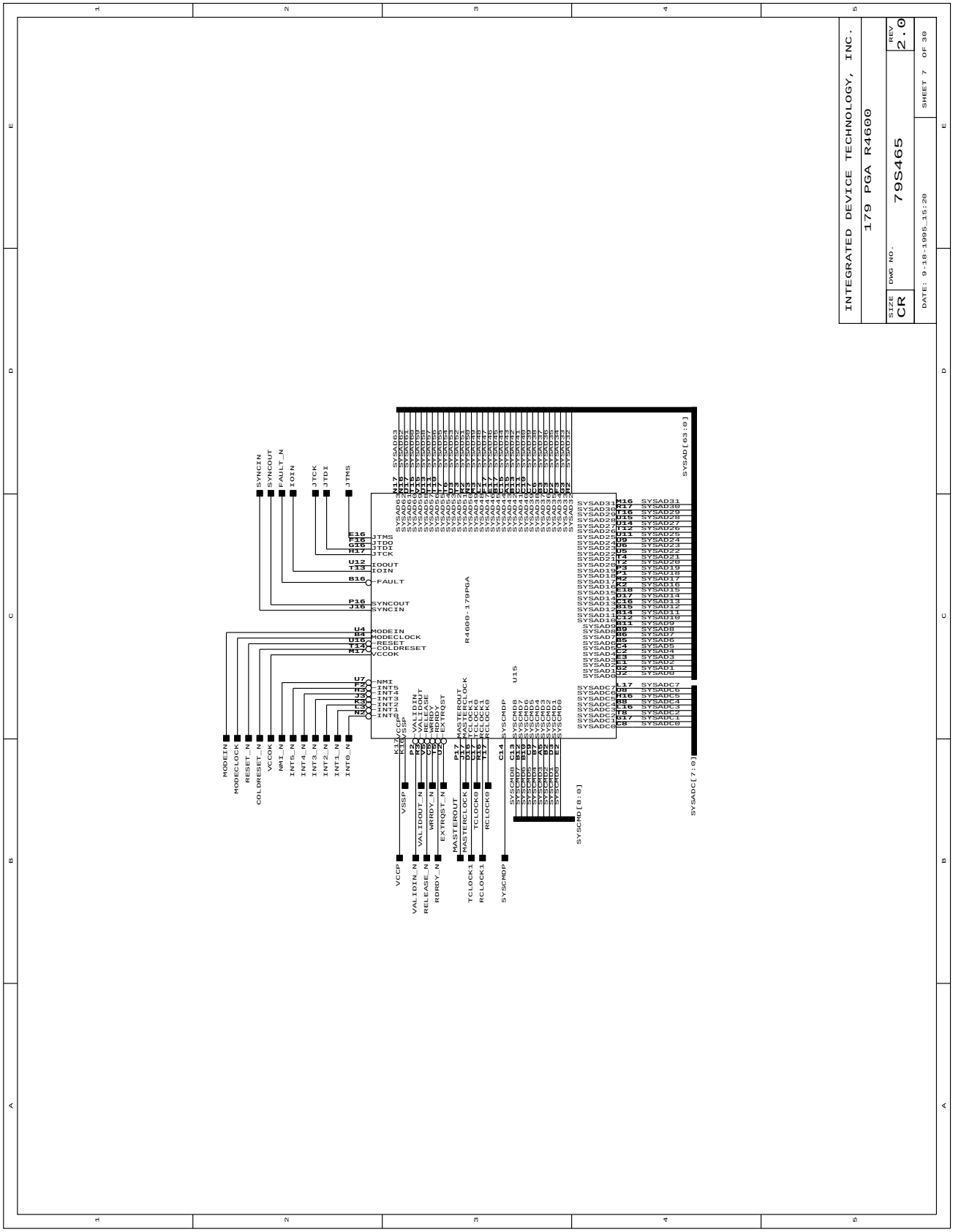

INTEGRATED DEVICE TECHNOLOGY, INC.
P3/R4600 EVALUATION BOARD
TOP LEVEL MOTHERBOARD PART 2
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 2 OF 30
Motherboard Connectors to Daughterboard
EXPANSION BUS
Ethernet
UART and RS-232 Conns
NVRAM and LED Display
SCSI controller
Connect if no Timer Interrupt
If not using Timer Interrupt can use SCSI_RESET0 as second SCSI interrupt
Thick Trace
Connect to VCC for a 5V CPU
Connect to VCC3.3 for a 3.3V CPU
THICK Trace for 3.3 V
RP33 10K(R9)
RP34 10K(R9)
RP32 10K(R9)
RP29 10K(R9)
RP30 10K(R9)
RP27 10K(R9)
RP28 10K(R9)
RP25 10K(R9)
RP26 10K(R9)

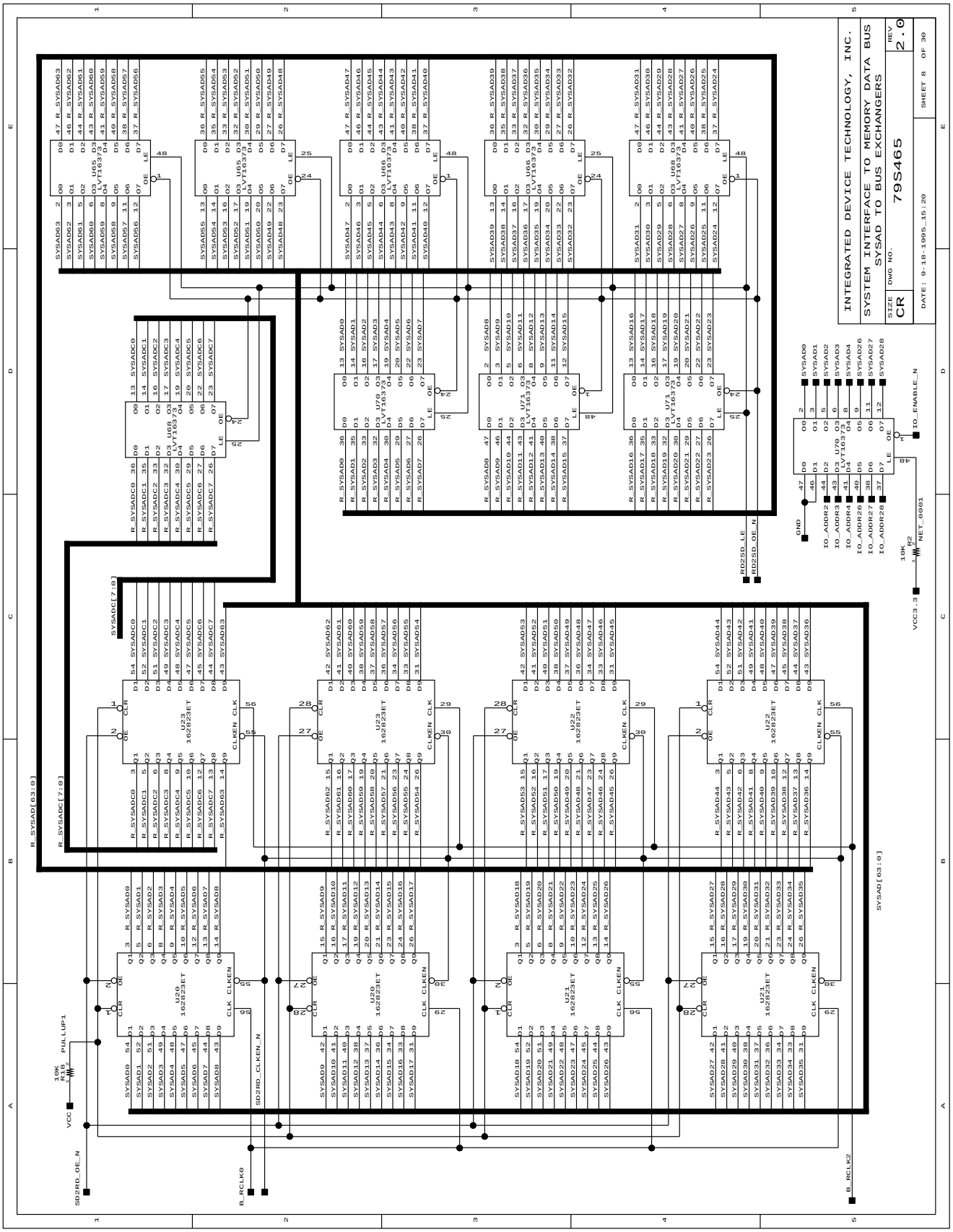

Power Connector
PWR-CON-12
J15
+12V
GND
VCC
Power On LED - 3.3V
VCC3.3
LED-GRN
DS1
NET_030
R34
330
Power On LED - 5V
VCC
LED-RED
DS2
NET_029
R33
330
5V to 3.3V Regulator
LT1085CT-3.3
VR1
VIN
VOUT
GND
VCC3.3
THICK Trace for 3.3 V
Tantalum
22uF_20V
C9
Aluminum
10uF_50V
C8
NOTE: Allow room for the Heat Sink when doing the layout of the 3.3V regulator
0.1uF_50V
CX1
CX2
CX3
CX4
CX5
CX6
CX7
CX8
GND
VCC
IO Controller
Memory Controllers
INTEGRATED DEVICE TECHNOLOGY, INC.
MISC ITEMS
3.3V POWER AND POWER ON LEDS
SIZE
CR
DWG NO.
79S465
REV
2.0
DATE: 9-18-1995_15:20
SHEET 3 OF 30

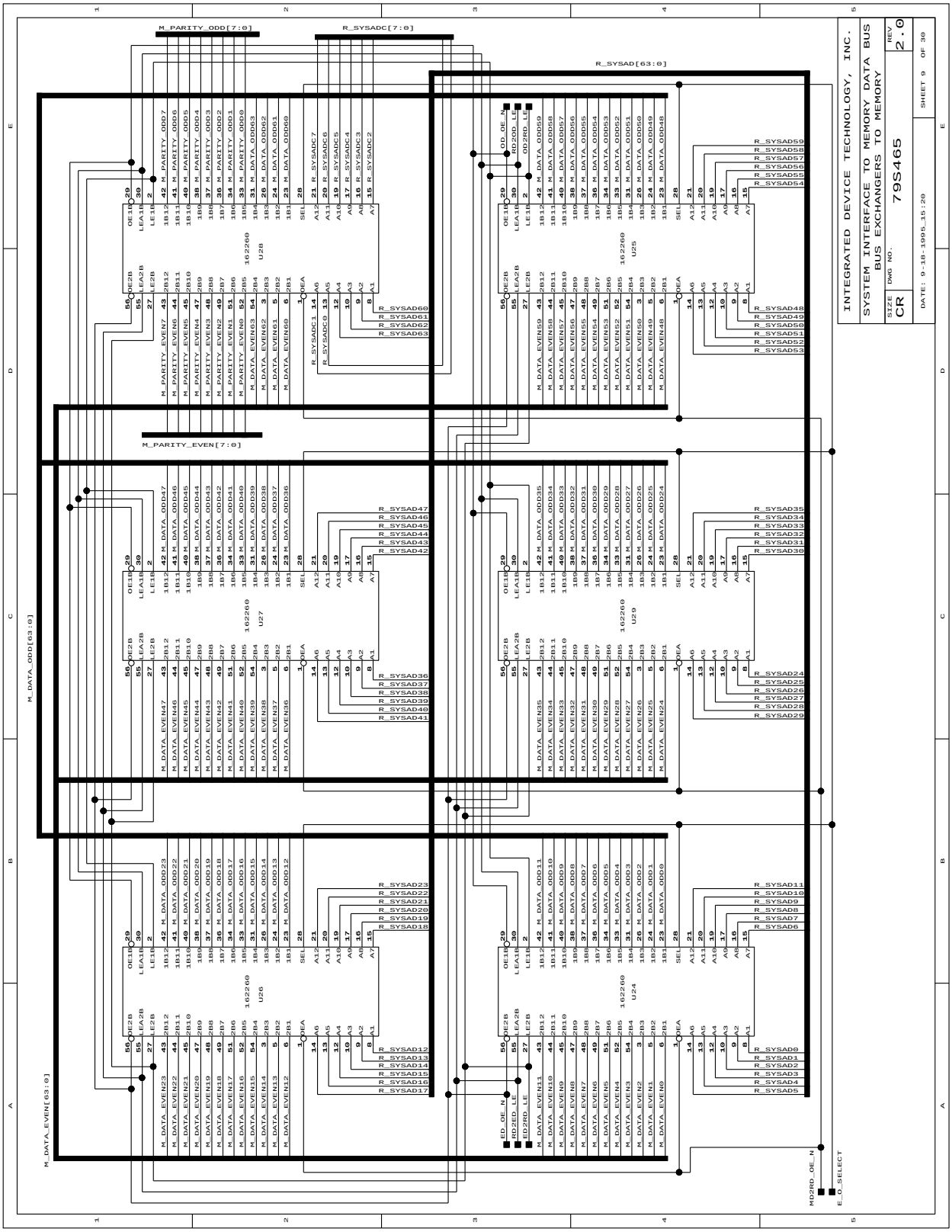

INTEGRATED DEVICE TECHNOLOGY, INC.
MOTHERBOARD CAPS
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_16:20
SHEET 4 OF 30
Motherboard CPU Bypass Caps
NOTE: There will be atleast one (1) small bypass cap per VDD/GND pair on each IC.
VCC
GND
VCC3.3
0.1UF_50V
22UF_10V

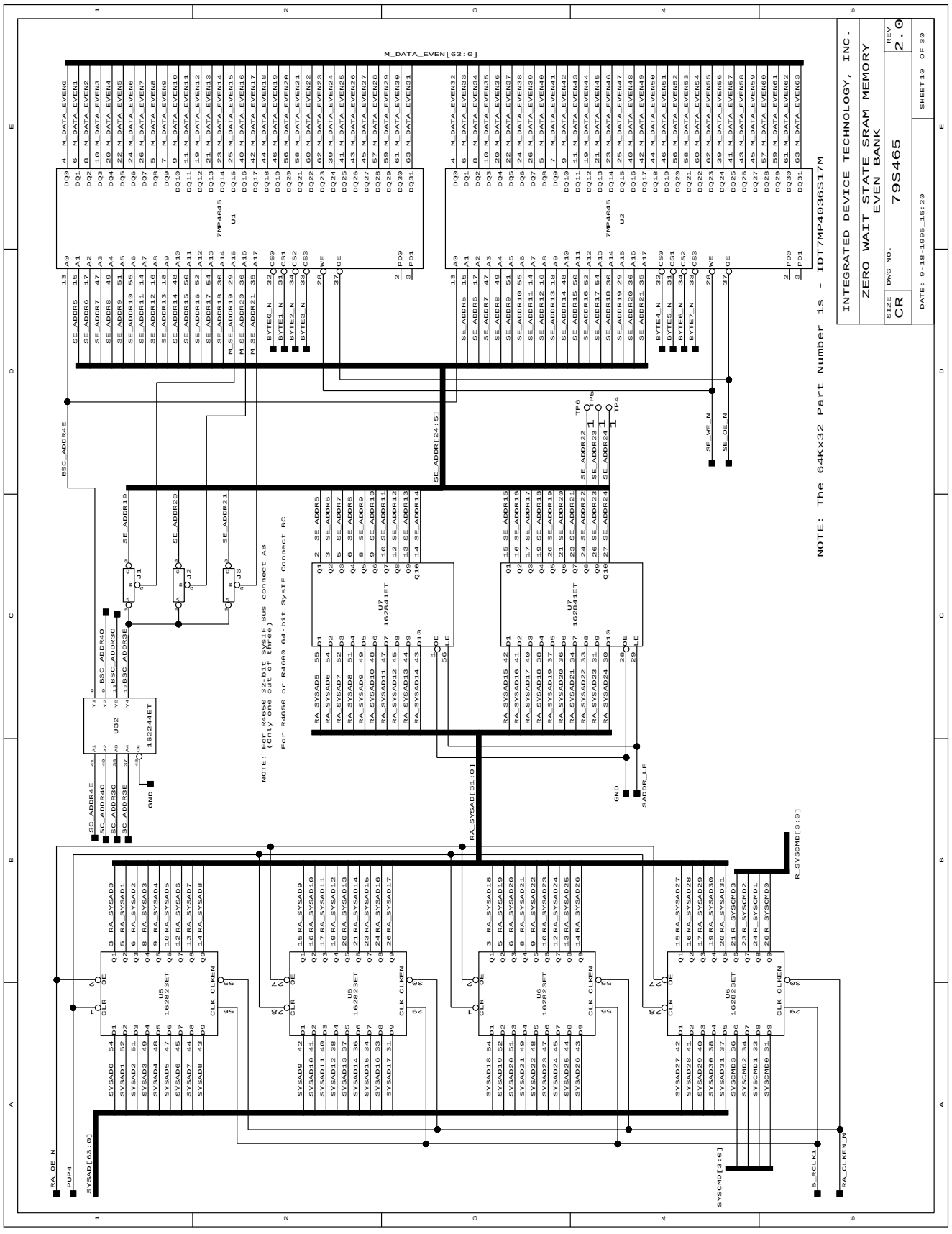

INTEGRATED DEVICE TECHNOLOGY, INC.
CPU SYSTEM
MOTHER BOARD CPU AND CLOCK CIRCUIT
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 5 OF 30
R4600 PGA
MASTERCLOCK
MASTERCLOCK2
OSC_50
SYSAD[63:0]
SYSADC[7:0]
SYSCMD[8:0]
SYSCMDP
VALIDIN_N
VALIDOUT_N
RELEASE_N
WRRDY_N
RDRDY_N
EXTRQST_N
TCLOCK1
TCLOCK0
RCLOCK1
RCLOCK0
JTMS
JTDI
JTCK
IOIN
FAULT_N
SYNCOUT
SYNCIN
MODEIN
MODECLOCK
RESET_N
COLDRESET_N
VCCOK
NMI_N
INT5_N
INT4_N
INT3_N
INT2_N
INT1_N
INT0_N
INT[5:0]_N
VSSP
VCCP
B_TCLK1_0
B_TCLK1_1
B_TCLK0_0
B_TCLK0_1
B_RCLK2
B_RCLK1
B_RCLK0
B_RCLK3
B_TCLK0_2
B_TCLK1_2
B3_TCLK1_0
B3_TCLK1_1
B3_TCLK0_0
B3_TCLK0_1
B3_RCLK2
B3_RCLK1
B3_RCLK0
B3_RCLK3
B3_TCLK0_2
B3_TCLK1_2
SYNCIN_B
SYNCOUT_B
PULLUP2
GND
VCC
VCC3.3

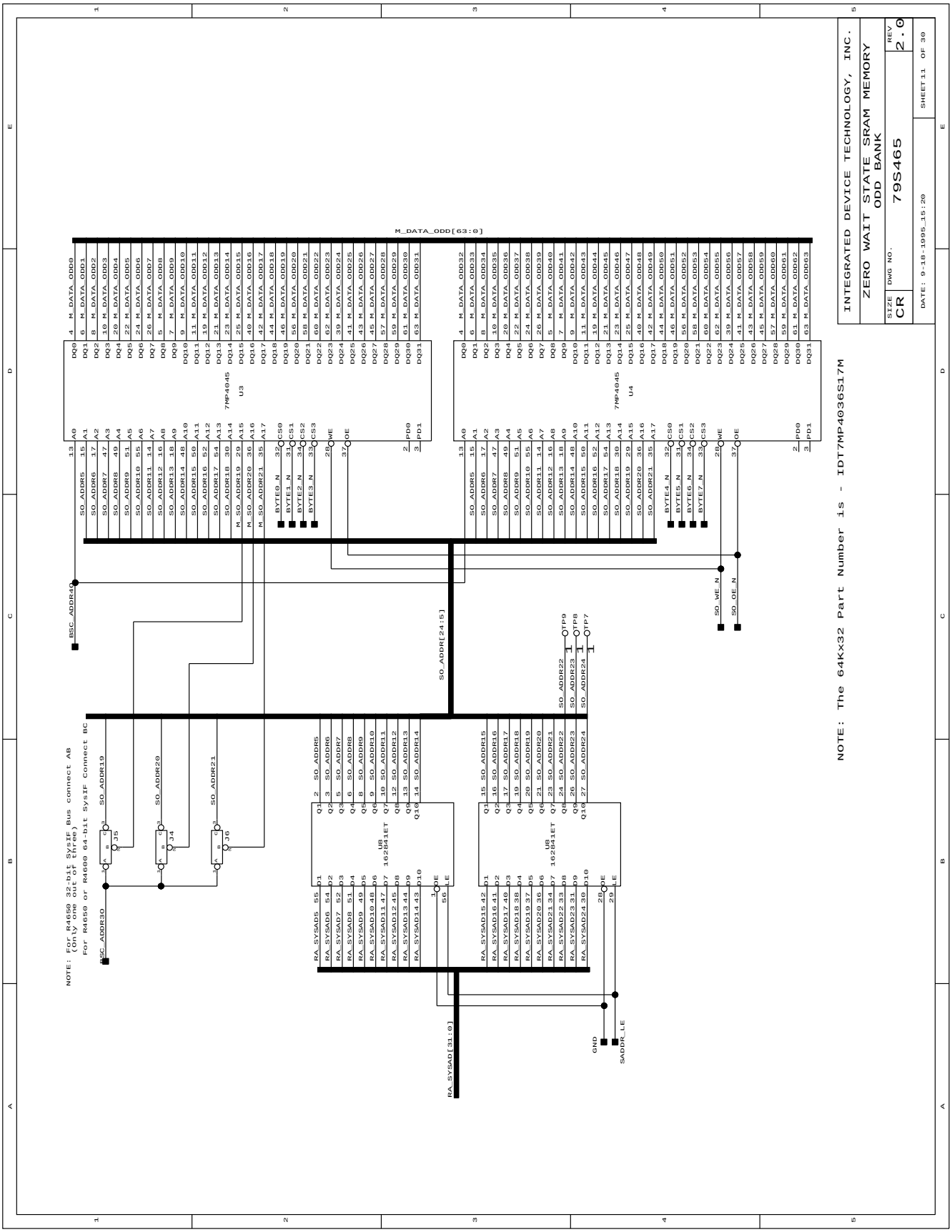

INTEGRATED DEVICE TECHNOLOGY, INC.
CPU SYSTEM
RESET AND CONFIGURATION CONTROL
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 6 OF 30
Pull-ups For CPU Unused Inputs
Power Connector for Fan
RESET_IN_N
MASTERCLOCK2
MODECLOCK
SCSI_RESET
DRAM_PRESENT_N
VCCOK
IO_RESET_N
CLOCK_DIV2
BIG_ENDIAN
PIPELINED
P3_SYS_IF
DRIVE
TIMER
RESET_N
COLDRESET_N
SYSTEM_RESET_N
SONIC_RESET_N
MODEIN
R4K_WRITE
CLOCK_DIV1
CLOCK_DIV0
DRAM_SIZE0
DRAM_SIZE1
SRAM_PRESENT_N
SRAM_SIZE
NMI_N
INT1_N
INT2_N
INT5_N
RESIN_N
FAULT_N
IOIN
JTCK
JTDI
JTMS
SYSCMDP
U43
EPM7032-12
U34
IDT 77065
S1
S2
S3
RP4 10K(R9)
RP5 10K(R9)
RP22 4.7K(R8)
R6 10K
R1 4.7K
R5 10K
C1 1UF 50V
C160 0.1UF_50V
J14
TPVCC1
TPVCC2
TPVCC3
TPVCC4
TPGND1
TPGND2
TPGND3
TPGND4
VCC
GND

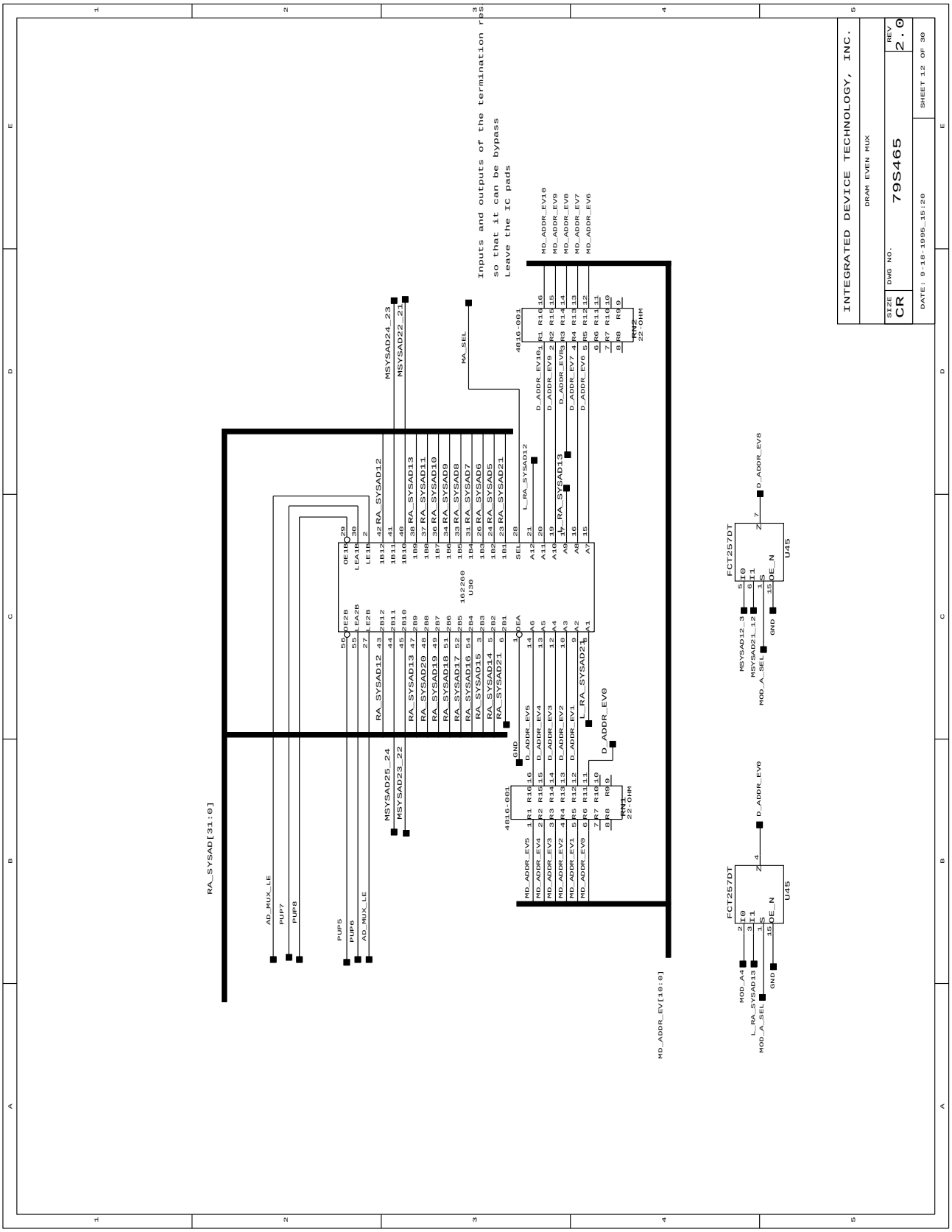

INTEGRATED DEVICE TECHNOLOGY, INC.
179 PGA R4600
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 7 OF 30
R4600-179PGA
SYSAD[63:0]
SYSADC[7:0]
SYSCMD[8:0]
MODEIN
MODECLOCK
RESET_N
COLDRESET_N
VCCOK
NMI_N
INT5_N
INT4_N
INT3_N
INT2_N
INT1_N
INT0_N
SYNCIN
SYNCOUT
FAULT_N
IOIN
JTCK
JTDI
JTMS
VALIDIN_N
VALIDOUT_N
RELEASE_N
WRRDY_N
RDRDY_N
EXTRQST_N
MASTERCLOCK
TCLOCK1
TCLOCK0
RCLOCK1
RCLOCK0
MASTEROUT
SYSCMDP
VCCP
VSSP

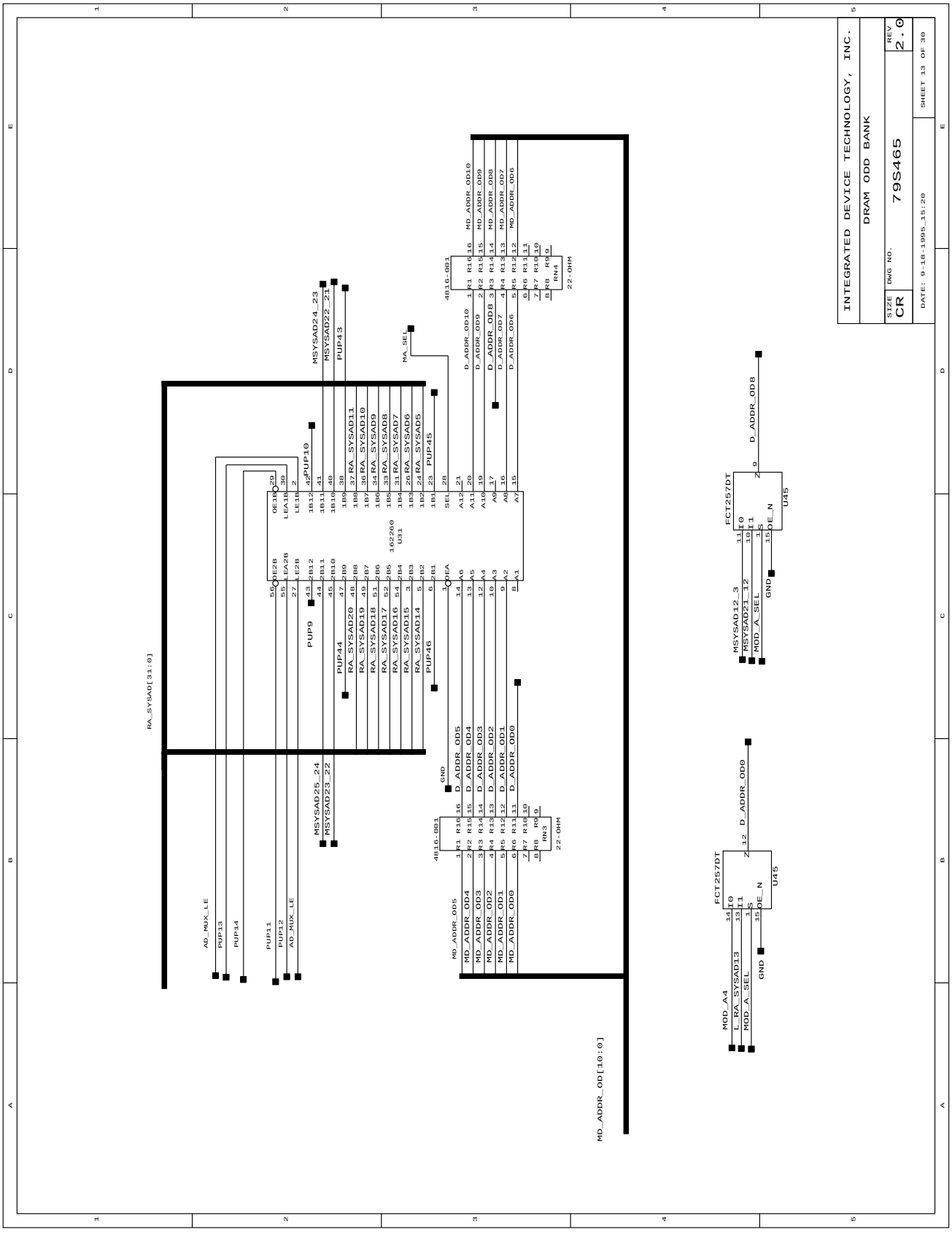

INTEGRATED DEVICE TECHNOLOGY, INC.
SYSTEM INTERFACE TO MEMORY DATA BUS
SYSAD TO BUS EXCHANGERS
79S465
SIZE CR
DWG NO.
REV 2.0
DATE: 9-18-1995_15:20
SHEET 8 OF 30
R_SYSAD[63:0]
R_SYSADC[7:0]
SYSAD[63:0]
SYSADC[7:0]
SD2RD_CLKEN_N
SD2RD_OE_N
SD2RD_LE
SD2SD_OE_N
B_RCLK0
B_RCLK2
IO_ENABLE_N
IO_ADDR2
IO_ADDR3
IO_ADDR4
IO_ADDR26
IO_ADDR27
IO_ADDR28
VCC
VCC3.3
GND
10K
PULLUP1
R1
R2
NET_0001
U20
U21
U22
U23
162823ET
LVT16373
CLK_CLKEN
CLKEN_CLK

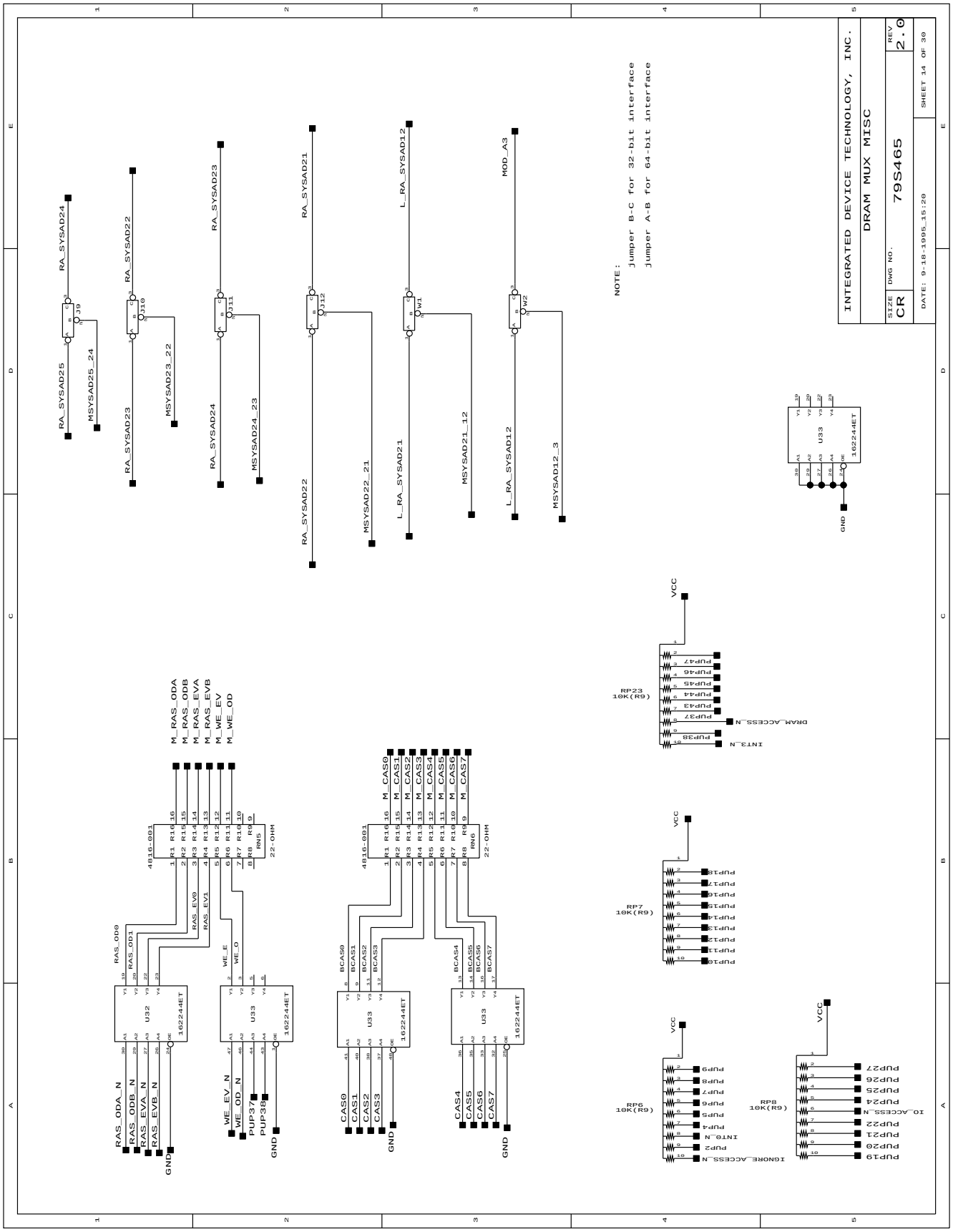

INTEGRATED DEVICE TECHNOLOGY, INC.
SYSTEM INTERFACE TO MEMORY DATA BUS
BUS EXCHANGERS TO MEMORY
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 9 OF 30
M_PARITY_ODD[7:0]
R_SYSADC[7:0]
R_SYSAD[63:0]
M_PARITY_EVEN[7:0]
M_DATA_ODD[63:0]
M_DATA_EVEN[63:0]
M2RD_OE_N
E_O_SELECT
162260
U28
U27
U26
U25
U29
U24

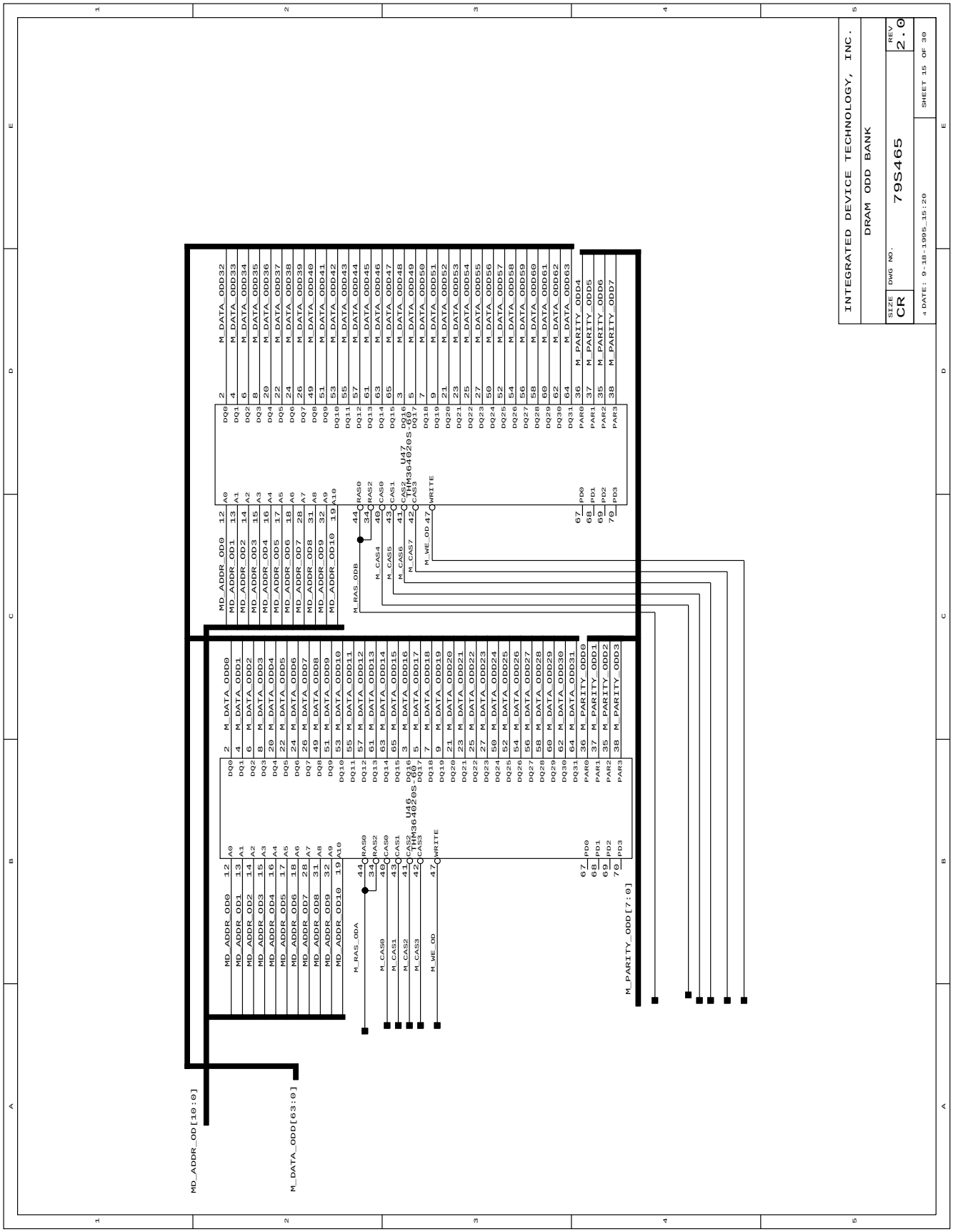

INTEGRATED DEVICE TECHNOLOGY, INC.
ZERO WAIT STATE SRAM MEMORY
EVEN BANK
79S465
SIZE CR
DWG NO.
REV 2.0
DATE: 9-18-1995 15:20
SHEET 10 OF 30
NOTE: The 64Kx32 Part Number is - IDT7MP4036S17M
M_DATA_EVEN[63:0]
7MP4045
U1
U2
BSC_ADDR4E
SE_ADDR[24:5]
SE_WE_N
SE_OE_N
NOTE: For R4650 32-bit SysIF Bus connect AB
(Only one out of three)
For R4650 or R4600 64-bit SysIF Connect BC
U32
162244ET
U7
162841ET
U5
U6
162823ET
RA_SYSAD[31:0]
R_SYSCMD[3:0]
SYSAD[63:0]
SYSCMD[3:0]
RA_OE_N
PUP4
B_RCLK1
RA_CLKEN_N
GND
SADDR_LE

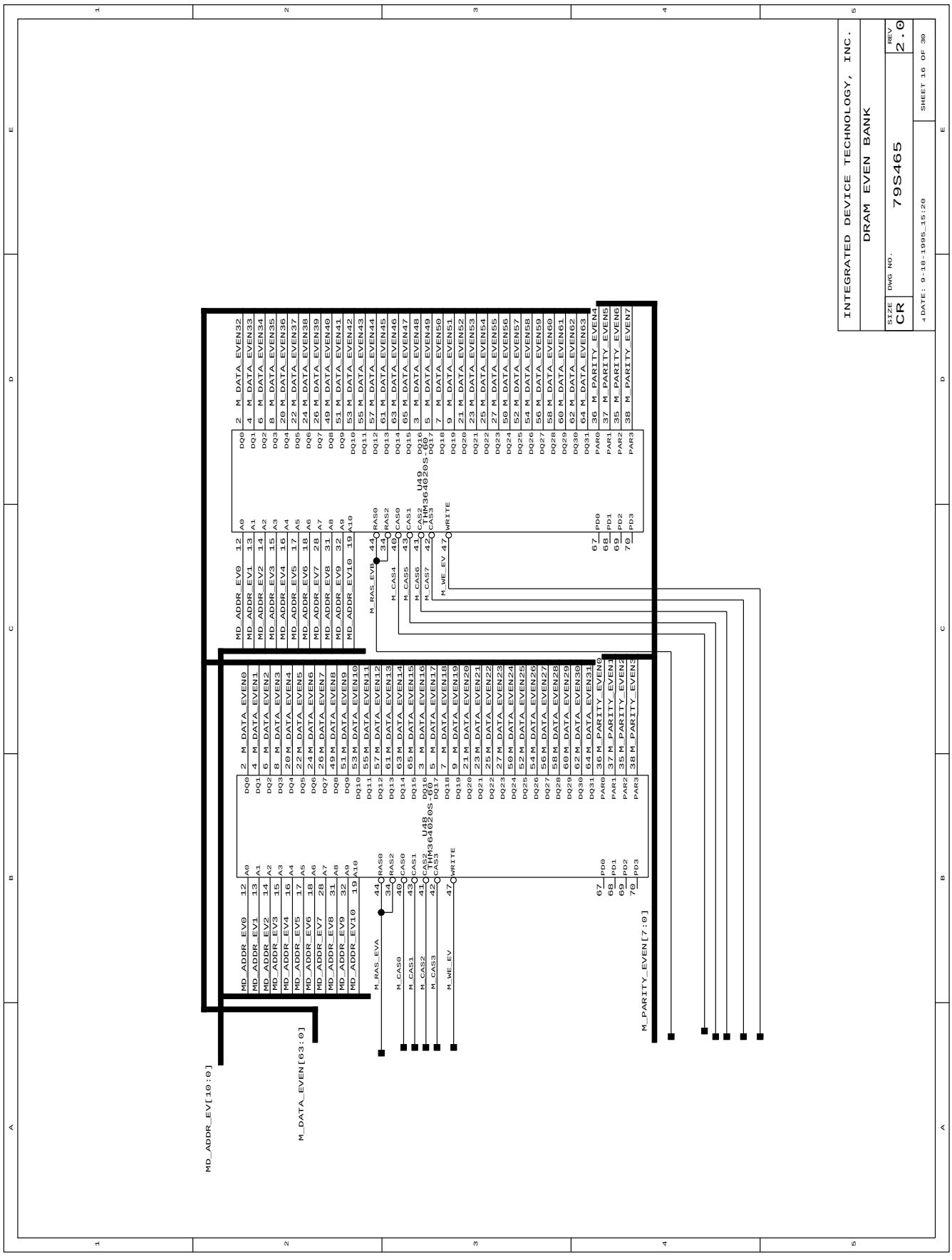

INTEGRATED DEVICE TECHNOLOGY, INC.
ZERO WAIT STATE SRAM MEMORY
ODD BANK
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 11 OF 30
M_DATA_ODD[63:0]
NOTE: The 64Kx32 Part Number is - IDT7MP4036S17M
NOTE: For R4650 32-bit SysIF Bus connect AB
(only one out of three)
For R4650 or R4600 64-bit SysIF Connect BC
SO_ADDR[24:5]
RA_SYSAD[31:0]
BSC_ADDR40
BSC_ADDR30
SO_ADDR19
SO_ADDR20
SO_ADDR21
SO_WE_N
SO_OE_N
GND
SADDR_LE
7MP4045
U3
U4
U8
162841ET

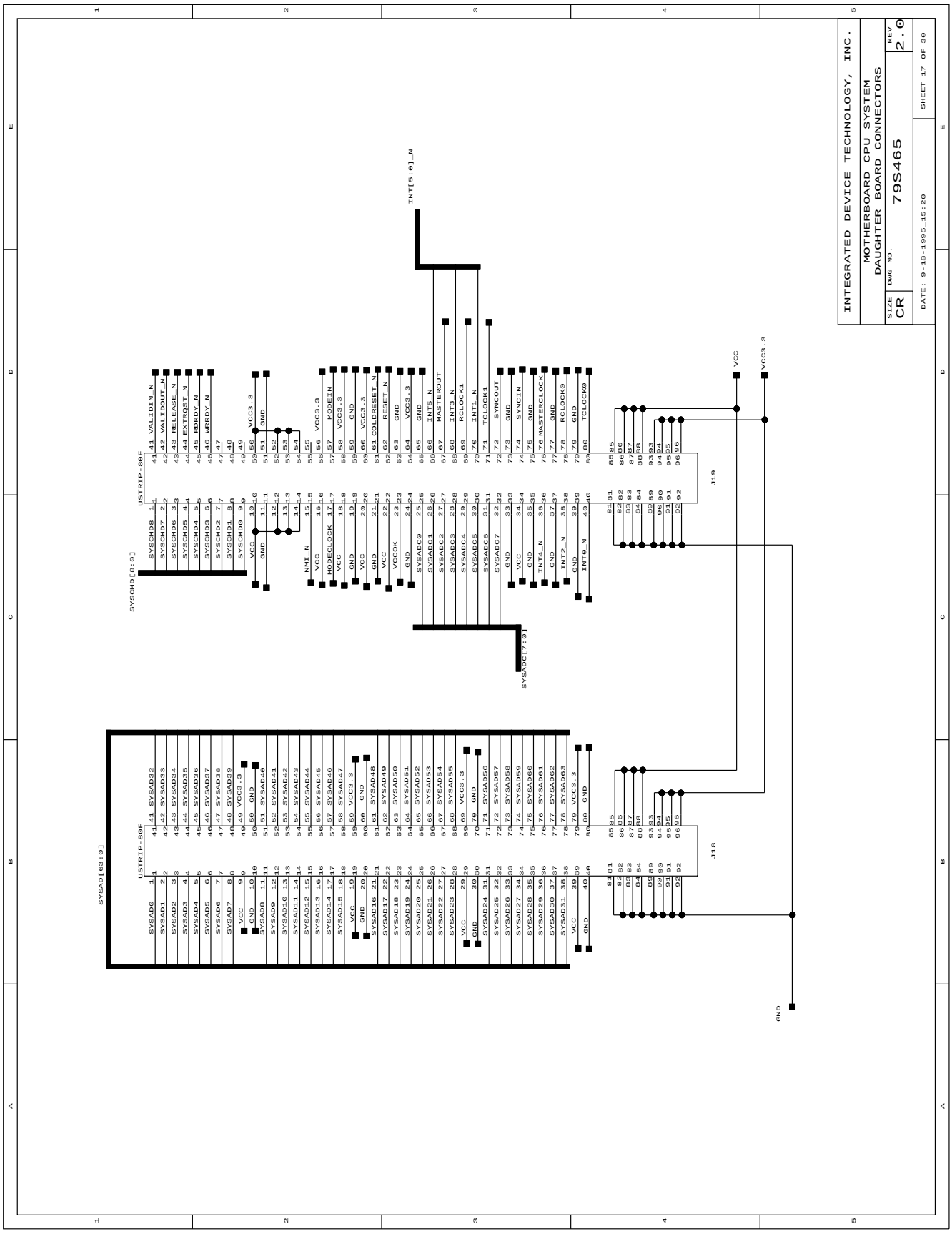

INTEGRATED DEVICE TECHNOLOGY, INC.
DRAM EVEN MUX
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 12 OF 30
Inputs and outputs of the termination res
so that it can be bypass
Leave the IC pads
RA_SYSAD[31:0]
MD_ADDR_EV[10:0]
MA_SEL
AD_MUX_LE
PUP7
PUP8
PUP5
PUP6
AD_MUX_LE
MSYSAD24_23
MSYSAD22_21
MSYSAD25_24
MSYSAD23_22
RA_SYSAD12
RA_SYSAD13
RA_SYSAD20
RA_SYSAD19
RA_SYSAD18
RA_SYSAD17
RA_SYSAD16
RA_SYSAD15
RA_SYSAD14
RA_SYSAD21
162260 U30
OE1B
LEA1B
LE1B
OE2B
LEA2B
LE2B
GND
OEA
D_ADDR_EV5
D_ADDR_EV4
D_ADDR_EV3
D_ADDR_EV2
D_ADDR_EV1
L_RA_SYSAD21
D_ADDR_EV0
4816-001
RN1 22-OHM
RN2 22-OHM
L_RA_SYSAD12
L_RA_SYSAD13
D_ADDR_EV10
D_ADDR_EV9
D_ADDR_EV8
D_ADDR_EV7
D_ADDR_EV6
FCT257DT
U45
MOD_A4
L_RA_SYSAD13
MOD_A_SEL
MSYSAD12_3
MSYSAD21_12
OE_N
GND

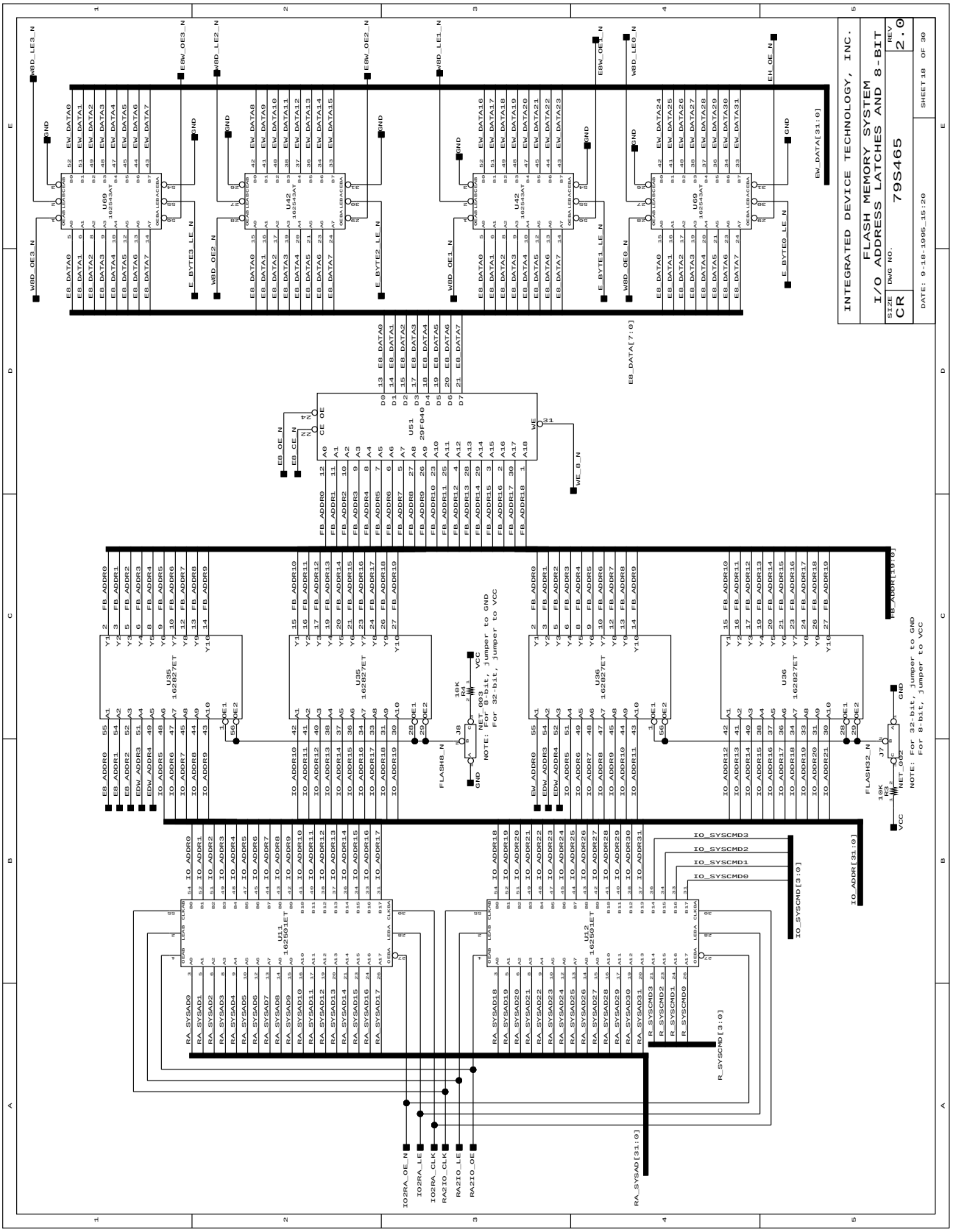

INTEGRATED DEVICE TECHNOLOGY, INC.
DRAM ODD BANK
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 13 OF 30
RA_SYSAD[31:0]
MD_ADDR_OD[10:0]
MSYSAD25_24
MSYSAD23_22
MSYSAD24_23
MSYSAD22_21
MSYSAD12_3
MSYSAD21_12
MOD_A4
L_RA_SYSAD13
MOD_A_SEL
AD_MUX_LE
PUP13
PUP14
PUP11
PUP12
PUP9
PUP10
PUP44
PUP43
PUP46
PUP45
MA_SEL
GND
U31
162260
4816-001
RN3
RN4
22-OHM
FCT257DT
U45
D_ADDR_OD0
D_ADDR_OD1
D_ADDR_OD2
D_ADDR_OD3
D_ADDR_OD4
D_ADDR_OD5
D_ADDR_OD6
D_ADDR_OD7
D_ADDR_OD8
D_ADDR_OD9
D_ADDR_OD10
MD_ADDR_OD0
MD_ADDR_OD1
MD_ADDR_OD2
MD_ADDR_OD3
MD_ADDR_OD4
MD_ADDR_OD5
MD_ADDR_OD6
MD_ADDR_OD7
MD_ADDR_OD8
MD_ADDR_OD9
MD_ADDR_OD10
RA_SYSAD11
RA_SYSAD10
RA_SYSAD9
RA_SYSAD8
RA_SYSAD7
RA_SYSAD6
RA_SYSAD5
RA_SYSAD20
RA_SYSAD19
RA_SYSAD18
RA_SYSAD17
RA_SYSAD16
RA_SYSAD15
RA_SYSAD14

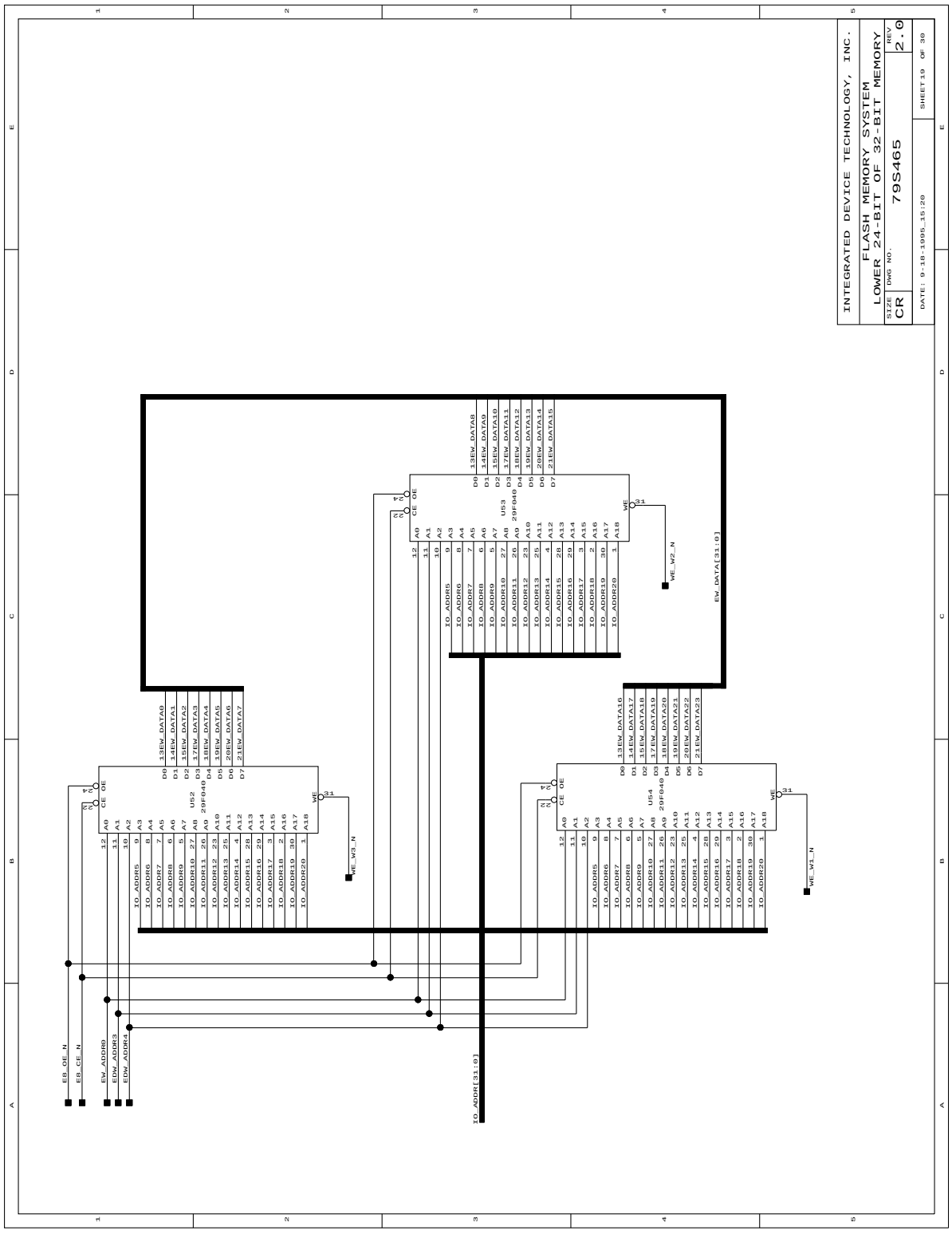

INTEGRATED DEVICE TECHNOLOGY, INC.
DRAM MUX MISC
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 14 OF 30
NOTE:
jumper B-C for 32-bit interface
jumper A-B for 64-bit interface
RA_SYSAD24
RA_SYSAD25
MSYSAD25_24
RA_SYSAD22
RA_SYSAD23
MSYSAD23_22
RA_SYSAD23
RA_SYSAD24
MSYSAD24_23
RA_SYSAD21
RA_SYSAD22
MSYSAD22_21
L_RA_SYSAD12
L_RA_SYSAD21
MSYSAD21_12
MOD_A3
L_RA_SYSAD12
MSYSAD12_3
U33
162244ET
GND
U32
RAS_ODA_N
RAS_ODB_N
RAS_EVA_N
RAS_EVB_N
RAS_OD0
RAS_OD1
RAS_EV0
RAS_EV1
4816-001
RN5
22-OHM
M_RAS_ODA
M_RAS_ODB
M_RAS_EVA
M_RAS_EVB
M_WE_EV
M_WE_OD
WE_E
WE_O
WE_EV_N
WE_OD_N
PUP37
PUP38
RN6
M_CAS0
M_CAS1
M_CAS2
M_CAS3
M_CAS4
M_CAS5
M_CAS6
M_CAS7
BCAS0
BCAS1
BCAS2
BCAS3
BCAS4
BCAS5
BCAS6
BCAS7
CAS0
CAS1
CAS2
CAS3
CAS4
CAS5
CAS6
CAS7
VCC
RP23
10K(R9)
PUP47
PUP46
PUP45
PUP44
PUP43
PUP37
PUP38
DRAM_ACCESS_N
INT3_N
RP7
10K(R9)
PUP18
PUP17
PUP16
PUP15
PUP14
PUP13
PUP12
PUP11
PUP10
RP8
10K(R9)
PUP9
PUP8
PUP7
PUP6
PUP5
PUP4
INT0_N
PUP2
IGNORE_ACCESS_N
RP8
10K(R9)
PUP27
PUP26
PUP25
PUP24
IO_ACCESS_N
PUP22
PUP21
PUP20
PUP19

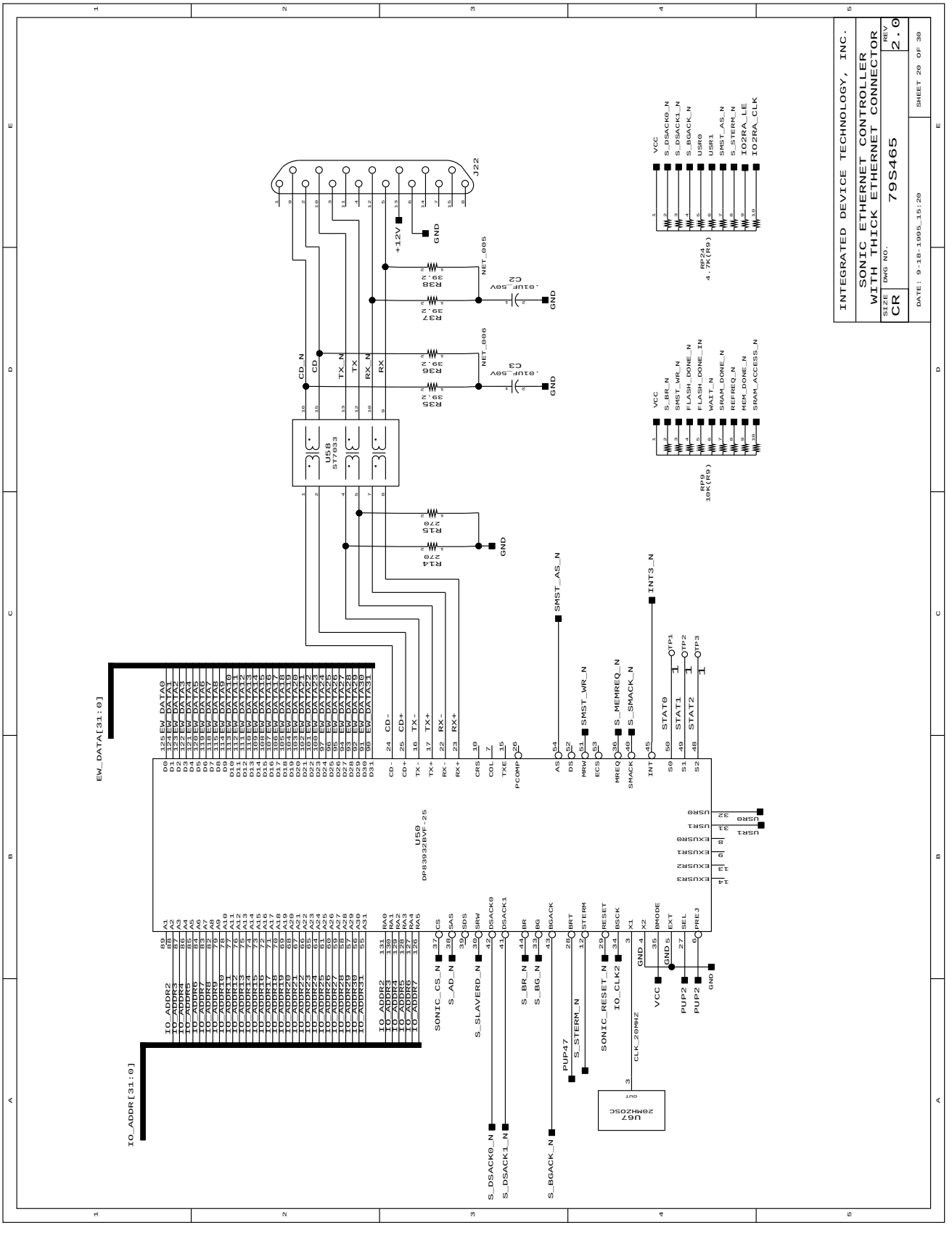

INTEGRATED DEVICE TECHNOLOGY, INC.
DRAM ODD BANK
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 15 OF 30
MD_ADDR_OD[10:0]
M_DATA_ODD[63:0]
M_PARITY_ODD[7:0]
U48
U47
TMM36402OS-60
M_RAS_ODA
M_RAS_ODB
M_CAS0
M_CAS1
M_CAS2
M_CAS3
M_CAS4
M_CAS5
M_CAS6
M_CAS7
M_WE_OD
WRITE

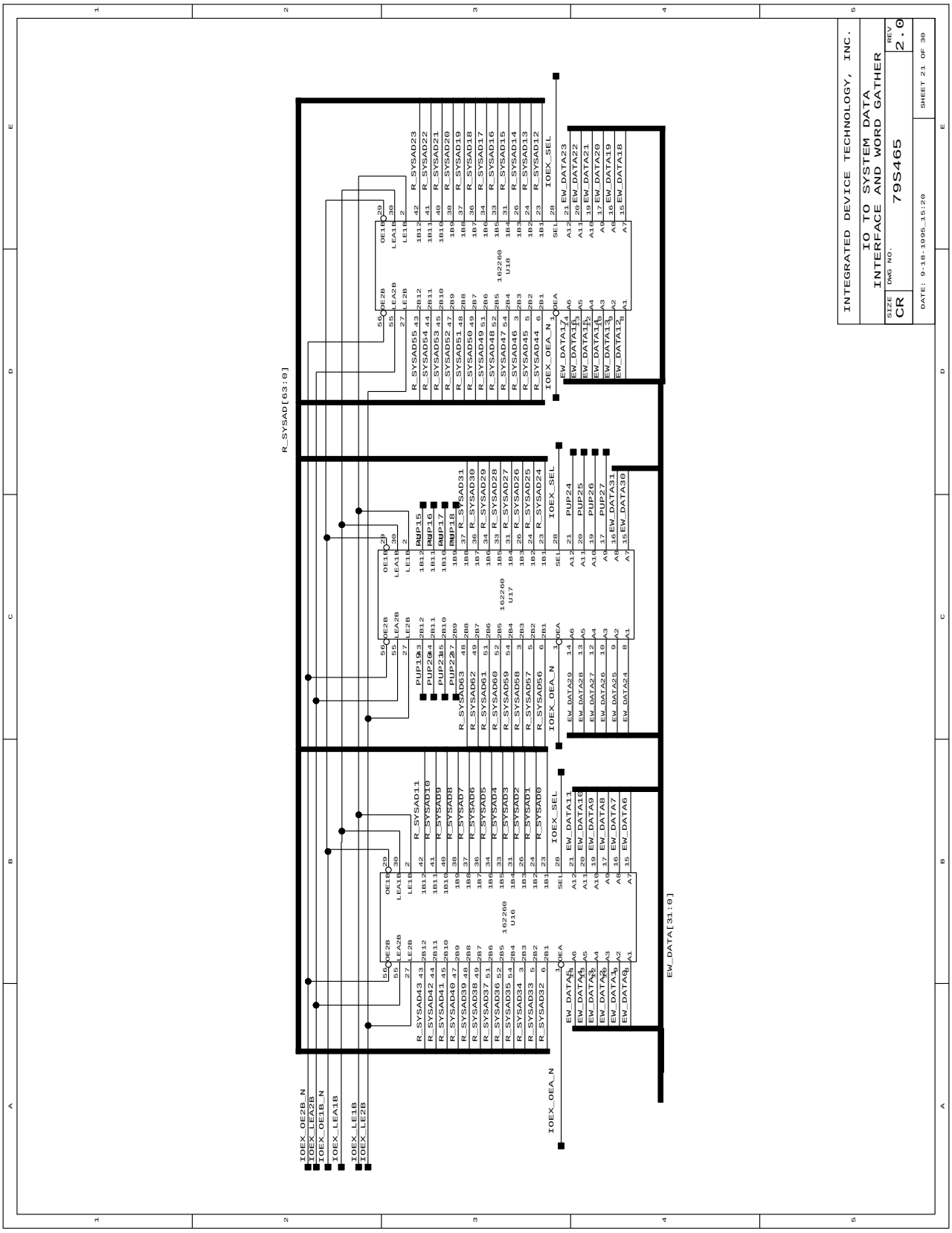

INTEGRATED DEVICE TECHNOLOGY, INC.
DRAM EVEN BANK
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 16 OF 30
MD_ADDR_EV[10:0]
M_DATA_EVEN[63:0]
M_PARITY_EVEN[7:0]
U48
U49
IDT79M3640202S
M_RAS_EVA
M_RAS_EV8
M_CAS0
M_CAS1
M_CAS2
M_CAS3
M_CAS4
M_CAS5
M_CAS6
M_CAS7
M_WE_EV
WRITE

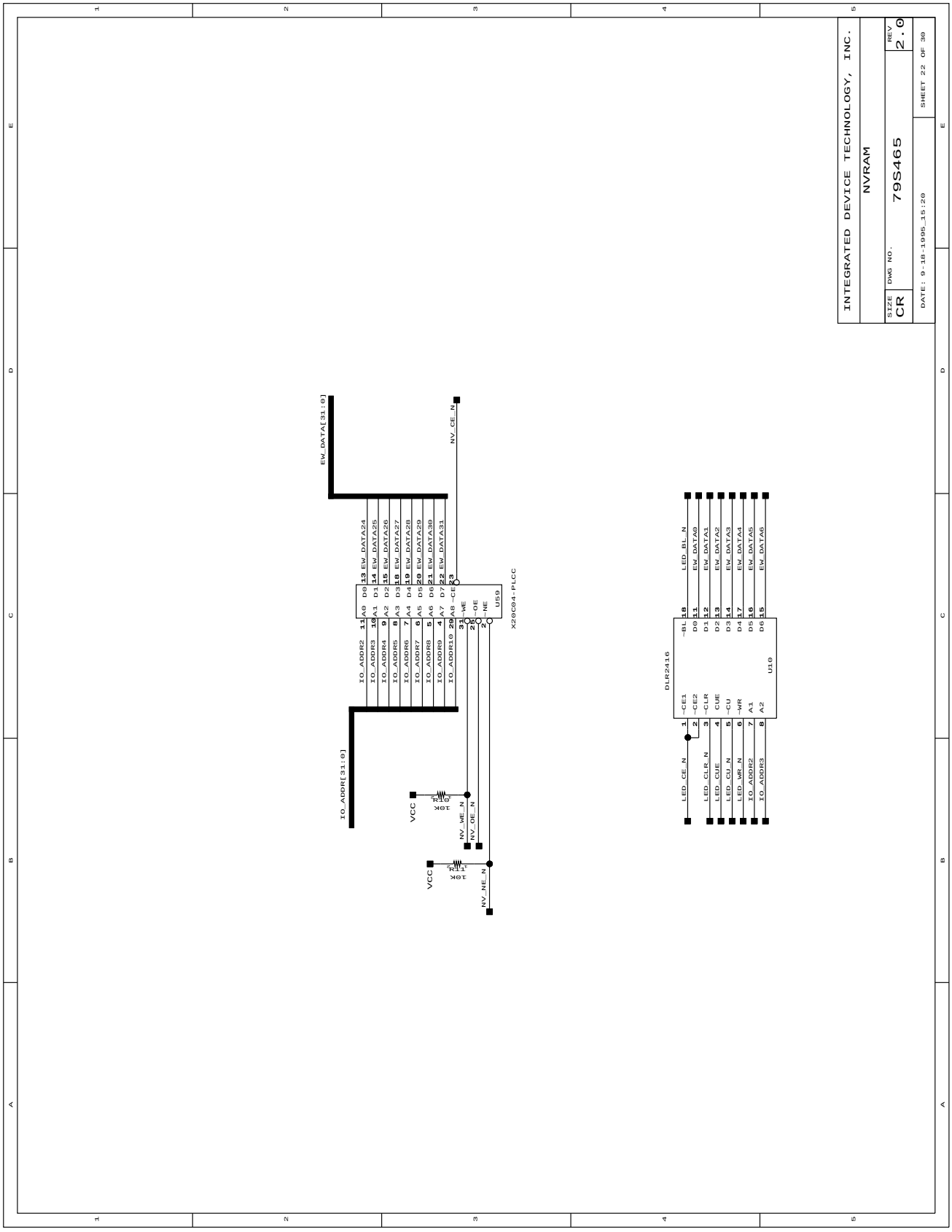

INTEGRATED DEVICE TECHNOLOGY, INC.
MOTHERBOARD CPU SYSTEM
DAUGHTER BOARD CONNECTORS
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 17 OF 30
SYSAD[63:0]
SYSCMD[8:0]
INT[5:0]_N
SYSADC[7:0]
USTRIP-80F
J18
J19
VCC
VCC3.3
GND

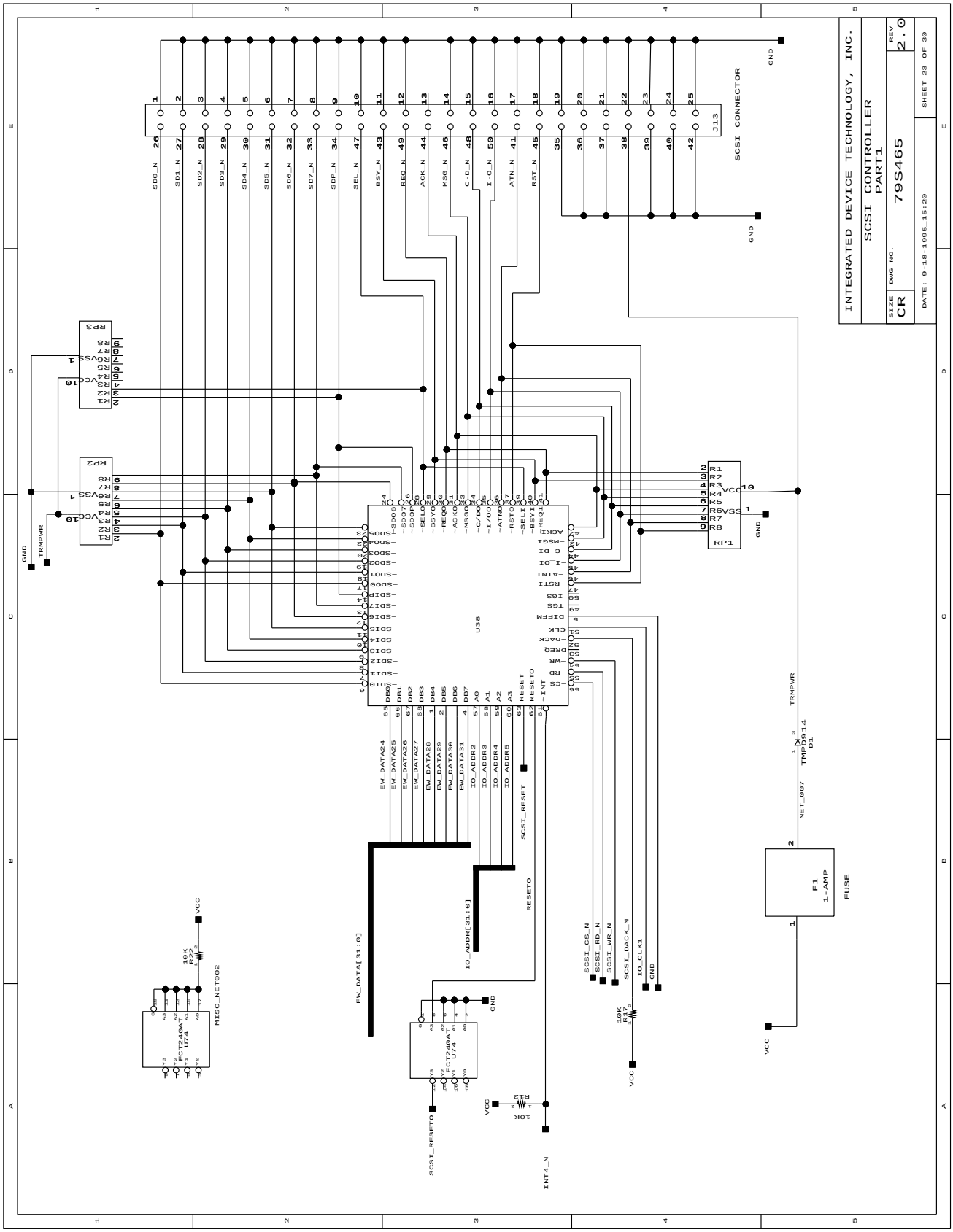

INTEGRATED DEVICE TECHNOLOGY, INC.
FLASH MEMORY SYSTEM
I/O ADDRESS LATCHES AND 8-BIT
79S465
SIZE CR
DWG NO.
REV. 2.0
DATE: 9-18-1995_15:20
SHEET 18 OF 30
EB_DATA[31:0]
EB_DATA[7:0]
FB_ADDR[19:0]
IO_ADDR[31:0]
IO_SYSCMD[3:0]
R_SYSCMD[3:0]
RA_SYSAD[31:0]
WE_8_N
FLASH8_N
FLASH32_N
NOTE: For 8-bit, jumper to GND
For 32-bit, jumper to VCC

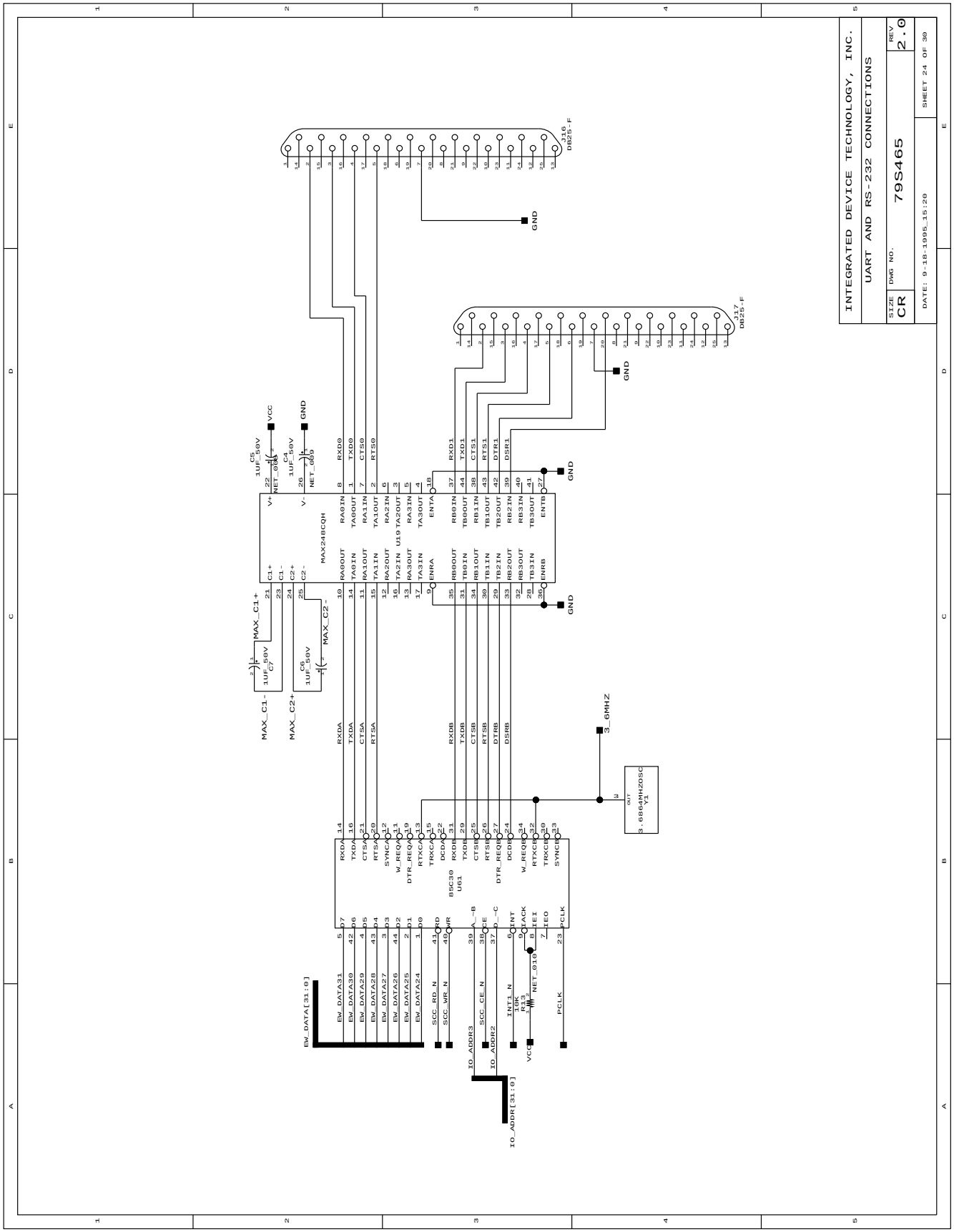

INTEGRATED DEVICE TECHNOLOGY, INC.
FLASH MEMORY SYSTEM
LOWER 24-BIT OF 32-BIT MEMORY
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 19 OF 30
U52 29F040
U53 29F040
U54 29F040
U55 29F040
EB_OE_N
EB_CE_N
EW_ADDR0
EW_ADDR3
EW_ADDR4
IO_ADDR[31:0]
EW_DATA[31:0]
WE_W1_N
WE_W2_N
WE_W3_N

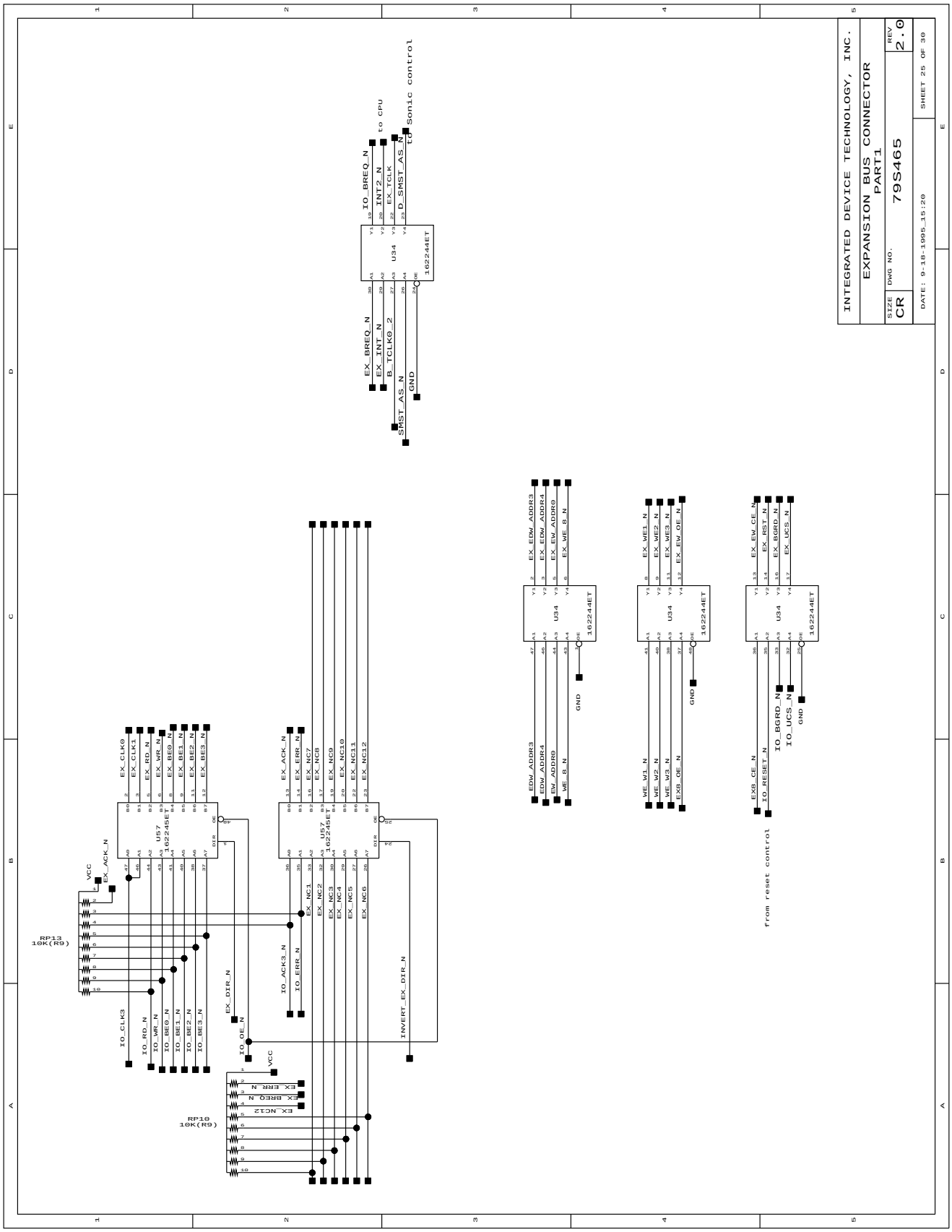

INTEGRATED DEVICE TECHNOLOGY, INC.
SONIC ETHERNET CONTROLLER
WITH THICK ETHERNET CONNECTOR
79S465
SIZE CR
DWG NO.
REV 2.0
DATE: 9-18-1995_15:20
SHEET 20 OF 30
U50 DP83932BVF-25
U58 ST7033
U67 20MHZOSC
J22
RP24 4.7K(09)
RP3 10K(09)
R35 39.2
R36 39.2
R37 39.2
R38 39.2
R14 270
R15 270
C2 .01UF_50V
C3 .01UF_50V
+12V
GND
VCC
EW_DATA[31:0]
IO_ADDR[31:0]
SONIC_CS_N
S_AD_N
S_SLAVERD_N
S_DSACK0_N
S_DSACK1_N
S_BR_N
S_BG_N
S_BGACK_N
PUP47
S_STERM_N
SONIC_RESET_N
IO_CLK2
CLK_20MHZ
PUP2
SMST_AS_N
SMST_WR_N
S_MEMREQ_N
S_SMACK_N
INT3_N
S_DSACK0_N
S_DSACK1_N
S_BGACK_N
S_BR_N
SMST_WR_N
FLASH_DONE_N
FLASH_DONE_IN
WAIT_N
SRAM_DONE_N
REFREQ_N
MEM_DONE_N
SRAM_ACCESS_N
S_STERM_N
IO2RA_LE
IO2RA_CLK
USR0
USR1
TP1
TP2
TP3

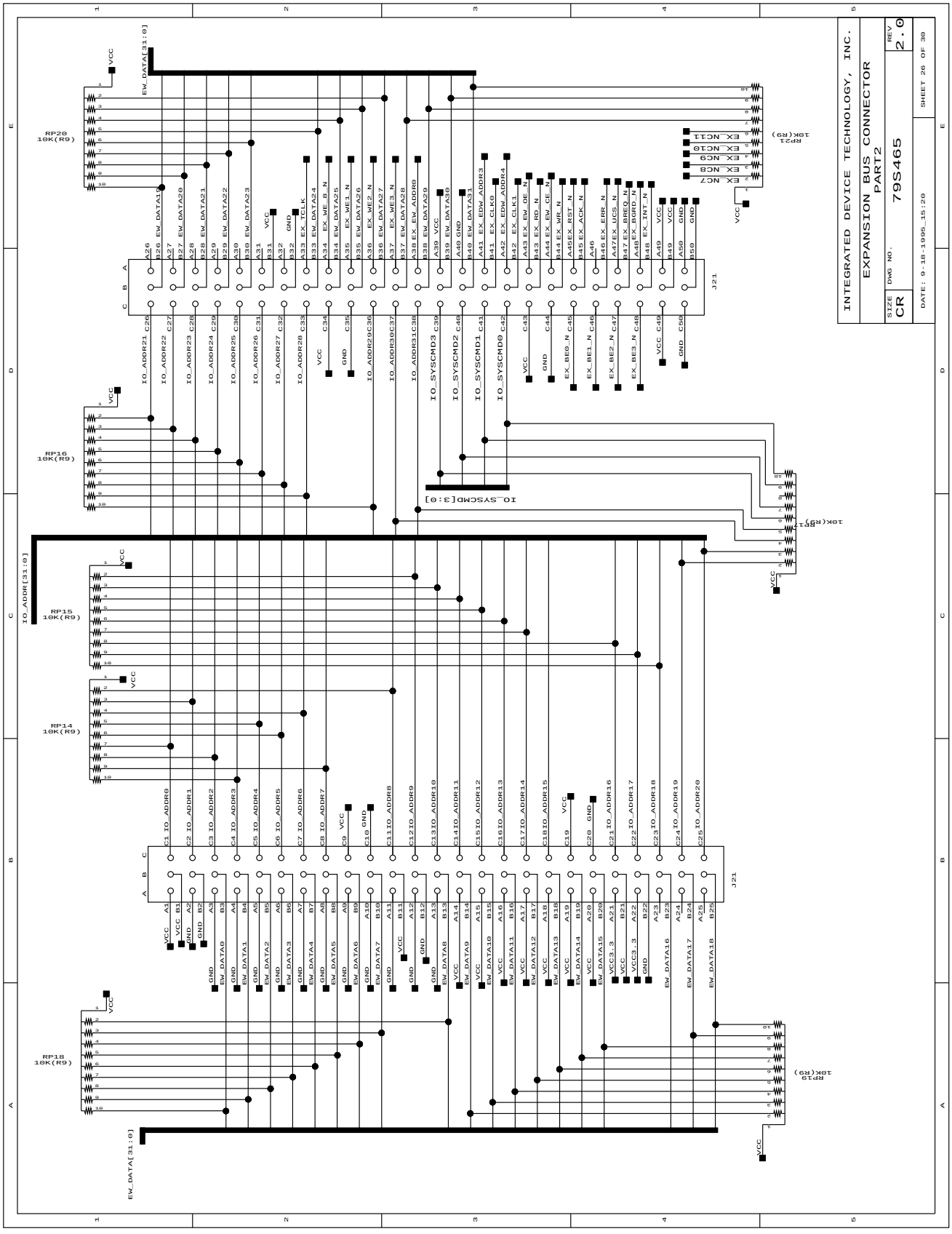

INTEGRATED DEVICE TECHNOLOGY, INC.
IO TO SYSTEM DATA
INTERFACE AND WORD GATHER
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 21 OF 30
R_SYSAD[63:0]
EW_DATA[31:0]
IOEX_OE2B_N
IOEX_LEA2B
IOEX_OE1B_N
IOEX_LEA1B
IOEX_LE1B
IOEX_LE2B
IOEX_OEA_N
IOEX_SEL
PUP15
PUP16
PUP17
PUP18
PUP19
PUP20
PUP21
PUP22
PUP23
PUP24
PUP25
PUP26
PUP27
162260 U16
162260 U17
162260 U18

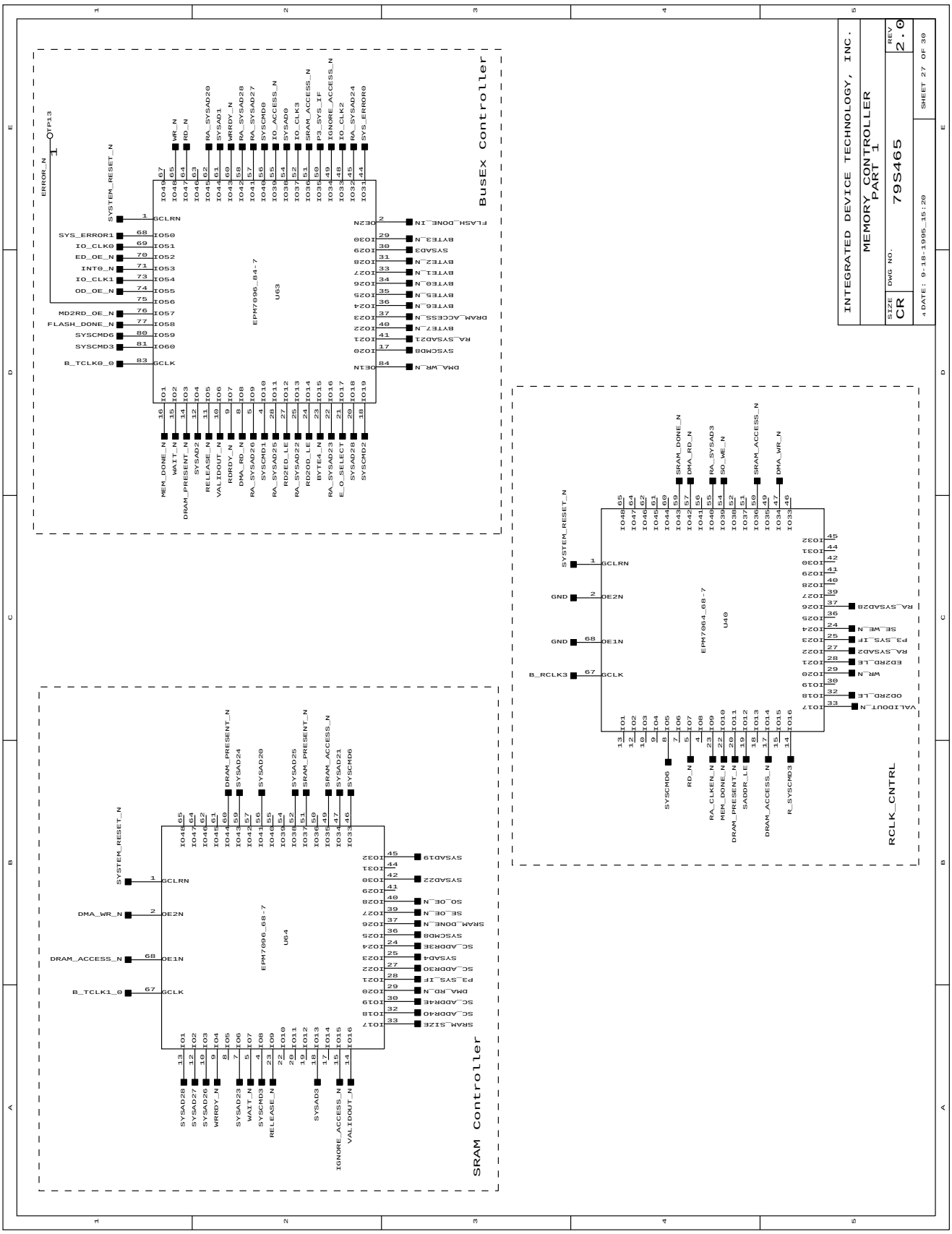

INTEGRATED DEVICE TECHNOLOGY, INC.
NVRAM
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 22 OF 30
EW_DATA[31:0]
NV_CE_N
IO_ADDR[31:0]
X28C04-PLCC
U59
VCC
10K
NV_WE_N
NV_OE_N
NV_NE_N
DLR2416
U10
LED_CE_N
LED_CLR_N
LED_CUE
LED_CU_N
LED_WR_N
IO_ADDR2
IO_ADDR3
LED_BL_N
EW_DATA0
EW_DATA1
EW_DATA2
EW_DATA3
EW_DATA4
EW_DATA5
EW_DATA6

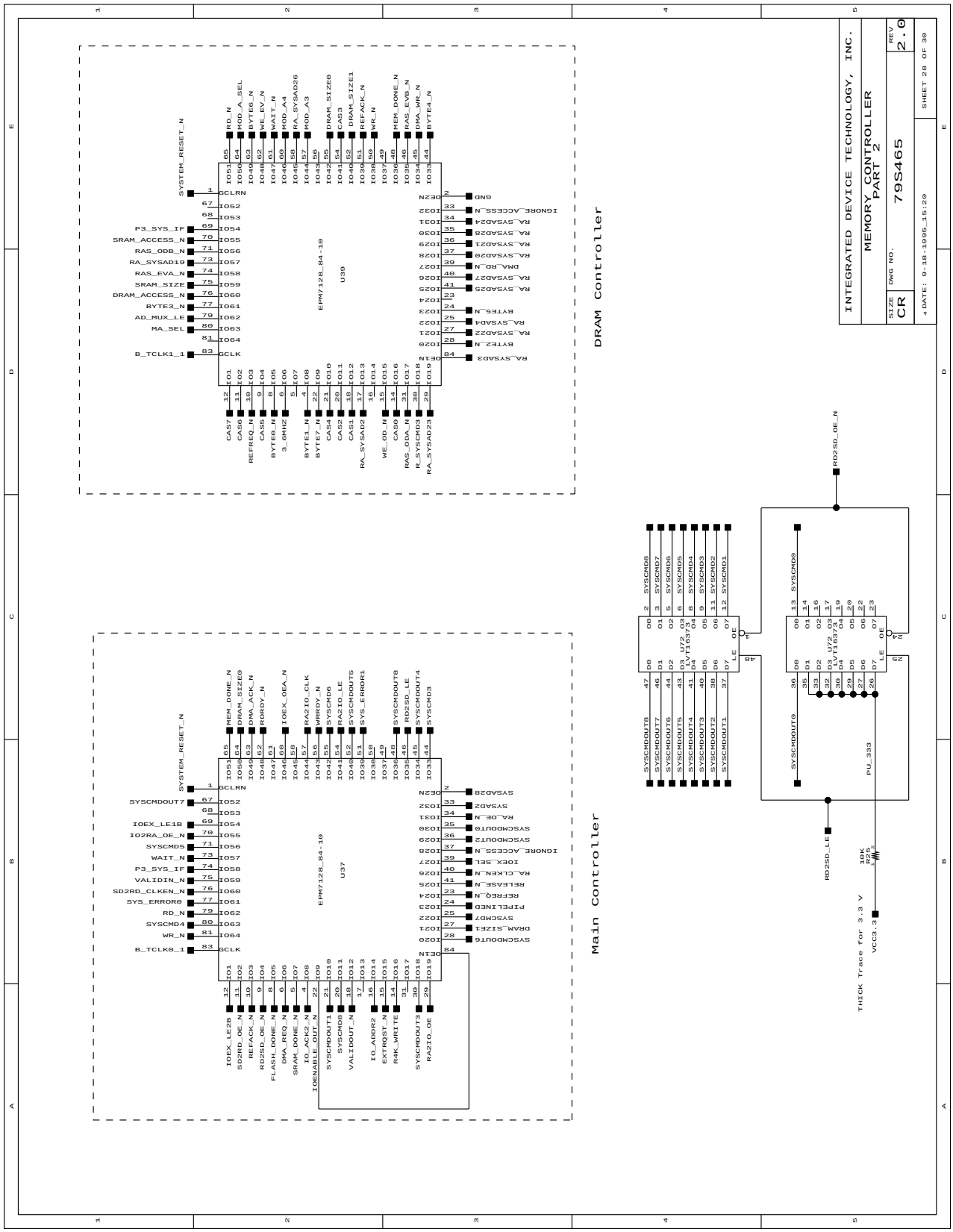

INTEGRATED DEVICE TECHNOLOGY, INC.
SCSI CONTROLLER PART1
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 23 OF 30
SCSI CONNECTOR
J13
U38
RP1
RP2
RP3
U74
U75
F1 1-AMP FUSE
D1 TMPD914
R12 10K
R22 10K
R23 10K
MISC_NET002
TRMPWR
GND
VCC
SCSI_RESET
RESET0
INT4_N
IO_ADDR[31:0]
EW_DATA[31:0]
SCSI_CS_N
SCSI_RD_N
SCSI_WR_N
SCSI_DACK_N
IO_CLK1

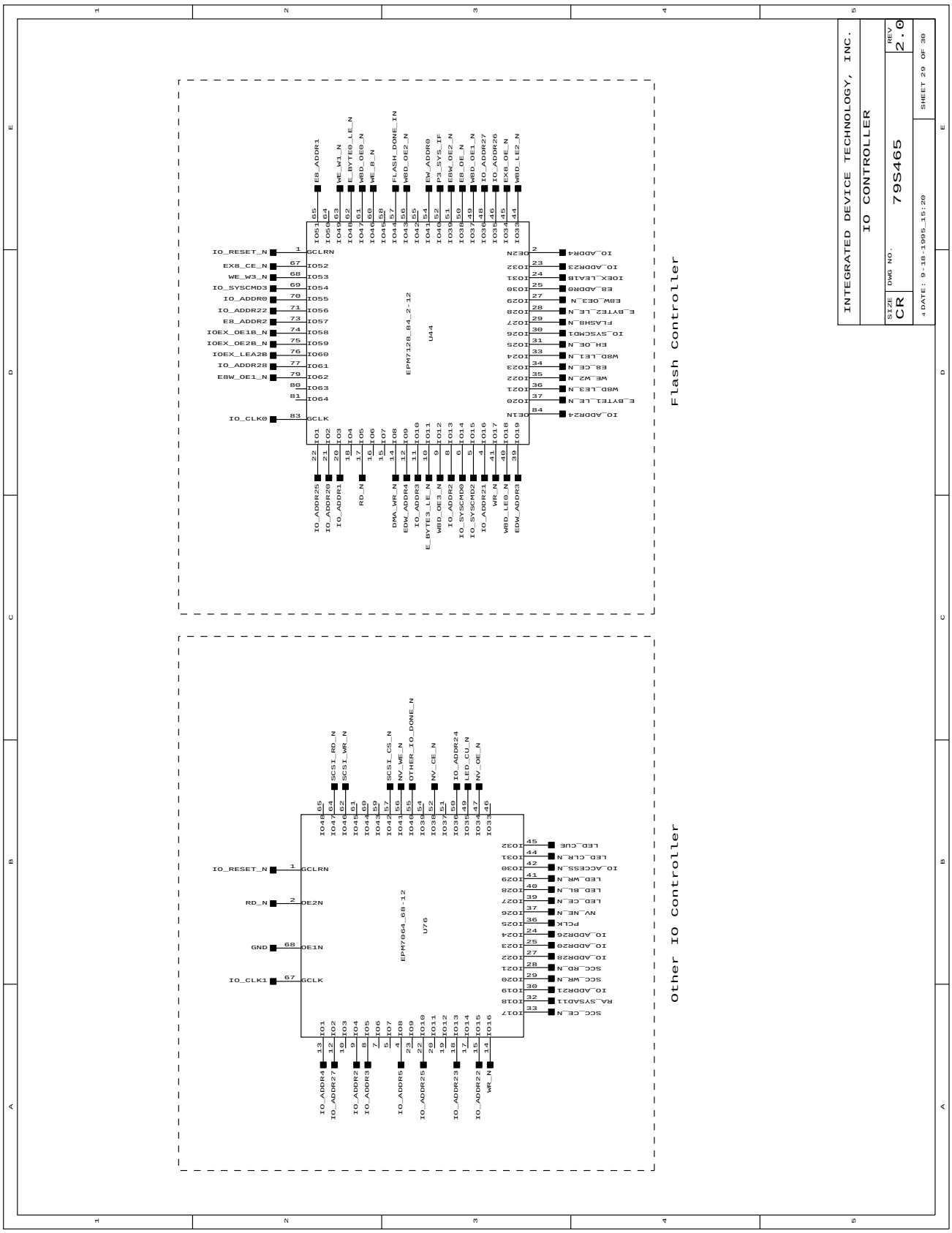

INTEGRATED DEVICE TECHNOLOGY, INC.
UART AND RS-232 CONNECTIONS
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 24 OF 30
MAX248CQH
SCC85C30
3.6864MHZOSC
DB25-F
GND
VCC
EW_DATA[31:0]
IO_ADDR[31:0]

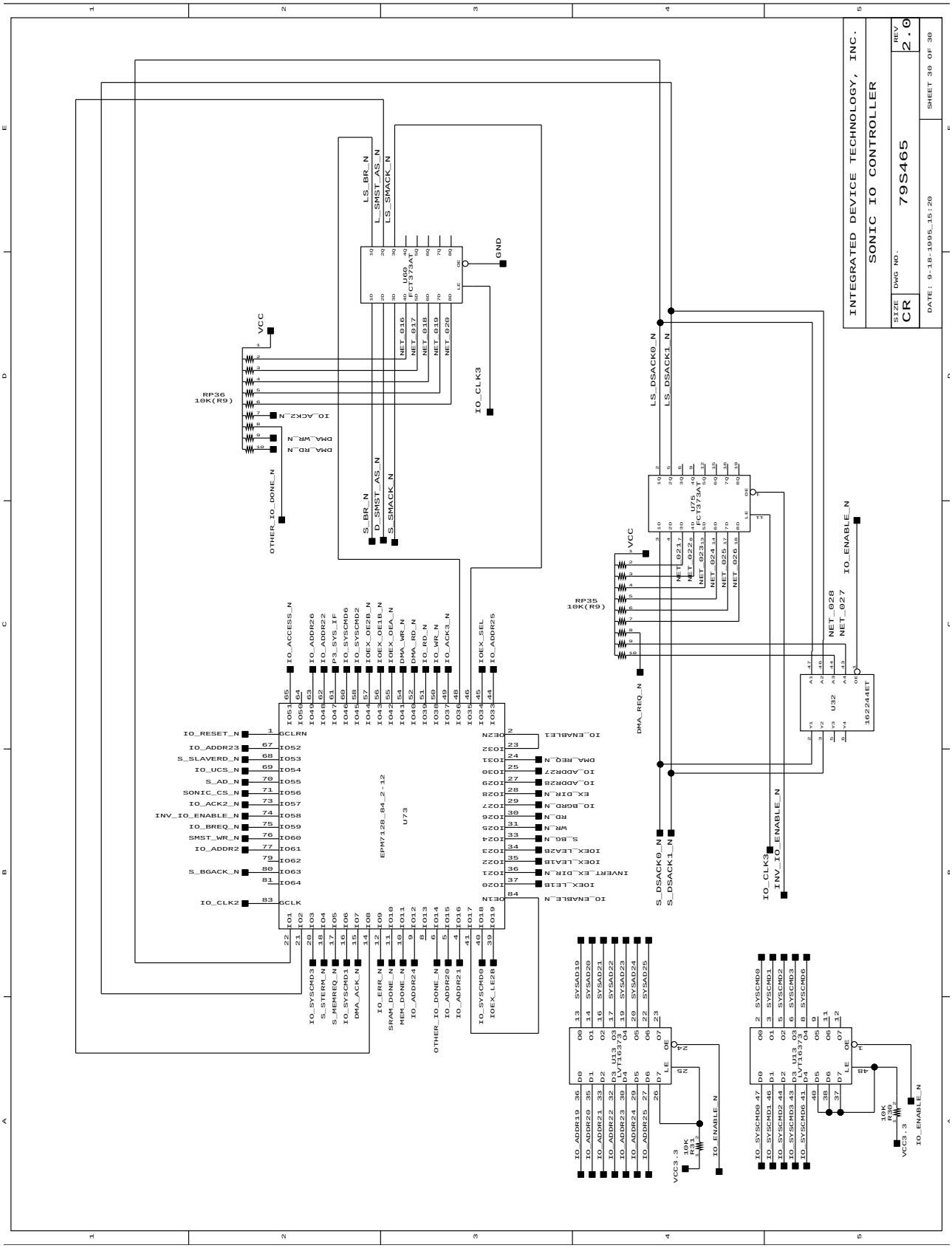

INTEGRATED DEVICE TECHNOLOGY, INC.
EXPANSION BUS CONNECTOR
PART1
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 25 OF 30
IO_BREQ_N
INT2_N
EX_TCLK
D_SMST_AS_N
to CPU
to Sonic control
U34
16244ET
EX_BREQ_N
EX_INT_N
B_TCLK0_2
SMST_AS_N
GND
EX_EDW_ADDR3
EX_EDW_ADDR4
EX_EW_ADDR0
EX_WE_0_N
EDW_ADDR3
EDW_ADDR4
EW_ADDR0
WE_0_N
EX_WE1_N
EX_WE2_N
EX_WE3_N
EX_EW_OE_N
WE_W1_N
WE_W2_N
WE_W3_N
EX8_OE_N
EX_EW_CE_N
EX_RST_N
EX_BGRD_N
EX_UCS_N
EX8_CE_N
TO_RESET_N
IO_BGRD_N
IO_UCS_N
from reset control
EX_CLK0
EX_CLK1
EX_RD_N
EX_WR_N
EX_BE0_N
EX_BE1_N
EX_BE2_N
EX_BE3_N
U57
162245ET
EX_ACK_N
EX_ERR_N
EX_NC7
EX_NC8
EX_NC9
EX_NC10
EX_NC11
EX_NC12
EX_NC1
EX_NC2
EX_NC3
EX_NC4
EX_NC5
EX_NC6
VCC
RP13
10K(R9)
RP10
10K(R9)
IO_CLK3
IO_RD_N
IO_WR_N
IO_BE0_N
IO_BE1_N
IO_BE2_N
IO_BE3_N
IO_ACK3_N
IO_ERR_N
EX_DIR_N
IO_OE_N
INVERT_EX_DIR_N
EX_ERR_N
EX_BREQ_N
EX_NC12


INTEGRATED DEVICE TECHNOLOGY, INC.
EXPANSION BUS CONNECTOR
PART2
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 26 OF 30
EW_DATA[31:0]
IO_ADDR[31:0]
IO_SYSCMD[3:0]
RP20 10K(R9)
RP16 10K(R9)
RP15 10K(R9)
RP14 10K(R9)
RP18 10K(R9)
RP21 10K(R9)
RP17 10K(R9)
RP19 10K(R9)
J21


INTEGRATED DEVICE TECHNOLOGY, INC.
MEMORY CONTROLLER
PART 1
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 27 OF 30
BusEx Controller
SRAM Controller
RCLK_CNTRL
EPM7096_84-7
U63
EPM7096_68-7
U64
EPM7064_68-7
U40
SYSTEM_RESET_N
ERROR_N
OTP13
SYS_ERROR1
IO_CLK0
ED_OE_N
INT0_N
IO_CLK1
OD_OE_N
MD2RD_OE_N
FLASH_DONE_N
SYSCMD6
SYSCMD3
B_TCLK0_0
MEM_DONE_N
WAIT_N
DRAM_PRESENT_N
SYSAD2
RELEASE_N
VALIDOUT_N
RDRDY_N
DMA_RD_N
RA_SYSAD26
SYSCMD1
RA_SYSAD25
RD2ED_LE
RA_SYSAD22
RD2OD_LE
BYTE4_N
RA_SYSAD23
E_O_SELECT
SYSAD28
SYSCMD2
WR_N
RD_N
RA_SYSAD20
SYSAD1
WRRDY_N
RA_SYSAD28
RA_SYSAD27
SYSCMD0
IO_ACCESS_N
SYSAD0
IO_CLK3
SRAM_ACCESS_N
P3_SYS_IF
IGNORE_ACCESS_N
IO_CLK2
RA_SYSAD24
SYS_ERROR0
FLASH_DONE_IN
BYTE3_N
SYSAD3
BYTE2_N
BYTE1_N
BYTE0_N
BYTE5_N
BYTE6_N
DRAM_ACCESS_N
BYTE7_N
RA_SYSAD21
SYSCMD8
DMA_WR_N
SRAM_DONE_N
DMA_RD_N
RA_SYSAD3
SO_WE_N
SRAM_ACCESS_N
DMA_WR_N
RA_SYSAD28
SE_WE_N
P3_SYS_IF
RA_SYSAD2
ED2RD_LE
WR_N
OD2RD_LE
VALIDOUT_N
SYSCMD6
RD_N
RA_CLKEN_N
MEM_DONE_N
DRAM_PRESENT_N
SADDR_LE
DRAM_ACCESS_N
R_SYSCMD3
SYSAD28
SYSAD27
SYSAD26
WRRDY_N
SYSAD23
WAIT_N
SYSCMD3
RELEASE_N
SYSAD3
IGNORE_ACCESS_N
VALIDOUT_N
SYSAD24
SYSAD20
SYSAD25
SYSAD21
SYSCMD6
SYSAD19
SYSAD22
SO_OE_N
SE_OE_N
SRAM_DONE_N
SYSCMD8
SC_ADDR3E
SYSAD4
SC_ADDR3O
P3_SYS_IF
DMA_RD_N
SC_ADDR4E
SC_ADDR4O
SRAM_SIZE


INTEGRATED DEVICE TECHNOLOGY, INC.
MEMORY CONTROLLER
PART 2
79S465
SIZE CR
DWG NO.
REV 2.0
DATE: 9-18-1995_15:20
SHEET 28 OF 30
DRAM Controller
Main Controller
EPM7128_84-10
U39
U37
SYSTEM_RESET_N
B_TCLK1_1
B_TCLK0_1
RD2SD_OE_N
RD2SD_LE
THICK Trace for 3.3 V
VCC3.3
PU_333
10K
R25


INTEGRATED DEVICE TECHNOLOGY, INC.
IO CONTROLLER
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995 15:20
SHEET 29 OF 30
Flash Controller
Other IO Controller
U44
EPM7128_84_2-12
U76
EPM7064_68-12
IO_RESET_N
EX8_CE_N
WE_W3_N
IO_SYSCMD3
IO_ADDR0
IO_ADDR22
E8_ADDR2
IOEX_OE1B_N
IOEX_OE2B_N
IOEX_LEA2B
IO_ADDR28
E8W_OE1_N
IO_CLK0
IO_CLK1
RD_N
GND
IO_ADDR20
IO_ADDR21
IO_ADDR25
IO_ADDR26
IO_ADDR27
IO_ADDR2
IO_ADDR3
IO_ADDR4
IO_ADDR5
IO_ADDR22
IO_ADDR23
WR_N
WBD_LE0_N
EDW_ADDR3
E_BYTE3_LE_N
WBD_OE3_N
EDW_ADDR4
DMA_WR_N
SCSI_RD_N
SCSI_WR_N
SCSI_CS_N
NV_WE_N
OTHER_IO_DONE_N
NV_CE_N
IO_ADDR24
LED_CU_N
NV_OE_N
E8_ADDR1
WE_W1_N
E_BYTE0_LE_N
WBD_OE0_N
WE_8_N
FLASH_DONE_IN
WBD_OE2_N
EDW_ADDR0
P3_SYS_IF
E8W_OE2_N
E8_OE_N
WBD_OE1_N
IO_ADDR27
IO_ADDR26
EX8_OE_N
WBD_LE2_N
IO_ADDR24
IO_ADDR23
E8W_OE3_N
E8_ADDR0
IO_ADDR22
IO_ADDR21
E_BYTE2_LE_N
FLASH8_N
IO_SYSCMD1
E8_OE_N
WBD_LE1_N
E8_CE_N
WE_W2_N
WBD_LE3_N
E_BYTE1_LE_N
IO_ADDR20
LED_CUE
LED_CLR_N
IO_ACCESS_N
LED_WR_N
LED_BL_N
LED_CE_N
NV_NE_N
PCLK
IO_ADDR26
IO_ADDR20
IO_ADDR28
SCC_RD_N
SCC_WR_N
IO_ADDR21
RA_SYSAD11
SCC_CE_N


INTEGRATED DEVICE TECHNOLOGY, INC.
SONIC IO CONTROLLER
SIZE CR
DWG NO. 79S465
REV 2.0
DATE: 9-18-1995_15:20
SHEET 30 OF 30
U73
EPM7128_84_2-12
U69
FCT373AT
U75
FCT373AT
U32
162244ET
RP36 10K(R9)
RP35 10K(R9)
U13
LVT16373
LS_BR_N
L_SMST_AS_N
LS_SMACK_N
S_BR_N
D_SMST_AS_N
S_SMACK_N
LS_DSACK0_N
LS_DSACK1_N
S_DSACK0_N
S_DSACK1_N
IO_CLK3
INV_IO_ENABLE_N
IO_ENABLE_N
DMA_REQ_N
OTHER_IO_DONE_N
IO_ACK2_N
DMA_RD_N
DMA_WR_N
VCC
GND
NET_016
NET_017
NET_018
NET_019
NET_020
NET_021
NET_022
NET_023
NET_024
NET_025
NET_026
NET_027
NET_028
IO_RESET_N
IO_ADDR23
S_SLAVERD_N
IO_UCS_N
S_AD_N
SONIC_CS_N
IO_ACK2_N
INV_IO_ENABLE_N
IO_BREQ_N
SMST_WR_N
IO_ADDR2
S_BGACK_N
IO_CLK2
IO_ACCESS_N
IO_ADDR26
IO_ADDR22
P3_SYS_IF
IO_SYSCMD6
IO_SYSCMD2
IOEX_OE2B_N
IOEX_OE1B_N
IOEX_OEA_N
DMA_WR_N
DMA_RD_N
IO_RD_N
IO_WR_N
IO_ACK3_N
IOEX_SEL
IO_ADDR25
IO_ENABLE1
DMA_REQ_N
IO_ADDR27
IO_ADDR28
EX_DIR_N
IO_BGRD_N
RD_N
WR_N
S_B0_N
IOEX_LEA2B
IOEX_LEA1B
INVERT_EX_DIR_N
IOEX_LE1B
IO_ENABLE_N
IO_ERR_N
S_TERM_N
S_MEMREQ_N
IO_SYSCMD1
DMA_ACK_N
SRAM_DONE_N
MEM_DONE_N
IO_ADDR24
OTHER_IO_DONE_N
IO_ADDR20
IO_ADDR21
IO_SYSCMD0
IOEX_LE2B
IO_SYSCMD3
S_TERM_N
S_MEMREQ_N
IO_SYSCMD1
DMA_ACK_N
IO_ADDR19
IO_ADDR20
IO_ADDR21
IO_ADDR22
IO_ADDR23
IO_ADDR24
IO_ADDR25
SYSAD19
SYSAD20
SYSAD21
SYSAD22
SYSAD23
SYSAD24
SYSAD25
IO_SYSCMD0
IO_SYSCMD1
IO_SYSCMD2
IO_SYSCMD3
IO_SYSCMD6
10K
VCC3.3

PLA Equations

This chapter contains the FPGA listing for the Reset Controller. Listings for the Bus Controller, DRAM Controller, Flash State Machine, Main, RCLK, SONIC, SRAM, I/O, and Reset Controllers are included in the *Addendum to IDT79S465 Evaluation Board User's Manual.*

Reset Controller

MAX+PLUS II AHDLv5.0

```
EDIT HISTORY
Date            Engineer     Check Sum         Changes
-------------------------------------------------------------
10/12/1994      R.Cummings                     Initial Release
%

TITLE “Reset Controller PLD”;

%=============================================================================%
%The Reset Controller is used ....

%

%========================== CONSTANTS ==================================%

%=============================================================================%
SUBDESIGN Reset
(
% Input signals from CPU%
MASTEROUT:INPUT;% The Global clock for the controller.  Used to %
% generate CPU Reset/ and Cold_reset/ signals%
MODECLOCK:INPUT;% CPU ModeClock used for control of Mode bits%

% Other input signals%
RESET_IN/:INPUT;% The reset signal from the TL7705B reset chip%
DRAM_SIZE[1..0]:INPUT;% Indicates if DRAM installed%
%DRAM_SIZE[1..0]Description%
%11No DRAM%
%00, 01, 10DRAM installed%
P3_SYS_IF:INPUT;% Indicates size of P3 sys I/F%
%P3_SYS_IFDescription%
%032-bit interface%
%164-bit interface%
R4K_WRITE:INPUT;% Indicates to use R4K compatible writes for %
% non-block writes from the CPU%
%R4K_WRITEDescription%
%0R4K write compatible mode%
%1either pipelined or re-issue %
PIPELINED:INPUT;% Indicates which of the new write mode if not%
% using R4K compatible write mode - non-block %
%PIPELINEDDescription%
%0Pipelined write mode%
%1Write re-issue mode%
DRIVE:INPUT;% Indicates drive strength for output%
%DRIVEDescription%
%0100 percent drive%
%183 percent drive%
TIMER_INT:INPUT;% Indicates if Timer interrupt is enabled%
%TIMER_INTDescription%
%0Timer interrupt enabled%
%1Timer interrupt disabled%
BIG_ENDIAN:INPUT;% Indicates if a big or little endian system%
%BIG_ENDIANDescription%
%0Little endian system%
%1Big endian system%
CLK_DIV[2..0]:INPUT;% Indicates the clock divisor for the System%
% interface clocks for R4600/R4700 or the clock%
% multiplier for a R4650 system%
%CLK_DIV[2..0]Divisor (multiplier)%
%0002%
%0013%
%0104%
%0115%
%1006%
%1017%
%1108%
```

```
%111Reserved%

% RESET control output signals %
DRAM_PRESENT/:OUTPUT;% Indicates that DRAM is installed in system%
%DRAM_PRESENT/Description%
%0DRAM Installed in system%
%1No DRAM in system%
MODEIN:OUTPUT;% The Mode bit stream to the CPU%
VCCOK:OUTPUT;% Indicates to the CPU that power is applied%
RESET/:OUTPUT;% The CPU Reset* signal used for all resets%
COLD_RESET/:OUTPUT;% The CPU ColdReset* signal, used for power on %
% and cold resets%
SYSTEM_RESET/:OUTPUT;% The reset signal to most of the system except%
% the SONIC, SCSI and IO sub-systems%
SONIC_RESET/:OUTPUT;% The reset for the SONIC %
SCSI_RESET:OUTPUT;% The reset for the SCSI chip (53C90A)%
IO_RESET/:OUTPUT;% The reset for the other IO devices%
)
%============================================================================%
% Bus Exchanger control state machine %

VARIABLE

master_sm:MACHINE WITH STATES
(mst0, mst1, mst2, mst3);

%--------------  Output Flip Flops  ----------------------%
DRAM_PRESENT/:DFF;
MODEIN:DFF;
VCCOK:DFF;
RESET/:DFF;
COLD_RESET/:DFF;
SYSTEM_RESET/:DFF;
SONIC_RESET/:DFF;
SCSI_RESET:DFF;
IO_RESET/:DFF;

%----------------------------------------------------------------------------%
% Buried nodes %
Mode_done/:NODE;% Indicates to the mater control that the mode%
% bits are all done and allows it to start%
master_count[6..0]:DFF;% masterout counter for CPU reset signals%
mode_count[7..0]:DFF;% modeclock counter for mode bit control%
divisor[7..5]:NODE;% the clock divisor mode bits%
Byte_order:NODE;% the CPU endianness%
Write_pat:NODE;% selects the block write-back pattern%
R4k_wrt:NODE;% selects the R4k write mode or not%
Piped:NODE;% selects pipelined or re-issue if not r4k mode%
Time_int:NODE;% Enables or disables the timer interrupt%
Sys_if:NODE;% selects the size of the system interface%
Drvr:NODE;% selects the drive strength of the outputs%

%============================================================================%

BEGIN

%----------------------------------------------------------------------------%
% Default state for all outputs is not asserted %
DEFAULTS
DRAM_PRESENT/=VCC;
MODEIN=GND;
VCCOK=VCC;
RESET/=VCC;
COLD_RESET/=VCC;
SYSTEM_RESET/=VCC;
SONIC_RESET/=VCC;
SCSI_RESET=GND;
IO_RESET/=VCC;
```

```
Mode_done/=VCC;
master_count[6..0]=GND;
mode_count[7..0]=GND;
END DEFAULTS;

%---------------- Output DFF clock and clear assignments  ------------------------%

DRAM_PRESENT/.clk=GLOBAL(MASTEROUT);
DRAM_PRESENT/.prn=GLOBAL(RESET_IN/);

MODEIN.clk=MODECLOCK;
MODEIN.clrn=GLOBAL(RESET_IN/);

VCCOK.clk=GLOBAL(MASTEROUT);
VCCOK.clrn=GLOBAL(RESET_IN/);

RESET/.clk=GLOBAL(MASTEROUT);
RESET/.clrn=GLOBAL(RESET_IN/);

COLD_RESET/.clk=GLOBAL(MASTEROUT);
COLD_RESET/.clrn=GLOBAL(RESET_IN/);

SYSTEM_RESET/.clk=GLOBAL(MASTEROUT);
SYSTEM_RESET/.clrn=GLOBAL(RESET_IN/);

SONIC_RESET/.clk=GLOBAL(MASTEROUT);
SONIC_RESET/.clrn=GLOBAL(RESET_IN/);

SCSI_RESET.clk=GLOBAL(MASTEROUT);
SCSI_RESET.prn=GLOBAL(RESET_IN/);

IO_RESET/.clk=GLOBAL(MASTEROUT);
IO_RESET/.clrn=GLOBAL(RESET_IN/);

%--------- State machine clock and clear ---------------------------------------%

master_sm.clk=GLOBAL(MASTEROUT);
master_sm.reset=GLOBAL(!RESET_IN/);

%---------- Buried nodes clock and clear ---------------------------------------%

master_count[6..0].clk=GLOBAL(MASTEROUT);
master_count[6..0].clrn=GLOBAL(RESET_IN/);

mode_count[7..0].clk=MODECLOCK;
mode_count[7..0].clrn=GLOBAL(RESET_IN/);

%----------- Equations ----------------------------------------------------------%

mode_count[7..0] = mode_count[7..0] + B"00000001";

Mode_done/ = !(mode_count[7..0] == B"11111111");

MODEIN = Byte_order # divisor5 # divisor6 # divisor7 # Write_pat # R4k_wrt #
Piped # Time_int # Sys_if # Drvr;

DRAM_PRESENT/ = (DRAM_SIZE[1..0] == B"11");
Byte_order = BIG_ENDIAN & (mode_count[7..0] == B"00000111");
divisor5 = CLK_DIV[0] & (mode_count[7..0] == B"00000100");
divisor6 = CLK_DIV[1] & (mode_count[7..0] == B"00000101");
divisor7 = CLK_DIV[2] & (mode_count[7..0] == B"00000110");
Write_pat = !DRAM_PRESENT/ & (mode_count[7..0] == B"00000001");
R4k_wrt = R4K_WRITE & (mode_count[7..0] == B"00001000");
Piped = PIPELINED & (mode_count[7..0] == B"00001001");
Time_int = TIMER_INT & (mode_count[7..0] == B"00001010");
Sys_if = !P3_SYS_IF & (mode_count[7..0] == B"00001011");
Drvr =(DRIVE & (mode_count[7..0] == B"00001100")) #
(mode_count[7..0] == B"00001101");
```

```
%-------------------------------------------------------------------------------%

% state machine design %

CASE master_sm IS

WHEN mst0 =>
% master control IDLE%
RESET/ = GND;
SYSTEM_RESET/ = GND;
SONIC_RESET/ = GND;
SCSI_RESET = VCC;
IO_RESET/ = GND;

IF (Mode_done/) THEN
% Wait for mode bits to all load%
COLD_RESET/ = GND;
master_sm = mst0;
ELSE
% mode bits done de-assert ColdReset*%
COLD_RESET/ = VCC;
master_count[6..0] = B"0000000";
master_sm = mst1;
END IF;

WHEN mst1 =>
% now wait for 64 master clocks and de-assert Reset*%
master_count[6..0] = master_count[6..0] + B"0000001";
SYSTEM_RESET/ = GND;
SONIC_RESET/ = GND;
SCSI_RESET = VCC;
IO_RESET/ = GND;

IF (master_count[6..0] == B"0111111") THEN
RESET/ = VCC;
master_sm = mst2;
ELSE
RESET/ = GND;
master_sm = mst1;
END IF;

WHEN mst2 =>
% now de-assert the other resets after another 64 master clocks%
master_count[6..0] = master_count[6..0] + B"0000001";

IF (master_count[6..0] == B"1111110") THEN
SYSTEM_RESET/ = VCC;
SONIC_RESET/ = VCC;
SCSI_RESET = GND;
IO_RESET/ = VCC;
master_sm = mst3;
ELSE
SYSTEM_RESET/ = GND;
SONIC_RESET/ = GND;
SCSI_RESET = VCC;
IO_RESET/ = GND;
master_sm = mst2;
END IF;

WHEN mst3 =>
% reset now done just stay here until another one starts things again%

master_sm = mst3;

END CASE;

END;
%
```

PLA Equations

This chapter contains the FPGA listings for the Bus Controller, DRAM Controller, Flash State Machine, Main Controller, RCLK Controller, Reset Controller, SONIC Controller, and SRAM Controller.

Bus Exchanger Controller PLD

MAX+PLUS II AHDLv5.0

```
EDIT HISTORY
Date            Engineer     Check Sum         Changes
-------------------------------------------------------------
10/12/1994    R.Cummings                      Initial Release
%

%=============================================================================%

%

%========================== CONSTANTS ==================================%
CONSTANT BYTE=B"000";
CONSTANT HALF_WORD=B"001";
CONSTANT THREE_BYTE=B"010";
CONSTANT WORD= B"011";
CONSTANT FIVE_BYTE= B"100";
CONSTANT SIX_BYTE= B"101";
CONSTANT SEVEN_BYTE= B"110";
CONSTANT DOUBLEWORD= B"111";
%=============================================================================%
SUBDESIGN Bus_Ex
(
% Input signals from CPU%
SYSAD28:  INPUT;% SYSAD28 = 0 for Memory access%
% SYSAD28 = 1 for IO accesses%
LSYSAD[3..0]:INPUT;% SYSAD[3..0] used decode byte enable%
VALIDOUT/:INPUT;% Indicates CPU has valid SYSAD and%
% SYSCMD values driven%
RELEASE/:INPUT;% Indicates the CPU has released %
% the bus to the external agent%
SYSCMD8:INPUT;% Indicates the type of CPU request%
SYSCMD6:INPUT;% Indicates the type of CPU request%
SYSCMD3:INPUT;% Indicates the type of CPU request%
LSYSCMD[2..0]:INPUT;% Indicates size of CPU requests%
B3_TCLK0_0:INPUT;% Buffered TClock from CPU.%

% Used as GLOBAL clock%
IO_Access/:INPUT;% Indicates an IO access is in progress %
% Other input signals%
WAIT/:INPUT;% Form main controller to let BusEx%
% know an IO request is in progress%
DRAM_Access/:INPUT;% From DRAM controller to indicate%
% that a DRAM access is in progress%
SRAM_Access/:INPUT;% From SRAM controller to indicate%
% that a SRAM access is in progress%
MEM_DONE/:INPUT;% Indicates that the DRAM access is%
% done (at least for this datum)%
DRAM_PRESENT/:INPUT;% Indicates if DRAM installed%
%DRAM_PRESENT/Description%
%0DRAM installed%
%1No DRAM %
P3_SYS_IF:INPUT;% Indicates size of P3 sys I/F%
%P3_SYS_IFDescription%
```

```
%032-bit interface%
%164-bit interface%
IGNORE_Access/:INPUT;% Asserted by DRAM Controller during%
% refresh cycles to indicate that the%
% CPU will not issue any Writes until%
% refresh is done and WrRdy* asserted%
DMA_RD/:INPUT;% Indicates a valid DMA Read request%
% is on the bus%
DMA_WR/:INPUT;% Indicates a valid DMA Write request%
% is on the bus%
WRRDY/:INPUT;% Need WrRdy* to determine if a write%
% can be issued in the current cycle%
RDRDY/:INPUT;% Need RdRdy* to determine if a read%
% can be issued in the current cycle%
RESETIN/:INPUT;% Global Reset signal%
FLASH_DONE_in:INPUT;% The flash done signal to be synced for the%
% main controller%

WR/:INPUT;% Input from main control %
RD/:INPUT;% Input from main control %
RA_SYSAD[28..20]: INPUT;% Input address, only for clearing the intr output %

% BusEx control output signals %
BYTE/[7..0]:OUTPUT;% Byte enables for memory system%
ED_OE/:OUTPUT;% OE for the even data to memory from Bus EX%
OD_OE/:OUTPUT;% OE for the odd data to memory from Bus EX%
MD2RD_OE/:OUTPUT;% Enable for data from memory to the RD side%
% of the Bus Ex on a CPU od DMA read%
E_O_SELECT:OUTPUT;% Selects even or odd half of memory data %
% for the CPU of DMA read%
RD2ED_LE:OUTPUT;% Latch data form CPU or DMA write to the %
% even side of the memory%
RD2OD_LE:OUTPUT;% Latch data form CPU or DMA write to the %
% odd side of the memory%
FLASH_DONE/:OUTPUT;% Flash done indicator to main controller%
INTR/:OUTPUT;% Interrupts CPU for faulty write cycles %
SYS_ERROR[1..0]:OUTPUT;% Indicates an error code for faulty reads%
ERROR/:OUTPUT;% indicates that an error has happened %
IO_CLK1:OUTPUT;% One of the IO Clocks. Sys IF freq / 2%
IO_CLK2:OUTPUT;% Another of the IO Clocks.%
IO_CLK3:OUTPUT;% Another of the IO Clocks.%
IO_CLK4:OUTPUT;% Another of the IO Clocks.%
)

%=============================================================================%
% Bus Exchanger control state machine %

VARIABLE

bexsm:MACHINE WITH STATES
(b0, b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, b12, b13, b14, b15, b16, b17, b18, b19, b20, b21, b22, b23, b24,
b25, b26, b27, b28, b29, b30, b31);

% state machine to sync flash done from falsh control with TCLK%
fdsm:MACHINE WITH STATES
(f0, f1);
% Error code  machine %
errsm: MACHINE WITH STATES(
er0, er1, er2, er3, er4, er5, er6, er7, er8
);
int_clr:  MACHINE WITH STATES( int0, int1);

%---------------  Output Flip Flops  ----------------------%
BYTE/[7..0]:DFFE;
ED_OE/:DFF;
OD_OE/:DFF;
MD2RD_OE/:DFF;
E_O_SELECT:DFF;
RD2ED_LE:DFF;
```

```
RD2OD_LE:DFF;
FLASH_DONE/:DFF;
INTR/:DFF;
SYS_ERROR[1..0]:DFF;
ERROR/:DFF;
IO_CLK1:DFF;
IO_CLK2:DFF;
IO_CLK3:DFF;
IO_CLK4:DFF;

%-------------------------------------------------------------------------------%
% Buried nodes %
Byte_en:NODE;% Enable for byte enables%
%DMA_Hold:DFF; Hold for one cycle for end of DMA%
%Removed 112895 to see if DRAM will work again%
Count[3..0]:DFF;% Counter for block accesses%
Wrrdy1/:DFF;% A one cycle delayed version of WrRdy* signal%
Wrrdy2/:DFF;% A two cycle delayed version of WrRdy* signal%
Rdrdy1/:DFF;% A one cycle delayed version of RdRdy* signal%
Rdrdy2/:DFF;% A two cycle delayed version of RdRdy* signal%
INTR_prn:DFF;% This signal resets the Interrupt output %
MEM_DONE_delay/:DFF;     % Delayed version of MEM_DONE. Only used in DRAM block reads %

%===============================================================================%

BEGIN

%-------------------------------------------------------------------------------%
% Default state for all outputs is not asserted %
DEFAULTS
BYTE/[7..0]=VCC;
Byte_en=GND;
ED_OE/=VCC;
OD_OE/=VCC;
MD2RD_OE/=VCC;
E_O_SELECT=VCC;
RD2ED_LE=GND;
RD2OD_LE=GND;
Count[3..0]=GND;
FLASH_DONE/=VCC;
SYS_ERROR[1..0]=B"00";
INTR/=VCC;
INTR_prn=VCC;
ERROR/=VCC;
IO_CLK1=VCC;
IO_CLK2=VCC;
IO_CLK3=VCC;
IO_CLK4=VCC;
MEM_DONE_delay/=VCC;
%DMA_Hold=GND;Removed 112895 to see if DRAM will work again%
END DEFAULTS;

%----------------- Output DFF clock and clear assignments  -------------------------%
SYS_ERROR[1..0].clk= GLOBAL(B3_TCLK0_0);
SYS_ERROR[1..0].clrn= GLOBAL(RESETIN/);

ERROR/.clk= GLOBAL(B3_TCLK0_0);
ERROR/.prn= GLOBAL(RESETIN/);

INTR/.clk= GLOBAL(B3_TCLK0_0);
INTR/.prn= GLOBAL(RESETIN/) & INTR_prn;

INTR_prn.clk= GLOBAL(B3_TCLK0_0);
INTR_prn.prn= GLOBAL(RESETIN/);

BYTE/[7..0].clk=GLOBAL(B3_TCLK0_0);
BYTE/[7..0].clrn=GLOBAL(RESETIN/);
BYTE/[7..0].ena=Byte_en;
```

```
ED_OE/.clk=GLOBAL(B3_TCLK0_0);
ED_OE/.clrn=GLOBAL(RESETIN/);

OD_OE/.clk=GLOBAL(B3_TCLK0_0);
OD_OE/.clrn=GLOBAL(RESETIN/);

MD2RD_OE/.clk=GLOBAL(B3_TCLK0_0);
MD2RD_OE/.prn=GLOBAL(RESETIN/);

E_O_SELECT.clk=GLOBAL(B3_TCLK0_0);
E_O_SELECT.prn=GLOBAL(RESETIN/);

RD2ED_LE.clk=GLOBAL(B3_TCLK0_0);
RD2ED_LE.prn=GLOBAL(RESETIN/);

RD2OD_LE.clk=GLOBAL(B3_TCLK0_0);
RD2OD_LE.prn=GLOBAL(RESETIN/);

FLASH_DONE/.clk=GLOBAL(B3_TCLK0_0);
FLASH_DONE/.prn=GLOBAL(RESETIN/);

IO_CLK1.clk=GLOBAL(B3_TCLK0_0);
IO_CLK1.prn=GLOBAL(RESETIN/);

IO_CLK2.clk=GLOBAL(B3_TCLK0_0);
IO_CLK2.prn=GLOBAL(RESETIN/);

IO_CLK3.clk=GLOBAL(B3_TCLK0_0);
IO_CLK3.prn=GLOBAL(RESETIN/);

IO_CLK4.clk=GLOBAL(B3_TCLK0_0);
IO_CLK4.prn=GLOBAL(RESETIN/);

%--------- State machine clock and clear ------------------------------------%

bexsm.clk=GLOBAL(B3_TCLK0_0);
bexsm.reset=GLOBAL(!RESETIN/);

fdsm.clk=GLOBAL(B3_TCLK0_0);
fdsm.reset=GLOBAL(!RESETIN/);

errsm.clk= GLOBAL(B3_TCLK0_0);
errsm.reset= GLOBAL(!RESETIN/);

int_clr.clk= GLOBAL(B3_TCLK0_0);
int_clr.reset= GLOBAL(!RESETIN/);

%---------- Buried nodes clock and clear ------------------------------------%

Count[3..0].clk=GLOBAL(B3_TCLK0_0);
Count[3..0].clrn=GLOBAL(RESETIN/);

Wrrdy1/.clk=GLOBAL(B3_TCLK0_0);
Wrrdy1/.clrn=GLOBAL(RESETIN/);

Wrrdy2/.clk=GLOBAL(B3_TCLK0_0);
Wrrdy2/.clrn=GLOBAL(RESETIN/);

Rdrdy1/.clk=GLOBAL(B3_TCLK0_0);
Rdrdy1/.clrn=GLOBAL(RESETIN/);

Rdrdy2/.clk=GLOBAL(B3_TCLK0_0);
Rdrdy2/.clrn=GLOBAL(RESETIN/);

MEM_DONE_delay/.clk = GLOBAL(B3_TCLK0_0);
MEM_DONE_delay/.prn = GLOBAL(RESETIN/);
```

```
%DMA_Hold.clk=GLOBAL(B3_TCLK0_0);
DMA_Hold.clrn=GLOBAL(RESETIN/);%

%---------- Equations ----------------------------------------------------%

Byte_en = ( bexsm == b0 );

!BYTE/7 = P3_SYS_IF &
((((!SYSCMD8 & SYSCMD6 & SYSCMD3) # !DMA_WR/) & (LSYSAD[2..0] == B"000")) #
(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/);

!BYTE/6 = P3_SYS_IF & ((((!SYSCMD8 & SYSCMD6 & SYSCMD3) # !DMA_WR/) &
(((LSYSAD[2..0] == B"001")& (LSYSCMD[2..0] == B"XXX")) #
((LSYSAD[2..0] == B"000") & (LSYSCMD[2..0] != BYTE)))) #
(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/);

!BYTE/5 = P3_SYS_IF & ((((!SYSCMD8 & SYSCMD6 & SYSCMD3) # !DMA_WR/)  &
((((LSYSAD[2..0] == B"010") & (LSYSCMD[2..0] == B"XXX"))#
((LSYSAD[2..0] == B"000") &( LSYSCMD[2..0] != BYTE) & (LSYSCMD[2..0] != HALF_WORD)) #
((LSYSAD[2..0] == B"001") & ((LSYSCMD[2..0] == THREE_BYTE) # (LSYSCMD[2..0] ==
SEVEN_BYTE)))))) #
(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/);

!BYTE/4 = P3_SYS_IF & ((((!SYSCMD8 & SYSCMD6 & SYSCMD3) # !DMA_WR/) &
((((LSYSAD[2..0] == B"011") & (LSYSCMD[2..0] == B"XXX"))#
((LSYSAD[2..0] == B"010") & (LSYSCMD[2..0] != BYTE)) #
((LSYSAD[2..0] == B"001") & ((LSYSCMD[2..0] == THREE_BYTE) # (LSYSCMD[2..0] ==
SEVEN_BYTE)) #
((LSYSAD[2..0] == B"000") & (LSYSCMD[2..0]>= WORD) & (LSYSCMD[2..0] <= DOUBLEWORD)))))) #
(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/);

!BYTE/3 =P3_SYS_IF & ((((!SYSCMD8 & SYSCMD6 & SYSCMD3) # !DMA_WR/) &
(((LSYSAD[2..0] == B"100") & (LSYSCMD[2..0] == B"XXX"))#
((LSYSCMD[2..0]>= FIVE_BYTE) & (LSYSCMD[2..0] <= DOUBLEWORD)))) #
(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/) ;
%(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/) #

!P3_SYS_IF & ((((!SYSCMD8 & SYSCMD6 & SYSCMD3) # !DMA_WR/)
 & (LSYSAD[1..0] == B"00")) #
(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/);%

!BYTE/2 =P3_SYS_IF &((((!SYSCMD8 & SYSCMD6 & SYSCMD3) # !DMA_WR/) &
(((LSYSAD[2..0] == B"101")& (LSYSCMD[2..0] == B"XXX"))#
((LSYSAD[2..0] == B"100") & (LSYSCMD[2..0]>= HALF_WORD) & (LSYSCMD[2..0] <= WORD)) #
((LSYSAD[2..0] == B"011") & (LSYSCMD[2..0] == FIVE_BYTE))#
((LSYSCMD[2..0]>= B"101") & (LSYSCMD[2..0] <= DOUBLEWORD)))) #
(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/);
%(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/) #

!P3_SYS_IF & ((((!SYSCMD8 & SYSCMD6 & SYSCMD3) # !DMA_WR/) &
(((LSYSAD[1..0] == B"01")& (LSYSCMD[2..0] == B"XXX")) #
((LSYSAD[1..0] == B"00") & (LSYSCMD[2..0] != BYTE)))) #
(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/);%

!BYTE/1 =P3_SYS_IF & ((((!SYSCMD8 & SYSCMD6 & SYSCMD3) # !DMA_WR/) &
(((LSYSAD[2..0] == B"110") & (LSYSCMD[2..0] == B"XXX")) #
((LSYSAD[2..0] == B"100") & ((LSYSCMD[2..0] == THREE_BYTE) # (LSYSCMD[2..0] == WORD))) #
((LSYSAD[2..0] == B"101") & (LSYSCMD[2..0] == THREE_BYTE)) #
((LSYSAD[2..0] == B"011") & (LSYSCMD[2..0] == FIVE_BYTE)) #
((LSYSAD[2..0] == B"010") & (LSYSCMD[2..0] == SIX_BYTE)) #
(LSYSCMD[2..0] == SEVEN_BYTE) #
(LSYSCMD[2..0] == DOUBLEWORD))) #
(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/);
%(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/) #

!P3_SYS_IF & ((((!SYSCMD8 & SYSCMD6 & SYSCMD3) # !DMA_WR/) &
((((LSYSAD[1..0] == B"10") & (LSYSCMD[2..0] == B"XXX"))#
((LSYSAD[1..0] == B"00") &( LSYSCMD[2..0] != BYTE) & (LSYSCMD[2..0] != HALF_WORD)) #
```

```
((LSYSAD[1..0] == B"01") & ((LSYSCMD[2..0] == THREE_BYTE)))))) #
(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/);%

!BYTE/0 =P3_SYS_IF & ((((!SYSCMD8 & SYSCMD6 & SYSCMD3) # !DMA_WR/) &
(((LSYSAD[2..0] == B"111") & (LSYSCMD[2..0] == B"XXX")) #
((LSYSAD[2..0] == B"110") & (LSYSCMD[2..0] == HALF_WORD)) #
((LSYSAD[2..0] == B"101") & (LSYSCMD[2..0] == THREE_BYTE)) #
((LSYSAD[2..0] == B"100") & (LSYSCMD[2..0] == WORD)) #
((LSYSAD[2..0] == B"011") & (LSYSCMD[2..0] == FIVE_BYTE)) #
((LSYSAD[2..0] == B"010") & (LSYSCMD[2..0] == SIX_BYTE)) #
((LSYSAD[2..0] == B"001") & (LSYSCMD[2..0] == SEVEN_BYTE)) #
 (LSYSCMD[2..0] == DOUBLEWORD))) #
 (!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/);
% (!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/) #

!P3_SYS_IF & ((((!SYSCMD8 & SYSCMD6 & SYSCMD3) # !DMA_WR/) &
((((LSYSAD[1..0] == B"11") & (LSYSCMD[2..0] == B"XXX"))#
((LSYSAD[1..0] == B"10") & (LSYSCMD[2..0] != BYTE)) #
((LSYSAD[1..0] == B"01") & ((LSYSCMD[2..0] == THREE_BYTE)) #
((LSYSAD[1..0] == B"00") & (LSYSCMD[2..0]== WORD)))))) #
(!SYSCMD8 & !SYSCMD6) # !SYSCMD3 # !DMA_RD/);%

Wrrdy1/.d = WRRDY/;
Wrrdy2/.d = Wrrdy1/;

Rdrdy1/.d = RDRDY/;
Rdrdy2/.d = Rdrdy1/;

IO_CLK1 = !IO_CLK1;
IO_CLK2 = !IO_CLK2;
IO_CLK3 = !IO_CLK3;
IO_CLK4 = !IO_CLK4;

MEM_DONE_delay/.d = MEM_DONE/;

%-------------------------------------------------------------------------------%

% state machine design %

CASE bexsm IS

WHEN b0 =>% BusEx IDLE%
E_O_SELECT = (LSYSAD3 & P3_SYS_IF) # (LSYSAD2 & !P3_SYS_IF);
MD2RD_OE/ = !((!SYSAD28 & !RELEASE/ & !VALIDOUT/ &
 (!SYSCMD6 # !DMA_RD/) & DRAM_Access/ & WAIT/)
 # (!SYSAD28 & !DMA_RD/ & WAIT/ & !SYSCMD6));

IF (!DRAM_Access/ # !WAIT/) THEN
ED_OE/ = ED_OE/ $ GND;
OD_OE/ = OD_OE/ $ GND;
ELSIF (DRAM_Access/ & WAIT/ & SYSCMD6 & !VALIDOUT/) THEN
%ELSIF ((DRAM_Access/ & WAIT/ & SYSCMD6 & !VALIDOUT/) # DMA_Hold) THEN%
%Removed DMA_Hold to see if DRAM will work again%
ED_OE/ = GND;
OD_OE/ = GND;
ELSE
ED_OE/ = VCC;
OD_OE/ = VCC;
END IF;

IF (!SYSAD28 & DRAM_Access/ & WAIT/ & !Rdrdy2/ &
(((!SYSCMD8 & !SYSCMD6 & SYSCMD3) & !RELEASE/ & !VALIDOUT/) #
!DMA_RD/)) THEN
% Single Read with no DRAM Access still in progress or DMA read%
bexsm = b1;
ELSIF (!SYSAD28 & (!DRAM_Access/ # !WAIT/) & !Rdrdy2/ &
(!SYSCMD8 & !SYSCMD6 & SYSCMD3) & !RELEASE/ & !VALIDOUT/) THEN
% Single Read with DRAM Access still in progress%
```

```
bexsm = b2;
ELSIF (!SYSAD28 &  !RELEASE/ & !VALIDOUT/ & DRAM_Access/ & !Rdrdy2/ &
WAIT/ & (!SYSCMD8 & !SYSCMD6 & !SYSCMD3)) THEN
% Block Read with no DRAM Access still in progress%
bexsm = b3;
ELSIF (!SYSAD28 & !RELEASE/ & !VALIDOUT/ & !Rdrdy2/ &
(!DRAM_Access/ # !WAIT/) & (!SYSCMD8 & !SYSCMD6 & !SYSCMD3)) THEN
% Block Read with DRAM Access still in progress%
bexsm = b4;
ELSIF (!SYSAD28 & ((!VALIDOUT/ & DRAM_Access/ & WAIT/ &
IGNORE_Access/ & (!SYSCMD8 & SYSCMD6 & SYSCMD3) & !Wrrdy2/) #
!DMA_WR/)) THEN
% Single writewith no DRAM Access still in progress or DMA write%
bexsm = b5;
ELSIF (!SYSAD28 & !VALIDOUT/ & (!DRAM_Access/ # !WAIT/) &
(!SYSCMD8 & SYSCMD6 & SYSCMD3) & IGNORE_Access/ & !Wrrdy2/) THEN
% Single writewith DRAM Access still in progress%
bexsm = b6;
ELSIF (!SYSAD28 & !VALIDOUT/ & IGNORE_Access/ &
(!SYSCMD8 & SYSCMD6 & !SYSCMD3) & !Wrrdy2/) THEN
% Block write%
RD2ED_LE = VCC;
RD2OD_LE = VCC;
Count[3..0] = B"0000";
bexsm = b7;
ELSE
bexsm = b0;
END IF;

WHEN b1 =>
% single read bus turn around cycle no DRAM cycle in progress or  after DRAM access done%
MD2RD_OE/ = GND;
E_O_SELECT = E_O_SELECT;

IF (!DMA_RD/) THEN
bexsm = b15;
ELSIF (SRAM_Access/) THEN
bexsm = b19;
ELSE
bexsm = b16;
END IF;

WHEN b2 =>
% single read bus turn around cycle DRAM cycle in progress.%
RD2ED_LE = RD2ED_LE;
RD2OD_LE = RD2OD_LE;
E_O_SELECT = E_O_SELECT;
ED_OE/ = ED_OE/ $ GND;
OD_OE/ = OD_OE/ $ GND;

IF (!DRAM_Access/ # !WAIT/) THEN
MD2RD_OE/ = MD2RD_OE/;
bexsm = b2;
ELSE
MD2RD_OE/ = GND;
bexsm = b1;
END IF;

WHEN b3 =>
% block read bus turn around cycle no DRAM cycle in progress%
MD2RD_OE/ = GND;
Count[3..0] = B"0000";
E_O_SELECT = E_O_SELECT;

IF (P3_SYS_IF & !SRAM_Access/) THEN
bexsm = b14;
ELSE
bexsm = b17;
```

```
END IF;
%ELSIF (P3_SYS_IF & SRAM_Access/) THEN
bexsm = b17;
ELSIF (!P3_SYS_IF & !SRAM_Access/) THEN
bexsm = b18;
ELSE
bexsm = b11;
END IF;%

WHEN b4 =>
% block read bus turn around cycle DRAM cycle in progress.%
RD2ED_LE = RD2ED_LE;
RD2OD_LE = RD2OD_LE;
E_O_SELECT = E_O_SELECT;

IF (!DRAM_Access/ # !WAIT/) THEN
MD2RD_OE/ = MD2RD_OE/;
bexsm = b4;
ELSE
MD2RD_OE/ = GND;
bexsm = b3;
END IF;

WHEN b5 =>
% Single write data cycle when no DRAM cycle still in progress%
ED_OE/ = GND;
OD_OE/ = GND;
RD2ED_LE = VCC;
RD2OD_LE = VCC;

IF ((!DMA_WR/) # (SRAM_Access/)) THEN
% DMA or DRAM single write%
bexsm = b20;
ELSE
bexsm = b0;
END IF;

WHEN b6 =>
% Data cycle with DRAM cycle in progress. Wait here till done then move data to the bus exchangers and memory
as needed%
IF (!DRAM_Access/) THEN
ED_OE/ = ED_OE/;
OD_OE/ = OD_OE/;
bexsm = b6;
ELSE
ED_OE/ = GND;
OD_OE/ = GND;
bexsm = b5;
END IF;

WHEN b7 =>
% data cycles for block write%
ED_OE/ = GND;
OD_OE/ = GND;

IF (Count[3..0] == B"0000" & P3_SYS_IF) THEN
RD2OD_LE = VCC;
Count[3..0] = B"0001";% 1st data cycle 64-bit Sys I/F%
bexsm = b7;
%ELSIF (Count[3..0] == B"0000" & !P3_SYS_IF) THEN
RD2OD_LE = VCC;
Count[3..0] = B"0001"; 1st data cycle 32-bit Sys I/F
bexsm = b12;%
ELSIF (Count[3..0] == B"0001" & DRAM_PRESENT/) THEN
Count[3..0] = B"0010";% 2nd data cycle no DRAM%
RD2ED_LE = VCC;
 bexsm = b7;
ELSIF (Count[3..0] == B"0001" & !DRAM_PRESENT/) THEN
```

```
Count[3..0] = B"0010";% 2nd data cycle with DRAM%
 bexsm = b7;
 ELSIF (Count[3..0] == B"0010" & DRAM_PRESENT/) THEN
Count[3..0] = B"1111";% 3rd data cycle no DRAM%
RD2OD_LE = VCC;
 bexsm = b7;
 ELSIF (Count[3..0] == B"0010" & !DRAM_PRESENT/) THEN
Count[3..0] = B"0011";% 1st X cycle with DRAM%
 bexsm = b7;
 ELSIF (Count[3..0] == B"0011" & !DRAM_PRESENT/) THEN
RD2ED_LE = VCC;
Count[3..0] = B"0100";% 2nd X cycle with DRAM%
 bexsm = b7;
 ELSIF (Count[3..0] == B"0100") THEN
Count[3..0] = B"0101";% 3rd data cycle with DRAM%
RD2OD_LE = VCC;
 bexsm = b7;
 ELSIF (Count[3..0] == B"0101") THEN
Count[3..0] = B"0110";% last data cycle with DRAM%
 bexsm = b7;
 ELSIF (Count[3..0] == B"0110") THEN
Count[3..0] = B"0111";% 1st X cycle with DRAM%
 bexsm = b7;
 ELSE
bexsm = b0;% 2nd X cycle with DRAM%
 END IF;

%WHEN b11 =>%
% DRAM block read data cycles - 32-bit Sys I/F%
%MD2RD_OE/ = GND;

IF ((Count[3..0] == B"0000") & !MEM_DONE/)THEN%
% 1st data cycle%
%E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0001";
bexsm = b11;
ELSIF ((Count[3..0] == B"0001") & !MEM_DONE/)THEN%
% 2nd data cycle%
%E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0010";
bexsm = b11;
ELSIF ((Count[3..0] == B"0010") & !MEM_DONE/)THEN%
% 3rd data cycle%
%E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0011";
bexsm = b11;
ELSIF ((Count[3..0] == B"0011") & !MEM_DONE/)THEN%
% 4th data cycle%
%E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0100";
bexsm = b11;
ELSIF ((Count[3..0] == B"0100") & !MEM_DONE/)THEN%
% 5th data cycle%
%E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0101";
bexsm = b11;
ELSIF ((Count[3..0] == B"0101") & !MEM_DONE/)THEN%
% 6th data cycle%
%E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0110";
bexsm = b11;
ELSIF ((Count[3..0] == B"0110") & !MEM_DONE/)THEN%
% 7th data cycle%
%E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0111";
bexsm = b16;
ELSE  wait for MEM_DONE/%
% No valid data%
```

```
%Count[3..0] = Count[3..0];
bexsm = b11;
END IF;%

%WHEN b12 =>
 data cycles for block write
ED_OE/ = GND;
OD_OE/ = GND;

IF (Count[3..0] == B"0001" & DRAM_PRESENT/) THEN
Count[3..0] = B"0010"; 2nd data cycle no DRAM
RD2ED_LE = VCC;
 bexsm = b12;
ELSIF (Count[3..0] == B"0001" & !DRAM_PRESENT/) THEN
Count[3..0] = B"0010"; 2nd data cycle with DRAM
 bexsm = b12;
 ELSIF (Count[3..0] == B"0010" & DRAM_PRESENT/) THEN
Count[3..0] = B"0011"; 3rd data cycle no DRAM
RD2OD_LE = VCC;
 bexsm = b12;
 ELSIF (Count[3..0] == B"0010" & !DRAM_PRESENT/) THEN
Count[3..0] = B"0011"; 1st X cycle with DRAM - 1st set
 bexsm = b12;
 ELSIF (Count[3..0] == B"0011" & DRAM_PRESENT/) THEN
RD2ED_LE = VCC; 4th data cycle no DRAM
 Count[3..0] = B"0100";
 bexsm = b12;
 ELSIF (Count[3..0] == B"0011" & !DRAM_PRESENT/) THEN
RD2ED_LE = VCC;
Count[3..0] = B"0100"; 2nd X cycle with DRAM - 1st set
 bexsm = b12;
 ELSIF (Count[3..0] == B"0100" & DRAM_PRESENT/) THEN
 RD2OD_LE = VCC; 5th data cycle no DRAM
 Count[3..0] = B"0101";
 bexsm = b12;
 ELSIF (Count[3..0] == B"0100" & !DRAM_PRESENT/) THEN
Count[3..0] = B"0101"; 3rd data cycle with DRAM
RD2OD_LE = VCC;
 bexsm = b12;
 ELSIF (Count[3..0] == B"0101" & DRAM_PRESENT/) THEN
 RD2ED_LE = VCC; 6th data cycle no DRAM
 Count[3..0] = B"0110";
 bexsm = b12;
 ELSIF (Count[3..0] == B"0101" & !DRAM_PRESENT/) THEN
Count[3..0] = B"0110"; 4th data cycle with DRAM
 bexsm = b12;
 ELSIF (Count[3..0] == B"0110" & DRAM_PRESENT/) THEN
 RD2OD_LE = VCC; 7th data cycle no DRAM
 Count[3..0] = B"0111";
 bexsm = b12;
 ELSIF (Count[3..0] == B"0110" & !DRAM_PRESENT/) THEN
Count[3..0] = B"0111"; 1st X cycle with DRAM - 2nd set
 bexsm = b12;
 ELSIF (Count[3..0] == B"0111" & !DRAM_PRESENT/) THEN
RD2ED_LE = VCC;
Count[3..0] = B"1000"; 2nd X cycle with DRAM - 2nd set
 bexsm = b12;
 Count[3..0] = B"1000";
 ELSIF (Count[3..0] == B"1000") THEN
RD2OD_LE = VCC; 5th data cycle with DRAM
 bexsm = b12;
 Count[3..0] = B"1001";
 ELSIF (Count[3..0] == B"1001") THEN
bexsm = b12; 6th data cycle with DRAM
 Count[3..0] = B"1010";
 ELSIF (Count[3..0] == B"1010") THEN
bexsm = b12; 1st X cycle with DRAM - 3rd set
 Count[3..0] = B"1011";
```

```
 ELSIF (Count[3..0] == B”1011”) THEN
RD2ED_LE = VCC;
Count[3..0] = B”1100”; 2nd X cycle with DRAM - 3rd set
 bexsm = b12;
 ELSIF (Count[3..0] == B”1100”) THEN
RD2OD_LE = VCC; 7th data cycle with DRAM
 bexsm = b12;
 Count[3..0] = B”1101”;
 ELSIF (Count[3..0] == B”1101”) THEN
bexsm = b12; 8th data cycle with DRAM
 Count[3..0] = B”1110”;
 ELSIF (Count[3..0] == B”1110”) THEN
bexsm = b12; 1st X cycle with DRAM - 4th set
 Count[3..0] = B”1111”;
 ELSE
bexsm = b0; 2nd X cycle with DRAM - 4th set
 END IF;%

WHEN b14 =>
% SRAM block read data cycles - 64-bit Sys I/F%
MD2RD_OE/ = GND;

IF (Count[3..0] == B”0000”) THEN
% 1st data cycle%
E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B”0001”;
bexsm = b14;
ELSIF (Count[3..0] == B”0001”) THEN
% 2nd data cycle%
E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B”0010”;
bexsm = b14;
ELSE% Count[3..0] == B”0010”%
% 3rd data cycle%
E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B”0011”;
bexsm = b16;
END IF;

WHEN b15 =>% Wait for DMA read cycle to end%
E_O_SELECT = E_O_SELECT;

IF (!DMA_RD/) THEN
MD2RD_OE/ = GND;
bexsm = b15;
ELSE
bexsm = b16;
END IF;

WHEN b16 =>
% single read data cycle or last data for block read%
E_O_SELECT = (E_O_SELECT $ !(Count[3..0] == B”0000”));
MD2RD_OE/ = GND;
bexsm = b0;

WHEN b17 =>
% DRAM block read data cycles - 64-bit Sys I/F%
MD2RD_OE/ = GND;

IF ((Count[3..0] == B”0000”) & !MEM_DONE_delay/)THEN
% 1st data cycle%
E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B”0001”;
bexsm = b17;
ELSIF ((Count[3..0] == B”0001”) & !MEM_DONE_delay/)THEN
% 2nd data cycle%
E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B”0010”;
```

```
bexsm = b17;
ELSIF ((Count[3..0] == B"0010") & !MEM_DONE_delay/)THEN
% 3rd data cycle%
E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0011";
bexsm = b16;
ELSE% Wait for MEM_DONE_delay/%
% No valid data%
Count[3..0] = Count[3..0];
E_O_SELECT = E_O_SELECT;
bexsm = b17;
END IF;

%WHEN b18 =>
 SRAM block read data cycles - 32-bit Sys I/F
MD2RD_OE/ = GND;

IF (Count[3..0] == B"0000") THEN
 1st data cycle
E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0001";
bexsm = b18;
ELSIF (Count[3..0] == B"0001") THEN
 2nd data cycle
E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0010";
bexsm = b18;
ELSIF (Count[3..0] == B"0010") THEN
 3rd data cycle
E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0011";
bexsm = b18;
ELSIF (Count[3..0] == B"0011") THEN
 4th data cycle
E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0100";
bexsm = b18;
ELSIF (Count[3..0] == B"0100") THEN
 5th data cycle
E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0101";
bexsm = b18;
ELSIF (Count[3..0] == B"0101") THEN
 6th data cycle
E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0110";
bexsm = b18;
ELSE Count[3..0] == B"0110"
 7th data cycle
E_O_SELECT = E_O_SELECT $ VCC;
Count[3..0] = B"0111";
bexsm = b16;
END IF;%


WHEN b19 =>
% Wait for DRAM read cycle to end%
MD2RD_OE/ = GND;
E_O_SELECT = E_O_SELECT;

IF ((MEM_DONE/) & (SYS_ERROR[1..0] == B"00")) THEN
bexsm = b19;
ELSE
bexsm = b16;
END IF;

% Wait for DRAM single write to finish %
WHEN b20 =>
```

```
ED_OE/ = GND;
OD_OE/ = GND;
IF(MEM_DONE/ & DMA_WR/) THEN
bexsm = b20;
ELSIF (!DMA_WR/) THEN
bexsm = b21;
ELSE
bexsm = b0;
END IF;

% Wait for DMA Write to finish%
WHEN b21 =>
%ED_OE/ = GND;
OD_OE/ = GND;%
% put OEs back into If to see if can re-fix DRAM - 112895%

IF(!DMA_WR/) THEN
ED_OE/ = GND;
OD_OE/ = GND;
bexsm = b21;
ELSE
%DMA_Hold = VCC;%
% Removed to see if will re-fix DRAM - 112895 %
ED_OE/ = VCC;
OD_OE/ = VCC;
bexsm = b0;
END IF;

WHEN OTHERS =>
bexsm = b0;
END CASE;

CASE fdsm IS

WHEN f0 =>
IF (!FLASH_DONE_in) THEN
FLASH_DONE/ = GND;
fdsm = f1;
ELSE
fdsm = f0;
END IF;

WHEN f1 =>
fdsm = f0;

END CASE;

%---------------------------------- Error code state machine ------------------- %
CASE errsm IS

WHEN er0 =>
IF((!WR/ # !RD/) & SRAM_Access/ & DRAM_Access/ & IO_Access/) THEN

ERROR/ = GND;
errsm = er1;
ELSE
errsm = er0;
END IF;

WHEN er1 =>
IF((!WR/ # !RD/) & SRAM_Access/ & DRAM_Access/ & IO_Access/) THEN

ERROR/ = GND;
errsm = er2;
ELSE
errsm = er0;
END IF;
WHEN er2 =>
```

```
IF((!WR/ # !RD/) & SRAM_Access/ & DRAM_Access/ & IO_Access/) THEN

ERROR/ = GND;
errsm = er5;
ELSE
errsm = er0;
END IF;


WHEN er3 =>
IF(!WR/ & SRAM_Access/ & DRAM_Access/ & IO_Access/) THEN
SYS_ERROR[1..0] = B"01";
INTR/= GND;

ERROR/ = GND;
errsm = er4;

ELSIF(!RD/ & SRAM_Access/ & DRAM_Access/ & IO_Access/) THEN
SYS_ERROR[1..0] = B"10";
ERROR/ = GND;
errsm = er3;

ELSE

ERROR/ = GND;
errsm = er0;


END IF;



WHEN er4 =>
IF(!INTR/) THEN
SYS_ERROR[1..0] = B"01";
INTR/ = INTR/;


errsm = er4;
ELSE
errsm = er0;
END IF;

% added state 5 and 6 REC - 041895 - 1432 %
WHEN er5 =>
IF((!WR/ # !RD/) & SRAM_Access/ & DRAM_Access/ & IO_Access/) THEN

ERROR/ = GND;
errsm = er6;
ELSE
errsm = er0;
END IF;
WHEN er6 =>
IF((!WR/ # !RD/) & SRAM_Access/ & DRAM_Access/ & IO_Access/) THEN

ERROR/ = GND;
errsm = er3;
ELSE
errsm = er0;
END IF;

END CASE;

%_____________________________________________________________________________ %

CASE int_clr IS

WHEN int0 =>
```

```
IF (P3_SYS_IF & !WR/ & (RA_SYSAD[28..20] ==H"1f0") & WAIT/) THEN

INTR_prn= GND;
int_clr = int1;

ELSE
int_clr = int0;
END IF;

WHEN int1 =>
int_clr = int0;
END CASE;

END;



%
```

DRAM Controller PLD

MAX+PLUS II AHDL v5.00

DRAM Controller.

EDIT HISTORY

Date	Engineer	Checksum	Change	Comments
	S.Khan R.Napaa			Initial release

%

%Besides providing DRAM control signals, it should control the data paths e.g. bus exchangers and should also generate error codes SYS_CODE[1..0] for main_ctr and generate INTR/ signal for faulty write cycles %

TITLE " dram_ctr";
DESIGN IS "dram_ctr"
DEVICE "dram_ctr" IS "EPM7096LC84-10";

%==%

% DRAM control info table

32-bit interface selection:

P3_SYS_IFDescription
032-bit interface
164-bit interface

SRAM size selection:

SRAM_SIZE[1..0]Description
001 MEG
014 MEG
1Xno SRAM

DRAM Memory configuration:

DRAM_SIZE[1..0]SRAM_PRESENT, SRAM_SIZE0

0000MEM0SRAM 1 Meg, DRAM 4 Meg
0001MEM1SRAM 4 Meg, DRAM 4 Meg
001XMEM2NO SRAM , DRAM 4 Meg
0100MEM3SRAM 1 Meg, DRAM 16 Meg
0101MEM4SRAM 4 Meg, DRAM 16 Meg
011XMEM5NO SRAM, DRAM 16 Meg
1000MEM6SRAM 1 Meg, DRAM 64 Meg
1001MEM7SRAM 4 Meg, DRAM 64 Meg
101XMEM8NO SRAM, DRAM 64 Meg
1100MEM9SRAM 1 Meg, No DRAM
1101MEM10SRAM 4 Meg. No DRAM
111XMEM11No SRAM, No DRAM

Error Description:

SYS_ERROR[1..0]Description

00Normal operation
01Write error
10Single read error
11Block read error

```
%
%=============================================================================%

SUBDESIGN dram_ctr
(

RA_SYSAD[28..16]:INPUT;%latched demux address coming to
dram_ctr one cycle later  %
RA_SYSADn[4..2]:INPUT;

R_Syscmd3:INPUT;%latched Syscmd[3..0] bits coming
to dram_ctr one cycle later   %
TCLK:INPUT;
MRes/:INPUT;
3_6MHz:INPUT;
IGNORE_ACCESS/:INPUT;


% Signals from main_ctr %

RD/:INPUT;
WR/:INPUT;

DMA_RD/:INPUT;
DMA_WR/:INPUT;

% DRAM acknowledge signals %


P3_SYS_IF:INPUT;


SRAM_SIZE[1..0]:INPUT;

SRAM_PRESENT/:INPUT;

DRAM_SIZE[1..0]:INPUT;

SRAM_Access/:INPUT;
WAIT/:INPUT;

RefAck/:INPUT;


% Error code to main_ctr %

%INTR/:OUTPUT;
SYS_ERROR[1..0]:OUTPUT;%
RefReq/:OUTPUT;

MOD_A4:OUTPUT;
MOD_A3:OUTPUT;
MOD_A_SEL:OUTPUT; % Controls the sel. input of fct257 %

DRAM_Access_node:OUTPUT; % DRAM_Access/ %


% Wait from main_ctr to stop second buffer cycle from completing
before the first cycle is finished   %

MEM_DONE/:OUTPUT;

BE/[7..0]:INPUT;

% DRAM Control signals %

RAS_ODA/:OUTPUT;
RAS_ODB/:OUTPUT;
```

```
RAS_EVA/:OUTPUT;
RAS_EVB/:OUTPUT;

AD_MUX_LE:OUTPUT;
WE_OD/:OUTPUT;
WE_EV/:OUTPUT;
MA_SEL:OUTPUT;

CAS[7..0]:OUTPUT;
RefAck2/:OUTPUT;
RefReq2/:OUTPUT;

%DRAM_On/:OUTPUT;
DRAM_32_On/:OUTPUT;%
)

%============================================================================%

VARIABLE

%Outputs %

%INTR/:DFF;
INTR_prn:DFF;

SYS_ERROR[1..0]:DFF;%

MEM_DONE/:DFF;

R_Syscmd_st3:DFFE;

RA_SYSAD_st[4..3]:DFFE;

%BE/[7..0]:DFF;%

RAS_ODA/:DFFE;

RAS_ODB/:DFFE;

RAS_EVA/:DFFE;

RAS_EVB/:DFFE;

AD_MUX_LE:DFF;
WE_OD/:DFF;
WE_EV/:DFF;
MA_SEL:DFF;

P3_BL_COUNT:DFFE;
P3_BL_COUNT_ena:DFF;

CAS[7..0]:DFF;

% Refresh logic %

RCnt[5..0]:DFF; % Refresh counter %
RefReq/:DFF; % Refresh request %
RefReq1/:DFF;
RefReq2/:DFF;

RefAck1/:DFF;
RefAck2/:DFF;

RCnt_clrn/:DFF;
RefDone:DFF;
RefDone_pr/:DFF;
RefDone_clr/:DFF;
```

```
Stag_Ref:DFF;

% Internal burried nodes %

%DRAM_32_On/:NODE; %
R_Syscmd_st0_ena:NODE;
RA_SYSAD_st_en:NODE;
RAS_ODA/_en:DFF;
RAS_ODB/_en:DFF;
RAS_EVA/_en:DFF;
RAS_EVB/_en:DFF;
Stag_Ref_clr:NODE;
Stag_Ref_clk:DFF;
RefAck1/_prn:NODE;

MOD_A4:DFF;
MOD_A3:DFF;
MOD_A_SEL:DFF;

DRAM_Access_node:DFF;
DRAM_Off/:DFF;
DRAM_On/:NODE;
DRAM:DFF;
ERROR/:DFF;
MEM[11..0]:DFF;
%SRAM_PRE/:DFF;%



%--------------------------------------------------------------------------%

dramsm:MACHINE WITH STATES ( st0, st1, st2, st3, st4, st5, st6,st7, st8, st9, st10,st11, st12, st13, st14, st15,
st16,st17,st18, st19, st20, st21, st22, st23, st24, st25, st26, st27, st28)  ;

rfsm: MACHINE WITH STATES( r0, r1, r2,r3   ) ;

%errsm: MACHINE WITH STATES( er0, er1, er2, er3, er4  ) ;%
%==========================================================================%

BEGIN

DEFAULTS
MEM_DONE/= VCC;
R_Syscmd_st0_ena= GND;
RA_SYSAD_st_en= GND;
WE_EV/= VCC;
WE_OD/= VCC;
MA_SEL= GND;
CAS[7..0]= B"11111111";
RA_SYSAD_st[4..3]  = B"00";
R_Syscmd_st3  = B"1";
RAS_ODA/ = VCC;
RAS_ODB/ = VCC;
RAS_EVA/ = VCC;
RAS_EVB/ = VCC;
P3_BL_COUNT= GND;
P3_BL_COUNT_ena= GND;
RAS_ODA/_en= GND;
RAS_ODB/_en= GND;
RAS_EVA/_en= GND;
RAS_EVB/_en= GND;
%DRAM_32_On/= VCC; %
RCnt[5..0]= B"000000";
RefReq/= VCC;
RefReq1/= VCC;
RefReq2/= VCC;
RefAck1/= VCC;
RefAck2/= VCC;
```

```
RCnt_clrn/= VCC;
RefDone= GND;
%Stag_Ref= GND;%
Stag_Ref_clr= VCC;
RefAck1/_prn= VCC;
Stag_Ref_clk= GND;
MOD_A_SEL = VCC;
DRAM_Access_node= VCC;
RefDone_pr/= VCC;
RefDone_clr/= VCC;
DRAM= GND;
ERROR/= VCC;
DRAM_On/= VCC;
DRAM_Off/= VCC;
AD_MUX_LE= VCC;
MEM[11..0]= H"000";
END DEFAULTS;

%-------------------------------------------------------------------------------%

% Outputs %

P3_BL_COUNT.clk= GLOBAL(TCLK);
P3_BL_COUNT.clrn= GLOBAL(MRes/);
P3_BL_COUNT.ena= P3_BL_COUNT_ena;

P3_BL_COUNT_ena.clk= GLOBAL(TCLK);
P3_BL_COUNT_ena.clrn= GLOBAL(MRes/);



MEM_DONE/.clk= GLOBAL(TCLK);
MEM_DONE/.prn= GLOBAL(MRes/);

R_Syscmd_st3.clk= GLOBAL(TCLK);
R_Syscmd_st3.prn= GLOBAL(MRes/);
R_Syscmd_st3.ena= R_Syscmd_st0_ena;


RAS_ODA/.clk= GLOBAL(TCLK);
RAS_ODA/.prn= GLOBAL(MRes/);
RAS_ODA/.ena= RAS_ODA/_en;

RAS_ODA/_en.clk= GLOBAL(TCLK);
RAS_ODA/_en.clrn= GLOBAL(MRes/);

RAS_ODB/.clk= GLOBAL(TCLK);
RAS_ODB/.prn= GLOBAL(MRes/);
RAS_ODB/.ena= RAS_ODB/_en;

RAS_ODB/_en.clk= GLOBAL(TCLK);
RAS_ODB/_en.clrn= GLOBAL(MRes/);


RAS_EVA/.clk= GLOBAL(TCLK);
RAS_EVA/.prn= GLOBAL(MRes/);
RAS_EVA/.ena= RAS_EVA/_en;

RAS_EVA/_en.clk= GLOBAL(TCLK);
RAS_EVA/_en.clrn= GLOBAL(MRes/);


RAS_EVB/.clk= GLOBAL(TCLK);
RAS_EVB/.prn= GLOBAL(MRes/);
RAS_EVB/.ena= RAS_EVB/_en;

RAS_EVB/_en.clk= GLOBAL(TCLK);
RAS_EVB/_en.clrn= GLOBAL(MRes/);
```

```
AD_MUX_LE.clk= GLOBAL(TCLK);
AD_MUX_LE.prn= GLOBAL(MRes/);

WE_OD/.clk= GLOBAL(TCLK);
WE_OD/.prn= GLOBAL(MRes/);

WE_EV/.clk= GLOBAL(TCLK);
WE_EV/.prn= GLOBAL(MRes/);

MA_SEL.clk= GLOBAL(TCLK);
MA_SEL.clrn= GLOBAL(MRes/);

CAS[7..0].clk= GLOBAL(TCLK);
CAS[7..0].prn= GLOBAL(MRes/);

RA_SYSAD_st[4..3].clk= GLOBAL(TCLK);
RA_SYSAD_st[4..3].clrn= GLOBAL(MRes/);
RA_SYSAD_st[4..3].ena= RA_SYSAD_st_en;

%RA_SYSAD_st_en.clk= GLOBAL(TCLK);
RA_SYSAD_st_en.clrn= GLOBAL(MRes/);%

%BE/[7..0].clk= GLOBAL(TCLK);
BE/[7..0].prn= GLOBAL(MRes/);%

RCnt[5..0].clk= 3_6Mhz;
RCnt[5..0].clrn= GLOBAL(MRes/) & RCnt_clrn/;

RefReq/.clk= GLOBAL(TCLK);
RefReq/.prn= GLOBAL(MRes/);

RefReq1/.clk= GLOBAL(TCLK);
RefReq1/.prn= GLOBAL(MRes/);

RefReq2/.clk= GLOBAL(TCLK);
RefReq2/.prn= GLOBAL(MRes/);

RefAck1/.clk= GLOBAL(TCLK);
RefAck1/.prn= GLOBAL(MRes/) & RefAck1/_prn;
%RefAck1/.clrn= RefAck/;%

RefAck2/.clk= GLOBAL(TCLK);
RefAck2/.prn= GLOBAL(MRes/);

RefDone.clk= VCC;
RefDone.clrn= GLOBAL(MRes/) & RefDone_clr/;
RefDone.prn= RefDone_pr/;

RefDone_pr/.clk= GLOBAL(TCLK);
RefDone_pr/.prn= GLOBAL(MRes/);

RefDone_clr/.clk= GLOBAL(TCLK);
RefDone_clr/.prn= GLOBAL(MRes/);

RCnt_clrn/.clk= GLOBAL(TCLK);
RCnt_clrn/.prn= GLOBAL(MRes/);

Stag_Ref.clk= Stag_Ref_clk;
Stag_Ref.clrn= GLOBAL(MRes/) & Stag_Ref_clr;
Stag_Ref.d= !Stag_Ref.q;

Stag_Ref_clk.clk= GLOBAL(TCLK);
Stag_Ref_clk.clrn= GLOBAL(MRes/);

MOD_A4.clk= GLOBAL(TCLK);
MOD_A4.prn= GLOBAL(MRes/);
```

```
MOD_A3.clk= GLOBAL(TCLK);
MOD_A3.prn= GLOBAL(MRes/);

MOD_A_SEL.clk= GLOBAL(TCLK);
MOD_A_SEL.prn= GLOBAL(MRes/);

%DRAM_Access/.clk= GLOBAL(TCLK);
DRAM_Access/.prn= GLOBAL(MRes/) ;%

DRAM_Access_node.clk= GLOBAL(TCLK);
DRAM_Access_node.prn= GLOBAL(MRes/);

%DRAM_On/.clk= GLOBAL(TCLK);
DRAM_On/.prn= GLOBAL(MRes/);%

DRAM_Off/.clk= GLOBAL(TCLK);
DRAM_Off/.prn= GLOBAL(MRes/);

%DRAM_32_On/.clk= GLOBAL(TCLK); DRAM_32_On/.prn= GLOBAL(MRes/);%

DRAM.clk= GLOBAL(TCLK);
DRAM.clrn= GLOBAL(MRes/) & DRAM_Off/;
DRAM.d= GND;
DRAM.prn= DRAM_On/;

ERROR/.clk= GLOBAL(TCLK);
ERROR/.prn= GLOBAL(MRes/);

%INTR_prn.clk= GLOBAL(TCLK); INTR_prn.prn= GLOBAL(MRes/);%

dramsm.clk= GLOBAL(TCLK);
dramsm.reset= GLOBAL(!MRes/);

rfsm.clk= GLOBAL(TCLK);
rfsm.reset= GLOBAL(!MRes/);

%errsm.clk= GLOBAL(TCLK); errsm.reset= GLOBAL(MRes/);%

MEM[11..0].clk= GLOBAL(TCLK);
MEM[11..0].clrn= GLOBAL(MRes/);

%SRAM_PRE/.clk=GLOBAL(TCLK);SRAM_PRE/.prn=GLOBAL(MRes/);SRAM_PRE/
d=SRAM_PRESENT/;%
%-------------------------------------------------------------------------%

% Equations %

RA_SYSAD_st[4..3].d= RA_SYSADn[4..3];
R_Syscmd_st3.d= R_Syscmd3;
R_Syscmd_st0_ena= ((dramsm == st0) # (dramsm == st1));
RA_SYSAD_st_en= ((dramsm == st0) # (dramsm == st1));

%DRAM_Access_prn= !((dramsm == st0) # (dramsm == st1));%

RCnt[5..0].d = RCnt[5..0].q + 1;

RefReq1/.d = RefReq2/.q ;
RefReq/.d = RefReq1/.q # RefDone;

RefAck1/.d = RefAck/;
RefAck2/.d = RefAck1/.q;

%MEM0 = !DRAM_SIZE1 & !DRAM_SIZE0 & !SRAM_PRESENT/ & !SRAM_SIZE0;
MEM1 = !DRAM_SIZE1 & !DRAM_SIZE0 & !SRAM_PRESENT/ &  SRAM_SIZE0;
MEM2 = !DRAM_SIZE1 & !DRAM_SIZE0 & SRAM_PRESENT/;
MEM3 = !DRAM_SIZE1 & DRAM_SIZE0 & !SRAM_PRESENT/ & !SRAM_SIZE0;
MEM4 = !DRAM_SIZE1 & DRAM_SIZE0 & !SRAM_PRESENT/ & SRAM_SIZE0;
MEM5 = !DRAM_SIZE1 & DRAM_SIZE0 & SRAM_PRESENT/ ;
```

```
MEM6 = DRAM_SIZE1 & !DRAM_SIZE0 & !SRAM_PRESENT/ & !SRAM_SIZE0;
MEM7 = DRAM_SIZE1 & !DRAM_SIZE0 & !SRAM_PRESENT/ & SRAM_SIZE0;
MEM8 = DRAM_SIZE1 & !DRAM_SIZE0 & SRAM_PRESENT/ ;
MEM9 = DRAM_SIZE1 & DRAM_SIZE0 & !SRAM_PRESENT/ & !SRAM_SIZE0;
MEM10 = DRAM_SIZE1 & DRAM_SIZE0 & !SRAM_PRESENT/ & SRAM_SIZE0;
MEM11 = DRAM_SIZE1 & DRAM_SIZE0 & SRAM_PRESENT/;%

!DRAM_On/= ((SRAM_SIZE0 == B"0") & SRAM_Access/ &
 (DRAM_SIZE0==B"0") & (RA_SYSAD[28..20] < H"005")) #% SRAM = 1M, DRAM = 4M %
((SRAM_SIZE0 == B"0") & SRAM_Access/ &
 (DRAM_SIZE0==B"1") & (RA_SYSAD[28..20] < H"011")) # % SRAM = 1M, DRAM = 16M %
((SRAM_SIZE0 == B"1") & SRAM_Access/ &
 (DRAM_SIZE0==B"0") & (RA_SYSAD[28..20] < H"008")) # % SRAM = 4M, DRAM = 4M %
((SRAM_SIZE0 == B"1") & SRAM_Access/ &
 (DRAM_SIZE0==B"1") & (RA_SYSAD[28..20] < H"014")) ; % SRAM = 4M, DRAM = 16M %

%!DRAM_32_On/ =((SRAM_SIZE0 == B"0") & SRAM_Access/ &
 (DRAM_SIZE0==B"0") & (RA_SYSAD[28..16] < H"0028")) #  SRAM = 0.5M, DRAM = 2M %
%((SRAM_SIZE0 == B"0") & SRAM_Access/ &
 (DRAM_SIZE0==B"1") & (RA_SYSAD[28..16] < H"0088")) #  SRAM = 0.5M, DRAM = 8M %
%((SRAM_SIZE0 == B"1") & SRAM_Access/ &
 (DRAM_SIZE0==B"0") & (RA_SYSAD[28..16] < H"0040")) #  SRAM = 2M, DRAM = 2M %
%((SRAM_SIZE0 == B"1") & SRAM_Access/ &
 (DRAM_SIZE0==B"1") & (RA_SYSAD[28..16] < H"00A0")) ;  SRAM = 2M, DRAM = 8M %


%_____Main state machine _____ %

CASE dramsm IS

WHEN st0 =>

RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;

IF(IGNORE_ACCESS/ & (!RD/ # !WR/ # !DMA_RD/ # !DMA_WR/) & RefAck/ & WAIT/) &

 (((SRAM_SIZE0 == B"0") & SRAM_Access/ &
 (DRAM_SIZE0==B"0") & (RA_SYSAD[28..20] < H"004")) #

((SRAM_SIZE0 == B"0") & SRAM_Access/ &
 (DRAM_SIZE0==B"1") & (DRAM_SIZE1==B"0") & (RA_SYSAD[28..20] < H"011")) #

((SRAM_SIZE0 == B"1") & SRAM_Access/ &
 (DRAM_SIZE0==B"0") & (RA_SYSAD[28..20] < H"008")) #


((SRAM_SIZE0 == B"1") & SRAM_Access/ &
 (DRAM_SIZE0==B"1") & (DRAM_SIZE1==B"0") & (RA_SYSAD[28..20] < H"014")))  THEN

RAS_ODA/ =  (!RA_SYSADn3)  & R_Syscmd3;
RAS_ODB/ =  (!RA_SYSADn3)  & R_Syscmd3;
RAS_EVA/ =  (RA_SYSADn3) & R_Syscmd3;
RAS_EVB/ =  (RA_SYSADn3) & R_Syscmd3;

AD_MUX_LE = GND;
MOD_A_SEL = VCC;
DRAM_Access_node = GND;
DRAM_Off/ = GND;
dramsm = st2;

ELSIF (!RefAck/) THEN

AD_MUX_LE = GND;
DRAM_Access_node = VCC;
dramsm = st21;
```

```
%ELSIF (P3_SYS_IF & (!WR/ # !DMA_WR/)  & (RA_SYSAD[28..20] ==H"1f0") & WAIT/) THEN

MEM_DONE/ = GND;
ERROR/= GND;
INTR_prn= GND;
dramsm = st0;

ELSIF (!P3_SYS_IF & !(!RD/ # !WR/# !DMA_RD/ # !DMA_WR/)) THEN

dramsm = st1;%


ELSE
dramsm = st0;

END IF;

WHEN st1 =>

%RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;

IF(IGNORE_ACCESS/ & !P3_SYS_IF & (!RD/ # !WR/# !DMA_RD/ # !DMA_WR/) & DRAM & RefAck/  &
ERROR/ & WAIT/)  THEN

RAS_ODA/ =  !RA_SYSADn2 & R_Syscmd3;
RAS_EVA/ =   RA_SYSADn2 & R_Syscmd3;
AD_MUX_LE = GND;
MOD_A4 = RA_SYSAD_st4;
MOD_A3 = RA_SYSAD_st3;
MOD_A_SEL = VCC;
DRAM_Access_node = GND;

dramsm = st2;



ELSIF (!P3_SYS_IF & !RefAck/) THEN

AD_MUX_LE = GND;
DRAM_Access_node = VCC;

dramsm = st21;

ELSIF ((!WR/ # !DMA_WR/) & (RA_SYSAD[28..20] ==H"1f0") & WAIT/) THEN

MEM_DONE/ = GND;
ERROR/ = GND;
dramsm = st1;


ELSIF(P3_SYS_IF & !(!RD/ # !WR/# !DMA_RD/ # !DMA_WR/)) THEN
dramsm = st0;

ELSE

dramsm = st1;

END IF;%
dramsm = st1;


WHEN st2 =>
```

```
RAS_ODA/ = RAS_ODA/;
RAS_ODB/ = RAS_ODB/;
RAS_EVA/ = RAS_EVA/;
RAS_EVB/ = RAS_EVB/;

IF(!WR/# !DMA_WR/) THEN
WE_OD/ = RAS_ODA/ & RAS_ODB/;
WE_EV/ = RAS_EVA/ & RAS_EVB/;
MA_SEL = VCC;
AD_MUX_LE = GND;
MOD_A4= RA_SYSAD_st4;
MOD_A3 = RA_SYSAD_st3;
MOD_A_SEL = GND;
DRAM_Access_node = GND;


dramsm = st4;

%ELSIF(DRAM_On/ & DRAM_32_On/) THEN DRAM_Access_node = VCC; dramsm = st0;%

ELSE
MA_SEL = VCC;
AD_MUX_LE = GND;
MOD_A4= RA_SYSAD_st4;
MOD_A3 = RA_SYSAD_st3;
MOD_A_SEL = GND;
DRAM_Access_node = GND;


dramsm = st12;

END IF;

WHEN st4 =>
DRAM_Access_node = GND;
%P3_BL_COUNT = VCC;%
WE_OD/ = WE_OD/;
WE_EV/ = WE_EV/;
MA_SEL = VCC;
AD_MUX_LE = GND;
MEM_DONE/ = !R_Syscmd_st3;

MOD_A4= (!P3_SYS_IF & !P3_BL_COUNT & RA_SYSAD_st4) #
 (!P3_SYS_IF & P3_BL_COUNT & !RA_SYSAD_st4) #
  (P3_SYS_IF & RA_SYSAD_st4);
MOD_A3 =  RA_SYSAD_st3 ;
MOD_A_SEL = GND;

CAS[7..0] =  BE/[7..0];

dramsm = st3;

WHEN st3 =>
DRAM_Access_node = GND;
IF !(R_Syscmd_st3) THEN
WE_OD/ = WE_OD/;
WE_EV/ = WE_EV/;
MA_SEL = VCC;
P3_BL_COUNT_ena = VCC;
AD_MUX_LE = GND;
MOD_A4= (!P3_SYS_IF & !P3_BL_COUNT & RA_SYSAD_st4) #
 (!P3_SYS_IF & P3_BL_COUNT & !RA_SYSAD_st4) #
  (P3_SYS_IF & RA_SYSAD_st4);
MOD_A3 = RA_SYSAD_st3;
MOD_A_SEL = GND;

CAS[7..0] =  BE/[7..0];
```

```
dramsm = st7;

ELSE

WE_OD/= WE_OD/;
WE_EV/ = WE_EV/;
MA_SEL= VCC;
AD_MUX_LE = GND;

RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;
MOD_A4= (!P3_SYS_IF & RA_SYSAD_st4) # (P3_SYS_IF & RA_SYSAD_st4);
MOD_A3 = RA_SYSAD_st3;
MOD_A_SEL = GND;

CAS[7..0] =  BE/[7..0];
dramsm = st5;

END IF;

WHEN st5 =>
DRAM_Access_node = GND;
R_Syscmd_st0_ena= VCC;
RA_SYSAD_st_en= VCC;

IF(P3_SYS_IF) THEN
dramsm = st0;
RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;
dramsm = st0;

ELSIF( !P3_SYS_IF) THEN
RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;
dramsm = st1;

END IF;


WHEN st7 =>
DRAM_Access_node = GND;
WE_OD/ = GND;
WE_EV/ = GND;
MA_SEL = VCC;
AD_MUX_LE = GND;
P3_BL_COUNT = VCC;
MOD_A4= (!P3_SYS_IF & !P3_BL_COUNT & RA_SYSAD_st4) #
 (!P3_SYS_IF & P3_BL_COUNT & !RA_SYSAD_st4) #
  (P3_SYS_IF & !RA_SYSAD_st4);
MOD_A3 = !RA_SYSAD_st3;
MOD_A_SEL = GND;

dramsm = st8;

WHEN st8 =>
DRAM_Access_node = GND;
WE_OD/ = GND;
WE_EV/ = GND;
MA_SEL = VCC;

AD_MUX_LE = GND;
MEM_DONE/ = !P3_BL_COUNT & (R_Syscmd_st3 # !P3_SYS_IF);
```

```
MOD_A4= (!P3_SYS_IF & !P3_BL_COUNT & RA_SYSAD_st4) #
 (!P3_SYS_IF & P3_BL_COUNT & !RA_SYSAD_st4) #
  (P3_SYS_IF & !RA_SYSAD_st4);
MOD_A3 = (!RA_SYSAD_st3 & !P3_BL_COUNT) # (!RA_SYSAD_st3 & P3_BL_COUNT);
MOD_A_SEL = GND;

CAS[7..0] =  BE/[7..0];
dramsm = st9;

WHEN st9 =>
DRAM_Access_node = GND;
WE_OD/ = GND;
WE_EV/ = GND;
MA_SEL = VCC;
AD_MUX_LE = GND;
CAS[7..0] =  BE/[7..0];

IF(P3_SYS_IF) THEN

RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;
MOD_A4= (!P3_SYS_IF & !P3_BL_COUNT & RA_SYSAD_st4) #
 (!P3_SYS_IF & P3_BL_COUNT & !RA_SYSAD_st4) #
  (P3_SYS_IF & !RA_SYSAD_st4);
MOD_A3 = (!RA_SYSAD_st3 & !P3_BL_COUNT) # (!RA_SYSAD_st3 & P3_BL_COUNT);
MOD_A_SEL = GND;


dramsm = st10;

ELSIF(P3_BL_COUNT & !P3_SYS_IF) THEN

RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;
MOD_A4= (!P3_SYS_IF & !P3_BL_COUNT & RA_SYSAD_st4) #
 (!P3_SYS_IF & P3_BL_COUNT & !RA_SYSAD_st4) #
  (P3_SYS_IF & !RA_SYSAD_st4);
MOD_A3 = (!RA_SYSAD_st3 & !P3_BL_COUNT) # (!RA_SYSAD_st3 & P3_BL_COUNT);
MOD_A_SEL = GND;
P3_BL_COUNT_ena = VCC;

dramsm = st10;


ELSE

RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;

MOD_A4= (!P3_SYS_IF & !P3_BL_COUNT & RA_SYSAD_st4) #
 (!P3_SYS_IF & P3_BL_COUNT & !RA_SYSAD_st4) #
  (P3_SYS_IF & !RA_SYSAD_st4);
MOD_A3 = (!RA_SYSAD_st3 & !P3_BL_COUNT) # (RA_SYSAD_st3 & P3_BL_COUNT);
MOD_A_SEL = GND;
P3_BL_COUNT_ena = VCC;


dramsm = st10;

END IF;
```

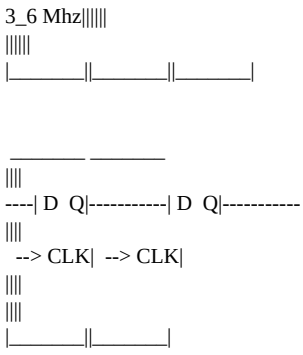
```
WHEN st10 =>

DRAM_Access_node = GND;
IF(!P3_SYS_IF & !P3_BL_COUNT) THEN

P3_BL_COUNT = VCC;
WE_OD/ = WE_OD/;
WE_EV/ = WE_EV/;
MA_SEL = VCC;
RAS_ODA/ =  !RA_SYSADn2 & R_Syscmd3;
RAS_EVA/ =   RA_SYSADn2 & R_Syscmd3;

MOD_A4 = !RA_SYSAD_st4;
MOD_A3 = RA_SYSAD_st3;
MOD_A_SEL = GND;


dramsm = st4;

ELSIF(!P3_SYS_IF & P3_BL_COUNT) THEN

RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;

P3_BL_COUNT = GND;
MOD_A4= !RA_SYSAD_st4;
MOD_A3 = !RA_SYSAD_st3;
MOD_A_SEL = VCC;

dramsm =   st1;

ELSE

P3_BL_COUNT = GND;
MOD_A4= (!P3_SYS_IF & !P3_BL_COUNT & RA_SYSAD_st4) #
 (!P3_SYS_IF & P3_BL_COUNT & !RA_SYSAD_st4) #
  (P3_SYS_IF & !RA_SYSAD_st4);

MOD_A3 = !RA_SYSAD_st3;
MOD_A_SEL = VCC;


dramsm = st0;

END IF;

WHEN st12 =>
DRAM_Access_node = GND;
P3_BL_COUNT = VCC;
MA_SEL = VCC;
AD_MUX_LE = GND;
!MEM_DONE/ = (P3_SYS_IF & R_Syscmd_st3) # (!P3_SYS_IF & P3_BL_COUNT & !R_Syscmd_st3)
# (!P3_SYS_IF & !P3_BL_COUNT & R_Syscmd_st3);

MOD_A4= MOD_A4;
MOD_A3 = RA_SYSAD_st3;
MOD_A_SEL = GND;

CAS[7..0] =  BE/[7..0];

dramsm = st13;

WHEN st13 =>
DRAM_Access_node = GND;
IF !(R_Syscmd_st3) THEN
MA_SEL = VCC;
```

```
AD_MUX_LE = GND;

CAS[7..0] =  BE/[7..0];
MOD_A4= MOD_A4 ;
MOD_A3 = RA_SYSAD_st3;
MOD_A_SEL = GND;
MEM_DONE/ = GND;


dramsm = st16;

ELSE

MA_SEL = VCC;
AD_MUX_LE = GND;
MOD_A3= RA_SYSAD_st3;
MOD_A_SEL = GND;

CAS[7..0] =  BE/[7..0];
dramsm = st14;

END IF;

WHEN st14 =>
DRAM_Access_node = GND;
R_Syscmd_st0_ena= VCC;
RA_SYSAD_st_en= VCC;

IF(P3_SYS_IF) THEN
dramsm = st0;
RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;

P3_BL_COUNT_ena = VCC;
P3_BL_COUNT = GND;

ELSIF( !P3_SYS_IF) THEN
dramsm = st1;
END IF;

WHEN st16 =>
DRAM_Access_node = GND;
MA_SEL = VCC;
AD_MUX_LE = GND;
MOD_A3= !RA_SYSAD_st3;
MOD_A4= (!P3_SYS_IF & !P3_BL_COUNT & RA_SYSAD_st4) #
 (!P3_SYS_IF & P3_BL_COUNT & !RA_SYSAD_st4) #
  (P3_SYS_IF & !RA_SYSAD_st4);

MOD_A_SEL = GND;
!MEM_DONE/ = (P3_SYS_IF & !R_Syscmd_st3) # (!P3_SYS_IF & P3_BL_COUNT & R_Syscmd_st3)
# (!P3_SYS_IF & !P3_BL_COUNT & !R_Syscmd_st3);


dramsm = st17;

WHEN st17 =>
DRAM_Access_node = GND;
MA_SEL = VCC;

AD_MUX_LE = GND;
MEM_DONE/ = GND;
MOD_A4= (!P3_SYS_IF & !P3_BL_COUNT & RA_SYSAD_st4) #
 (!P3_SYS_IF & P3_BL_COUNT & !RA_SYSAD_st4) #
  (P3_SYS_IF & !RA_SYSAD_st4);
```

```
MOD_A3= !RA_SYSAD_st3;
MOD_A_SEL = GND;

CAS[7..0] =  BE/[7..0];
dramsm = st18;

WHEN st18 =>
DRAM_Access_node = GND;
MA_SEL = VCC;
MOD_A_SEL = GND;
AD_MUX_LE = GND;
MEM_DONE/ = GND;
MOD_A3= !RA_SYSAD_st3;

MOD_A4= (!P3_SYS_IF & !P3_BL_COUNT & RA_SYSAD_st4) #
 (!P3_SYS_IF & P3_BL_COUNT & !RA_SYSAD_st4) #
  (P3_SYS_IF & !RA_SYSAD_st4);

CAS[7..0] =  BE/[7..0];
P3_BL_COUNT_ena = VCC;

dramsm = st19;

WHEN st19 =>
DRAM_Access_node = GND;
P3_BL_COUNT = VCC;

IF(!P3_SYS_IF & !P3_BL_COUNT) THEN

P3_BL_COUNT_ena = VCC;
P3_BL_COUNT = VCC;
WE_OD/ = WE_OD/;
WE_EV/ = WE_EV/;
MA_SEL = VCC;
MOD_A4= (!P3_SYS_IF & !P3_BL_COUNT & RA_SYSAD_st4) #
 (!P3_SYS_IF & P3_BL_COUNT & !RA_SYSAD_st4) #
  (P3_SYS_IF & !RA_SYSAD_st4);

MOD_A4 = !RA_SYSAD_st4;
MOD_A3= RA_SYSAD_st3;
MOD_A_SEL = VCC;

dramsm = st12;

ELSIF(!P3_SYS_IF & P3_BL_COUNT) THEN

RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;

P3_BL_COUNT_ena = VCC;
P3_BL_COUNT = GND;
MOD_A4= !RA_SYSAD_st4;
MOD_A3 = !RA_SYSAD_st3;
MOD_A_SEL = VCC;
CAS[7..0] =  BE/[7..0];

dramsm =   st20;


ELSE

MOD_A4= !RA_SYSAD_st4;
MOD_A3 = !RA_SYSAD_st3;
MOD_A_SEL = VCC;
CAS[7..0] =  BE/[7..0];
```

```
dramsm = st20;

END IF;

WHEN st20 =>
DRAM_Access_node = GND;
RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;

P3_BL_COUNT_ena = VCC;
P3_BL_COUNT = GND;
MOD_A4= !RA_SYSAD_st4;
MOD_A3 = RA_SYSAD_st3;
MOD_A_SEL = VCC;

IF (P3_SYS_IF) THEN

dramsm = st0;
ELSE
dramsm = st1;
END IF;


WHEN st21 =>

AD_MUX_LE = GND;
DRAM_Access_node = VCC;
dramsm = st22;

WHEN st22 =>
CAS[7..0] = GND;
RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;
DRAM_Access_node = VCC;
dramsm = st23;

WHEN st23 =>


RAS_ODA/ = Stag_Ref;
RAS_ODB/ = Stag_Ref;
RAS_EVA/ = !Stag_Ref;
RAS_EVB/ = !Stag_Ref;

CAS[7..0] = GND;
DRAM_Access_node = VCC;
dramsm = st24;

WHEN st24 =>

CAS[7..0] = GND;
DRAM_Access_node = VCC;
dramsm = st25;

WHEN st25 =>

CAS[7..0] = GND;
RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;
DRAM_Access_node = VCC;
```

```
dramsm = st26;

WHEN st26 =>


RAS_ODA/ = VCC;
RAS_ODB/ = VCC;
RAS_EVA/ = VCC;
RAS_EVB/ = VCC;

DRAM_Access_node = VCC;
CAS[7..0] = GND;
dramsm = st27;

WHEN st27 =>
IF(Stag_Ref) THEN
Stag_Ref_clk = VCC;
DRAM_Access_node = VCC;
dramsm = st28;

ELSE
CAS[7..0] = GND;
Stag_Ref_clk = VCC;
RAS_ODA/_en= VCC;
RAS_ODB/_en = VCC;
RAS_EVA/_en = VCC;
RAS_EVB/_en = VCC;
DRAM_Access_node = VCC;

dramsm = st23;

END IF;
WHEN st28 =>
IF(RefAck/) THEN
RefDone_pr/ = GND;
RefAck1/_prn = GND;
DRAM_Access_node = VCC;
dramsm = st0;
ELSE
RefDone_pr/ = GND;
DRAM_Access_node = VCC;
dramsm = st28;
END IF;

%WHEN st29 =>
IF(WR/ & DMA_WR/ & RD/ & DMA_RD/) THEN

dramsm = st0;

ELSE

dramsm = st29;

END IF;%

WHEN OTHERS =>
dramsm = st0;

END CASE;

%_____________________________________________________________________________ %


% ___O___ _______ _______
RefReq2/|| RefReq1/||RefReq/ ||
----| D  Q|----------| D  Q|----------| D  Q|---------
||||||
  --> CLK|  Vcc --> CLK|> CLK|
```



%

```
CASE rfsm IS

WHEN r0 =>
IF((RCnt[5..0]==H”24”) & MRes/) THEN

RefReq2/ = GND;
RCnt_clrn/ = GND;

rfsm = r1;
ELSE
rfsm = r0;

END IF;

WHEN r1 =>
IF(!RefAck2/) THEN

RefReq2/ = RefReq2/;
RCnt_clrn/ = GND;
rfsm = r2;

ELSE
RefReq2/ = GND;
RCnt_clrn/ = GND;
rfsm = r1;

END IF;

WHEN r2 =>
IF(RefAck2/) THEN
RefDone_clr/ = GND;
rfsm = r0;
%RCnt_clrn/ = GND;%
RefAck1/_prn = GND;
ELSE

RefReq2/ = GND;
RCnt_clrn/ = GND;
rfsm = r2;
END IF;

WHEN OTHERS =>
rfsm = r0;


END CASE;
```

% _____%

```
%CASE errsm IS
```

```
WHEN er0 =>
IF((!WR/ # !RD/ # !DMA_WR/ # !DMA_RD/) & SRAM_Access/ & DRAM_Access_node & IO_Access/)
THEN

ERROR/ = GND;
errsm = er1;
ELSE
errsm = er0;
END IF;

WHEN er1 =>
IF((!WR/ # !RD/ # !DMA_WR/ # !DMA_RD/) & SRAM_Access/ & DRAM_Access_node & IO_Access/)
THEN

ERROR/ = GND;
errsm = er2;
ELSE
errsm = er0;
END IF;
WHEN er2 =>
IF((!WR/ # !RD/ # !DMA_WR/ # !DMA_RD/) & SRAM_Access/ & DRAM_Access_node & IO_Access/)
THEN

ERROR/ = GND;
errsm = er3;
ELSE
errsm = er0;
END IF;


WHEN er3 =>
IF((!WR/ # !DMA_WR/)  & SRAM_Access/ & DRAM_Access_node & IO_Access/) THEN
SYS_ERROR[1..0] = B”01”;
INTR/= GND;

ERROR/ = GND;
errsm = er4;

ELSIF((!RD/ # !DMA_RD/) & SRAM_Access/ & DRAM_Access_node & IO_Access/) THEN
SYS_ERROR[1..0] = B”10”;
ERROR/ = GND;
errsm = er3;

ELSE

ERROR/ = GND;
errsm = er0;


END IF;



WHEN er4 =>
IF(!INTR/) THEN

SYS_ERROR[1..0] = B”01”;
INTR/ = INTR/;


errsm = er4;
ELSE
errsm = er0;
END IF;


END CASE;%
```

```
END;



%
```

IO state Machine

MAX+PLUS II AHDLv5.0

EDIT HISTORY

Date	Engineer	Check Sum	Changes
10/26/94	R.Cummings		Initial Release
06/15/95	R.Cummings		Modified for board problems for 8-bit

%

FLASH State Machine;

%==%

% I/O memory Map
Reserved0x1F00_0000 to 0x1F2F_FFFF
SCC (85C30)0x1F30_0000 to 0x1F3F_FFFF
SCSI (53C90A)0x1F40_0000 to 0x1F4F_FFFF
NVRAM0x1F50_0000 to 0x1F5F_FFFF
Ethernet0x1F60_0000 to 0x1F6F_FFFF(In SONIC Controller)
Expansion CS0x1F70_0000 to 0x1F7F_FFFF(In SONIC Controller)
LED CS0x1F80_0000 to 0x1F8F_FFFF
Reserved0x1F90_0000 to 0x1FAF_FFFF
Flash Write0x1FB0_0000 to 0x1FBF_FFFF (setup writes for real write to flash)
Flash Bank00x1FC0_0000 to 0x1FDF_FFFF(On Mother Board)
Flash Bank10x1FE0_0000 to 0x1FFF_FFFF(On Expansion Connector)
%
%==%

CONSTANTfwcs/=h"FB";
CONSTANTflash1_start=h"FC";
CONSTANT flash1_top=h"FD";
CONSTANTflash2_start=h"FE";
CONSTANT flash2_top=h"FF";
CONSTANT byte=h"8";
CONSTANT halfword=h"9";
CONSTANT word=h"B";
CONSTANT doubleword=h"F";
CONSTANT non_double=b"10XX";
CONSTANT block=B"0XXX";

%==%

SUBDESIGN FLASH
(
% system interface control input signals %
IO_CLK1 : INPUT; % One of the IO Clocks from the Main Controller%
IO_RESET/: INPUT;% system IO reset from RESET controller%
WR/:INPUT; % write signal from MAIN Controller%
RD/:INPUT; % read signal from MAIN Controller%
P3_SYS_IF:INPUT; % Indicates system I/F size. 0 = 32-bit; 1 = 64-bits%
FLASH8/:INPUT; % Indicates width of FLASH memory.%
%FLASH8/Width%
%08-bit%
%132-bit%

% Other control inputs%
DMA_WR/:INPUT;% Indicates that a DMA write is in progress. Used for%
% the control of the IO data bus exchangers%

% address input signals %
LIO_ADDR[4..0]:INPUT;% IO Address[4:0]. Used for FLASH address%
IO_ADDR[28..20]:INPUT;% bit 28 Determines IO (= 1) or memory (= 0) %
% access. [23:20] determine device accessed.%
IO_SYSCMD[3..0]:INPUT;% IO syscmd signals for FLASH access control%

% ioflash acknowledge signal%
Flash_Done/:OUTPUT;% Access done to sonic controller %

```
% FLASH Address control signals - These are only used in 8-bit FLASH system%
E8_ADDR0:OUTPUT;% Byte address%
E8_ADDR1:OUTPUT;% Halfword address %
E8_ADDR2:OUTPUT;% Word address%

% This FLASH Address is only for a 32-bit FLASH system%
EW_ADDR0:OUTPUT;% Word address 32bit FLASH system%

% These are the common FLASH address control signals. Used for block accesses%
EDW_ADDR3:OUTPUT;% Double word address%
EDW_ADDR4:OUTPUT;% Quad word address%

% Other FLASH Control signals%
E8_OE/:OUTPUT;% Output enable for the FLASH for on board FLASH%
E8_CE/:OUTPUT;% FLASH Chip enable for the on board FLASH%
Ex8_OE/:OUTPUT;% Output enable for expansion FLASH%
Ex8_CE/:OUTPUT;% Chip enable for expansion FLASH%
WE_8/:OUTPUT;% FLASH Write enable signal for 8-bit FLASH or%
% for EW_DATA[31:24]in 32-bit FLASH system%
WE_W3/:OUTPUT;% Write enable for FLASH at EW_DATA[7:0] - 32-bit %
WE_W2/:OUTPUT;% Write enable for FLASH at EW_DATA[15:8] - 32-bit %
WE_W1/:OUTPUT;% Write enable for FLASH at EW_DATA[23:16] - 32-bit%

% Control signals for the 8-bit FLASH System Xcievers (FCT162543CT)%
W8D_OE/[3..0]:OUTPUT;% Output enable for read data from FLASH%
% Normally HIGH but LOW to read data from FLASH%
W8D_LE/[3..0]:OUTPUT;% Latch enable for read data from FLASH.%
% May toggle or may always be LOW - not sure%
E8W_OE/[3..1]:OUTPUT;% Output enable for the write data path to FLASH %
% Normally HIGH but LOW for writes. This controls%
% the lower 3 bytes of the EW_DATA bus for a%
% FLASH write%
EH_OE/:OUTPUT;% Output enable for upper byte for FLASH write %
E_BYTE0_LE/:OUTPUT;% Latch enable for EW_DATA[31:24] on FLASH write%
E_BYTE1_LE/:OUTPUT;% Latch enable for EW_DATA[23:16] on FLASH read%
E_BYTE2_LE/:OUTPUT;% Latch enable for EW_DATA[15:8] on FLASH read%
E_BYTE3_LE/:OUTPUT;% Latch enable for EW_DATA[7:0] on FLASH read%

% IO Bus Exchanger control signals%
IOEX_LEA1B:OUTPUT;% EW_DATA to R_SYSAD lower half latch enable%
% Latches EX_DATA[31..0] to R_SYSAD[31..0] for %
% read data to CPU or DMA writes IO-2-MEMORY%
IOEX_LEA2B:OUTPUT;% EW_DATA to R_SYSAD upper half latch enable%
% Latches EX_DATA[31..0] to R_SYSAD[63..32] for %
% read data to CPU or DMA writes IO-2-MEMORY%
IOEX_OE1B/:OUTPUT;% Output enable for read (DMA write) data to%
% R_SYSAD[31..0]%
IOEX_OE2B/:OUTPUT;% Output enable for read (DMA write) data to%
% R_SYSAD[63..32]%

)

%============================================================================%

VARIABLE
Flash_Done/:    DFF;
E8_ADDR0:     DFF;
E8_ADDR1:     DFF;
E8_ADDR2:     DFF;
EW_ADDR0:   DFFE;
EDW_ADDR3:DFFE;
EDW_ADDR4:DFFE;
E8_OE/:DFF;
E8_CE/:DFF;
Ex8_OE/:DFF;
Ex8_CE/:DFF;
WE_8/:DFF;
```

```
WE_W3/:DFF;
WE_W2/:DFF;
WE_W1/:DFF;
W8D_OE/[3..0]:DFF;
W8D_LE/[3..0]:DFF;
E8W_OE/[3..1]:DFF;
EH_OE/:DFF;
E_BYTE0_LE/:DFF;
E_BYTE1_LE/:DFF;
E_BYTE2_LE/:DFF;
E_BYTE3_LE/:DFF;
IOEX_LEA1B:DFF;
IOEX_LEA2B:DFF;
IOEX_OE1B/:DFF;
IOEX_OE2B/:DFF;

%-------------------------------------------------------------------------%

%the io state machine is subdivided into sub-state machines.
each handles a spesific io or a number of different io %

flash:MACHINE WITH STATES (
flashidle,
flash1, flash2, flash3, flash4, flash5, flash6, flash7,
flash8, flash9, flash10, flash11, flash12, flash13,
flash14, flash15
);

%-------------------------------------------------------------------------%
% Buried nodes%

F_cyc_cnt[3..0]:DFF;% Cycle count for byte accesses from 8-bit%
% Flash memory system. Used for each byte of%
% of the access%
F_word_cnt[2..0]:DFFE;% Counts the number of words accessed for a%
% block read from FLASH memory%
F_wd_cnt_en:NODE;% Enable for word counter%
%E8_addr0_en:NODE; Enables for the FLASH address signals%
EW_addr0_en:NODE;
EDW_addr3_en:NODE;
EDW_addr4_en:NODE;

%=========================================================================%

BEGIN

%-------------------------------------------------------------------------%

% Default state for all outputs is not asserted %

DEFAULTS
Flash_Done/=VCC;
WE_8/=VCC;
WE_W3/=VCC;
WE_W2/=VCC;
WE_W1/=VCC;
W8D_OE/[3..0]=VCC;
W8D_LE/[3..0]=VCC;
E8W_OE/[3..1]=VCC;
E_BYTE0_LE/=VCC;
E_BYTE1_LE/=VCC;
E_BYTE2_LE/=VCC;
E_BYTE3_LE/=VCC;
IOEX_LEA1B=GND;
IOEX_LEA2B=GND;
IOEX_OE1B/=VCC;
IOEX_OE2B/=VCC;
F_cyc_cnt[3..0]=GND;
```

```
F_word_cnt[2..0]=GND;
F_wd_cnt_en=GND;
%E8_addr0_en=GND;%
EW_addr0_en=GND;
EDW_addr3_en=GND;
EDW_addr4_en=GND;
E8_ADDR0=VCC;
E8_ADDR1=VCC;
E8_ADDR2=VCC;
END DEFAULTS;

%-------------------------------------------------------------------------------%

Flash_Done/.clk = GLOBAL(IO_CLK1);%Define clock for IOAck/ output %
Flash_Done/.prn = GLOBAL(IO_RESET/);%Define preset condition for IOAck/ output %

E8_ADDR0.clk = GLOBAL(IO_CLK1);
E8_ADDR0.prn = GLOBAL(IO_RESET/);

E8_ADDR1.clk = GLOBAL(IO_CLK1);
E8_ADDR1.prn = GLOBAL(IO_RESET/);

E8_ADDR2.clk = GLOBAL(IO_CLK1);
E8_ADDR2.prn = GLOBAL(IO_RESET/);

EW_ADDR0.clk = GLOBAL(IO_CLK1);
EW_ADDR0.prn = GLOBAL(IO_RESET/);
EW_ADDR0.ena = EW_addr0_en;

EDW_ADDR3.clk = GLOBAL(IO_CLK1);
EDW_ADDR3.prn = GLOBAL(IO_RESET/);
EDW_ADDR3.ena = EDW_addr3_en;

EDW_ADDR4.clk = GLOBAL(IO_CLK1);
EDW_ADDR4.prn = GLOBAL(IO_RESET/);
EDW_ADDR4.ena = EDW_addr4_en;

E8_OE/.clk = GLOBAL(IO_CLK1);
E8_OE/.prn = GLOBAL(IO_RESET/);

E8_CE/.clk = GLOBAL(IO_CLK1);
E8_CE/.prn = GLOBAL(IO_RESET/);

Ex8_OE/.clk = GLOBAL(IO_CLK1);
Ex8_OE/.prn = GLOBAL(IO_RESET/);

Ex8_CE/.clk = GLOBAL(IO_CLK1);
Ex8_CE/.prn = GLOBAL(IO_RESET/);

WE_8/.clk = GLOBAL(IO_CLK1);
WE_8/.prn = GLOBAL(IO_RESET/);

WE_W3/.clk = GLOBAL(IO_CLK1);
WE_W3/.prn = GLOBAL(IO_RESET/);

WE_W2/.clk = GLOBAL(IO_CLK1);
WE_W2/.prn = GLOBAL(IO_RESET/);

WE_W1/.clk = GLOBAL(IO_CLK1);
WE_W1/.prn = GLOBAL(IO_RESET/);

W8D_OE/[3..0].clk = GLOBAL(IO_CLK1);
W8D_OE/[3..0].prn = GLOBAL(IO_RESET/);

W8D_LE/[3..0].clk = GLOBAL(IO_CLK1);
W8D_LE/[3..0].prn = GLOBAL(IO_RESET/);

E8W_OE/[3..1].clk = GLOBAL(IO_CLK1);
```

```
E8W_OE/[3..1].prn = GLOBAL(IO_RESET/);

EH_OE/.clk = GLOBAL(IO_CLK1);
EH_OE/.prn = GLOBAL(IO_RESET/);

E_BYTE0_LE/.clk = GLOBAL(IO_CLK1);
E_BYTE0_LE/.prn = GLOBAL(IO_RESET/);

E_BYTE1_LE/.clk = GLOBAL(IO_CLK1);
E_BYTE1_LE/.prn = GLOBAL(IO_RESET/);

E_BYTE2_LE/.clk = GLOBAL(IO_CLK1);
E_BYTE2_LE/.prn = GLOBAL(IO_RESET/);

E_BYTE3_LE/.clk = GLOBAL(IO_CLK1);
E_BYTE3_LE/.prn = GLOBAL(IO_RESET/);

IOEX_LEA1B.clk = GLOBAL(IO_CLK1);
IOEX_LEA1B.prn = GLOBAL(IO_RESET/);

IOEX_LEA2B.clk = GLOBAL(IO_CLK1);
IOEX_LEA2B.prn = GLOBAL(IO_RESET/);

IOEX_OE1B/.clk = GLOBAL(IO_CLK1);
IOEX_OE1B/.prn = GLOBAL(IO_RESET/);

IOEX_OE2B/.clk = GLOBAL(IO_CLK1);
IOEX_OE2B/.prn = GLOBAL(IO_RESET/);

 flash.clk = GLOBAL(IO_CLK1);
 flash.reset = GLOBAL(!IO_RESET/);

%-------------------------------------------------------------------------------%
% Buried nodes%

F_cyc_cnt[3..0].clk = GLOBAL(IO_CLK1);
F_cyc_cnt[3..0].clrn=GLOBAL(IO_RESET/);

F_word_cnt[2..0].clk = GLOBAL(IO_CLK1);
F_word_cnt[2..0].clrn=GLOBAL(IO_RESET/);
F_word_cnt[2..0].ena=F_wd_cnt_en;

%-------------------------------------------------------------------------------%

% equations %

!Flash_Done/ =((flash == flash7) & (F_cyc_cnt[3..0] == B"0100")) #
((flash == flash10) & (F_cyc_cnt[3..0] == B"0011")) #
((flash == flash12) & (F_cyc_cnt[3..0] == B"0011")) #
((flash == flash13) & (F_cyc_cnt[3..0] == B"0010")) #
((flash == flash14) & (F_cyc_cnt[3..0] == B"0010")) #
((flash == flash1) & (F_cyc_cnt[3..0] == B"0100")) #
((flash == flash2) & (F_cyc_cnt[3..0] == B"1000")) #
((flash == flash4) & (F_cyc_cnt[3..0] == B"0111"));

IOEX_LEA1B=(!RD/ & (IO_ADDR[28] == B"1") & (flash == flashidle)) #
!DMA_WR/;

IOEX_LEA2B=(!RD/ & (IO_ADDR[28] == B"1") & (flash == flashidle)) #
!DMA_WR/;

!IOEX_OE1B/=((!RD/ & (IO_ADDR[28] == B"1")) #
!DMA_WR/);

!IOEX_OE2B/=((!RD/ & (IO_ADDR[28] == B"1")) #
!DMA_WR/);

F_wd_cnt_en=((flash == flashidle) #
```

```
((F_cyc_cnt[3..0] == B"0100") & (IO_SYSCMD[3..0] == block) &
FLASH8/));
EW_addr0_en=(FLASH8/ & ((flash == flashidle) # (F_cyc_cnt[3..0] == B"0100")));
EDW_addr3_en=((flash == flashidle) #
((F_cyc_cnt[3..0] == B"0100") &
((flash == flash10) # (flash == flash12))));
EDW_addr4_en=((flash == flashidle) #
((flash == flash12) & (F_cyc_cnt[3..0] == B"0100")));
!E8_OE/=(!(flash == flashidle) & !RD/ &
((IO_ADDR[27..20] == flash1_start) # (IO_ADDR[27..20] == flash1_top)));
!E8_CE/=(!(flash == flashidle) &
((IO_ADDR[27..20] == flash1_start) # (IO_ADDR[27..20] == flash1_top)));
!Ex8_OE/=(!(flash == flashidle) & !RD/ &
((IO_ADDR[27..20] == flash2_start) # (IO_ADDR[27..20] == flash2_top)));
!Ex8_CE/=(!(flash == flashidle) &
((IO_ADDR[27..20] == flash2_start) # (IO_ADDR[27..20] == flash2_top)));
!E8W_OE/3=(!(flash == flashidle) & !FLASH8/ & !WR/ & (LIO_ADDR[1..0] == B"11"));
!E8W_OE/2=(!(flash == flashidle) & !FLASH8/ & !WR/ & (LIO_ADDR[1..0] == B"10"));
!E8W_OE/1=(!(flash == flashidle) & !FLASH8/ & !WR/ & (LIO_ADDR[1..0] == B"01"));
!EH_OE/=(!(flash == flashidle) & !FLASH8/ & !WR/ & (LIO_ADDR[1..0] == B"00"));
!W8D_OE/3=(!(flash == flashidle) & !FLASH8/ & !RD/);
!W8D_OE/2=(!(flash == flashidle) & !FLASH8/ & !RD/);
!W8D_OE/1=(!(flash == flashidle) & !FLASH8/ & !RD/);
!W8D_OE/0=(!(flash == flashidle) & !RD/);

%-------------------------------------------------------------------------------%

% state machine design %

% flash - Flash memory control state machine. It controls all reads and writes to
  the on board FLASH as well as the expansion board FLASH. Reads are just like
  normal eprom reads. Writes require software to preform the required command writes
  before the actual write of new data to the FLASH. The following are the supported
  commamds:
  READ/RESET, AUTOSELECT, BYTE PROGRAM, CHIP and SECTOR ERASE
  It does not support sector erase suspend or resume. It assumes that the width of
  the expansion FLASH is the same as the on board FLASH (ie, if on board is 8-bit
  then the expansion must be 8-bit FLASH also). For expansion accesses, it assumes
  one (1) extra cycle at the beginning to allow for buffer delays %

CASE flash IS

WHEN flashidle => % idle state for flash %

% Check for Single BYTE read from FLASH first for 8-bit FLASH%
IF (IO_ADDR28 & (IO_ADDR[27..20] >= flash1_start) & !RD/ & !FLASH8/ &
(IO_SYSCMD[3..0] == byte)) THEN
E8_ADDR0 = LIO_ADDR0;
E8_ADDR1 = LIO_ADDR1;
E8_ADDR2 =LIO_ADDR2;
EDW_ADDR3 =LIO_ADDR3;
EDW_ADDR4 =LIO_ADDR4;
F_cyc_cnt[3..0] = B"0000";
flash = flash1;
% Check for Single HALFWORD read from FLASH for 8-bit FLASH%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] >= flash1_start) & !RD/ & !FLASH8/ &
(IO_SYSCMD[3..0] == halfword)) THEN
E8_ADDR0 = VCC;
E8_ADDR1 = LIO_ADDR1;
E8_ADDR2 =LIO_ADDR2;
EDW_ADDR3 =LIO_ADDR3;
EDW_ADDR4 =LIO_ADDR4;
F_cyc_cnt[3..0] = B"0000";
flash = flash2;
% Check for Single WORD read from FLASH for 8-bit FLASH%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] >= flash1_start) & !RD/ & !FLASH8/ &
(IO_SYSCMD[3..0] == word)) THEN
E8_ADDR0 = VCC;
```

```
E8_ADDR1 = VCC;
E8_ADDR2 =LIO_ADDR2;
EDW_ADDR3 =LIO_ADDR3;
EDW_ADDR4 =LIO_ADDR4;
F_cyc_cnt[3..0] = B"0000";
flash = flash3;
% Check for Single DOUBLEWORD read from FLASH for 8-bit FLASH%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] >= flash1_start) & !RD/ & !FLASH8/ &
(IO_SYSCMD[3..0] == doubleword)) THEN
E8_ADDR0 = VCC;
E8_ADDR1 = VCC;
E8_ADDR2 =VCC;
EDW_ADDR3 =LIO_ADDR3;
EDW_ADDR4 =LIO_ADDR4;
F_cyc_cnt[3..0] = B"0000";
flash = flash5;
% Check for Single non-doubleword read from FLASH for 32-bit FLASH%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] >= flash1_start) & !RD/ & FLASH8/ &
(IO_SYSCMD[3..0] == non_double)) THEN
EW_ADDR0 =LIO_ADDR2;
EDW_ADDR3 =LIO_ADDR3;
EDW_ADDR4 =LIO_ADDR4;
F_cyc_cnt[3..0] = B"0000";
flash = flash7;
% Check for Single doubleword read from FLASH for 32-bit FLASH%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] >= flash1_start) & !RD/ & FLASH8/ &
(IO_SYSCMD[3..0] == doubleword)) THEN
EW_ADDR0 =VCC;
EDW_ADDR3 =LIO_ADDR3;
EDW_ADDR4 =LIO_ADDR4;
F_cyc_cnt[3..0] = B"0000";
flash = flash8;
% Check for block read from FLASH for 32-bit FLASH, with
  64-bit system interface width%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] >= flash1_start) & !RD/ & FLASH8/ &
P3_SYS_IF & (IO_SYSCMD[3..0] == block)) THEN
EW_ADDR0 =VCC;
EDW_ADDR3 =LIO_ADDR3;
EDW_ADDR4 =LIO_ADDR4;
F_cyc_cnt[3..0] = B"0000";
F_word_cnt[2..0] = B"111";
flash = flash9;
% Check for block read from FLASH for 32-bit FLASH, with
  32-bit system interface width%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] >= flash1_start) & !RD/ & FLASH8/ &
!P3_SYS_IF & (IO_SYSCMD[3..0] == block)) THEN
EW_ADDR0 =LIO_ADDR2;
EDW_ADDR3 =LIO_ADDR3;
EDW_ADDR4 =LIO_ADDR4;
F_cyc_cnt[3..0] = B"0000";
F_word_cnt[2..0] = B"111";
flash = flash9;
% Check for a write to 8-bit FLASH. Only byte writes are supported to
  an 8-bit FLASH area%
  ELSIF (IO_ADDR28 & (IO_ADDR[27..20] >= flash1_start) & !WR/ & !FLASH8/ &
  (IO_SYSCMD[3..0] == byte)) THEN
  E8_ADDR0 = LIO_ADDR0;
E8_ADDR1 = LIO_ADDR1;
E8_ADDR2 =LIO_ADDR2;
EDW_ADDR3 =LIO_ADDR3;
EDW_ADDR4 =LIO_ADDR4;
F_cyc_cnt[3..0] = B"0000";
flash = flash13;
% Check for a write to 32-bit FLASH. Only word writes are supported to
  the 32-bit FLASH area%
 ELSIF (IO_ADDR28 & (IO_ADDR[27..20] >= flash1_start) & !WR/ & FLASH8/ &
  (IO_SYSCMD[3..0] == word)) THEN
  EW_ADDR0 =LIO_ADDR2;
```

```
EDW_ADDR3 =LIO_ADDR3;
EDW_ADDR4 =LIO_ADDR4;
F_cyc_cnt[3..0] = B"0000";
flash = flash14;
% Check for a command write to 8-bit FLASH%
  ELSIF (IO_ADDR28 & (IO_ADDR[27..20] == fwcs/) & !WR/ & !FLASH8/ &
  (IO_SYSCMD[3..0] == byte)) THEN
  E8_ADDR0 = LIO_ADDR0;
E8_ADDR1 = LIO_ADDR1;
E8_ADDR2 =LIO_ADDR2;
EDW_ADDR3 =LIO_ADDR3;
EDW_ADDR4 =LIO_ADDR4;
F_cyc_cnt[3..0] = B"0000";
flash = flash13;
% Check for a command write to 32-bit FLASH%
 ELSIF (IO_ADDR28 & (IO_ADDR[27..20] == fwcs/) & !WR/ & FLASH8/ &
  (IO_SYSCMD[3..0] == word)) THEN
  EW_ADDR0 =LIO_ADDR2;
EDW_ADDR3 =LIO_ADDR3;
EDW_ADDR4 =LIO_ADDR4;
F_cyc_cnt[3..0] = B"0000";
flash = flash14;
ELSE
flash = flashidle;
END IF;

WHEN flash1 => % state 1 for flash - single byte read, 8-bit FLASH%
IF (F_cyc_cnt[3..0] == B"0000") THEN
F_cyc_cnt[3..0] = B"0001";
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
flash = flash1;
ELSIF (F_cyc_cnt[3..0] == B"0001") THEN
F_cyc_cnt[3..0] = B"0010";
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
flash = flash1;
ELSIF (F_cyc_cnt[3..0] == B"0010") THEN
W8D_LE/0= GND;
W8D_LE/1= GND;
W8D_LE/2= GND;
W8D_LE/3= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
F_cyc_cnt[3..0] = B"0011";
flash = flash1;
ELSIF (F_cyc_cnt[3..0] == B"0011") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0100";
flash = flash1;
ELSIF (F_cyc_cnt[3..0] == B"0100") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
```

```
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0101”;
flash = flash1;
ELSE
flash = flashidle;
END IF;

 WHEN flash2 => % state 2 for flash - halfword read, 8-bit FLASH, 1st byte%
IF (F_cyc_cnt[3..0] == B”0000”) THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0001”;
flash = flash2;
ELSIF (F_cyc_cnt[3..0] == B”0001”) THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0010”;
flash = flash2;
ELSIF (F_cyc_cnt[3..0] == B”0010”) THEN
W8D_LE/1= GND;
W8D_LE/3= GND;
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0011”;
flash = flash2;
ELSIF (F_cyc_cnt[3..0] == B”0011”) THEN
E8_ADDR0 = GND;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0100”;
flash = flash2;
% halfword read, 8-bit FLASH, 2nd byte%
ELSIF (F_cyc_cnt[3..0] == B”0100”) THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0101”;
flash = flash2;
ELSIF (F_cyc_cnt[3..0] == B”0101”) THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0110”;
flash = flash2;
ELSIF (F_cyc_cnt[3..0] == B”0110”) THEN
W8D_LE/0 = GND;
W8D_LE/2= GND;
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0111”;
```

```
flash = flash2;
ELSIF (F_cyc_cnt[3..0] == B"0111") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"1000";
flash = flash2;
ELSIF (F_cyc_cnt[3..0] == B"1000") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"1001";
flash = flash2;
ELSE
flash = flashidle;
END IF;

WHEN flash3 => % state 3 for flash - word read, 8-bit FLASH, 1st byte%
% or the second word of a single doubleword read%
IF (F_cyc_cnt[3..0] == B"0000") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0001";
flash = flash3;
ELSIF (F_cyc_cnt[3..0] == B"0001") THEN
W8D_LE/3= GND;
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0010";
flash = flash3;
ELSIF (F_cyc_cnt[3..0] == B"0010") THEN
E8_ADDR0 = GND;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0011";
flash = flash3;
ELSIF (F_cyc_cnt[3..0] == B"0011") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0100";
flash = flash3;
% word (or second word of doubleword) read, 8-bit FLASH, 2nd byte %
ELSIF (F_cyc_cnt[3..0] == B"0100") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0101";
flash = flash3;
ELSIF (F_cyc_cnt[3..0] == B"0101") THEN
W8D_LE/2= GND;
E8_ADDR0 = E8_ADDR0;
```

```
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0110”;
flash = flash3;
ELSIF (F_cyc_cnt[3..0] == B”0110”) THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0111”;
flash = flash3;
ELSE
E8_ADDR0 = VCC;
E8_ADDR1 = GND;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0000”;
flash = flash4;
END IF;

WHEN flash4 => % state 4 for flash - word read, 8-bit FLASH, 3rd byte%
% or for the second word of a single doubleword read%
IF (F_cyc_cnt[3..0] == B”0000”) THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0001”;
flash = flash4;
ELSIF (F_cyc_cnt[3..0] == B”0001”) THEN
W8D_LE/1= GND;
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0010”;
flash = flash4;
ELSIF (F_cyc_cnt[3..0] == B”0010”) THEN
E8_ADDR0 = GND;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0011”;
flash = flash4;
ELSIF (F_cyc_cnt[3..0] == B”0011”) THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0100”;
flash = flash4;
% word (doubleword second word) read, 8-bit FLASH, last byte%
ELSIF (F_cyc_cnt[3..0] == B”0100”) THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0101”;
flash = flash4;
```

```
ELSIF (F_cyc_cnt[3..0] == B"0101") THEN
W8D_LE/0= GND;
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0110";
flash = flash4;
ELSIF (F_cyc_cnt[3..0] == B"0110") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0111";
flash = flash4;
ELSIF (F_cyc_cnt[3..0] == B"0111") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"1000";
flash = flash4;
ELSE
flash = flashidle;
END IF;

WHEN flash5 => % state 5 for flash - doubleword read, 8-bit FLASH,%
% 1st byte of first word of the doubleword %
IF (F_cyc_cnt[3..0] == B"0000") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0001";
flash = flash5;
ELSIF (F_cyc_cnt[3..0] == B"0001") THEN
W8D_LE/3= GND;
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0010";
flash = flash5;
ELSIF (F_cyc_cnt[3..0] == B"0010") THEN
E8_ADDR0 = GND;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0011";
flash = flash5;
ELSIF (F_cyc_cnt[3..0] == B"0011") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0100";
flash = flash5;
% first word of doubleword read, 8-bit FLASH, 2nd byte %
ELSIF (F_cyc_cnt[3..0] == B"0100") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
```

```
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0101”;
flash = flash5;
ELSIF (F_cyc_cnt[3..0] == B”0101”) THEN
W8D_LE/2= GND;
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0110”;
flash = flash5;
ELSIF (F_cyc_cnt[3..0] == B”0110”) THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0111”;
flash = flash5;
ELSE
E8_ADDR0 = VCC;
E8_ADDR1 = GND;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0000”;
flash = flash6;
END IF;

WHEN flash6 => % state 6 for flash - doubleword read, 8-bit FLASH,%
% 3rd byte of the first word of the doubleword read%
IF (F_cyc_cnt[3..0] == B”0000”) THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0001”;
flash = flash6;
ELSIF (F_cyc_cnt[3..0] == B”0001”) THEN
W8D_LE/1= GND;
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0010”;
flash = flash6;
ELSIF (F_cyc_cnt[3..0] == B”0010”) THEN
E8_ADDR0 = GND;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0011”;
flash = flash6;
ELSIF (F_cyc_cnt[3..0] == B”0011”) THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B”0100”;
flash = flash6;
% doubleword first word read, 8-bit FLASH, last byte%
```

```
ELSIF (F_cyc_cnt[3..0] == B"0100") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0101";
flash = flash6;
ELSIF (F_cyc_cnt[3..0] == B"0101") THEN
W8D_LE/0= GND;
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0110";
flash = flash6;
ELSIF (F_cyc_cnt[3..0] == B"0110") THEN
E8_ADDR0 = E8_ADDR0;
E8_ADDR1 = E8_ADDR1;
E8_ADDR2 = E8_ADDR2;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0111";
flash = flash6;
ELSE
E8_ADDR0 = VCC;
E8_ADDR1 = VCC;
E8_ADDR2 = GND;
IOEX_LEA1B = GND;
IOEX_LEA2B = VCC;
F_cyc_cnt[3..0] = B"0000";
flash = flash3;
END IF;

WHEN flash7 =>% state7 for flash - single non_double read, 32-bit FLASH,%
% or second word of doubleword read%
IF (F_cyc_cnt[3..0] == B"0000") THEN
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0001";
flash = flash7;
ELSIF (F_cyc_cnt[3..0] == B"0001") THEN
W8D_LE/0= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0010";
flash = flash7;
ELSIF (F_cyc_cnt[3..0] == B"0010") THEN
W8D_LE/0= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0011";
flash = flash7;
ELSIF (F_cyc_cnt[3..0] == B"0011") THEN
W8D_LE/0= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0100";
flash = flash7;
ELSIF (F_cyc_cnt[3..0] == B"0100") THEN
EW_ADDR0 = EW_ADDR0;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0101";
flash = flash7;
ELSE
flash = flashidle;
```

```
END IF;

WHEN flash8 =>%state8 for flash - single doubleword read, 32-bit FLASH%
IF (F_cyc_cnt[3..0] == B"0000") THEN
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0001";
flash = flash8;
ELSIF (F_cyc_cnt[3..0] == B"0001") THEN
W8D_LE/0= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0010";
flash = flash8;
ELSIF (F_cyc_cnt[3..0] == B"0010") THEN
W8D_LE/0= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0011";
flash = flash8;
ELSIF (F_cyc_cnt[3..0] == B"0011") THEN
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0100";
flash = flash8;
ELSE
EW_ADDR0 = GND;
IOEX_LEA1B = GND;
IOEX_LEA2B = VCC;
F_cyc_cnt[3..0] = B"0000";
flash = flash7;
END IF;

WHEN flash9 =>% state17 for flash - block read, 32-bit FLASH%
% 1st or 5th word of block%

EW_ADDR0 = !EW_ADDR0;

IF (F_cyc_cnt[3..0] == B"0000") THEN
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0001";
flash = flash9;
ELSIF (F_cyc_cnt[3..0] == B"0001") THEN
W8D_LE/0= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0010";
flash = flash9;
ELSIF (F_cyc_cnt[3..0] == B"0010") THEN
W8D_LE/0= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0011";
flash = flash9;
ELSIF (F_cyc_cnt[3..0] == B"0011") THEN
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0100";
flash = flash9;
ELSIF ((F_cyc_cnt[3..0] == B"0100") & (F_word_cnt[2..0] == B"111")) THEN
IOEX_LEA1B = GND # !P3_SYS_IF;
IOEX_LEA2B = VCC # !P3_SYS_IF;
F_cyc_cnt[3..0] = B"0000";
F_word_cnt[2..0] =  B"110";
flash = flash10;
ELSE
IOEX_LEA1B = GND # !P3_SYS_IF;
```

```
IOEX_LEA2B = VCC # !P3_SYS_IF;
F_cyc_cnt[3..0] = B"0000";
F_word_cnt[2..0] = B"010";
flash = flash10;
END IF;

WHEN flash10 =>% state10 for flash - block read, 32-bit FLASH%
% 2nd or 6th word of block%

EW_ADDR0 = !EW_ADDR0;

IF (F_cyc_cnt[3..0] == B"0000") THEN
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0001";
flash = flash10;
ELSIF (F_cyc_cnt[3..0] == B"0001") THEN
W8D_LE/0= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0010";
flash = flash10;
ELSIF (F_cyc_cnt[3..0] == B"0010") THEN
W8D_LE/0= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0011";
flash = flash10;
ELSIF (F_cyc_cnt[3..0] == B"0011") THEN
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0100";
flash = flash10;
ELSIF ((F_cyc_cnt[3..0] == B"0100") & (F_word_cnt[2..0] == B"110")) THEN
EDW_ADDR3 = !EDW_ADDR3;
IOEX_LEA1B = GND;
IOEX_LEA2B = GND;
F_cyc_cnt[3..0] = B"0101";
F_word_cnt[2..0] = B"101";
flash = flash10;
ELSIF ((F_cyc_cnt[3..0] == B"0100") & (F_word_cnt[2..0] == B"010")) THEN
EDW_ADDR3 = !EDW_ADDR3;
IOEX_LEA1B = GND;
IOEX_LEA2B = GND;
F_cyc_cnt[3..0] = B"0101";
F_word_cnt[2..0] = B"001";
flash = flash10;
ELSE
IOEX_LEA1B = VCC # !P3_SYS_IF;
IOEX_LEA2B = GND # !P3_SYS_IF;
F_cyc_cnt[3..0] = B"0000";
flash = flash11;
END IF;

WHEN flash11 =>% state11 for flash - block read, 32-bit FLASH%
% 3rd or 7th word of block%

EW_ADDR0 = !EW_ADDR0;

IF (F_cyc_cnt[3..0] == B"0000") THEN
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0001";
flash = flash11;
ELSIF (F_cyc_cnt[3..0] == B"0001") THEN
W8D_LE/0= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
```

```
F_cyc_cnt[3..0] = B"0010";
flash = flash11;
ELSIF (F_cyc_cnt[3..0] == B"0010") THEN
W8D_LE/0= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0011";
flash = flash11;
ELSIF (F_cyc_cnt[3..0] == B"0011") THEN
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0100";
flash = flash11;
ELSIF ((F_cyc_cnt[3..0] == B"0100") & (F_word_cnt[2..0] == B"101")) THEN
IOEX_LEA1B = GND # !P3_SYS_IF;
IOEX_LEA2B = VCC # !P3_SYS_IF;
F_cyc_cnt[3..0] = B"0000";
F_word_cnt[2..0] = B"100";
flash = flash12;
ELSE
IOEX_LEA1B = GND # !P3_SYS_IF;
IOEX_LEA2B = VCC # !P3_SYS_IF;
F_cyc_cnt[3..0] = B"0000";
F_word_cnt[2..0] = B"000";
flash = flash12;
END IF;

WHEN flash12 =>% state12 for flash - block read, 32-bit FLASH%
% 4th or 8th word of block%

EW_ADDR0 = !EW_ADDR0;

IF (F_cyc_cnt[3..0] == B"0000") THEN
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0001";
flash = flash12;
ELSIF (F_cyc_cnt[3..0] == B"0001") THEN
W8D_LE/0= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0010";
flash = flash12;
ELSIF (F_cyc_cnt[3..0] == B"0010") THEN
W8D_LE/0= GND;
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0011";
flash = flash12;
ELSIF (F_cyc_cnt[3..0] == B"0011") THEN
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0100";
flash = flash12;
ELSIF ((F_cyc_cnt[3..0] == B"0100") & (F_word_cnt[2..0] == B"100")) THEN
EDW_ADDR3 = !EDW_ADDR3;
EDW_ADDR4 = !EDW_ADDR4;
IOEX_LEA1B = GND;
IOEX_LEA2B = GND;
F_word_cnt[2..0] = B"011";
F_cyc_cnt[3..0] = B"0101";
flash = flash12;
ELSIF ((F_cyc_cnt[3..0] == B"0100") & (F_word_cnt[2..0] == B"000")) THEN
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0101";
flash = flash12;
ELSIF ((F_cyc_cnt[3..0] == B"0101") & (F_word_cnt[2..0] == B"011")) THEN
```

```
IOEX_LEA1B = VCC # !P3_SYS_IF;
IOEX_LEA2B = GND # !P3_SYS_IF;
F_cyc_cnt[3..0] = B"0000";
flash = flash9;
ELSIF (F_cyc_cnt[3..0] == B"0101") THEN
IOEX_LEA1B = IOEX_LEA1B;
IOEX_LEA2B = IOEX_LEA2B;
F_cyc_cnt[3..0] = B"0110";
flash = flash12;
ELSE
flash = flashidle;
END IF;

WHEN flash13 =>% state13 for flash - byte write, 8-bit FLASH%
IF (F_cyc_cnt[3..0] == B"0000") THEN
F_cyc_cnt[3..0] = B"0001";
flash = flash13;
ELSIF (F_cyc_cnt[3..0] == B"0001") THEN
WE_8/ = GND;
F_cyc_cnt[3..0] = B"0010";
flash = flash13;
ELSIF (F_cyc_cnt[3..0] == B"0010") THEN
WE_8/ = GND;
F_cyc_cnt[3..0] = B"0011";
flash = flash13;
ELSIF (F_cyc_cnt[3..0] == B"0011") THEN
WE_8/ = GND;
F_cyc_cnt[3..0] = B"0100";
flash = flash13;
ELSE
flash = flashidle;
END IF;

WHEN flash14 =>% state14 for flash - word write, 32-bit FLASH%
IF (F_cyc_cnt[3..0] == B"0000") THEN
F_cyc_cnt[3..0] = B"0001";
flash = flash14;
ELSIF (F_cyc_cnt[3..0] == B"0001") THEN
WE_8/ = GND;
WE_W3/ = GND;
WE_W2/ = GND;
WE_W1/ = GND;
F_cyc_cnt[3..0] = B"0010";
flash = flash14;
ELSIF (F_cyc_cnt[3..0] == B"0010") THEN
WE_8/ = GND;
WE_W3/ = GND;
WE_W2/ = GND;
WE_W1/ = GND;
F_cyc_cnt[3..0] = B"0011";
flash = flash14;
ELSIF (F_cyc_cnt[3..0] == B"0011") THEN
WE_8/ = GND;
WE_W3/ = GND;
WE_W2/ = GND;
WE_W1/ = GND;
F_cyc_cnt[3..0] = B"0100";
flash = flash14;
ELSE
flash = flashidle;
END IF;

END CASE;

%-------------------------------------------------------------------------------%

END;
%
```

IO state Machine

MAX+PLUS II AHDLv5.0

EDIT HISTORY

Date	Engineer	Check Sum	Changes
10/26/94	R.Cummings		Initial Release

%

IO_FLASH State Machine

```
%============================================================================%

% I/O memory Map
Reserved0x1F00_0000 to 0x1F2F_FFFF
SCC (85C30)0x1F30_0000 to 0x1F3F_FFFF
SCSI (53C90A)0x1F40_0000 to 0x1F4F_FFFF
NVRAM0x1F50_0000 to 0x1F5F_FFFF
Ethernet0x1F60_0000 to 0x1F6F_FFFF(In SONIC Controller)
Expansion CS0x1F70_0000 to 0x1F7F_FFFF(In SONIC Controller)
LED CS0x1F80_0000 to 0x1F8F_FFFF
Reserved0x1F90_0000 to 0x1FAF_FFFF
Flash Write0x1FB0_0000 to 0x1FBF_FFFF (setup writes for real write to flash)
Flash Bank00x1FC0_0000 to 0x1FDF_FFFF(On Mother Board)
Flash Bank10x1FE0_0000 to 0x1FFF_FFFF(On Expansion Connector)
%
%============================================================================%

CONSTANTscccs/=h"F3";
CONSTANTscsics/=h"F4";
CONSTANT nvramcs/=h"F5";
CONSTANTledcs/=h"F8";
CONSTANTfwcs/=h"FB";
CONSTANTflash1_start=h"FC";
CONSTANT flash1_top=h"FD";
CONSTANTflash2_start=h"FE";
CONSTANT flash2_top=h"FF";
CONSTANT byte=h"8";
CONSTANT halfword=h"9";
CONSTANT word=h"B";
CONSTANT doubleword=h"F";
CONSTANT non_double=B"10XX";
CONSTANT block=B"0XXX";

%============================================================================%

SUBDESIGN IO
(
% system interface control input signals %
IO_CLK1:INPUT;% One of the IO Clocks from the Main Controller%
IO_RESET/:INPUT;% system IO reset from RESET controller%
WR/:INPUT;% write signal from MAIN Controller%
RD/:INPUT;% read signal from MAIN Controller%

% address input signals %
LIO_ADDR[4..2]:INPUT;% Lower address lines%
NV_ADDR11:INPUT;% Actually IO_ADDR11. Indicates if NVRAM access%
% is read/write (=0) or recall/store (=1)%
LED_ADDR5:INPUT;% Actually IO_ADDR5. Indicates if LED access%
% is for cursor (=0) or for display (=1)%
IO_ADDR[28..20]:INPUT;% bit 28 Determines IO (= 1) or memory (= 0) %
% access. [23:20] determine device accessed.%

% ioflash acknowledge signal%
Other_IO_Done/:OUTPUT;% Access done to sonic controller %

% Misc IO outputs%
```

```
IO_Access/:OUTPUT;% Indicates an IO access is in progress%

% NVRAM Control signals%
NV_WE/:OUTPUT;% NVRAM write enable%
NV_OE/:OUTPUT;% NVRAM output enable%
NV_NE/:OUTPUT;% NVRAM nonvolatile enable%
NV_CE/:OUTPUT;% NVRAM chip enable%

% LED Control signals%
LED_CE/:OUTPUT;% LED Display chip enable (both CE1/ and CE2/)%
LED_CLR/:OUTPUT;% LED Display clear%
LED_CUE:OUTPUT;% LED Display cursor enable%
LED_CU/:OUTPUT;% LED Display cursor select%
LED_WR/:OUTPUT;% LED Display write signal%
LED_BL/:OUTPUT;% LED Display blank signal%

% scsi control signals %
SCSI_CS/:OUTPUT;% SCSI Chip select%
SCSI_WR/:OUTPUT;% write signal to 53c90 %
SCSI_RD/:OUTPUT;% read signal to 53c90%

% scc control signals %
PCLK:OUTPUT;% SCC clock. IO_CLK1 / 2 (TCLK/4)%
SCC_CE/:OUTPUT;% used to select the 8530 controller %
SCC_RD/:OUTPUT;% read signal to 8530 %
SCC_WR/:OUTPUT;% write signal to 8530 %
)

%============================================================================%

VARIABLE
Other_IO_Done/:DFF;
SCSI_CS/:DFF;
SCSI_WR/:DFF;
SCSI_RD/:DFF;
SCC_CE/:DFF;
SCC_RD/:DFF;
SCC_WR/:DFF;
PCLK:DFF;
NV_WE/:DFF;
NV_OE/:DFF;
NV_NE/:DFF;
NV_CE/:DFF;
LED_CE/:DFF;
LED_CLR/:DFF;
LED_CUE:DFF;
LED_CU/:DFF;
LED_WR/:DFF;
LED_BL/:DFF;
IO_Access/:DFF;

%----------------------------------------------------------------------------%

%the io state machine is subdivided into sub-state machines.
each handles a spesific io or a number of different io %

scsi:MACHINE WITH STATES(
scsiidle,
scsi1,scsi2,scsi3,scsi4
);

scc:MACHINE WITH STATES (
sccidle,
scc1,scc2,scc3,scc4,scc5,scc6,
scc7,scc8,scc9,scc10,scc11,scc12,scc13
);

 nvram:MACHINE WITH STATES (
```

```
nvramidle,
nvram1, nvram2, nvram3,
nvram4, nvram5, nvram6, nvram7
);

led:MACHINE WITH STATES (
ledidle,
led1, led2, led3, led4
);

%-------------------------------------------------------------------------------%
% Buried nodes%

VIACLK:DFF;% PCLK /2%

%===============================================================================%

BEGIN

%-------------------------------------------------------------------------------%

% Default state for all outputs is not asserted %

DEFAULTS
Other_IO_Done/=VCC;
SCSI_CS/=VCC;
SCSI_WR/=VCC;
SCSI_RD/=VCC;
SCC_CE/=VCC;
SCC_RD/=VCC;
SCC_WR/=VCC;
PCLK=VCC;
NV_WE/=VCC;
NV_OE/=VCC;
NV_NE/=VCC;
NV_CE/=VCC;
LED_CE/=VCC;
LED_CU/=VCC;
LED_WR/=VCC;
VIACLK=VCC;
IO_Access/=VCC;
END DEFAULTS;

%-------------------------------------------------------------------------------%

Other_IO_Done/.clk = GLOBAL(IO_CLK1);%Define clock for IOAck/ output %
Other_IO_Done/.prn = GLOBAL(IO_RESET/);%Define preset condition for IOAck/ output %

SCSI_CS/.clk = GLOBAL(IO_CLK1);%Define clock for SCSI_CS/ output %
SCSI_CS/.prn = GLOBAL(IO_RESET/);%Define preset condition for SCSI_CS/ output %

SCSI_WR/.clk = GLOBAL(IO_CLK1);%Define clock for SCSI_WR/ output %
SCSI_WR/.prn = GLOBAL(IO_RESET/);%Define preset condition for SCSI_WR/ output %

SCSI_RD/.clk = GLOBAL(IO_CLK1);%Define clock for SCSI_RD/ output %
SCSI_RD/.prn = GLOBAL(IO_RESET/);%Define preset condition for SCSI_RD/ output %

SCC_CE/.clk = GLOBAL(IO_CLK1);%Define clock for SCC_CE/ output %
SCC_CE/.prn = GLOBAL(IO_RESET/);%Define preset condition for SCC_CE/ output %

SCC_WR/.clk = GLOBAL(IO_CLK1);%Define clock for SCC_WR/ output %
SCC_WR/.prn = GLOBAL(IO_RESET/);%Define preset condition for SCC_WR/ output %

SCC_RD/.clk = GLOBAL(IO_CLK1);%Define clock for SCC_RD/ output %
SCC_RD/.prn = GLOBAL(IO_RESET/);%Define preset condition for SCC_RD/ output %

PCLK.clk = GLOBAL(IO_CLK1);
PCLK.prn = GLOBAL(IO_RESET/);
```

```
NV_WE/.clk = GLOBAL(IO_CLK1);
NV_WE/.prn = GLOBAL(IO_RESET/);

NV_OE/.clk = GLOBAL(IO_CLK1);
NV_OE/.prn = GLOBAL(IO_RESET/);

NV_NE/.clk = GLOBAL(IO_CLK1);
NV_NE/.prn = GLOBAL(IO_RESET/);

NV_CE/.clk = GLOBAL(IO_CLK1);
NV_CE/.prn = GLOBAL(IO_RESET/);

LED_CE/.clk = GLOBAL(IO_CLK1);
LED_CE/.prn = GLOBAL(IO_RESET/);

LED_CLR/.clk = GLOBAL(IO_CLK1);
LED_CLR/.prn = GLOBAL(IO_RESET/);

LED_CUE.clk = GLOBAL(IO_CLK1);
LED_CUE.clrn = GLOBAL(IO_RESET/);

LED_CU/.clk = GLOBAL(IO_CLK1);
LED_CU/.prn = GLOBAL(IO_RESET/);

LED_WR/.clk = GLOBAL(IO_CLK1);
LED_WR/.prn = GLOBAL(IO_RESET/);

LED_BL/.clk = GLOBAL(IO_CLK1);
 LED_BL/.prn = GLOBAL(IO_RESET/);

 IO_Access/.clk = GLOBAL(IO_CLK1);
 IO_Access/.prn = GLOBAL(IO_RESET/);

 scsi.clk= GLOBAL(IO_CLK1);
 scsi.reset = GLOBAL(!IO_RESET/);

 scc.clk= GLOBAL(IO_CLK1);
 scc.reset = GLOBAL(!IO_RESET/);

 nvram.clk = GLOBAL(IO_CLK1);
 nvram.reset = GLOBAL(!IO_RESET/);

 led.clk = GLOBAL(IO_CLK1);
 led.reset = GLOBAL(!IO_RESET/);

%-------------------------------------------------------------------------------%
% Buried nodes%

VIACLK.clk=PCLK;
VIACLK.prn=GLOBAL(IO_RESET/);

%-------------------------------------------------------------------------------%

% equations %

!Other_IO_Done/ =((nvram == nvram4) #% assert for nvram access done%
(led == led2) #% assert for led access done%
(scc == scc11)); % only have SCC control this for now for debug%
% REC 070595 - changes made to allow for SONIC%
% in the system and other IO done one cycle %
% earlier to SONIC controller%
% Changed to 4 for nvram access - REC 080195%
%!Other_IO_Done/ =((nvram == nvram4) # assert for nvram access done%
%(led == led3) # assert for led access done%
%(scc == scc12));  only have SCC control this for now for debug%
% REC 041295%
```

```
%IO_Access/ = !(IO_Addr28 & ((IO_ADDR[27..20] == scccs/) #
(IO_ADDR[27..20] == scsics/) #
(IO_ADDR[27..20] == nvramcs/) #
(IO_ADDR[27..20] == ledcs/) #
(IO_ADDR[27..20] == fwcs/) #
(IO_ADDR[27..20] == flash1_start) #
(IO_ADDR[27..20] == flash1_top) #
(IO_ADDR[27..20] == flash2_start) #
(IO_ADDR[27..20] == flash2_top)));%
% IO_Access experiment - REC - 041895 - 1540%
IO_Access/ = !(IO_Addr28);

PCLK=!PCLK;

VIACLK=!VIACLK;

%--------------------------------------------------------------------------%

% state machine design %

% scsi state machine to control transfer from/to the 53C90 scsi controller. It can
  do both read/write operations.%

CASE scsi IS

WHEN scsiidle => % idle state for scsi %
!SCSI_CS/=GND;
!SCSI_WR/=GND;
!SCSI_RD/=GND;

% check for scsi write %
IF (IO_ADDR28 & (IO_ADDR[27..20] == scsics/) & !WR/) THEN
!SCSI_CS/=VCC; % assert SCSI_CS/ %
!SCSI_WR/=VCC; % assert SCSI_WR/ %
!SCSI_RD/=GND;
scsi =scsi1; % go to state 1 %

% check for scsi read %
 ELSIF (IO_ADDR28 & (IO_ADDR[27..20] == scsics/) & !RD/) THEN
!SCSI_CS/=VCC; % assert SCSI_CS/ %
!SCSI_WR/=GND;
!SCSI_RD/=VCC; % assert SCSI_RD/ %
scsi =scsi1; % go to state 1 %

% if no scsi access, return to idle %
ELSE
scsi=scsiidle;
END IF;

% state 1, keep asserted signals in the same state %
WHEN scsi1 =>
!SCSI_CS/=!SCSI_CS/; % keep as is %
!SCSI_WR/=!SCSI_WR/; % keep as is %
!SCSI_RD/=!SCSI_RD/; % keep as is %
scsi=scsi2; % go to state 2 %

% if scsi write, exit state 2 and negate all signals %
% if scsi read, stay for one more clock cycle %
WHEN scsi2 =>
IF (!SCSI_WR/) THEN % scsi write %
!SCSI_CS/=GND;
!SCSI_WR/=GND;
!SCSI_RD/=GND; % negate all signals %
scsi=scsi4; % go to state 4 %

ELSE% scsi read %
!SCSI_CS/=!SCSI_CS/; % keep as is %
!SCSI_WR/=!SCSI_WR/; % keep as is %
```

```
!SCSI_RD/=!SCSI_RD/; % keep as is %
scsi=scsi3; % go to state 3 %
END IF;

% will get to state 3 only during scsi read operations %
WHEN scsi3 =>% keep all signals as is %
!SCSI_CS/=!SCSI_CS/; % keep as is %
!SCSI_WR/=!SCSI_WR/; % keep as is %
!SCSI_RD/=!SCSI_RD/; % keep as is %
scsi=scsi4; % go to state 4 %

% negate all signals and exit %
WHEN scsi4 =>
!SCSI_CS/=GND;
!SCSI_WR/=GND;
!SCSI_RD/=GND; % negate all signals %
scsi=scsiidle; % go to  idle state %

END CASE;

%-------------------------------------------------------------------------------%

% scc state machine to control transfer from/to the 85C30 scc controller. It supports
  normal read/write operations as well as reset operations. The address of the access
  determines the type of the access. If a4 = 0, normal read or write operations. If
  a4 = 1, reset operation. For a reset operation, noth the SCC_RD/ and the SCC_WR/
  signals are asserted together.  The scc requires a revovery time of 3.5 pclk min. This recovery time is an
  integral part of the scc state machine and is implemented at the front
  end of the transfer. This insures that subsequent accesses to the 8530
  meet the required recovery time. Further, this makes the design of the overall
  state machine of the board synchronous without overlapping the transfer of
  another io with the recovery time of the scc. %

CASE scc IS

WHEN sccidle =>% idle state for scc %
!SCC_CE/=GND;
!SCC_WR/=GND;
!SCC_RD/=GND;

% check for an scc access %
IF (IO_ADDR28 & (IO_ADDR[27..20] == scccs/) & (!WR/ # !RD/)) THEN
scc=scc1; % go to state 1 %
ELSE
scc=sccidle;
END IF;

% this is where the recovery time of the scc state machine starts %
% The scc requires 3.5 PCLK of recovery time. The scc state machine implements a maximum of 6 PCLK as the
recovery time (between 5 and 6 PCLK).  The VIACLK is used as the counting mechanism for the recovery time.

TCLK = 50 mhz (20 nsec)
PCLK = TCLK/4 (80 nsec)
VIACLK = PCLK/2 (160 nsec)
%

% At the end of state 1, the scc state machine could have waited between
  0 and 4 TCLK cycles (0 to 1 PCLK). This is determined by the phase
  relationship between VIACLK and TCLK.  At the end of state 1, the state machine waited 0 to 80 nsec. %

WHEN scc1 =>
!SCC_CE/=GND;
!SCC_WR/=GND;
!SCC_RD/=GND; % negate all signals %
IF (!VIACLK) THEN% check viaclk, if right phase ... %
scc =scc2; % go to state 2 %
ELSE
scc=scc1; % otherwise, loop in state 1 %
```

```
END IF;

% at the end of state 2, the state machine waited 80 to 160 nsec. %
WHEN scc2 =>
!SCC_CE/=GND;
!SCC_WR/=GND;
!SCC_RD/=GND; % negate all signals %
IF (VIACLK) THEN% check viaclk, if right phase ... %
scc =scc3; % go to state 3 %
ELSE
scc=scc2; % otherwise, loop in state 2 %
END IF;

% at the end of state 3, the state machine waited 160 to 240 nsec. %
WHEN scc3 =>
!SCC_CE/=GND;
!SCC_WR/=GND;
!SCC_RD/=GND; % negate all signals %
IF (!VIACLK) THEN% check viaclk, if right phase ... %
scc =scc4; % go to state 4 %
ELSE
scc=scc3; % otherwise, loop in state 3 %
END IF;

% at the end of state 4, the state machine waited 240 to 320 nsec. %
WHEN scc4 =>
!SCC_CE/=GND;
!SCC_WR/=GND;
!SCC_RD/=GND; % negate all signals %
IF (VIACLK) THEN% check viaclk, if right phase ... %
scc =scc5; % go to state 5 %
ELSE
scc=scc4; % otherwise, loop in state 4 %
END IF;

% at the end of state 5, the state machine waited 320 to 400 nsec. %
WHEN scc5 =>
!SCC_CE/=GND;
!SCC_WR/=GND;
!SCC_RD/=GND; % negate all signals %
IF (!VIACLK) THEN% check viaclk, if right phase ... %
scc =scc6; % go to state 6 %
ELSE
scc=scc5; % otherwise, loop in state 5 %
END IF;

% at the end of state 6, the state machine waited 400 to 480 nsec. %
WHEN scc6 =>
!SCC_CE/=GND;
!SCC_WR/=GND;
!SCC_RD/=GND; % negate all signals %
IF (VIACLK) THEN% check viaclk, if right phase ... %
scc =scc7; % go to state 7 %
ELSE
scc=scc6; % otherwise, loop in state 6 %
END IF;

% this is the end of the recovery time. from here, the actual access
  to the scc controller starts %
WHEN scc7 =>
!SCC_CE/=VCC; % assert SCC_CE/ %
!SCC_WR/=GND;
!SCC_RD/=GND;
scc=scc8; % go to state 8 %

WHEN scc8 =>
!SCC_CE/=VCC; % assert SCC_CE/ %
!SCC_WR/=(!WR/ & IO_ADDR[20]) #% assert for a write operation or %
```

```
(LIO_ADDR[4] & IO_ADDR[20]) # % for a reset operation or %
(!SCC_WR/); % if it were previously asserted %
!SCC_RD/=(!RD/ & IO_ADDR[20]) # % assert for a read operation or %
(LIO_ADDR[4] & IO_ADDR[20]) # % for a reset operation or %
(!SCC_RD/); % if it were previously asserted %
scc=scc9; % go to state 9 %

WHEN scc9 =>
!SCC_CE/=VCC; % assert SCC_CE/ %
!SCC_WR/=!SCC_WR/;% keep as is %
!SCC_RD/=!SCC_RD/;% keep as is %
scc=scc10; % go to state 10 %

WHEN scc10 =>
!SCC_CE/=VCC; % assert SCC_CE/ %
!SCC_WR/=!SCC_WR/;% keep as is %
!SCC_RD/=!SCC_RD/;% keep as is %
scc=scc11; % go to state 11 %

WHEN scc11 =>
!SCC_CE/=VCC; % assert SCC_CE/ %
!SCC_WR/=!SCC_WR/;% keep as is %
!SCC_RD/=!SCC_RD/;% keep as is %
scc=scc12; % go to state 12 %

% negate all signals and exit %
WHEN scc12 =>
!SCC_CE/=VCC; % assert SCC_CE/ %
!SCC_WR/=!SCC_WR/;% keep as is %
!SCC_RD/=!SCC_RD/;% keep as is %
scc=scc13; % go to state 13%

% Added wait state so scc does not retrigger - REC 041295 - 1111 %
WHEN scc13 =>
!SCC_CE/=GND;
!SCC_WR/=GND;
!SCC_RD/=GND;% negate all signals  %
scc=sccidle; % go to idle state %

END CASE;

% nvram state machine to control transfer from/to the NVRAM. It can do both read/write operations as well as
store/recall operations.%

CASE nvram IS

WHEN nvramidle => % idle state for nvram %
NV_WE/=VCC;
NV_OE/=VCC;
NV_NE/=VCC;
NV_CE/=VCC;

% check for NVRAM read%
IF (IO_ADDR28 & (IO_ADDR[27..20] == nvramcs/) & !NV_ADDR11 & !RD/) THEN
NV_OE/ = GND;
NV_CE/ = GND;
NV_WE/ =VCC;
NV_NE/ =VCC;
nvram = nvram1;

% check for NVRAM write%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] == nvramcs/) & !NV_ADDR11 & !WR/) THEN
NV_OE/ = VCC;
NV_CE/ = GND;
NV_WE/ =VCC;
NV_NE/ =VCC;
nvram = nvram7;
```

```
% check for NVRAM recall%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] == nvramcs/) & NV_ADDR11 & !RD/) THEN
NV_OE/ = GND;
NV_CE/ = GND;
NV_WE/ =VCC;
NV_NE/ =GND;
nvram = nvram3;% can use fewer cycles as start of recall only needs%
% 100ns as opposed to read/write cycle of 150ns%

% check for NVRAM store%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] == nvramcs/) & NV_ADDR11 & !WR/) THEN
NV_OE/ = VCC;
NV_CE/ = GND;
NV_WE/ =GND;
NV_NE/ =GND;
nvram = nvram3;% can use fewer cycles as start of store only needs%
% 100ns as opposed to read/write cycle of 150ns%

ELSE
nvram = nvramidle;
END IF;

WHEN nvram1 =>
% state 1 for nvram - keep all signals in same state%
NV_OE/ = NV_OE/;
NV_CE/ = NV_CE/;
NV_WE/ =NV_WE/;
NV_NE/ =NV_NE/;
nvram = nvram2;

WHEN nvram2 =>
% state 2 for nvram - keep all signals in same state%
NV_OE/ = NV_OE/;
NV_CE/ = NV_CE/;
NV_WE/ =NV_WE/;
NV_NE/ =NV_NE/;
nvram = nvram3;

WHEN nvram3 =>
% state 3 for nvram - keep all signals in same state%
NV_OE/ = NV_OE/;
NV_CE/ = NV_CE/;
NV_WE/ =NV_WE/;
NV_NE/ =NV_NE/;
nvram = nvram4;

WHEN nvram4 =>
% state 4 for nvram - keep all signals in same state%
NV_OE/ = NV_OE/;
NV_CE/ = NV_CE/;
NV_WE/ =NV_WE/;
NV_NE/ =NV_NE/;
nvram = nvram5;

WHEN nvram5 =>
% state 5 for nvram - keep all signals in same state%
NV_OE/ = NV_OE/;
NV_CE/ = NV_CE/;
NV_WE/ =NV_WE/;
NV_NE/ =NV_NE/;
nvram = nvram6;

WHEN nvram6 =>
% state 6 for nvram - negate all signals and return to idle%
NV_OE/ = VCC;
NV_CE/ = VCC;
NV_WE/ =VCC;
NV_NE/ =VCC;
```

```
nvram = nvramidle;

WHEN nvram7 =>
% state 7 for nvram - write so now assert WE and goto state 2%
NV_OE/ = NV_OE/;
NV_CE/ = NV_CE/;
NV_WE/ =GND;
NV_NE/ =NV_NE/;
nvram = nvram2;

END CASE;
```

% led - LED Display state machine. It controls all writes to the LED DIsplay - including reset, show blocks, blanking and setting and displaying of the various characters and cursor positions.%

```
CASE led IS

WHEN ledidle => % idle state for led %

% Check for clear write%
IF (IO_ADDR28 & (IO_ADDR[27..20] == ledcs/) & !WR/ &
(LIO_ADDR[4..2] == B"000")) THEN
LED_CLR/ = !LED_CLR/;
LED_BL/ = LED_BL/;
LED_CUE = LED_CUE;
LED_CU/ = VCC;
LED_WR/ = VCC;
LED_CE/ = VCC;
led = led1;

% Check for display blocks%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] == ledcs/) & !WR/ &
(LIO_ADDR[4..2] == B"001")) THEN
LED_CLR/ = LED_CLR/;
LED_BL/ = LED_BL/;
LED_CUE = !LED_CUE;
LED_CU/ = VCC;
LED_WR/ = VCC;
LED_CE/ = VCC;
led = led1;

% Check for blank display%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] == ledcs/) & !WR/ &
(LIO_ADDR[4..2] == B"010")) THEN
LED_CLR/ = LED_CLR/;
LED_BL/ = !LED_BL/;
LED_CUE = LED_CUE;
LED_CU/ = VCC;
LED_WR/ = VCC;
LED_CE/ = VCC;
led = led1;

% Else write cursor character%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] == ledcs/) & !WR/ &
(LIO_ADDR[4..2] == B"1XX") & !LED_ADDR5) THEN
LED_CLR/ = LED_CLR/;
LED_BL/ = LED_BL/;
LED_CUE = VCC;
LED_CU/ = GND;
LED_WR/ = GND;
LED_CE/ = GND;
led = led1;

% Else write display character%
ELSIF (IO_ADDR28 & (IO_ADDR[27..20] == ledcs/) & !WR/ &
(LIO_ADDR[4..2] == B"1XX") & LED_ADDR5) THEN
LED_CLR/ = LED_CLR/;
LED_BL/ = LED_BL/;
```

```
LED_CUE = GND;
LED_CU/ = VCC;
LED_WR/ = GND;
LED_CE/ = GND;
led = led1;

ELSE
LED_CLR/ = LED_CLR/;
LED_BL/ = LED_BL/;
LED_CUE = LED_CUE;
led = ledidle;
END IF;

WHEN led1 => % led state 1 %
LED_CLR/ = LED_CLR/;
LED_BL/ = LED_BL/;
LED_CUE = LED_CUE;
LED_CU/ = LED_CU/;
LED_WR/ = LED_WR/;
LED_CE/ = LED_CE/;
led = led2;

WHEN led2 => % led state 2 %
LED_CLR/ = LED_CLR/;
LED_BL/ = LED_BL/;
LED_CUE = LED_CUE;
LED_CU/ = LED_CU/;
LED_WR/ = LED_WR/;
LED_CE/ = LED_CE/;
led = led3;

WHEN led3 => % led state 3 %
LED_CLR/ = LED_CLR/;
LED_BL/ = LED_BL/;
LED_CUE = LED_CUE;
LED_CU/ = LED_CU/;
LED_WR/ = LED_WR/;
LED_CE/ = LED_CE/;
led = led4;

WHEN led4 => % led state 4 %
LED_CLR/ = LED_CLR/;
LED_BL/ = LED_BL/;
LED_CUE = LED_CUE;
LED_CU/ = VCC;
LED_WR/ = VCC;
LED_CE/ = VCC;
led = ledidle;

END CASE;

%-------------------------------------------------------------------------------%

END;
%
```

MAIN Controller PLD

MAX+PLUS II AHDL v4.02

EDIT HISTORY

Date	Engineer	Checksum	Change	Comments
S. Khan			initial release	

%

Main Controller

```
DESIGN IS “main_ctr”;
%DEVICE “main_ctr” IS “EPM7096LC84-10”;%


%_____CONSTANTS _____ %

%  Combinations for 2nd_cyc    %

CONSTANT idle = B”0”;
CONSTANT 2nd_cyc_serv = B”1”;
%_____
2nd_cycDescription

0idle
1Second cycle buffered
----------%

CONSTANT  last_read = B”1000”;
CONSTANT  no_last_read  = B”1100”;
CONSTANT  last_err_data = B”1001”;
CONSTANT  no_last_err_data = B”1101”;
CONSTANT  null_req= B”001100000”;
CONSTANT  Res_bits= B”0000”;


SUBDESIGN main_ctr
(
% CPU signals%

VALIDIN/:OUTPUT;
WrRdy/:OUTPUT;
RdRdy/:OUTPUT;
ExtRqst/:OUTPUT;
VALIDOUT/:INPUT;
Release/:INPUT;

SysAD[28..20]:INPUT;
SysADn2:INPUT;


Syscmdin[8..0]:INPUT;
Syscmdout[8..0]:OUTPUT;

% Bus interface size %

P3_SYS_IF:INPUT;
PIPELINED/:INPUT;
R4K_WRITE:INPUT;
DRAM_SIZE[1..0]:INPUT;

%_____

P3_SYS_IFDescription

032-bit CPU interface
164-bit CPU interface
```

```
_____%



% FLASH CONTROL %

FLASH_DONE/:INPUT;


TCLK:INPUT; % global clock%
MRes/:INPUT; % global reset %

% Mem acknowledge signals %

SRAM_DONE/:INPUT; % SRAM ack signal %

MEM_DONE/:INPUT; % DRAM Ack signal %


RefReq/:INPUT;
RefAck/:OUTPUT;

IO_ADDR2:INPUT;


% IO acknowledge  %

IO_ACK2/:INPUT;

IOEX_OEA/:OUTPUT; % output enable for CPU data writing to I/O bus %
IOEX_SEL:OUTPUT; % Selects which word should be written to I/O bus %
IOEX_LEA1B:OUTPUT;  %Latch enable for latching data from CPU to I/O bus %
IOEX_LEA2B:OUTPUT;

% DMA interface %
DMA_REQ/ :INPUT;
DMA_ACK/:OUTPUT;

RD/:OUTPUT;  % Global Read Signal %
WR/:OUTPUT;  % Global Write signal %

RA2IO_OE:OUTPUT; % Buffer output enable control %
RA2IO_LE:OUTPUT;
RA2IO_CLK:OUTPUT;

IO2RA_OE/:OUTPUT;


% Error reporting from mem control %

SYS_ERROR[1..0]:INPUT; % Error code coming from DRAM_ctr about faulty cycles %
%_____

SYS_ERROR[1..0]Description

00Normal operation
01Write error
10Single read error
11Block read error

_____ %


% Global output enables %
IOEnable/:INPUT;
%Syscmd_out/:OUTPUT;  GLOBAL I/O enable, must be connected to IOEnable/
input %
```

```
IOEnable_out/:OUTPUT;  % io enable %

% Buffer control %

RA_CLKEN/:OUTPUT;
RA_OE/:OUTPUT;

SD2RD_OE/:OUTPUT;
SD2RD_CLKEN/:OUTPUT;

RD2SD_OE/:OUTPUT;
RD2SD_LE:OUTPUT;

IGNORE_ACCESS/:OUTPUT; % This signal should be asserted during Refresh cycles.We assert only WrRdy
and a valid address and control signals will be driven by CPU. Hence the DRAM_ctr will trigger as valid cycle.
Hence the idle state in in DRAM_ctr should use this signal as a qualifier to for a valid issue cycle %

WAIT/:OUTPUT;  % Wait signal for memory controlers to stop out of order completions %

IOEX_LEA1B_in:OUTPUT; % Actually IOEX_LE1B%


% Decoded signals %


%State Machine%

)

VARIABLE

% Main state machine %

dsm:MACHINE WITH STATES
  (s0 ,s1,s2, s3, s4, s5, s6, s7, s8, s9,s10, s11, s12,s13, s14, s15, s16, s17, s18, s19, s20,s21,s22, s23,s24,s25,
  s26, s27, s28, s29, s30, s31);

% DMA control state machine %

dmasm: MACHINE WITH STATES
( c0, c1, c2, c3);

delaysm: MACHINE WITH STATES
(d0, d1);


%_____OUTPUT
_____%

VALIDIN/:DFF;
WrRdy/:DFF;
D_WrRdy1:DFF;
D_WrRdy2:DFF;
RdRdy/:DFF;
ExtRqst/:DFF;
RD/:DFF;
WR/:DFF;
RA_CLKEN/:DFF;
RA_OE/:DFF;
DMA_ACK/:DFF;
SD2RD_OE/:DFF;
SD2RD_CLKEN/:DFF;
RD2SD_OE/:DFF;
RD2SD_LE:DFF;
WAIT/:DFF;
IOEX_OEA/:TRI;
IOEX_SEL:TRI;
IOEX_LEA1B:TRI;
```

```
IOEX_LEA2B:TRI;
IOEnable_out/:DFF;

%_____________________________________________________________________________ %

% burried internal flip flops for storing Syscmd bus%


IOEX_OEA/_in:DFF;
IOEX_SEL_in:DFF;
IOEX_LEA1B_in:DFF;
IOEX_LEA2B_in:DFF;

Syscmd1_st[7..3]:DFFE; %  To store Syscmd for second buffered cycle %
  % when it is input     %

Syscmd0_st[7..3]:DFFE; %  To store Syscmd for first  cycle %
% when it is input. This is always enabled %

SysAD_st[8..7]:DFFE;% To store the address of 2nd cycle %
SysADn0_st2:DFFE;  % To store the A2 address for the first buffered write %
SysADn1_st2:DFFE;  % To store the A2 address for the second buffered write %

SysAD_latch_en:NODE; % node that controls the syscmd1 latch enable %


%Syscmd_tri[8..0]:TRI;  To drive Syscmd when it is output%

%Syscmd_out/:DFF;  node that controls the tri-state buffer output%

Syscmd1_latch_en:DFF; % node that controls the syscmd1 latch enable %

Syscmd0_latch_en:NODE; % node that controls the syscmd0 latch enable %


Syscmd_id[8..0]:DFF; % Value to be back driven on Syscmd   %

Release_BUF:DFF; % registers whether Release/ has been issued for first(current) cycle or not  %

Release_BUF1:DFF; % registers whether Release/ has been issued for buffered cycle or not  %

Release_BU_clr/:DFF; % this would clear the Release_BU ff %
Release_BU1_clr/:DFF; % this would clear the Release_BU1 ff %


2nd_cyc:DFFE; % register to store what type of 2nd cycle has been
 buffered%


2nd_cyc_ena:DFF; % Enable signal for 2nd_cyc[2..0] ff   %


COUNT_en:DFF;
COUNT[1..0]:DFFE;% Counter to count two clock after WrRdy & RdRdy are
deasserted%

ST_COUNT_en:DFF;
ST_COUNT[2..0]:DFFE;% Counter to count 5 clocks when DMA_REQ is asserted and   dsm is in s0 for 5 clocks.
After 5 clocks, controller should  started DMA acknowledge sequence %


DMA:DFF; % To differentiate between normal and DMA cycles %
DMA_access:DFF;

DATUM_COUNT[1..0]:DFFE; % Counts 0 to 4 datums in 8 datums of 32-bit block read %
DATUM_COUNT_ena:DFF;
```

```
RefAck/:DFF;
IGNORE_ACCESS/:DFF; % see description in the Subdesign section %

REF_DMA/:DFF; % FF which is set to 1 when either refresh cycle or a DMA cycle is pending %
REF_DMA_clr:DFF;

RA2IO_OE:DFF;
RA2IO_LE:DFF;
RA2IO_CLK:DFF;
IO2RA_OE/:DFF;
IO_ACK2_delay/:DFF;
Syscmd0_st3_delay:DFF;



%_____________________________________________________ %


BEGIN
DEFAULTS
VALIDIN/=VCC;
WrRdy/=GND;
D_WrRdy1= GND;
D_WrRdy2= GND;
RdRdy/=GND;
ExtRqst/=VCC;

%Syscmd_out/= VCC;%
IOEnable_out/= GND;
Syscmd1_latch_en = GND;
Syscmd0_latch_en = GND;
SysAD_latch_en = VCC;
Syscmd_id[8..0] = B"000000000";
Release_BU_clr/= VCC;
Release_BU1_clr/= VCC;
2nd_cyc = GND;
2nd_cyc_ena = GND;
Release_BUF= GND;
Release_BUF1 = GND;

RD/= VCC;
WR/= VCC;

COUNT[1..0]= B"00";
ST_COUNT[2..0]= B"000";

COUNT_en= GND;
ST_COUNT_en= GND;

Syscmd_id[8..0]= GND;
Syscmd0_st[7..3]= VCC;

RA_CLKEN/= GND;
SD2RD_CLKEN/= VCC;
RA_OE/= GND;
SD2RD_OE/= GND;

DMA_ACK/= VCC;

DMA= GND;
DMA_access= GND;

RD2SD_OE/= VCC;
RD2SD_LE= VCC;

DATUM_COUNT[1..0] = B"00";
```

```
DATUM_COUNT_ena  = GND;

WAIT/= VCC;
RefAck/= VCC;
IGNORE_ACCESS/= VCC;
VALIDIN/= VCC;
REF_DMA/= VCC;
REF_DMA_clr= VCC;
RA2IO_OE= VCC;
RA2IO_LE= VCC;
RA2IO_CLK= GND;
IO2RA_OE/= VCC;
IOEX_OEA/_in= VCC;
IOEX_SEL_in= GND;
IOEX_LEA1B_in= VCC;
IOEX_LEA2B_in= VCC;
IO_ACK2_delay/= VCC;
Syscmd0_st3_delay = VCC;

END DEFAULTS;
%_____________________________  Syscmd  _____________________________________%


Syscmd1_st[7..3].clk = GLOBAL(TCLK);
Syscmd1_st[7..3].prn = GLOBAL(MRes/);
Syscmd1_st[7..3].ena = Syscmd1_latch_en;

Syscmd0_st[7..3].clk = GLOBAL(TCLK);
Syscmd0_st[7..3].prn = GLOBAL(MRes/);
Syscmd0_st[7..3].ena = Syscmd0_latch_en;

SysAD_st[8..7].clk = GLOBAL(TCLK);
SysAD_st[8..7].prn = GLOBAL(MRes/);
SysAD_st[8..7].ena =  Syscmd0_latch_en;
%SysAD_st[8..7].ena = SysAD_latch_en;%

SysADn0_st2.clk= GLOBAL(TCLK);
SysADn0_st2.prn= GLOBAL(MRes/);
SysADn0_st2.ena= Syscmd0_latch_en;

SysADn1_st2.clk= GLOBAL(TCLK);
SysADn1_st2.prn= GLOBAL(MRes/);
SysADn1_st2.ena= Syscmd1_latch_en;



%Syscmd_tri[8..0].oe = !GLOBAL(IOEnable/);
Syscmd_tri[8..0].in = Syscmd_id[8..0];%

Release_BUF.clk= !Release/;
%Release_BUF.prn= Release/;%

Release_BUF.clrn= GLOBAL(MRes/) & Release_BU_clr/;
Release_BUF.d= VCC;

Release_BUF1.clk=  !Release/;
%Release_BUF1.prn= Release/;%

Release_BUF1.clrn= GLOBAL(MRes/) & Release_BU1_clr/;
Release_BUF1.d= VCC;


2nd_cyc.ena = 2nd_cyc_ena;
2nd_cyc.clrn = GLOBAL(MRes/);
2nd_cyc.clk = GLOBAL(TCLK);

COUNT[1..0].ena= COUNT_en;
COUNT[1..0].clrn= GLOBAL(MRes/);
```

```
COUNT[1..0].clk= GLOBAL(TCLK);

ST_COUNT[2..0].ena= ST_COUNT_en;
ST_COUNT[2..0].clrn= GLOBAL(MRes/);
ST_COUNT[2..0].clk= GLOBAL(TCLK);

ST_COUNT_en.clk= GLOBAL(TCLK);
ST_COUNT_en.clrn= GLOBAL(MRes/);


Syscmd_id[8..0].clk= GLOBAL(TCLK);
Syscmd_id[8..0].clrn = GLOBAL(MRes/);


%_____                                                    Outputs
_____%

VALIDIN/.clk= GLOBAL(TCLK);
VALIDIN/.prn= GLOBAL(MRes/);

WrRdy/.clk= GLOBAL(TCLK);
WrRdy/.prn= GLOBAL(MRes/);

D_WrRdy1.clk= GLOBAL(TCLK);
D_WrRdy1.prn= GLOBAL(MRes/);

D_WrRdy2.clk= GLOBAL(TCLK);
D_WrRdy2.prn= GLOBAL(MRes/);

RdRdy/.clk= GLOBAL(TCLK);
RdRdy/.prn= GLOBAL(MRes/);

ExtRqst/.clk= GLOBAL(TCLK);
ExtRqst/.prn= GLOBAL(MRes/);

RD/.clk= GLOBAL(TCLK);
RD/.prn= GLOBAL(MRes/);

WR/.clk= GLOBAL(TCLK);
WR/.prn= GLOBAL(MRes/);

%
MEM_RD/.clk= GLOBAL(TCLK);
MEM_RD/.prn= GLOBAL(MRes/);

MEM_WR/.clk= GLOBAL(TCLK);
MEM_WR/.prn= GLOBAL(MRes/);%


RA_OE/.clk= GLOBAL(TCLK);
RA_OE/.clrn= GLOBAL(MRes/);

SD2RD_OE/.clk= GLOBAL(TCLK);
SD2RD_OE/.clrn= GLOBAL(MRes/);

RA_CLKEN/.clk= GLOBAL(TCLK);
RA_CLKEN/.clrn= GLOBAL(MRes/);

SD2RD_CLKEN/.clk= GLOBAL(TCLK);
SD2RD_CLKEN/.prn= GLOBAL(MRes/);

DMA_ACK/.clk= GLOBAL(TCLK);
DMA_ACK/.prn= GLOBAL(MRes/);

%DMA.ena= DMA_en;%
DMA.clrn= GLOBAL(MRes/);
DMA.clk= GLOBAL(TCLK);
```

```
DMA_access.clk= GLOBAL(TCLK);
DMA_access.clrn= GLOBAL(MRes/);

RD2SD_OE/.clk= GLOBAL(TCLK);
RD2SD_OE/.prn= GLOBAL(MRes/);

RD2SD_LE.clk= GLOBAL(TCLK);
RD2SD_LE.clrn= GLOBAL(MRes/);

DATUM_COUNT[1..0].clk= GLOBAL(TCLK);
DATUM_COUNT[1..0].clrn= GLOBAL(MRes/);
DATUM_COUNT[1..0].ena= DATUM_COUNT_ena;

DATUM_COUNT_ena.clk= GLOBAL(TCLK);
DATUM_COUNT_ena.clrn= GLOBAL(MRes/);

WAIT/.clk= GLOBAL(TCLK);
WAIT/.prn= GLOBAL(MRes/);

IGNORE_ACCESS/.clk= GLOBAL(TCLK);
IGNORE_ACCESS/.prn= GLOBAL(MRes/);

%_____________________________________________________________________________ %

%Syscmd_out/.clk= GLOBAL(TCLK);
Syscmd_out/.prn= GLOBAL(MRes/);%

IOEnable_out/.clk= GLOBAL(TCLK);
IOEnable_out/.clrn= GLOBAL(MRes/);

Syscmd1_latch_en.clk = GLOBAL(TCLK);

%Syscmd0_latch_en.clk = GLOBAL(TCLK);%

Release_BU_clr/.clk = GLOBAL(TCLK);
Release_BU_clr/.prn = GLOBAL(MRes/);

Release_BU1_clr/.clk = GLOBAL(TCLK);
Release_BU1_clr/.prn = GLOBAL(MRes/);


RefAck/.clk= GLOBAL(TCLK);
RefAck/.prn= GLOBAL(MRes/);

REF_DMA/.clk= GLOBAL(TCLK);
REF_DMA/.prn= GLOBAL(MRes/);
REF_DMA/.clrn= REF_DMA_clr;

REF_DMA_clr.clk= GLOBAL(TCLK);
REF_DMA_clr.prn= GLOBAL(MRes/);

2nd_cyc_ena.clk = GLOBAL(TCLK);
2nd_cyc_ena.clrn = GLOBAL(MRes/);


COUNT_en.clk = GLOBAL(TCLK);

RA2IO_OE.clk = GLOBAL(TCLK);
%RA2IO_OE.prn = GLOBAL(MRes/);%

RA2IO_LE.clk = GLOBAL(TCLK);
RA2IO_LE.prn = GLOBAL(MRes/);

RA2IO_CLK.clk = GLOBAL(TCLK);
RA2IO_CLK.clrn = GLOBAL(MRes/);

IO2RA_OE/.clk= GLOBAL(TCLK);
IO2RA_OE/.prn = GLOBAL(MRes/);
```

```
IOEX_OEA/_in.clk = GLOBAL(TCLK);
IOEX_OEA/_in.prn = GLOBAL(MRes/);

IOEX_SEL_in.clk  = GLOBAL(TCLK);
IOEX_SEL_in.clrn = GLOBAL(MRes/);

IOEX_LEA1B_in.clk = GLOBAL(TCLK);
IOEX_LEA1B_in.clrn = GLOBAL(MRes/);

IOEX_LEA2B_in.clk= GLOBAL(TCLK);
IOEX_LEA2B_in.clrn = GLOBAL(MRes/);

IOEX_OEA/.oe=!GLOBAL(IOEnable/);
IOEX_OEA/.in= IOEX_OEA/_in;

IOEX_SEL.oe= VCC;
IOEX_SEL.in= IOEX_SEL_in;

IOEX_LEA1B.oe=GND;
IOEX_LEA1B.in= VCC;

IOEX_LEA2B.oe= VCC;
IOEX_LEA2B.in= IOEX_LEA2B_in;

IO_ACK2_delay/.clk= GLOBAL(TCLK);
IO_ACK2_delay/.prn = GLOBAL(MRes/);

Syscmd0_st3_delay.clk=  GLOBAL(TCLK);
Syscmd0_st3_delay.prn= GLOBAL(MRes/);
Syscmd0_st3_delay.clrn= Syscmd0_st3;
Syscmd0_st3_delay.d= VCC;

%  Outputs  %?

% Connecting the input at Syscmd[8..0] to internal latch%

Syscmd1_st[7..3] = Syscmdin[7..3];
Syscmd0_st[7..3] = Syscmdin[7..3];
SysAD_st[8..7] = SysAD[28..27];
SysADn0_st2= SysADn2;
SysADn1_st2= SysADn2;

D_WrRdy1.d= WrRdy/;
D_WrRdy2.d= D_WrRdy1.q;


% When Syscmd is an output %

Syscmdout[8..0] = Syscmd_id[8..0];



%MAIN STATE MACHINE AND DECODER%

dsm.clk = GLOBAL(TCLK);
dsm.reset = GLOBAL(!MRes/);

dmasm.clk = GLOBAL(TCLK);
dmasm.reset = GLOBAL(!MRes/);

delaysm.clk = GLOBAL(TCLK);
delaysm.reset = GLOBAL(!MRes/);


% Equations %

RA2IO_OE = !DMA_access;
```

```
IO2RA_OE/ = !DMA_access;

Syscmd0_latch_en =  (dsm==s0); % Latch Syscmd of current cycle %
%SysAD_latch_en = (dsm==s0);  Latch SysAD  of current cycle %

IOEX_SEL_in= IO_ADDR2;



WrRdy/ = !DMA_REQ/ # !RefReq/;
RdRdy/ = !DMA_REQ/ # !RefReq/;

%  Address Decoder  %


%_____ Actual State Machine _____%



CASE dsm IS

WHEN s0 =>% MAIN DECODE IN IDLE STATE %
%_____ IO Space _____ %
IF (!DMA &  !Release/ & !VALIDOUT/ & (Syscmdin[8..5] == B"0000") &
(SysAD28==B"1")) THEN% IO read %
    % with Release/ %
WrRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RdRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
Release_BU_clr/ = !(2nd_cyc == idle);
Release_BU1_clr/ = !(2nd_cyc == idle);
RD/ = GND;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
dsm = s1;



ELSIF(!DMA & Release/& !D_WrRdy2 & !VALIDOUT/ & (SysAD28==B"1") &   % IO write%
    (Syscmdin[8..5] == B"0010")) THEN

WrRdy/ = VCC;
RdRdy/ = VCC;
WR/ = GND;
RA_CLKEN/ = VCC;
SD2RD_CLKEN/ = GND;
WAIT/ = GND;
IOEX_OEA/_in = GND;
dsm = s6;


ELSIF (!DMA & !Release/ & !VALIDOUT/ &    %memory read%
      (SysAD28==B"0") & (Syscmdin[8..5] == B"0000")) THEN  %with Release/%

WrRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RdRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
Release_BU_clr/ = !(2nd_cyc == idle);
Release_BU1_clr/ = !(2nd_cyc == idle);
RD/ = GND;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
dsm = s1;



ELSIF(!DMA & Release/ & !D_WrRdy2 & !VALIDOUT/ & (SysAD28==B"0") &    %Memory  write%
 (Syscmdin[8..5] == B"0010")) THEN

    WrRdy/ = VCC;
```

```
RdRdy/ = VCC;
WR/ = GND;
    RA_CLKEN/ = VCC;
    SD2RD_CLKEN/ = GND;
    dsm = s6;


ELSIF(!REF_DMA/) THEN

dsm = s28;

ELSE

    dsm = s0;



    END IF;



WHEN s1 =>% BUS TURN ARROUND %

Syscmd0_latch_en = GND;

% If Release/ comes in issue cycle %

IF (Syscmd0_st3.q & !(FLASH_DONE/ & IO_ACK2_delay/ & SRAM_DONE/ & MEM_DONE/)) THEN


WrRdy/ = !(2nd_cyc == idle) # !DMA_REQ/; % For single read %
RdRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
VALIDIN/ = GND;
 Syscmd_id[8..5] = last_read;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
2nd_cyc= idle;
2nd_cyc_ena = VCC;
RD/ = RD/;
RA_CLKEN/ = VCC;

RD2SD_OE/ = GND;
SD2RD_OE/ = VCC;
dsm = s2;%FLASH_DONE/ or IO_ACK2 or SRAM_DONE
 signal should be active %

ELSIF (!Syscmd0_st3.q & !(FLASH_DONE/ & IO_ACK2_delay/ & SRAM_DONE/ & MEM_DONE/)) THEN


WrRdy/ = !(2nd_cyc == idle) # !DMA_REQ/; % For block reads %
RdRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RD/ = RD/;
VALIDIN/ = GND;
Syscmd_id[8..5] = no_last_read;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;

dsm = s17;



ELSIF !(SYS_ERROR[1..0]==B"00") THEN  % If instead of acknowledge, we get error
code from mem control %
```

```
%Syscmd_out/ = GND;%
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
RD/ = GND;
dsm = s3;% Return erronous data  %


ELSE

WrRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RdRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
Release_BU_clr/ = !(2nd_cyc == idle);
RD/ = RD/;
Syscmd_id[8..5] = last_read;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
%Syscmd_out/ = GND;%
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;

dsm = s1;% Wait for acknowledge signal from slave controllers     %
END IF;


WHEN s2 =>

SD2RD_OE/ = VCC;

IF(COUNT[1..0] == B"11") THEN

COUNT_en = VCC;
COUNT[1..0] = B"10";
RA_CLKEN/ = VCC;
dsm = s2;


ELSIF (COUNT[1..0] == B"10") THEN

COUNT_en = VCC;
COUNT[1..0] = B"00";
2nd_cyc= idle;
2nd_cyc_ena = VCC;
RA_CLKEN/ = VCC;
dsm = s2;

ELSIF !( DMA_REQ/ # (COUNT1 == B"1")) THEN

WrRdy/ = VCC;
RdRdy/ = VCC;
RA_CLKEN/ = GND;
dsm = s0;

ELSE

Release_BU_clr/ = GND;
Release_BU1_clr/ = GND;
RA_CLKEN/ = GND;
dsm = s0;

END IF;




WHEN s3 =>% ERROR CONDITIONS %
```

```
2nd_cyc= idle;
2nd_cyc_ena = VCC;

CASE (SYS_ERROR[1..0],Syscmd0_st3) IS% This is data phase for writes
or bus turn around for reads %

WHEN B"101" =>% Single read %
VALIDIN/ = GND;
Syscmd_id[8..5] = last_err_data;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;
RD2SD_OE/ = GND;
SD2RD_OE/ = VCC;
%Syscmd_out/ = GND;%
RD/ = GND;
dsm = s2;

WHEN B"100" =>% Block read %
VALIDIN/ = GND;
Syscmd_id[8..5] = no_last_err_data;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;

RA_CLKEN/ = VCC;
RD2SD_OE/ = GND;
SD2RD_OE/ = VCC;
RD/ = GND;
%Syscmd_out/ = GND;%
dsm = s17;

WHEN B"01X" =>% write %

dsm = s8;


WHEN OTHERS =>
RA_CLKEN/ = VCC;
dsm = s9;

END CASE;


WHEN s4 =>% waiting for Release/ for single read %

WrRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RdRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
% If Release/ does not come in issue cylce %
IF  ((SYS_ERROR[1..0]==B"00") & !Release/ ) THEN  %Release_BUF does internally does not meet the setup
time internally when Release/ comes in the same cycle when this buffer is sampled.Thus we are using Release/
for this cycle%

RD/ = GND;
Release_BU_clr/ = GND;
Release_BU1_clr/ = GND;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
dsm = s1; %When Release comes in, start the cycle %


%ELSIF !(SYS_ERROR[1..0]==B"00") THEN
dsm = s3; Return erronous data  %

ELSIF (!REF_DMA/) THEN
dsm = s28; % Refresh and CPU issue can come in same cycle. The CPU starts decoding the cycle and then checks
```

```
for refresh. By this time RdRdy/and WrRdy/ have been deasserted.%


ELSE

SD2RD_OE/ = VCC;
RA_CLKEN/ = VCC;
RD/ = GND;
dsm = s4;

END IF;

WHEN s5 =>

IF (!Syscmd0_st3.q & !(FLASH_DONE/ & IO_ACK2_delay/ & SRAM_DONE/ & MEM_DONE/)) THEN


WrRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RdRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RD/ = GND;
VALIDIN/ = GND;
Syscmd_id[8..5] = no_last_read;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
dsm = s17;


ELSIF !(SYS_ERROR[1..0]==B"00") THEN
dsm = s3;% Return erronous data  %


ELSE

WrRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RdRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
Release_BU_clr/ = !(2nd_cyc == idle);
RD/ = GND;
Syscmd_id[8..5] = last_read;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
%Syscmd_out/ = GND;%
dsm = s5;% asserted     %
END IF;



WHEN s6 =>

RA2IO_LE = GND;

IF !(!(2nd_cyc == idle) # (FLASH_DONE/ & IO_ACK2_delay/ & SRAM_DONE/ & MEM_DONE/)) THEN  %
Zero wait state cycle %

RA_CLKEN/ = GND;% Write cycle %
WrRdy/= !DMA_REQ/;
RdRdy/= !DMA_REQ/;
IOEX_OEA/_in = !SysAD_st8;
IOEX_SEL_in= IO_ADDR2;
dsm = s0;

ELSIF !((2nd_cyc == idle) # (FLASH_DONE/ & IO_ACK2_delay/ & SRAM_DONE/ & MEM_DONE/)) THEN

COUNT_en = VCC;
COUNT[1..0] = B"10";
WrRdy/ =  !DMA_REQ/;
RdRdy/ =  !DMA_REQ/;
```

```
RA_CLKEN/ = VCC;
2nd_cyc = GND;
2nd_cyc_ena = VCC;
IOEX_OEA/_in = !SysAD_st8;
IOEX_SEL_in= IO_ADDR2;
dsm = s9;

%ELSIF !((2nd_cyc == idle) # !(FLASH_DONE/ & IO_ACK2_delay/ & SRAM_DONE/ & MEM_DONE/))
THEN


WrRdy/ = VCC;
RdRdy/ = VCC;
RA_CLKEN/ = VCC;
dsm = s10; %


ELSIF !(SYS_ERROR[1..0]==B"00") THEN
dsm = s3;% Return erronous data  %


ELSE
WrRdy/ = VCC;
RdRdy/ = VCC;
WR/= GND;
2nd_cyc_ena = VCC;
Syscmd1_latch_en = VCC;
SysAD_latch_en = VCC;
RA_CLKEN/ = VCC;
%WAIT/ = WAIT/;%
RA2IO_LE = RA2IO_LE;
RA2IO_CLK = VCC;
IOEX_OEA/_in = !SysAD_st8;
IOEX_SEL_in= IO_ADDR2;
    SD2RD_CLKEN/ = GND;
dsm = s7;
END IF;

WHEN s7 =>
RA2IO_LE = GND;
IOEX_OEA/_in = !SysAD_st8;

IF(!VALIDOUT/ & !Release/ & !Syscmdin8) THEN

2nd_cyc = VCC;% This can be second issue cycle  %
% & first cycle termination %

ELSIF(!VALIDOUT/ & !Syscmdin8) THEN

2nd_cyc = VCC;

ELSE

2nd_cyc = GND;

END IF;

IF !(!(!PIPELINED/ & R4K_WRITE) # (FLASH_DONE/ & IO_ACK2_delay/ & SRAM_DONE/ &
MEM_DONE/)) THEN
WrRdy/ = VCC;% Waiting for write completion %
RdRdy/ = VCC;
COUNT_en = VCC;
SD2RD_CLKEN/ = !(2nd_cyc == B"0");
RA_CLKEN/ = VCC;
Syscmd0_latch_en = 2nd_cyc;
WAIT/ = WAIT/;
RA2IO_LE = RA2IO_LE;
dsm = s8;% decode next cycle %
```

```
ELSIF!((!PIPELINED/ & R4K_WRITE) # (FLASH_DONE/ & IO_ACK2_delay/ & SRAM_DONE/ & MEM_DONE/)) THEN
WrRdy/ = VCC;
RdRdy/ = VCC;
COUNT_en = VCC;
SD2RD_CLKEN/ = !(2nd_cyc == B"0");
RA_CLKEN/ = VCC;
Syscmd0_latch_en = 2nd_cyc;
WAIT/ = WAIT/;
RA2IO_LE = RA2IO_LE;
IOEX_LEA1B_in = GND;
IOEX_LEA2B_in = GND;

dsm = s9;

ELSIF !(SYS_ERROR[1..0]==B"00") THEN
dsm = s3;% Return erronous data  %


ELSE
WrRdy/ = VCC;
RdRdy/ = VCC;
WR/= GND;
SD2RD_CLKEN/ = !(2nd_cyc == B"0");
RA_CLKEN/ = VCC;
WAIT/ = WAIT/;
RA2IO_LE = RA2IO_LE;
RA2IO_CLK = VCC;
IOEX_LEA1B_in = GND;
IOEX_LEA2B_in = GND;


dsm = s7;

END IF;

WHEN s8 =>

Syscmd0_latch_en = 2nd_cyc;

CASE(2nd_cyc, SysAD_st8, Syscmd1_st6) IS

WHEN B"110" =>% IO  read %

WrRdy/ = VCC;
RdRdy/ = VCC;
Syscmd0_latch_en = VCC;
Syscmd0_st[7..3] = Syscmd1_st[7..3];
SysADn0_st2.d= SysADn1_st2.q;
COUNT_en = GND;
COUNT[1..0] = B"11";
RD/ = GND;
RA_CLKEN/ = VCC;
WAIT/ = VCC;
dsm = s4;


WHEN B"111" =>% IO single write %
WrRdy/ = VCC;
RdRdy/ = VCC;
Syscmd0_latch_en = VCC;
Syscmd0_st[7..3].d = Syscmd1_st[7..3].q;
SysADn0_st2.d= SysADn1_st2.q;
COUNT_en = GND;
COUNT[1..0] = B"11";
WR/ = GND;
RA_CLKEN/ = VCC;
```

```
WAIT/ = VCC;
RA2IO_LE = RA2IO_LE;
RA2IO_CLK = VCC;
IOEX_OEA/_in = GND;
IOEX_SEL_in=IO_ADDR2;
dsm = s6;



WHEN B”100” =>% memory read %

WrRdy/ = VCC;
RdRdy/ = VCC;
Syscmd1_latch_en = VCC;
Syscmd0_st[7..3].d = Syscmd1_st[7..3].q;
SysADn0_st2.d= SysADn1_st2.q;
COUNT_en = GND;
COUNT[1..0] = B”10”;
RD/ = GND;
RA_CLKEN/ = VCC;
WAIT/ = VCC;
dsm = s4;


WHEN B”101” =>% memory write %

WrRdy/ = VCC;
RdRdy/ = VCC;
Syscmd0_latch_en = VCC;
Syscmd0_st[7..3].d = Syscmd1_st[7..3].q;
SysADn0_st2.d= SysADn1_st2.q;
COUNT_en = GND;
COUNT[1..0] = B”10”;
WR/ = GND;
RA_CLKEN/ = VCC;
WAIT/ = VCC;
dsm = s6;

WHEN B”0XX” =>

WrRdy/ =!DMA_REQ/;
RdRdy/ = !DMA_REQ/;
COUNT_en = VCC;
COUNT[1..0] = B”10”;
RA_CLKEN/ = VCC;
WAIT/ = VCC;
IOEX_OEA/_in = GND;
dsm = s9;


WHEN OTHERS =>
2nd_cyc_ena = GND;
2nd_cyc = GND;
RA_CLKEN/ = VCC;
WAIT/ = VCC;
dsm = s3;

END CASE;

WHEN s9 =>
IF(COUNT[1..0] == B”11”) THEN

WrRdy/ =!DMA_REQ/;
RdRdy/ = !DMA_REQ/;
COUNT[1..0] = B”10”;
COUNT_en= VCC;
dsm = s9;
```

```
ELSIF (COUNT[1..0] == B"10") THEN

WrRdy/ =!DMA_REQ/;
RdRdy/ = !DMA_REQ/;
COUNT[1..0] = B"00";
COUNT_en= VCC;
RA_CLKEN/ = VCC;
dsm = s9;

ELSIF !( DMA_REQ/ # (COUNT1 == B"1")) THEN

WrRdy/ = VCC;
RdRdy/ = VCC;
RA_CLKEN/ = GND;
dsm = s0;

ELSE
dsm = s0;

END IF;


WHEN s10 =>

IF !(FLASH_DONE/ & IO_ACK2_delay/ & SRAM_DONE/ & MEM_DONE/) THEN
% Waiting for write completion %

2nd_cyc.d= idle;
2nd_cyc_ena = VCC;

dsm = s9;

ELSIF !(SYS_ERROR[1..0]==B"00") THEN
dsm = s3;% Return erronous data  %


ELSE
WrRdy/ = VCC;
RdRdy/ = VCC;
dsm = s10;

END IF;

WHENs11 =>

WrRdy/ = !(2nd_cyc == B"0") # !DMA_REQ/;
RdRdy/ = !(2nd_cyc == B"0") # !DMA_REQ/;
% If Release/ does not come in issue cylce %
IF Release_BUF THEN
Release_BU_clr/ = GND;
dsm  = s5;
ELSE
dsm = s11;
END IF;



WHEN s17 =>% Block read from FLASH %

WrRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RdRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
Release_BU_clr/ = GND;
RD2SD_OE/ = GND;


IF !(FLASH_DONE/ & IO_ACK2_delay/ & MEM_DONE/ & SRAM_DONE/) THEN
VALIDIN/ = GND;
%Syscmd_out/ = GND;%
```

```
Syscmd_id[8..5] = no_last_read;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
RD/ = GND;
dsm = s19;
ELSIF !(SYS_ERROR[1..0]==B"00") THEN

VALIDIN/ = GND;
%Syscmd_out/ = GND;%
Syscmd_id[8..5] = no_last_err_data;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
RD/ = GND;
dsm = s19;% Return erronous data  %


ELSE
%Syscmd_out/ = GND;%
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
RD/ = GND;
dsm = s17;
END IF;


WHEN s18 =>
WrRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RdRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RD2SD_OE/ = GND;
RD/ = GND;


IF !(FLASH_DONE/ & IO_ACK2_delay/ & MEM_DONE/ & SRAM_DONE/) THEN
VALIDIN/ = GND;
%Syscmd_out/ = GND;%
Syscmd_id[8..5] = last_read;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
dsm = s20;


ELSIF !(SYS_ERROR[1..0]==B"00") THEN
VALIDIN/ = GND;
%Syscmd_out/ = GND;%
Syscmd_id[8..5] = last_err_data;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
dsm = s20;% Return erronous data  %

ELSE
%Syscmd_out/ = GND;%
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
dsm = s18;
```

```
END IF;


WHEN s19 =>
% s8 %
WrRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RdRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RD/ = GND;

IF !(!P3_SYS_IF # (FLASH_DONE/ & IO_ACK2_delay/ & MEM_DONE/ & SRAM_DONE/)) THEN
VALIDIN/ = GND;
%Syscmd_out/ = GND;%
Syscmd_id[8..5] = no_last_read;% 64 bit interface %
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
dsm = s18;

ELSIF !(P3_SYS_IF # (FLASH_DONE/ & IO_ACK2_delay/ & MEM_DONE/ & SRAM_DONE/)) THEN
VALIDIN/ = GND;
%Syscmd_out/ = GND;%
Syscmd_id[8..5] = no_last_read;% 32-bit interface %
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
DATUM_COUNT_ena = VCC;
DATUM_COUNT[1..0] = B"01";
DATUM_COUNT_ena = VCC;
dsm = s27;



ELSIF !((P3_SYS_IF # SYS_ERROR[1..0]==B"00")) THEN
VALIDIN/ = GND;
%Syscmd_out/ = GND;%
Syscmd_id[8..5] = no_last_err_data;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
DATUM_COUNT_ena = VCC;
dsm = s27;% Return erronous data  %

ELSIF !(!P3_SYS_IF # (SYS_ERROR[1..0]==B"00")) THEN
VALIDIN/ = GND;
%Syscmd_out/ = GND;%
Syscmd_id[8..5] = no_last_err_data;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
dsm = s18; %Return erronous data  %


ELSE
%Syscmd_out/ = GND;%
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
dsm = s19;
END IF;
```

```
WHEN s20 =>

IF(COUNT[1..0] == B”11”) THEN

COUNT[1..0] = B”10”;
COUNT_en= GND;
RA_CLKEN/ = VCC;
dsm = s20;

ELSIF (COUNT[1..0] == B”10”) THEN

COUNT[1..0] = B”00”;
COUNT_en= GND;
RA_CLKEN/ = VCC;
dsm = s20;

ELSIF !( DMA_REQ/ # (COUNT1 == B”1”)) THEN

WrRdy/ = VCC;
RdRdy/ = VCC;
RA_CLKEN/ = GND;
dsm = s0;


ELSE
RA_CLKEN/ = GND;
dsm = s0;
END IF;

WHEN s21 =>
COUNT_en = VCC;
dsm = s9;


WHEN s22 =>
VALIDIN/ = GND;
Syscmd_id[8..5] = no_last_err_data;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B”00”);
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
%Syscmd_out/ = GND;%
RD2SD_OE/ = GND;
dsm = s23;

WHEN s23 =>
VALIDIN/ = GND;
Syscmd_id[8..5] = no_last_err_data;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B”00”);
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
%Syscmd_out/ = GND;%
RD2SD_OE/ = GND;
dsm = s24;

WHEN s24 =>
VALIDIN/ = GND;
Syscmd_id[8..5] =last_err_data;
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B”00”);
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
%Syscmd_out/ = GND;%
RD2SD_OE/ = GND;
```

```
dsm = s20;

WHEN s25 =>
RA_CLKEN/ = VCC;
dsm = s0;

WHEN s27 =>

WrRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RdRdy/ = !(2nd_cyc == idle) # !DMA_REQ/;
RD2SD_OE/ = GND;
RD/ = GND;


IF !(!(DATUM_COUNT[1..0] == B"00") # (SYS_ERROR[1..0]==B"00") & (FLASH_DONE/ &
IO_ACK2_delay/ & MEM_DONE/ & SRAM_DONE/)) THEN
VALIDIN/ = GND;
%Syscmd_out/ = GND;%
Syscmd_id[8..5] = no_last_read;% 64-bit interface %
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;% or 32-bit interface 2nd loop %
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
DATUM_COUNT_ena = VCC;
DATUM_COUNT[1..0] = B"01";
dsm = s27;

ELSIF !(!(DATUM_COUNT[1..0] == B"01") # (SYS_ERROR[1..0]==B"00") & (FLASH_DONE/ &
IO_ACK2_delay/ & MEM_DONE/ & SRAM_DONE/)) THEN
VALIDIN/ = GND;
%Syscmd_out/ = GND;%
Syscmd_id[8..5] = no_last_read;% 64-bit interface %
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;% or 32-bit interface 2nd loop %
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
DATUM_COUNT_ena = VCC;
DATUM_COUNT[1..0] = B"10";
dsm = s27;


ELSIF !(!(DATUM_COUNT[1..0] == B"10") # (SYS_ERROR[1..0]==B"00") & (FLASH_DONE/ &
IO_ACK2_delay/ & MEM_DONE/ & SRAM_DONE/)) THEN
VALIDIN/ = GND;
%Syscmd_out/ = GND;%
Syscmd_id[8..5] = no_last_read;% 64-bit interface %
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
RA_CLKEN/ = VCC;% or 32-bit interface 2nd loop %
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
DATUM_COUNT_ena = VCC;
DATUM_COUNT[1..0] = B"11";
dsm = s27;



ELSIF !(!(DATUM_COUNT[1..0] == B"11") # (SYS_ERROR[1..0]==B"00") & (FLASH_DONE/ &
IO_ACK2_delay/ & MEM_DONE/ & SRAM_DONE/)) THEN
VALIDIN/ = GND;
%Syscmd_out/ = GND;%
Syscmd_id[8..5] = no_last_read;% 64-bit interface %
Syscmd_id4 = MEM_DONE/ # (DRAM_SIZE[1..0] == B"00");
Syscmd_id[3..0] = Res_bits;
```

```
RA_CLKEN/ = VCC;% or 32-bit interface 2nd loop %
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
DATUM_COUNT_ena = VCC;
DATUM_COUNT[1..0] = B"00";
dsm = s18;



ELSE
%Syscmd_out/ = GND;%
RA_CLKEN/ = VCC;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
DATUM_COUNT_ena = VCC;
DATUM_COUNT[1..0] = DATUM_COUNT[1..0];
dsm = s27;

END IF;

WHEN s28 =>

IF (!Release/ & !DMA_REQ/) THEN  % Is it because of DMA %

 % DMA has higher priority. However, once  refresh has started, it would complete before DMA will be serviced
%
ExtRqst/ = GND;
%DMA_en = VCC;%
dsm = s29;
ELSIF(!RefReq/ & ExtRqst/) THEN   % or because of Refresh %
dsm = s16;

ELSIF(RefReq/ & DMA_REQ/) THEN

dsm = s0;

ELSE
ExtRqst/ = GND;
dsm = s28;

END IF;

WHEN s29 =>

 SD2RD_OE/ = VCC;
RA_OE/ = VCC;
DMA_access = VCC;
ExtRqst/ = GND;
IOEnable_out/ = VCC;
dsm = s30;

WHEN s30 =>
 SD2RD_OE/ = VCC;
IOEnable_out/ = VCC;
RA_OE/ = VCC;
DMA_ACK/ = GND ;

IF !(DMA_REQ/) THEN  % Polling DMA_REQ/. If DMA_REQ/ disappears, start DMA termination
 sequence and finally release bus back to CPU %
DMA_access  = VCC;
dsm = s30;

ELSE
DMA_access = VCC;
dsm = s31;

END IF;
```

```
WHEN s31 =>
WrRdy/ = VCC; % This is necessory because wrrdy/ will disappear as  soon as DMA_REQ is gone but system is not ready to take new cycles from the CPU for next 3 clocks. %
RdRdy/ = VCC;
IOEnable_out/ = VCC;
SD2RD_OE/ = VCC;
RA_OE/ = VCC;
%DMA_ACK/ = GND ;%
DMA_access = GND;
%Syscmd_out/ = GND;%
dsm = s13;

WHEN s13 => % Releasing the bus back to CPU %
WrRdy/ = VCC;
RdRdy/ = VCC;
VALIDIN/ = GND;
%Syscmd_out/ = GND;%
Syscmd_id[8..0] = null_req;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
IOEnable_out/ = VCC;
dsm = s14;

WHEN s14 =>
WrRdy/ = VCC;
RdRdy/ = VCC;
IOEnable_out/ = VCC;
dsm = s28;



WHEN s15 =>
dsm = s16;
IOEnable_out/ = VCC;
IGNORE_ACCESS/ = GND;

WHEN s16 =>
dsm = s26;
RefAck/ = GND;
IGNORE_ACCESS/ = GND;

WHEN s26 =>
IF(!RefReq/) THEN
RefAck/ = GND;
IGNORE_ACCESS/ = GND;
dsm = s26;

ELSE
RefAck/ = GND;
IGNORE_ACCESS/ = GND;
dsm = s9;
END IF;

WHEN s12 =>
dsm = s0;


%WHEN s32 =>
 WrRdy/ = VCC;
RdRdy/ = VCC;
WR/= GND;
2nd_cyc_ena = VCC;
Syscmd1_latch_en = VCC;
SysAD_latch_en = VCC;
RA_CLKEN/ = !(2nd_cyc == B”0”);
%WAIT/ = WAIT/;%
RA2IO_LE = RA2IO_LE;
```

```
RA2IO_CLK = VCC;
IOEX_OEA/_in = GND;
IOEX_SEL_in= IO_ADDR2;
dsm = s7;%


WHEN OTHERS =>

RA_CLKEN/ = VCC;
dsm = s0;


END CASE;

%_____ DMA Control
_____%


% If DMA_REQ is asserted and dsm state machine stays in s0 continously for 5 clocks, DMA sequence
  is initiated  %

 % Also, if RefReq/ is asserted, this state machine will deassert WrRdy/ and will respond with  RefAck/ to the
DRAM controller. %

CASE dmasm IS

WHEN c0 =>
dmasm = c1;


WHEN c1 =>

IF ((!RefReq/ # !DMA_REQ/) & (ST_COUNT[2..0] == B"000") & (dsm == s0)) THEN

ST_COUNT[2..0] = B"001";
ST_COUNT_en= VCC;
dmasm = c1;

ELSIF ((!RefReq/ # !DMA_REQ/) & (ST_COUNT[2..0] == B"001") & (dsm == s0))THEN

ST_COUNT[2..0] = B"010";
ST_COUNT_en= VCC;
dmasm = c1;


ELSIF ((!RefReq/ # !DMA_REQ/) & (ST_COUNT[2..0] == B"010") & (dsm == s0))THEN

ST_COUNT[2..0] = B"011";
ST_COUNT_en= VCC;
dmasm = c1;


ELSIF ((!RefReq/ # !DMA_REQ/) & (ST_COUNT[2..0] == B"011") & (dsm == s0))THEN

ST_COUNT[2..0] = B"100";
ST_COUNT_en= VCC;
dmasm = c1;


ELSIF ((!RefReq/ # !DMA_REQ/) & (ST_COUNT[2..0] == B"100") & (dsm == s0))THEN

ST_COUNT[2..0] = B"101";
ST_COUNT_en= VCC;
dmasm = c1;


ELSIF ((!RefReq/ # !DMA_REQ/) & (ST_COUNT[2..0] == B"101") & (dsm == s0))THEN
```

```
ST_COUNT[2..0] = B"000";
ST_COUNT_en= VCC;
 REF_DMA/ = GND;
 DMA = VCC;
 dmasm = c2;

%ELSIF (!RefReq/) THEN
   dmasm = c8;%


ELSE

ST_COUNT[2..0] = B"000";
ST_COUNT_en= VCC;
dmasm = c1;

END IF;

WHEN c2 =>
REF_DMA/ = GND;
DMA = VCC;
%RefAck/ = GND;%
dmasm = c3;

WHEN c3 =>
IF(!DMA_REQ/ # !RefReq/) THEN
%RefAck/ = RefAck/;%
REF_DMA/ = GND;
DMA = VCC;
dmasm = c3;
ELSE
dmasm = c0;
END IF;



%WHEN c2 =>

IF (!Release/ & !DMA_REQ/) THEN

DMA = VCC;
DMA_en = VCC;
ExtRqst/ = GND;
DMA_en = VCC;
dmasm = c3;

ELSE
ExtRqst/ = GND;
dmasm = c2;

END IF;

WHEN c3 =>
SD2RD_OE/ = VCC;
RA_OE/ = VCC;
DMA.d = VCC;
ExtRqst/ = GND;
dmasm = c4;

WHEN c4 =>

SD2RD_OE/ = VCC;
RA_OE/ = VCC;
DMA_ACK/ = GND ;
IF !(DMA_REQ/) THEN   Polling DMA_REQ/. If DMA_REQ/ disappears, start DMA termination  sequence and
finally release bus back to CPU %
%dmasm = c4;
```

```
ELSE

DMA_en = VCC;
dmasm = c5;

END IF;

WHEN c5 =>

SD2RD_OE/ = VCC;
RA_OE/ = VCC;
DMA_ACK/ = GND ;
DMA.d = GND;
Syscmd_out/ = GND;
dmasm = c6;

WHEN c6 =>  Releasing the bus back to CPU %
%VALIDIN/ = GND;
Syscmd_out/ = GND;
Syscmd_id[8..0] = null_req;
SD2RD_OE/ = VCC;
RD2SD_OE/ = GND;
dmasm = c7;

WHEN c7 =>
dmasm = c8;



WHEN c8 =>
dmasm = c10;
IGNORE_ACCESS/ = GND;

WHEN c10 =>
dmasm = c11;
RefAck/ = GND;
IGNORE_ACCESS/ = GND;

WHEN c11 =>
IF(!RefReq/) THEN
RefAck/ = GND;
IGNORE_ACCESS/ = GND;
dmasm = c11;

ELSE
RefAck/ = GND;
IGNORE_ACCESS/ = GND;
dmasm = c12;
END IF;

WHEN c12 =>
dmasm = c0;%

END CASE;



%---------------------------------- Delay state machine ----------------------------------------%

% This delay state machine takes IO_ACK/ signal running from IO_clk, makes it a pulse equal to one TCLK and
feeds it to the main state machine %

 CASE delaysm IS

 WHEN d0 =>

 IF(!IO_ACK2/) THEN
 IO_ACK2_delay/ = GND;
```

```
delaysm = d1;
ELSE
delaysm = d0;
END IF;

WHEN d1 =>
delaysm = d0;


END CASE;

END;

%
```

RClk Controller

MAX+PLUS II AHDLv5.0

EDIT HISTORY

Date	Engineer	Check Sum	Changes
10/20/1994	R.Cummings		Initial Release

%

RCLK Controller PLD

```
%============================================================================%
%The RCLK Controller is used for all signals controlled by the RClock.

%

%========================== CONSTANTS ======================================%
CONSTANT 1MEG=B"001";
CONSTANT 4MEG=B"111";
CONSTANT _5MEG=B"000";
CONSTANT 2MEG=B"011";
CONSTANTBLOCK=B"1";
CONSTANTSINGLE=B"0";
%============================================================================%
SUBDESIGN RCLK_CNTRL
(
% Input signals from CPU%
B3_RCLK0:INPUT;% Buffered RClock from CPU. GLOBAL clock%

% Other input signals%
RA_CLKEN/:INPUT;% Clock enable for the Address 823's When 0 we%
% register address in 823s and then latch into%
% SRAM address 841's when it goes to 1 for all%
% SRAM accesses%
P3_SYS_IF:INPUT;% Indicates width of the SYSAD bus -%
% 0 -  32-bit bus; 1 - 64-bit bus%
SRAM_Access/:INPUT;% Indicates that the current access is for SRAM%
SRAM_DONE/:INPUT;% Indicates the SRAM is either done with the %
% current access or has valid data for the%
% current access %
DRAM_Access/:INPUT;% Indicates that the current access is for DRAM%
DRAM_PRESENT/:INPUT;% Indicates if DRAM in system (=0) or not (=1)%
MEM_DONE/:INPUT;% Indicates the DRAM is either done with the %
% current access or has valid data for the%
% current access %
RA_SYSAD28:INPUT;% Used to indicate if access is for memory (0)%
% or IO (1).%
RA_SYSAD3:INPUT;% Registered version of SysAD3 from the address%
% issue cycle. Used to determine the order for %
% LEs for a block read from memory 64-bit bus%
RA_SYSAD2:INPUT;% Registered version of SysAD2 from the address%
% issue cycle. Used to determine the order for %
% LEs for a block read from memory 32-bit bus%
R_SYSCMD3:INPUT;% Registered version of lower SysCmd bits from%
% the issue cycle. Used to determine if the %
% access is for a single datum or a block%
SYSCMD6:INPUT;% Direct version of SysCmd6 to determine read%
% and write for 50MHz SRAM accesses%
VALIDOUT/:INPUT;% ValidOut* from CPU. Used to decide if we%
% should latch the SysCmd6 signal or not%
RD/:INPUT;% The read signal from the main controller%
WR/:INPUT;% The write signal from the main controller%
RESETIN/:INPUT;% Global Reset signal%
DMA_RD/:INPUT;% Indicates a DMA read is requested%
DMA_WR/:INPUT;% Indicates a DMA write is requested%

% RCLK control output signals %
SCADDR_LE:OUTPUT;% Latch enable for the SRAM Address%
```

```
ED2RD_LE:OUTPUT;% Latch enable for the even data from memory on%
% a read request%
OD2RD_LE:OUTPUT;% Latch enable for the odd data from memory on%
% a read request%
SE_WE/:OUTPUT;% SRAM Even bank write enable%
SO_WE/:OUTPUT;% SRAM Odd bank write enable%
)

%=============================================================================%
% RCLK control state machine %

VARIABLE

rclksm:MACHINE WITH STATES
(r0, r1, r2, r3, r4, r5, r6, r7);

%---------------  Output Flip Flops  ----------------------%
SCADDR_LE:DFFE;
ED2RD_LE:DFF;
OD2RD_LE:DFF;
SE_WE/:DFF;
SO_WE/:DFF;
%-----------------------------------------------------------------------------%
% Buried nodes %
SCAddr_le_pre:NODE;% preset for SCADDR_LE latch%
SCAddr_le_clr:NODE;% clear for SCADDR_LE latch%
SCAddr_le_en:NODE;% enable for SCADDR_LE latch%
RSYSCMD6_en:NODE;% Enable for R_SYSCMD6 latch%
Count[3..0]:DFF;% Counter for block reads%
R_SYSCMD6:DFFE;% registered syscmd6 for write cycles%

%=============================================================================%

BEGIN

%-----------------------------------------------------------------------------%
% Default state for all outputs is not asserted %
DEFAULTS
SCADDR_LE=GND;
%SCADDR_LE=VCC;%
ED2RD_LE=VCC;
OD2RD_LE=VCC;
SE_WE/=VCC;
SO_WE/=VCC;
Count[3..0]=GND;
R_SYSCMD6=GND;
RSYSCMD6_en=GND;
SCAddr_le_en=GND;
SCAddr_le_pre=VCC;
SCAddr_le_clr=VCC;
END DEFAULTS;

%----------------- Output DFF clock and clear assignments  -------------------%

SCADDR_LE.clk=GLOBAL(B3_RCLK0);
SCADDR_LE.clrn=GLOBAL(RESETIN/) & SCAddr_le_clr;
SCADDR_LE.prn=SCAddr_le_pre;
SCADDR_LE.ena=SCAddr_le_en;

ED2RD_LE.clk=GLOBAL(B3_RCLK0);
ED2RD_LE.clrn=GLOBAL(RESETIN/);

OD2RD_LE.clk=GLOBAL(B3_RCLK0);
OD2RD_LE.clrn=GLOBAL(RESETIN/);

SE_WE/.clk=GLOBAL(B3_RCLK0);
SE_WE/.clrn=GLOBAL(RESETIN/);
```

```
SO_WE/.clk=GLOBAL(B3_RCLK0);
SO_WE/.clrn=GLOBAL(RESETIN/);

%--------- State machine clock and clear ------------------------------------%

rclksm.clk=GLOBAL(B3_RCLK0);
rclksm.reset=GLOBAL(!RESETIN/);

%---------- Buried nodes clock and clear -------------------------------------%

Count[3..0].clk=GLOBAL(B3_RCLK0);
Count[3..0].clrn=GLOBAL(RESETIN/);

R_SYSCMD6.clk=GLOBAL(B3_RCLK0);
R_SYSCMD6.clrn=GLOBAL(RESETIN/);
R_SYSCMD6.ena=RSYSCMD6_en;

%----------- Equations ---------------------------------------------------------%

SCADDR_LE=!(RA_CLKEN/);
RSYSCMD6_en=(!VALIDOUT/ & (rclksm == r0));
SCAddr_le_en=(rclksm == r0);
R_SYSCMD6=SYSCMD6;
%SCAddr_le_pre=!(!DMA_RD/ # !DMA_WR/);
SCAddr_le_clr=!(DMA_RD/ # DMA_WR/);%
%-----------------------------------------------------------------------------%

% state machine design %

CASE rclksm IS

WHEN r0 =>% IDLE %
IF (((!RD/ & R_SYSCMD3) # !DMA_RD/) & !RA_SYSAD28 &
(!SRAM_DONE/ # !MEM_DONE/)) THEN
% Single memory read with the done signal asserted%
SCAddr_le_pre = DMA_RD/;
ED2RD_LE = VCC;
OD2RD_LE = VCC;
rclksm = r2;
ELSIF (((!RD/ & R_SYSCMD3) # !DMA_RD/) & !RA_SYSAD28 &
(SRAM_DONE/ & MEM_DONE/)) THEN
% Single memory read with the done signal de-asserted%
SCAddr_le_pre = DMA_RD/;
ED2RD_LE = VCC;
OD2RD_LE = VCC;
rclksm = r1;
ELSIF (!RD/ & !R_SYSCMD3 & !RA_SYSAD28 & P3_SYS_IF &
(!SRAM_DONE/ # !MEM_DONE/)) THEN
% Block memory read with the 1st done signal asserted%
ED2RD_LE = VCC;
OD2RD_LE = VCC;
Count[3..0] = B"0001";
rclksm = r3;
ELSIF (!RD/ & !R_SYSCMD3 & !RA_SYSAD28 & P3_SYS_IF &
(SRAM_DONE/ & MEM_DONE/)) THEN
% Block memory read with the done signal de-asserted%
ED2RD_LE = VCC;
OD2RD_LE = VCC;
Count[3..0] = B"0000";
rclksm = r3;
ELSIF (!RD/ & !R_SYSCMD3 & !RA_SYSAD28 & !P3_SYS_IF &
(!SRAM_DONE/ # !MEM_DONE/)) THEN
% Block memory read with the 1st done signal asserted%
ED2RD_LE = VCC;
OD2RD_LE = VCC;
Count[3..0] = B"0001";
rclksm = r4;
ELSIF (!RD/ & !R_SYSCMD3 & !RA_SYSAD28 & !P3_SYS_IF &
```

```
(SRAM_DONE/ & MEM_DONE/)) THEN
% Block memory read with the done signal de-asserted%
ED2RD_LE = VCC;
OD2RD_LE = VCC;
Count[3..0] = B"0000";
rclksm = r4;
ELSIF (((R_SYSCMD6 & R_SYSCMD3) # !DMA_WR/) & !RA_SYSAD3 & P3_SYS_IF &
!SRAM_Access/) THEN
% Single write to even SRAM, 64-bit system interface%
SCAddr_le_pre = DMA_WR/;
SE_WE/ = GND;
rclksm = r5;
ELSIF (((R_SYSCMD6 & R_SYSCMD3) # !DMA_WR/) & RA_SYSAD3 & P3_SYS_IF &
!SRAM_Access/) THEN
% Single write to even SRAM, 64-bit system interface%
SCAddr_le_pre = DMA_WR/;
SO_WE/ = GND;
rclksm = r5;
ELSIF (((R_SYSCMD6 & R_SYSCMD3) # !DMA_WR/) & !RA_SYSAD2 & !P3_SYS_IF &
!SRAM_Access/) THEN
% Single write to even SRAM, 32-bit system interface%
SCAddr_le_pre = DMA_WR/;
SE_WE/ = GND;
rclksm = r5;
ELSIF (((R_SYSCMD6 & R_SYSCMD3) # !DMA_WR/) & RA_SYSAD2 & !P3_SYS_IF &
!SRAM_Access/) THEN
% Single write to even SRAM, 32-bit system interface%
SCAddr_le_pre = DMA_WR/;
SO_WE/ = GND;
rclksm = r5;
ELSIF (R_SYSCMD6 & !R_SYSCMD3 & DRAM_PRESENT/ & P3_SYS_IF & !SRAM_Access/) THEN
% Block write with no DRAM in the system, pattern is DDDD (64-bit)%
SE_WE/ = GND;
Count[3..0] = B"0000";
rclksm = r6;
ELSIF (R_SYSCMD6 & !R_SYSCMD3 & DRAM_PRESENT/ & !P3_SYS_IF & !SRAM_Access/) THEN
% Block write with no DRAM in the system, pattern is WWWWWWWW (32-bit)%
SE_WE/ = GND;
Count[3..0] = B"0000";
rclksm = r6;
ELSIF (R_SYSCMD6 & !R_SYSCMD3 & !DRAM_PRESENT/ & P3_SYS_IF & !SRAM_Access/) THEN
% Block write with no DRAM in the system, pattern is DDxxDDxx (64-bit)%
SE_WE/ = GND;
Count[3..0] = B"0000";
rclksm = r7;
ELSIF (R_SYSCMD6 & !R_SYSCMD3 & !DRAM_PRESENT/ & !P3_SYS_IF & !SRAM_Access/) THEN
% Block write with no DRAM in the system, pattern is WWxxWWxxWWxxWWxx (32-bit)%
SE_WE/ = GND;
Count[3..0] = B"0000";
rclksm = r7;
ELSE
rclksm = r0;
END IF;

WHEN r1 =>
% Single read waiting for data to become valid%
ED2RD_LE = VCC;
OD2RD_LE = VCC;

IF ((!SRAM_DONE/ & !SRAM_Access/) # (!MEM_DONE/ & !DRAM_Access/)) THEN
rclksm = r2;
ELSE
rclksm = r1;
END IF;
WHEN r2 =>
% Wait for DMA read to end from SRAM%
IF (!DMA_RD/) THEN
ED2RD_LE = VCC;
```

```
OD2RD_LE = VCC;
rclksm = r2;
ELSE
% Single read done so latch data and goto r0%
rclksm = r0;
END IF;

WHEN r3 =>
% Process block read%
IF ((Count[3..0] == B"0000") & ((SRAM_DONE/ & !SRAM_Access/) #
(MEM_DONE/ & !DRAM_Access/))) THEN
% No data ready yet. leave latch opened%
Count[3..0] = B"0000";
rclksm = r3;
ELSIF ((Count[3..0] == B"0000") & ((!SRAM_DONE/ & !SRAM_Access/) #
(!MEM_DONE/ & !DRAM_Access/))) THEN
% 1st set of doublewords ready to latch%
Count[3..0] = B"0001";
ED2RD_LE = GND;
OD2RD_LE = GND;
rclksm = r3;
ELSIF (Count[3..0] == B"0001") THEN
% Close latch to save 1st set of doublewords%
Count[3..0] = B"0010";
ED2RD_LE = GND;
OD2RD_LE = GND;
rclksm = r3;
ELSIF ((Count[3..0] == B"0010") & ((SRAM_DONE/ & !SRAM_Access/) #
(MEM_DONE/ & !DRAM_Access/))) THEN
% Waiting for 2nd set of doublewords%
Count[3..0] = B"0010";
ED2RD_LE = GND;
OD2RD_LE = GND;
rclksm = r3;
ELSIF ((Count[3..0] == B"0010") & ((!SRAM_DONE/ & !SRAM_Access/) #
(!MEM_DONE/ & !DRAM_Access/)) & !RA_SYSAD3) THEN
% 2nd set of doublewords ready to latch. Only close odd side%
Count[3..0] = B"0011";
OD2RD_LE = GND;
rclksm = r3;
ELSIF ((Count[3..0] == B"0010") & ((!SRAM_DONE/ & !SRAM_Access/) #
(!MEM_DONE/ & !DRAM_Access/)) & RA_SYSAD3) THEN
% 2nd set of doublewords ready to latch. Only close even side%
Count[3..0] = B"0011";
ED2RD_LE = GND;
rclksm = r3;
ELSIF (Count[3..0] == B"0011") THEN
% Close both sides%
Count[3..0] = B"0100";
rclksm = r3;
ELSE
ED2RD_LE = GND;
OD2RD_LE = GND;
rclksm = r0;
END IF;

WHEN r4 =>
% Process block read%
IF ((Count[3..0] == B"0000") & ((SRAM_DONE/ & !SRAM_Access/) #
(MEM_DONE/ & !DRAM_Access/))) THEN
% No data ready yet. leave latch open%
Count[3..0] = B"0000";
rclksm = r4;
ELSIF ((Count[3..0] == B"0000") & ((!SRAM_DONE/ & !SRAM_Access/) #
(!MEM_DONE/ & !DRAM_Access/))) THEN
% 1st set of words ready to latch%
Count[3..0] = B"0001";
ED2RD_LE = GND;
```

```
OD2RD_LE = GND;
rclksm = r4;
ELSIF (Count[3..0] == B"0001") THEN
% Close latch to save 1st set of words%
ED2RD_LE = GND;
OD2RD_LE = GND;
Count[3..0] = B"0010";
rclksm = r4;
ELSIF ((Count[3..0] == B"0010") & ((SRAM_DONE/ & !SRAM_Access/) #
(MEM_DONE/ & !DRAM_Access/))) THEN
% Waiting for 2nd set of words%
ED2RD_LE = GND;
OD2RD_LE = GND;
Count[3..0] = B"0010";
rclksm = r4;
ELSIF ((Count[3..0] == B"0010") & ((!SRAM_DONE/ & !SRAM_Access/) #
(!MEM_DONE/ & !DRAM_Access/)) & !RA_SYSAD2) THEN
% 2nd set of words ready to latch. Only close odd side%
Count[3..0] = B"0011";
OD2RD_LE = GND;
rclksm = r4;
ELSIF ((Count[3..0] == B"0010") & ((!SRAM_DONE/ & !SRAM_Access/) #
(!MEM_DONE/ & !DRAM_Access/)) & RA_SYSAD2) THEN
% 2nd set of words ready to latch. Only close even side%
Count[3..0] = B"0011";
ED2RD_LE = GND;
rclksm = r4;
ELSIF ((Count[3..0] == B"0011") & !RA_SYSAD2) THEN
% Close even side and open odd side%
Count[3..0] = B"0100";
ED2RD_LE = GND;
rclksm = r4;
ELSIF ((Count[3..0] == B"0011") & RA_SYSAD2) THEN
% Close odd side and open even side%
Count[3..0] = B"0100";
OD2RD_LE = GND;
rclksm = r4;
ELSIF ((Count[3..0] == B"0100") & ((SRAM_DONE/ & !SRAM_Access/) #
(MEM_DONE/ & !DRAM_Access/))) THEN
% Waiting for 3rd set of words%
ED2RD_LE = GND;
OD2RD_LE = GND;
Count[3..0] = B"0100";
rclksm = r4;
ELSIF ((Count[3..0] == B"0100") & ((!SRAM_DONE/ & !SRAM_Access/) #
(!MEM_DONE/ & !DRAM_Access/)) & !RA_SYSAD2) THEN
% 3rd set of words ready to latch. Only close odd side%
Count[3..0] = B"0101";
OD2RD_LE = GND;
rclksm = r4;
ELSIF ((Count[3..0] == B"0100") & ((!SRAM_DONE/ & !SRAM_Access/) #
(!MEM_DONE/ & !DRAM_Access/)) & RA_SYSAD2) THEN
% 3rd set of words ready to latch. Only close even side%
Count[3..0] = B"0101";
ED2RD_LE = GND;
rclksm = r4;
ELSIF ((Count[3..0] == B"0101") & !RA_SYSAD2) THEN
% Close even side and open odd side%
Count[3..0] = B"0110";
ED2RD_LE = GND;
rclksm = r4;
ELSIF ((Count[3..0] == B"0101") & RA_SYSAD2) THEN
% Close odd side and open even side%
Count[3..0] = B"0110";
OD2RD_LE = GND;
rclksm = r4;
ELSIF ((Count[3..0] == B"0110") & ((SRAM_DONE/ & !SRAM_Access/) #
(MEM_DONE/ & !DRAM_Access/))) THEN
```

```
ED2RD_LE = GND;
OD2RD_LE = GND;
Count[3..0] = B"0110";
rclksm = r4;
ELSIF ((Count[3..0] == B"0110") & ((!SRAM_DONE/ & !SRAM_Access/) #
(!MEM_DONE/ & !DRAM_Access/)) & !RA_SYSAD2) THEN
% 3rd set of words ready to latch. Only close odd side%
Count[3..0] = B"0111";
OD2RD_LE = GND;
rclksm = r4;
ELSIF ((Count[3..0] == B"0110") & ((!SRAM_DONE/ & !SRAM_Access/) #
(!MEM_DONE/ & !DRAM_Access/)) & RA_SYSAD2) THEN
% 3rd set of words ready to latch. Only close even side%
Count[3..0] = B"0111";
ED2RD_LE = GND;
rclksm = r4;
ELSIF ((Count[3..0] == B"0111") & !RA_SYSAD2) THEN
% Close even side and open odd side%
Count[3..0] = B"1000";
ED2RD_LE = GND;
rclksm = r4;
ELSIF ((Count[3..0] == B"0111") & RA_SYSAD2) THEN
% Close odd side and open even side%
Count[3..0] = B"1000";
OD2RD_LE = GND;
rclksm = r4;
ELSE
ED2RD_LE = GND;
OD2RD_LE = GND;
rclksm = r0;
END IF;

WHEN r5 =>
% Loop until DMA write is done keep WE asserted%
SCAddr_le_clr = SRAM_DONE/;

IF (!DMA_WR/) THEN
SO_WE/ = SO_WE/;
SE_WE/ = SE_WE/;
rclksm = r5;
ELSE
% Single write to SRAM from CPU or DMA write done%
rclksm = r0;
END IF;

WHEN r6 =>
% Block write to SRAM with no DRAM in system%
IF (P3_SYS_IF & (Count[3..0] == B"0000")) THEN
% Write 1st data - DDDD pattern%
SO_WE/ = GND;
Count[3..0] = B"0001";
rclksm = r6;
ELSIF (P3_SYS_IF & (Count[3..0] == B"0001")) THEN
% Write 2nd data - DDDD pattern%
SE_WE/ = GND;
Count[3..0] = B"0010";
rclksm = r6;
ELSIF (P3_SYS_IF & (Count[3..0] == B"0010")) THEN
% Write 3rd data - DDDD pattern%
SO_WE/ = GND;
Count[3..0] = B"0011";
rclksm = r6;
ELSIF (P3_SYS_IF & (Count[3..0] == B"0011")) THEN
% Write 4th data - DDDD pattern%
rclksm = r0;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0000")) THEN
% Write 1st data - WWWWWWWW pattern%
SO_WE/ = GND;
```

```
Count[3..0] = B"0001";
rclksm = r6;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0001")) THEN
% Write 2nd data - WWWWWWWW pattern%
SE_WE/ = GND;
Count[3..0] = B"0010";
rclksm = r6;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0010")) THEN
% Write 3rd data - WWWWWWWW pattern%
SO_WE/ = GND;
Count[3..0] = B"0011";
rclksm = r6;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0011")) THEN
% Write 4th data - WWWWWWWW pattern%
SE_WE/ = GND;
Count[3..0] = B"0100";
rclksm = r6;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0100")) THEN
% Write 5th data - WWWWWWWW pattern%
SO_WE/ = GND;
Count[3..0] = B"0101";
rclksm = r6;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0101")) THEN
% Write 6th data - WWWWWWWW pattern%
SE_WE/ = GND;
Count[3..0] = B"0110";
rclksm = r6;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0110")) THEN
% Write 7th data - WWWWWWWW pattern%
SO_WE/ = GND;
Count[3..0] = B"0111";
rclksm = r6;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0111")) THEN
% Write 8th data - WWWWWWWW pattern%
rclksm = r0;
ELSE
rclksm = r0;
END IF;

WHEN r7 =>
% Block write to SRAM with DRAM in system%
IF (P3_SYS_IF & (Count[3..0] == B"0000")) THEN
% Write 1st data - DDxxDDxx pattern%
SO_WE/ = GND;
Count[3..0] = B"0001";
rclksm = r7;
ELSIF (P3_SYS_IF & (Count[3..0] == B"0001")) THEN
% Write 2nd data - DDxxDDxx pattern%
Count[3..0] = B"0010";
rclksm = r7;
ELSIF (P3_SYS_IF & (Count[3..0] == B"0010")) THEN
% 1st X of 1st set - DDxxDDxx pattern%
Count[3..0] = B"0011";
rclksm = r7;
ELSIF (P3_SYS_IF & (Count[3..0] == B"0011")) THEN
% 2nd X of 1st set - DDxxDDxx pattern%
SE_WE/ = GND;
Count[3..0] = B"0100";
rclksm = r7;
ELSIF (P3_SYS_IF & (Count[3..0] == B"0100")) THEN
% Write 3rd data - DDxxDDxx pattern%
SO_WE/ = GND;
Count[3..0] = B"0101";
rclksm = r7;
ELSIF (P3_SYS_IF & (Count[3..0] == B"0101")) THEN
% Write 4th data - DDxxDDxx pattern%
Count[3..0] = B"0110";
rclksm = r7;
```

```
ELSIF (P3_SYS_IF & (Count[3..0] == B"0110")) THEN
% 1st X of 2nd set - DDxxDDxx pattern%
% go back to idle and wait for next %
rclksm = r0;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0000")) THEN
% Write 1st data - WWxxWWxxWWxxWWxx pattern%
SO_WE/ = GND;
Count[3..0] = B"0001";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0001")) THEN
% Write 2nd data - WWxxWWxxWWxxWWxx pattern%
Count[3..0] = B"0010";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0010")) THEN
% 1st X of 1st set - WWxxWWxxWWxxWWxx pattern%
Count[3..0] = B"0011";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0011")) THEN
% 2nd X of 1st set - WWxxWWxxWWxxWWxx pattern%
SE_WE/ = GND;
Count[3..0] = B"0100";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0100")) THEN
% Write 3rd data - WWxxWWxxWWxxWWxx pattern%
SO_WE/ = GND;
Count[3..0] = B"0101";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0101")) THEN
% Write 4th data - WWxxWWxxWWxxWWxx pattern%
Count[3..0] = B"0110";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0110")) THEN
% 1st X of 2nd set - WWxxWWxxWWxxWWxx pattern%
Count[3..0] = B"0111";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"0111")) THEN
% 2nd X of 2nd set - WWxxWWxxWWxxWWxx pattern%
SE_WE/ = GND;
Count[3..0] = B"1000";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"1000")) THEN
% Write 5th data - WWxxWWxxWWxxWWxx pattern%
SO_WE/ = GND;
Count[3..0] = B"1001";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"1001")) THEN
% Write 6th data - WWxxWWxxWWxxWWxx pattern%
Count[3..0] = B"1010";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"1010")) THEN
% 1st X of 3rd set - WWxxWWxxWWxxWWxx pattern%
Count[3..0] = B"1011";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"1011")) THEN
% 2nd X of 3rd set - WWxxWWxxWWxxWWxx pattern%
SE_WE/ = GND;
Count[3..0] = B"1100";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"1100")) THEN
% Write 7th data - WWxxWWxxWWxxWWxx pattern%
SO_WE/ = GND;
Count[3..0] = B"1101";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"1101")) THEN
% Write 8th data - WWxxWWxxWWxxWWxx pattern%
Count[3..0] = B"1110";
rclksm = r7;
ELSIF (!P3_SYS_IF & (Count[3..0] == B"1110")) THEN
```

```
% 1st X of 4th set - WWxxWWxxWWxxWWxx pattern%
% go back to idle and wait for next %
rclksm = r0;
ELSE
rclksm = r0;
END IF;

WHEN OTHERS =>
rclksm = r0;
END CASE;

END;
%
```

Reset Controller

MAX+PLUS II AHDLv5.0

EDIT HISTORY

Date	Engineer	Check Sum	Changes
10/12/1994	R.Cummings		Initial Release

%

Reset Controller PLD

```
%============================================================================%

%========================== CONSTANTS ======================================%

%============================================================================%
SUBDESIGN Reset
(
% Input signals from CPU%
MASTEROUT:INPUT;% The Global clock for the controller.  Used to %
% generate CPU Reset/ and Cold_reset/ signals%
MODECLOCK:INPUT;% CPU ModeClock used for control of Mode bits%

% Other input signals%
RESET_IN/:INPUT;% The reset signal from the TL7705B reset chip%
DRAM_SIZE[1..0]:INPUT;% Indicates if DRAM installed%
%DRAM_SIZE[1..0]Description%
%11No DRAM%
%00, 01, 10DRAM installed%
P3_SYS_IF:INPUT;% Indicates size of P3 sys I/F%
%P3_SYS_IFDescription%
%032-bit interface%
%164-bit interface%
R4K_WRITE:INPUT;% Indicates to use R4K compatible writes for %
% non-block writes from the CPU%
%R4K_WRITEDescription%
%0R4K write compatible mode%
%1either pipelined or re-issue %
PIPELINED:INPUT;% Indicates which of the new write mode if not%
% using R4K compatible write mode - non-block %
%PIPELINEDDescription%
%0Pipelined write mode%
%1Write re-issue mode%
DRIVE:INPUT;% Indicates drive strength for output%
%DRIVEDescription%
%0100 percent drive%
%183 percent drive%
TIMER_INT:INPUT;% Indicates if Timer interrupt is enabled%
%TIMER_INTDescription%
%0Timer interrupt enabled%
%1Timer interrupt disabled%
BIG_ENDIAN:INPUT;% Indicates if a big or little endian system%
%BIG_ENDIANDescription%
%0Little endian system%
%1Big endian system%
CLK_DIV[2..0]:INPUT;% Indicates the clock divisor for the System%
% interface clocks for R4600/R4700 or the clock%
% multiplier for a R4650 system%
%CLK_DIV[2..0]Divisor (multiplier)%
%0002%
%0013%
%0104%
%0115%
%1006%
%1017%
%1108%
%111Reserved%
```

```
% RESET control output signals %
DRAM_PRESENT/:OUTPUT;% Indicates that DRAM is installed in system%
%DRAM_PRESENT/Description%
%0DRAM Installed in system%
%1No DRAM in system%
MODEIN:OUTPUT;% The Mode bit stream to the CPU%
VCCOK:OUTPUT;% Indicates to the CPU that power is applied%
RESET/:OUTPUT;% The CPU Reset* signal used for all resets%
COLD_RESET/:OUTPUT;% The CPU ColdReset* signal, used for power on %
% and cold resets%
SYSTEM_RESET/:OUTPUT;% The reset signal to most of the system except%
% the SONIC, SCSI and IO sub-systems%
SONIC_RESET/:OUTPUT;% The reset for the SONIC %
SCSI_RESET:OUTPUT;% The reset for the SCSI chip (53C90A)%
IO_RESET/:OUTPUT;% The reset for the other IO devices%
)
%=============================================================================%
% Bus Exchanger control state machine %

VARIABLE

master_sm:MACHINE WITH STATES
(mst0, mst1, mst2, mst3);

%---------------  Output Flip Flops  ----------------------%
DRAM_PRESENT/:DFF;
MODEIN:DFF;
VCCOK:DFF;
RESET/:DFF;
COLD_RESET/:DFF;
SYSTEM_RESET/:DFF;
SONIC_RESET/:DFF;
SCSI_RESET:DFF;
IO_RESET/:DFF;

%-----------------------------------------------------------------------------%
% Buried nodes %
Mode_done/:NODE;% Indicates to the mater control that the mode%
% bits are all done and allows it to start%
master_count[6..0]:DFF;% masterout counter for CPU reset signals%
mode_count[7..0]:DFF;% modeclock counter for mode bit control%
divisor[7..5]:NODE;% the clock divisor mode bits%
Byte_order:NODE;% the CPU endianness%
Write_pat:NODE;% selects the block write-back pattern%
R4k_wrt:NODE;% selects the R4k write mode or not%
Piped:NODE;% selects pipelined or re-issue if not r4k mode%
Time_int:NODE;% Enables or disables the timer interrupt%
Sys_if:NODE;% selects the size of the system interface%
Drvr:NODE;% selects the drive strength of the outputs%

%=============================================================================%

BEGIN

%-----------------------------------------------------------------------------%
% Default state for all outputs is not asserted %
DEFAULTS
DRAM_PRESENT/=VCC;
MODEIN=GND;
VCCOK=VCC;
RESET/=VCC;
COLD_RESET/=VCC;
SYSTEM_RESET/=VCC;
SONIC_RESET/=VCC;
SCSI_RESET=GND;
IO_RESET/=VCC;

Mode_done/=VCC;
```

```
master_count[6..0]=GND;
mode_count[7..0]=GND;
END DEFAULTS;

%---------------- Output DFF clock and clear assignments  ------------------------%

DRAM_PRESENT/.clk=GLOBAL(MASTEROUT);
DRAM_PRESENT/.prn=GLOBAL(RESET_IN/);

MODEIN.clk=MODECLOCK;
MODEIN.clrn=GLOBAL(RESET_IN/);

VCCOK.clk=GLOBAL(MASTEROUT);
VCCOK.clrn=GLOBAL(RESET_IN/);

RESET/.clk=GLOBAL(MASTEROUT);
RESET/.clrn=GLOBAL(RESET_IN/);

COLD_RESET/.clk=GLOBAL(MASTEROUT);
COLD_RESET/.clrn=GLOBAL(RESET_IN/);

SYSTEM_RESET/.clk=GLOBAL(MASTEROUT);
SYSTEM_RESET/.clrn=GLOBAL(RESET_IN/);

SONIC_RESET/.clk=GLOBAL(MASTEROUT);
SONIC_RESET/.clrn=GLOBAL(RESET_IN/);

SCSI_RESET.clk=GLOBAL(MASTEROUT);
SCSI_RESET.prn=GLOBAL(RESET_IN/);

IO_RESET/.clk=GLOBAL(MASTEROUT);
IO_RESET/.clrn=GLOBAL(RESET_IN/);

%--------- State machine clock and clear ----------------------------------%

master_sm.clk=GLOBAL(MASTEROUT);
master_sm.reset=GLOBAL(!RESET_IN/);

%---------- Buried nodes clock and clear ----------------------------------%

master_count[6..0].clk=GLOBAL(MASTEROUT);
master_count[6..0].clrn=GLOBAL(RESET_IN/);

mode_count[7..0].clk=MODECLOCK;
mode_count[7..0].clrn=GLOBAL(RESET_IN/);

%----------- Equations ------------------------------------------------------%

mode_count[7..0] = mode_count[7..0] + B"00000001";

Mode_done/ = !(mode_count[7..0] == B"11111111");

MODEIN = Byte_order # divisor5 # divisor6 # divisor7 # Write_pat # R4k_wrt #
Piped # Time_int # Sys_if # Drvr;

DRAM_PRESENT/ = (DRAM_SIZE[1..0] == B"11");
Byte_order = BIG_ENDIAN & (mode_count[7..0] == B"00000111");
divisor5 = CLK_DIV[0] & (mode_count[7..0] == B"00000100");
divisor6 = CLK_DIV[1] & (mode_count[7..0] == B"00000101");
divisor7 = CLK_DIV[2] & (mode_count[7..0] == B"00000110");
Write_pat = !DRAM_PRESENT/ & (mode_count[7..0] == B"00000001");
R4k_wrt = R4K_WRITE & (mode_count[7..0] == B"00001000");
Piped = PIPELINED & (mode_count[7..0] == B"00001001");
Time_int = TIMER_INT & (mode_count[7..0] == B"00001010");
Sys_if = !P3_SYS_IF & (mode_count[7..0] == B"00001011");
Drvr =(DRIVE & (mode_count[7..0] == B"00001100")) #
(mode_count[7..0] == B"00001101");
```

```
%-------------------------------------------------------------------------------%

% state machine design %

CASE master_sm IS

WHEN mst0 =>
% master control IDLE%
RESET/ = GND;
SYSTEM_RESET/ = GND;
SONIC_RESET/ = GND;
SCSI_RESET = VCC;
IO_RESET/ = GND;

IF (Mode_done/) THEN
% Wait for mode bits to all load%
COLD_RESET/ = GND;
master_sm = mst0;
ELSE
% mode bits done de-assert ColdReset*%
COLD_RESET/ = VCC;
master_count[6..0] = B"0000000";
master_sm = mst1;
END IF;

WHEN mst1 =>
% now wait for 64 master clocks and de-assert Reset*%
master_count[6..0] = master_count[6..0] + B"0000001";
SYSTEM_RESET/ = GND;
SONIC_RESET/ = GND;
SCSI_RESET = VCC;
IO_RESET/ = GND;

IF (master_count[6..0] == B"0111111") THEN
RESET/ = VCC;
master_sm = mst2;
ELSE
RESET/ = GND;
master_sm = mst1;
END IF;

WHEN mst2 =>
% now de-assert the other resets after another 64 master clocks%
master_count[6..0] = master_count[6..0] + B"0000001";

IF (master_count[6..0] == B"1111110") THEN
SYSTEM_RESET/ = VCC;
SONIC_RESET/ = VCC;
SCSI_RESET = GND;
IO_RESET/ = VCC;
master_sm = mst3;
ELSE
SYSTEM_RESET/ = GND;
SONIC_RESET/ = GND;
SCSI_RESET = VCC;
IO_RESET/ = GND;
master_sm = mst2;
END IF;

WHEN mst3 =>
% reset now done just stay here until another one starts things again%

master_sm = mst3;

END CASE;

END;
%
```

SONIC Controller

MAX+PLUS II AHDL v5.00

EDIT HISTORY

Date	Engineer	Checksum	Change	Comments
	S. Khan			Initial release

This controller controls the sonic interface and provides DMA control for expansion bus.

%

Sonic Controller PLD

```
DESIGN IS “sonic_ctr”
DEVICE “sonic_ctr” IS “ EPM7096LC84-12”;

%_____ %

SUBDESIGN sonic_ctr
(
IO_ADDR[28..20]:INPUT;
IO_ADDRn2:INPUT;
IO_Syscmd[3..0]:BIDIR; % Outputs during DMA %


HTCLK:INPUT;
HMRes/:INPUT;
SMST_WR:INPUT;

IO_ERR/:INPUT;

SY_RD/:INPUT; % syncronized RD/ & WR/ signals from main_ctr %
SY_WR/:INPUT;
LS_SMACK/:INPUT;
LS_DSACK0/:BIDIR;
LS_DSACK1/:BIDIR;

P3_SYS_IF:INPUT;

LS_BR/:INPUT;

IOEnable/:INPUT;
IOEnable1:INPUT;

IO_RD/:BIDIR;
IO_WR/:BIDIR;

DMA_ACK/:INPUT;
S_BGACK/:INPUT;
L_SMST_AS/:INPUT;
MEM_DONE/:INPUT;
SRAM_DONE/:INPUT;
OTHER_IO_DONE/:INPUT;


IO_Access/:OUTPUT;
S_SLAVERD/:OUTPUT;
S_AD/:OUTPUT;
SONIC_CS/:OUTPUT;
IO_DONE/:OUTPUT;
IO_Syscmd6:OUTPUT;
DMA_REQ/:OUTPUT;
IO_ENABLE/:OUTPUT;

INV_IO_ENABLE/:OUTPUT;
```

```
IO_ENABLE1:OUTPUT;
S_BG/:OUTPUT;
DMA_RD/:OUTPUT;
DMA_WR/:OUTPUT;
SRAM_ACK/:OUTPUT; % A dummy output for the node. If not declared as output compiler  totally ignores this node %

IOEX_OE1B_R:OUTPUT; %EW_DATA to R_SYSAD %
IOEX_OE2B_R:OUTPUT;
IOEX_LEA1B_R:OUTPUT;
IOEX_LEA2B_R:OUTPUT;

IOEX_OEA/:BIDIR;% R_SYSAD to EW_DATA %
IO_EX_SEL:BIDIR;
IOEX_LE1B:BIDIR;
IOEX_LE2B:BIDIR;

S_STERM/:OUTPUT;
S_MEMREQ/:OUTPUT;

% Expansion Bus %

EX_BREQ/:INPUT;
EX_BGRD/:OUTPUT;
EX_DIR/:OUTPUT;
INVERT_EX_DIR/:OUTPUT;
IO_ACK3/:BIDIR;
EX_UCS/:OUTPUT;


)
%_____ %

VARIABLE


IO_Access/:DFF;
S_SLAVERD/:DFF;
S_AD/:DFF;
SONIC_CS/:DFF;
IO_DONE/:DFF;
IO_Syscmd_tri[3..0]:TRI;
IO_Syscmd_id[3..0]:DFF;
IO_Syscmd_trin6:TRI;
IO_Syscmd_idn6:DFF;
DMA_REQ/:DFF;
S_DMA/:DFF;
IO_ENABLE/:DFF;
IO_ENABLE1:DFF;
S_BG/:DFF;
LS_DSACK0/_tri:TRI;
LS_DSACK1/_tri:TRI;
LS_DSACK0/_id:DFF;
LS_DSACK1/_id:DFF;
SRAM_ACK/:DFF; % This FF delays the SRAM_Done/ coming from SRAM when sonic is doing DMA to SRAM. This is because SRAM accesses are zero wait state cycles which are too fast for Sonic which is running at half of the clock %
DMA_RD/:DFF;
DMA_WR/:DFF;
IOEX_OE1B/:DFF;
IOEX_OE2B/:DFF;
IOEX_LEA1B:DFF;
IOEX_LEA2B:DFF;

IOEX_OE1B_R:TRI;
IOEX_OE2B_R:TRI;
```

```
IOEX_LEA1B_R:TRI;
IOEX_LEA2B_R:TRI;


IOEX_OEA/_tri:TRI;
IO_EX_SEL_tri:TRI;
IOEX_LE1B_tri:TRI;
IOEX_LE2B_tri:TRI;

IOEX_OEA/_id:DFF;
IO_EX_SEL_id:DFF;
IOEX_LE1B_id:DFF;
IOEX_LE2B_id:DFF;

S_STERM/:DFF;
S_MEMREQ/:DFF;

EX_BGRD/:DFF;
EX_DIR/:DFF;

EX_ACK_NODE/:DFF;

IO_RD/_in:DFF;
IO_WR/_in:DFF;

IO_RD/:TRI;
IO_WR/:TRI;

EX_UCS/:DFF;
IO_ACK3/:TRI;
IO_ACK3/_in:DFF;
IO_DONE_ET/:DFF;
SONIC_DELAY[1..0]:DFF;
SONIC_DELAY_pr:DFF;

%============================================================================== %


sonicsm: MACHINE WITH STATES(
sc0, sc1, sc2, sc3, sc4, sc5, sc6, sc7, sc8,sc9);

massm: MACHINE WITH STATES(
m0, m1, m2, m3, m4, m5, m6, m7, m8, m9, m10, m11, m12, m13, m14, m15);

exsm: MACHINE WITH STATES(
ex0, ex1, ex2, ex3, ex4, ex5);


%==============================================================================%


BEGIN

DEFAULTS
IO_Access/= VCC;
S_AD/= VCC;
S_SLAVERD/= VCC;
SONIC_CS/= VCC;
IO_DONE/= VCC;
IO_Syscmd_id[3..0]= VCC;
DMA_REQ/= VCC;
S_DMA/= VCC;
IO_ENABLE/= VCC;
IO_ENABLE1= GND;
S_BG/= VCC;
LS_DSACK0/_id= VCC;
LS_DSACK1/_id= VCC;
IO_Syscmd_idn6= VCC;
```

```
DMA_RD/= VCC;
DMA_WR/= VCC;
IOEX_OE1B/= VCC;
IOEX_OE2B/= VCC;
IOEX_LEA1B= VCC;
IOEX_LEA2B= VCC;
S_STERM/= VCC;
S_MEMREQ/= VCC;
IOEX_OEA/_id= VCC;
IO_EX_SEL_id= VCC;
IOEX_LE1B_id= VCC;
IOEX_LE2B_id= VCC;
EX_BGRD/= VCC;
EX_DIR/= VCC;
IO_RD/_in = VCC;
IO_WR/_in= VCC;
EX_ACK_NODE/= VCC;
EX_UCS/= VCC;
IO_ACK3/_in= VCC;
%SRAM_ACK/= VCC;%
IO_DONE_ET/= VCC;
SONIC_DELAY[1..0] = B"00";
SONIC_DELAY_pr= VCC;


END DEFAULTS;

%_____________________________________________________________________________ %


IO_Access/.clk= GLOBAL(HTCLK);
IO_Access/.prn= GLOBAL(HMRes/);

exsm.clk= GLOBAL(HTCLK);
exsm.reset= GLOBAL(!HMRes/);

sonicsm.clk= GLOBAL(HTCLK);
sonicsm.reset= GLOBAL(!HMRes/);

massm.clk= GLOBAL(HTCLK);
massm.reset= GLOBAL(!HMRes/);

S_SLAVERD/.clk= GLOBAL(HTCLK);
S_SLAVERD/.prn= GLOBAL(HMRes/);

S_AD/.clk= GLOBAL(HTCLK);
S_AD/.prn= GLOBAL(HMRes/);

SONIC_CS/.clk= GLOBAL(HTCLK);
SONIC_CS/.prn= GLOBAL(HMRes/);

IO_DONE/.clk= GLOBAL(HTCLK);
IO_DONE/.prn= GLOBAL(HMRes/);

IO_Syscmd_tri[3..0].oe = !GLOBAL(IOEnable/);
IO_Syscmd_tri[3..0].in = IO_Syscmd_id[3..0];

IO_Syscmd_id[3..0].clk= GLOBAL(HTCLK);
IO_Syscmd_id[3..0].prn= GLOBAL(HMRes/);

IO_Syscmd_trin6.oe= !GLOBAL(IOEnable/);
IO_Syscmd_trin6.in= IO_Syscmd_idn6;

IO_Syscmd_idn6.clk= GLOBAL(HTCLK);
IO_Syscmd_idn6.prn= GLOBAL(HMRes/);

DMA_REQ/.clk= GLOBAL(HTCLK);
DMA_REQ/.prn= GLOBAL(HMRes/);
```

```
S_DMA/.clk= GLOBAL(HTCLK);
S_DMA/.prn= GLOBAL(HMRes/);

IO_ENABLE/.clk= GLOBAL(HTCLK);
IO_ENABLE/.prn= GLOBAL(HMRes/);

IO_ENABLE1.clk= GLOBAL(HTCLK);
IO_ENABLE1.clrn= GLOBAL(HMRes/);

S_BG/.clk= GLOBAL(HTCLK);
S_BG/.prn= GLOBAL(HMRes/);

LS_DSACK0/_tri.oe= !GLOBAL(IOEnable/);
LS_DSACK0/_tri.in= LS_DSACK0/_id;

LS_DSACK1/_tri.oe= !GLOBAL(IOEnable/);
LS_DSACK1/_tri.in= LS_DSACK1/_id;

LS_DSACK0/_id.clk= GLOBAL(HTCLK);
LS_DSACK0/_id.prn= GLOBAL(HMRes/);
LS_DSACK0/_id.clrn= MEM_DONE/ # S_BGACK/;

LS_DSACK1/_id.clk= GLOBAL(HTCLK);
LS_DSACK1/_id.prn= GLOBAL(HMRes/);
LS_DSACK1/_id.clrn= MEM_DONE/ # S_BGACK/;

SRAM_ACK/.clk= GLOBAL(HTCLK);
SRAM_ACK/.prn= GLOBAL(HMRes/);
SRAM_ACK/.clrn= SRAM_DONE/ # S_BGACK/;
SRAM_ACK/.d= VCC;

DMA_RD/.clk= GLOBAL(HTCLK);
DMA_RD/.prn= GLOBAL(HMRes/);

DMA_WR/.clk= GLOBAL(HTCLK);
DMA_WR/.prn= GLOBAL(HMRes/);

IOEX_OE1B/.clk= GLOBAL(HTCLK);
IOEX_OE1B/.prn= GLOBAL(HMRes/);

IOEX_OE2B/.clk= GLOBAL(HTCLK);
IOEX_OE2B/.prn= GLOBAL(HMRes/);


IOEX_LEA1B.clk= GLOBAL(HTCLK);
IOEX_LEA1B.prn= GLOBAL(HMRes/);

IOEX_LEA2B.clk= GLOBAL(HTCLK);
IOEX_LEA2B.prn= GLOBAL(HMRes/);

IOEX_OE1B_R.oe= GND;
IOEX_OE1B_R.in= IOEX_OE1B/;

IOEX_OE2B_R.oe= GND;
IOEX_OE2B_R.in= IOEX_OE2B/;

IOEX_LEA1B_R.oe= GND;
IOEX_LEA1B_R.in= IOEX_LEA1B;

IOEX_LEA2B_R.oe= GND;
IOEX_LEA2B_R.in= IOEX_LEA2B;


IOEX_OEA/_tri.oe= !GLOBAL(IOEnable/);
IOEX_OEA/_tri.in= IOEX_OEA/_id;

IO_EX_SEL_tri.oe= !GLOBAL(IOEnable/);
```

```
IO_EX_SEL_tri.in= IO_EX_SEL_id;

IOEX_LE1B_tri.oe= !GLOBAL(IOEnable/);
IOEX_LE1B_tri.in= IOEX_LE1B_id;

IOEX_LE2B_tri.oe= !GLOBAL(IOEnable/);
IOEX_LE2B_tri.in= IOEX_LE2B_id;

IOEX_OEA/_id.clk= GLOBAL(HTCLK);
IOEX_OEA/_id.prn= GLOBAL(HMRes/);

IO_EX_SEL_id.clk= GLOBAL(HTCLK);
IO_EX_SEL_id.prn= GLOBAL(HMRes/);

IOEX_LE1B_id.clk= GLOBAL(HTCLK);
IOEX_LE1B_id.prn= GLOBAL(HMRes/);

IOEX_LE2B_id.clk= GLOBAL(HTCLK);
IOEX_LE2B_id.prn= GLOBAL(HMRes/);


S_STERM/.clk= GLOBAL(HTCLK);
S_STERM/.prn= GLOBAL(HMRes/);

S_MEMREQ/.clk= GLOBAL(HTCLK);
S_MEMREQ/.prn= GLOBAL(HMRes/);


EX_BGRD/.clk= GLOBAL(HTCLK);
EX_BGRD/.prn= GLOBAL(HMRes/);

EX_DIR/.clk= GLOBAL(HTCLK);
EX_DIR/.prn= GLOBAL(HMRes/);

IO_RD/.oe= !GLOBAL(IOEnable1);
IO_RD/.in= IO_RD/_in;

IO_WR/.oe= !GLOBAL(IOEnable1);
IO_WR/.in= IO_WR/_in;

IO_WR/_in.clk= GLOBAL(HTCLK);
IO_WR/_in.prn= GLOBAL(HMRes/);
IO_WR/_in.d= VCC;

IO_RD/_in.clk= GLOBAL(HTCLK);
IO_RD/_in.prn= GLOBAL(HMRes/);
IO_RD/_in.d= VCC;

EX_ACK_NODE/.clk= GLOBAL(HTCLK);
EX_ACK_NODE/.prn= GLOBAL(HMRes/);
EX_ACK_NODE/.clrn= (MEM_DONE/ & SRAM_DONE/) # !S_BGACK/;
EX_ACK_NODE/.d= VCC;

IO_ACK3/.oe= !GLOBAL(IOEnable/);
IO_ACK3/.in= IO_ACK3/_in;

IO_ACK3/_in.clk= GLOBAL(HTCLK);
IO_ACK3/_in.prn= GLOBAL(HMRes/);


EX_UCS/.clk= GLOBAL(HTCLK);
EX_UCS/.prn= GLOBAL(HMRes/);

IO_DONE_ET/.clk= GLOBAL(HTCLK);
IO_DONE_ET/.prn= GLOBAL(HMRes/);

% Equations %
```

```
IO_Syscmd[3..0]= IO_Syscmd_tri[3..0].out;
IO_Syscmd6= IO_Syscmd_trin6.out;

IOEX_OEA/= IOEX_OEA/_tri.out;
IO_EX_SEL= IO_EX_SEL_tri.out;
IOEX_LE1B= IOEX_LE1B_tri.out;
IOEX_LE2B= IOEX_LE2B_tri.out;

LS_DSACK0/= LS_DSACK0/_tri.out;
LS_DSACK1/= LS_DSACK1/_tri.out;

SONIC_DELAY[1..0].clk = GLOBAL(HTCLK);
SONIC_DELAY[1..0].clrn = GLOBAL(HMRes/) & SONIC_DELAY_pr;

SONIC_DELAY_pr.clk= GLOBAL(HTCLK);
SONIC_DELAY_pr.prn= GLOBAL(HMRes/);

LS_DSACK0/_id= SRAM_ACK/;
LS_DSACK1/_id= SRAM_ACK/;

INV_IO_ENABLE/= !IO_ENABLE/;

INVERT_EX_DIR/= EX_DIR/;

IO_DONE/ = OTHER_IO_DONE/ & IO_DONE_ET/;


%_____________________________________________________________________________ %


CASE sonicsm IS

WHEN sc0 =>
IF((!SY_RD/ # !SY_WR/) & (IO_ADDR[28..20] == H"1f6") &
(IO_Syscmd3 == B"1")) THEN

IO_Access/ = GND;
S_SLAVERD/ = !SY_RD/ & SY_WR/;
%S_AD/= GND;%
sonicsm = sc9;
ELSE
sonicsm = sc0;
END IF;

WHEN sc9 =>
IO_Access/ = GND;
 S_SLAVERD/ = !SY_RD/ & SY_WR/;
S_AD/= GND;
sonicsm = sc1;


WHEN sc1 =>
IO_Access/ = GND;
SONIC_CS/ = GND;
S_SLAVERD/ = !SY_RD/ & SY_WR/;
S_AD/= GND;
sonicsm = sc2;

WHEN sc2 =>
IF(!LS_SMACK/) THEN
SONIC_CS/ = GND;
S_SLAVERD/ = !SY_RD/ & SY_WR/;
S_AD/= GND;
IO_Access/ = GND;


sonicsm = sc3;
ELSE
```

```
S_SLAVERD/ = !SY_RD/ & SY_WR/;
SONIC_CS/ = GND;
S_AD/= GND;
IO_Access/ = GND;
sonicsm = sc2;
END IF;


WHEN sc3 =>
IF(!LS_DSACK0/ & !LS_DSACK1/) THEN
sonicsm = sc4;
S_AD/= GND;
IO_DONE_ET/ = GND;
S_SLAVERD/ =!SY_RD/ & SY_WR/;
SONIC_CS/ = GND;
IO_Access/ = GND;


ELSE
S_SLAVERD/ = !SY_RD/ & SY_WR/;
S_AD/= GND;
sonicsm = sc3;
SONIC_CS/ = GND;
IO_Access/ = GND;
END IF;

WHEN sc4 =>
SONIC_CS/ = GND;
S_SLAVERD/ = !SY_RD/ & SY_WR/;
S_AD/= GND;
IO_Access/ = GND;
sonicsm = sc5;

WHEN sc5 =>
S_SLAVERD/ = !SY_RD/ & SY_WR/;
SONIC_CS/ = GND;
IO_Access/ = GND;
sonicsm =sc7;


   WHEN sc7 =>
sonicsm = sc8;
IO_Access/ = GND;


WHEN sc8 =>
IO_Access/ = GND;
sonicsm = sc0;

WHEN OTHERS =>
sonicsm = sc0;


END CASE;

%_____________________________________________________________________________ %

CASE massm IS

WHEN m0 =>

IF(!LS_BR/ # !EX_BREQ/) THEN
DMA_REQ/ = GND;
S_DMA/ = GND;
massm = m1;
ELSE
massm = m0;
END IF;
```

```
WHEN m1 =>
IF(!DMA_ACK/) THEN
DMA_REQ/ = GND;
massm = m2;
ELSE
DMA_REQ/ = GND;
S_DMA/ = GND;
massm = m1;
END IF;

WHEN m2 =>
IF(!LS_BR/ & EX_BREQ/) THEN  % Sonic has higher priority %
S_BG/ = LS_BR/;
DMA_REQ/= GND;
S_DMA/ = GND;
massm = m3;

ELSIF(LS_BR/ & !EX_BREQ/) THEN

DMA_REQ/ = GND;
IO_ENABLE1 = VCC;
IO_ENABLE/ = GND;
EX_BGRD/ = GND;% Acknowledge to exp. bus and not Sonic %
massm = m10;

ELSIF(!LS_BR/ & !EX_BREQ/) THEN
DMA_REQ/ = GND;% When both req. are active, sonic has
  priority %
S_BG/ = LS_BR/;
S_DMA/ = GND;
massm = m3;

ELSIF(LS_BR/ & EX_BREQ/) THEN

DMA_REQ/ = GND;% When both req. are active, sonic has
  priority %
massm = m0;

END IF;


WHEN m3 =>
IF(!S_BGACK/) THEN

IO_ENABLE/ = GND;
S_BG/ = LS_BR/;
DMA_REQ/ = GND;
S_DMA/ = GND;
%SONIC_DELAY_pr = GND;%
massm = m4;
%ELSIF(!S_BGACK/ & (SONIC_DELAY[1..0] == B"00")) THEN
SONIC_DELAY[1..0] = SONIC_DELAY[1..0] + 1;
IO_ENABLE/ = GND;
S_BG/ = LS_BR/;
DMA_REQ/ = GND;
S_DMA/ = GND;
massm = m3;
ELSIF(!S_BGACK/ & (SONIC_DELAY[1..0] == B"01")) THEN
SONIC_DELAY[1..0] = SONIC_DELAY[1..0] + 1;
IO_ENABLE/ = GND;
S_BG/ = LS_BR/;
DMA_REQ/ = GND;
S_DMA/ = GND;
massm = m3;
ELSIF(!S_BGACK/ & (SONIC_DELAY[1..0] == B"10")) THEN
SONIC_DELAY[1..0] = SONIC_DELAY[1..0] + 1;
IO_ENABLE/ = GND;
```

```
S_BG/ = LS_BR/;
DMA_REQ/ = GND;
S_DMA/ = GND;
massm = m3;%

ELSE
DMA_REQ/ = GND;
S_BG/ = LS_BR/;
S_DMA/ = GND;
massm = m3;
END IF;

WHEN m4 =>

IF(!L_SMST_AS/) THEN
IO_ENABLE/ = GND;
DMA_REQ/ = GND;
S_DMA/ = GND;
massm = m6;

ELSIF(S_BGACK/) THEN
massm = m0;
S_DMA/ = GND;
ELSE
S_DMA/ = GND;
DMA_REQ/ = GND;
IO_ENABLE/ = GND;
massm = m4;

END IF;

WHEN m6 =>
%DMA_RD/ = !SMST_WR;
DMA_WR/= SMST_WR;%
IO_Syscmd_id[3..2]= B”10”;
IO_Syscmd_id1= VCC;
IO_Syscmd_id0= VCC;
IO_Syscmd_idn6 = !SMST_WR;
IOEX_OE1B/ = SMST_WR # !IO_ADDRn2;
IOEX_OE2B/ = SMST_WR # IO_ADDRn2 # !P3_SYS_IF;
IOEX_LEA1B= VCC;
IOEX_LEA2B= VCC;
IOEX_OEA/_id= !SMST_WR;
IO_EX_SEL_id= IO_ADDRn2;
IOEX_LE1B_id= VCC;
IOEX_LE2B_id= VCC;
IO_ENABLE/ = GND;
DMA_REQ/ = GND;
S_DMA/ = GND;
massm = m9;
WHEN m9 =>
DMA_RD/ = !SMST_WR;
DMA_WR/= SMST_WR;
IO_Syscmd_id[3..2]= B”10”;
IO_Syscmd_id1= VCC;
IO_Syscmd_id0= VCC;
IO_Syscmd_idn6 = !SMST_WR;
IOEX_OE1B/ = SMST_WR # !IO_ADDRn2;
IOEX_OE2B/ = SMST_WR # IO_ADDRn2 # !P3_SYS_IF;
IOEX_LEA1B= VCC;
IOEX_LEA2B= VCC;
IOEX_OEA/_id= !SMST_WR;
IO_EX_SEL_id= IO_ADDRn2;
IOEX_LE1B_id= VCC;
IOEX_LE2B_id= VCC;
IO_ENABLE/ = GND;
DMA_REQ/ = GND;
S_DMA/ = GND;
```

```
massm = m7;


WHEN m7 =>
IF(!LS_DSACK0/_id & !LS_DSACK1/_id) THEN

IOEX_OE1B/ = SMST_WR # !IO_ADDRn2;
IOEX_OE2B/ = SMST_WR # IO_ADDRn2 # !P3_SYS_IF;
IOEX_LEA1B= VCC;
IOEX_LEA2B= VCC;
IO_ENABLE/ = GND;
IOEX_OEA/_id= !SMST_WR;
IO_EX_SEL_id= IO_ADDRn2;
IOEX_LE1B_id= VCC;
IOEX_LE2B_id= VCC;

%DMA_RD/ = !SMST_WR;
DMA_WR/= SMST_WR;%
DMA_REQ/ = GND;
S_DMA/= GND;
massm = m8;
IO_Syscmd_id[3..2]= B"10";
IO_Syscmd_id1= VCC;
IO_Syscmd_id0= VCC;
IO_Syscmd_idn6 = !SMST_WR;


ELSE

IOEX_OE1B/ = SMST_WR # !IO_ADDRn2;
IOEX_OE2B/ = SMST_WR # IO_ADDRn2 # !P3_SYS_IF;
IOEX_LEA1B= VCC;
IOEX_LEA2B= VCC;
IOEX_OEA/_id= !SMST_WR;
IO_EX_SEL_id= IO_ADDRn2;
IOEX_LE1B_id= VCC;
IOEX_LE2B_id= VCC;
IO_ENABLE/ = GND;
DMA_REQ/ = GND;
DMA_RD/ = !SMST_WR;
DMA_WR/= SMST_WR;
S_DMA/ = GND;
IO_Syscmd_id[3..2]= B"10";
IO_Syscmd_id1= VCC;
IO_Syscmd_id0= VCC;
IO_Syscmd_idn6 = !SMST_WR;

massm = m7;

END IF;

WHEN m8 =>
IOEX_OE1B/ = SMST_WR # !IO_ADDRn2;
IOEX_OE2B/ = SMST_WR # IO_ADDRn2 # !P3_SYS_IF;
IOEX_LEA1B= VCC;
IOEX_LEA2B= VCC;
IOEX_OEA/_id= !SMST_WR;
IO_EX_SEL_id= IO_ADDRn2;
IOEX_LE1B_id= VCC;
IOEX_LE2B_id= VCC;
IO_ENABLE/ = GND;
DMA_REQ/ = GND;
S_DMA/ = GND;
massm = m4;


WHEN m10 =>
```

```
IF(!IO_RD/ & !DMA_ACK/) THEN

DMA_RD/= GND;
IO_Syscmd_id[3..2]= B”10”;
IO_Syscmd_id1= VCC;
IO_Syscmd_id0= VCC;
IO_Syscmd_idn6= IO_RD/;
IOEX_OE1B/ = IO_WR/ # !IO_ADDRn2;
IOEX_OE2B/ = IO_WR/ # IO_ADDRn2 # !P3_SYS_IF;
IOEX_LEA1B= VCC;
IOEX_LEA2B= VCC;
IOEX_OEA/_id= !IO_WR/;
IO_EX_SEL_id= IO_ADDRn2;
IOEX_LE1B_id= VCC;
IOEX_LE2B_id= VCC;
IO_ENABLE1 = VCC;
IO_ENABLE/ = GND;
DMA_REQ/= GND;
massm = m11;


ELSIF(!IO_WR/ & !DMA_ACK/) THEN
DMA_WR/= GND;
IO_Syscmd_id[3..2]= B”10”;
IO_Syscmd_id1= VCC;
IO_Syscmd_id0= VCC;
IO_Syscmd_idn6= IO_RD/;
IOEX_OE1B/ = IO_WR/ # !IO_ADDRn2;
IOEX_OE2B/ = IO_WR/ # IO_ADDRn2 # !P3_SYS_IF;
IOEX_LEA1B= VCC;
IOEX_LEA2B= VCC;
IOEX_OEA/_id= !IO_WR/;
IO_EX_SEL_id= IO_ADDRn2;
IOEX_LE1B_id= VCC;
IOEX_LE2B_id= VCC;
IO_ENABLE1 = VCC;
IO_ENABLE/ = GND;
DMA_REQ/= GND;
massm = m12;

ELSE
DMA_REQ/= GND;
IO_ENABLE1 = VCC;
IO_ENABLE/ = GND;
massm = m10;

END IF;


WHEN m11 =>

IO_Syscmd_id[3..2]= B”10”;
IO_Syscmd_id1= VCC;
IO_Syscmd_id0= VCC;
IO_Syscmd_idn6= IO_RD/;
IOEX_OE1B/ = IO_WR/ # !IO_ADDRn2;
IOEX_OE2B/ = IO_WR/ # IO_ADDRn2 # !P3_SYS_IF;
IOEX_LEA1B= VCC;
IOEX_LEA2B= VCC;
IOEX_OEA/_id= !IO_WR/;
IO_EX_SEL_id= IO_ADDRn2;
IOEX_LE1B_id= VCC;
IOEX_LE2B_id= VCC;
IO_ENABLE1 = VCC;
IO_ENABLE/ = GND;
DMA_REQ/= GND;
```

```
IF(!EX_ACK_NODE/) THEN

DMA_REQ/= GND;
IO_ACK3/_in= GND;
IO_ENABLE1 = VCC;
IO_ENABLE/ = GND;
massm = m14;


ELSE
IO_ENABLE1 = VCC;
IO_ENABLE/ = GND;
DMA_REQ/= GND;
massm = m11;

END IF;

WHEN m14 =>
IF(IO_RD/ & IO_WR/) THEN
IO_ENABLE1 = VCC;
IO_ENABLE/ = GND;
DMA_REQ/= GND;
massm = m15;

ELSE

DMA_REQ/= GND;
IO_ENABLE1 = VCC;
IO_ENABLE/ = GND;
massm = m14;

END IF;

WHEN m15 =>
DMA_REQ/= GND;
IO_ENABLE1 = VCC;
IO_ENABLE/ = GND;
massm = m2;
WHEN OTHERS =>

massm = m0;

END CASE;

%_____________________________________________________________________________%

CASE exsm IS

WHEN ex0 =>
IF(!SY_RD/ & (IO_ADDR[28..20] == H"1f7") &
(IO_Syscmd3 == B"1")) THEN


IO_Access/ = GND;
EX_UCS/ = GND;
IO_RD/ = GND;
exsm = ex1;

ELSIF(!SY_WR/ & (IO_ADDR[28..20] == H"1f7") &
(IO_Syscmd3 == B"1")) THEN


IO_Access/ = GND;
EX_UCS/ = GND;
IO_WR/ = GND;
exsm = ex1;

END IF;
```

```
WHEN ex1 =>
IF(!IO_ACK3/ ) THEN


IO_RD/ = IO_RD/;
IO_WR/ = IO_WR/;

IO_Access/ = GND;
EX_UCS/ = GND;
IO_DONE/ = GND;
exsm = ex2;

ELSIF (!IO_ERR/) THEN

IO_DONE/ = GND;
IO_RD/ = IO_RD/;
IO_WR/ = IO_WR/;
exsm = ex3;


ELSE


IO_RD/ = IO_RD/;
IO_WR/ = IO_WR/;

IO_Access/ = GND;
EX_UCS/ = GND;
exsm = ex1;

END IF;

WHEN ex2 =>

exsm = ex0;

WHEN ex3 =>

exsm = ex0;

WHEN OTHERS =>

exsm = ex0;



END CASE;

END;

%
```

SRAM Controller

MAX+PLUS II AHDLv5.0

EDIT HISTORY

Date	Engineer	Check Sum	Changes
10/10/1994	R.Cummings		Initial Release

%

SRAM Controller PLD

```
%============================================================================%
%The SRAM Controller is used for the zero wait state SRAM memory system. It controls all accesses to the
SRAM from the CPU or the I/O.
%

%========================== CONSTANTS ==================================%
CONSTANT 1MEG=B"001";
CONSTANT 4MEG=B"111";
CONSTANT _5MEG=B"000";
CONSTANT 2MEG=B"011";
CONSTANTBLOCK=B"1";
CONSTANTSINGLE=B"0";
%============================================================================%
SUBDESIGN SRAM
(
% Input signals from CPU%
SYSAD[28..19]:INPUT;% SYSAD28 = 0 for Memory access%
% SYSAD28 = 1 for IO accesses%
% SYSAD[27..18] used to determine %
% if access is for SRAM or DRAM  %
LSYSAD[4..3]:INPUT;% SYSAD[4..0] are used on all reads%
% for sub-block and for writes to %
% decode byte enable (single write)%
VALIDOUT/:INPUT;% Indicates CPU has valid SYSAD and%
% SYSCMD values driven%
RELEASE/:INPUT;% Indicates the CPU has released %
% the bus to the external agent%
WRRDY/:INPUT;% Need WrRdy* to determine if there %
% can be write issued in current cycle%
SYSCMD8:INPUT;% Indicates the type of CPU request%
SYSCMD6:INPUT;% Indicates the type of CPU request%
SYSCMD3:INPUT;% Indicates the type of CPU request%
B3_TCLK0_0:INPUT;% Buffered TClock from CPU.%
% Used as GLOBAL clock%

% Other input signals%
WAIT/:INPUT;% Form main controller to let SRAM%
% know an IO request is in progress%
% SRAM will delay SRAM_DONE/ to the%
% Main controller if WAIT/ is LOW%
DRAM_Access/:INPUT;% From DRAM controller to indicate%
% that a DRAM write access is in%
% progress to stop SRAM access%
SRAM_PRESENT/:INPUT;% Indicates if SRAM installed%
%SRAM_PRESENT/Description%
%0SRAM installed%
%1No SRAM %
SRAM_SIZE:INPUT;% If SRAM installed, indicates size%
%0 = 1MB(512K) 64Kx32 SIMMs%
%1 = 4MB (2MB) 256Kx32 SIMMs%
% value in () for 32-bit P3 sys I/F%
P3_SYS_IF:INPUT;% Indicates size of P3 sys I/F%
%P3_SYS_IFDescription%
%032-bit interface%
%164-bit interface%
RESETIN/:INPUT;% Global Reset signal%
IGNORE_Access/:INPUT;% Asserted by DRAM Controller during%
```

```
% refresh cycles to indicate that the%
% CPU will not issue any Writes until%
% refresh is done and WrRdy* asserted%
DRAM_PRESENT/:INPUT;% Indicates presence of DRAM%
%0DRAM installed%
%1No DRAM %
DMA_RD/:INPUT;% Indicates a valid DMA Read request%
% is on the bus%
DMA_WR/:INPUT;% Indicates a valid DMA Write request%
% is on the bus%

% SRAM control output signals %
SC_ADDR3E:OUTPUT;% For 32-bit P3 sysIF, provides LSB of%
% SRAM address - Even Bank addressing%
SC_ADDR3O:OUTPUT;% For 32-bit P3 sysIF, provides LSB of%
% SRAM address - Odd Bank addressing%
SC_ADDR4E:OUTPUT;% Provides LSB for 64-bit sysIF or next %
% LSB for 32-bit sysIF for 32-bit IF for%
% Even bank addressing%
SC_ADDR4O:OUTPUT;% Provides LSB for 64-bit sysIF or next %
% LSB for 32-bit sysIF for 32-bit IF for%
% Odd bank addressing%
SE_OE/:OUTPUT;% SRAM Even bank output enable%
SO_OE/:OUTPUT;% SRAM Odd bank output enable%
SRAM_Access/:OUTPUT;% Indicates if the current access is for%
% SRAM or other memory device%
% 0 = SRAM Access1 = Not SRAM access%
SRAM_DONE/:OUTPUT;% Indicates that the current SRAM access%
% will either complete or have partially%
% complete in the next cycle%
)

%================================================================================%
% SRAM control state machine %

VARIABLE

sramsm:MACHINE WITH STATES
(s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, s14, s15, s16,
 s17, s18, s19, s20, s21, s22, s23, s24, s25, s26, s27, s28, s29, s30, s31);

%---------------  Output Flip Flops  ----------------------%
SC_ADDR3E:DFF;
SC_ADDR3O:DFF;
SC_ADDR4E:DFF;
SC_ADDR4O:DFF;
SE_OE/:DFF;
SO_OE/:DFF;
SRAM_Access/:DFF;
SRAM_DONE/:DFF;
%-------------------------------------------------------------------------------%
% Buried nodes %
Count[2..0]:DFF;% Counter for block read data - 32-bit mode%
DMA_cnt[1..0]:DFF;% Counter for DMA reads and writes%
Wrrdy1/:DFF;% A one cycle delayed version of WrRdy* signal%
Wrrdy2/:DFF;% A two cycle delayed version of WrRdy* signal%
FIRST:DFF;% Indicates 1st (= 0) or 2nd (=1) X cycle of%
% block write for 32-bit access or for DMA read%
% or write to add time before DONE asserted%
Not_SRAMD_pre:NODE;% Used to keep SRAM_DONE/ HIGH for non SRAM %
% cycles and at start of DMA read or write%
Not_SRAM_pre:NODE;% Used to keep and SE_OE/ and SO_OE/ HIGH for%
% non-SRAM access%

%================================================================================%

BEGIN
```

```
%-------------------------------------------------------------------------------%
% Default state for all outputs is not asserted %
DEFAULTS
SC_ADDR3E=VCC;
SC_ADDR3O=VCC;
SC_ADDR4E=VCC;
SC_ADDR4O=VCC;
SE_OE/=VCC;
SO_OE/=VCC;
SRAM_Access/=VCC;
SRAM_DONE/=VCC;
Count[2..0]=GND;
DMA_cnt[1..0]=GND;
FIRST=GND;
Not_SRAM_pre=VCC;
Not_SRAMD_pre=VCC;
END DEFAULTS;

%----------------- Output DFF clock and clear assignments  -------------------------%

SC_ADDR3E.clk=GLOBAL(B3_TCLK0_0);
SC_ADDR3E.clrn=GLOBAL(RESETIN/);

SC_ADDR3O.clk=GLOBAL(B3_TCLK0_0);
SC_ADDR3O.clrn=GLOBAL(RESETIN/);

SC_ADDR4E.clk=GLOBAL(B3_TCLK0_0);
SC_ADDR4E.clrn=GLOBAL(RESETIN/);

SC_ADDR4O.clk=GLOBAL(B3_TCLK0_0);
SC_ADDR4O.clrn=GLOBAL(RESETIN/);

SE_OE/.clk=GLOBAL(B3_TCLK0_0);
SE_OE/.prn=GLOBAL(RESETIN/) & Not_SRAM_pre;

SO_OE/.clk=GLOBAL(B3_TCLK0_0);
SO_OE/.prn=GLOBAL(RESETIN/) & Not_SRAM_pre;

SRAM_Access/.clk=GLOBAL(B3_TCLK0_0);
SRAM_Access/.prn=GLOBAL(RESETIN/);

SRAM_DONE/.clk=GLOBAL(B3_TCLK0_0);
SRAM_DONE/.prn=GLOBAL(RESETIN/) & Not_SRAMD_pre;

FIRST.clk=GLOBAL(B3_TCLK0_0);
FIRST.clrn=GLOBAL(RESETIN/);

%--------- State machine clock and clear ----------------------------------------%

sramsm.clk=GLOBAL(B3_TCLK0_0);
sramsm.reset=GLOBAL(!RESETIN/);

%---------- Buried nodes clock and clear ----------------------------------------%
Count[2..0].clk=GLOBAL(B3_TCLK0_0);
Count[2..0].clrn=GLOBAL(RESETIN/);

DMA_cnt[1..0].clk=GLOBAL(B3_TCLK0_0);
DMA_cnt[1..0].clrn=GLOBAL(RESETIN/);

Wrrdy1/.clk=GLOBAL(B3_TCLK0_0);
Wrrdy1/.clrn=GLOBAL(RESETIN/);

Wrrdy2/.clk=GLOBAL(B3_TCLK0_0);
Wrrdy2/.clrn=GLOBAL(RESETIN/);

%----------- Equations ------------------------------------------------------%

Wrrdy1/.d = WRRDY/;
```

```
Wrrdy2/.d = Wrrdy1/;

%-------------------------------------------------------------------------%

% state machine design %

CASE sramsm IS

WHEN s0 =>% SRAM IDLE %
SC_ADDR4E = LSYSAD4;
SC_ADDR4O = LSYSAD4;
SC_ADDR3E = LSYSAD3;
SC_ADDR3O = LSYSAD3;
SO_OE/ = !(WAIT/ & DRAM_Access/ & (!SYSCMD6 # !DMA_RD/));
SE_OE/ = !(WAIT/ & DRAM_Access/ & (!SYSCMD6 # !DMA_RD/));
SRAM_DONE/ = GND;
Not_SRAMD_pre = DMA_RD/ & DMA_WR/ & WAIT/ & DRAM_Access/ &
 !(SYSCMD6 & !SYSCMD3);

IF (!SYSAD28 & P3_SYS_IF &
(SYSAD[27..22] == B"000000") & !SRAM_PRESENT/ &
(((!SYSCMD8 & !SYSCMD6 & SYSCMD3) & !RELEASE/ & !VALIDOUT/) #
!DMA_RD/)) THEN
% Single read issue or DMA read%
SRAM_Access/ = GND;
sramsm = s1;
ELSIF (!SYSAD28 & P3_SYS_IF & !RELEASE/ & !VALIDOUT/ &
(SYSAD[27..22] == B"000000") & !SRAM_PRESENT/ &
(!SYSCMD8 & !SYSCMD6 & !SYSCMD3)) THEN
% Block read issue%
SRAM_Access/ = GND;
 sramsm = s3;
ELSIF (!SYSAD28 & P3_SYS_IF & !SRAM_PRESENT/ &
(SYSAD[27..22] == B"000000") &
(((!SYSCMD8 & SYSCMD6 & SYSCMD3) & IGNORE_Access/ &
!VALIDOUT/ & WAIT/ & DRAM_Access/ & !Wrrdy2/) #
!DMA_WR/)) THEN
% Single write issue with No other writes or DMA write%
SRAM_Access/ = GND;
sramsm = s5;
ELSIF ( !SYSAD28 & P3_SYS_IF & !VALIDOUT/ & ( !WAIT/ # !DRAM_Access/ ) &
(SYSAD[27..22] == B"000000") & ( !SRAM_PRESENT/ ) & !Wrrdy2/ &
(!SYSCMD8 & SYSCMD6 & SYSCMD3) & IGNORE_Access/) THEN
% Single write issue with other writes%
SRAM_Access/ = GND;
sramsm = s6;
ELSIF ( !SYSAD28 & P3_SYS_IF & !VALIDOUT/ &
(SYSAD[27..22] == B"000000") & ( !SRAM_PRESENT/ ) & !Wrrdy2/ &
(!SYSCMD8 & SYSCMD6 & !SYSCMD3) & IGNORE_Access/) THEN
% Block write issue %
Count[2..0] = B"000";
SRAM_Access/ = GND;
sramsm = s7;
ELSE
Not_SRAMD_pre = GND;
Not_SRAM_pre = GND;
sramsm = s0;
END IF;

WHEN s1 =>
SO_OE/ = !(WAIT/ & DRAM_Access/);
SE_OE/ = !(WAIT/ & DRAM_Access/);
SC_ADDR4E = SC_ADDR4E;
SC_ADDR4O = SC_ADDR4O;
SRAM_Access/ = GND;

IF (WAIT/ & DRAM_Access/ & !SRAM_DONE/) THEN
```

```
% Single read bus turn around cycle%
sramsm = s2;
ELSIF (WAIT/ & DRAM_Access/ & SRAM_DONE/ & DMA_RD/) THEN
% psuedo read bus turn around cycle because of WAIT/ or DRAM_Access%
 SRAM_DONE/ = GND;
sramsm = s1;
ELSIF (!DMA_RD/) THEN
% DMA Read cycle%
sramsm = s2;
ELSE
% If WAIT/ or DRAM_Access/ is LOW stay here until ready%
sramsm = s1;
END IF;
WHEN s2 =>
SC_ADDR4E = SC_ADDR4E;
SC_ADDR4O = SC_ADDR4O;

IF (!DMA_RD/) THEN
% Single DMA read data cycle%
SRAM_Access/ = GND;
DMA_cnt[1..0] = B"00";
sramsm = s8;
ELSE
% Single read data cycle to CPU%
SRAM_Access/ = VCC;
sramsm = s0;
END IF;

WHEN s3 =>
SO_OE/ = !(WAIT/ & DRAM_Access/);
SE_OE/ = !(WAIT/ & DRAM_Access/);
SC_ADDR4E = SC_ADDR4E;
SC_ADDR4O = SC_ADDR4O;
SRAM_Access/ = GND;

IF (WAIT/ & DRAM_Access/ & !SRAM_DONE/) THEN
% Block read bus turn around cycle%
SRAM_DONE/ = GND;
Count[2..0] = B"000";
sramsm = s4;
ELSIF (WAIT/ & DRAM_Access/ & SRAM_DONE/) THEN
% WAIT/ or DRAM_Access were LOW but not any more and this
  will act like the bus turn around cycle%
SRAM_DONE/ = GND;
sramsm = s3;
ELSE
% If WAIT/ or DRAM_Access/ is LOW stay here until ready%
sramsm = s3;
END IF;
WHEN s4 =>
SRAM_Access/ = GND;
SO_OE/ = GND;
SE_OE/ = GND;
SC_ADDR4E = (SC_ADDR4E $ (Count[2..0] == B"000"));
SC_ADDR4O = (SC_ADDR4O $ (Count[2..0] == B"000"));
SRAM_DONE/ = GND;

IF (Count[2..0] == B"000") THEN
% Block read data 0 cycle%
Count[2..0] = B"001";
sramsm = s4;
ELSIF (Count[2..0] == B"001") THEN
% Block read data 1 cycle%
Count[2..0] = B"010";
sramsm = s4;
ELSIF (Count[2..0] == B"010") THEN
% Block read data 2 cycle%
Count[2..0] = B"011";
```