AlphaStation 600 Series

Technical Reference Information

Part Number: EK-AS800-RM. A01

July 1995

The purpose of this manual is to provide programming information that will assist system programmers in writing AlphaStation 600 System support code for their operating system.

Revision Information: This is a new manual.

Digital Equipment Corporation
Maynard, Massachusetts

July 1995

Digital Equipment Corporation makes no representation that the use of its products in the manner described in this publication will not infringe on existing or future rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

Possession, use, or copying of the software described in this publication is authorized only pursuant to a valid written license from Digital or an authorized sublicensor.

© Digital Equipment Corporation 1995. All rights reserved.

The postpaid Reader's Comments form at the end of this document requests your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation: Alpha, AlphaStation and the DIGITAL logo.

The following are third-party trademarks:

Microsoft and Windows NT are registered trademarks of Microsoft Corporation.

SIMM is a trademark of Molex Corporation.

OSF/1 is a trademark of the Open Software Foundation, Inc.

All other trademarks and registered trademarks are the property of their respective holders.

[S2739]

Contents

Preface

Introduction xv
Document Contents xv
Conventions xvi
Abbreviations xvi

1 AlphaStation 600 System Overview

Introduction 1-1
ASIC Summary 1-5
Hardware Jumpers 1-5
.... 1-6

2 AlphaStation 600 ASIC Overview

Introduction 2-1
The CIA ASIC 2-2
EV5 Instruction and Address Region 2-2
PCI Datapath 2-3
Memory Logic 2-4
I/O Address Logic 2-6
DMA Address Logic 2-6
The DSW ASIC 2-6
Flash ROM 2-7
CPU Memory Read 2-8
CPU Memory Read with Victim 2-9
CPU I/O Read 2-9
CPU I/O Write 2-10
DMA Transactions 2-11
DMA Read 2-12
DMA Read Prefetching 2-12
DMA Write 2-13
MB Instruction 2-13
LOCK Instructions 2-14
Locks to Uncached Space 2-15
GRU ASIC 2-15
IOD Interface 2-15
GRU Addressing 2-16
Flash ROM Interface 2-17

Interrupt Logic 2-18
Configuration Registers for Cache and Memory 2-19
Reset Logic 2-19
AlphaStation 600 PCI-EISA Bridge 2-20
ESC functionality 2-21
PCEB Functionality 2-21
PCI-to-EISA Address Decode 2-21
PC Compatibility Addressing and Holes 2-21
MEMCS# 2-22
PCI Arbitration 2-22
PCI Arbitration - Power-Up 2-22
Data Buffering in the PCEB 2-23

3 AlphaStation 600 Addressing

Introduction 3-1
Address Mapping Introduction 3-1
PCI Addressing 3-2
CPU Address Space 3-2
CPU Address <38:35> 3-3
Cacheable Memory Space 3-7
PCI Sparse Memory Space 3-8
Hardware Extension Registers (HAE) 3-12
PCI Sparse I/O Space 3-13
PCI Configuration Space 3-15
PCI Special/Interrupt Cycles 3-18
Hardware Specific and Miscellaneous Register Space 3-19
PCI to Physical Memory Addressing 3-19
Direct-mapped Addressing 3-22
Scatter/Gather Addressing 3-23
Scatter/Gather TLB 3-24
PCI Window Suggested Use 3-26
PC Compatibility Addressing and Holes 3-27
MEMCS# 3-27

4 Modules

Memory MotherBoard 4-1
Presence Detect Bits 4-2
Cache SIMM 4-3
I/O Subsystem Module 4-4
Major Components 4-4

5 Power Up and Reset

Introduction 5-1
Power Up 5-1
Halt and Reset 5-1

6 AlphaStation 600 Physical Partitioning

Introduction 6-1
Hardware Jumpers 6-1
Fan Fail Detect Jumpers 6-1
Flash ROM Write Jumper 6-2
Alternate Console Jumper 6-2
Secure Console Jumper 6-2
SROM Code Select Jumper 6-2
EV5 clock multiple DIP switch 6-3
Physical Organization 6-3
I/O Subsystem Organization 6-10
AlphaStation 600 Module Overview 6-11
System Board 6-11
Memory Motherboard 6-11
Cache SIMM 6-12
PCI Options 6-12
EISA Options 6-13
AlphaStation 600 SystemBoard - ASICs 6-13
EV5 CPU 6-13
CIA ASIC 6-30
AlphaStation 600 I/O & External Interconnect 6-38
Serial Ports 6-38
Parallel Port 6-38
Keyboard/Mouse Connectors 6-39
PCI Connector 6-40
EISA Connector 6-42
Power Connectors 6-44
Fan Connectors 6-45
OCP Connector 6-45
Floppy Connector 6-48
Serial ROM Connector 6-49
System Testability Features 6-50

7 Control and Status Registers

Register Types 7-1
Register Addressing 7-1
General Registers 7-2
.... 7-3
Memory Control Registers 7-3
PCI Address-related Registers 7-3
Scatter/Gather Address Translation Registers 7-4
Flash ROM Space 7-5
Addressing 7-5
EV5 Configuration Registers 7-7
EV5 Registers 7-7
General CIA Registers - Description 7-8
CIA Revison Register (CIA_REV) 7-8
PCI Latency Register (PCI_LAT) 7-9
CIA Control Register (CIA_CTRL) 7-10
Hardware Address Extension Register (HAE_MEM) 7-13
Hardware Address Extension Register (HAE_IO) 7-14
Configuration Type Register (CFG) 7-15
CIA Acknowledgement Control Register (CACK_EN) 7-16
Diagnostic Registers - description 7-17

CIA Diagnostic Control Register (CIA_DIAG) 7-17
Diagnostic Check Register (DIAG_CHECK) 7-18
CIA Performance Monitor Register (PERF_MONITOR) 7-19
CIA Performance Control Register (PERF_CONTROL) 7-20
CPU Error Information Register 0 (CPU_ERR0) 7-22
CPU Error Information Register 1 (CPU_ERR1) 7-23
CIA Error Register (CIA_ERR)... 7-24
CIA Status Register (CIA_STAT) 7-27
CIA Error Mask Register (ERR_MASK) 7-29
CIA Error Syndrome Register (CIA_SYN) 7-30
CIA Memory Port Status Register 0 (MEM_ERR0) 7-31
CIA Memory Port Status Register 1 (MEM_ERR1) 7-32
PCI Error Register 0 (PCI_ERR0) 7-35
PCI Error Register 1 (PCI_ERR1) 7-37
PCI Error Register 2 (PCI_ERR2) 7-38
Memory Control Registers - description 7-39
Memory Configuration Register (MCR) 7-39
Memory Base Address Registers 0-E (MBA0-E) 7-41
Memory Timing Registers (TMG0-TMG2) 7-43
.... 7-44
PCI Address-related Registers - Description.. 7-45
Scatter/Gather Translation Buffer Invalidate Register (TBIA)........ 7-45
Window Base Registers (Wx_BASE, x=0-3) 7-46
Window Mask Registers (Wx_MASK, x=0-3) 7-48
Translated Base Registers (Tx_BASE, x=0-3) 7-49
Window DAC Base Register (W_DAC) 7-52
Scatter/Gather Address Translation Registers 7-53
Lockable Translation Buffer Tag Registers (LTB_TAG0 - LTB_TAG3) 7-53
Translation Buffer Tag Registers (TB_TAG0 - TB_TAG3) 7-54
Translation Buffer Page Register (TBx_PAGEn) 7-55
GRU ASIC - related Registers 7-56
Interrupt Mask Register (INT_MASK) 7-57
Interrupt Level/Edge Select Register (INT_EDGE) 7-58
Interrupt Clear Register (INT_CLEAR).. 7-59
Cache & Memory Configuration Register (CACHE_CNFG) 7-60
SET Configuration Register (SCR) 7-62
LED Register (LED) 7-63
Reset Register (RESET) 7-64
EV5 Configuration Registers - description 7-65
Scache Control Register, SC_CTL 7-65
Bcache Control Register, BC_CONTROL 7-66
Bcache Configuration Register, (BC_CONFIG) 7-69

8 Hardware Exceptions and Interrupts

Introduction 8-1
System Interrupts 8-1
SYS_MCH_CHK_IRQ 8-4
Halt/Reset Switches 8-5
EV5 Error Handling 8-6
PCI Error Handling 8-6
PERR# Implications 8-7
SERR# implications 8-7
AlphaStation 600 Error Handling 8-10
CIA ASIC Error Registers 8-12
CIA Error Mask Register 8-15

CIA Error Reporting 8-16
CIA Detected Errors 8-16
Error Insertion 8-21
Force Error Registers 8-21
Accessing Main Memory Via CPU Uncached Space 8-21
Writing Bad ECC Into Memory 8-21
Machine Check Logout Data Structure 8-22
Correctable Error Logout Frame 8-23
Deciphering a Correctable Error Machine Check Logout Frame 8-25
ECC Syndromes for Single-Bit Errors 8-27
Uncorrectable Error Logout Frame 8-28
Deciphering an Uncorrectable Error Logout Frame 8-35

9 AlphaStation 600 System Initialization

Introduction 9-1
Serial ROM Performed Initialization 9-1
System Firmware Performed Initialization 9-3
Overview of Sizing the PCI Bus 9-4
Sizing/Configuring the PCI Bus 9-5
Accessing PCI Bus Configuration 9-5
PCI-PCI Bridge Configuration 9-5
Driver Initiated PCI Configuration 9-6
PCI-EISA Bridge Configuration 9-7
COM1 (87312) Device Initialization 9-8
COM2 (87312) Device Initialization 9-8
Parallel Port (87312) Device Initialization 9-8
Floppy Controller (87312) Device Initialization 9-8
Keyboard/Mouse Device Initialization 9-8
Battery Backed SRAM Device Initialization 9-8
TOY Driver Device Initialization 9-9
NCR810 Driver Device Initialization 9-9
TULIP Driver Device Initialization 9-9
TGA Driver Device Initialization 9-9
Memory Initialization 9-10
Miscellaneous CIA Related Initialization 9-11
Address Map 9-13
ISA Devices Address Map 9-13
Software Considerations 9-14

10 AlphaStation 600 PCI-EISA Bridge

Introduction 10-1
Related Documentation 10-1
ESC Functionality 10-6
ESC Registers 10-6
PCEB Functionality 10-8
PCI-to-EISA Address Decode 10-9
PC Compatibility Addressing and Holes 10-9
MEMCS# Details 10-11
PCI Arbitration 10-13
PCI Arbitration - Power-up 10-13
Potential for PCI-EISA Bridge Starvation 10-13
Coherency Implications 10-14
Coherency: Posted Write Buffer in the PCI Device 10-15

PCI Deadlock Avoidance Rule 10-16
Coherency: CIA and FLSHREQ# 10-16
Guaranteed Access Time Mode 10-17
Gat-Mode Software Notes 10-18
Data Buffering in the PCEB 10-18

11 System Coherency

Introduction 11-1
Referenced Documents 11-1
Alpha System Reference Manual 11-1
PCI Local Bus Specification 11-1
NCR 53C820 11-1
Other Devices 11-1
Coherency Summary 11-1
Classification of Coherency Situations 11-2
Generic Event Types in the AlphaStation 600 System 11-2
Possible Write - Read and Write - Write interactions 11-4
Basic Properties of the AlphaStation 600 System 11-5
Analysis of Interactions with Writes 11-6
Data fails to make it to a later read 11-6
Data overwrites some later write. 11-12
Data is overwritten before readers are finished with it 11-19
Data fails to make it to destination 11-19
Side Effects happen out of order 11-21
Analysis of I/O Page Table Modification Interactions 11-22
Failure to use the latest I/O Page Table State 11-22
I/O Page Table Changed While In Use 11-24
Triangle Inequality for I/O Busses 11-25

A CIA - DSW Command Fields (CMC and IOC)

Introduction A-1

Figures

Figure 1-1 The AlphaStation 600 System Block Diagram 1-3
Figure 2-1 System Block Diagram Showing CIA and DSW 2-1
Figure 2-2 The CIA Block Diagram 2-2
Figure 2-3 Memory Address Swizzling 2-4
Figure 2-4 Victim Aliasing 2-5
Figure 2-5 AlphaStation 600 System Memory Addressing 2-5
Figure 2-6 The Data Switch Block Diagram 2-8
Figure 2-7 Scatter/Gather TLB 2-12
Figure 2-8 Flash ROM Address scheme 2-18
Figure 2-9 GRU Interrupt Logic 2-18
Figure 2-10 Cache and Memory Configuration Register 2-19
Figure 2-11 SET Configuration[1] 2-19
Figure 2-12 AlphaStation 600 System Standard I/O Busses 2-20
Figure 2-13 AlphaStation 600 System PCI arbiter scheme 2-22
Figure 3-1 Address Space Overview 3-1
Figure 3-2 CPU and DMA Reads and Writes 3-4

Figure 3-3 CPU Addressing ... 3-4
Figure 3-4 Dense Space Address Generation ... 3-8
Figure 3-5 CI Memory Sparse Space Address Generation - Region 1 ... 3-11
Figure 3-6 PCI Memory Sparse Space Address Generation - Region 2 ... 3-11
Figure 3-7 PCI Memory Sparse Space Address Generation - Region 3 ... 3-11
Figure 3-8 AlphaStation 600 System PCI and (E)ISA I/O Map ... 3-13
Figure 3-9 PCI sparse I/O Space Address Translation ... 3-15
Figure 3-10 PCI Configuration Space Definition ... 3-16
Figure 3-11 PCI Configuration Space Read/Write Encodings ... 3-17
Figure 3-12 AlphaStation 600 System PCI Bus Hierarchy ... 3-18
Figure 3-13 PCI DMA Addressing Example ... 3-21
Figure 3-14 PCI Target Window Compare ... 3-21
Figure 3-15 Direct-mapped Translation ... 3-23
Figure 3-16 Scatter/Gather PTE Format ... 3-24
Figure 3-17 Scatter/Gather Associative TLB ... 3-25
Figure 3-18 Scatter/Gather Map Translation ... 3-26
Figure 3-19 Default PCI Window Allocation ... 3-27
Figure 3-20 MEMCS# Decode Area ... 3-28
Figure 3-21 MEMCS# Logic ... 3-28
Figure 4-1 MMB Layout ... 4-1
Figure 4-2 SIMM Population Order ... 4-2
Figure 4-3 I/O Subsystem Module Block Diagram ... 4-5
Figure 6-1 AlphaStation 600 System Block Diagram ... 6-5
Figure 6-2 AlphaStation 600 System Board Layout ... 6-6
Figure 6-3 AlphaStation 600 System Board Function Map ... 6-7
Figure 6-4 Memory Data Mapping ... 6-9
Figure 6-5 EV5 CPU Package - Top View ... 6-14
Figure 6-6 DSW Pinout - Top View ... 6-21
Figure 6-7 GRU Pinout - Top View ... 6-26
Figure 6-8 CIA Pinout - Top View ... 6-31
Figure 6-9 Serial Port Connector ... 6-38
Figure 6-10 Parallel Port Connector ... 6-39
Figure 6-11 Keyboard/Mouse ... 6-39
Figure 6-12 Control Power Connector Pinout ... 6-44
Figure 6-13 OCP Connector Pinout ... 6-47
Figure 6-14 Floppy Connector Pinout ... 6-48
Figure 6-15 SROM Port Connector Pinout ... 6-49
Figure 7-1 GRU Interrupt Logic ... 7-57
Figure 8-1 The AlphaStation 600 System Interrupt Scheme ... 8-2
Figure 8-2 AlphaStation 600 Error Logic ... 8-10
Figure 8-3 Possible Errors ... 8-11
Figure 8-4 Correctable Error Machine Check Logout Frame ... 8-23
Figure 8-5 AlphaStation 600 Specific Error Information ... 8-29
Figure 9-1 AlphaStation 600 Memory Map After Initialization ... 9-11
Figure 10-1 AlphaStation 600 Standard I/O Busses ... 10-2
Figure 10-2 EISA Access to PCI and Memory ... 10-9
Figure 10-3 PCI-EISA Bridge: EISA Address Decode ... 10-10
Figure 10-4 PCI-EISA and CIA Hole Example ... 10-11
Figure 10-5 MEMCS# Decode Area ... 10-12

Figure 10-6 MEMCS# Logic 10-12
Figure 10-7 AlphaStation 600 PCI Arbiter Scheme 10-13
Figure 10-8 EISA Deadlock Example 10-14
Figure 10-9 Interacting Deadlock Example 10-16

Tables

Table 1-1 The AlphaStation 600 System I/O Summary 1-4
Table 2-1 CIA PCI Commands 2-3
Table 2-2 AlphaStation 600 Series Prefetch Strategy 2-13
Table 2-3 GRU Address Space 2-17
Table 2-4 Round-robin PCI Arbitration 2-22
Table 3-1 AlphaStation 600 Series CPU Address Space 3-5
Table 3-2 AlphaStation 600 Series Address Map 3-6
Table 3-3 PCI Memory Sparse Space Read/Write Encodings 3-10
Table 3-4 High-order Sparse Space bits 3-10
Table 3-5 PCI Sparse I/O Space Read/Write Encodings 3-14
Table 3-6 Primary 64-bit PCI Slot to IDSEL Mapping 3-18
Table 3-7 Hardware Specific Register Address Map 3-19
Table 3-8 PCI Target Window MASK Register 3-20
Table 3-9 Direct-mapped PCI Target Address Translation 3-22
Table 3-10 Scatter/Gather Mapped PCI Target Address Translation. 3-24
Table 3-11 PCI Window POST Configuration 3-27
Table 4-1 SIMM Speed 4-3
Table 4-2 Cache Speed Encodings 4-3
Table 4-3 Cache Size Encodings 4-4
Table 6-1 AlphaStation 600 System Busses 6-1
Table 6-2 Data Word / Bit Range Map 6-8
Table 6-3 AlphaStation 600 Interconnect Reference 6-11
Table 6-4 AlphaStation 600 MMB Feature Comparison 6-12
Table 6-5 BCache Features 6-12
Table 6-6 PCI Slot Assignments 6-13
Table 6-7 EV5 PIN OUT - Sorted by Pin Number 6-15
Table 6-8 EV5 PINS - Alphabetic Order 6-18
Table 6-9 DSW Features 6-21
Table 6-10 DSW PIN OUT - Sorted by Pin Number 6-22
Table 6-11 DSW PIN OUT - Sorted Alphabetically 6-24
Table 6-12 GRU Features 6-26
Table 6-13 GRU PIN OUT - Sorted by Pin Number 6-27
Table 6-14 GRU PINS-Sorted Alphabetically 6-28
Table 6-15 CIA Features 6-30
Table 6-16 CIA PIN OUT - Sorted by Pin Number 6-32
Table 6-17 CIA PIN OUT - Sorted Alphabetically 6-35
Table 6-18 PCI Pin Out - Sorted by pin number 6-40
Table 6-19 PCI Pin Out - Sorted by Signal Name 6-41
Table 6-20 EISA Pin Out - Sorted by Pin Number 6-42
Table 6-21 EISA Pin Out Sorted by Signal Name 6-43
Table 6-22 Control Power Connector Pinout 6-45
Table 6-23 OCP Connector Pinout 6-47

Table 6-24 Floppy Connector Pinout (Signals only) 6-48
Table 6-25 SROM Port Connector Pinout 6-49
Table 6-26 ASIC Test Mode Settings 6-50
Table 7-1 AlphaStation 600 Register Categories 7-1
Table 7-2 Hardware Specific Register Address Map 7-1
Table 7-3 General CIA CSRs (Base = 87.4000.0000 Hex) 7-2
Table 7-4 Diagnostic Registers (Base = 87.4000.0000 Hex) 7-2
Table 7-5 Performance Monitoring Registers (Base = 87.4000.0000 Hex) 7-2
Table 7-6 Error Registers (Base = 87.4000.0000 Hex) 7-2
Table 7-7 System Configuration Registers 7-3
Table 7-8 PCI Address and Scatter/Gather Registers 7-3
Table 7-9 Address Translation Registers 7-4
Table 7-10 GRU Space - (Base Address = 87.8000.0000 Hex) 7-6
Table 7-11 EV5 System Specific Registers 7-7
Table 7-12 CIA Revision Register (CIA_REV) 7-8
Table 7-13 CIA Configuration Register (CIA_CNFG) 7-9
Table 7-14 CIA Control Register (CIA_CTRL) 7-11
Table 7-15 PCI READ Prefetch Algorithm 7-13
Table 7-16 High-order Sparse Space Bits 7-13
Table 7-17 Hardware Address Extension Register (HAE_MEM) 7-13
Table 7-18 Hardware Address Extension Register (HAE_IO) 7-14
Table 7-19 CFG Register 7-15
Table 7-20 CIA Acknowledgement Control Register (CACK_EN) 7-16
Table 7-21 CIA Diagnostic Control Register (CIA_DIAG) 7-17
Table 7-22 Diagnostic Check Register (DIAG_CHECK) 7-18
Table 7-23 CIA Performance Monitor Register (PERF_MONITOR) 7-19
Table 7-24 CIA Performance Control Register (PERF_CONTROL) 7-20
Table 7-25 PERF_CONTROL Register low/high_selects Encoding 7-21
Table 7-26 CPU Error Information Register 0 (CPU_ERR0) 7-22
Table 7-27 CPU Error Information Register 1 (CPU_ERR1) 7-23
Table 7-28 CIA_Error Register (CIA_ERR) 7-25
Table 7-29 CIA Status Register (CIA_STAT) 7-28
Table 7-30 CIA Error Mask Register (ERR_MASK) 7-29
Table 7-31 CIA Error Syndrome Register (CIA_SYN) 7-30
Table 7-32 CIA Memory Port Status Register 0 (MEM_ERR0) 7-31
Table 7-33 CIA Memory Port Status Register 1 (MEM_ERR1) 7-32
Table 7-34 MEM_PORT_CMD Encodings 7-33
Table 7-35 SEQ_ST Encodings 7-33
Table 7-36 SET_SEL_ENC Encodings 7-34
Table 7-37 PCI Error Register 0 (PCI_ERR0) 7-36
Table 7-38 PCI Error Register 1 (PCI_ERR1) 7-37
Table 7-39 PCI Error Register 2 (PCI_ERR2) 7-38
Table 7-40 Memory Configuration Register (MCR) 7-40
Table 7-41 Memory Base Address Registers 2,4,6,8,A,C,E 7-42
Table 7-42 Memory Timing Parameters, Encoding Values 7-43
Table 7-43 Memory Timing Registers (TMG0-TMG2) 7-44
Table 7-44 Scatter/Gather Translation Buffer Invalidate Register (TBIA) 7-45
Table 7-45 Window Base Registers (Wx_BASE, x=0-3) 7-47
Table 7-46 Window Mask Registers (Wx_MASK, x=0-3) 7-49

Table 7-47 Translated Base Registers (Tx_BASE, x=0-3) 7-50
Table 7-48 PCI Address Translation - Scatter/Gather Mapping Disabled 7-51
Table 7-49 PCI Address Translation - Scatter/Gather Mapping Enabled 7-51
Table 7-50 Window DAC Base Register (W_DAC) 7-52
Table 7-51 Lockable Translation Buffer Tag Registers 7-53
Table 7-52 Translation Buffer TAG Registers (TB_TAG0 - TB_TAG3) 7-54
Table 7-53 Translation Buffer Data Register (TBx_PAGEn) 7-55
Table 7-54 INT_REQ Register 7-56
Table 7-55 Main Interrupt Logic IRQ Assignment 7-56
Table 7-56 INT_MASK Register 7-57
Table 7-57 INT_EDGE Register 7-58
Table 7-58 INT_HILO Register 7-58
Table 7-59 INT_CLEAR Register 7-59
Table 7-60 MMB and Cache Configuration Register (CACHE_CNFG) 7-61
Table 7-61 Cache Speed 7-61
Table 7-62 Cache Size 7-61
Table 7-63 MMB Configuration 7-62
Table 7-64 SIMM PD Speed Select Pins 7-62
Table 7-65 SET Configuration Register (SCR) 7-63
Table 7-66 LED Register 7-63
Table 7-67 RESET Register 7-64
Table 7-68 SC_CTL Field Descriptions 7-65
Table 7-69 BC_CONTROL Field Descriptions 7-66
Table 7-70 BC_TAG_STAT FIeld Descrlptions 7-68
Table 7-71 BC_CONFIG FIeld Descrlptions 7-69
Table 7-72 BC_SIZE FIeld Descrlptions 7-71
Table 8-1 EV5 Interrupt Assignment 8-2
Table 8-2 Main Interrupt Logic IRQ Pin Assignment 8-3
Table 8-3 EISA Interrupt Assignment 8-4
Table 8-4 ESC NMI Generation 8-5
Table 8-5 EV5 Error Detection Features 8-6
Table 8-6 AlphaStation 600 Handling of PCI Data Parity Errors 8-7
Table 8-7 CIA Error Registers 8-12
Table 8-8 CIA_ERR Register 8-13
Table 8-9 CIA_ERR Register Fault Indication 8-14
Table 8-10 CIA Error Mask Register 8-15
Table 8-11 Error Reporting to EV5 8-16
Table 8-12 DMA Read Associated Errors 8-17
Table 8-13 DMA Write Associated Errors 8-18
Table 8-14 I/O Write and Special Cycle Errors 8-19
Table 8-15 CPU I/O Read and PCI Interrupt-ACK Errors 8-20
Table 8-16 CPU/Memory Read Associated Errors 8-21
Table 8-17 CPU/Memory Victim/Write Associated Errors 8-21
Table 8-18 Correctable Machine Check Error Codes 8-23
Table 8-19 EV5 Single Bit Error Syndromes 8-27
Table 8-20 Uncorrectable Machine Check Error Codes 8-30
Table 9-1 Summary of SROM Initialization 9-3
Table 9-2 PCI-PCI Bridge Initialization 9-6
Table 9-3 PCI-EISA Bridge Chip Initialization 9-7

Table 9-4 CIA Main CSR Register Initialization 9-12
Table 9-5 CIA Memory CSR initialization 9-12
Table 9-6 CIA Physical Address Translation CSR Initialization 9-12
Table 9-7 CIA Error CSR initialization 9-13
Table 9-8 ISA device Address Map (Sparse IO Space) 9-13
Table 10-1 Documentation 10-1
Table 10-2 ESC chip -- AlphaStation 600 System Requirements 10-3
Table 10-3 PCEB chip - AlphaStation 600 Requirements 10-5
Table 10-4 ESC Registers 10-7
Table 10-5 Round-Robin PCI Arbitration 10-13
Table 10-6 AlphaStation 600 GAT Latency Delay 10-18

Preface

Introduction

The AlphaStation 600 Series Technical Reference Information is provided to assist programmers writng operating system support code. It describes the AlphaStation 600 System design from the system block diagram level, down to discusssions relating to individual registers, software/firmware design information, and detailed major physical component layout designs, and signal paths.

Document Contents

The contents of this documents are organized as follows:

- This preface, which includes an overview of the manual, a summary of its contents and a list of conventions used throughout the manual.
- An overview of the AlphaStation 600 system that includes a system block diagram and information on cache sizes, memory capacity, number of IO slots, and so on.
- A high-level description of the basic ASIC functionality that includes basic system transactions (for example, CPU I/O read and write, DMA read and writes).
- A describes the EV5 address space partitioning and how this space coexists with the PCI address space. Emphasis on dense- and sparse-space CPU I/O addressing and details of the DMA scatter/gather address translation.
- A discussion of memory motherboard SIMM and I/O Subsystem Module configurations.
- A physical description of the system that includes board layout diagrams, pin designations/signals, and logic functions.
- A description of all Control and Status registers.
- A hardware description of the AlphaStation 600 interrupts and error strategy. The errors are defined from a hardware point of view (which bits in which error registers are set for the various errors). This includes the format of the AlphaStation 600 Machine Check logout to assist software developers write the machine check handler.
- System hardware and firmware power-up, initialization, and reset .
- A definition of the AlphaStation 600 I/O Subsystem Module bridge chip set, the component parts it uses (for example, arbiter, interrupt logic, etc.), and suggestions/requirements for programming the internal registers.
- System coherency and instruction ordering for the conjunction of three architectures (EISA, PCI, and Alpha). An explanation is provided to clarify which architectural requirement subsumes the other architectures. Some issues are in the hardware domain, but in certain cases, the architecture/hardware has specific requirements that the software must specify (for example, when to flush buffers; when I/O reads are required to guarantee coherency).

Conventions

The following conventions are used in this manual:

- All Numbers Are Hex unless otherwise noted
- UNP = stands for UNPredictable.
- MBZ = stands for Must Be Zero.
- IO or I/O. The term IO is used within technical terminology when a / is not appropriate in a number string. The I/O is used to reference the Input/Output functionality in text.
- Word = 2 bytes
- Longword = Doubleword (PCI term) = Dword = 4 bytes.
- Quadword = 8 bytes

Abbreviations

The following abbreviations are used throughout the manual.

Symbol	Description
RO	Read Only
RW	Read Write
WO	Write Only
RWC	Read, Write to Clear

1

AlphaStation 600 System Overview

Introduction

The AlphaStation 600 system is a high-performance, deskside workstation based on the EV5 implementation of the Alpha architecture. It supports WNT, OSF/1, and OpenVMS operating systems. Figure 1-1 shows the AlphaStation 600 system. From the perspective of the system programmer, the AlphaStation 600 system is:

- **EV5 CPU** with a 128 KB serial boot ROM. Jumpers are provided so that one-of-eight alternative serial ROM patterns can be loaded into the EV5 (mainly for lab debug). The EV5 8 KB I-cache can be fully loaded by any of the serial-ROM patterns.

EV5 chip summary	
Cycle time	4.4 ns - 3.2 ns
Address size	43 bit virtual address, 40 bit physical address, 8 KByte page size
Pipeline depth	7 stage integer, 9 stage floating.
On-chip Icache:	8 KB, virtual, direct-mapped
On-chip Dcache:	8 KB, physical, direct-mapped
On-chip Secondary cache	96 KB, physical, 3-way set-associative
On-chip TLB	64-entry ITB and 48-entry DTB, 128 Address space numbers.
Write Buffer	six 32-byte entries
Issue rate	4 instructions per cycle (2 integer, 2 floating point).

- **EV5 speed bins:** Simple crystal swaps and serial ROM changes are all that are required to support various EV5 speed bins. The AlphaStation 600 system is synchronous to the EV5 clock.

  Thus, the PCI clock, the cache timing and the memory timing are an integer multiple of the CPU frequency, as shown in the next table. Consequently, not all CPU frequencies will optimize the Bcache and system timing (for example, a 4 ns EV5 will result in a 7% slower PCI running at 32 ns).

-

CPU speed		Bcache timing		System timing	
Mhz	ns	ns	CPU cycles	ns	CPU cycles
250	4.00	28	7	32	8
266.67	3.75	26.25	7	30	8
275	3.636	25.46	7	32.7	9
300	3.333	26.67	8	30	9
312.5	3.20	25.6	8	32	10

- **Direct-mapped, write-back, ECC protected, module-level Bcache**. The Bcache is a plug-in option allowing various speed and size configurations. The Bcache is partitioned across 3 SIMMs which must all be inserted for the Bcache to function. The block size is fixed at 64 B. The AlphaStation 600 system does not support a duplicate TAG store.

  Current cache designs vary from 2 MB to 16 MB, with 4 MB as the "typical" size. The first Bcache design uses 15 ns SRAMs and achieves an access time of 24.9 ns[1] for the first 128b and 21 ns thereafter for all subsequent, contiguous reads (called wave-pipelining). Faster Bcache designs will be available.

  DMA writes invalidate the Bcache.

- **System memory: 256-bit data-width, ECC protected**. The system memory comprises two Memory Motherboards into which the memory SIMMs are inserted. The AlphaStation 600 system cannot be configured with only one memory motherboard; both must be resident and symmetrically populated with SIMMs. The memory latency with 60 ns SIMMs is 180 ns (timed from the EV5 requesting a fill, until the data is received by the EV5). Currently, there is only one memory motherboard variant planned.
- Each motherboard tower holds up to 16 standard 36-bit SIMMs providing a capacity from 32 MB to 1 GB (and eventually 4 GB when 64-Mbit chips become available).
- **Maximum memory:** Although the AlphaStation 600 system chip set can support 8 GB of physical memory, the workstation implementation is limited to 4 GB.
- **The AlphaStation 600 system I/O modules.** The system comes with three I/O cards: a PCI graphics card; the I/O Subsystem Module, PCI-based, SCSI/Ethernet card; and an ISA-based Audio card. The remainder of the system I/O (serial lines, etc.) are provided on the SystemBoard. Table 1-1 summarizes the I/O arrangement.

[1] Depends on the EV5 frequency and the multiple selected for the cache loop time (for example, a 3.25 ns EV5 will have a 26 ns cache loop time).

Figure 1-1 **The AlphaStation 600 System** Block Diagram

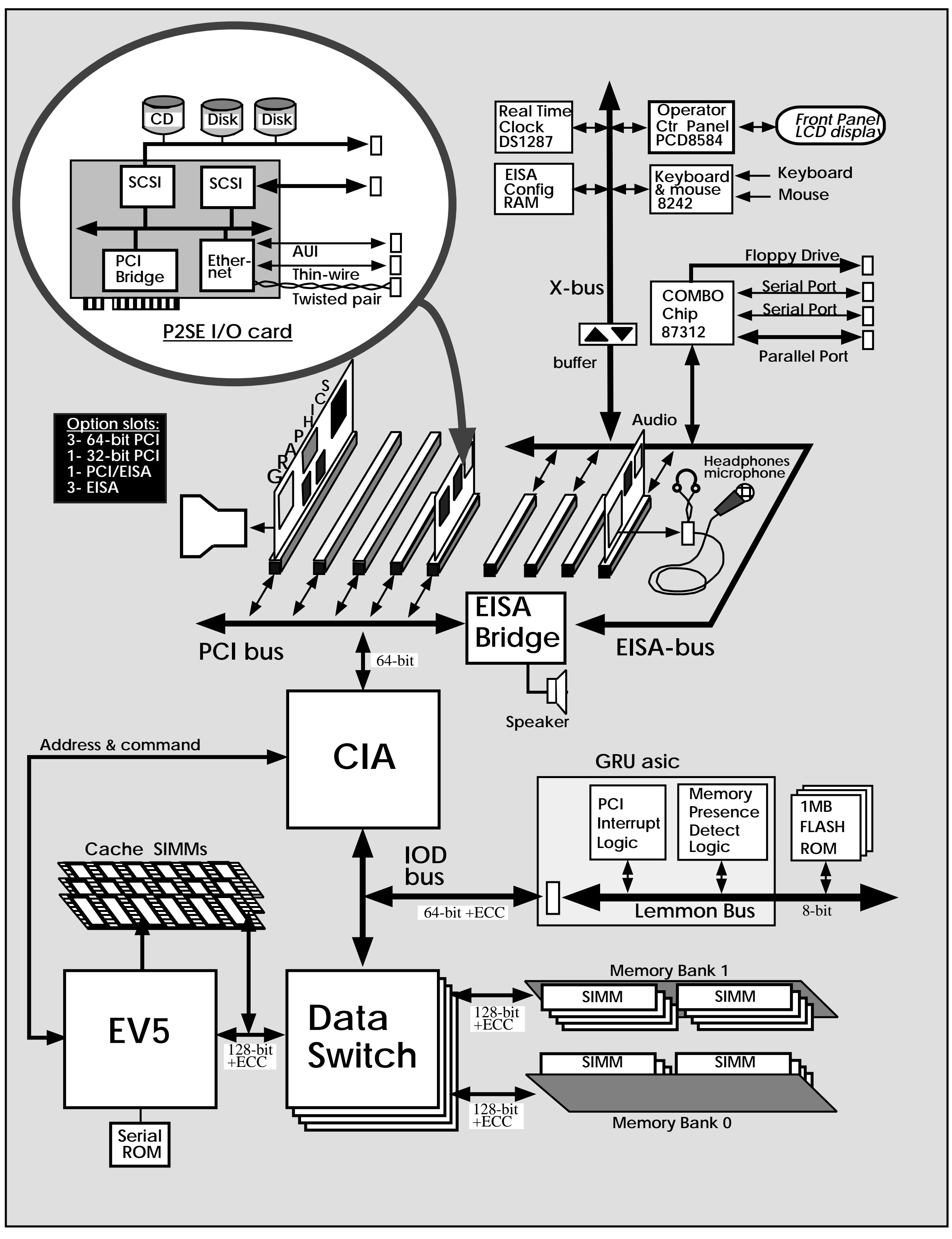


Table 1-1 **The AlphaStation 600 System** I/O Summary

What	Where	Additional information
Keyboard and Mouse	System module (on the X-bus)	Intel 8242
Time of year (TOY)		Dallas 1287
Battery-backed SRAM-8 KB		Dallas 1225
Operator Control Panel		PCD 8584
Parallel Port Serial Port (2) Floppy IDE	System module (EISA bus)	87312 Combo Chip (hardware auto-configured with FAR = 11)
Audio	ISA card	Microsoft Sound or OAK Card
Ethernet	I/O Subsystem PCI option module	TULIP - DC287
Internal SCSI		Qlogic ISP 1020
External SCSI		
Graphics	PCI option module	TGA

- **I/O Subsystem Module Ethernet/SCSI PCI I/O module:** The internal and external SCSI PCI devices, with the Ethernet device, are mounted on a separate I/O module. This allows for future upgrades (for example, better SCSI chip). This I/O module will occupy one of the 32-bit slots. The devices on this card are:
  — **Qlogic ISP1020 PCI-SCSI** - two chips
  — **PCI-Ethernet chip** (DECchip 21040 - Tulip)
  — **PCI-PCI bridge chip** (DECchip 21050 - PPB)
- **The AlphaStation 600 System** uses the 87312 chip for serial/parallel I/O. This is hardware auto-configured with CFGn<4:0> = 1.
- **Industry standard ISA Audio Module** - Microsoft Sound System or the OAK card.
- **No on-board graphics** - Graphics arre provided by plug-in options.
- **64-bit PCI** - The AlphaStation 600 system generates and accepts 64-bit data but does not generate 64-bit PCI addresses.
- **33Mhz PCI -** The AlphaStation 600 system does not support the latest 66MHz incarnation of the PCI.
- **Back-to-Back PCI cycles -** The AlphaStation 600 system will issue and accept fast back-to-back PCI cycles in dense-space only (this logic can be disabled via a CSR).
- **8 PCI and (E)ISA slots** - The AlphaStation 600 system has four PCI slots, three (E)ISA slots and one shared PCI/EISA slot -- see table below. Three 64-bit slots are provided. Two PCI slots are used for the graphics and the SCSI/Ethernet, and one (E)ISA slot is used for audio. Five slots remain for customer expansion.

Slot	Type	AlphaStation 600 reserved Usage
1 2 3 4	PCI -- 64 bit PCI -- 64 bit PCI -- 64 bit PCI -- 32 bit	Graphics PCI I/O module (ethernet/SCSI)
5	EISA/PCI (32bit) shared slot	
6 7 8	(E)ISA (E)ISA (E)ISA	 Audio

- **PCI-EISA bridge chip set -** Intel 82374EB (ESC chip) and Intel 82375EB (PCEB chip)
- **1 MB Flash ROM and serial ROM** - for console/diagnostics.
- **Scatter/gather PCI to memory addressing** - as defined for the reference machine by the I/O task force. Thus both physical and virtual DMA capability are available. The CIA ASIC contains a 32-entry TLB for the PTEs (configured as 8 TLB entries of 4 contiguous PTEs).
- **Dense/Sparse space:** as defined for the reference machine by the I/O task force.

ASIC Summary

The AlphaStation 600 system block diagram is shown in Figure 1-1. The major components are:-

- **Five ASICS (2 designs):** 4 Data Switch ASICs and 1 CIA ASIC:
  — **The CIA** - this ASIC accepts the address and command from the EV5 and drives the memory array with the address, RAS and CAS, signals, among others.. In addition, it also provides an interface to the PCI bus. The current CIA is in a 383-pin PGA design. Other packages are being evaluated to reduce cost.

  — **The DSW -** this is the data slice ASIC and provides the data path between the EV5, memory and the CIA (for PCI data). It fits in a 208-pin PQFP. When used in the AlphaStation 600 system, the DSW provides a 256-bit wide memory path; but the ASIC can also be configured for a 128-bit memory datapath. Four ASICs are always required, regardless of the memory width.
- **Miscellaneous ASIC** - the GRU ASIC is on the IOD bus. This small ASIC handles the PCI-interrupt logic, the memory/cache presence detect logic, and generates the 8-bit Lemmon Bus used to access the Flash ROMs and drive optional system LEDs.

Hardware Jumpers

The following jumpers and DIP switches are for debugging prototypes. Details are not provided since these are purely for lab use.

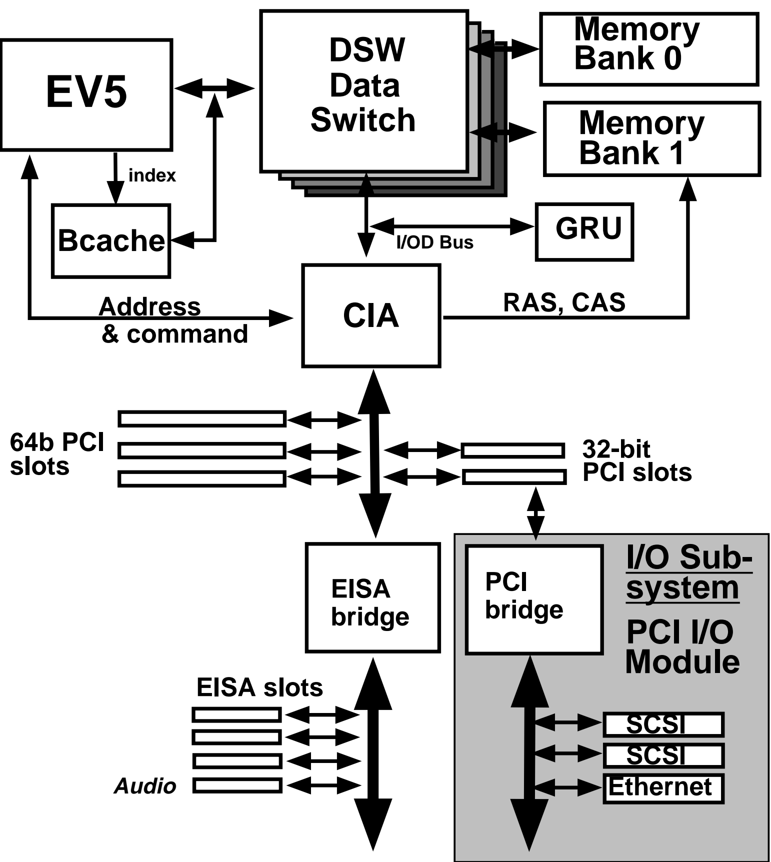
- **SROM code select.** Used to select from one of the 8 stored patterns in the SROM. Only one jumper should be shorted at a time.
- **EV5 clock multiple DIP switch.** This DIP switch will only be provided on the first few debug systemboards. This is used during reset time to select the EV5 system clock multiple.

2
AlphaStation 600 ASIC Overview

Introduction

This chapter describes the AlphaStation 600 system data path and focuses on the internals of the Control, I/O and Addressing (CIA) and Data SWitch (DSW) ASICS. Figure 2-1 shows a simplified system block diagram to help place the CIA and DSW in context:

Figure 2-1 System Block Diagram Showing CIA and DSW

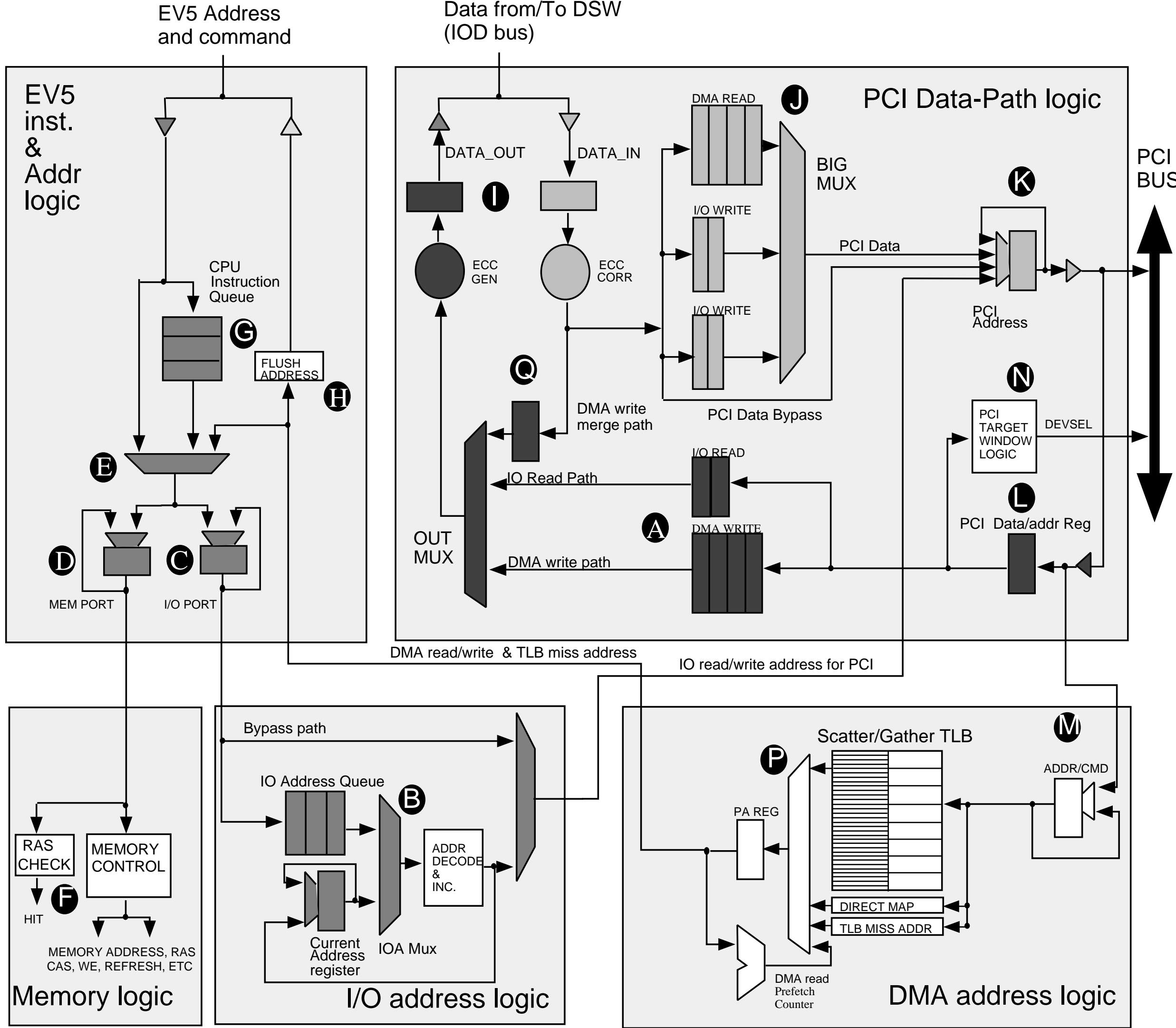


Note that the I/O Subsystem module is shown plugged into a 32-bit PCI. This is the preferred location but, as the module is a standard PCI module, it can be inserted in any of the 64/32-bit slots. This chapter focuses on the main system data-path which is implemented in two ASICs.

The CIA ASIC

Figure 2-2 shows the CIA internals pertaining to the CIA implementation used with a 64-bit PCI interface and 64-bit datapath to the DSW); the 32-bit PCI version of the CIA is not described.

Figure 2-2 The CIA Block Diagram



Shading is used in the above figure to highlight the major datapaths and the major logical entities. For instance, the darkest shaded datapath (bullets L, A, Q, I) represents the path for data returning from the PCI bus to the DSW (that is, the DMA write and I/O read paths). All these highlighted paths sre described later in this chapter. The functional entities shown partitioned into five shaded boxes are described next. These are:

EV5 Instruction and Address Region

Conceptually, this logic accepts the commands from the EV5 and directs the instruction to the memory port (bullet D) or the I/O port (bullet C). The DMA read/write address (or the scatter/gather TLB miss servicing address) also have access to the memory port through the multiplexer at bullet E. A 3-deep CPU Instruction queue is provided to capture EV5 commands should the memory/I/O port be busy.

For example, three addresses would be held if the EV5 issues two read misses with, at most, one having a victim[1]. This accounts for the three-entry buffer.

The Flush Address register (bullet H) is used during DMA reads or writes to interrogate the Bcache for the latest data (the Bcache is a write-back cache, and hence may have the only valid copy of the required data).

PCI Datapath

The CIA is, in a PCI sense, the host-bridge. It generates/decodes the PCI addresses and supplies/receives the data. Part of the PCI data path is in the CIA chip, and the remainder resides in the DSW ASIC. The ECC generation and check logic is provided here since the sliced nature of the DSW precludes placing ECC there.

Separate buffers are provided in the CIA (bullet J and A) for DMA and I/O read/writes. These buffers are either 32B or 64B in size, and are shown partitioned into 16B entities (which corresponds to the width of the EV5 data bus).

The PCI commands which the CIA responds/sends are listed in the following table:

Table 2-1 CIA PCI Commands

PCI command	Command type	CIA slave	CIA master
0000	Interrupt Ack	No	Yes
0001	Special cycle	No	Yes
0010	I/O read	No	Yes
0011	I/O write	No	Yes
0100	reserved	--	--
0101	*reserved*	--	--
0110	Memory read	Yes	Yes
0111	Memory write	Yes	Yes
1000	reserved	--	--
1001	*reserved*	--	--
1010	Configuration read	No	Yes
1011	Configuration write	No	Yes
1100	Mem Read multiple	Yes	No
1101	Dual addr cycle	Yes	No
1110	Mem Read Line	Yes	No
1111	Mem Write and Inv	Yes(1)	No
Note: (1) Aliased to Memory write			

CIA PCI features:

- 64-bit PCI bus width
- 64-bit PCI addressing (using DAC cycles)
- Capability to issue PCI fast back-to-back cycles in dense space

Note that the DSW also has buffers for the DMA and I/O paths. This duplication simplifies control logic (the PCI control logic uses a buffer if the IOD bus to the DSW is not free); simplifies handling PCI target stalls and retries; the ECC logic requires that 32-bit PCI data be built up to 64-bits; buffering is required to compensate for the various bus widths (IOD is either 32 or 64 bits, memory is either 256 or 128 bits, and the EV5 bus is 128 bits).

[1] A **victim** is the cache block that has to be displaced to make room for the read miss (fill) data.

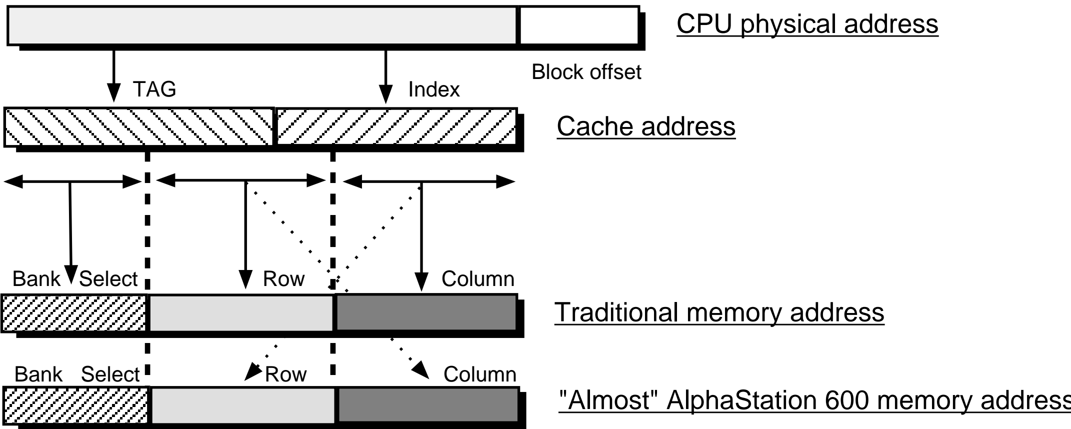
Memory Logic

This logic (bullet F) provides the row and column address for the memory banks as well as all the control signals (RAS, CAS, write enables, memory enable). The memory logic provides control signals to the DSW to inform it when to send/strobe the data to/from memory. Finally, this logic controls the memory refresh.

The next few paragraphs describe how the AlphaStation 600 system *eliminates RAS cycles for victim writes. This is a performance enhancement and is transparent to software.*

The AlphaStation 600 system design uses a special technique to eliminate RAS cycles on victim data. Instead of using the traditional approach of mapping the CPU address bit-for-bit to the memory address (see Figure 2-3), The AlphaStation 600 system shuffles the memory address bits so that the high-order CPU address bits (part of the Bcache Tag portion) become the memory column address and the low-order CPU address bits (part of the cache index portion) become the memory row address. Note that the high-order Tag bits are used for memory bank selects in both schemes[1].

Figure 2-3 Memory Address Swizzling



The reason for this method is apparent in Figure 2-4. Since a cache is much smaller than the available memory, multiple memory locations will alias to the same cache location -- this is shown in the left-hand side of the figure below where data A1, B1, C1, etc. all map to the same cache location. Suppose that data A1 currently resides in the Bcache, but the CPU wishes to access data C1. A cache miss will occur which will fetch data C1 and overwrite A1 in the cache; if A1 is the only valid copy of the data (for example, if an earlier CPU write had previously updated A1) then A1 has to be written to memory before the requested data, C1, can overwrite A1. This displaced data is referred to as a victim, and is fairly common (around 50% of read misses suffer victim displacements).

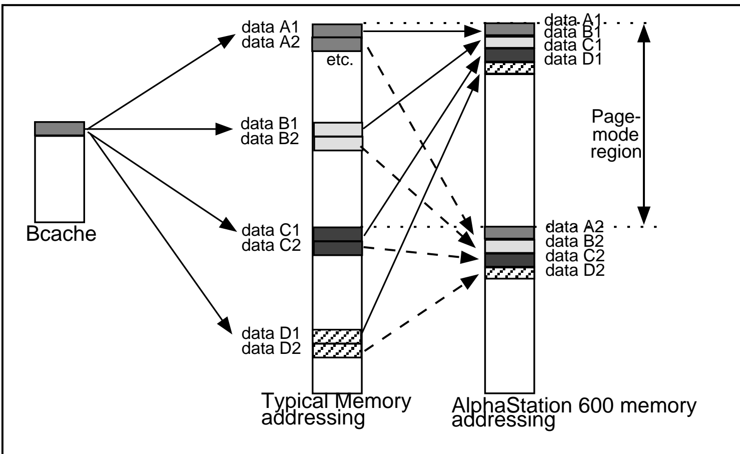
Referring again to Figure 2-4, note that in the traditional memory addressing scheme, the data (for example, data A1) and the potential victim (for example, data C1) are well separated in memory -- they are separated by a multiple of the cache size[2]. This sparse distribution of potential victim locations means a read fill block and its victim may not reside in the same memory page-mode region.

[1] This description is only a first order approximation; the actual AlphaStation 600 system implementation is a little different.

[2] With a typical AlphaStation 600 system Bcache size of 4 MB, this separation will be many, many megabyes.

However, by interchanging the memory Row and Column address bits, the potential victim locations now reside in the same memory SIMMs and the same row within the SIMMs. Now the chance of a cache block and its victim residing in the same memory page-mode region increases astronomically. This will remove most, if not all, RAS cycles from victim writes. In fact, for any cache greater than 1 MByte, all victims will be able to share a RAS with the read Miss.

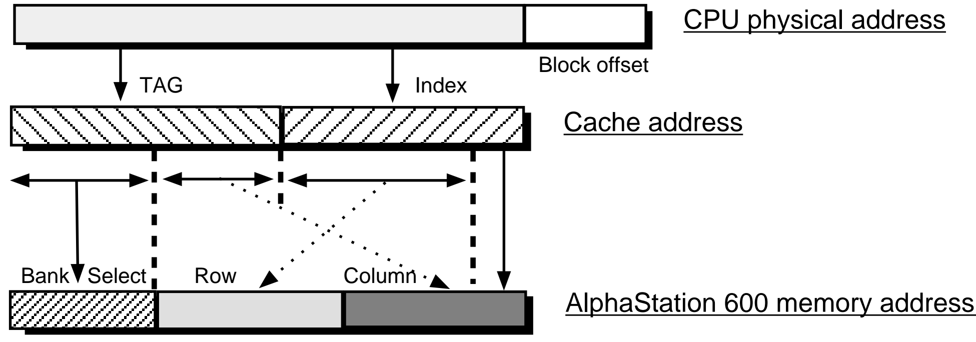
Figure 2-4 **Victim Aliasing**



However, this approach hinders DMA transactions: the figure above shows that consecutive blocks (for example, data A1, A2, A3) end up being scattered a page-mode region apart in the AlphaStation 600-type memory addressing scheme. This implies that consecutive DMA read/write blocks will require a RAS cycle, constraining the attainable bandwidth. The traditional scheme does not suffer this problem.

Hence, a compromise between the two schemes is used in the AlphaStation 600 system : four low-order bits of the index map straight to the low-order bits of the column address (just like in a traditional scheme), while the remaining high-order index bits go to the memory row-address. The remainder of the column address uses the Tag-portion of the physical address.

Figure 2-5 **AlphaStation 600 System Memory Addressing**



With this compromise scheme, a DMA transaction will march through four 64B blocks (that is, 64 longwords), *on average,* before requiring a RAS cycle. This constitutes a large DMA transfer, minimizing the RAS penalty. Although the AlphaStation 600 system was originally designed to take advantage of this scheme, it transpired that the simpler approach of a RAS cycle on every DMA 64-byte block provides more than enough bandwidth for even a 64-bit PCI transfer (DMA read prefetch).

The logic which determines if a RAS cycle is required is shown by bullet F. The CPU-to-memory path needs to be checked in case the Bcache is disabled, or if the Bcache is too small.[1]

I/O Address Logic

This logic (bullet B) handles the I/O read/write addressing. The Address decode logic extracts the PCI address from the dense/sparse space CPU address encodings (see Chapter 4, AlphaStation 600 Addressing, for more details). This logic also increments the current address each data cycle for two reasons: first, in case of a PCI retry, which will require the transaction to be resumed later, we need to start with the address of the aborted data; and second, to provide a pointer to the next data item to be loaded/sent from the I/O read/write buffers.

The AlphaStation 600 system can queue up to six I/O writes: two of the I/O writes can be waiting in the CPU queue (bullet G); and four I/O writes can be sitting in the I/O address queue (bullet B -- a three-entry I/O address queue is provided together with a single entry Current Address register). This allows six I/O writes to be outstanding (the DSW provides four corresponding I/O write data buffers -- bullet 4 in Figure 2-6; and the CIA provides two more buffers -- bullet J). These buffers are required to sustain maximum bandwidth for memory copy to I/O space. A bypass path is provided for certain dense-space I/O writes (that is, the longword valid bits for the first 16-bytes of data are either 1111, 1011, 1010 and 1001 -- these cases are optimized for a 64-bit PCI with at least two PCI data cycles).

DMA Address Logic

This logic converts the PCI's address to the CPU memory address space. Two conversion methods are provided: a direct path where a base offset is concatenated with the PCI address; or a scatter/gather map which maps any 8 KB PCI page to any 8 KB memory space page (bullet P). See Chapter 3, AlphaStation 600 Addressing, for more details. The scatter/gather TLB is 8 entries deep; but each entry holds four consecutive PTEs. A TLB miss is handled by hardware; but software is required to invalidate stale entries by writing to the SG_TBIA CSR.

A counter is used on the output of the PA register to generate the prefetch address for certain DMA read misses. An 8 KB detector prevents prefetching across page boundaries.

The DSW ASIC

The DSW ASIC interfaces the data paths between the memory, EV5 and the CIA (for PCI data). The ASIC is composed of mainly data buffers and multiplexers-- see Figure 2-6. All control for the DSW is supplied by the CIA (albeit, some encoding and simple sequencing is performed by the DSW). Features of the DSW design are:

- Victim buffer -- 64B
- I/O read buffer -- 32B
- Four I/O write buffers -- 4 * 32B
- Two DMA buffer sets are provided for reads and writes. Each buffer set consists of three buffers: one for the memory data, one for the Bcache data and one for the PCI DMA write data (not used during DMA reads). The Bcache and memory data could have shared one buffer (since only one of these two will provide valid data), but two

[1] As an example: a 1 MB Bcache the Tag bits are <32:20>. The AlphaStation 600 system has 16 memory banks which are addressed by <32:29>. This leaves <28:20> which is 2**9 = 512 victim blocks. A 1K-entry memory page corresponds to 512 blocks (the memory width is half a block) which will just hold all the victim blocks. A smaller 512 KB cache will have twice as many victims and would not fit. In the AlphaStation 600 system's case, the situation is slightly worse since the low-order address bits map straight to the column address.

buffers simplified the control logic.

Quadword valid bits are provided for the DMA write buffer. These valid bits are used to merge the appropriate DMA write quadword with the memory/Bcache data. Finer granularity merging (for example, bytes) is performed in the CIA (because of ECC requirements) by looping the appropriate memory/Bcache quadword through the CIA and merging in the valid DMA write bytes (see bullet Q for this path in the CIA).

No data is preserved in the buffers across transactions (that is, no DMA read prefetched data or posted DMA write data). Consequently, there are no coherency issues regarding DMA read/write data lingering in the buffers.

Flash ROM

A 1 MB Flash ROM is provided on the system module (see bullet 6 in Figure 2-6). The Flash ROM is visible to software by reading/writing certain CSRs in the GRU ASIC.

Figure 2-6 The Data Switch Block Diagram

CPU Memory Read

The EV5 Read Miss command and address are sent to the CIA. If the CIA is idle the command will be stored directly in the memory port register (bullet D in Figure 2-2); otherwise the command enters the three-deep CPU Instruction queue (bullet G) and remains there until the memory port is free. The CIA will accept one more subsequent Read Miss from the EV5 and store it in the CPU Instruction Queue.

There are three paths to the memory port which have to be arbitrated for (bullet E): the two paths described above -- namely the direct path and the CPU Instruction queue; and thirdly, the DMA read/write address path (which is also the path for scatter/gather TLB miss addresses). Note that, up-stream of the multiplexer at E, all instruction ordering from the EV5 is preserved; downstream, the ordering between I/O write transactions and CPU memory transactions is lost. This "post and run" I/O write coherency issue is discussed in Chapter 11, System Coherency.

Once the CPU Read Miss gets into the memory port, the memory controller (bullet F) generates the RAS, CAS, and address signals for the memory array.

The memory controller then waits for the memory access delay before instructing the DSW to accept the memory data. The memory width for the AlphaStation 600 system is 256-bits (32 B), and thus two memory cycles (typically 60 ns) are required to access the 64 B block.

The DSW clocks the data into the Memory Data-in register (bullet 2 of Figure 2-6). This register is clocked on a 15 ns clock (not a 30ns clock) in order to minimize memory latency. The data is then sent to the EV5 through a further flip-flop (bullet 7) which synchronizes the data to the EV5 system clock (30 ns). The read data is returned to the EV5 in wrapped-order. The block size for read data is fixed at 64 bytes.

CPU Memory Read with Victim

This is similar to the Read Miss case previously described except that the EV5 will also send out the victim block (that is, the modified [dirty] block which will be displaced from the Bcache by the Read Miss data).

The command and address for the Victim always follow the Read Miss command[1]. Consequently, the Read Miss command will get to the Memory port first and thus the Victim command and address are always written into the CPU Instruction Queue (G). The victim block data is saved in the Victim Buffer in the DSW (bullet 1).

The memory controller in the CIA (bullet F) generates the memory write pulses and instructs the DSW to send the victim data to memory. The victim data path is straightforward: the data is sent out of the victim buffer (bullet 1) and through the Memory Data-Out register (bullet 2) to the two memory banks.

The victim data is written to memory after the read data has been fetched from memory (the CIA arbiter ensures that a read miss and victim write are an atomic operation). The row address portion of the victim address is compared to the row address of the current read miss (bullet F). If they match then no memory RAS cycle is required, instead only CAS strobes are performed. The address bits for the memory have been carefully interspersed to maximize the chance of a victim row address "hitting" the read address -- this performance feature is totally transparent to the software.

The Victim buffer is invalidated if a DMA write (or DMA read with lock) "hits" the victim buffer. Until the EV5 has its read data returned, the victim block is still in the Bcache and is still "owned" by the EV5 (even though the EV5 sent a copy of the victim data to the DSW). It is possible for the Read-with-victim command from the EV5 to be stalled behind a DMA write. The DMA write could "hit" the victim block -- in this case, the DMA write will issue a *FLUSH* command to the EV5, which will result in the EV5 providing the victim data to the DSW (to one of the Bcache buffers -- bullet 5) and then the EV5 will invalidate the victim block in the Bcache. Consequently, the victim data waiting in the victim buffer (bullet 1) is no longer valid and is invalidated by logic in the CIA.

CPU I/O Read

An I/O read by the CPU can be to one of five places: the PCI memory space; the PCI I/O space; the PCI configuration space; the GRU CSRs (which includes the Flash ROM); and the CSRs in the CIA (there are no CSRs in the DSW). The address for the I/O read is either in sparse space or dense space -- for details please refer to Chapter 3, AlphaStation 600 Addressing.

The I/O read command is accepted by the CIA in a manner similar to a memory read except that the instruction ends up in the I/O port (bullet C). All I/O read commands go to the I/O address queue (bullet B); no bypass path is provided for dense space I/O reads.

Decode logic is provided on the output of the I/O address queue to extract the byte address for the PCI from the CPUs sparse space encoding, and to decode the address for

[1] The EV5 must have "Victim first" disabled for the AlphaStation 600 system.

the correct region (PCI memory, I/O, configuration, CSR or Flash ROM). We will only consider an I/O read destined for the PCI since this is the more interesting case.

An incrementer is provided to increment the current longword/quadword[1] address stored in the Current address register each data cycle. This is needed in case of a PCI retry and is also used to index the next data item in the I/O write/read data buffers. The value in the Current Address register corresponds to the address of the data item on the PCI bus (it has the same canonical time as the PCI data-out register at bullet K).

The I/O read address is sent to the PCI after any prior I/O writes have completed[2] (that is, strict ordering is maintained). The PCI returns the requested data and places it in the I/O Read buffer (bullet A). The contents of this I/O read buffer are next copied to the I/O read buffer in the DSW (bullet 3) and then sent to the EV5.

One may wonder why we have replicated the I/O read buffers. The I/O read buffer in the CIA is provided for the following reasons: first, because the path to the DSW may be busy with the tail end of a prior DMA write (especially if the data has to be merged with memory data to build it up to the ECC width); second, at least 64 bits of storage are required for ECC since the PCI can return 32 bits of data at a time; and finally, as a control convenience to handle the vagaries of the PCI protocol (for example, retries). The I/O read buffer in the DSW was provided to build the data up from 64-bits (width of the bus between the CIA and DSW) to the 128 bits required by the EV5.

I/O reads from the EV5 are of an 8 B resolution and the CIA always returns 32 B. If a finer resolution is required, then sparse space must be used, the details of which are fully covered in Chapter 3, AlphaStation 600 Addressing. Data is returned in the appropriate byte lanes.

CPU I/O Write

The EV5 issues uncached writes to its I/O space with a longword resolution. For a finer granularity, use sparse space (see Chapter 3, AlphaStation 600 Addressing).

The data for I/O writes is captured in the I/O write buffer in the DSW (bullet 4). Four 32 B buffers are available in the DSW and a further two in the CIA: this number of entries allows AlphaStation 600 system to sustain maximum bandwidth on a large copy operation from memory through the CPU to I/O space. The data from these I/O Write buffers is sent to the two I/O write buffers in the CIA (bullet J): two buffers are provided, allowing one to be emptied to the PCI bus while the other is filled. Each 32B buffer in the CIA constitutes a separate PCI transaction (that is, no merging of the write buffers occurs). A secondary benefit for the CIA's I/O write buffers is simpler data-flow management when an obnoxious target PCI device stalls the I/O writes.

The address for an I/O write is sent to the CIA I/O port (bullet C). Thence, for a 32B aligned dense space write, the I/O write is either sent directly to the PCI bus via the bypass path, or queued up in the I/O address queue (bullet B). The fast, direct (bypass) path is used if: (1) there are no outstanding I/O commands; and (2) the command is an I/O write in dense space and the first 16-bytes[3] are mostly valid (that is, 1111, 1011, 1010, 1001 -- see the CIA ASIC chip spec for details). Note that this command also goes into the I/O address queue, but only as a convenient path in the I/O addressing logic section, to get the address into the Current Address register.

A total of six I/O write addresses can be queued: two in the CPU-queue, three in the I/O address queue, and the first I/O write in the Current Address register. The I/O address queue maintains strict ordering for I/O operations (but does not maintain strict ordering of I/O writes relative to any memory reads or writes -- see Chapter 11, System Coherency, for the implications).

[1] Depending if a 32- or 64-bit PCI transfer is in effect.

[2] Completed means that the CIA has issued the writes on the PCI bus and the target devices have accepted them.

[3] Since we only see 16B each cycle from the EV5.

DMA Transactions

The PCI address and command are captured in the Address/Command register (bullet M) and the data/address register (bullet L). The address is compared against four address windows to determine if this PCI command should be accepted or ignored by the CIA[1]. Address windows are a requirement of the PCI specification and are software programmable -- they are described in detail in Chapter 3, AlphaStation 600 Addressing. All PCI commands destined for memory are accepted by the CIA.

> *Implementation detail:* Two registers sample the incoming PCI bus. One is used to capture the address and command (bullet M), and to hold onto the values for the duration of the transaction; while the other (bullet L) is used to primarily capture data cycles, but will also strobe in the address. The Target Window logic is attached to the Data-in register (L) rather than the Address register (M). The reason for this is the case of a DMA Write Scatter/Gather TLB miss. The intention is to grab a buffer's worth of data (64B) and then retry the master PCI device should it have more data. Thus we have freed the PCI bus for further transactions; but as we are busy servicing the TLB miss, we need to hold the virtual address in the Address register (M). Should another PCI transaction occur while we are servicing the TLB miss, we must accept that address and check it against our Target window. Consequently, the Target Window logic is attached to the free-running PCI data-in register (L).

There are three registers associated with each of the four PCI Windows:

- **Window Base register:** This defines the start of the Target Window. This register holds the SG bit which determines if the scatter/gather map is used for the translation.
- **Window Mask Register:** This defines the size of the Window
- **Translation Base register:** This holds a Base address used to relocate the PCI address in the CPU memory space (for direct mapping); and is also used to hold the Scatter/Gather Map Base address for scatter/gather mapping.

The Window Base register (see Chapter 3, AlphaStation 600 Addressing) SG bit determines how the PCI address is translated: if SG is clear, then the address is directly mapped by concatenating theTranslation Base Register to it; otherwise the address is mapped through the scatter/gather table, allowing any 8 KB of PCI address to map to any 8 KB of memory address.

[1] DOS address space is decoded by the PCI-EISA bridge chip and signaled to the CIA via the MEMCS# wire. See Chapter 3, AlphaStation 600 Addressing, for more information.

Figure 2-7 Scatter/Gather TLB

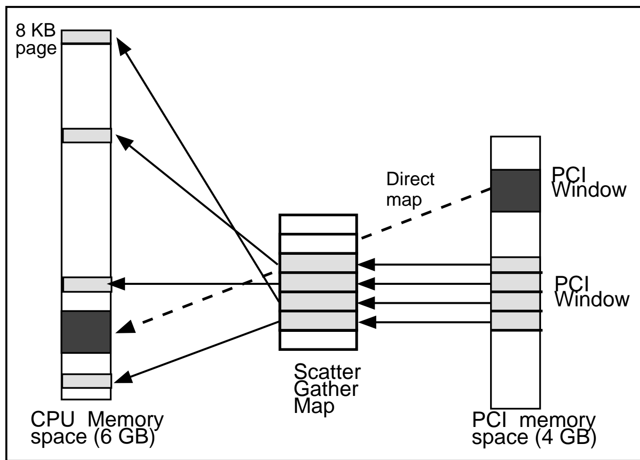


Figure 2-7 illustrates this mapping. This Scatter/gather table is located in memory, but the CIA provides an 8-entry TLB which caches the most recent scatter/gather table entries. Each TLB entry holds 4 consecutive scatter/gather table entries (thus mapping a contiguous 32 KB of virtual PCI addresses to any four 8 KB memory pages).

DMA Read

The translated address stored in the PA register (bullet P) is sent to the EV5 through the Flush/Read Address register (bullet H), and also to memory via the memory port (bullet D). If an EV5 data cache contains valid (modified) data then a copy is sent to the Bcache-buffer portion of one of the two[1] DMA buffers in the DSW (bullet 5); and the valid bit is set. Memory data is always fetched and lands in the Memory-buffer of the DMA buffer. The PCI-buffer part of the DMA buffer is not used for DMA reads (see Appendix A).

As soon as the first valid QW is available in the DSW's DMA buffer (bullet 5) it is sent to the DMA read buffer in the CIA (bullet J) via the multiplexers at bullets 5 and 8. Any ECC correction is done in the CIA (bullet I). From the DMA buffer in the CIA the data goes out on the PCI a cycle later (through the register at K). The DMA read data also takes the bypass path around the Big Multiplexer (bullet J) for as long as the PCI can accept this stream of data; once the PCI stalls then the buffers start to be used.

DMA Read Prefetching

The PCI supports three types of memory read commands. The following is the use as specified in the PCI Local Bus specification:

- **Read command:** used for small transfers (up to 1/2 a cache line)
- **Read Line command:** used for medium transfers (1/2 to 3 cache lines)
- **Read Multiple:** used for large transfers (more than 3 cache lines)

Note that for the PCI environment most devices will tacitly assume an Intel processor cache of 32 bytes (compared to the AlphaStation 600 system's cache line of 64 bytes). Hence, the AlphaStation 600 system's prefetch strategy is:

[1] PCI address<6> -- the even/odd 64B block bit -- determines which DMA buffer is used

Table 2-2 AlphaStation 600 Series Prefetch Strategy[1]

PCI memory read command	Prefetching
Read	None
Read Line	Prefetch 1 block
Read multiple	Prefetch till end of transaction

The counter used to increment the memory block address for prefetching is shown by bullet P.
This address goes to the Bcache and memory as per a normal DMA read. The returned data is sent to the next free DMA buffer in the DSW. So as one buffer is being copied down to the CIA, the other is free to accept DMA prefetched data. At the completion of a transaction, all prefetched data in the DMA read buffers is ignored (that is, no prefetch caching is performed). Prefetching does not occur over an 8 KB page boundary.

The 64B DMA read buffer in the CIA (bullet J) acts like two 32B halves during DMAs -- as one half is emptying to the PCI bus the other half is being filled.

DMA Write

The translated address stored in the PA register (bullet P) is sent to the Bcache through the Flush/Read Address register (bullet H), and also to memory via the memory port (bullet D). If the Bcache contains valid (modified) data, then the data is sent to the Bcache-buffer portion of the DMA buffer in the DSW (bullet 5), and the Bcache data is invalidated. Memory data is always fetched and lands in the Memory-buffer of the DMA buffer (see Appendix A).

The DMA write data is taken off the PCI bus and placed in the DMA write buffer (bullet A). If the data is a complete quadword then ECC is generated (bullet I) and the data is sent to the PCI-write portion of the DMA buffer in the DSW. If the data is an incomplete quadword then a merge operation has to be performed (bullet Q). The valid Bcache or memory quadword is sent to the CIA from the DMA buffer in the DSW, and the valid bytes of the DMA write data are merged in. This merged quadword then is ECC generated and is sent to the PCI-write buffer portion of the DMA buffer in the DSW.

The DMA buffer in the DSW builds the data up to 64B (the block size) before sending the data to memory. The memory logic (bullet F) generates the memory write pulses. The memory address is read from the PA register (bullet P) using the mux at bullet E. Note that should the I/O port (bullet C) be busy with an I/O write/read transaction, the memory port (bullet D) will be available for DMA writes[2].

One reason for providing a copy of the DMA write buffer in the CIA is because the data path to the DSW may not be available at the start of a DMA write (it could be busy transferring I/O write data from the DSW to the I/O write buffers if CPU I/O write had been received concurrently with a PCI DMA write).

Note that the DMA write data is always written to memory, and never stored (cached) in the DMA write buffers across transactions. The complexity of caching was deemed too risky relative to the potential benefit gained.

MB Instruction

The MB instruction can be disabled from leaving the EV5 (this is the expected AlphaStation 600 mode). If MB instructions are allowed to the CIA, then the CIA will effectively treat the MBs as NOPs (that is, in accordance with the "posted-write" ECO to the Alpha SRM -- see Chapter 11, System Coherency).

[1] Any PCI read can be mapped to any prefetch algorithm using the CIA_CTRL CSR.

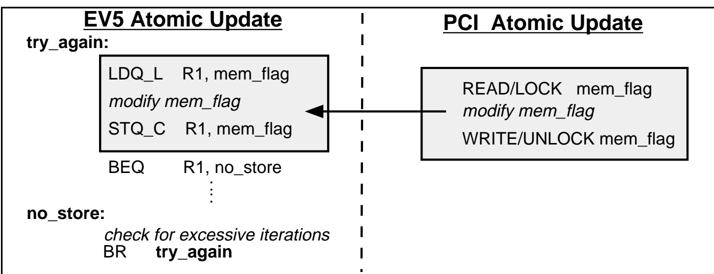
[2] The memory port may be busy with a CPU operation; but once the CPU operation completes, the DMA will get through.

LOCK Instructions

The AlphaStation 600 system has to contend with locks originating on either the CPU or the PCI. There are two cases which require that the EV5's internal lock-state on a block be cleared (from a system point of view). The first is during a DMA write to the locked block, and the second is when a PCI device has established a lock. The things to consider about system lock behavior are:

- **DMA write case:** The EV5 maintains its own lock flag and lock address in the EV5's BIU. This state is correctly maintained, even if the locked block is displaced from the Scache, as long as the system sends all DMA write FLUSH commands to the EV5 (that is, no duplicate cache tag). This is the case for the AlphaStation 600 system.
- **PCI lock established:** A PCI device can only establish a lock on a 16B block of data during a DMA READ. Once the lock is established (by the read succeeding), the PCI device has exclusive access to that 16B block (although the PCI target -- in this case, the AlphaStation 600 system -- is at liberty to lock a larger block). This PCI lock allows the device to do an atomic read-modify-write on a 16B entity.

  The Alpha and PCI architectures have different mechanism for establishing a lock. For the PCI, it is a successful READ (with the PCI LOCK signal asserted) that establishes a lock; for Alpha, it is a successful WRITE (STx_C) that establishes a lock. Hence, a PCI READ with LOCK must be treated by the EV5 as if it were a WRITE (that is, in terms of the Alpha architecture, it must have the same effect as would a write from some other processor). The figure below gives an example of the CPU and PCI vying for a memory flag:



  Both the EV5 and the PCI are attempting to do an atomic read-modify-write of a memory flag (for example, a semaphore). The EV5 starts first with the LDQ_L but will not know if the update is successful until the STQ_C completes successfully. However, assume that the PCI obtains the lock to the memory flag before the STQ_C -- with the READ/LOCK PCI command. For correct operation, the EV5's STQ_C must fail, otherwise the EV5 and the PCI device will both believe they have successfully modified the flag (for example, obtained a semaphore). To achieve the correct operation, the follwing is required:

  Before a PCI lock is established, the EV5 must clear its lock flag. The only mechanism to do this is a system write/flush to the locked block. Normally the CIA will handle a DMA read by sending a READ command to the EV5. <u>However, in this case, a FLUSH command is required</u> (in order to clear any lock which may be set on that block in the EV5). This unfortunately means that the system must not only do a DMA read operation, <u>but must also write the flushed-out block to memory</u>. Once the CIA successfully sends data to the PCI device, the PCI lock is established.

  During the PCI lock, the CPU may attempt to refill the flushed-out block (this will happen if it is in a loop repeatedly executing LDX_L - as depicted in the figure above). This requested block however cannot be provided by the system until the PCI lock has completed (because, once the block gets inside the EV5, the system loses control

of the block. The system could prevent the EV5 from setting its internal lock bit on the block initially, by using the SYSTEM_LOCK_FLAG_H, but the next time around the LDx_L/STx_C loop, the EV5 will enjoy a cache hit on the block and will be oblivious to the system lock state; thus, the loop will succeed this second time around).

The lock rules are:

- The SYSTEM_LOCK_FLAG_H signal to the EV5 is not used and is always tied true.
- All DMA write operations will result in the CIA issuing a FLUSH command to the EV5 (this is normal AlphaStation 600 system behavior).
- All DMA reads with the PCI LOCK asserted will result in the CIA issuing a FLUSH command to the EV5; the CIA must write the flushed-out block to memory. The internal CIA Lock Address register will be set with the locking address.
- If the EV5 requests a fill which hits the CIA lock address register then the fill is stalled until the PCI lock is relinquished by the PCI device.
- The EV5 LOCK command will be a NOP in the AlphaStation 600 system (this command is only required for systems with duplicate TAGs).
- The EV5 WRITE_BLOCK_LOCK will be treated as a WRITE BLOCK (that is, a plain write).

Locks to Uncached Space

The AlphaStation 600 system does not support LDx_L and STx_C to uncached space; these are converted to plain LDx and STx.

GRU ASIC

The GRU chip (DC7560A) contains sections of miscellaneous logic necessary for system operation. The major sections are:

- Flash ROM Interface
- Interrupt Logic
- Configuration registers for Cache and Memory
- System Reset

Refer to the GRU ASIC Specification for details.

IOD Interface

The GRU interfaces with the CIA ASIC via the low byte of the IOD bus and two control signals (GRU_SEL and GRU_ACK). CIA asserts GRU_SEL to pass a command on the IOD bus and GRU asserts GRU_ACK to inform CIA that read data is being returned. The commands are:

- CSR Write: CIA sends a CMD (1 byte) to the GRU, followed by 4 bytes of data
- CSR Read: CIA sends a CMD (1 byte) to the GRU, the GRU sends back 1 LW of data
- Flash ROM Read: CIA sends a CMD (1 byte) to the GRU, and then 3 bytes of address, the GRU sends back 1 LW of data
- Flash ROM Write: CIA sends a CMD (1 byte) to the GRU, and then 3 bytes of ad-

dress, followed by 4 bytes of data

The command sent to the GRU is based on the CSR_ADDR from the EV5 and whether the cycle is a read or write. The format is:

Bit[7]	Bit[6]	Bits[5:0]
CSR/Flash	Read/write	Address
1 = CSR 0 = Flash ROM	1 = Read 0 = Write	Address of internal CSR or partial address of the FROM

GRU Addressing

The physical address region 87.8000.0000 to 87.FFFF.FFFF is used to access the GRU ASIC on the IOD bus. These addresses access a number of CSRs as well as external Flash ROM space.

Table 2-3 GRU Address Space

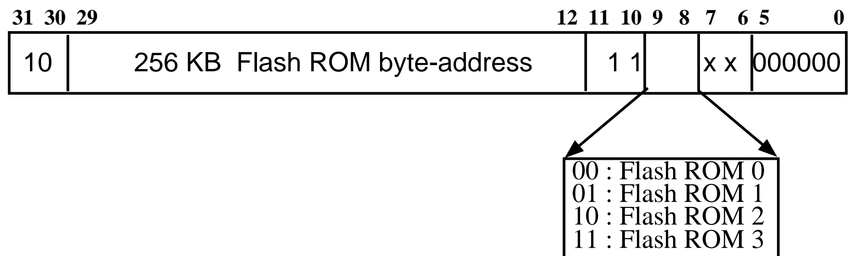
CPU Address	Selected Region	Mnemonic
87.8000.0000 87.8000.0040 87.8000.0080 87.8000.00C0 87.8000.0100	Interrupt Request register Interrupt Mask register Interrupt Level/Edge Select register Interrupt High/Low IRQ select register Interrupt Clear register	INT_REQ INT_MASK INT_EDGE INT_HILO INT_CLEAR
87.8000.0140 to 87.8000.01C0	*reserved*	
87.8000.0200	Cache and Memory Configuration register[1]	CACHE_CNFG
87.8000.0240 to 87.8000.02C0	*reserved*	
87.8000.0300	SET Configuration register[1]	SCR
87.8000.0340 to 87.8000.07C0	*reserved*	
87.8000.0800	LEDs (not used in current theAlphaStation 600 system)	LED
87.8000.0840 to 87.8000.08C0	*reserved*	
87.8000.0900	Force System Reset	RESET
87.8000.0940 to 87.8000.0BC0	*reserved*	
87.8000.0Cxx to 87.BFFF.FCxx	Flash ROM bank 0 256 KB byte-addressed by CPU address<29:12>	
87.8000.0Dxx to 87.BFFF.FDxx	Flash ROM bank 1 256 KB byte-addressed by CPU address<29:12>	
87.8000.0Exx to 87.BFFF.FExx	Flash ROM bank 2 256 KB byte-addressed by CPU address<29:12>	
87.8000.0Fxx to 87.FFFF.FFxx	Flash ROM bank 3 256 KB byte-addressed by CPU address<29:12>	

Flash ROM Interface

The GRU ASIC is controlled and addressed over the IOD bus. The GRU also has a bi-directional interface called the GRU_DAT<7:0> bus. The GRU_DAT bus attaches to 1024 KByte of Flash ROM and LEDs. The Flash ROM is divided into four banks each having 256 KB. The Flash ROM addressing scheme is shown in Figure 2-8. Also refer to Table 2-3 for the GRU address space.

[1] This functionality is not provided in version 1.0 of the GRU asic

Figure 2-8 Flash ROM Address scheme

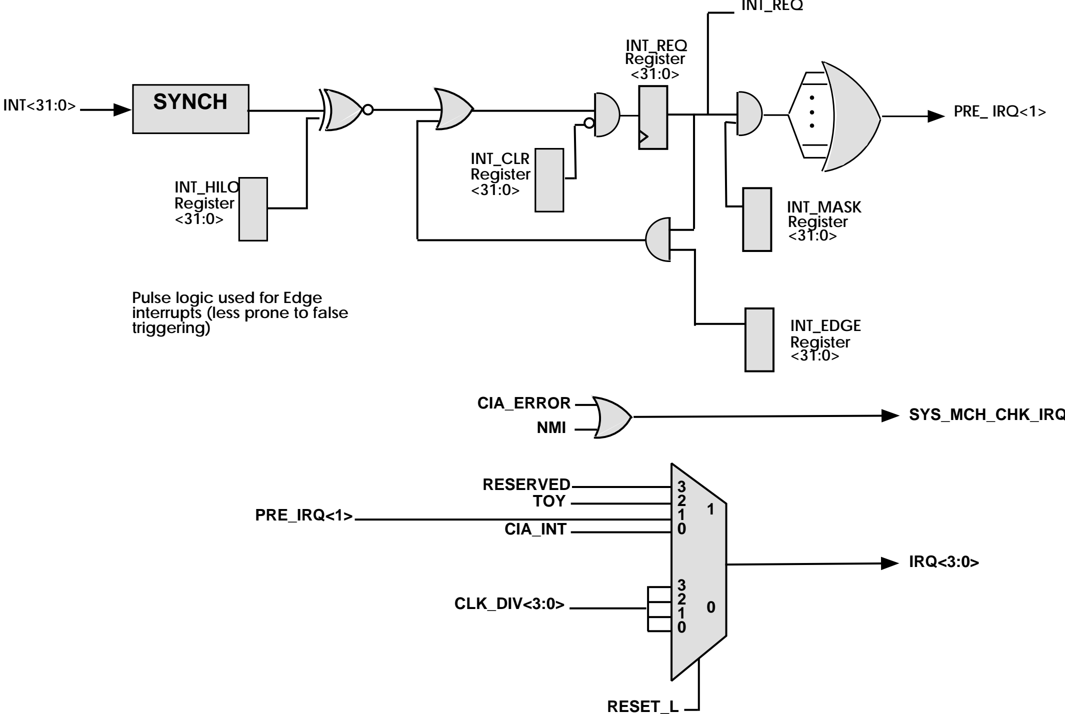


The data read from the Flash ROM travels over the GRU_DAT bus and back to the CIA on the IOD bus. It travels the opposite direction for Flash ROM writes. In all cases the address is sent to the Flash ROM from the GRU on a dedicated output bus (FROM_ADDR<17:0>). Also controlled from the GRU are the Flash ROM chip enable, output enable, and write enable signals.

Interrupt Logic

The Interrupt logic in the GRU is controlled by the primary interrupt inputs (PCI interrupts, CIA_ERROR interrupt, NMI, TOY and CIA_INT) and the values of the interrupt registers These registers include the Interrupt Request register, the Interrupt Mask register, the Interrupt Level/Edge register, the Interrupt High/Low register and the Interrupt Clear register. A block diagram of the interrupt logic is shown Figure 2-9

Figure 2-9 GRU Interrupt Logic



Configuration Registers for Cache and Memory

The GRU contains a Cache and Memory Configuration register (CACHE_CNFG) which contains the size and speed information for each individual cache SIMM (see Figure 2-10). Also included is the SET Configuration register. This register contains the access rate (speed) information for each memory SIMM (see Figure 2-11). Both the registers are loaded following system reset from the information on the presence detect pins of the SIMMs.

Figure 2-10 Cache and Memory Configuration Register

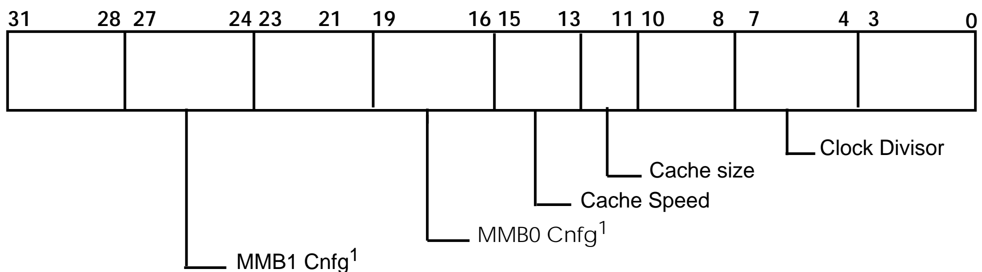
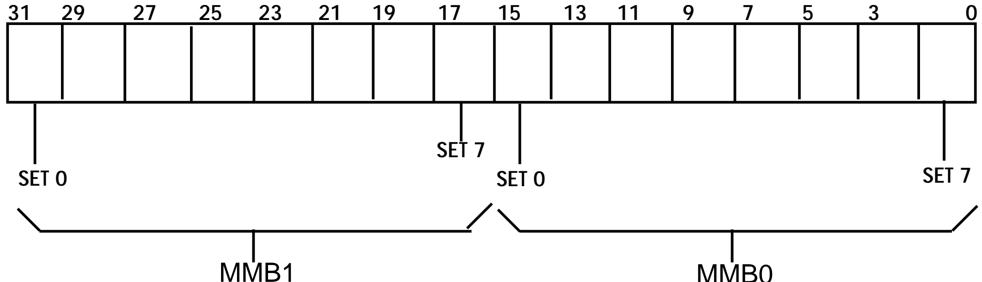


Figure 2-11 SET Configuration[1]



Reset Logic

The system reset signal (SYS_RST_L) is generated and output by the GRU. This signal resets all the chips on the module. Reset can be asserted asynchronously with the primary inputs DC_OK_L or OCP_RESET_L. It can also be asserted synchronously by writing a value of "$0000DEAD_{hex}$" into the Reset register in the GRU. In all cases SYS_RST_L is asserted for 256 cycles and deasserted synchronously..

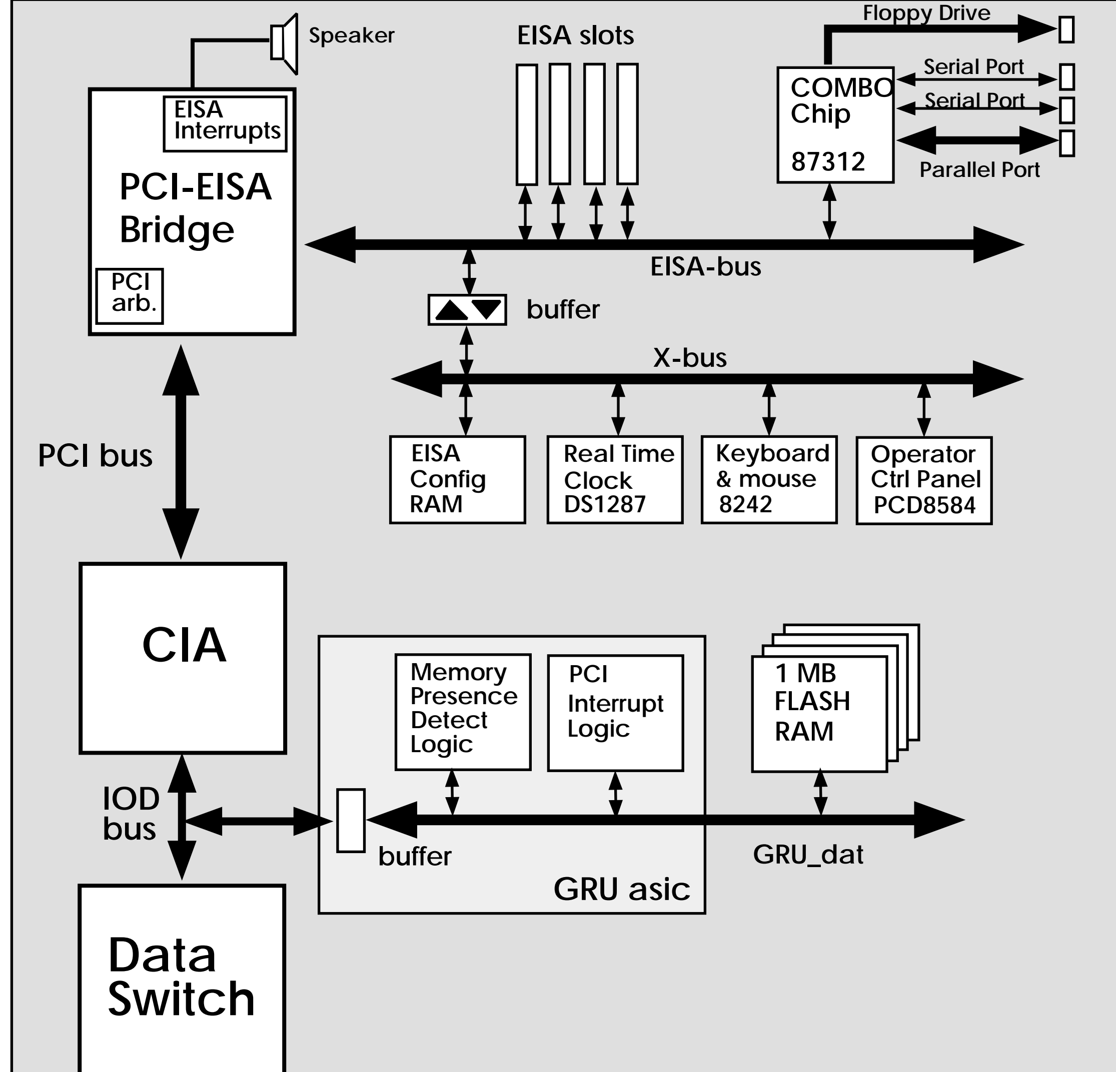
In the GRU, following reset, the presence detect logic begins shifting data bits from the module into the CACHE_CNFG register and the SCR. Control to the SIMMs to select the bits to be shifted in is output from the GRU on the GRU_DAT bus.

AlphaStation 600 PCI-EISA Bridge

The PCI-EISA chip bridge set consists of the PCI to EISA bridge chip, the PCEB (82375EB); and the EISA system controller, the ESC (82374EB). These chips are essentially a collection of peripheral chip designs (such as interrupt logic, timers, DMA control, arbiters, etc.) coerced into two packages.

Figure 2-12 shows the AlphaStation 600 system's standard I/O busses and devices, and indicates how the PCI-EISA bridge is used and what devices are attached to the X-bus.

Figure 2-12 AlphaStation 600 **System** Standard I/O Busses



[1] This functionality is not provided in version 1.0 of the GRU asic

ESC functionality

The stepping (revision) of the chip the AlphaStation 600 system will use at power-up is A-2. The AlphaStation 600 system’s use of the ESC chip is almost a standard implementation. Some of the specific details/features are:

- **Keyboard controller:** This is based on the standard 8242. The mouse interrupt (the so called ABFULL# signal) is **not** wired to the ABFULL# input. Software should disable its path to IRQ<12> via the ESC register CLKDIV bit<4>.
- **SERR and PERR:** Software must **NOT** enable these inputs to the NMI logic in the ESC.
- **Interrupt logic:** This logic is used for the EISA interrupts. The ESC should disable the connection of ABFULL# to irq<12>. The Real Time Clock is **not** wired to this interrupt logic; instead it is wired directly to an EV5 interrupt pin.
- **TOY - Real time clock:** This is based on the Dallas 1287 and is wired to the X-bus in the conventional manner.
- **Configuration RAM:** 8 KByte of non-volatile RAM is provided on the X-bus for the EISA configuration space.
- **BIOS:** There is **NO** BIOS ROM on the X-bus. Instead, the AlphaStation 600 system provides Flash ROM on the GRU_DAT bus. The GRU_DAT bus is "closer" to the CPU and thus a preferred location for this RAM (that is, serial ROM code needs to check less of the system before it is confident it can access the Flash RAM code).
- **Speaker:** This is provided to enable the operating systems to signal the operator audibly.
- **General Purpose device**. The Operator Control Panel interface chip (PCD8584) is wired to GPCS[0] -- the GPCS[0] signal is gated with CMD_L just as the SABLE implementation to allow consecutive access of the PCD8584 -- that is, software does not need to do anything special)

PCEB Functionality

The AlphaStation 600 system’s use of the PCEB is relatively standard.

PCI-to-EISA Address Decode

Subtractive decode must be used for the PCEB (negative decode does not work).

PC Compatibility Addressing and Holes

The PC architecture allows certain (E)ISA devices to respond to hardwired memory addresses. For example, a VGA graphics devices that has its frame buffer located in memory address region A0000 - BFFFF Main memory must be made inaccessible for such memory-mapped regions, and this inaccessible region is called a PC compatibility hole (or "hole" for short).

The EISA-PCI bridge provides access for (E)ISA devices to main memory (which is normally behind a HOST-PCI bridge) via positive address decode. The lower 512 MB of EISA address range is partitioned into many sub-segments which can be enabled by the MCSTOM, MCSTOH, MCSBOH, EADC1, EADC2 registers. These registers allow main memory "holes" to be created.

For more detail refer to the Intel 82375EB specification.

[1] This functionality is not provided in version 1.0 of the GRU ASIC

MEMCS#

PCI window 0 in the CIA can be enabled to accept the MEMCS# signal as the PCI memory decode signal. With this path enabled, the PCI window hit logic simply uses the MEMCS# signal (that is, if MEMCS# is asserted then a PCI window 0 hit occurs and the PCI DEVSEL signal is asserted).

PCI Arbitration

The PCEB does not allow the use of an external arbiter; the internal PCI arbiter must be used. The AlphaStation 600 system has one PCI slot more than the PCEB can handle. This problem is solved by providing a sub-arbiter on the system module for the 32 bit slots in conjunction with the PCEBs arbiter.

PCI Arbitration - Power-Up

The PCEB arbiter is initialized to provide round-robin arbitration but parks the host bridge (CIA) on the PCI. This means that the CIA normally is driving AD[31:0], C/BE[3:0] and PAR. The remaining 64-bit PCI signals are pulled up by resistors and need not be driven.

Figure 2-13 AlphaStation 600 System PCI arbiter scheme

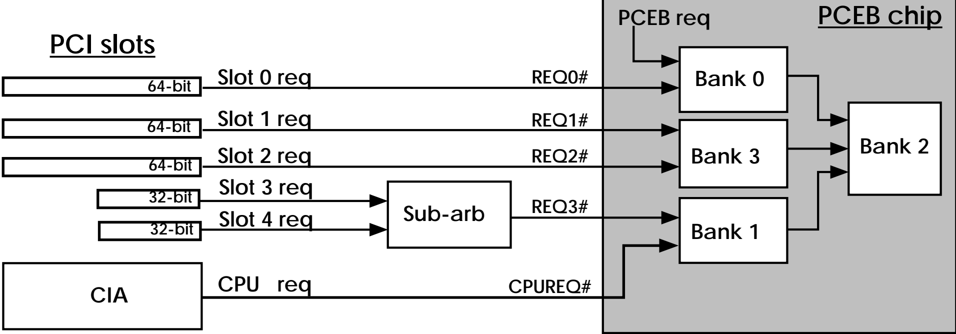


Table 2-4 Round-robin PCI Arbitration

Current Least Recently Used State					Highest Priority	Next Least Recently Used State				
Bank 2	Bank 0	Bank 3	Bank 1	Sub-arb		Bank 2	Bank 0	Bank 3	Bank 1	Sub-arb
Bank 0	x	Slot 1	x	x	Slot 2	Bank 3	same	Slot 2	same	same
Bank 0	x	Slot 2	x	x	Slot 1	Bank 3	same	Slot 2	same	same
Bank 3	x	x	Sub-arb	x	CPU	Bank 1	same	same	CPU	same
Bank 3	x	x	CPU	Slot 3	Slot 4	Bank 1	same	same	Sub-arb	Slot 4
Bank 3	x	x	CPU	Slot 4	Slot 3	Bank 1	same	same	Sub-arb	Slot 3
Bank 1	PCEB	x	x	x	Slot 0	Bank 0	Slot 0	same	same	same
Bank 1	Slot 0	x	x	x	PCEB	Bank 0	PCEB	same	same	same

Round-robin arbitration assumes all slots are requesting. If a bank or slot is not requesting, the priority is passed on to the next slot or bank that is requesting.

Data Buffering in the PCEB

The AlphaStation 600 system expects that the Line Buffer is enabled in the PCEB chip.

The latest errata from Intel requires that the PCEB Posted Write Buffer is disabled. This means that a PCI master requesting a PCI-to-EISA transfer is retried until the PCEB owns the EISA bus. Each PCI-to-EISA transfer must complete all the way to the EISA destination before the next transfer may begin. In other words, performance through to EISA will be abysmal (and the fewer EISA options installed the better).

3
AlphaStation 600 Addressing

Introduction

This chapter describes the mapping of the 40-bit processor physical address space into memory and I/O space addresses; it explains the translation of processor initiated addresses into a PCI address; and the translation of a PCI initiated address into physical memory address.

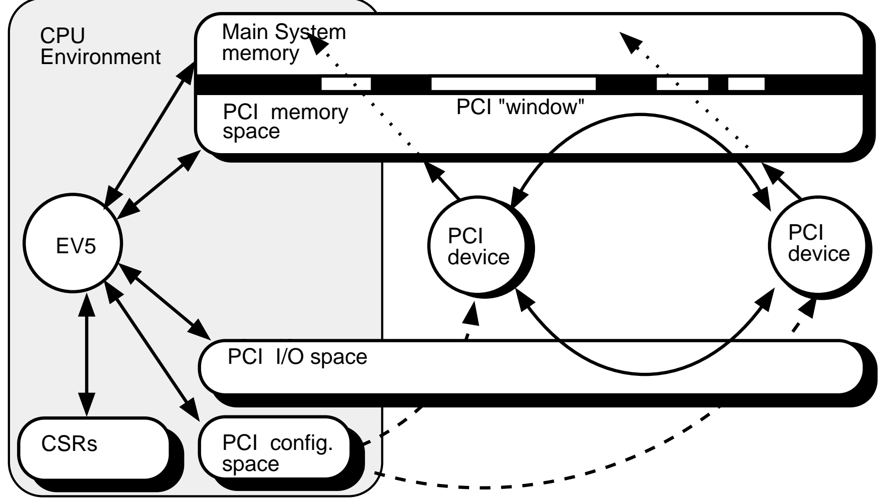
Topics include dense and sparse address spaces; scatter/gather address translation for DMA operations, PCI addressing, and EISA requirements.

Address Mapping Introduction

The EV5 address space is divided into two regions using physical address <39>: if clear, then EV5 access is to the cached memory space; if set, then the accesses are not cached. This uncached space is used in the AlphaStation 600 system to access memory-mapped I/O devices -- It does not support mailboxes.

The uncached space for the AlphaStation 600 system contains the CSRs, uncached memory access (for diagnostics), and the PCI address space. The PCI defines three physical address spaces: firstly, a 64-bit PCI memory space; secondly, a 4 GB PCI I/O space; and thirdly, a 256B per device, PCI configuration space. In addition to these three address spaces on the PCI, the CPU's uncached space is also used to generate PCI *Interrupt Acknowledge* cycles and PCI *Special* cycles.

Figure 3-1 Address Space Overview

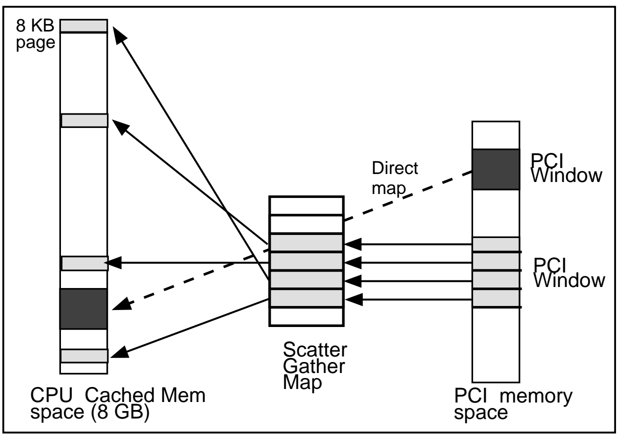


The CPU has visibility to the complete address space: it can access cached memory, CSRs and all the PCI memory, I/O and configuration regions (see Figure 3-1).

The PCI devices have a restricted view of the address space and can only access any PCI device through the PCI memory or PCI I/O space. They have no access to the PCI configuration space. Furthermore, the AlphaStation 600 system restricts access to the

system memory (for DMA operations) through four programmable "windows" (that is, memory regions) in the PCI memory space -- see Figure 3-1. Address "windows" are a PCI requirement (the window is defined via the Base register), and are implemented (either directly/indirectly via positive/subtractive decode) by all PCI devices.

DMA access to the system memory is achieved in one of two ways: either "directly-mapped" by concatenating an offset to a portion of the PCI address; or "virtually" through a scatter/gather translation map. This scatter/gather map allows any 8 KB PCI memory address region (page) to be redirected to any 8 KB cached memory page, as shown below.



PCI Addressing

The AlphaStation 600 system generates 32-bit PCI addresses but accepts both 64-bit address (DAC[1]) cycles and 32-bit PCI address (SAC[2]) cycles. However, the 64-bit addressing support is constrained as follows:

- The least-significant 40-bits of the PCI address are used in the window comparison logic, the remaining PCI address<63:40> must be zero;
- Only one of the four PCI windows can be programmed to accept a 64-bit (DAC) PCI address; the remaining three windows only accept 32-bit (SAC) address cycles.
- The AlphaStation 600 system does not generate DAC cycles; it only accepts DAC cycles. With a 4 GB DAC window and a 4 GB SAC window(s), a PCI agent can access all 8 GB of memory supported by the AlphaStation 600 system chip set.

CPU Address Space

Figure 3-3 shows an overview of the CPU address space and Table 3-2 defines the address regions in more detail. Figure 3-2 shows how the CPU address map translates to the PCI address space; and also shows how the PCI access the CPU memory space via DMAs. Note how the PCI memory space is double mapped via dense and sparse space.

The rational behind the CPU I/O address map is as follows:

- Limit the number of address pins sent to the pin constrained CIA.
- Provide 4 GB of dense[3] space to completely map the 32-bit PCI memory space.

[1] Double Address Cycle (PCI 64-bit address transfer) -- only used if address <63:32> are non-zero.
[2] Single Address Cycle -- used for 32-bit PCI addresses, or if <63:32> are zero for a 64-bit address.
[3] Dense and Sparse -space are explained later in this chapter.

- Provide abundant PCI sparse[1] memory space since sparse-space has byte granularity and is the safest memory-space to use (for example, no prefetching). Furthermore, the larger the space the less likely software will need to dynamically relocate the sparse space segments. The main problem with sparse space is that it is wasteful of CPU address space (for example, 16 GB of CPU address space maps to 512 MB of PCI sparse space).

  The AlphaStation 600 system provides 3 PCI memory, sparse-space regions, allowing 704 MB of total sparse memory space. The three regions are relocatable via the HAE_MEM CSR, and the simplest configuration allows for 704 MB of contiguous memory space.

  — 512 MB region which may be located in any naturally-aligned 512 MB segment of the PCI memory space. Software may find this region sufficient for their needs and can ignore the remaining two regions.

  — 128 MB regions which may be located on any naturally-aligned 128 MB segment of the PCI memory space.

  — 64 MB region which may be located on any naturally-aligned 64 MB segment of the PCI memory space.

- Limit the PCI I/O space to sparse space: although the PCI I/O space can handle 4 GB, the Pentium chip can only access 64 KB. Consequently, most PCI devices will not exceed 64 KB for the foreseeable future. The AlphaStation 600 system provides 64 MB of sparse I/O space because the hardware decode is faster.

  The AlphaStation 600 system provides two PCI IO sparse-space regions: region A, which is 32 MB and is fixed in PCI segment 0-32 MB; and region B, which is also 32 MB, but is relocatable using the HAE_IO register.

CPU Address <38:35>

Pin constraints on the CIA ASIC prevent CPU address <38:35> from being used. The software must ensure that CPU address <38:35> is zero (strictly speaking, even parity); otherwise the CIA will induce a parity error interrupt.

[1] Dense and Sparse -space are explained later in this chapter.

Figure 3-2 CPU and DMA Reads and Writes

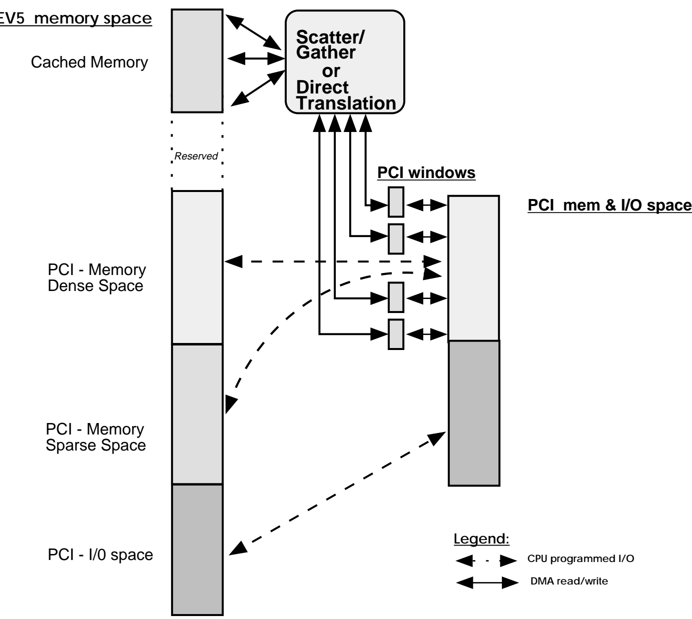


Figure 3-3 CPU Addressing

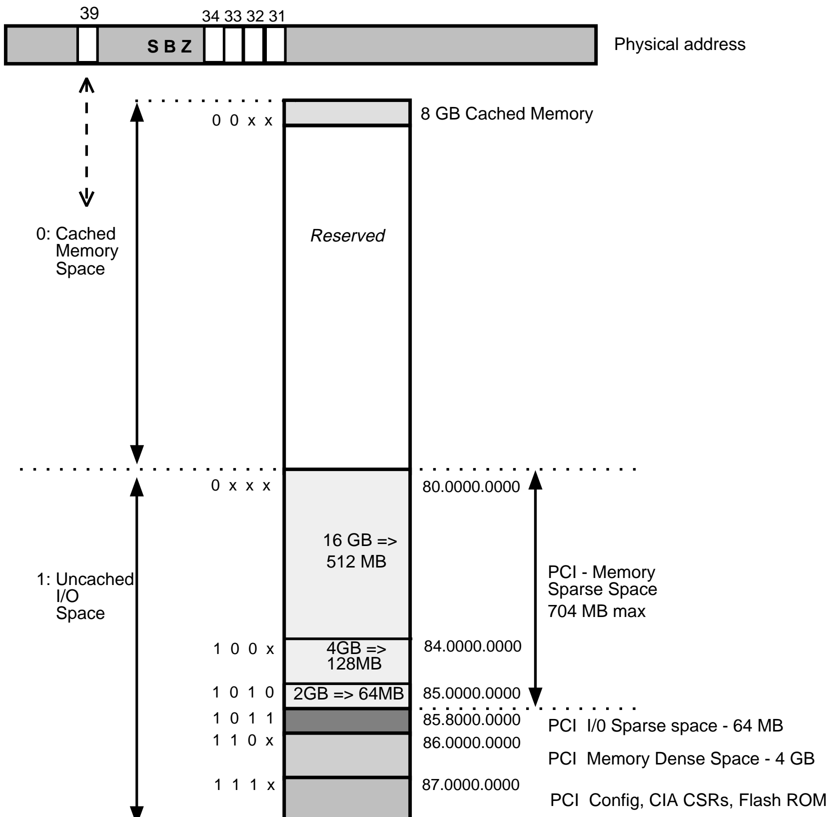


Table 3-1 AlphaStation 600 Series CPU Address Space

CPU address	Size ()	Selection
00.0000.0000 -- 01.FFFF.FFFF	8	Main memory
80.0000.0000 -- 83.FFFF.FFFF	16	PCI Memory, 512 MB -- Sparse Space - Region 0
84.0000.0000 -- 84.FFFF.FFFF	4	PCI Memory, 128 MB -- Sparse Space - Region 1
85.0000.0000 -- 85.7FFF.FFFF	2	PCI Memory, 64 MB -- Sparse Space - Region 2
85.8000.0000 -- 85.BFFF.FFFF	1	PCI I/O space, 32 MB -- Sparse Space - Region A
85.C000.0000 -- 85.FFFF.FFFF	1	PCI I/O space, 32 MB -- Sparse Space - Region B
86.0000.0000 -- 86.FFFF.FFFF	4	PCI Memory, 4 GB -- Dense Space
87.0000.0000 -- 87.1FFF.FFFF	0.5	PCI Configuration, -- Sparse Space
87.2000.0000 -- 87.3FFF.FFFF	0.5	PCI Special/Int. Ack. -- Sparse Space
87.4000.0000 -- 87.4FFF.FFFF	0.25	CIA Main CSRs, -- Pseudo Sparse[1]
87.5000.0000 -- 87.5FFF.FFFF	0.25	CIA Memory control CSRs, -- Pseudo Sparse[1]
87.6000.0000 -- 87.6FFF.FFFF	0.25	CIA PCI address translation, -- Pseudo Sparse[1]
87.7000.0000 -- 87.7FFF.FFFF	0.25	Reserved
87.8000.0000 -- 87.FFFF.FFFF	2	Flash ROM, GRU asic CSRs -- Pseudo Sparse[1]
Note 1: Pseudo sparse space is a hardware-specific, restricted version of sparse-space		

Table 3-2 AlphaStation 600 Series Address Map

CPU address	Description
Bits 39, 34 33, 5: [0 | SBZ | 0 | 0 | Memory address |]	**AlphaStation 600 main memory - 8 GB** All accesses are cache block (64B) aligned and are cached by the EV5. Istream and Dstream access.
Bit 34: [1 | SBZ | 0 | PCI Memory LW addr. | Size | 000] Size = used to generate byte enables and PCI address <2:0>	**PCI sparse mem space - 512 MB** **Region 1** Uncached EV5 access. Byte, word, tri-byte, LW, QW read/write allowed. No read prefetching.
Bits 39, 34 33 32 31, 7 6 5 4 3 2 1 0: [1 | SBZ | 1 | 0 | 0 | PCI Memory LW addr. | Size | 000]	**PCI sparse mem space - 128 MB** **Region 2**
Bits 39, 34 33 32 31 30, 7 6 5 4 3 2 1 0: [1 | SBZ | 1 | 0 | 1 | 0 | PCI I/O LW addr. | Size | 000]	**PCI sparse mem space - 64 MB** **Region 3**
Bits 39, 34 33 32 31 30, 7 6 5 4 3 2 1 0: [1 | SBZ | 1 | 0 | 1 | 1 | 0 | PCI I/O LW addr. | Size | 000]	**PCI I/O sparse space - 32 MB** **Region A** Uncached EV5 access. Byte, word, tri-byte, LW, QW read/write allowed. No read prefetching. Used to address (E)ISA devices.
Bits 39, 34 33 32 31 30, 7 6 5 4 3 2 1 0: [1 | SBZ | 1 | 0 | 1 | 1 | 1 | PCI I/O LW addr. | Size | 000]	**PCI I/O sparse space - 32 MB** Region B Relocatable via HAE_IO
Bits 39, 34 33 32 31, 7 6 5 4 3 2 1 0: [1 | SBZ | 1 | 1 | 0 | PCI Memory LW addr. | 00]	**PCI dense memory space - 4 GB** Uncached EV5 access Used for devices with access granularity greater or equal to a LW. Read Prefetching is allowed, and thus reads can have no side effects.
Bits 39, 34 33 32 31, 28, 7 6 5 4 3 2 1 0: [1 | SBZ | 1 | 1 | 1 | (bits 31–28) | address | Size | 000]	**PCI configuration space.** Uncached EV5 access Sparse space Byte, word, tri-byte, LW, QW read/write No read prefetching
(see decode of bits 31–28 below)	**CIA CSRs (including Flash ROM)** Uncached EV5 access. The CSRs addresses are chosen for hardware convenience. See CSR section for specific addresses.

CPU addr 31 30 29 28		Size GB	
0 0 0	PCI	0.5	PCI Configuration Space
0 0 1		0.5	PCI IACK/Special cycle
0 1 0 0	CIA CSRs	0.25	CIA main CSRs
0 1 0 1		0.25	Memory Control
0 1 1 0		0.25	Scatter/Gather Translation
0 1 1 1		0.25	reserved
1 x x x	Misc.	2.0	Flash ROM, GRU asic CSRs

Cacheable Memory Space

This is located in the range: 00 0000 0000 to 01 FFFF FFFF.

The AlphaStation 600 chip set recognizes the first 8 GB to be in cacheable memory space. The block size is fixed at 64B. Read and Flush commands to the EV5 caches occur for DMA traffic.

PCI Dense Memory Space

This is located in the range: 86 0000 0000 to 86 FFFF FFFF.

PCI dense memory space is typically used for "memory-like" data buffers such as a video frame buffer or a non-volatile RAM. Dense space does not allow byte or word access unlike sparse space (see later), but enjoys the following advantages (over sparse space):

- **Memory model:** some software, for example, WNT default graphics routines, require memory-like accesses. It cannot use sparse space addressing, since it requires accesses on the PCI bus to be at adjacent Alpha addresses, instead of being widely separated as in sparse space. As as result, if the user-mode driver uses sparse-space for its frame-buffer manipulation, it cannot "hand over" the buffer to the common Windows NT graphics code.
- **Higher bus bandwidth**: PCI bus burst transfers are not usable in sparse space apart from a two-longword burst for quadword writes. Dense space is defined to allow both burst reads and writes.
- **Efficient read/write buffering:** In sparse space, separate accesses use separate read or write buffer entries. Dense space allows separate accesses to be "collapsed" in read and write buffers (this is exactly what the EV5 does).
- **Few memory barriers (MBs):** In general, sparse space accesses are separated by memory barriers to avoid read/write buffer collapsing. Dense space accesses only require barriers when explicit ordering is required by the software.

Dense space is provided for CPU addresses accessing PCI memory space only, and not for accessing PCI IO space. Dense space has the following characteristics:

- There is a one-to-one mapping between CPU addresses and PCI addresses: a longword address from the CPU will map to a longword on the PCI with no shifting of the address field. Hence the term dense space (as compared to sparse space, which maps a large chunk of CPU memory space (for example, 32B) to a byte on the PCI -- see section on the PCI sparse space).
- The concept of dense space (and sparse space) is only applicable to generated addresses. There is no such thing as dense space (or sparse space) for PCI generated address.
- Byte or word accesses are **NOT** possible in cacheable space. The minimum access granularity is a longword on writes and a quadword on reads. The maximum transfer length is 32 bytes (performed as a burst of 8 longwords on the PCI). Any combination of longwords may be valid on writes. Valid longwords surrounding an invalid longword(s) (called a "hole") are required to be handled correctly by all PCI devices. The AlphaStation 600 system allows such combinations to be issued.
- Reads will always be performed as a burst of two or more longwords on the PCI because the minimum granularity is a quadword. The processor can request a longword but the AlphaStation 600 system will always fetch a quadword that is, prefetch a longword. Hence this space cannot be used for devices which have read side effects. Although a longword may be prefetched the prefetch buffer is not treated as a cache and thus coherency is not an issue. Note that a quadword read is not atomic on the PCI -- that is, the target device is at liberty to force a retry after the first longword of

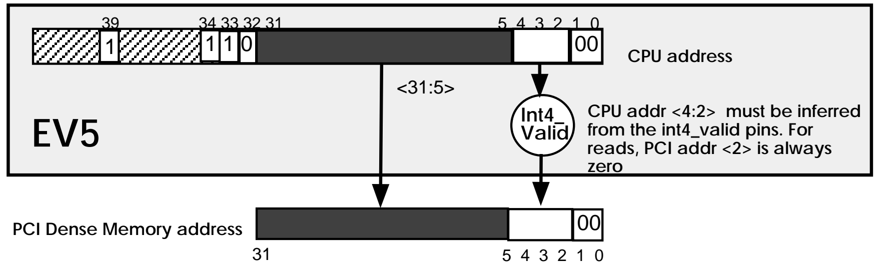
data is sent, and then allow another PCI device in[1].

The EV5 merges uncached reads up to 32B maximum. The largest dense space read is thus 32B from the PCI bus.

- Writes to this space are buffered in the EV5 chip. The AlphaStation 600 system supports a burst length of 8 on the PCI, corresponding to 32B of data. In addition, the CIA ASIC provides four 32B write buffers to maximize IO write performance. These four buffers are strictly ordered (See Chapter 11, System Coherency, for information on coherency issues).

The address generation in dense space is as follows:

Figure 3-4 Dense Space Address Generation



CPU address<31:5> is directly sent out on PCI address <31:5>. CPU address<4:2> is not sent out of the EV5 and instead is inferred from the Int4_valid pins; however, for software concerns, this is a mere implementation detail. PCI address<4:3> is a copy of CPU address<4:3>. For a read transaction, the PCI_address<2> is zero (that is, minimum read resolution in uncached space is a quadword). For a write PCI address<2> equals CPU address<2>.

PCI Sparse Memory Space

The AlphaStation 600 system provides three regions of contiguous CPU address space which maps to PCI sparse memory space. The total CPU range is from 80 0000 0000 to 85 7FFF FFFF.

The Alpha instruction set can express only aligned longword and quadword data references. The PCI bus requires the ability to express byte, word, tri-byte, longword (doubleword) and quadword references. Furthermore, Intel processors are also capable of generating unaligned references, and it should be possible to emulate the resulting PCI transactions to insure compatibility with PCI devices designed for Intel-based systems.

For an Alpha architecture, it is necessary to encode the size of the data transfer (byte, word, etc.) and the byte enables in the CPU address. Address bits <6:3> are used for this purpose, leaving the remaining bits <31:7> This loss of address bits has resulted in a "sparse" 22 GB CPU 32-bit address space that maps to only 704 MB of address space on the PCI.

The rules for accessing sparse space are as follows:

- Sparse space supports all the byte encodings which may be generated in an Intel system to ensure compatibility with PCI devices/drivers. The results of some references are not explicitly defined -- these are the missing entries in Table 3-3 (for example, word size with address<6:5> = 11). The hardware will complete the reference but the reference is not required to produce any particular result nor will the AlphaStation 600 system report an error. The error strategy is defined in the Hardware Exceptions and Interrupts chapter.

[1] The AlphaStation 600 system does not drive the PCI LOCK signal and thus cannot ensure atomicity. This is true of all current alpha platforms.

- Software must use longword load or store instructions (LDL/STL) to perform a reference which is of longword length or less on the PCI bus. The bytes to be transferred must be positioned within the longword in the correct byte lanes as indicated by the PCI byte enables. The hardware will do no byte shifting within the longword. Quadword loads and stores must only be used to perform a quadword transfer. Use of STQ/LDQ instructions for any other references will produce unpredictable results.
- Read-ahead (prefetch) is not performed in sparse space by the AlphaStation 600 system hardware since the read-ahead may result in detrimental side-effects.
- Programmers are required to insert MB instructions between sparse space accesses to prevent collapsing in the EV5 write buffer. However, this is not always required: for instance, consecutive sparse space addresses will be separated by 32B (and will not be collapsed by the EV5).
- Programmers are required to insert MB instructions if the sparse space address synonyms to a dense space address (that is, if ordering/coherency is to be maintained).
- The encoding of the EV5 address for sparse space read accesses to PCI space is shown in Table 3-3. An important point to note is that CPU address[33..5] are directly available from the processor chip pins. On read transactions the processor sends out address bits [4.3] indirectly on the Int4_valid pins. CPU address [2:0] are required to be zero: accesses with [2:0] non-zero will produce unpredictable results.
- The relation between Int4_valid[3:0] and CPU address[4:3] for a sparse space write is shown below. The important point is that all other int4_valid patterns will produce unpredictable results -- for example, as a result of collapsing in the EV5 write buffer; or by issuing a STQ when a STL was required.

EV5 Data cycle	Int4_Valid3:0>	Address<4:3>
First	00 01	0 0
	00 10	0 0
	01 00	0 1
	10 00	0 1
Second	00 01	1 0
	00 10	1 0
	01 00	1 1
	10 00	1 1
	11 00 (STQ)	1 1

Note: (1) All other Int4_valid patterns result in unpredictable results.
(2) Only one valid STQ case is allowed.

Table 3-3 defines the low-order PCI sparse memory address bits. CPU address<7:3> is used to generate the length of the PCI transaction in bytes, the byte enables, and address bits<2:0>. CPU address<30:8> correspond to the quadword PCI address and is sent out on PCI address <25:3>.

The high-order PCI address bits <31:26> are obtained from either the *Hardware Extension Register (HAE_MEM)* or the CPU address depending on sparse space regions, as shown in Table 3-4. The HAE_MEM is described in the next section and is a CSR in the CIA ASIC. Figures 3-5 through 3-7 shows the mapping for the three regions.

Table 3-3 PCI Memory Sparse Space Read/Write Encodings

Size	CPU_Addr <4:3>	Byte Offset CPU_Addr <6:5>	CPU Instruction allowed	PCI Addr <2:0> see notes	PCI Byte Enable	Data in Register byte lanes 63 31 0
Byte	00	00	LDL, STL	A<7>,00	1110	
		01		A<7>,00	1101	
		10		A<7>,00	1011	
		11		A<7>,00	0111	
Word	01	00	LDL, STL	A<7>,00	1100	
		01		A<7>,00	1001	
		10		A<7>,00	0011	
Byte	10	00	LDL, STL	A<7>,00	1000	
		01		A<7>,00	0001	
Long-Word	11	00	LDL, STL	A<7>,00	0000	
Quad-Word	11	11	LDQ, STQ	000	0000	

Note: A<7> = CPU_address<7>.
Byte Enable set to 0 indicates that byte lane carries meaningful data.
In PCI sparse memory space, PCI Address<1:0> is always zero.
Missing entries (for example, word size with CPU address <6:5> = 11) enjoy UNPREDICTABLE results.

Table 3-4 High-order Sparse Space bits

CPU address	ROM	PCI_ Address 31	30	29	28	27	26
80.0000.0000 - 83.FFFF.FFFF	1	HAE_MEM<31>	HAE_MEM<30>	HAE_MEM<29>	CPU<33>	CPU<32>	CPU<31>
84.0000.0000 - 84.FFFF.FFFF	2	HAE_MEM<15>	HAE_MEM<14>	HAE_MEM<13>	HAE_MEM<12>	HAE_MEM<11>	CPU<31>
85.0000.0000 - 85.7FFF.FFFF	3	HAE_MEM<7>	HAE_MEM<6>	HAE_MEM<5>	HAE_MEM<4>	HAE_MEM<3>	HAE_MEM<2>

Figure 3-5 CI Memory Sparse Space Address Generation - Region 1

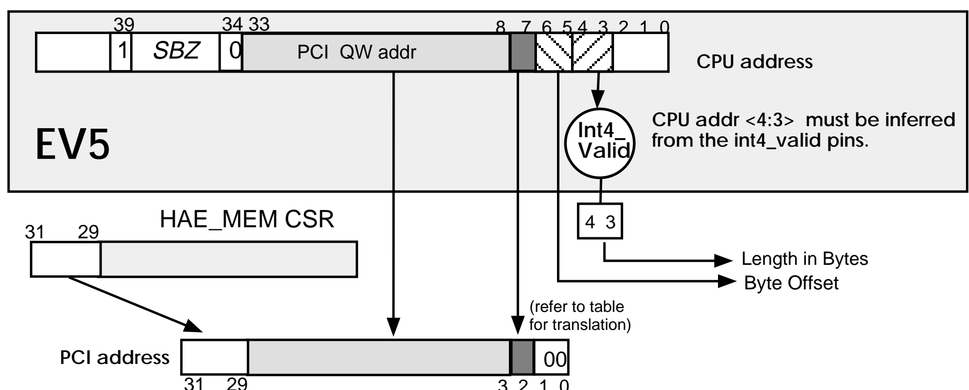


Figure 3-6 PCI Memory Sparse Space Address Generation - Region 2

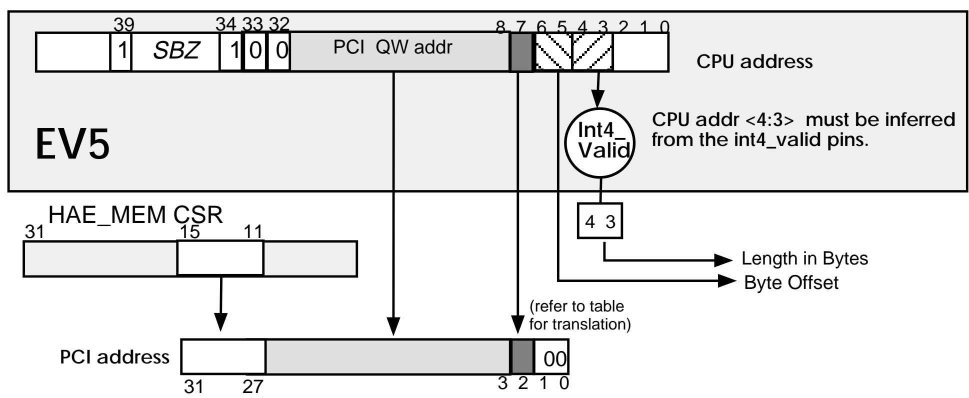
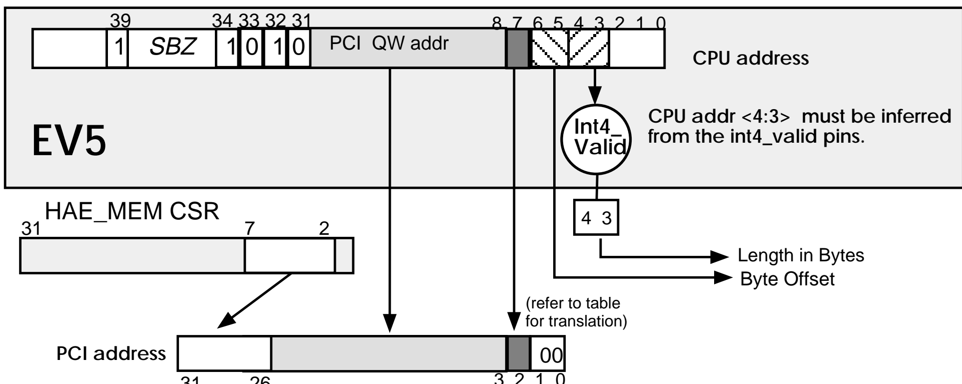


Figure 3-7 PCI Memory Sparse Space Address Generation - Region 3



Hardware Extension Registers (HAE)

In sparse space, CPU_Address[7:3] are "wasted" on encoding byte enables, size and the low-order PCI address<2:0>. This means that there are now 5 fewer address bits available to generate the PCI physical address. This problem is solved in the EV4 based systems (Sable, APECS, and LCalpha) by a Hardware Extension register (HAE[1]), which is used to provide the missing high-order bits. The HAE registers are intended to be system specific and are not defined by the *Reference Implementation.* The expectation is that the HAE registers are set by POST[2] software and thereafter never modified.

Compared to the EV4, the EV5 provides six extra physical address bits <39:34>. These extra bits could be use to back-fill the "lost" sparse space bits. However, the CIA ASIC is pin-constrained and the high-order address bits <38:35> are not available.

Furthermore, other EV5 platforms use these high-order bits in different ways (encoding multiple PCI ports for instance), and so for easier software portability these bits are best not used. The Sable/APECS/LC-alpha designs effectively provide two address regions for sparse PCI memory access: one region has CPU address bits <31:29> = 0 (lower 16 MB of the PCI sparse address range) which is not relocated; and a second region, when bits <31:29> are non-zero, which is relocated using bits <31:29> of the HAE register.

The AlphaStation 600 system provides more PCI sparse memory space than the other designs and consequently has a different address decode scheme: the AlphaStation 600 system provides three sparse space PCI memory regions. Furthermore, It allows all three sparse space regions to be relocated via bits in the HAE_MEM register. This provides software with far greater flexibility.

Finally, to complete this section on HAE registers, we will note that a similar technique is used for the PCI IO sparse space. Two regions are provided: region A addresses the lower 32 MB of PCI IO space and is never relocated. This region will be used to address the (E)ISA devices. Region B, is used to address a further 32 MB of PCI IO space and is relocatable using HAE_IO. More details will be found in Chapter 7, Control and Status Registers.

1 generally pronounced *hay*
2 Power-on Self Test (PCI-speak for firmware).

PCI Sparse I/O Space

PCI sparse I/O space is located in the range: 85 8000 0000 to 85 FFFF FFFF and has similar characteristics to the PCI Sparse Memory Space. This 2 GB CPU address segment maps to two 32 MB regions of PCI I/O address space. A read or write to this space causes a PCI I/O read or write command.

The high order PCI address bits are handled as follows:

- **Region A:** This region has CPU address<34:30> = 10110 and addresses the lower 32 MB of PCI sparse I/O space; thus PCI address<31:25> isset to zero by the hardware (see top of Figure 3-9). This region is used for (E)ISA addressing (the EISA 64 KB I/O space cannot be relocated).
- **Region B:** This region has CPU address<34:30> = 10111 and addresses a relocatable 32 MB of PCI sparse I/O space. This 32 MB segment is relocated by assigning PCI address <31:25> to equal HAE_IO<31:25>.

The remainder of the PCI I/O address is formed in the same way for both regions.

- PCI address<24:3> are derived from CPU address<29:8>
- PCI_address<2:0> are defined in Table 3-5.

The (E)ISA devices have reserved the lower 64 KB of this space. Hence all PCI devices should be relocated above this region. The four AlphaStation 600 system (E)ISA slots are hardwired through the AEN* allocating 4 KB per slot (as per EISA standard) -- the first slot is reserved for the EISA system board (that is, X-bus addressing, Interrupt controller, etc). Figure 3-8 shows the PCI and (E)ISA I/O map for the AlphaStation 600 system.

Figure 3-8 AlphaStation 600 System PCI and (E)ISA I/O Map

CPU address	PCI addr	Range (KB)	Selection	
85.8000.0000	0000.0000	0 - 4	EISA system board (X-bus, I/O ports, etc)	EISA 0-64 KB region
85.8002.0000	0000.1000	4 - 8	EISA slot 1	
85.8004.0000	0000.2000	8 - 12	EISA slot 2	
85.8006.0000	0000.3000	12 - 16	EISA slot 3	
85.8008.0000	0000.4000	16 - 20	EISA slot 4	
85.800A.0000	0000.5000	20 - 24	Reserved	
85.800C.0000	0000.6000	24 - 28	Reserved	
85.800E.0000	0000.7000	28 - 32	Reserved	
85.8010.0000	0000.8000	32 - 36	Reserved	
85.8012.0000 : 85.801F.FFFF	0000.9000 : 0000.FFFF	36 - 64	Reserved	
85.8020.0000 : : 85.BFFF.FFFF	0001.0000 : : 01FF.FFFF	64 KB to 32 MB	PCI I/O area -- fixed	PCI region
85.C000.0000 : : 85.FFFF.FFFF	0200.0000 : : 03FF.FFFF	32 MB	PCI I/O area -- relocatable	

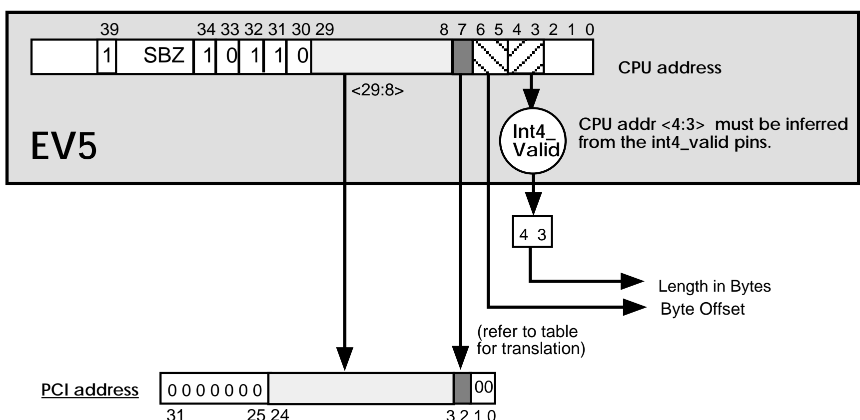
WARNING: *A quadword access to the PCI sparse I/O space will result in a 2 longword burst on the PCI. However, PCI devices may not support bursting in I/O space.*

Table 3-5 PCI Sparse I/O Space Read/Write Encodings

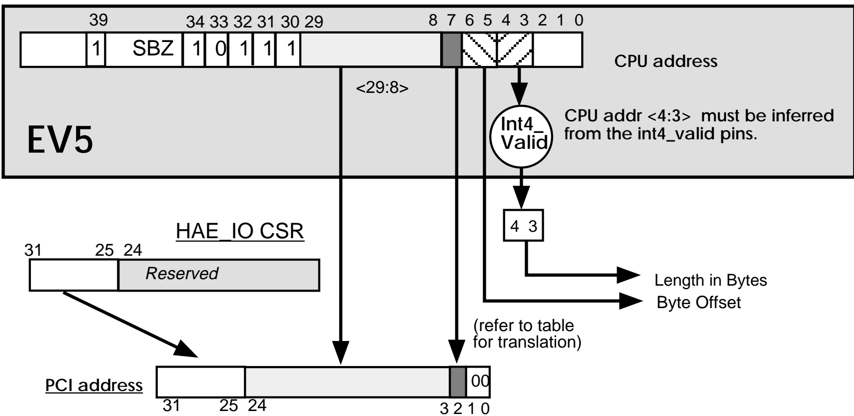
Size		Byte Offset	CPU Instruction allowed		PCI Addr <2:0> see notes	PCI Byte Enable	Data in Register byte lanes 63 31 0
	CPU_Addr <4:3>	CPU_Addr <6:5>					
Byte	00	00	LDL, STL		A<7>,00	1110	
		01			A<7>,01	1101	
		10			A<7>,10	1011	
		11			A<7>,11	0111	
Word	01	00	LDL, STL		A<7>,00	1100	
		01			A<7>,01	1001	
		10			A<7>,10	0011	
Tri-Byte	10	00	LDL, STL		A<7>,00	1000	
		01			A<7>,01	0001	
Long-Word	11	00	LDL, STL		A<7>,00	0000	
Quad-Word	11	11	LDQ, STQ		000	0000	

Note: A<7> = CPU_address<7>.
Byte Enable set to 0 indicates that byte lane carries meaningful data.
Missing entries (for example, word size with CPU address <6:5> = 11) have UNPREDICTABLE results.

Figure 3-9   PCI sparse I/O Space Address Translation



Address translation for lower 32 MB of PCI sparse IO space



Address translation for remainder of PCI sparse IO space

PCI Configuration Space

This is located in the range: 87 0000 0000 to 87 1FFF FFFF. Software is advised to clear CIA_CTRL<fill_err_en> when probing for PCI devices via configuration space reads. This will prevent the CIA from generating an ECC error if no device responds to the configuration cycle (and garbage data is on the PCI bus).

A read or write access to this space causes a Configuration read or write cycle on the PCI. There are two classes of targets, which are selected based on the value of the CFG CSR.

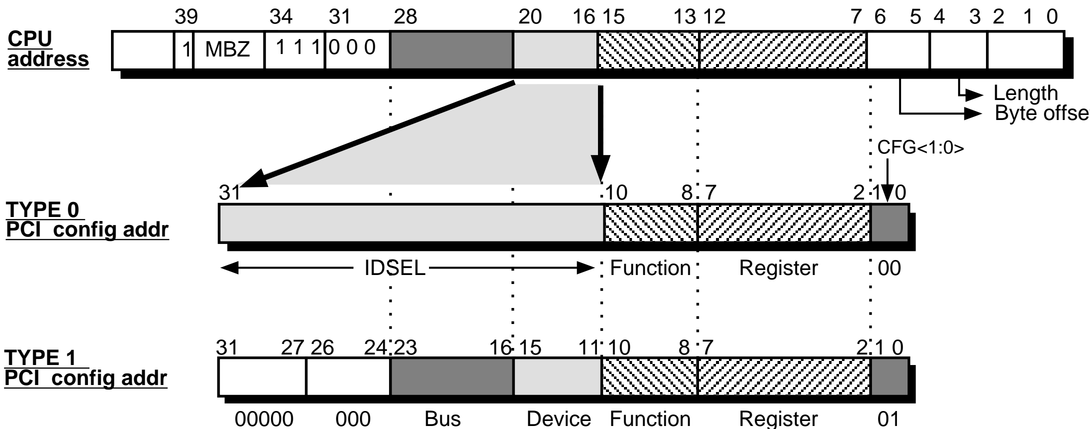
- **Type 0:** These are targets on the primary 64-bit the AlphaStation 600 system PCI bus. These are selected by making the CSR CFG<1:0> = 0.
- **Type 1:** These are targets on the secondary 32-bit AlphaStation 600 system PCI bus (that is, behind a PCI-PCI bridge). These are selected by making CFG<1:0> = 1.
- CFG<1:0> = 10 or 11 are reserved (by the PCI spec).

Software must first program the CFG register before running a configuration cycle. Note that the AlphaStation 600 system uses the CFG<1:0> instead of unused CPU address bits <38:35> to be compatible with Sable and the APECS chip set.

Sparse address decoding is used. CPU address<6:3> is used to generate both the length of the PCI transaction in bytes and the byte enables. PCI Address bits <1:0> are obtained

from CFG <1:0>. CPU address<28:7> corresponds to PCI address<23:2> and provides the configuration command information (for example, which device to select). The high-order PCI address bits <31:24> are always zero.

Figure 3-10 PCI Configuration Space Definition



Peripherals are selected during a PCI configuration cycle if: (a) their IDSEL pin is asserted; (b) the PCI bus command indicates a configuration read or write; and (c) address bits <1:0> are 00. Address bits <7:2> select a dword (longword) register in the peripheral's 256-Byte configuration address space. Accesses can use byte masks.

Peripherals that integrate multiple functional units (for example, SCSI and ethernet) can provide configuration space for each function. Address bits <10:8> can be decoded by the peripheral to select one of eight functional units.

PCI Address bits <31:11> are available to generate the IDSELs (note that IDSELs behind a PCI-PCI bridge are determined from the Device field encoding of a type 1 access). The IDSEL pin of each device is connected to a unique PCI address bit from the set <31:11>. The binary value of CPU address <20:16> is used to select which PCI address <31:11> is asserted, as follows:

CPU Address <20:16>	PCI address <31:11> -- IDSEL
00000	0000 0000 0000 0000 0000 1
00001	0000 0000 0000 0000 0001 0
00010	0000 0000 0000 0000 0010 0
00011	0000 0000 0000 0000 0100 0
:	:
10011	0100 0000 0000 0000 0000 0
10100	1000 0000 0000 0000 0000 0
10101 : 11111	0000 0000 0000 0000 0000 0 (No device selected)

WARNING: If a quadword access is specified for the configuration cycle then the least significant bit of the *register number* field (that is, PCI address<2>) **must be zero** -- that is, quadword accesses must access quadword aligned registers.

If the PCI cycle is a configuration read or write cycle but the PCI address<1:0> are 01 (that is, a type 1 transfer), then a device on a hierarchical bus is being selected via a PCI/PCI bridge. This cycle is accepted by the PCI/PCI bridge for propagation to its secondary PCI bus. During this cycle <23:16> select a unique bus number, and address <15:8>

selects a device on that bus (typically decoded by the PCI/PCI bridge to generate the secondary PCI address pattern for IDSEL), and address <7:2> selects a Dword (longword) in the devices configuration space.

Figure 3-11 PCI Configuration Space Read/Write Encodings

Size	CPU_Addr <4:3>	Byte Offset CPU_Addr <6:5>	CPU Instruction Allowed	PCI Addr <1:0>	PCI Byte Enable	Data in Register Byte Lanes 63 31 0
Byte	00	00	LDL, STL	CFG<1:0>	1110	
		01		CFG<1:0>	1101	
		10		CFG<1:0>	1011	
		11		CFG<1:0>	0111	
Word	01	00	LDL, STL	CFG<1:0>	1100	
		01		CFG<1:0>	1001	
		10		CFG<1:0>	0011	
Tri-Byte	10	00	LDL, STL	CFG<1:0>	1000	
		01		CFG<1:0>	0001	
Long-Word	11	00	LDL, STL	CFG<1:0>	0000	
Quad-Word	11	11	LDQ, STQ	CFG<1:0>	0000	

Note: A<7> = CPU_address<7>.
Byte Enable set to 0 indicates that byte lane carries meaningful data.
Missing entries (for example, word size with CPU address <6:5> = 11) have UNPREDICTABLE results.

Each PCI/PCI bridge can be configured via PCI configuration cycles on its primary PCI interface. Configuration parameters in the PCI/PCI bridge will identify the bus number for its secondary PCI interface, and a range of bus numbers that may exist hierarchically behind it.

If the bus number of the configuration cycle matches the bus number of the bridge chips secondary PCI interface, it will accept the configuration cycle, decode it, and generate a PCI configuration cycle with address <1:0> = 00 on its secondary PCI interface. If the bus number is within the range of bus numbers that may exist hierarchically behind its secondary PCI interface, the bridge chip passes the PCI configuration cycle on unmodified (address <1:0> = 01). It will be accepted by a bridge further downstream.

Figure 3-12 shows the AlphaStation 600 system's PCI hierarchy. The IDSEL lines are significant in Type 0 Configuration cycles, and the PCI nodes are connected as tabulated in Table 3-6. (The choice of address bit assignments to the IDSEL lines was because of a module ECO)

Figure 3-12 AlphaStation 600 System PCI Bus Hierarchy

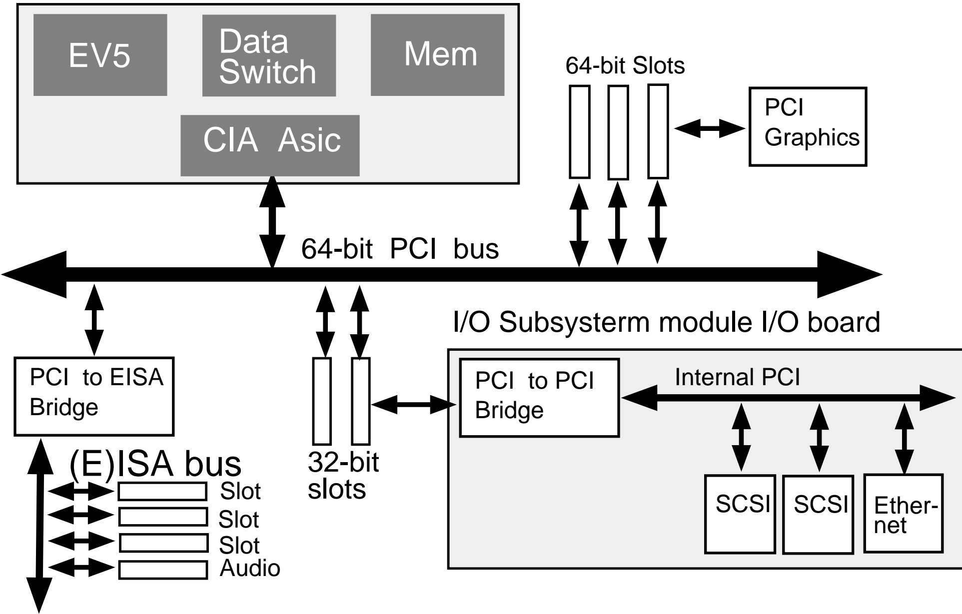


Table 3-6 Primary 64-bit PCI Slot to IDSEL Mapping

SLOT	PCI Address used as IDSEL	Slot location on system	Module Reference	PCI Slot Number (Firmware)
64-bit PCI slot 0	<18>	Next to Memory board	J11	7
64-bit PCI slot 1	<23>		J10	12
64-bit PCI slot 2	<22>		J9	11
32-bit PCI slot 3	<19>		J8	8
32-bit PCI slot 4	<20>	Next to EISA slots	J7	9
PCI/EISA Bridge	<21>			10

PCI Special/Interrupt Cycles

PCI Special/Interrupt Cycles are located in the range 87 2000 0000 to 87 3FFF FFFF.

The Special cycle command provides a simple message broadcasting mechanism on the PCI. The Intel processor uses this cycle to broadcast processor status; but in general it may be used for logical sideband signaling between PCI agents.

The Special cycle contains no explicit destination address, but is broadcast to all agents. The AlphaStation 600 system will drive all zero's as the Special cycle address. Each receiving agent must determines if the message contained in the data field is applicable to it.

A write access in the range 87.2000.0000 to 87.3FFF.FFFF causes a special cycle on the PCI. The CPU's write data will be passed unmodified to the PCI. Software must write the data in longword 0 of the hexword with the following field:

- Byte 0 and 1 contain the encoded message
- Bytes 2 and 3 are message dependent (optional) data field.

A read of the same address range will result in an Interrupt Acknowledge cycle on the PCI and return the vector data provided by the PCI-EISA bridge to the CPU.

Hardware Specific and Miscellaneous Register Space

This register space is located in the range: 87 4000 0000 to 87 FFFF FFFF.

Table 3-7 Hardware Specific Register Address Map

CPU Address <39:28>	**Selected Region**	CPU Address <27:6>	CPU Address <5:0>
1000 0111 0100	CIA control, diagnostic, error registers	LW address	000000
1000 0111 0101	CIA Memory Control registers.	LW address	000000
1000 0111 0110	CIA: PCI Address Translation (S/G, Windows, etc)	LW address	000000
1000 0111 0111	Reserved		
1000 0111 1xxx	Flash RAM, GRU asic CSRs	Byte address	000000

The address space here is a hardware-specific variant of sparse space encoding. For the CSRs, CPU address bits <27:6> specify a longword address where CPU address <5:0> must be zero. All the CIA registers are accessed with a LW granularity. For more specific details on the CIA CSRs please refer to the CSR chapter.

For the Flash ROM, CPU address <30:6> defines a byte address; please refer to the CSR chapter. The fetched byte is always returned in the first byte lane (bits <7:0>).

A number of CSRs in the GRU ASIC (for example, the main interrupt registers) are accessed in this region.

PCI to Physical Memory Addressing

Incoming PCI addresses (32-bit or 64-bit) have to be mapped to the CPU cached memory space (8 GB). The AlphaStation 600 system provides four programmable address windows that control access of PCI peripherals to system memory[1]. The mapping from the PCI address to the physical address can be *direct mapped* (physical mapping with an address offset) or Scatter/Gather mapped (virtual mapping). These four address windows are referred to as the PCI target Windows.

Each window has three registers associated with it. These are:

- **Window Base** (W_BASE) register
- **Window Mask** (W_MASK) register
- **Translated Base** (T_BASE) register

In addition, there is an extra register associated with Window 3 only. This is the **Window DAC** register and is used for PCI 64-bit addressing (that is, the Dual Address Cycle mode).

[1] DOS compatibility is included later in this chapter.

The Window Mask register provides a mask corresponding to bits <31:20> of an incoming PCI address. The size of each window can be programmed to be from 1 MB to 4 GB in powers of two, by masking bits of the incoming PCI address using the Window Mask register as shown in Table 3-8 (note that the Mask field pattern was chosen to speed-up timing critical hardware logic).

Table 3-8 PCI Target Window MASK Register

PCI_MASK <31:20>	Size of Window	Value of n
0000 0000 0000	1 Megabyte	20
0000 0000 0001	2 Megabyte	21
0000 0000 0011	4 Megabyte	22
0000 0000 0111	8 Megabyte	23
0000 0000 1111	16 Megabyte	24
0000 0001 1111	32 Megabyte	25
0000 0011 1111	64 Megabyte	26
0000 0111 1111	128 Megabyte	27
0000 1111 1111	256 Megabyte	28
0001 1111 1111	512 Megabyte	29
0011 1111 1111	1 Gigabyte	30
0111 1111 1111	2 Gigabyte	31
1111 1111 1111	4 Gigabyte	32
other	*Unpredictable*	---
Only the incoming PCI address bits <31:n> are compared with <31:n> of the Window Base register as shown in figure 3-14. If n=32 no comparison is performed. Windows are not allowed to overlap.		

Based on the value of the Window Mask register, the unmasked bits of the incoming PCI address are compared with the corresponding bits of each window's Window Base register. If one of the Window Base registers and the incoming PCI address match, then the PCI address has hit the PCI target window; otherwise it has missed the window. A window enable bit WENB, is provided in each window's Window Base register to allow windows to be independently enabled (WENB = 1) or disabled (WENB = 0).

If a hit occurs in any of the four windows that are enabled, then the CIA will respond to the PCI cycle by asserting the DEVSEL signal. The PCI target windows must be programmed so that their address ranges do not overlap (otherwise the hardware gets confused and results are undefined).

The Window base address must be on a naturally aligned address boundary depending on the size of the window[1]. This restriction is not particularly onerous, since the address space of any PCI device can be located anywhere in the PCI's 4 GB memory space. This scheme is also compatible with the PCI specification:

- A PCI device specifies the amount of memory space it requires via the Base registers in its configuration space. The Base Address registers are implemented such that, the address space consumed by the device is a power of two in size, and is naturally aligned on the size of the space consumed.

A PCI device need not use all the address range it consumes (that is, the size of the PCI address window defined by the Base Address); nor need it respond to unused portions of the address space. The one exception to this is a PCI-bridge which requires two additional registers (the Base and Limit address registers). These registers accurately specify the address space which the bridge device will respond to[2] and are programmed by the POST code. The CIA, as a PCI host-bridge device, does not have BASE and LIMIT regis-

[1] for example, a 4 MB window cannot start at address 1 MB; it must start at addresses 4 MB, 8 MB, 12 MB, etc.

[2] a bridge responds to all addresses in the range: Base <= address < Limit.

ters[1], but does respond to all the addresses defined by the Window Base register (that is, all addresses within a window.)

Figure 3-13 PCI DMA Addressing Example

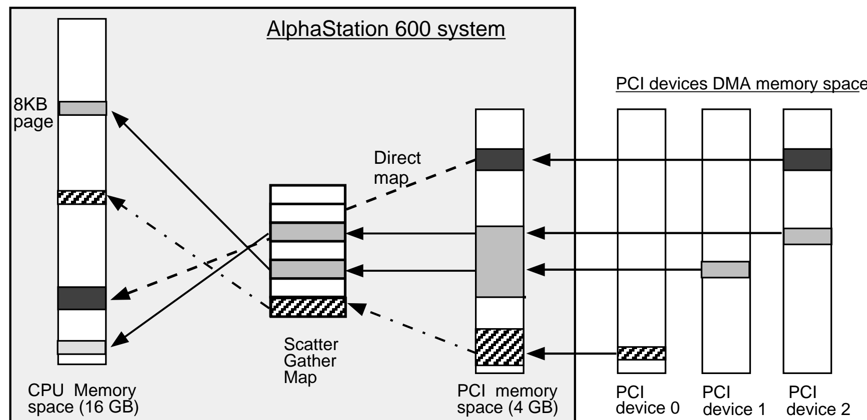


Figure 3-13 shows how the DMA address ranges of a number of PCI devices are accepted by the AlphaStation 600 system PCI-window ranges. Note that PCI devices are allowed to have multiple DMA address ranges (for example, device 2). The example also shows that the AlphaStation 600 system window can be larger than the corresponding devices DMA address range (see device 0). Device 1 and device 2 have address ranges which are accepted by one AlphaStation 600 system window. Each window determines whether direct- or scatter/gather-mapping is used to access physical memory (see S/G-bit later)

Figure 3-14 PCI Target Window Compare

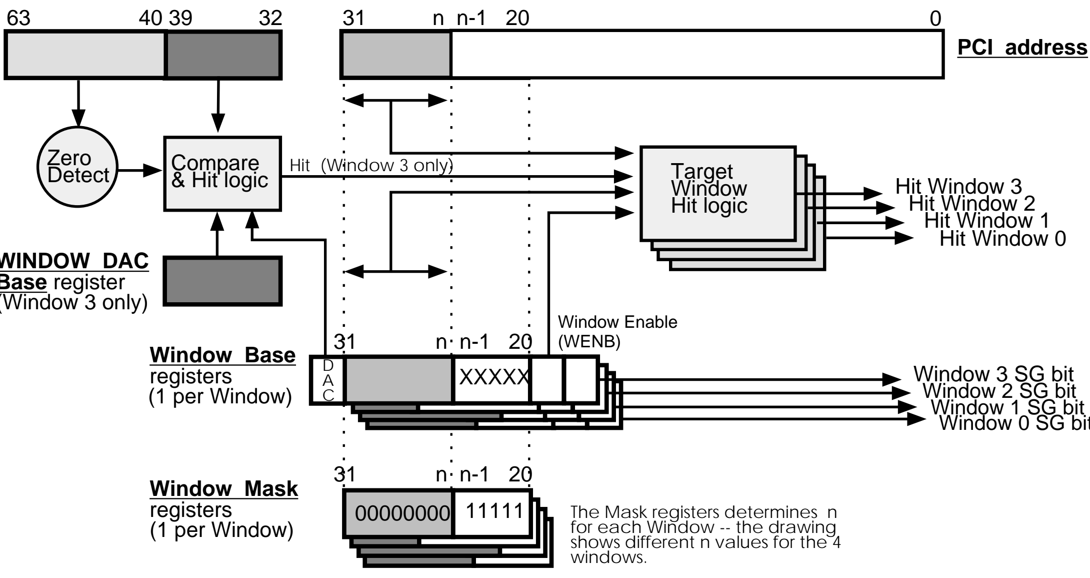


Figure 3-14 depicts the PCI window logic. The comparison logic associated with PCI address bits <63:32> is only used for the DAC[2] mode; and only if enabled by a bit in the Window Base register for Window 3. This logic is only applicable to Window 3; the remaining Windows only recognize 32-bit PCI addresses (that is, SAC[3] cycles). For a hit to occur in a DAC address, address bits <63:40> must be zero; bits<39:32> must match the Window DAC Base register; and the low-order address bits <31:20> must also hit. This

1 Host-bridges, since they are under system control, are free to violate the rules!

2 Dual-Address cycle -- only issued if <63:32> are non-zero for a 64-bit PCI address.

3 Single Address cycle -- all 32-bit addresses. A PCI device must use SAC if <63:32> = 0 of a 64-bit address.

scheme allows a naturally aligned, 1 MB-4 GB PCI window to be placed anywhere in the first 1TB of a 64-bit PCI address.

When an address match occurs with a PCI Target Window, the CIA ASIC translates the 32-bit PCI address to a memory address <33:0>. The translated address is generated in one of two ways as determined by the SG (Scatter/Gather) bit of the Window's PCI BASE register.

Direct-mapped Addressing

If the SG bit is cleared, the DMA address is direct mapped, and the translated address is generated by concatenating bits from the matching window's Translated Base register (T_BASE) with bits from the incoming PCI address. The bits involved in the concatenation are defined by the Window Mask register as shown in Table 3-9 . Note that the unused bits of the Translated Base register as indicated in Table 3-9 must be cleared (that is, the hardware performs an AND-OR for the concatenation). Since memory is located in the lower 8 GB of the CPU address space, the AlphaStation 600 chip-set ensures (implicitly) that address <39:33> is always zero.

Note that since the Translated Base is simply concatenated to the PCI address, then the direct mapping is to a naturally-aligned memory region. For example, a 4 MB direct-mapped window will map to any 4 MB region in main memory which falls on a 4 MB boundary (for instance, it is not possible to map a 4 MB region to the main memory region 1 MB-5 MB).

Table 3-9 Direct-mapped PCI Target Address Translation

WINDOW_MASK <31:20>	Size of Window	Translated Address<32:2>
0000 0000 0000	1 MB	32 20 19 2
0000 0000 0001	2 MB	32 21 20 2
0000 0000 0011	4 MB	32 22 21 2
0000 0000 0111	8 MB	32 23 22 2
0000 0000 1111	16 MB	32 24 23 2
0000 0001 1111	32 MB	32 25 24 2
0000 0011 1111	64 MB	32 26 25 2
0000 0111 1111	128 MB	32 27 26 2
0000 1111 1111	256 MB	32 28 27 2
0001 1111 1111	512 MB	32 29 28 2
0011 1111 1111	1 GB	32 29 28 2
0111 1111 1111	2 GB	32 30 29 2
1111 1111 1111	4 GB	32 31 2
		T_BASE PCI address
Note: unused bits of the Translation Base register must be zero for correct operation.		

Figure 3-15 Direct-mapped Translation

Mask <31:20>	n
0000 0000 0000	20
0000 0000 0001	21
0000 0000 0011	22
0000 0000 0111	23
0000 0000 1111	24
0000 0001 1111	25
:	:
0111 1111 1111	31
1111 1111 1111	32

Scatter/Gather Addressing

If the SG bit of the PCI Base register is set, then the translated address is generated by a table lookup. This table is referred to as a Scatter/Gather Map. Figure 3-18 shows the scatter/gather addressing scheme -- full details of this scheme are provided later in this section; but for now a quick description is provided: The incoming PCI address is compared to the PCI Window addresses for a hit. The Translated Base register, associated with the PCI-window which hit, is used to specify the starting address of the Scatter/Gather map table in memory. Bits of the incoming PCI address are used as an offset from this starting address, to access the scatter/gather PTE. This PTE in conjunction with the remaining, least-significant PCI address bits, forms the required memory address.

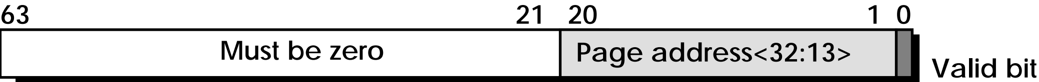
Each Scatter Gather (SG) map entry maps an 8 KB page of PCI address space into an 8 KB page of the processor's address space. This offers a number of advantages to software:

- Performance: ISA devices map to the lower 16 MB of memory. NT currently copies data from here to user space. The Scatter/Gather map avoids this copy.
- User IO buffers cannot be counted on to be physically contiguous nor contained within a page. Without scatter/gather, the software needs to manage the "scattered" nature of the user buffer by copies.

In the PC world, the term scatter/gather is not an address translation scheme but instead is used to signify a DMA transfer list. An element in this transfer list contains the DMA address and the number of data items to transfer. The DMA device fetches each item of the list until the list is empty. Many of the PCI devices (for example, EISA bridge) support this form of scatter/gather.

Each SG entry (PTE) is a quadword and has a valid bit in bit position 0. Address bit 13 is at bit position 1 of the map entry. Since the AlphaStation 600 chip set only implements valid memory addresses up to 8 GB, then bits <63:21> of the SG map entry must be programmed to 0. Bits <20:1> of the SG map entry are used to generate the physical page address. This is appended to the bits <12:5> of the incoming PCI address to generate the memory address.

Figure 3-16 Scatter/Gather PTE Format



The size of the Scatter/Gather Map table is determined by the size of the PCI Target Window as defined by the Window Mask register as shown in Table 3-10. The number of entries is the Window size divided by the page size (8 KB). The size of the table is simply the number of entries multiplied by 8B.

The Scatter/Gather map table address is obtained from the Translated Base register and the PCI address as shown in Table 3-10.

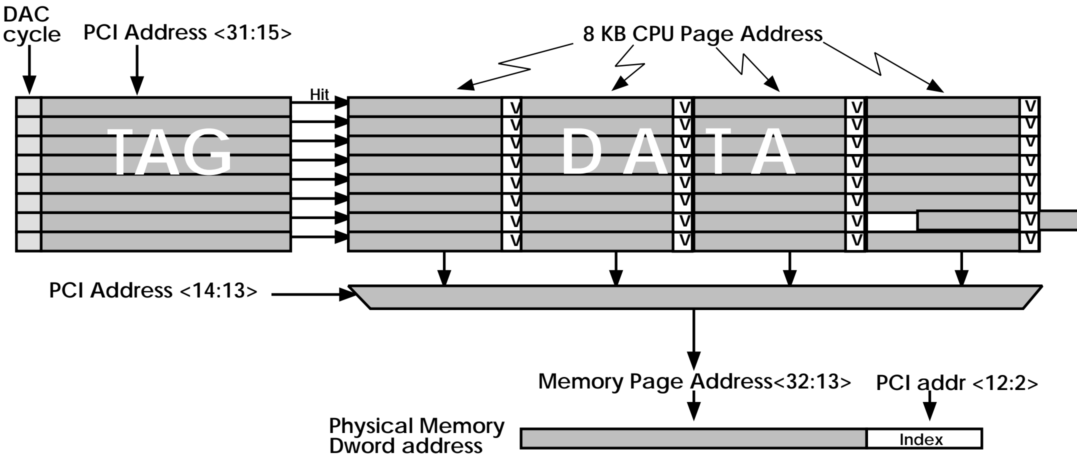
Table 3-10 Scatter/Gather Mapped PCI Target Address Translation.

WINDOW_MASK <31:20>	Size of Window	SG Map Table size	Scatter Gather Map Address<33:3>
0000 0000 0000	1 MB	1 KB	32 10 19 13
0000 0000 0001	2 MB	2 KB	32 11 20 13
0000 0000 0011	4 MB	4 KB	32 12 21 13
0000 0000 0111	8 MB	8 KB	32 13 22 13
0000 0000 1111	16 MB	16 KB	32 14 23 13
0000 0001 1111	32 MB	32 KB	32 15 24 13
0000 0011 1111	64 MB	64 KB	32 16 25 13
0000 0111 1111	128 MB	128 KB	32 17 26 13
0000 1111 1111	256 MB	256 KB	32 18 27 13
0001 1111 1111	512 MB	512 KB	32 19 28 13
0011 1111 1111	1 GB	1 MB	32 20 29 13
0111 1111 1111	2 GB	2 MB	32 21 30 13
1111 1111 1111	4 GB	4 MB	32 22 31 13
			T_BASE PCI address
Note: unused bits of the Translated Base register must be zero for correct operation.			

Scatter/Gather TLB

An eight-entry Translation-lookaside Buffer (TLB) is provided in the CIA for Scatter/Gather map entries. The TLB is a fully associative cache and holds the eight most recent Scatter/Gather map look-ups. Four of these entries can be "locked" preventing their displacement by the hardware TLB-miss handler. Each of the eight TLB entries holds a PCI address for the tag, and four consecutive 8 KB CPU page addresses as the TLB data (see Figure 3-17).

Figure 3-17 Scatter/Gather Associative TLB



Each time an incoming PCI address hits in a PCI Target Window which has scatter/gather enabled, bits <31:15> of the PCI address are compared with the 32KB PCI page address in the TLB tag. If a match is found, the required CPU page address is one of the four items provided by the data of the matching TLB entry. PCI address <14:13> selects the correct 8 KB CPU page from the four fetched.

With a TLB hit, the Scatter/Gather map table look-up in memory is avoided, resulting in enhanced performance. If no match is found in the TLB, the Scatter/Gather map lookup is performed and four PTE entries are fetched and written over an existing entry in the TLB. The TLB entry to be replaced is determined by a round robin algorithm on the "unlocked" entries. Coherency of the TLB is maintained by software writes to the SG_TBIA (scatter gather translation buffer invalidate all) CSR.

The TAG portion contains a DAC flag to indicate that the PCI Tag address <31:15> corresponds to a 64-bit DAC address. Only one bit is required instead of the high-order PCI address bits <39:32> since only one window is assigned to a DAC cycle, and the Window-hit logic has already performed a comparison of the high-order bits against the PCI DAC BASE register.

Figure 3-18 shows the entire translation from PCI address to physical address on a window that implements scatter/gather. Both paths are indicated: the right side shows the path for a TLB hit, while the left side shows the path for a TLB miss. The Scatter/Gather TLB is shown in a slightly simplified, but functionally equivalent form.

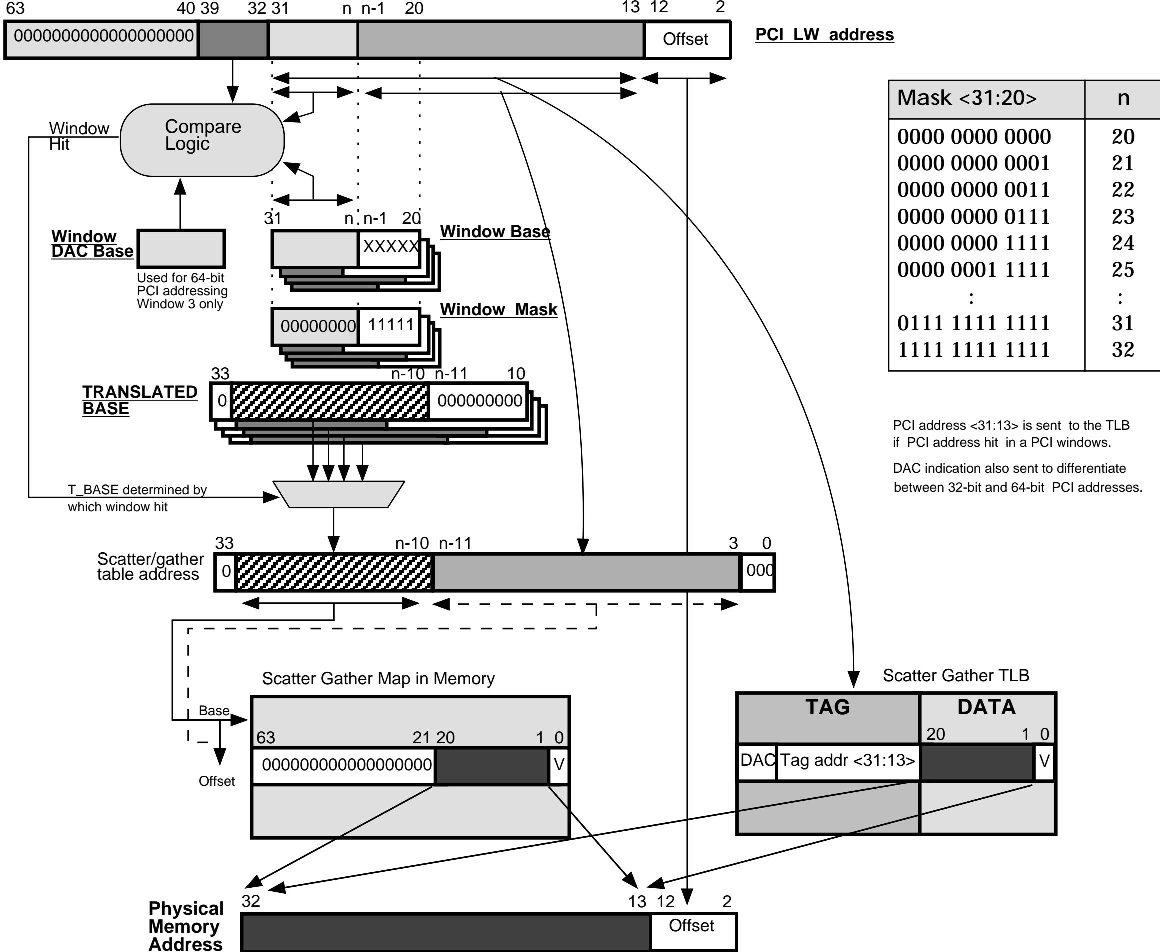
The process for a Scatter/gather TLB hit is as follows:

- The Window compare logic determines if the PCI address has hit in one of the four windows, and the PCI BASE<SG> bit determines if the scatter/gather path should be taken. If Window 3 has DAC mode enabled, and the PCI cycle is a DAC cycle, then a further comparison is made between the high-order PCI bits and the PCI DAC BASE register.
- PCI address <31:13> is sent to the TLB associative Tag together with the DAC-HIT indication. If the address and DAC bits match in the TLB then the corresponding CPU 8 KB page-address is read out of the TLB. If this entry is valid then a TLB hit has occurred and this page-address is concatenated with PCI address <12:2> to form the physical memory address. If the data entry is invalid, or if the TAG compare failed, then a TLB miss occurs (see Chapter 8, Control and Status Registers).

The process for a scatter/gather TLB miss is as follows:

- The relevant bits of the PCI address (as determined by the Window Mask register) are concatenated with the relevant TRANSLATED BASE register bits to form the address used to access the Scatter/Gather map entry from a table located in main memory.
- Bits <20:1> of the map entry (that is, the PTE from memory) are used to generate the physical page address, which is appended to the page offset to generate the physical memory address. The TLB is also updated at this point (round-robin algorithm) with the four PTE entries which correspond to the 32 KB PCI page address which first missed the TLB. The Tag portion of the TLB is loaded with this PCI page address and the DAC bit is set if this PCI cycle is a DAC cycle.
- If the requested PTE is marked invalid (bit 0 clear) then a TLB invalid entry exception is taken (see Chapter 9, Hardware Exceptions and Interrupts).

Figure 3-18 Scatter/Gather Map Translation



Mask <31:20>	n
0000 0000 0000	20
0000 0000 0001	21
0000 0000 0011	22
0000 0000 0111	23
0000 0000 1111	24
0000 0001 1111	25
:	:
0111 1111 1111	31
1111 1111 1111	32

PCI Window Suggested Use

Figure 3-19 shows the power-up PCI window assignment (configured by firmware) and Table 3-11 tabulates the details. PCI window 0 was chosen for the 8 MB-16 MB "EISA" region since this window incorporates the MEMCS# logic. PCI window 3 was avoided since this window incorporates the DAC-cycle logic. Of the remaining two windows, PCI window 1 was chosen arbitrarily for the 1 GB direct-mapped region, and PCI window 2 is not assigned.

Figure 3-19 Default PCI Window Allocation

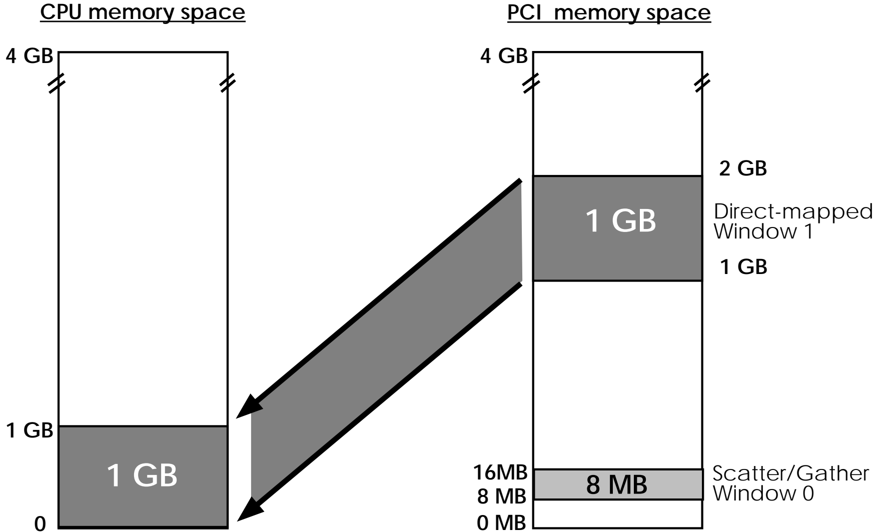


Table 3-11 PCI Window POST Configuration

PCI window	Assignment	Size	Comments
0	Scatter/Gather	8 MB	Not used by firmware. MEMCS disabled
1	Direct mapped	1 GB	Mapped to 0-1 GB of main memory
2	Disabled		
3	Disabled		

PC Compatibility Addressing and Holes

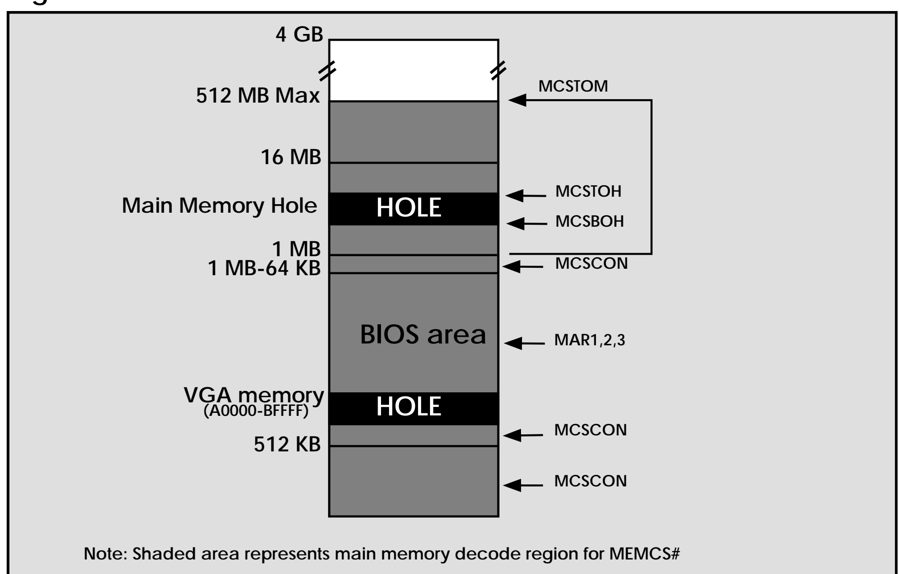
The PC architecture allows certain (E)ISA devices to respond to hardwired memory addresses. An example is a VGA graphics device which has its frame buffer located in memory address region A0000-BFFFF. Such devices, pepper memory space with " holes", which are collectively known as PC compatibility holes.

This is described in more detail in the PCI-EISA bridge chapter. This bridge chip decodes PCI addresses and generates a signal, MEMCS#, which takes into account the various PC compatibility holes.

MEMCS#

The PCEB chip of the PCI-EISA bridge provides address decode logic with considerable attributes and features (for example, read only, write only, VGA frame buffer, memory holes, BIOS shadowing) to help manage the EISA memory map and PC compatibility holes. This is known as *main memory decoding* in the PCEB chip, and results in the generation of the MEMCS# (**MEM**ory **C**hip **S**elect) signal. The CIA uses this signal if enabled via the PCI BASE register for window 0.

Figure 3-20 MEMCS# Decode Area



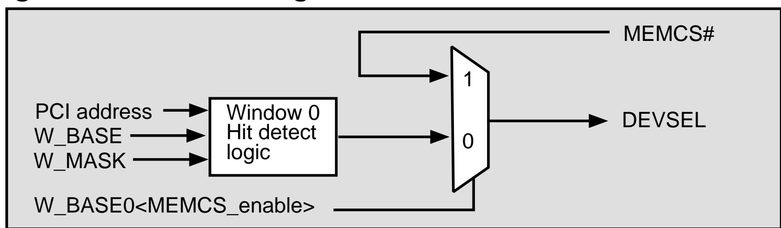
In Figure 3-21 the MEMCS# range is shown shaded lightly; the two main holes are shown shaded darkly. This range is subdivided into numerous portions (for example, BIOS areas) which are individually enabled/disabled using various registers.

- The MCSTOM (top of memory) register. This has a 2 MB granularity and can be programmed to select the regions from 1MB up to 512 MBs.
- The MCSTOH (top of hole) and MCSBOH (bottom of hole) registers define a memory hole region where MEMCS# is not selected. The granularity of the hole is 64 KB.
- The MAR1,2,3 registers. These enable various BIOS regions.
- The MCSCON (control) register. This register enables the MEMCS# decode logic, and selects a number of regions (for example, 0-512 KB).
- The VGA memory hole region never asserts MEMCS#.

For more detail refer to the Intel 82375EB specification.

PCI window 0 in the CIA can be enabled to accept the MEMCS# signal as the PCI memory decode signal. With this path enabled, the PCI window hit logic simply uses the MEMCS# signal (that is, if MEMCS# is asserted then a PCI window 0 hit occurs and the PCI DEVSEL signal is asserted).

Figure 3-21 MEMCS# Logic



Consequently, the PCI BASE address must be large enough to encompass the MEMCS region programmed into the PCI-EISA bridge. The remaining window attributes are still applicable and required:

- The SG bit in the PCI BASE determines if scatter/gather or direct-mapping is applicable.
- The MASK register size information must match the MEMCS# size (in order for the S/G and direct mapping algorithms to correctly use the Translated Base register).
- The MEMCS_Enable bit in the W_BASE0 CSR takes precedence over the PCI window enable bit (that is, W_BASE<W_EN>).

4 Modules

Memory MotherBoard

There are two Memory MotherBoards (MMBs) in the AlphaStation 600 system.

Each MMB supplies 144 bits of data (128 bits + ecc) making a total of 288 bits. Thus, there must be at least four industry-standard 36 bit SIMMs on each MMB, making the total minimum memory requirement eight SIMMs. Each MMB supports a maximum of sixteen SIMMs.

The AlphaStation 600 system supports 1M x 36, 2M x 36, 4M x 36, 8M x 36, 16M x 36, and 32M x 36 SIMMs. However the first implementation of the MMB does not support the 16M x 36, and 32M x 36 SIMMs.

The maximum memory is calculated as 32 x (8M x 36) = 1 GByte.

Figure 4-1 MMB Layout

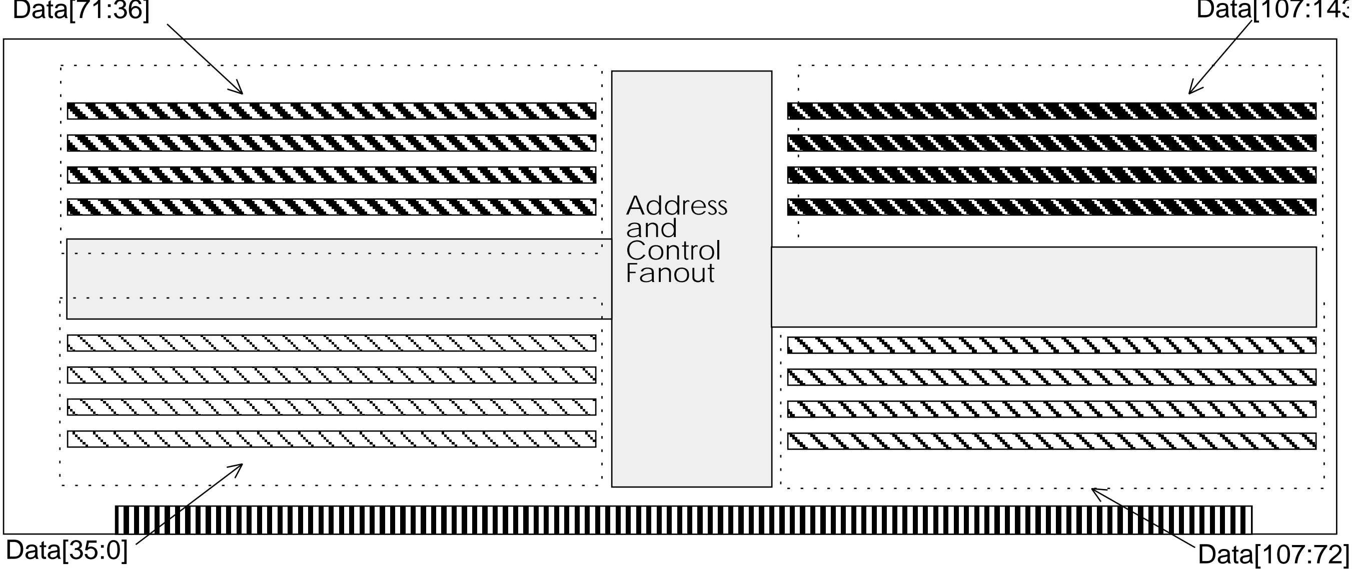


Figure 4-1 shows the layout of the MMB.

The address and control signals are routed to the center of the module and data to each end. Each MMB receives two copies of the ADDRESS, CAS, RAS and WE and one copy of the SET_SELECTs. The ADDRESS, CAS and WE are fanned out so that each SIMM gets its own copy. RAS is gated with SET SELECT and fanned out four times such that a SET SELECT drives a set of four SIMMs. Each SIMM receives two copies of RAS (each gated by a unique SET SELECT), one to drive side 0 and the other to drive side1.

The sets are organized from the bottom of each quadrant towards the top of the module (see Figure 4-2 SIMM Population Order). Single sided SIMMs contain one set and double sided SIMMs have two sets, Starting with SET 0, 1 at the bottom and SET 6, 7 at the top.

Figure 4-2 SIMM Population Order

Both MMBs must be populated with the same type of SIMMs in each set

First set of four SIMMs (SET 0, 1)

Second Set of four SIMMs

(SET 2, 3)

Third Set of four SIMMs

(SET 4, 5)

Fourth Set of four SIMMs

(SET 6, 7)

Presence Detect Bits

Each MMB provides presence detect bits to the GRU.

Two of the Presence Detect (PD) bits of each SIMM 0 (bits[4:3]) from each set are wired to a multiplexor. At power up, these bits are muxed out serially into the GRU. The mux selects are generated by the GRU and the output data is sampled at the relevant time.

Table 4-1 shows the various SIMM speeds. The size of the DRAMs on each SIMM and whether there are one or two sets per SIMM is determined by firmware during initialization.

Table 4-1 SIMM Speed

PD[4]	PD[3]	DRAM Speed
0	0	60/100 ns
0	1	80 ns
1	0	70 ns
1	1	60 ns

Cache SIMM

There are three Cache SIMMs in each AlphaStation 600 system. Each SIMM contains SRAMs to store 48 data bits and eight tag bits. This provides 144 data bits (= 16 Bytes + 4 Bytes of ECC) and 24 tag bits total. The data bits were assigned to the Cache SIMMs on an individual bit basis so there is not necessarily a correlation between bit number and a particular SRAM. The AlphaStation 600 system uses less than 24 tag bits so some bits are not connected.

The initial version of cache SIMM configurations using different SRAM speeds and sizes are:

- 4 MByte SIMM - 13 (128K x 8) SRAMs (7 on side 0, 6 on side 1)
- 2 MByte SIMM - 7 (128K x 8) SRAMs (7 on side 0)

The larger variant WIRE-ORs two SRAMs to each data bit. The smaller version does not. When suitable SRAMs become available, 8 MByte and 16 MByte versions will be offered.

The EV5 supplies the INDEX_H which is fanned out as the SRAM address. The data SRAMs use INDEX_H[21:4] (mapped to INDEX[17:0] at the connector) and the tag SRAMs use INDEX_H[21:6] (tag address does not change within a cache block).

The 4 MByte SIMM uses INDEX_H[22] to select between the WIRE-ORed SRAMs (a true copy is fanned out to one side output enable and an inverse copy fanned out to the other output enable). It also uses INDEX_H[22] as an extra tag address bit.

Each variant of Cache SIMM is defined by a coded placement of zero ohm resistors to form the PD_CACHE[4:0] field. At power up, these bits from Cache SIMM 0 are loaded into the GRU so they can be used by firmware during initialization (see CACHE_CNFG register - address 87.8000.0200). Firmware will assume all Cache SIMMs are the same variant as Cache SIMM 0. The encodings are shown in Table 4-2 and Table 4-3.

Table 4-2 Cache Speed Encodings

PD_CACHE[1:0]	Cache RAM Speed
00	8 ns
01	10 ns
10	12 ns
11	15 ns

Table 4-3 Cache Size Encodings

PD_CACHE[4:2]	Cache RAM Size
000	No Cache Present
001	reserved
010	2 Mbyte Total
011	4 Mbyte Total
100 - 111	reserved

I/O Subsystem Module

The I/O Subsystem module is a PCI option card intended for the AlphaStation 600 system and Sable families, but it could be considered for any PCI-based system. It allows for significant system expansion by providing four I/O ports in one PCI slot. The module features two high performance fast/wide SCSI controllers and an Ethernet controller which can be connected to twisted-pair, thickwire (AUI), or thinwire network.

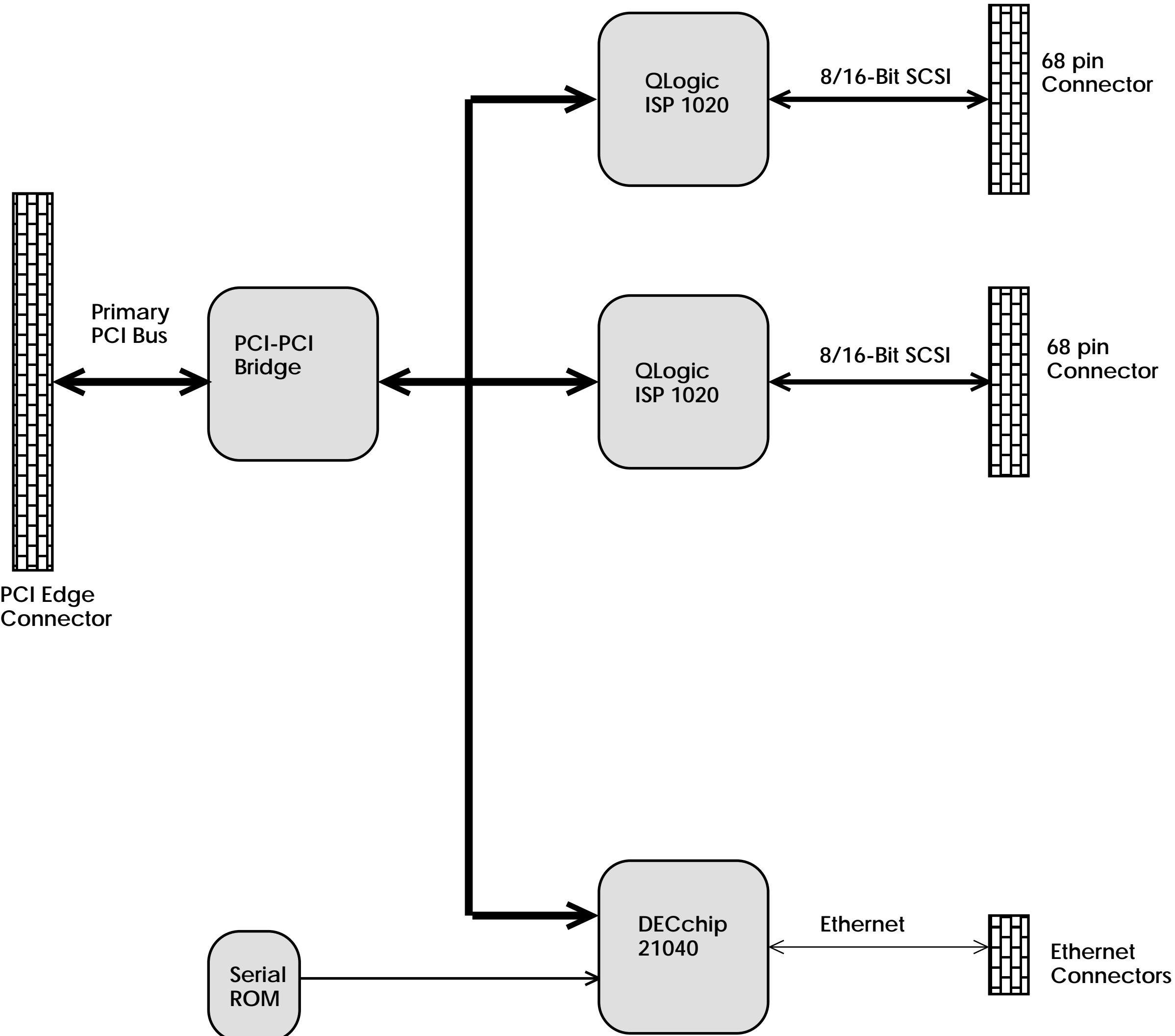
The block diagram is shown in Figure 4-3.

Major Components

The major components of the module are :

- PCI - PCI Bridge (DECchip 21050)
- PCI - Ethernet Controller (DECchip 21040)
- PCI - SCSI Controller (QLogic ISP1020) (two)
- DEC Standard 134-0, Rev-B, Version 2.0.0

Figure 4-3 I/O Subsystem Module Block Diagram



5 Power Up and Reset

Introduction

This chapter describes the power up signal sequences and the halt and reset sequences.

Power Up

The power supply outputs two signals, PRESENT_L and DC_OK_L. These four signals produce two signals SYS_DC_OK and SYS_DC_OK_L. If a power supply asserts PRESENT_L then it must assert DC_OK_L before SYS_DC_OK and SYS_DC_OK_L will be asserted.

The DECchip 21164-AA takes SYS_DC_OK and the GRU and CS on the cache RAMs take SYS_DC_OK_L. This protects the cache RAMs during power up. While SYS_DC_OK is deasserted, the DECchip 21164-AA tristates every output and bidirectional and weakly pulls them to ground.

Halt and Reset

The OCP outputs two signals, OCP_RESET_L and OCP_HALT_L., from the control panel. OCP_RESET_L goes to the GRU. OCP_HALT_L is inverted and then goes to the DECchip 21164-AA as MCH_HLT_IRQ_H on the DECchip 21164-AA.

SYS_DC_OK_L and OCP_RESET_L are synchronized in the GRU through a two-stage synchronizer. The output of the synchronizers are ORed with the input to provide an asynchronously asserting, synchronously deasserting master reset signal. SYS_DC_OK_L asserts the master reset when it is deasserted. OCP_RESET_L asserts the master reset when it is asserted.

The master reset initializes the reset counter in the GRU and asserts SYS_RST_L. The reset counter counts 256 cycles. While counting, SYS_RST_L remains asserted. While SYS_RST_L is asserted, the GRU internal reset, GRU_RESET_L, is also asserted. The GRU outputs CLK_DIV<3:0> onto IRQ<3:0> three cycles after SYS_RST_L is asserted and for three cycles after it is deasserted. SYS_RST_L is input into the DECchip 21164-AA, DSW, SROM counters, and CIA.

The DECchip 21164-AA takes SYS_RST_L into the input SYS_RESET_L. While in reset, the DECchip 21164-AA reads sysclock configuration parameters from the interrupt pins, IRQ_H<3:0>. It also reads delay configuration parameters for SYS_CLK_OUT2_x from the SYS_MCH_CHK_IRQ_H, PWR_FAIL_IRQ_H, and MCH_HLT_IRQ_H pins. This clock is not used by this system, so no special attention is given to the value of the signals during reset. For the values of the sysclock configuration and internal state after reset, see the DECchip 21164-AA Functional Specification.

The four DSW's take SYS_RST_L into the asynchronous clear pin of an FD2. This provides an asynchronously asserting, synchronously deasserting reset signal, RESET_L. While RESET_L is asserted, the internal state of the DSW is reset. The SROM counter is cleared when reset is asserted.

The CIA takes SYS_RST_L and synchronizes it with a two-stage synchronizer. The output of the synchronizers are ORed with the input to provide an asynchronously asserting, synchronously deasserting reset signal, RESET_L. While RESET_L is asserted, the internal state is reset to its default state. The CIA asserts RST_L, the PCI reset signal, until software sets the PCI enable in the CIA_CNTL register.

The following is a table of the CIA output and bi-directional signals and their state during reset:

Signal	Reset State		Signal	Reset State
ADDR_CMD_PAR	z		PAR	z
ADDR39	z		PAR64	z
ADDR[34:4]	z		REQ64_L	0
CMD[3:0]	z		ACK64_L	z
ADDR_BUS_REQ	z		FRAME_L	z
CACK	z		IRDY_L	z
DACK	z		DEVSEL_L	z
FILL	z		TRDY_L	z
FILL_ID	z		STOP_L	z
FILL_ERROR	z		PERR_L	z
IDLE_BC	z		SERR_L	z
TAG_DIRTY	z		REQ_L	1
TAG_CTL_PAR	z		RST_L	0
MEM_ADDR[12:0]	x		CMC[8:0]	0
SET_SEL[15:0]	0		IOC[6:0]	52
RAS[3:0]	0		IOD[63:0]	z
CAS[3:0]	0		IOD_E[7:0]	z
MEM_WE_L[1:0]	3		INT	0
MEM_EN	1		ERROR	0
AD[63:0]	z		TEST_OR_SCAN_OUT	0
CBE_L[7:0]	z			

RST_L is distributed to the ESC, EISA System Component, the PCEB, PCI-EISA Bridge, and to the PCI slots through an FCT805 buffer.

The ESC takes RST_L and resets its internal state and asserts RSTDRV, which is the ISA hardware reset signal. The PCEB takes RST_L and resets its internal state.

RSTDRV resets the TOY, 8242, and 87312.

6

AlphaStation 600 Physical Partitioning

Introduction

This chapter describes the physical partitioning of the AlphaStation 600 system. which is based on the EV5 implementation of the Alpha architecture. Refer to Figure 6-1 for the AlphaStation 600 system block diagram..

The widths of the AlphaStation 600 system busses shown in the block diagram are shown in Table 6-1.

Table 6-1 AlphaStation 600 System Busses

Bus Name	Description	Data Width	ECC Width	Cycle Time
CPU_DAT	CPU/Bcache Bus	128 bits	16 bits	25/30 ns
MEM_DAT	Memory Data Bus	256 bits	32 bits	
IOD	DSW/CIA/GRU I/O Bus	64 bits	8 bits	30 ns
AD	PCI Address/Data Bus	64/32 bits	2/1 (Parity)	30 ns
SD	EISA Data Bus	32 bits	None	120 ns
LA	EISA Latched Address	32 bits	None	120 ns
SA	EISA Slave Address	20 bits	None	120 ns
XD	X-Bus Data	8 bits	None	120 ns

Hardware Jumpers

There are various jumpers and DIP switches on the Systemboard which are mainly for Prototype debugging.

Fan Fail Detect Jumpers

The Fan Fail Detect circuit is design to operate in both the AlphaStation 600 "Tower" and the "Wide Tower". Wide Tower systems have two fans, and Tower systems have one. Both systems have a power supply fan as well. The Fan Fail Detect circuit senses the power used by each fan. If the fan stops or is disconnected the circuit triggers a failure.

Fans 1 and 2 are plugged into headers, J24 or J26. The system can work if FAN1 is plugged into either J24 or J26, and FAN2 (Wide Tower only) plugged into the other header.

The fan cable connectors are polarized and have wires in only 2 of 3 sockets. Therefore, if properly assembled, FAN1 connects to pins 1 and 2 of J24 and FAN2 connects to pins 2 and 3 of J26. The black wire is always on pin 2. Pin 1 of J24 and J26 and all jumpers is closest to the top of the module as oriented in Figure 6-3.

The logic on the system board detects a fan failure and initiates a power system shut-down. If the fan failure occurs on power up, the power-on light will be on for a half second.

The power light being on for a half second could indicate that the power supply detected a short. In the case of a fan failure the Fan Fail Detect LED will be on for a half second.

The "Fan Fail Detect" logic for FAN1, which is in both the Tower and Wide Tower enclosures, is enabled by installing a jumper between pins 1 and 2 of W11. (Manufacturing can disable this circuit by connecting a jumper from pin 2 to pin 3.)

The Fan Fail Detect logic for FAN2 should be disabled in Tower systems and enabled in Wide Tower systems. The FAN2 detect logic is disabled by installing a 2-pin jumper on W9 (connecting a jumper between pin 2 and pin 3) . It will be enabled if no jumper is used.

Manufacturing will leave the W11 jumper in the enable position and the W9 jumper in the disable position (in the Wide Tower the W9 jumper must be enabled).

Flash ROM Write Jumper

The Flash ROM Write Jumper, W13, enables or disables writes to all four Flash ROMs. Writes are enabled when a 2-pin jumper is installed between pins 2 and 3 and disabled when installed between pins 1 and 2. Manufacturing will normally leave the jumper in the disable position.

Alternate Console Jumper

The Alternate Console Jumper, W17, is used to report boot information through the serial port instead of through the graphics device. The alternate console is enabled by connecting a jumper from pins 1 and 2 of W17 and disabled by connecting a jumper from pins 2 and 3. Manufacturing will normally leave the jumper in the disable position.

Secure Console Jumper

When enabled , the Secure Console jumper, W14, disables all privileged console commands . This security feature is enabled by connecting a jumper between pins 1 and 2. When enabled, BOOT, START, CONTINUE and LOGIN are the only commands allowed. All console commands are enabled when W14's jumper is connected between pins 2 and 3 (Secure Console disabled). Manufacturing will normally leave the jumper in the disable position.

SROM Code Select Jumper

Jumpers W1-W8 are used to select boot or test code stored in the SROM. Only one jumper can be installed at a time. The contents are outlined below :

- W1 - normal power up flow. SROM will default to floppy boot if Flash ROM is not loaded.
- W2 - mini-console with initialized system interface
- W3 - floppy boot
- W4 - memory test
- W5 - normal power up flow with only set 1 of SCache enabled
- W6 - normal power up flow with only set 2 of SCache enabled
- w7 - normal power up flow with only set 3 of SCache enabled
- W8 - non-initialized mini-console with un-initialized system interface (requires user to type uppercase U before any output is seen)

For normal system operation, W1 is selected. Manufacturing will normally leave the jumper in the W1 position.

EV5 clock multiple DIP switch

This DIP switch will only be provided on the first few debug systemboards. This is used during reset time to select the EV5 system clock multiple.

Physical Organization

The base model of the AlphaStation 600 system is physically organized into six modules, of three different types. There is one system board, into which the other five modules fit. There are two MMB modules, which hold the memory SIMMs (not included in this count), and there are three cache SIMM modules. A real AlphaStation 600 system, however, is likely to include other modules, including memory SIMMs, PCI options for graphics and disk/ethernet access, EISA options for disks, audio, etc. Figure 6-2 shows the AlphaStation 600 system board layout, and the organization of logical functions on the system board. The following list provides a description of the functional areas shown in Figure 6-3.

System Board Functional Areas

1 An EV5 Alpha CPU, running at 266 MHz for prototype machines, and 300+ MHz for later implementations

2 SROM interface to allow serial loading of the EV5 from onboard EPROM, and serial communication via an SROM/RS-232 adapter card plugged into a special 10-pin connector.

3 A configurable third-level BCache, organized into three Cache SIMMs plugged into the system board. Cache SIMMs can be single or double sided, allowing 2 MB or 4 MB of cache, respectively. Next generation SRAMS will allow 8 MB and 16 MB cache sizes.

4 A data switch, composed of four custom ASIC chips, called DSW in a sliced configuration. The data switch allows data from the I/O and memory systems to get to and from the CPU. Total width of the data paths through the data is as follows: Memory - 256 bits, CPU/cache - 128 bits, I/O - 64 bits. All paths carry error correction code bits (ECC)

5 A control chip (CIA) which controls the PCI system and acts as a data path from PCI to the data switch, controls main memory and Bcache operations, and controls the system support chip, GRU.

6 A system support chip (GRU) which provides access to the four Flash ROMs used to store system firmware, provides access to the presence detect bits which are used to size memory and cache, and gathers system interrupts and sends them to the CPU.

7 A set of two memory motherboards (MMBs) which plug into the System Board. Both MMBs must be present for main memory to work; together they provide up to 1.0 GB of memory with current technology (4Mx4) DRAMS. The system board is designed to accommodate later versions of the MMB which could support up to 4 GB of main memory.

8 A PCI I/O system, consisting of three 64 bit PCI slots, and two 32 bit PCI slots. One of the two 32 bit slots is a "shared" slot with EISA, meaning that the slot can be occupied by a PCI option, or a neighboring EISA option, but not both simultaneously.

9 An EISA I/O system, which is connected off the PCI I/O system via the Intel 82374/82375 (Mercury) chip set. This chip set provides a bridge between the PCI and EISA busses, and ISA support functions for ISA devices such as TOY clock, NVRAM, and other ISA I/O devices.

10 An ISA / X-bus I/O system, which includes two serial ports, one parallel port, a keyboard/mouse port, floppy drive port, and Operator Control Panel (OCP) port.

11 Miscellaneous support functions, implemented directly on the System Board. These include generation of DC_OK for the system from signals supplied by one or two power supplies, fan fail detect logic, clock distribution.

Figure 6-1 AlphaStation 600 System Block Diagram

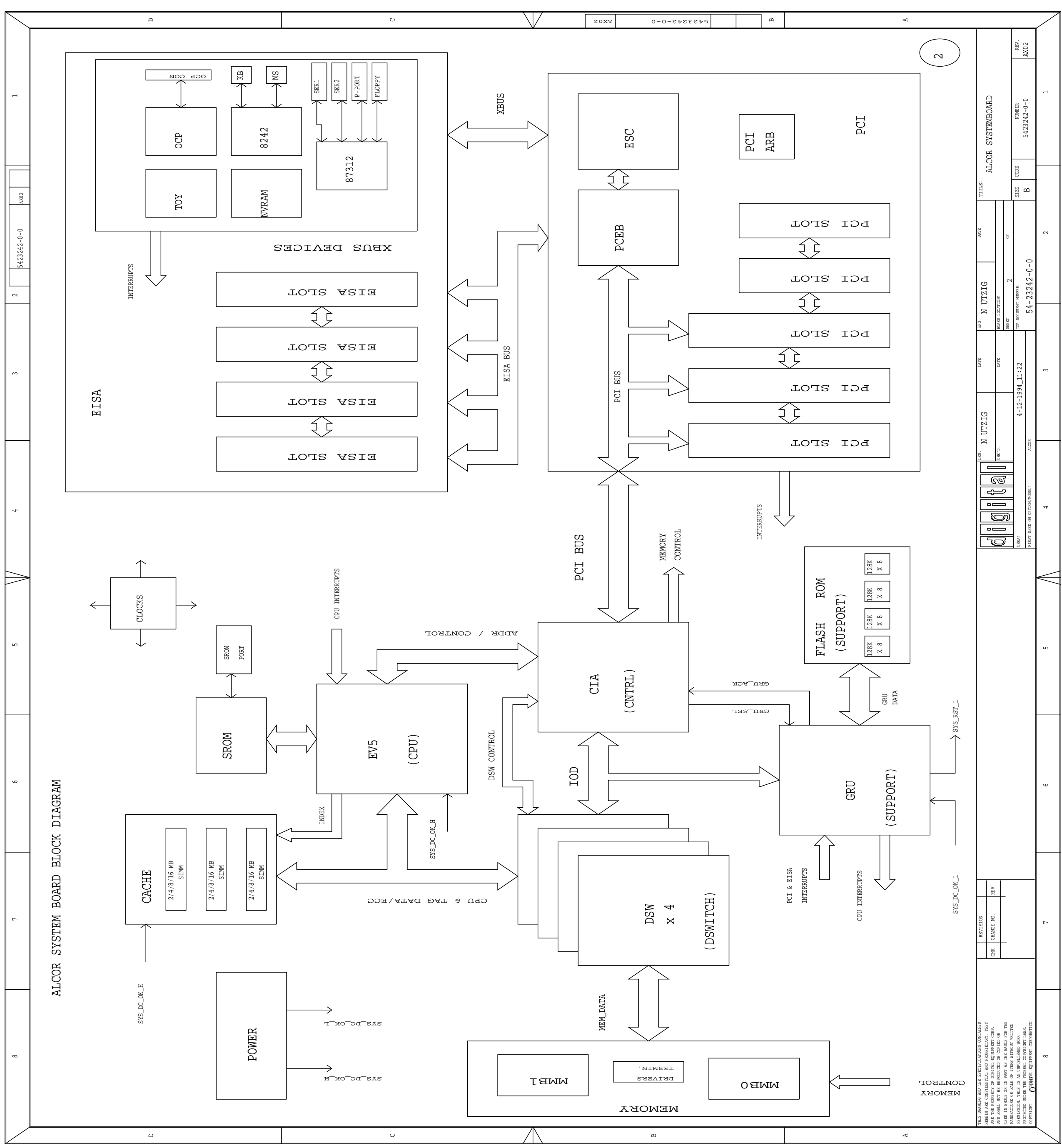


Figure 6-2 AlphaStation 600 System Board Layout

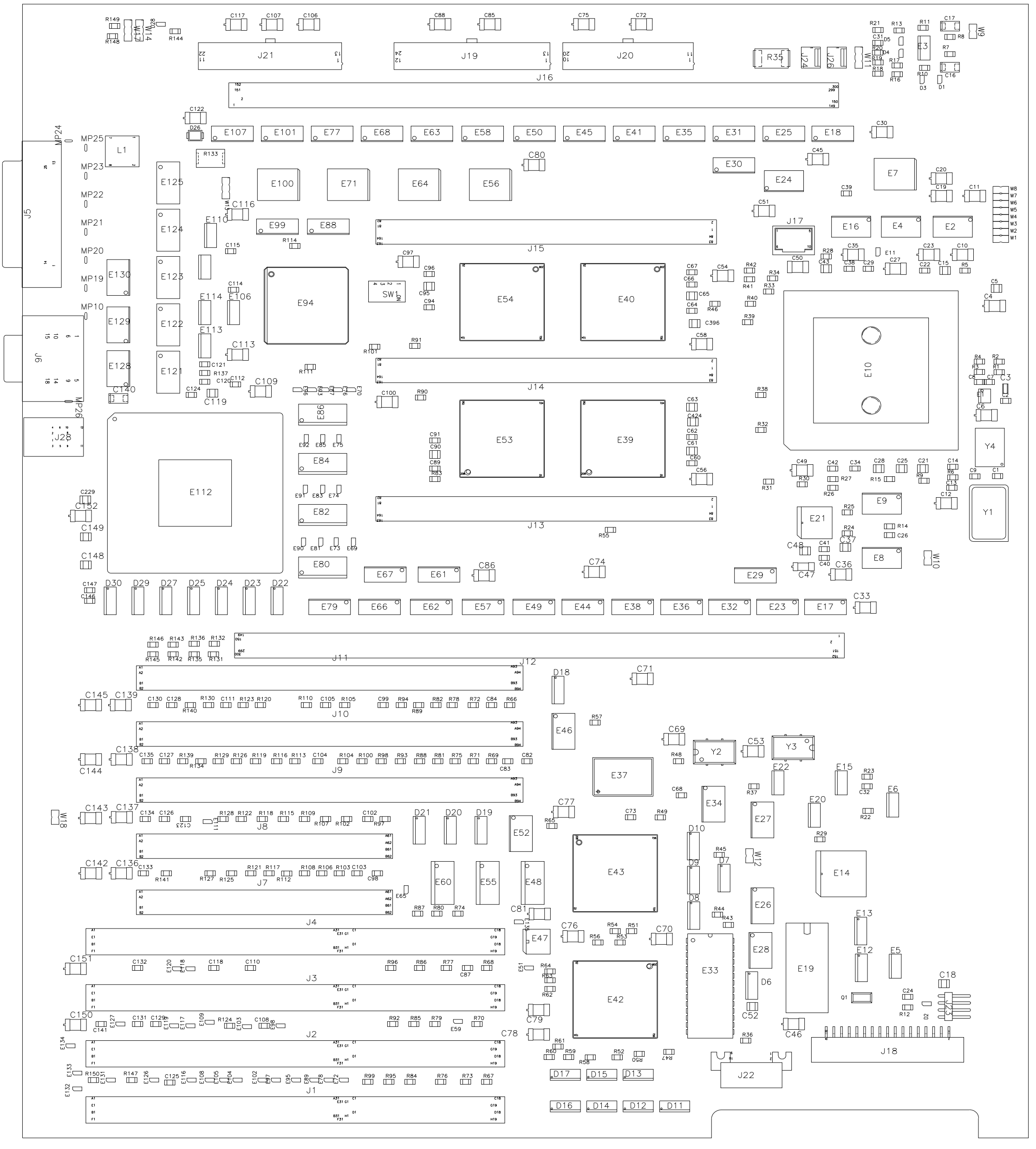
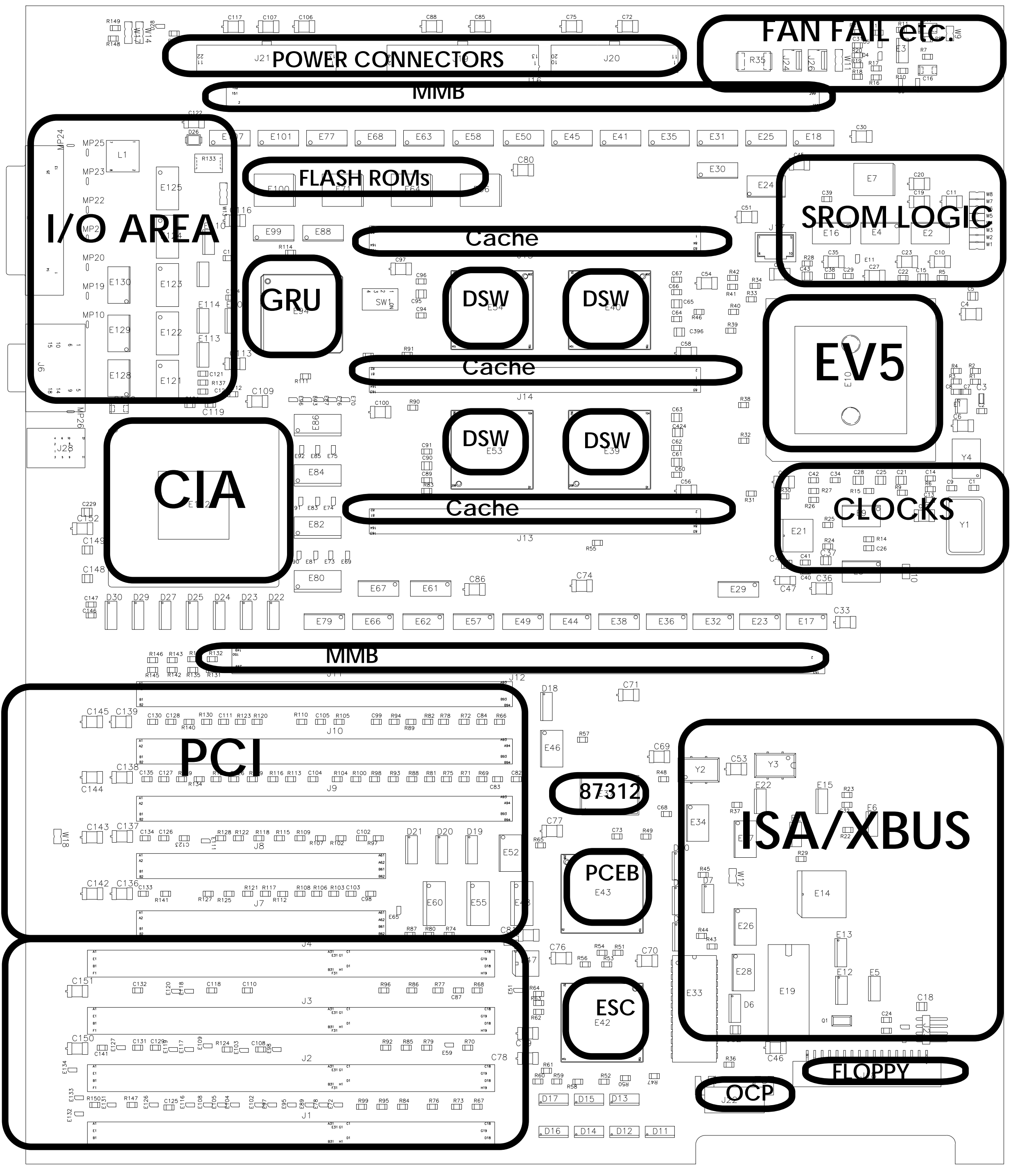


Figure 6-3 AlphaStation 600 System Board Function Map



The function map shows roughly what logic functions are in what area of the system board. Another useful map showing logical/physical relationships is the Cache/Memory map. Figure 6-4 shows the AlphaStation 600 system cache, data switch, CPU, and memory in their approximate physical locations on the system board. The numbers shown inside the components represent the CPU words that reside in that physical entity. In the case of EV5, cache and memory, the order in which the words are shown roughly corresponds to where the bits of the given word are pinned. A table of word versus bit range is also given as a reference in Table 6-2. This is useful when looking at schematics to determine which bits fall into which words, etc.

Table 6-2 Data Word / Bit Range Map

Word Number	Data Bits Only	ECC Bits Only	Data and ECC (mem data)
0	15:0	1:0	17:0
1	31:16	3:2	35:18
2	47:32	5:4	53:36
3	63:48	7:6	71:54
4	79:64	9:8	89:72
5	95:80	11:10	107:90
6	111:96	13:12	125:108
7	127:112	15:14	143:126
8	143:128		161:144
9	159:144		179:162
10	175:160		197:180
11	191:176		215:198
12	207:192		233:216
13	223:208		251:234
14	239:224		269:252
15	255:240		287:270

Figure 6-4   Memory Data Mapping

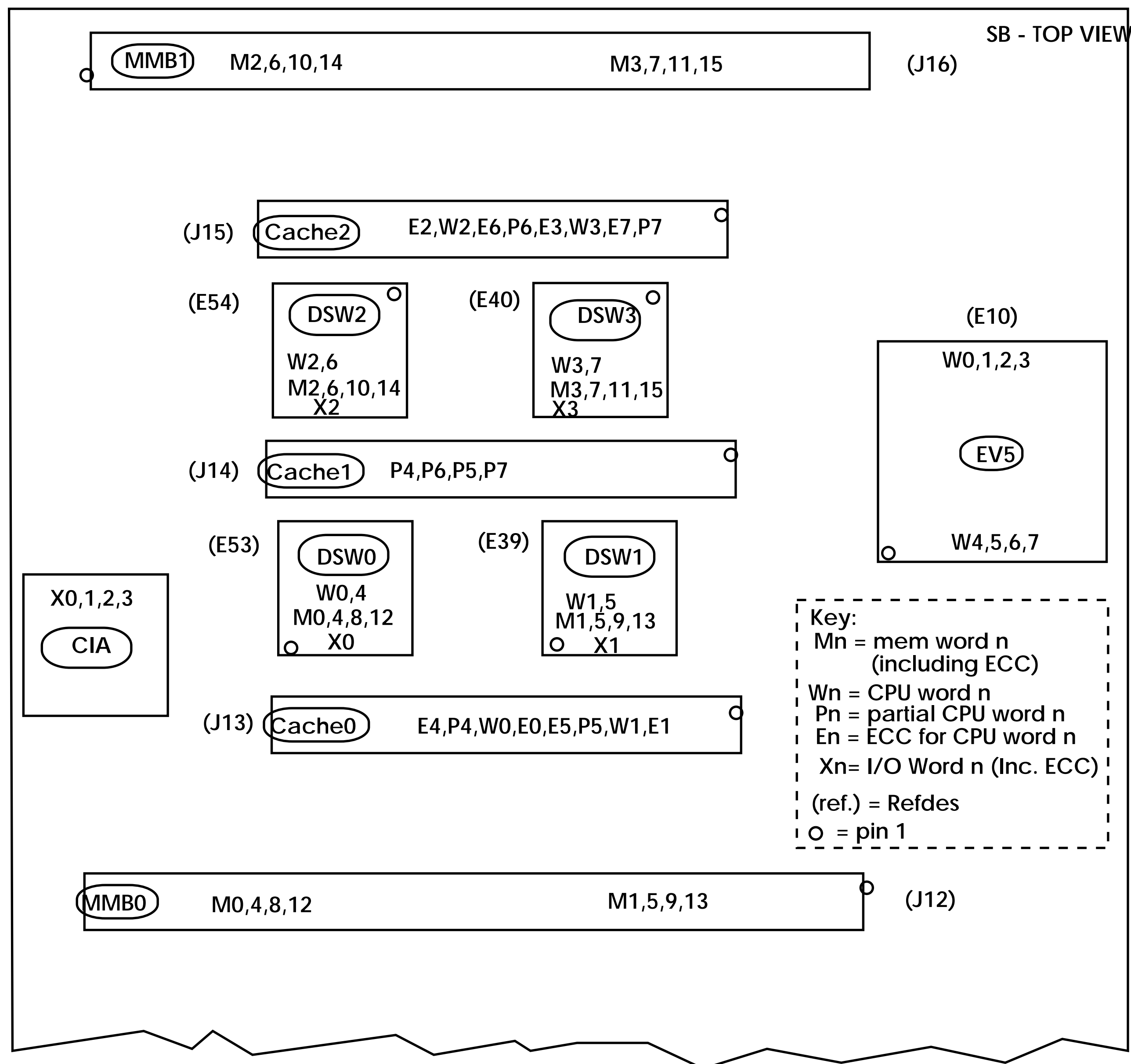


Figure 6-4 shows how CPU, memory, and I/O data is connected to the EV5, DSW, memory, cache, and CIA chip. Finer detail of exact bit assignments should be referenced from the AlphaStation 600 System Board schematics, MDA file **B_CS_5423242_0_0_AX02_ALL.PS**,or succeeding revision.

This map is intended to roughly show how data is routed on the system board; actual routing paths or nets are not shown.

The mapping shown above is the result of several constraints, mostly physical in nature, which drove the assignment of data switch, memory, and cache pinning, as well as the orientation of the CPU. Some of these constraints are:

- Due to pinning constraints, the data is sliced among four physical DSW chips. The same chip design must be used in all four chips, thus each DSW chip must have 1/4 of every bus: I/O, CPU, and memory.

- The I/O system must have access to the lower 64 bits of the EV5 data, that is, words 3-0. This, along with the previous constraint, caused words 0,1,2,3 to be sent to different data switches.
- Keeping cache address lines short required rotation of EV5 into the current position; a more optimal data routing position would be +90 degrees, to allow data to travel horizontally across without crisscrossing in the vertical. However, doing so lengthens the address lines enough to offset any gain in timing.
- Cache SIMM bit assignments were picked in order to minimize the overall CPU data line length. Word positions were chosen to be as close as possible to the DSW pins with the same word.
- Memory assignments were chosen based on the need to keep words within a set together, and the desire not to have memory lines crossing the already congested cache area (that is, DSWs talk to the MMB that is closest to them). Words are also organized to minimize memory data etch length; DSWs talk to the closest side of the MMB to them.

I/O Subsystem Organization

The AlphaStation 600 I/O subsystem is mostly confined to the system board, but some I/O functions reside on PCI or EISA/ISA option cards. The table below summarizes the AlphaStation 600 system I/O physical and logical locations:

I/O Function	Connector Location	Driven By	Bus Name
Keyboard	System Board (J28)	8242 (E14)	X-BUS (ISA subset)
Mouse	System Board (J28)	8242 (E14)	X-BUS (ISA subset)
Serial Ports (2)	System Board (J6)	87312 (E37)	ISA Bus (EISA subset)
Parallel Port	System Board (J5)	87312 (E37)	ISA Bus (EISA subset)
Floppy Drive	System Board (J18)	87312 (E37)	ISA Bus (EISA subset)
Operator Control Panel (OCP)	System Board (J22)	8584 (E28)	I2C Bus (via X-BUS)
Graphics	TGAx graphics		PCI
Ethernet	I/O Subsystem Card	I/O Subsystem	PCI
SCSI	I/O Subsystem card	I/O Subsystem	PCI
Sound/Multimedia	Microsoft Audio		ISA

AlphaStation 600 Module Overview

System Board

The AlphaStation 600 system board is the motherboard for the AlphaStation 600 system, and serves as a central interconnect for other AlphaStation 600 system components. The following is a brief summary of the major components of the AlphaStation 600 system board:

The AlphaStation 600 system board contains all the power, I/O, and module connectors for the system. These connectors are listed in the table below, along with their DEC Part Numbers, and their reference designators.

Table 6-3 AlphaStation 600 Interconnect Reference

Connector Part Number	Function	Designator
12-38939-01	Serial Port	J6
12-32998-02	Parallel Port	J5
12-14978-02	Fan Conn.	J24, J26
12-27247-05	Floppy Connector	J18
12-14630-03	Test Port	J17
12-14434-33	Speaker Connector	J23
12-19039-03	OCP Connector	J22
12-29570-06	Power Connector 5V	J19
12-29570-08	Power Connector 3V	J20
12-29570-11	Power Control	J21
12-33538-02	EISA 32 bit	J1-J4
12-39839-06	PCI 32 bit	J7,J8
12-39839-10	PCI 64 bit	J9-J11
12-39839-11	Cache Connector	J13-J15
12-39839-12	Memory Connector	J12,J16

The following sections briefly describe the major components of the system board:

Memory Motherboard

The Memory Motherboard (MMB) provides physical space for SIMM connectors, as well as fanout drivers for memory control lines. These MMB modules are similar in form factor and function to the MMB modules used on previous Alpha workstations, but there are several significant differences, described in Table 6-4.

Table 6-4 AlphaStation 600 MMB Feature Comparison

Feature	AlphaStation 600 MMB	MMB
MMB layers / etch width	8 / 5	8 / 5
SMT technology	Double sided SMT	Single sided SMT
MMB's per system (min/max)	2/2	4/4
SIMMs per MMB (min/max)	4/16	2/8
Total System Memory Capacity Using 16 MB RAMs	1.0 GB	1.0 GB
Logical Sets (AlphaStation 600) or Banks per System	8	8
SIMMs per Set (or Bank) per MMB	4	2
System Memory Data Bus	256 bits + ECC	256 bits + ECC
Minimum Data write size	128 bits	32 bits
Memory Data Bus cycle time	30ns	40 ns

Cache SIMM

The AlphaStation 600 system supports a third-level, direct-mapped write-back cache (also called BCache) in the form of SIMM modules on the system board. The system board has three cache SIMMs, which function together as a single logical unit; three SIMMs must always be in place for the BCache to function. SIMMs can use different sized SRAMs, and can be full or half populated, allowing different cache sizes. See Table 6-5 for a summary of Bcache features.

Table 6-5 BCache Features

Feature	AlphaStation 600 Cache
SIMM Layers / Etch Width	8 / 5
SIMMs per System	3
Cache Memory Size (min/max)	2 MB/4 MB (8/16 MB future)
Cache data width	128 bits
CPU read cycle time (min)	25 ns

PCI Options

The AlphaStation 600 system supports up to three 64-bit PCI options and two 32-bit PCI options. One of the 32-bit option slots is shared with an EISA slot (that is, either an EISA or PCI option may occupy that position in the cabinet, but not both simultaneously. The PCI system is controlled from the CIA, and also includes a bridge chip at the end of the PCI bus which connects to the EISA I/O system (see below). The PCI options plug into the system board via connectors at the lower left area of the system board (J7-J11); 64-bit options must plug in to the upper three connectors (J9-J11), while 32-bit options may plug in anywhere, in a 64-bit or 32-bit slot. All system board connectors are wired for 5 volt PCI options (connector orientation is the 5V system variety; 3V is supplied to the options, however).

PCI options are identified by their ID Select lines (IDSEL). Each option has its ID select tied to a different PCI address line so that it can be identified uniquely during configuration. The table below shows the PCI address for the various slots in the AlphaStation 600 system.

Table 6-6 PCI Slot Assignments

Slot	Designator	Size	PCI AD [31:0] (hex)
S0	J9	64B	0040 0000 (ID=AD22)
S1	J10	64B	0080 0000 (ID=AD23)
S2	J11	64B	0100 0000 (ID=AD24)
S3	J7	32B	0200 0000 (ID=AD25)
S4	J8	32B	0400 0000 (ID=AD26)
PCEB	E11	32B	0800 0000 (ID=AD27)

For more information about PCI pinouts and operation, see the *PCI System Specification V2.0.*

EISA Options

The AlphaStation 600 system supports an EISA/ISA subsystem, including up to four EISA or ISA plug-in option cards. There are four EISA connectors, J1-J4, in the lower left corner of the system board, one of which (J4) is a shared connector with a PCI slot. (See Table 6-6). EISA I/O operations are controlled from the Intel PCI/EISA bridge chip set.

AlphaStation 600 SystemBoard - ASICs

In the pinouts that follow, the key is: I=Input, O=Output, B=Bi-directional, N=Not Connected, P=Power. All pinouts for ASICs and connectors are shown top view.

EV5 CPU

The EV5 CPU is a follow-on to the EV4 processor, the Alpha processor used in previous workstations and PC's. There are many architectural differences between the two; EV5 is a new design, not simply a scaling of the current design (like EV45). Some important differences from the system perspective:

- More pins - EV5 has 499 pins in the package, compared to (431) for EV4.
- The EV5 package is an interstitial pin grid array, where pins are 100 mils apart from each other in a row, and the rows are offset by 50 mils from each other, and separated by 50 mils from each other. Compared to the EV4 package, which had a standard 100x100 mil PGA grid, the pin density is higher. The result is a package with more pins, in a smaller footprint than EV4. This allows good signal integrity characteristics, increased pin count, etc. in the same footprint. On the down side, routing is more difficult, and debug probing is much tighter.
- External Cache now has a private set of address lines (INDEX) so that the cache does not have to share loading with other system components.
- A third level of cache. The external cache on EV5 is a third level, as opposed to EV4 second level cache. The EV5 has an onboard "S-cache", which is broken into three sets. Total S-cache capacity is 96K bytes. The third level cache can be up to 16 MB in size on an AlphaStation 600 system.

Figure 6-5 EV5 CPU Package - Top View

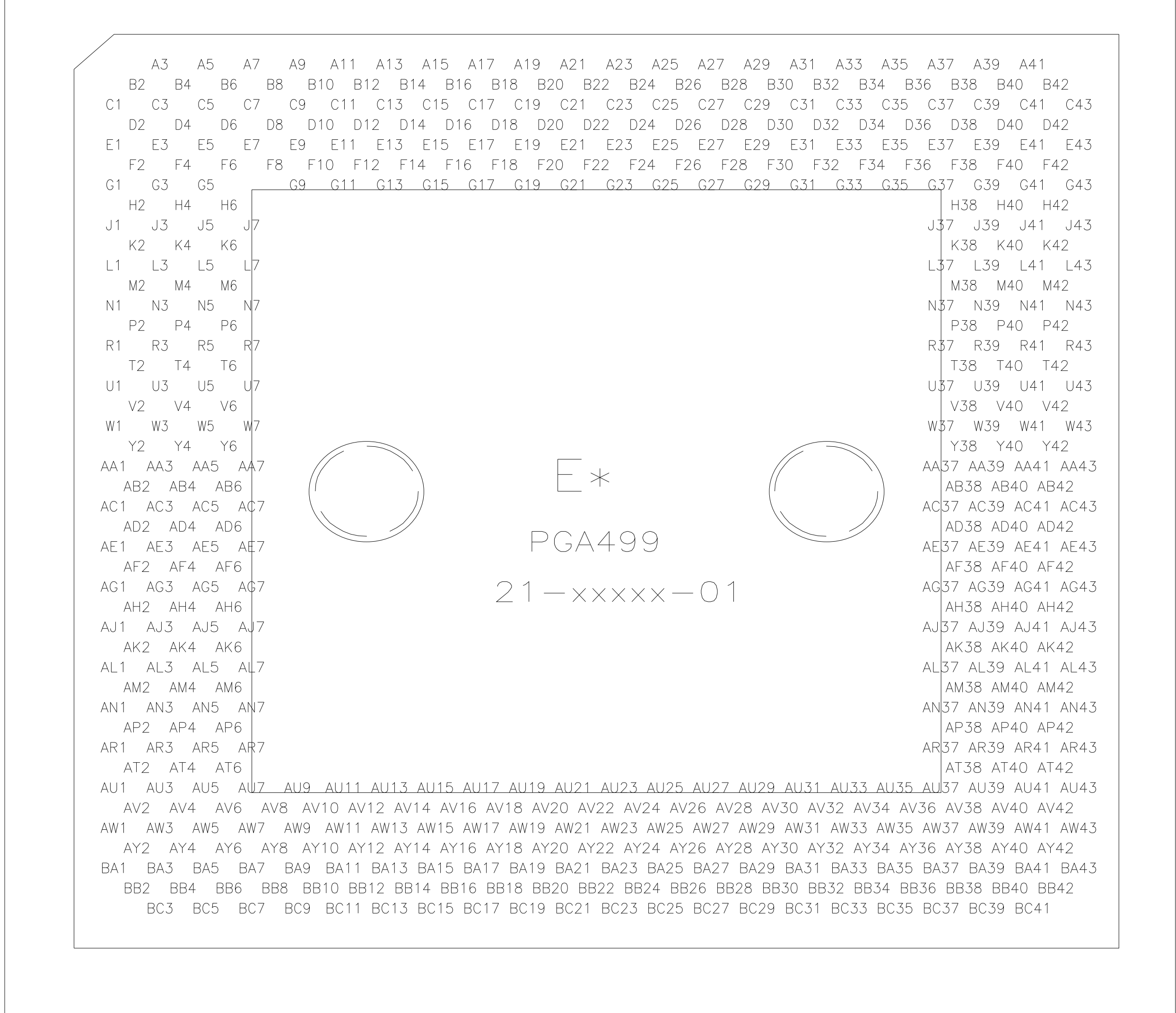


Table 6-7 EV5 PIN OUT - Sorted by Pin Number

PIN	SIGNAL	USE	PIN	SIGNAL	USE
A11	TAG_DATA_H[32]	B	AG1	DATA_H[99]	B
A13	TAG_DATA_H[36]	B	AG3	DATA_H[100]	B
A15	TAG_SHARED_H	B	AG37	GND	P
A17	SCACHE_SET_H[1]	O	AG39	DATA_H[38]	B
A19	CMD_H[1]	B	AG41	DATA_H[36]	B
A21	TAG_RAM_WE_H	O	AG43	DATA_H[35]	B
A23	DATA_RAM_WE_H	O	AG5	DATA_H[102]	B
A25	FILL_ERROR_H	I	AG7	GND	P
A27	IDLE_BC_H	I	AH2	PWR3	P
A29	INDEX_H[4]	O	AH38	DATA_H[41]	B
A3	GND	P	AH4	GND	P
A31	INDEX_H[9]	O	AH40	GND	P
A33	INDEX_H[13]	O	AH42	PWR3	P
A35	INDEX_H[17]	O	AH6	DATA_H[105]	B
A37	INDEX_H[21]	O	AJ1	DATA_H[103]	B
A39	INDEX_H[25]	O	AJ3	DATA_H[104]	B
A41	GND	P	AJ37	PWR3	P
A43	GND	P	AJ39	DATA_H[42]	B
A5	TAG_DATA_H[20]	B	AJ41	DATA_H[40]	B
A7	TAG_DATA_H[24]	B	AJ43	DATA_H[39]	B
A9	TAG_DATA_H[28]	B	AJ5	DATA_H[106]	B
AA1	DATA_H[89]	B	AJ7	PWR3	P
AA3	DATA_H[88]	B	AK2	DATA_H[107]	B
AA37	GND	P	AK38	DATA_H[46]	B
AA39	DATA_H[23]	B	AK4	PWR3	P
AA41	DATA_H[24]	B	AK40	PWR3	P
AA43	DATA_H[25]	B	AK42	DATA_H[43]	B
AA5	DATA_H[87]	B	AK6	DATA_H[110]	B
AA7	GND	P	AL1	DATA_H[108]	B
AB2	PWR3	P	AL3	DATA_H[109]	B
AB38	DATA_H[26]	B	AL37	GND	P
AB4	PWR3	P	AL39	DATA_H[47]	B
AB40	PWR3	P	AL41	DATA_H[45]	B
AB42	PWR3	P	AL43	DATA_H[44]	B
AB6	DATA_H[90]	B	AL5	DATA_H[111]	B
AC1	DATA_H[91]	B	AL7	GND	P
AC3	DATA_H[92]	B	AM2	PWR3	P
AC37	GND	P	AM38	DATA_H[50]	B
AC39	DATA_H[29]	B	AM4	GND	P
AC41	DATA_H[28]	B	AM40	GND	P
AC43	DATA_H[27]	B	AM42	PWR3	P
AC5	DATA_H[93]	B	AM6	DATA_H[114]	B
AC7	GND	P	AN1	DATA_H[112]	B
AD2	DATA_H[94]	B	AN3	DATA_H[113]	B
AD38	DATA_H[31]	B	AN37	PWR3	P
AD4	GND	P	AN39	DATA_H[51]	B
AD40	GND	P	AN41	DATA_H[49]	B
AD42	DATA_H[30]	B	AN43	DATA_H[48]	B
AD6	DATA_H[95]	B	AN5	DATA_H[115]	B
AE1	DATA_H[96]	B	AN7	PWR3	P
AE3	DATA_H[97]	B	AP2	GND	P
AE37	PWR3	P	AP38	DATA_H[54]	B
AE39	DATA_H[34]	B	AP4	PWR3	P
AE41	DATA_H[33]	B	AP40	PWR3	P
AE43	DATA_H[32]	B	AP42	GND	P
AE5	DATA_H[98]	B	AP6	DATA_H[118]	B
AE7	PWR3	P	AR1	DATA_H[116]	B
AF2	GND	P	AR3	DATA_H[117]	B
AF38	DATA_H[37]	B	AR37	GND	P
AF4	PWR3	P	AR39	DATA_H[55]	B
AF40	PWR3	P	AR41	DATA_H[53]	B
AF42	GND	P	AR43	DATA_H[52]	B
AF6	DATA_H[101]	B	AR5	DATA_H[119]	B

PIN	SIGNAL	USE
AR7	GND	P
AT2	PWR3	P
AT38	DATA_H[58]	B
AT4	GND	P
AT40	GND	P
AT42	PWR3	P
AT6	DATA_H[122]	B
AU1	DATA_H[120]	B
AU11	PWR3	P
AU13	GND	P
AU15	PWR3	P
AU17	GND	P
AU19	PWR3	P
AU21	CLK_MODE_H[0]	I
AU23	DC_OK_H	I
AU25	MCH_HLT_IRQ_H	I
AU27	IRQ_H[1]	I
AU29	PWR3	P
AU3	DATA_H[121]	B
AU31	GND	P
AU33	PWR3	P
AU35	GND	P
AU37	PWR3	P
AU39	DATA_H[59]	B
AU41	DATA_H[57]	B
AU43	DATA_H[56]	B
AU5	DATA_H[123]	B
AU7	PWR3	P
AU9	GND	P
AV10	ADDR_H[15]	B
AV12	ADDR_H[11]	B
AV14	ADDR_H[7]	B
AV16	TEST_STATUS_H[1]	O
AV18	TMS_H	I
AV2	GND	P
AV20	SROM_PRESENT_L	I
AV22	GND	P
AV24	SYS_CLK_OUT2_H	O
AV26	PWR_FAIL_IRQ_H	I
AV28	NC	N
AV30	ADDR_H[36]	B
AV32	ADDR_H[32]	B
AV34	ADDR_H[28]	B
AV36	ADDR_H[24]	B
AV38	DATA_H[62]	B
AV4	PWR3	P
AV40	PWR3	P
AV42	GND	P
AV6	DATA_H[126]	B
AV8	ADDR_H[19]	B
AW1	DATA_H[124]	B
AW11	ADDR_H[12]	B
AW13	ADDR_H[8]	B
AW15	TEMP_SENS_H	I
AW17	TCK_H	I
AW19	SROM_OE_L	O
AW21	GND	P
AW23	SYS_CLK_OUT1_H	O
AW25	REF_CLK_IN_H	I
AW27	IRQ_H[3]	I
AW29	PERF_MON_H	I
AW3	DATA_H[125]	B
AW31	ADDR_H[35]	B
AW33	ADDR_H[31]	B
AW35	ADDR_H[27]	B
AW37	ADDR_H[23]	B
AW39	DATA_H[63]	B
AW41	DATA_H[61]	B
AW43	DATA_H[60]	B
AW5	DATA_H[127]	B
AW7	ADDR_H[20]	B
AW9	ADDR_H[16]	B
AY10	PWR3	P
AY12	GND	P
AY14	PWR3	P
AY16	GND	P
AY18	PWR3	P
AY2	PWR3	P
AY20	PORT_MODE_H[0]	I
AY22	GND	P
AY24	GND	P
AY26	PWR3	P
AY28	GND	P
AY30	PWR3	P
AY32	GND	P
AY34	PWR3	P
AY36	GND	P
AY38	PWR3	P
AY4	GND	P
AY40	GND	P
AY42	PWR3	P
AY6	PWR3	P
AY8	GND	P
B10	GND	P
B12	PWR3	P
B14	TAG_DATA_H[37]	B
B16	PWR3	P
B18	GND	P
B2	GND	P
B20	ADDR_CMD_PAR_H	B
B22	PWR3	P
B24	DACK_H	I
B26	GND	P
B28	PWR3	P
B30	INDEX_H[8]	O
B32	PWR3	P
B34	GND	P
B36	PWR3	P
B38	GND	P
B4	PWR3	P
B40	PWR3	P
B42	GND	P
B6	GND	P
B8	PWR3	P
BA1	GND	P
BA11	ADDR_H[10]	B
BA13	ADDR_H[6]	B
BA15	TEST_STATUS_H[0]	O
BA17	TDO_H	O
BA19	SROM_CLK_H	O
BA21	GND	P
BA23	CLK_MODE_H[1]	I
BA25	CPU_CLK_OUT_H	O
BA27	SYS_MCH_CHK_IRQ_H	I
BA29	IRQ_H[0]	I
BA3	PWR3	P
BA31	ADDR_H[37]	B
BA33	ADDR_H[33]	B
BA35	ADDR_H[29]	B
BA37	ADDR_H[25]	B

BPIN	SIGNAL	USE
BA41	PWR3	P
BA43	GND	P
BA5	PWR3	P
BA7	ADDR_H[18]	B
BA9	ADDR_H[14]	B
BB10	GND	P
BB12	PWR3	P
BB14	ADDR_H[4]	B
BB16	PWR3	P
BB18	GND	P
BB2	GND	P
BB20	PORT_MODE_H[1]	I
BB22	OSC_CLK_IN_L	I
BB24	SYS_CLK_OUT1_L	O
BB26	GND	P
BB28	PWR3	P
BB30	ADDR_H[39]	B
BB32	PWR3	P
BB34	GND	P
BB36	PWR3	P
BB38	GND	P
BB4	PWR3	P
BB40	PWR3	P
BB42	GND	P
BB6	GND	P
BB8	PWR3	P
BC1	GND	P
BC11	ADDR_H[9]	B
BC13	ADDR_H[5]	B
BC15	TRST_L	I
BC17	TDI_H	I
BC19	SROM_DAT_H	I
BC21	OSC_CLK_IN_H	I
BC23	PWR3	P
BC25	SYS_CLK_OUT2_L	O
BC27	SYS_RESET_L	I
BC29	IRQ_H[2]	I
BC3	GND	P
BC31	ADDR_H[38]	B
BC33	ADDR_H[34]	B
BC35	ADDR_H[30]	B
BC37	ADDR_H[26]	B
BC39	ADDR_H[22]	B
BC41	GND	P
BC43	GND	P
BC5	ADDR_H[21]	B
BC7	ADDR_H[17]	B
BC9	ADDR_H[13]	B
C1	GND	P
C11	TAG_DATA_H[31]	B
C13	TAG_DATA_H[35]	B
C15	TAG_DATA_PAR_H	B
C17	SCACHE_SET_H[0]	O
C19	CMD_H[2]	B
C21	TAG_RAM_OE_H	O
C23	NC	N
C25	CFAIL_H	I
C27	ADDR_RES_H[0]	O
C29	INDEX_H[5]	O
C3	PWR3	P
C31	INDEX_H[10]	O
C33	INDEX_H[14]	O
C35	INDEX_H[18]	O
C37	INDEX_H[22]	O
C39	PWR3	P

PIN	SIGNAL	USE
C41	PWR3	P
C43	GND	P
C5	PWR3	P
C7	TAG_DATA_H[23]	B
C9	TAG_DATA_H[27]	B
D10	PWR3	P
D12	GND	P
D14	PWR3	P
D16	GND	P
D18	PWR3	P
D2	PWR3	P
D20	GND	P
D22	PWR3	P
D24	GND	P
D26	PWR3	P
D28	GND	P
D30	PWR3	P
D32	GND	P
D34	PWR3	P
D36	GND	P
D38	PWR3	P
D4	GND	P
D40	GND	P
D42	PWR3	P
D6	PWR3	P
D8	GND	P
E1	DATA_CHECK_H[15]	B
E11	TAG_DATA_H[29]	B
E13	TAG_DATA_H[33]	B
E15	TAG_DATA_H[38]	B
E17	TAG_DIRTY_H	B
E19	CMD_H[3]	B
E21	VICTIM_PENDING_H	O
E23	ADDR_BUS_REQ_H	I
E25	DATA_BUS_REQ_H	I
E27	NC	N
E29	INDEX_H[7]	O
E3	INT4_VALID_H[3]	O
E31	INDEX_H[12]	O
E33	INDEX_H[16]	O
E35	INDEX_H[20]	O
E37	INDEX_H[24]	O
E39	NC	N
E41	INT4_VALID_H[1]	O
E43	DATA_CHECK_H[7]	B
E5	NC	N
E7	TAG_DATA_H[21]	B
E9	TAG_DATA_H[25]	B
F10	TAG_DATA_H[26]	B
F12	TAG_DATA_H[30]	B
F14	TAG_DATA_H[34]	B
F16	TAG_VALID_H	B
F18	TAG_CTL_PAR_H	B
F2	GND	P
F20	CMD_H[0]	B
F22	DATA_RAM_OE_H	O
F24	FILL_ID_H	I
F26	ADDR_RES_H[1]	O
F28	INDEX_H[6]	O
F30	INDEX_H[11]	O
F32	INDEX_H[15]	O
F34	INDEX_H[19]	O
F36	INDEX_H[23]	O
F38	INT4_VALID_H[0]	O
F4	PWR3	P
F40	PWR3	P

PIN	SIGNAL	USE
F42	GND	P
F6	INT4_VALID_H[2]	O
F8	TAG_DATA_H[22]	B
G1	DATA_CHECK_H[11]	B
G11	PWR3	P
G13	GND	P

Table 6-8 EV5 PINS - Alphabetic Order

PIN	SIGNAL	USE
E23	ADDR_BUS_REQ_H	I
B20	ADDR_CMD_PAR_H	B
BA11	ADDR_H[10]	B
AV12	ADDR_H[11]	B
AW11	ADDR_H[12]	B
BC9	ADDR_H[13]	B
BA9	ADDR_H[14]	B
AV10	ADDR_H[15]	B
AW9	ADDR_H[16]	B
BC7	ADDR_H[17]	B
BA7	ADDR_H[18]	B
AV8	ADDR_H[19]	B
AW7	ADDR_H[20]	B
BC5	ADDR_H[21]	B
BC39	ADDR_H[22]	B
AW37	ADDR_H[23]	B
AV36	ADDR_H[24]	B
BA37	ADDR_H[25]	B
BC37	ADDR_H[26]	B
AW35	ADDR_H[27]	B
AV34	ADDR_H[28]	B
BA35	ADDR_H[29]	B
BC35	ADDR_H[30]	B
AW33	ADDR_H[31]	B
AV32	ADDR_H[32]	B
BA33	ADDR_H[33]	B
BC33	ADDR_H[34]	B
AW31	ADDR_H[35]	B
AV30	ADDR_H[36]	B
BA31	ADDR_H[37]	B
BC31	ADDR_H[38]	B
BB30	ADDR_H[39]	B
BB14	ADDR_H[4]	B
BC13	ADDR_H[5]	B
BA13	ADDR_H[6]	B
AV14	ADDR_H[7]	B
AW13	ADDR_H[8]	B
BC11	ADDR_H[9]	B
C27	ADDR_RES_H[0]	O
F26	ADDR_RES_H[1]	O
G21	CACK_H	I
C25	CFAIL_H	I
AU21	CLK_MODE_H[0]	I
BA23	CLK_MODE_H[1]	I
F20	CMD_H[0]	B
A19	CMD_H[1]	B
C19	CMD_H[2]	B
E19	CMD_H[3]	B
BA25	CPU_CLK_OUT_H	O
B24	DACK_H	I
E25	DATA_BUS_REQ_H	I
J41	DATA_CHECK_H[0]	B
J5	DATA_CHECK_H[10]	B
G1	DATA_CHECK_H[11]	B
G3	DATA_CHECK_H[12]	B
H6	DATA_CHECK_H[13]	B
G5	DATA_CHECK_H[14]	B
E1	DATA_CHECK_H[15]	B
K38	DATA_CHECK_H[1]	B
J39	DATA_CHECK_H[2]	B
G43	DATA_CHECK_H[3]	B
G41	DATA_CHECK_H[4]	B
H38	DATA_CHECK_H[5]	B
G39	DATA_CHECK_H[6]	B
E43	DATA_CHECK_H[7]	B
J3	DATA_CHECK_H[8]	B
K6	DATA_CHECK_H[9]	B
J43	DATA_H[0]	B
AG3	DATA_H[100]	B
AF6	DATA_H[101]	B
AG5	DATA_H[102]	B
AJ1	DATA_H[103]	B
AJ3	DATA_H[104]	B
AH6	DATA_H[105]	B
AJ5	DATA_H[106]	B
AK2	DATA_H[107]	B
AL1	DATA_H[108]	B
AL3	DATA_H[109]	B
R39	DATA_H[10]	B
AK6	DATA_H[110]	B
AL5	DATA_H[111]	B
AN1	DATA_H[112]	B
AN3	DATA_H[113]	B
AM6	DATA_H[114]	B
AN5	DATA_H[115]	B
AR1	DATA_H[116]	B
AR3	DATA_H[117]	B
AP6	DATA_H[118]	B
AR5	DATA_H[119]	B
T38	DATA_H[11]	B
AU1	DATA_H[120]	B
AU3	DATA_H[121]	B
AT6	DATA_H[122]	B
AU5	DATA_H[123]	B
AW1	DATA_H[124]	B
AW3	DATA_H[125]	B
AV6	DATA_H[126]	B
AW5	DATA_H[127]	B
R41	DATA_H[12]	B
R43	DATA_H[13]	B
U39	DATA_H[14]	B
V38	DATA_H[15]	B
U41	DATA_H[16]	B
U43	DATA_H[17]	B
W39	DATA_H[18]	B
W41	DATA_H[19]	B
L39	DATA_H[1]	B
W43	DATA_H[20]	B
Y38	DATA_H[21]	B
Y42	DATA_H[22]	B
AA39	DATA_H[23]	B
AA41	DATA_H[24]	B
AA43	DATA_H[25]	B
AB38	DATA_H[26]	B
AC43	DATA_H[27]	B
AC41	DATA_H[28]	B
AC39	DATA_H[29]	B
M38	DATA_H[2]	B
AD42	DATA_H[30]	B
AD38	DATA_H[31]	B

PIN	SIGNAL	USE	PIN	SIGNAL	USE
AE43	DATA_H[32]	B	AY30	PWR3	P
AE41	DATA_H[33]	B	AY34	PWR3	P
AE39	DATA_H[34]	B	AY38	PWR3	P
AG43	DATA_H[35]	B	AY42	PWR3	P
AG41	DATA_H[36]	B	AY6	PWR3	P
AF38	DATA_H[37]	B	B12	PWR3	P
AG39	DATA_H[38]	B	B16	PWR3	P
AJ43	DATA_H[39]	B	B22	PWR3	P
L41	DATA_H[3]	B	B28	PWR3	P
AJ41	DATA_H[40]	B	B32	PWR3	P
AH38	DATA_H[41]	B	B36	PWR3	P
AJ39	DATA_H[42]	B	B4	PWR3	P
AK42	DATA_H[43]	B	B40	PWR3	P
E41	INT4_VALID_H[1]	O	B8	PWR3	P
F6	INT4_VALID_H[2]	O	BA3	PWR3	P
E3	INT4_VALID_H[3]	O	BA39	PWR3	P
BA29	IRQ_H[0]	I	BA41	PWR3	P
AU27	IRQ_H[1]	I	BA5	PWR3	P
BC29	IRQ_H[2]	I	BB12	PWR3	P
AW27	IRQ_H[3]	I	BB16	PWR3	P
AU25	MCH_HLT_IRQ_H	I	BB28	PWR3	P
AV28	NC	N	BB32	PWR3	P
C23	NC	N	BB36	PWR3	P
E27	NC	N	BB4	PWR3	P
E39	NC	N	BB40	PWR3	P
E5	NC	N	BB8	PWR3	P
BC21	OSC_CLK_IN_H	I	BC23	PWR3	P
BB22	OSC_CLK_IN_L	I	C3	PWR3	P
AW29	PERF_MON_H	I	C39	PWR3	P
AY20	PORT_MODE_H[0]	I	C41	PWR3	P
BB20	PORT_MODE_H[1]	I	C5	PWR3	P
G33	PWR	P	D10	PWR3	P
AB2	PWR3	P	D14	PWR3	P
AB4	PWR3	P	D18	PWR3	P
AB40	PWR3	P	D2	PWR3	P
AB42	PWR3	P	D22	PWR3	P
AE37	PWR3	P	D26	PWR3	P
AE7	PWR3	P	D30	PWR3	P
AF4	PWR3	P	D34	PWR3	P
AF40	PWR3	P	D38	PWR3	P
AH2	PWR3	P	D42	PWR3	P
AH42	PWR3	P	D6	PWR3	P
AJ37	PWR3	P	F4	PWR3	P
AJ7	PWR3	P	F40	PWR3	P
AK4	PWR3	P	G11	PWR3	P
AK40	PWR3	P	G15	PWR3	P
AM2	PWR3	P	G19	PWR3	P
AM42	PWR3	P	G29	PWR3	P
AN37	PWR3	P	G37	PWR3	P
AN7	PWR3	P	G7	PWR3	P
AP4	PWR3	P	H2	PWR3	P
AP40	PWR3	P	H42	PWR3	P
AT2	PWR3	P	K4	PWR3	P
AT42	PWR3	P	K40	PWR3	P
AU11	PWR3	P	L37	PWR3	P
AU15	PWR3	P	L7	PWR3	P
AU19	PWR3	P	M2	PWR3	P
AU29	PWR3	P	M42	PWR3	P
AU33	PWR3	P	P4	PWR3	P
AU37	PWR3	P	P40	PWR3	P
AU7	PWR3	P	R37	PWR3	P
AV4	PWR3	P	R7	PWR3	P
AV40	PWR3	P	T2	PWR3	P
AY10	PWR3	P	T42	PWR3	P
AY14	PWR3	P	V4	PWR3	P
AY18	PWR3	P	V40	PWR3	P
AY2	PWR3	P			
AY26	PWR3	P			

PIN	SIGNAL	USE
W37	PWR3	P
W7	PWR3	P
AV26	PWR_FAIL_IRQ_H	I
AW25	REF_CLK_IN_H	I
C17	SCACHE_SET_H[0]	O
A17	SCACHE_SET_H[1]	O
BA19	SROM_CLK_H	O
BC19	SROM_DAT_H	I
AW19	SROM_OE_L	O
AV20	SROM_PRESENT_L	I
G27	SYSTEM_LOCK_FLAG_H	I
AW23	SYS_CLK_OUT1_H	O
BB24	SYS_CLK_OUT1_L	O
AV24	SYS_CLK_OUT2_H	O
BC25	SYS_CLK_OUT2_L	O
BA27	SYS_MCH_CHK_IRQ_H	I
BC27	SYS_RESET_L	I
F18	TAG_CTL_PAR_H	B
A5	TAG_DATA_H[20]	B
E7	TAG_DATA_H[21]	B
F8	TAG_DATA_H[22]	B
C7	TAG_DATA_H[23]	B
A7	TAG_DATA_H[24]	B
E9	TAG_DATA_H[25]	B
F10	TAG_DATA_H[26]	B
C9	TAG_DATA_H[27]	B
A9	TAG_DATA_H[28]	B
E11	TAG_DATA_H[29]	B
F12	TAG_DATA_H[30]	B
C11	TAG_DATA_H[31]	B
A11	TAG_DATA_H[32]	B
E13	TAG_DATA_H[33]	B
F14	TAG_DATA_H[34]	B
C13	TAG_DATA_H[35]	B
A13	TAG_DATA_H[36]	B
B14	TAG_DATA_H[37]	B
E15	TAG_DATA_H[38]	B
C15	TAG_DATA_PAR_H	B
E17	TAG_DIRTY_H	B
C21	TAG_RAM_OE_H	O
A21	TAG_RAM_WE_H	O
A15	TAG_SHARED_H	B
F16	TAG_VALID_H	B
AW17	TCK_H	I
BC17	TDI_H	I
BA17	TDO_H	O
AW15	TEMP_SENS_H	I
BA15	TEST_STATUS_H[0]	O
AV16	TEST_STATUS_H[1]	O
AV18	TMS_H	I
BC15	TRST_L	I
E21	VICTIM_PENDING_H	O

DSW ASIC

The DSW ASIC is a sliced data switch design that allows the AlphaStation 600 system cache, memory, and I/O systems to talk to each other. All DSW data paths are protected by quadword ECC. Table 6-9 summarizes DSW features:

Table 6-9 DSW Features

Feature	GRU ASIC
Vendor	LSI Logic
Technology	LCA 100K CMOS
Package type	208 Pin PQFP (.5 mm pin pitch)
Number of I/O pins	158
Gate Count	25 K gates
Supply Voltage	5V
Clocking Scheme	Single ended input; internal PLL circuit
External Clock cycle time	30ns
Internal Clock cycle time	15/30 ns
Chips per AlphaStation 600 system	4

Figure 6-6 DSW Pinout - Top View

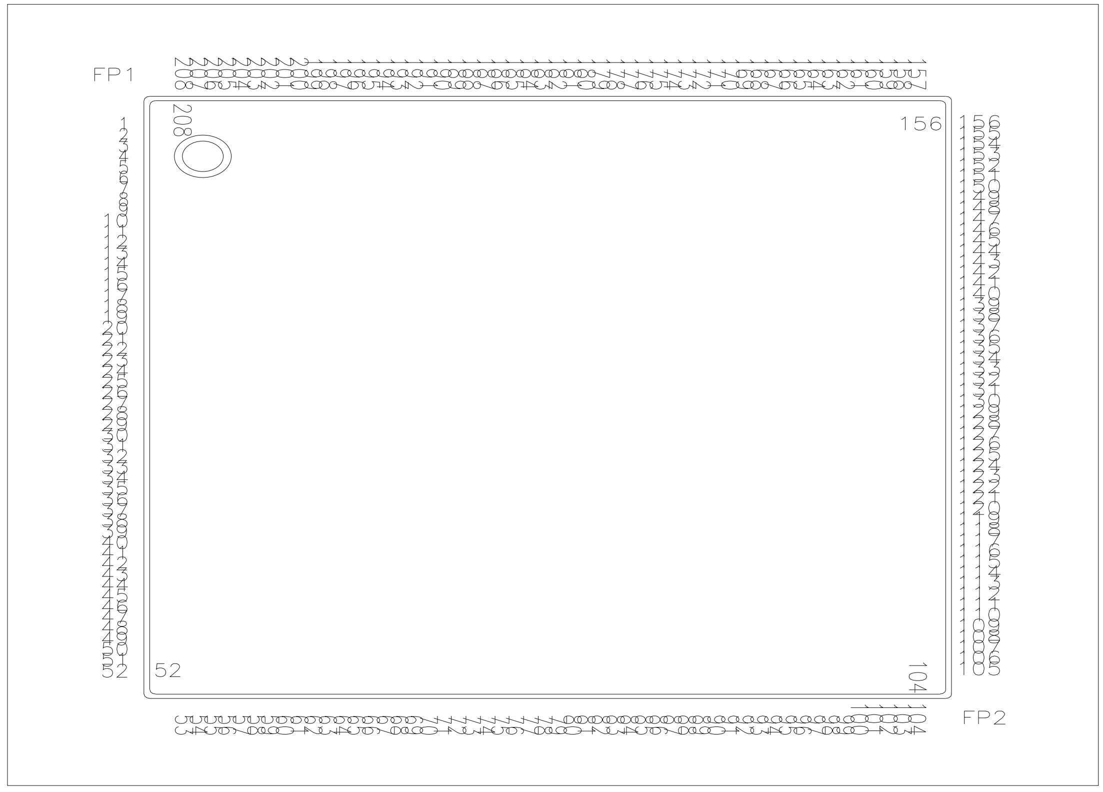


Table 6-10 DSW PIN OUT - Sorted by Pin Number

PIN	SIGNAL	USE
1	PWR5	P
2	IOD[0]	B
3	CPU_DAT[0]	B
4	CPU_DAT[1]	B
5	GND	P
6	MEM_DAT[0]	B
7	MEM_DAT[1]	B
8	MEM_DAT[2]	B
9	MEM_DAT[3]	B
10	PWR5	P
11	IOD[1]	B
12	CPU_DAT[2]	B
13	CPU_DAT[3]	B
14	GND	P
15	MEM_DAT[4]	B
16	MEM_DAT[5]	B
17	MEM_DAT[6]	B
18	MEM_DAT[7]	B
19	PWR5	P
20	IOD[2]	B
21	CPU_DAT[4]	B
22	CPU_DAT[5]	B
23	GND	P
24	MEM_DAT[8]	B
25	MEM_DAT[9]	B
26	MEM_DAT[10]	B
27	MEM_DAT[11]	B
28	PWR5	P
29	IOD[3]	B
30	CPU_DAT[6]	B
31	CPU_DAT[7]	B
32	GND	P
33	MEM_DAT[12]	B
34	MEM_DAT[13]	B
35	MEM_DAT[14]	B
36	MEM_DAT[15]	B
37	PWR5	P
38	IOC[6]	I
39	IOC[5]	I
40	IOC[4]	I
41	IOC[3]	I
42	IOC[2]	I
43	IOC[1]	I
44	GND	P
45	IOC[0]	I
46	MEM_EN	I
47	TEST_MODE[0]	I
48	TEST_MODE[1]	I
49	CONFIG[0]	I
50	CONFIG[1]	I
51	TEST_OR_SCAN_OUT	O
52	PWR5	P
53	GND	P
54	IOD[4]	B
55	CPU_DAT[8]	B
56	CPU_DAT[9]	B
57	PWR5	P
58	MEM_DAT[16]	B
59	MEM_DAT[17]	B
60	MEM_DAT[18]	B
61	MEM_DAT[19]	B
62	GND	P
63	IOD[5]	B
64	CPU_DAT[10]	B

PIN	SIGNAL	USE
65	CPU_DAT[11]	B
66	PWR5	P
67	MEM_DAT[20]	B
68	MEM_DAT[21]	B
69	MEM_DAT[22]	B
70	MEM_DAT[23]	B
71	GND	P
72	IOD[6]	B
73	CPU_DAT[12]	B
74	CPU_DAT[13]	B
75	PWR5	P
76	MEM_DAT[24]	B
77	MEM_DAT[25]	B
78	MEM_DAT[26]	B
79	MEM_DAT[27]	B
80	GND	P
81	IOD[7]	B
82	CPU_DAT[14]	B
83	CPU_DAT[15]	B
84	PWR5	P
85	MEM_DAT[28]	B
86	MEM_DAT[29]	B
87	MEM_DAT[30]	B
88	MEM_DAT[31]	B
89	GND	P
90	IOD[8]	B
91	CPU_DAT[16]	B
92	CPU_DAT[17]	B
93	PWR5	P
94	MEM_DAT[32]	B
95	MEM_DAT[33]	B
96	MEM_DAT[34]	B
97	MEM_DAT[35]	B
98	GND	P
99	CMC[4]	I
100	CMC[3]	I
101	CMC[2]	I
102	CMC[1]	I
103	CMC[0]	I
104	PWR5	P
105	PLL_CLK	I
106	GND	P
107	PLL_LP2	I
108	PWR5	P
109	PLL_AGND	I
110	PLL_VSS	I
111	GND	P
112	PLL_VDD	I
113	PLL_LP1	I
114	PWR5	P
115	RESET_L	I
116	CMC[5]	I
117	CMC[6]	I
118	CMC[7]	I
119	CMC[8]	I
120	GND	P
121	MEM_DAT[39]	B
122	MEM_DAT[38]	B
123	MEM_DAT[37]	B
124	MEM_DAT[36]	B
125	PWR5	P
126	CPU_DAT[19]	B
127	CPU_DAT[18]	B
128	IOD[9]	B
129	GND	P

PIN	SIGNAL	USE
130	MEM_DAT[43]	B
131	MEM_DAT[42]	B
132	MEM_DAT[41]	B
133	MEM_DAT[40]	B
134	PWR5	P
135	CPU_DAT[21]	B
136	CPU_DAT[20]	B
137	IOD[10]	B
138	GND	P
139	MEM_DAT[47]	B
140	MEM_DAT[46]	B
141	MEM_DAT[45]	B
142	MEM_DAT[44]	B
143	PWR5	P
144	CPU_DAT[23]	B
145	CPU_DAT[22]	B
146	IOD[11]	B
147	GND	P
148	MEM_DAT[51]	B
149	MEM_DAT[50]	B
150	MEM_DAT[49]	B
151	MEM_DAT[48]	B
152	GND	P
153	CPU_DAT[25]	B
154	CPU_DAT[24]	B
155	IOD[12]	B
156	PWR5	P
157	PWR5	P
158	IOD[13]	B
159	CPU_DAT[26]	B
160	CPU_DAT[27]	B
161	GND	P
162	MEM_DAT[52]	B
163	MEM_DAT[53]	B
164	MEM_DAT[54]	B
165	MEM_DAT[55]	B
166	PWR5	P
167	IOD[14]	B
168	CPU_DAT[28]	B
169	CPU_DAT[29]	B
170	GND	P
171	MEM_DAT[56]	B
172	MEM_DAT[57]	B
173	MEM_DAT[58]	B
174	MEM_DAT[59]	B
175	PWR5	P
176	IOD[15]	B
177	CPU_DAT[30]	B
178	CPU_DAT[31]	B
179	GND	P
180	MEM_DAT[60]	B
181	MEM_DAT[61]	B
182	MEM_DAT[62]	B
183	MEM_DAT[63]	B
184	PWR5	P
185	IOD[16]	B
186	CPU_DAT[32]	B
187	CPU_DAT[33]	B
188	GND	P
189	MEM_DAT[64]	B
190	MEM_DAT[65]	B
191	MEM_DAT[66]	B
192	MEM_DAT[67]	B
193	PWR5	P
194	IOD[17]	B
195	CPU_DAT[34]	B
196	CPU_DAT[35]	B

PIN	SIGNAL	USE
197	GND	P
198	MEM_DAT[68]	B
199	MEM_DAT[69]	B
200	MEM_DAT[70]	B
201	MEM_DAT[71]	B
202	PWR5	P
203	SCAN_IN	I
204	SPARE1_OR_COUNT_OUT	O
205	SPARE2	I
206	SPARE3	I
207	SPARE4	I
208	GND	P

Table 6-11 DSW PIN OUT - Sorted Alphabetically

PIN	SIGNAL	USE	PIN	SIGNAL	USE
103	CMC[0]	I	44	GND	P
102	CMC[1]	I	5	GND	P
101	CMC[2]	I	53	GND	P
100	CMC[3]	I	62	GND	P
99	CMC[4]	I	71	GND	P
116	CMC[5]	I	80	GND	P
117	CMC[6]	I	89	GND	P
118	CMC[7]	I	98	GND	P
119	CMC[8]	I	45	IOC[0]	I
49	CONFIG[0]	I	43	IOC[1]	I
50	CONFIG[1]	I	42	IOC[2]	I
3	CPU_DAT[0]	B	41	IOC[3]	I
64	CPU_DAT[10]	B	40	IOC[4]	I
65	CPU_DAT[11]	B	39	IOC[5]	I
73	CPU_DAT[12]	B	38	IOC[6]	I
74	CPU_DAT[13]	B	2	IOD[0]	B
82	CPU_DAT[14]	B	137	IOD[10]	B
83	CPU_DAT[15]	B	146	IOD[11]	B
91	CPU_DAT[16]	B	155	IOD[12]	B
92	CPU_DAT[17]	B	158	IOD[13]	B
127	CPU_DAT[18]	B	167	IOD[14]	B
126	CPU_DAT[19]	B	176	IOD[15]	B
4	CPU_DAT[1]	B	185	IOD[16]	B
136	CPU_DAT[20]	B	194	IOD[17]	B
135	CPU_DAT[21]	B	11	IOD[1]	B
145	CPU_DAT[22]	B	20	IOD[2]	B
144	CPU_DAT[23]	B	29	IOD[3]	B
154	CPU_DAT[24]	B	54	IOD[4]	B
153	CPU_DAT[25]	B	63	IOD[5]	B
159	CPU_DAT[26]	B	72	IOD[6]	B
160	CPU_DAT[27]	B	81	IOD[7]	B
168	CPU_DAT[28]	B	90	IOD[8]	B
169	CPU_DAT[29]	B	128	IOD[9]	B
12	CPU_DAT[2]	B	6	MEM_DAT[0]	B
177	CPU_DAT[30]	B	26	MEM_DAT[10]	B
178	CPU_DAT[31]	B	27	MEM_DAT[11]	B
186	CPU_DAT[32]	B	33	MEM_DAT[12]	B
187	CPU_DAT[33]	B	34	MEM_DAT[13]	B
195	CPU_DAT[34]	B	35	MEM_DAT[14]	B
196	CPU_DAT[35]	B	36	MEM_DAT[15]	B
13	CPU_DAT[3]	B	58	MEM_DAT[16]	B
21	CPU_DAT[4]	B	59	MEM_DAT[17]	B
22	CPU_DAT[5]	B	60	MEM_DAT[18]	B
30	CPU_DAT[6]	B	61	MEM_DAT[19]	B
31	CPU_DAT[7]	B	7	MEM_DAT[1]	B
55	CPU_DAT[8]	B	67	MEM_DAT[20]	B
56	CPU_DAT[9]	B	68	MEM_DAT[21]	B
106	GND	P	69	MEM_DAT[22]	B
111	GND	P	70	MEM_DAT[23]	B
120	GND	P	76	MEM_DAT[24]	B
129	GND	P	77	MEM_DAT[25]	B
138	GND	P	78	MEM_DAT[26]	B
14	GND	P	79	MEM_DAT[27]	B
147	GND	P	85	MEM_DAT[28]	B
152	GND	P	86	MEM_DAT[29]	B
161	GND	P	8	MEM_DAT[2]	B
170	GND	P	87	MEM_DAT[30]	B
179	GND	P	88	MEM_DAT[31]	B
188	GND	P	94	MEM_DAT[32]	B
197	GND	P	95	MEM_DAT[33]	B
208	GND	P	96	MEM_DAT[34]	B
23	GND	P	97	MEM_DAT[35]	B
32	GND	P	124	MEM_DAT[36]	B

PIN	SIGNAL	USE	PIN	SIGNAL	USE
123	MEM_DAT[37]	B	113	PLL_LP1	I
122	MEM_DAT[38]	B	107	PLL_LP2	I
121	MEM_DAT[39]	B	112	PLL_VDD	I
9	MEM_DAT[3]	B	110	PLL_VSS	I
133	MEM_DAT[40]	B	1	PWR5	P
132	MEM_DAT[41]	B	10	PWR5	P
131	MEM_DAT[42]	B	104	PWR5	P
130	MEM_DAT[43]	B	108	PWR5	P
142	MEM_DAT[44]	B	114	PWR5	P
141	MEM_DAT[45]	B	125	PWR5	P
140	MEM_DAT[46]	B	134	PWR5	P
139	MEM_DAT[47]	B	143	PWR5	P
151	MEM_DAT[48]	B	156	PWR5	P
150	MEM_DAT[49]	B	157	PWR5	P
15	MEM_DAT[4]	B	166	PWR5	P
149	MEM_DAT[50]	B	175	PWR5	P
148	MEM_DAT[51]	B	184	PWR5	P
162	MEM_DAT[52]	B	19	PWR5	P
163	MEM_DAT[53]	B	193	PWR5	P
164	MEM_DAT[54]	B	202	PWR5	P
165	MEM_DAT[55]	B	28	PWR5	P
171	MEM_DAT[56]	B	37	PWR5	P
172	MEM_DAT[57]	B	52	PWR5	P
173	MEM_DAT[58]	B	57	PWR5	P
174	MEM_DAT[59]	B	66	PWR5	P
16	MEM_DAT[5]	B	75	PWR5	P
180	MEM_DAT[60]	B	84	PWR5	P
181	MEM_DAT[61]	B	93	PWR5	P
182	MEM_DAT[62]	B	115	RESET_L	I
183	MEM_DAT[63]	B	203	SCAN_IN	I
189	MEM_DAT[64]	B	204	SPARE1_OR_COUNT_OUT	O
190	MEM_DAT[65]	B	205	SPARE2	I
191	MEM_DAT[66]	B	206	SPARE3	I
192	MEM_DAT[67]	B	207	SPARE4	I
198	MEM_DAT[68]	B	47	TEST_MODE[0]	I
199	MEM_DAT[69]	B	48	TEST_MODE[1]	I
17	MEM_DAT[6]	B	51	TEST_OR_SCAN_OUT	O
200	MEM_DAT[70]	B			
201	MEM_DAT[71]	B			
18	MEM_DAT[7]	B			
24	MEM_DAT[8]	B			
25	MEM_DAT[9]	B			
46	MEM_EN	I			
109	PLL_AGND	I			
105	PLL_CLK	I			

GRU ASIC

The GRU ASIC is a single support chip that implements miscellaneous system functions that normally require additional system board parts. These functions include Flash ROM control, gathering of presence detect bits (PD bits) from memory, gathering/masking of interrupt bits from the CIA and I/O subsystem, assertion of the system clock divisor to the EV5, and creation of the system reset pulse from the DC_OK signal.

Table 6-12 GRU Features

Feature	GRU ASIC
Vendor	LSI Logic
Technology	LSI 100K CMOS
Package type	144 Pin PQFP
Number of I/O pins	88
Gate Count	6K gates
Supply Voltage	5V
Clocking Scheme	Single ended input; internal PLL circuit
External Clock cycle time	30ns
Internal Clock cycle time	15/30 ns
Chips per AlphaStation 600 system	1

Figure 6-7 GRU Pinout - Top View

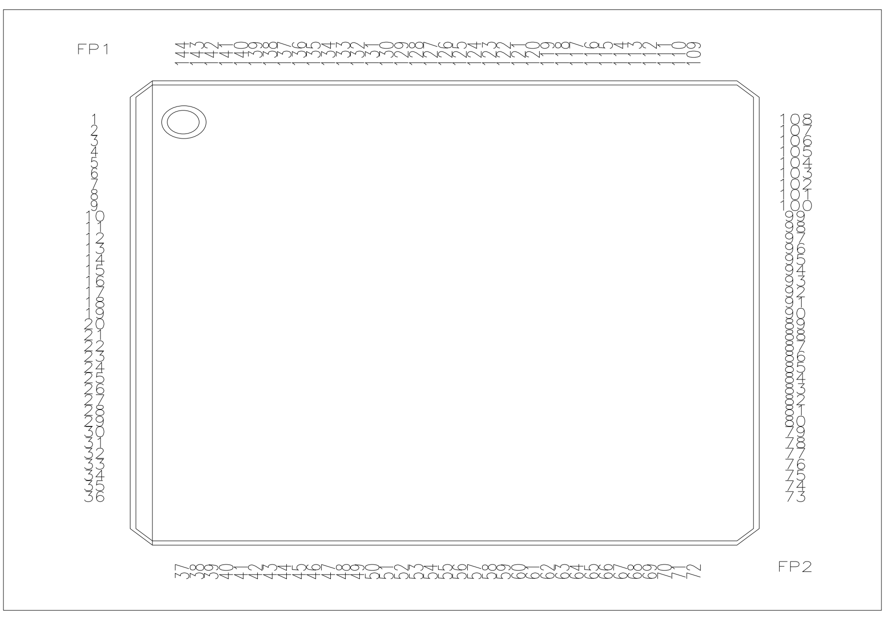


Table 6-13 GRU PIN OUT - Sorted by Pin Number

PIN	SIGNAL	USE
1	PWR5	P
2	FROM_ADDR[0]	O
3	FROM_ADDR[1]	O
4	FROM_ADDR[2]	O
5	FROM_ADDR[3]	O
6	FROM_ADDR[4]	O
7	FROM_ADDR[5]	O
8	GND	P
9	PWR5	P
10	FROM_ADDR[6]	O
11	FROM_ADDR[7]	O
12	FROM_ADDR[8]	O
13	FROM_ADDR[9]	O
14	FROM_ADDR[10]	O
15	FROM_ADDR[11]	O
16	PWR5	P
17	GND	P
18	GND	P
19	GND	P
20	FROM_ADDR[12]	O
21	FROM_ADDR[13]	O
22	FROM_ADDR[14]	O
23	FROM_ADDR[15]	O
24	FROM_ADDR[16]	O
25	FROM_ADDR[17]	O
26	PWR5	P
27	GND	P
28	SYS_RST_L	O
29	TEST_OUT_OR_COUNT_OUT	O
30	CIA_INT	I
31	CIA_ERROR	I
32	SCAN_IN	I
33	GRU_ACK	O
34	GRU_SEL	I
35	IOD[0]	B
36	PWR5	P
37	PWR5	P
38	GND	P
39	IOD[1]	B
40	IOD[2]	B
41	IOD[3]	B
42	IOD[4]	B
43	IOD[5]	B
44	IOD[6]	B
45	IOD[7]	B
46	INT[0]	I
47	INT[1]	I
48	INT[2]	I
49	INT[3]	I
50	INT[4]	I
51	INT[5]	I
52	INT[6]	I
53	GND	P
54	PLL_CLK	I
55	GND	P
56	PWR5	P
57	INT[7]	I
58	INT[8]	I
59	INT[9]	I
60	INT[10]	I
61	INT[11]	I
62	INT[12]	I
63	INT[13]	I
64	GND	P
65	INT[14]	I
66	INT[15]	I
67	INT[16]	I
68	INT[17]	I
69	INT[18]	I
70	INT[19]	I
71	INT[20]	I
72	PWR5	P
73	PWR5	P
74	PLL_LP2	I
75	PLL_AGND	I
76	PLL_VSS	I
77	PLL_VDD	I
78	PLL_LP1	I
79	INT[21]	I
80	INT[22]	I
81	INT[23]	I
82	INT[24]	I
83	INT[25]	I
84	INT[26]	I
85	INT[27]	I
86	INT[28]	I
87	INT[29]	I
88	INT[30]	I
89	INT[31]	I
90	GND	P
91	GND	P
92	GND	P
93	PWR5	P
94	TOY	I
95	IRQ[0]	O
96	IRQ[1]	O
97	IRQ[2]	O
98	IRQ[3]	O
99	NMI	I
100	GND	P
101	PWR5	P
102	GRU_DATA[0]	B
103	GRU_DATA[1]	B
104	GRU_DATA[2]	B
105	GRU_DATA[3]	B
106	GRU_DATA[4]	B
107	GRU_DATA[5]	B
108	PWR5	P
109	PWR5	P
110	GRU_DATA[6]	B
111	GRU_DATA[7]	B
112	PD_CACHE[0]	I
113	PD_CACHE[1]	I
114	PD_CACHE[2]	I
115	PD_CACHE[3]	I
116	PD_CACHE[4]	I
117	PWR5	P
118	GND	P
119	PD_MEM[0]	I
120	PD_MEM[1]	I
121	DC_OK_L	I
122	TEST_MODE[0]	I
123	TEST_MODE[1]	I
124	OCP_RESET_L	I
125	PWR5	P
126	GND	P
127	GND	P

PIN	SIGNAL	USE
128	GND	P
129	USER_INT	I
130	CLK_DIV[3]	I
131	CLK_DIV[2]	I
132	CLK_DIV[1]	I
133	CLK_DIV[0]	I
134	FROM_CE3_L	O
135	FROM_CE2_L	o
136	PWR5	P
137	GND	P
138	FROM_CE1_L	O
139	FROM_CE0_L	O
140	FROM_WE_L	O
141	FROM_OE_L	O
142	SCAN_OUT	O
143	SYS_MCHK	O
144	PWR5	P

Table 6-14 GRU PINS-Sorted Alphabetically

PIN	SIGNAL	USE
31	CIA_ERROR	I
30	CIA_INT	I
133	CLK_DIV[0]	I
132	CLK_DIV[1]	I
131	CLK_DIV[2]	I
130	CLK_DIV[3]	I
121	DC_OK_L	I
2	FROM_ADDR[0]	O
14	FROM_ADDR[10]	O
15	FROM_ADDR[11]	O
20	FROM_ADDR[12]	O
21	FROM_ADDR[13]	O
22	FROM_ADDR[14]	O
23	FROM_ADDR[15]	O
24	FROM_ADDR[16]	O
25	FROM_ADDR[17]	O
3	FROM_ADDR[1]	O
4	FROM_ADDR[2]	O
5	FROM_ADDR[3]	O
6	FROM_ADDR[4]	O
7	FROM_ADDR[5]	O
10	FROM_ADDR[6]	0
11	FROM_ADDR[7]	O
12	FROM_ADDR[8]	O
13	FROM_ADDR[9]	O
139	FROM_CE0_L	O
138	FROM_CE1_L	O
135	FROM_CE2_L	O
134	FROM_CE3_L	O
141	FROM_OE_L	O
140	FROM_WE_L	O
100	GND	P
118	GND	P
126	GND	P
127	GND	P
128	GND	P
137	GND	P
17	GND	P
18	GND	P
19	GND	P
27	GND	P
8	GND	P
53	GND	P

PIN	SIGNAL	USE
55	GND	P
64	GND	P
8	GND	P
90	GND	P
91	GND	P
92	GND	P
33	GRU_ACK	O
102	GRU_DATA[0]	B
103	GRU_DATA[1]	B
104	GRU_DATA[2]	B
105	GRU_DATA[3]	B
106	GRU_DATA[4]	B
107	GRU_DATA[5]	B
110	GRU_DATA[6]	B
111	GRU_DATA[7]	B
34	GRU_SEL	I
46	INT[0]	I
60	INT[10]	I
61	INT[11]	I
62	INT[12]	I
63	INT[13]	I
65	INT[14]	I
66	INT[15]	I
67	INT[16]	I
68	INT[17]	I
69	INT[18]	I
70	INT[19]	I
47	INT[1]	I
71	INT[20]	I
79	INT[21]	I
80	INT[22]	I
81	INT[23]	I
82	INT[24]	I
83	INT[25]	I
84	INT[26]	I
85	INT[27]	I
86	INT[28]	I
87	INT[29]	I
48	INT[2]	I
88	INT[30]	I
89	INT[31]	I
49	INT[3]	I
50	INT[4]	I
51	INT[5]	I
52	INT[6]	I
57	INT[7]	I
58	INT[8]	I
59	INT[9]	I
35	IOD[0]	B
39	IOD[1]	B
40	IOD[2]	B
41	IOD[3]	B
42	IOD[4]	B
43	IOD[5]	B
44	IOD[6]	B
45	IOD[7]	B
95	IRQ[0]	O
96	IRQ[1]	O
97	IRQ[2]	O
98	IRQ[3]	O
99	NMI	I
124	OCP_RESET_L	I
112	PD_CACHE[0]	I
113	PD_CACHE[1]	I
114	PD_CACHE[2]	I
115	PD_CACHE[3]	I

PIN	SIGNAL	USE
116	PD_CACHE[4]	I
119	PD_MEM[0]	I
120	PD_MEM[1]	I
75	PLL_AGND	I
54	PLL_CLK	I
78	PLL_LP1	I
74	PLL_LP2	I
77	PLL_VDD	I
76	PLL_VSS	I
1	PWR5	P
101	PWR5	P
108	PWR5	P
109	PWR5	P
117	PWR5	P
125	PWR5	P
136	PWR5	P
144	PWR5	P
16	PWR5	P
26	PWR5	P
36	PWR5	P
37	PWR5	P
56	PWR5	P
72	PWR5	P
73	PWR5	P
9	PWR5	P
93	PWR5	P
32	SCAN_IN	I
142	SCAN_OUT	O
143	SYS_MCHK	O
28	SYS_RST_L	O
122	TEST_MODE[0]	I
123	TEST_MODE[1]	I
29	TEST_OUT_OR_COUNT_OUT	O
94	TOY	I
129	USER_INT	I

CIA ASIC

The CIA is the control ASIC for the AlphaStation 600 System; it controls PCI, Cache, and Memory transactions. The CIA also controls the data switch (DSW) and support chip (GRU) operations. Table 6-15 summarizes CIA features:

Table 6-15 CIA Features

Feature	CIA ASIC
Vendor	LSI Logic
Technology	LSI 300K CMOS
Package type	383 Pin Ceramic PGA
Number of I/O pins	292
Gate Count	105K gates
Supply Voltage	5V
Clocking Scheme	Single ended input; internal PLL circuit
External Clock cycle time	30ns
Internal Clock cycle time	15/30 ns
Chips per AlphaStation 600 System	1

Figure 6-8 CIA Pinout - Top View

Table 6-16 CIA PIN OUT - Sorted by Pin Number

PIN	SIGNAL	USE	PIN	SIGNAL	USE
A10	GND	P	B1	GND	P
A11	PWR5	P	B10	IOD[0]	B
A12	GND	P	B11	RES[1]	I
A13	PWR5	P	B12	IOC[5]	O
A14	GND	P	B13	IOC[3]	O
A15	PWR5	P	B14	CMC[8]	O
A16	GND	P	B15	CMC[3]	O
A17	GND	P	B16	CMC[1]	O
A18	GND	P	B17	VICTIM_PENDING	I
A19	PWR5	P	B18	CMD[0]	B
A2	GND	P	B19	ADDR[32]	B
A20	GND	P	B2	PWR5	P
A21	GND	P	B20	IDLE_BC	O
A22	PWR5	P	B21	CACK	O
A3	PWR5	P	B22	PWR5	P
A4	GND	P	B3	PLL_LP1	I
A5	GND	P	B4	IOD[22]	B
A6	GND	P	B5	IOD[17]	B
A7	PWR5	P	B6	IOD[13]	B
A8	GND	P	B7	IOD[9]	B
A9	PWR5	P	B8	IOD[6]	B
AA1	PWR5	P	B9	IOD[3]	B
AA10	AD[15]	B	C1	PWR5	P
AA11	AD[14]	B	C10	IOD[1]	B
AA12	AD[13]	B	C11	IOC[6]	O
AA13	AD[12]	B	C12	IOC[4]	O
AA14	AD[10]	B	C13	IOC[1]	O
AA15	AD[39]	B	C14	CMC[5]	O
AA16	AD[38]	B	C15	CMC[2]	O
AA17	AD[35]	B	C16	INT4_VALID[1]	I
AA18	AD[33]	B	C17	CMD[2]	B
AA19	CBE_L[0]	B	C18	ADDR[34]	B
AA2	AD[29]	B	C19	FILL	O
AA20	CBE_L[1]	B	C2	IOD_E[3]	B
AA21	PERR_L	B	C20	SET_SEL[2]	O
AA22	GND	P	C21	SET_SEL[4]	O
AA3	AD[28]	B	C22	GND	P
AA4	AD[26]	B	C3	IOD_E[1]	B
AA5	AD[55]	B	C4	AUX_VDD	P
AA6	AD[53]	B	C5	IOD[20]	B
AA7	AD[51]	B	C6	IOD[15]	B
AA8	AD[18]	B	C7	IOD[11]	B
AA9	AD[48]	B	C8	IOD[7]	B
AB1	PWR5	P	C9	IOD[5]	B
AB10	PWR5	P	D1	GND	P
AB11	PWR5	P	D10	RES[0]	I
AB12	GND	P	D11	TEST_MODE[1]	I
AB13	GND	P	D12	IOC[2]	O
AB14	PWR5	P	D13	CMC[7]	O
AB15	GND	P	D14	CMC[0]	O
AB16	PWR5	P	D15	INT4_VALID[3]	I
AB17	GND	P	D16	CMD[3]	B
AB18	GND	P	D17	ADDR39	B
AB19	GND	P	D18	ADDR_BUS_REQ	B
AB2	GND	P	D19	GND	P
AB20	PWR5	P	D2	IOD[35]	B
AB21	GND	P	D20	SET_SEL[5]	O
AB22	PWR5	P	D21	SET_SEL[8]	O
AB3	GND	P	D22	PWR5	P
AB4	PWR5	P	D3	IOD[32]	B
AB5	GND	P	D4	PLL_VSS	I
AB6	PWR5	P	D5	AUX_VSS	P
AB7	GND	P	D6	IOD[19]	B
AB8	PWR5	P	D7	IOD[14]	B
AB9	GND	P	D8	IOD[12]	B

PIN	SIGNAL	USE	PIN	SIGNAL	USE
D9	IOD[8]	B	H4	IOD[41]	B
E1	PWR5	P	H5	IOD[39]	B
E10	IOD[2]	B	H6	IOD[36]	B
E11	TEST_MODE[0]	I	J1	GND	P
E12	IOC[0]	O	J17	SET_SEL[14]	O
E13	CMC[4]	O	J18	RAS[0]	O
E14	INT4_VALID[0]	I	J19	RAS[2]	O
E15	ADDR_CMD_PAR	B	J2	IOD[54]	B
E16	ADDR[31]	B	J20	MEM_ADDR[0]	O
E17	TEST_OR_SCAN_OUT	O	J21	MEM_ADDR[2]	O
E18	SPARE5	I	J22	GND	P
E19	SET_SEL[6]	O	J3	IOD[52]	B
E2	IOD[40]	B	J4	IOD[49]	B
E20	SET_SEL[10]	O	J5	IOD[47]	B
E21	SET_SEL[13]	O	J6	IOD[45]	B
E22	GND	P	K1	GND	P
E3	IOD[37]	B	K17	MEM_ADDR[1]	O
E4	IOD[33]	B	K18	MEM_ADDR[3]	O
E5	PLL_VDD	p	K19	MEM_ADDR[6]	O
E6	PLL_AGND	O	K2	IOD[57]	B
E7	IOD[21]	B	K20	MEM_ADDR[4]	O
E8	IOD[18]	B	K21	MEM_ADDR[5]	O
E9	IOD[10]	B	K22	PWR5	P
F1	GND	P	K3	IOD[56]	B
F10	IOD[4]	B	K4	IOD[58]	B
F11	PLL_CLK	I	K5	IOD[55]	B
F12	CMC[6]	O	K6	IOD[53]	B
F13	INT4_VALID[2]	I	L1	PWR5	P
F14	CMD[1]	B	L17	INT	O
F15	ADDR[33]	B	L18	SYS_RST_L	I
F16	DACK	O	L19	SCAN_IN	I
F17	SET_SEL[0]	O	L2	IOD[59]	B
F18	SET_SEL[3]	O	L20	MEM_ADDR[8]	O
F19	SET_SEL[11]	O	L21	MEM_ADDR[7]	O
F2	IOD[44]	B	L22	PWR5	P
F20	SET_SEL[15]	O	L3	IOD[60]	B
F21	CAS[1]	O	L4	SPARE4	I
F22	GND	P	L5	GRU_SEL	O
F3	IOD[42]	B	L6	GRU_ACK	I
F4	IOD[38]	B	M1	GND	P
F5	IOD_E[0]	B	M17	FILL_ERROR	O
F6	IOD[31]	B	M18	MEM_WE_L[1]	O
F7	PLL_LP2	P	M19	MEM_ADDR[12]	O
F8	IOD[23]	B	M2	IOD[61]	B
F9	IOD[16]	B	M20	MEM_ADDR[10]	O
G1	GND	P	M21	MEM_ADDR[9]	O
G17	SET_SEL[1]	O	M22	GND	P
G18	SET_SEL[9]	O	M3	IOD[62]	B
G19	CAS[0]	O	M4	IOD[30]	B
G2	IOD[48]	B	M5	IOD_E[4]	B
G20	CAS[3]	O	M6	IOD_E[6]	B
G21	RAS[1]	O	N1	PWR5	P
G22	GND	P	N17	ADDR[8]	B
G3	IOD[46]	B	N18	ADDR[4]	B
G4	IOD[43]	B	N19	TAG_DIRTY	O
G5	IOD[34]	B	N2	IOD[63]	B
G6	IOD_E[2]	B	N20	MEM_WE_L[0]	O
H1	PWR5	P	N21	MEM_ADDR[11]	O
H17	SET_SEL[7]	O	N22	GND	P
H18	SET_SEL[12]	O	N3	IOD_E[5]	B
H19	CAS[2]	O	N4	IOD[24]	B
H2	IOD[51]	B	N5	IOD[29]	B
H20	RAS[3]	O	N6	IOD[27]	B
H21	MEM_EN	O	P1	GND	P
H22	PWR5	P	P17	ADDR[16]	B
H3	IOD[50]	B	P18	ADDR[12]	B

PIN	SIGNAL	USE
P19	ADDR[10]	B
P2	IOD_E[7]	B
P20	FILL_ID	B
P21	TAG_CTL_PAR	O
P22	PWR5	P
P3	IOD[25]	B
P4	IOD[26]	B
P5	ERROR	O
P6	REQ_L	O
R1	PWR5	P
R17	ADDR[21]	B
R18	ADDR[18]	B
R19	ADDR[9]	B
R2	SPARE3	I
R20	ADDR[6]	B
R21	ADDR[5]	B
R22	GND	P
R3	SPARE2	I
R4	MEM_ACK_L	O
R5	PAR	B
R6	COUNT_OUT	O
T1	GND	P
T17	ADDR[27]	B
T18	ADDR[23]	B
T19	ADDR[14]	B
T2	IOD[28]	B
T20	ADDR[11]	B
T21	ADDR[7]	B
T22	PWR5	P
T3	SPARE7	I
T4	MEM_REQ_L	I
T5	CBE_L[5]	B
T6	AD[62]	B
U1	GND	P
U10	AD[17]	B
U11	SPARE1	I
U12	AD[42]	B
U13	AD[6]	B
U14	AD[2]	B
U15	LOCK_L	I
U16	RST_L	O
U17	ADDR[30]	B
U18	ADDR[29]	B
U19	ADDR[19	B
U2	MEM_CS_L	I
U20	ADDR[15]	B
U21	ADDR[13]	B
U22	GND	P
U3	GNT_L	I
U4	CBE_L[7]	B
U5	ACK64_L	B
U6	AD[30]	B
U7	AD[60]	B
U8	AD[57]	B
U9	AD[21]	B
V1	GND	P
V10	AD[16]	B
V11	IRDY_L	B
V12	AD[43]	B
V13	AD[8]	B
V14	AD[4]	B
V15	AD[1]	B
V16	ADDR[26]	B
V17	DEVSEL_L	B
V18	FRAME_L	B
V19	ADDR[24]	B
V2	PAR64	B
V20	ADDR[20]	B
V21	ADDR[17]	B
V22	PWR5	P
V3	CBE_L[6]	B
V4	REQ64_L	B
V5	SPARE6	I
V6	AD[61]	B
V7	AD[58]	B
V8	AD[24]	B
V9	AD[20]	B
W1	PWR5	P
W10	AD[46]	B
W11	TRDY_L	B
W12	AD[9]	B
W13	AD[41]	B
W14	AD[5]	B
W15	AD[37]	B
W16	AD[3]	B
W17	AD[32]	B
W18	CBE_L[3]	B
W19	GND	P
W2	CBE_L[4]	B
W20	ADDR[25]	B
W21	ADDR[22]	B
W22	GND	P
W3	SERR_L	B
W4	GND	P
W5	AD[27]	B
W6	AD[56]	B
W7	AD[22]	B
W8	AD[23]	B
W9	AD[19]	B
Y1	GND	P
Y10	AD[47]	B
Y11	AD[45]	B
Y12	AD[44]	B
Y13	AD[11]	B
Y14	AD[40]	B
Y15	AD[7]	B
Y16	AD[36]	B
Y17	AD[34]	B
Y18	AD[0]	B
Y19	STOP_L	B
Y2	AD[63]	B
Y20	ADDR[28]	B
Y21	CBE_L[2]	B
Y22	PWR5	P
Y3	AD[31]	B
Y4	AD[59]	B
Y5	AD[25]	B
Y6	AD[54]	B
Y7	AD[52]	B
Y8	AD[50]	B
Y9	AD[49]	B

Table 6-17 CIA PIN OUT - Sorted Alphabetically

PIN	SIGNAL	USE
U5	ACK64_L	B
D17	ADDR39	B
P19	ADDR[10]	B
T20	ADDR[11]	B
P18	ADDR[12]	B
U21	ADDR[13]	B
T19	ADDR[14]	B
U20	ADDR[15]	B
P17	ADDR[16]	B
V21	ADDR[17]	B
R18	ADDR[18]	B
U19	ADDR[19	B
V20	ADDR[20]	B
R17	ADDR[21]	B
W21	ADDR[22]	B
T18	ADDR[23]	B
V19	ADDR[24]	B
W20	ADDR[25]	B
V16	ADDR[26]	B
T17	ADDR[27]	B
Y20	ADDR[28]	B
U18	ADDR[29]	B
U17	ADDR[30]	B
E16	ADDR[31]	B
B19	ADDR[32]	B
F15	ADDR[33]	B
C18	ADDR[34]	B
N18	ADDR[4]	B
R21	ADDR[5]	B
R20	ADDR[6]	B
T21	ADDR[7]	B
N17	ADDR[8]	B
R19	ADDR[9]	B
D18	ADDR_BUS_REQ	B
E15	ADDR_CMD_PAR	B
Y18	AD[0]	B
AA14	AD[10]	B
Y13	AD[11]	B
AA13	AD[12]	B
AA12	AD[13]	B
AA11	AD[14]	B
AA10	AD[15]	B
V10	AD[16]	B
U10	AD[17]	B
AA8	AD[18]	B
W9	AD[19]	B
V15	AD[1]	B
V9	AD[20]	B
U9	AD[21]	B
W7	AD[22]	B
W8	AD[23]	B
V8	AD[24]	B
Y5	AD[25]	B
AA4	AD[26]	B
W5	AD[27]	B
AA3	AD[28]	B
AA2	AD[29]	B
U14	AD[2]	B
U6	AD[30]	B
Y3	AD[31]	B
W17	AD[32]	B
AA18	AD[33]	B
Y17	AD[34]	B

PIN	SIGNAL	USE
AA17	AD[35]	B
Y16	AD[36]	B
W15	AD[37]	B
AA16	AD[38]	B
AA15	AD[39]	B
W16	AD[3]	B
Y14	AD[40]	B
W13	AD[41]	B
U12	AD[42]	B
V12	AD[43]	B
Y12	AD[44]	B
Y11	AD[45]	B
W10	AD[46]	B
Y10	AD[47]	B
AA9	AD[48]	B
Y9	AD[49]	B
V14	AD[4]	B
Y8	AD[50]	B
AA7	AD[51]	B
Y7	AD[52]	B
AA6	AD[53]	B
Y6	AD[54]	B
AA5	AD[55]	B
W6	AD[56]	B
U8	AD[57]	B
V7	AD[58]	B
Y4	AD[59]	B
W14	AD[5]	B
U7	AD[60]	B
V6	AD[61]	B
T6	AD[62]	B
Y2	AD[63]	B
U13	AD[6]	B
Y15	AD[7]	B
V13	AD[8]	B
W12	AD[9]	B
C4	AUX_VDD	P
D5	AUX_VSS	P
B21	CACK	O
G19	CAS[0]	O
F21	CAS[1]	O
H19	CAS[2]	O
G20	CAS[3]	O
AA19	CBE_L[0]	B
AA20	CBE_L[1]	B
Y21	CBE_L[2]	B
W18	CBE_L[3]	B
W2	CBE_L[4]	B
T5	CBE_L[5]	B
V3	CBE_L[6]	B
U4	CBE_L[7]	B
D14	CMC[0]	O
B16	CMC[1]	O
C15	CMC[2]	O
B15	CMC[3]	O
E13	CMC[4]	O
C14	CMC[5]	O
F12	CMC[6]	O
D13	CMC[7]	O
B14	CMC[8]	O
B18	CMD[0]	B
F14	CMD[1]	B
C17	CMD[2]	B

PIN	SIGNAL	USE	PIN	SIGNAL	USE
D16	CMD[3]	B	F13	INT4_VALID[2]	I
R6	COUNT_OUT	O	D15	INT4_VALID[3]	I
F16	DACK	O	E12	IOC[0]	O
V17	DEVSEL_L	B	C13	IOC[1]	O
P5	ERROR	O	D12	IOC[2]	O
C19	FILL	O	B13	IOC[3]	O
M17	FILL_ERROR	O	C12	IOC[4]	O
P20	FILL_ID	B	B12	IOC[5]	O
V18	FRAME_L	B	C11	IOC[6]	O
A10	GND	P	B10	IOD[0]	B
A12	GND	P	E9	IOD[10]	B
A14	GND	P	C7	IOD[11]	B
A16	GND	P	D8	IOD[12]	B
A17	GND	P	B6	IOD[13]	B
A18	GND	P	D7	IOD[14]	B
A2	GND	P	C6	IOD[15]	B
A20	GND	P	F9	IOD[16]	B
A21	GND	P	B5	IOD[17]	B
A4	GND	P	E8	IOD[18]	B
A5	GND	P	D6	IOD[19]	B
A6	GND	P	C10	IOD[1]	B
A8	GND	P	C5	IOD[20]	B
AA22	GND	P	E7	IOD[21]	B
AB12	GND	P	B4	IOD[22]	B
AB13	GND	P	F8	IOD[23]	B
AB15	GND	P	N4	IOD[24]	B
AB17	GND	P	P3	IOD[25]	B
AB18	GND	P	P4	IOD[26]	B
AB19	GND	P	N6	IOD[27]	B
AB2	GND	P	T2	IOD[28]	B
AB21	GND	P	N5	IOD[29]	B
AB3	GND	P	E10	IOD[2]	B
AB5	GND	P	M4	IOD[30]	B
AB7	GND	P	F6	IOD[31]	B
AB9	GND	P	D3	IOD[32]	B
B1	GND	P	E4	IOD[33]	B
C22	GND	P	G5	IOD[34]	B
D1	GND	P	D2	IOD[35]	B
D19	GND	P	H6	IOD[36]	B
E22	GND	P	E3	IOD[37]	B
F1	GND	P	F4	IOD[38]	B
F22	GND	P	H5	IOD[39]	B
G1	GND	P	B9	IOD[3]	B
G22	GND	P	E2	IOD[40]	B
J1	GND	P	H4	IOD[41]	B
J22	GND	P	F3	IOD[42]	B
K1	GND	P	G4	IOD[43]	B
M1	GND	P	F2	IOD[44]	B
M22	GND	P	J6	IOD[45]	B
N22	GND	P	G3	IOD[46]	B
P1	GND	P	J5	IOD[47]	B
R22	GND	P	G2	IOD[48]	B
T1	GND	P	J4	IOD[49]	B
U1	GND	P	F10	IOD[4]	B
U22	GND	P	H3	IOD[50]	B
V1	GND	P	H2	IOD[51]	B
W19	GND	P	J3	IOD[52]	B
W22	GND	P	K6	IOD[53]	B
W4	GND	P	J2	IOD[54]	B
Y1	GND	P	K5	IOD[55]	B
U3	GNT_L	I	K3	IOD[56]	B
L6	GRU_ACK	I	K2	IOD[57]	B
L5	GRU_SEL	O	K4	IOD[58]	B
B20	IDLE_BC	O	L2	IOD[59]	B
L17	INT	O	C9	IOD[5]	B
E14	INT4_VALID[0]	I	L3	IOD[60]	B
C16	INT4_VALID[1]	I	M2	IOD[61]	B

PIN	SIGNAL	USE
M3	IOD[62]	B
N2	IOD[63]	B
B8	IOD[6]	B
C8	IOD[7]	B
D9	IOD[8]	B
B7	IOD[9]	B
F5	IOD_E[0]	B
C3	IOD_E[1]	B
G6	IOD_E[2]	B
C2	IOD_E[3]	B
M5	IOD_E[4]	B
N3	IOD_E[5]	B
M6	IOD_E[6]	B
P2	IOD_E[7]	B
V11	IRDY_L	B
U15	LOCK_L	I
R4	MEM_ACK_L	O
J20	MEM_ADDR[0]	O
M20	MEM_ADDR[10]	O
N21	MEM_ADDR[11]	O
M19	MEM_ADDR[12]	O
K17	MEM_ADDR[1]	O
J21	MEM_ADDR[2]	O
K18	MEM_ADDR[3]	O
K20	MEM_ADDR[4]	O
K21	MEM_ADDR[5]	O
K19	MEM_ADDR[6]	O
L21	MEM_ADDR[7]	O
L20	MEM_ADDR[8]	O
M21	MEM_ADDR[9]	O
U2	MEM_CS_L	I
H21	MEM_EN	O
T4	MEM_REQ_L	I
N20	MEM_WE_L[0]	O
M18	MEM_WE_L[1]	O
R5	PAR	B
V2	PAR64	B
AA21	PERR_L	B
E6	PLL_AGND	O
F11	PLL_CLK	I
B3	PLL_LP1	I
F7	PLL_LP2	P
E5	PLL_VDD	P
D4	PLL_VSS	I
A11	PWR5	P
A13	PWR5	P
A15	PWR5	P
A19	PWR5	P
A22	PWR5	P
A3	PWR5	P
A7	PWR5	P
A9	PWR5	P
AA1	PWR5	P
AB1	PWR5	P
AB10	PWR5	P
AB11	PWR5	P
AB14	PWR5	P
AB16	PWR5	P
AB20	PWR5	P
AB22	PWR5	P
AB4	PWR5	P
AB6	PWR5	P
AB8	PWR5	P
B2	PWR5	P
B22	PWR5	P
C1	PWR5	P
D22	PWR5	P
E1	PWR5	P

PIN	SIGNAL	USE
H1	PWR5	P
H22	PWR5	P
K22	PWR5	P
L1	PWR5	P
L22	PWR5	P
N1	PWR5	P
P22	PWR5	P
R1	PWR5	P
T22	PWR5	P
V22	PWR5	P
W1	PWR5	P
Y22	PWR5	P
J18	RAS[0]	O
G21	RAS[1]	O
J19	RAS[2]	O
H20	RAS[3]	O
V4	REQ64_L	B
P6	REQ_L	O
D10	RES[0]	I
B11	RES[1]	I
U16	RST_L	O
L19	SCAN_IN	I
W3	SERR_L	B
F17	SET_SEL[0]	O
E20	SET_SEL[10]	O
F19	SET_SEL[11]	O
H18	SET_SEL[12]	O
E21	SET_SEL[13]	O
J17	SET_SEL[14]	O
F20	SET_SEL[15]	O
G17	SET_SEL[1]	O
C20	SET_SEL[2]	O
F18	SET_SEL[3]	O
C21	SET_SEL[4]	O
D20	SET_SEL[5]	O
E19	SET_SEL[6]	O
H17	SET_SEL[7]	O
D21	SET_SEL[8]	O
G18	SET_SEL[9]	O
U11	SPARE1	I
R3	SPARE2	I
R2	SPARE3	I
L4	SPARE4	I
E18	SPARE5	I
V5	SPARE6	I
T3	SPARE7	I
Y19	STOP_L	B
L18	SYS_RST_L	I
P21	TAG_CTL_PAR	O
N19	TAG_DIRTY	O
E11	TEST_MODE[0]	I
D11	TEST_MODE[1]	I
E17	TEST_OR_SCAN_OUT	O
W11	TRDY_L	B
B17	VICTIM_PENDING	I

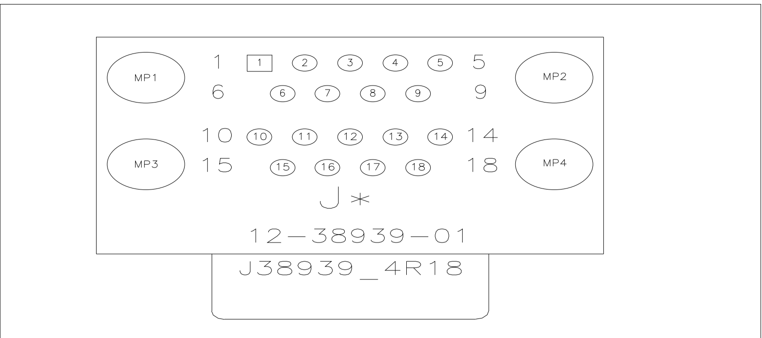
AlphaStation 600 I/O & External Interconnect

The AlphaStation 600 system has a number of connectors that provide I/O, control, power, and diagnostic interfaces to the system. These interconnects, outlined briefly in the physical description, are described in more detail in this section. Connector pinout drawings and tables are listed for most connectors, especially nonstandard ones. All connector pinout drawings are shown from a side 1 (top) view.

Serial Ports

The AlphaStation 600 system has two serial port connectors, which are arranged in a stacked configuration in the I/O area at the upper left of the system board. Figure 6-9 shows the pinning of the serial port connector as viewed from the top of the connector.

Figure 6-9 Serial Port Connector

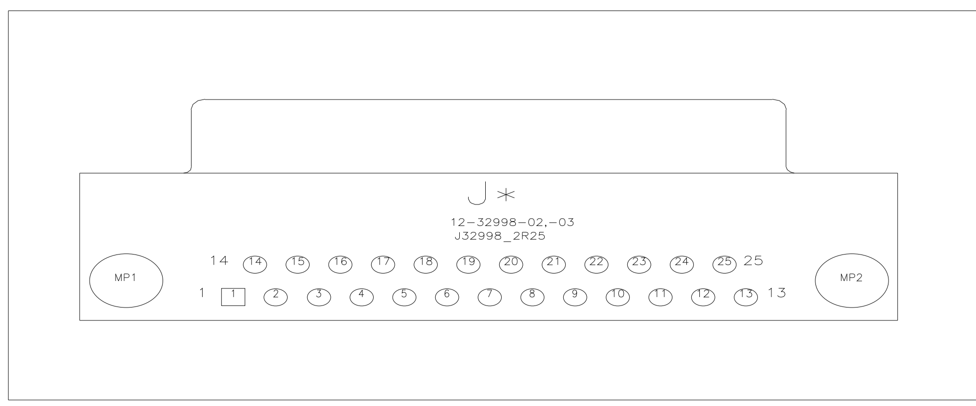


Parallel Port

The AlphaStation 600 SystemBoard contains one 25-pin parallel port, intended for use as a printer port or other compatible Centronix device. The parallel Port is located at the upper left hand area of the SystemBoard, and is identified as J5 on the UA drawing.

Both the serial and Parallel ports have standard I/O pin assignments. See the RS232 or Centronics specs for pinout information, or the AlphaStation 600 System Board schematics.

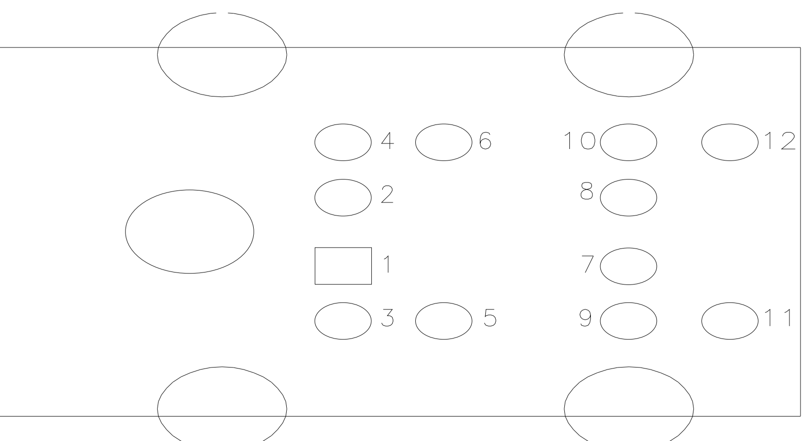
Figure 6-10 Parallel Port Connector



Keyboard/Mouse Connectors

The AlphaStation 600 system provides a keyboard/mouse connector in a stacked, mini-DIN configuration. Both the keyboard and mouse connectors accept industry standard PC-compatible keyboards and mice. The Keyboard/Mouse connector is designated as J28, in the upper left hand corner of the system board. The pinout organization of the connector is shown in Figure 6-11

Figure 6-11 Keyboard/Mouse



PCI Connector

Table 6-18 PCI Pin Out - Sorted by pin number

PIN#	PIN NAME	PIN TYPE	PIN#	PIN NAME	PIN TYPE
A1	TRST_L	BI	B11	PRSNT2_L	BI
A15	RST_L	IN	B16	CLK	IN
A17	GNT_L	BI	B18	REQ_L	BI
A20	AD[30]	BI	B2	TCK	BI
A22	AD[28]	BI	B20	AD[31]	BI
A23	AD[26]	BI	B21	AD[29]	BI
A25	AD[24]	BI	B23	AD[27]	BI
A26	IDSEL	IN	B24	AD[25]	BI
A28	AD[22]	BI	B26	CBE_L[3]	BI
A29	AD[20]	BI	B27	AD[23]	BI
A3	TMS	BI	B29	AD[21]	BI
A31	AD[18]	BI	B30	AD[19]	BI
A32	AD[16]	BI	B32	AD[17]	BI
A34	FRAME_L	BI	B33	CBE_L[2]	BI
A36	TRDY_L	BI	B35	IRDY_L	BI
A38	STOP_L	BI	B37	DEVSEL_L	BI
A4	TDI	BI	B39	LOCK_L	BI
A40	SDONE	BI	B4	TDO	BI
A41	SBO_L	BI	B40	PERR_L	BI
A43	PAR	BI	B42	SERR_L	BI
A44	AD[15]	BI	B44	CBE_L[1]	BI
A46	AD[13]	BI	B45	AD[14]	BI
A47	AD[11]	BI	B47	AD[12]	BI
A49	AD[9]	BI	B48	AD[10]	BI
A52	CBE_L[0]	BI	B52	AD[8]	BI
A54	AD[6]	BI	B53	AD[7]	BI
A55	AD[4]	BI	B55	AD[5]	BI
A57	AD[2]	BI	B56	AD[3]	BI
A58	AD[0]	BI	B58	AD[1]	BI
A6	INTA_L	BI	B60	ACK64_L	BI
A60	REQ64_L	BI	B65	CBE_L[6]	BI
A64	CBE_L[7]	BI	B66	CBE_L[4]	BI
A65	CBE_L[5]	BI	B68	AD[63]	BI
A67	PAR64	BI	B69	AD[61]	BI
A68	AD[62]	BI	B7	INTB_L	BI
A7	INTC_L	BI	B71	AD[59]	BI
A70	AD[60]	BI	B72	AD[57]	BI
A71	AD[58]	BI	B74	AD[55]	BI
A73	AD[56]	BI	B75	AD[53]	BI
A74	AD[54]	BI	B77	AD[51]	BI
A76	AD[52]	BI	B78	AD[49]	BI
A77	AD[50]	BI	B8	INTD_L	BI
A79	AD[48]	BI	B80	AD[47]	BI
A80	AD[46]	BI	B81	AD[45]	BI
A82	AD[44]	BI	B83	AD[43]	BI
A83	AD[42]	BI	B84	AD[41]	BI
A85	AD[40]	BI	B86	AD[39]	BI
A86	AD[38]	BI	B87	AD[37]	BI
A88	AD[36]	BI	B89	AD[35]	BI
A89	AD[34]	BI	B9	PRSNT1_L	BI
A91	AD[32]	BI	B90	AD[33]	BI

Table 6-19 PCI Pin Out - Sorted by Signal Name

PIN#	PIN NAME	PIN TYPE
B60	ACK64_L	BI
A58	AD[0]	BI
B48	AD[10]	BI
A47	AD[11]	BI
B47	AD[12]	BI
A46	AD[13]	BI
B45	AD[14]	BI
A44	AD[15]	BI
A32	AD[16]	BI
B32	AD[17]	BI
A31	AD[18]	BI
B30	AD[19]	BI
B58	AD[1]	BI
A29	AD[20]	BI
B29	AD[21]	BI
A28	AD[22]	BI
B27	AD[23]	BI
A25	AD[24]	BI
B24	AD[25]	BI
A23	AD[26]	BI
B23	AD[27]	BI
A22	AD[28]	BI
B21	AD[29]	BI
A57	AD[2]	BI
A20	AD[30]	BI
B20	AD[31]	BI
A91	AD[32]	BI
B90	AD[33]	BI
A89	AD[34]	BI
B89	AD[35]	BI
A88	AD[36]	BI
B87	AD[37]	BI
A86	AD[38]	BI
B86	AD[39]	BI
B56	AD[3]	BI
A85	AD[40]	BI
B84	AD[41]	BI
A83	AD[42]	BI
B83	AD[43]	BI
A82	AD[44]	BI
B81	AD[45]	BI
A80	AD[46]	BI
B80	AD[47]	BI
A79	AD[48]	BI
B78	AD[49]	BI
A55	AD[4]	BI
A77	AD[50]	BI
B77	AD[51]	BI
A76	AD[52]	BI
B75	AD[53]	BI
A74	AD[54]	BI
B74	AD[55]	BI

PIN#	PIN NAME	PIN TYPE
A73	AD[56]	BI
B72	AD[57]	BI
A71	AD[58]	BI
B71	AD[59]	BI
B55	AD[5]	BI
A70	AD[60]	BI
B69	AD[61]	BI
A68	AD[62]	BI
B68	AD[63]	BI
A54	AD[6]	BI
B53	AD[7]	BI
B52	AD[8]	BI
A49	AD[9]	BI
A52	CBE_L[0]	BI
B44	CBE_L[1]	BI
B33	CBE_L[2]	BI
B26	CBE_L[3]	BI
B66	CBE_L[4]	BI
A65	CBE_L[5]	BI
B65	CBE_L[6]	BI
A64	CBE_L[7]	BI
B16	CLK	IN
B37	DEVSEL_L	BI
A34	FRAME_L	BI
A17	GNT_L	BI
A26	IDSEL	IN
A6	INTA_L	BI
B7	INTB_L	BI
A7	INTC_L	BI
B8	INTD_L	BI
B35	IRDY_L	BI
B39	LOCK_L	BI
A43	PAR	BI
A67	PAR64	BI
B40	PERR_L	BI
B9	PRSNT1_L	BI
B11	PRSNT2_L	BI
A60	REQ64_L	BI
B18	REQ_L	BI
A15	RST_L	IN
A41	SBO_L	BI
A40	SDONE	BI
B42	SERR_L	BI
A38	STOP_L	BI
B2	TCK	BI
A4	TDI	BI
B4	TDO	BI
A3	TMS	BI
A36	TRDY_L	BI
A1	TRST_L	BI

EISA Connector

Table 6-20 EISA Pin Out - Sorted by Pin Number

PIN#	PIN NAME	PIN TYPE
A1	IOCHK_L	OUT
A10	CHRDY	BI
A11	AEN	IN
A12	SA[19]	BI
A13	SA[18]	BI
A14	SA[17]	BI
A15	SA[16]	BI
A16	SA[15]	BI
A17	SA[14]	BI
A18	SA[13]	BI
A19	SA[12]	BI
A2	D[7]	BI
A20	SA[11]	BI
A21	SA[10]	BI
A22	SA[9]	BI
A23	SA[8]	BI
A24	SA[7]	BI
A25	SA[6]	BI
A26	SA[5]	BI
A27	SA[4]	BI
A28	SA[3]	BI
A29	SA[2]	BI
A3	D[6]	BI
A30	SA[1]	BI
A31	SA[0]	BI
A4	D[5]	BI
A5	D[4]	BI
A6	D[3]	BI
A7	D[2]	BI
A8	D[1]	BI
A9	D[0]	BI
B11	SMWTC_L	IN
B12	SMRDC_L	IN
B13	IOWC_L	IN
B14	IORC_L	IN
B15	DAK_L[3]	IN
B16	DRQ[3]	OUT
B17	DAK_L[1]	IN
B18	DRQ[1]	OUT
B19	REFRESH_L	BI
B2	RESDRV	IN
B20	BCLK	IN
B21	IRQ[7]	OUT
B22	IRQ[6]	OUT
B23	IRQ[5]	OUT
B24	IRQ[4]	OUT
B25	IRQ[3]	OUT
B26	DAK_L[2]	IN
B27	TC	BI
B28	BALE	IN
B30	OSC	IN
B4	IRQ[9]	OUT
B6	DRQ[2]	OUT
B8	NOWS_L	OUT
C1	SBHE_L	BI
C10	MWTC_L	BI
C11	D[8]	BI
C12	D[9]	BI
C13	D[10]	BI
C14	D[11]	BI
C15	D[12]	BI
C16	D[13]	BI
C17	D[14]	BI
C18	D[15]	BI
C2	LA[23]	BI
C3	LA[22]	BI
C4	LA[21]	BI
C5	LA[20]	BI
C6	LA[19]	BI
C7	LA[18]	BI
C8	LA[17]	BI
C9	MRDC_L	IN
D1	M16_L	IN
D10	DAK_L[5]	IN
D11	DRQ[5]	OUT
D12	DAK_L[6]	IN
D13	DRQ[6]	OUT
D14	DAK_L[7]	IN
D15	DRQ[7]	OUT
D17	MASTER16_L	OUT
D2	IO16_L	OUT
D3	IRQ[10]	OUT
D4	IRQ[11]	OUT
D5	IRQ[12]	OUT
D6	IRQ[15]	OUT
D7	IRQ[14]	OUT
D8	DAK_L[0]	IN
D9	DRQ[0]	OUT
E1	CMD_L	IN
E10	WR	BI
E17	BE_L[1]	BI
E18	LA_L[31]	BI
E2	START_L	BI
E20	LA_L[30]	BI
E21	LA_L[28]	BI
E22	LA_L[27]	BI
E23	LA_L[25]	BI
E26	LA[15]	BI
E27	LA[13]	BI
E28	LA[12]	BI
E29	LA[11]	BI
E3	EXRDY	BI
E31	LA[9]	BI
E4	EX32_L	BI
E7	EX16_L	BI
E8	SLBURST_L	BI
E9	MSBURST_L	BI
F10	MIO	BI
F11	LOCK_L	BI
F15	BE_L[3]	BI
F17	BE_L[2]	BI
F18	BE_L[0]	BI
F21	LA_L[29]	BI
F23	LA_L[26]	BI
F24	LA_L[24]	BI
F26	LA[16]	BI
F27	LA[14]	BI
F31	LA[10]	BI
G1	LA[7]	BI
G10	D[22]	BI
G12	D[25]	BI
G13	D[26]	BI
G14	D[28]	BI
G17	D[30]	BI
G18	D[31]	BI
G19	MREQ_L	OUT
G3	LA[4]	BI
G4	LA[3]	BI
G7	D[17]	BI
G8	D[19]	BI
G9	D[20]	BI
H1	LA[8]	BI
H10	D[21]	BI
H11	D[23]	BI
H12	D[24]	BI
H14	D[27]	BI
H16	D[29]	BI
H19	MAK_L	IN
H2	LA[6]	BI
H3	LA[5]	BI
H5	LA[2]	BI
H7	D[16]	BI
H8	D[18]	BI

Table 6-21 EISA Pin Out Sorted by Signal Name

PIN#	PIN NAME	PIN TYPE
A11	AEN	IN
B28	BALE	IN
B20	BCLK	IN
F18	BE_L[0]	BI
E17	BE_L[1]	BI
F17	BE_L[2]	BI
F15	BE_L[3]	BI
A10	CHRDY	BI
E1	CMD_L	IN
D8	DAK_L[0]	IN
B17	DAK_L[1]	IN
B26	DAK_L[2]	IN
B15	DAK_L[3]	IN
D10	DAK_L[5]	IN
D12	DAK_L[6]	IN
D14	DAK_L[7]	IN
D9	DRQ[0]	OUT
B18	DRQ[1]	OUT
B6	DRQ[2]	OUT
B16	DRQ[3]	OUT
D11	DRQ[5]	OUT
D13	DRQ[6]	OUT
D15	DRQ[7]	OUT
A9	D[0]	BI
C13	D[10]	BI
C14	D[11]	BI
C15	D[12]	BI
C16	D[13]	BI
C17	D[14]	BI
C18	D[15]	BI
H7	D[16]	BI
G7	D[17]	BI
H8	D[18]	BI
G8	D[19]	BI
A8	D[1]	BI
G9	D[20]	BI
H10	D[21]	BI
G10	D[22]	BI
H11	D[23]	BI
H12	D[24]	BI
G12	D[25]	BI
G13	D[26]	BI
H14	D[27]	BI
G14	D[28]	BI
H16	D[29]	BI
A7	D[2]	BI
G17	D[30]	BI
G18	D[31]	BI
A6	D[3]	BI
A5	D[4]	BI
A4	D[5]	BI
A3	D[6]	BI
A2	D[7]	BI
C11	D[8]	BI
C12	D[9]	BI
E7	EX16_L	BI
E4	EX32_L	BI
E3	EXRDY	BI
D2	IO16_L	OUT
A1	IOCHK_L	OUT
B14	IORC_L	IN
B13	IOWC_L	IN
D3	IRQ[10]	OUT
D4	IRQ[11]	OUT
D5	IRQ[12]	OUT
D7	IRQ[14]	OUT
D6	IRQ[15]	OUT
B25	IRQ[3]	OUT
B24	IRQ[4]	OUT
B23	IRQ[5]	OUT
B22	IRQ[6]	OUT
B21	IRQ[7]	OUT
B4	IRQ[9]	OUT
F31	LA[10]	BI
E29	LA[11]	BI
E28	LA[12]	BI
E27	LA[13]	BI
F27	LA[14]	BI
E26	LA[15]	BI
F26	LA[16]	BI
C8	LA[17]	BI
C7	LA[18]	BI
C6	LA[19]	BI
C5	LA[20]	BI
C4	LA[21]	BI
C3	LA[22]	BI
C2	LA[23]	BI
H5	LA[2]	BI
G4	LA[3]	BI
G3	LA[4]	BI
H3	LA[5]	BI
H2	LA[6]	BI
G1	LA[7]	BI
H1	LA[8]	BI
E31	LA[9]	BI
F24	LA_L[24]	BI
E23	LA_L[25]	BI
F23	LA_L[26]	BI
E22	LA_L[27]	BI
E21	LA_L[28]	BI
F21	LA_L[29]	BI
E20	LA_L[30]	BI
E18	LA_L[31]	BI
F11	LOCK_L	BI
D1	M16_L	IN
H19	MAK_L	IN
D17	MASTER16_L	OUT
F10	MIO	BI
C9	MRDC_L	IN
G19	MREQ_L	OUT
E9	MSBURST_L	BI
C10	MWTC_L	BI
B8	NOWS_L	OUT
B30	OSC	IN
B19	REFRESH_L	BI
B2	RESDRV	IN
A31	SA[0]	BI
A21	SA[10]	BI
A20	SA[11]	BI
A19	SA[12]	BI
A18	SA[13]	BI
A17	SA[14]	BI
A16	SA[15]	BI
A15	SA[16]	BI
A14	SA[17]	BI
A13	SA[18]	BI
A12	SA[19]	BI
A30	SA[1]	BI
A29	SA[2]	BI
A28	SA[3]	BI
A27	SA[4]	BI
A26	SA[5]	BI
A25	SA[6]	BI
A24	SA[7]	BI
A23	SA[8]	BI
A22	SA[9]	BI
C1	SBHE_L	BI
E8	SLBURST_L	BI
B12	SMRDC_L	IN
B11	SMWTC_L	IN
E2	START_L	BI
B27	TC	BI
E10	WR	BI

Power Connectors

The SystemBoard has three power connectors, each connectors is keyed to prevent reversal and all three are different sizes to prevent mixing connector and plug. The three connectors supply the following voltages: +5.0, +3.3 +12.0, -5.0, -12.0. The primary voltages (that is, significant current supplied) are +5 and +3 volts. One connector is dedicated to +5.0V, another to +3.3V, while the final connector supplies the remaining voltages, and the control signals to the power supply. The AlphaStation 600 system is designed to accommodate dual power supplies for additional power requirements *(not* as a redundancy feature). The current AlphaStation 600 configurations will only use a single supply, however. Table 6-22 lists the signal pinout and description for the third (control) power connector.

Figure 6-12 Control Power Connector Pinout

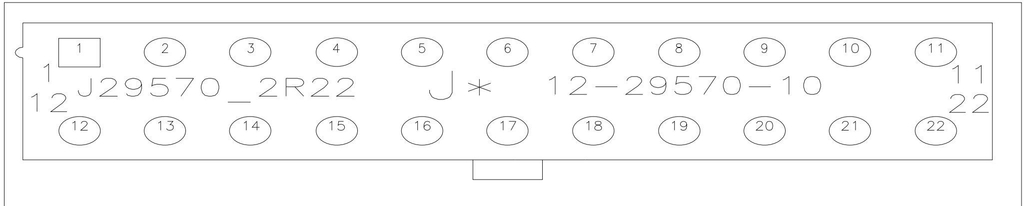


Table 6-22 Control Power Connector Pinout

PIN	SIGNAL	DESCRIPTION
1	FAN_P	Positive Fan Power (actually GND)
2	GND	System GND
3	PWR12	+ 12.0 Volts
4	PWR-12	- 12.0 Volts
5	PWR5	+5.0 Volt Sense Line
6	GND	+5.0 Volt Sense Return
7	FAN_FAULT_H	0=Fan OK, 1= Fan Failure (shutdown supply)
8	DC_OK1_H	Power supply 1 DC OK. (1= power OK, 0 = bad)
9	DC_OK2_H	Power supply 2 DC OK. (as above)
10	NC	Reserved for future use
11	NC	Reserved for future use
12	FAN_M	Negative Fan Power (actually -12V)
13	PRESENT_L1	Power Supply 1 Present (0=not present, 1= present)
14	PRESENT_L2	Power Supply 2 Present (as above)
15	NC	Reserved for future use
16	PWR-5	-5.0 Volts
17	PWR3	+3.3 Volt Sense line
18	GND	+3.3 Volt Sense Return
19	DC_ENABLE_L	0= DC power off, 1 = DC power on
20	NC	Reserved for future use
21	NC	Reserved for future use
22	NC	Reserved for future use

The other two power connectors (J19 and J20) supply +5.0 and +3.3 Volts respectively; the connector keying and pin numbering is identical to J21, except for the pin count.

Pinout for these is as follows:

- J19 : pins 1-12 = GND (Ground), pins 13-24 = PWR5 (+5.0 Volts)
- J20 : pins 1-10 = GND (Ground), pins 11-20 = PWR3 (+3.3 Volts)

Fan Connectors

The AlphaStation 600 SystemBoard has connectors for two fans, which run off a separately supplied voltage from the control power connector (J21). This voltage varies from -6.0V to -12.0V depending on thermal conditions. There are two keyed connectors on the SystemBoard (J24, J26) for fan power, into which the fan cables connect. The SystemBoard monitors the current being drawn through these connectors and asserts FAN_FAULT_H if either fan stops running, causing the power supply to shut down the system. Pinning for J24, J26 is as follows:

- Pin 1 = FAN_P (ground)
- Pin 2 = FAN_M (-12.0 Volts)

OCP Connector

The Operator Control Panel (OCP) plugs into the SystemBoard via a connector at the bottom of the SystemBoard (J22). The OCP is a liquid crystal display panel and push-button switch assembly which is located at the front of the cabinet. The following functions are controlled from the OCP:

- System Power (DC_ENABLE is connected to the leftmost of the OCP switches)
- CPU Halt (the EV5 Halt interrupt is connected to the middle OCP switch)
- System Reset (OCP_RESET, which is combined to form SYS_RST inside the GRU, is connected to the rightmost OCP switch)
- Status display via the LCD panel. This panel is driven from an I2C controller which sits on the X-BUS, a utility bus controlled from the PCI/EISA bridge chips. The console uses the OCP to display system status.

Figure 6-13 OCP Connector Pinout

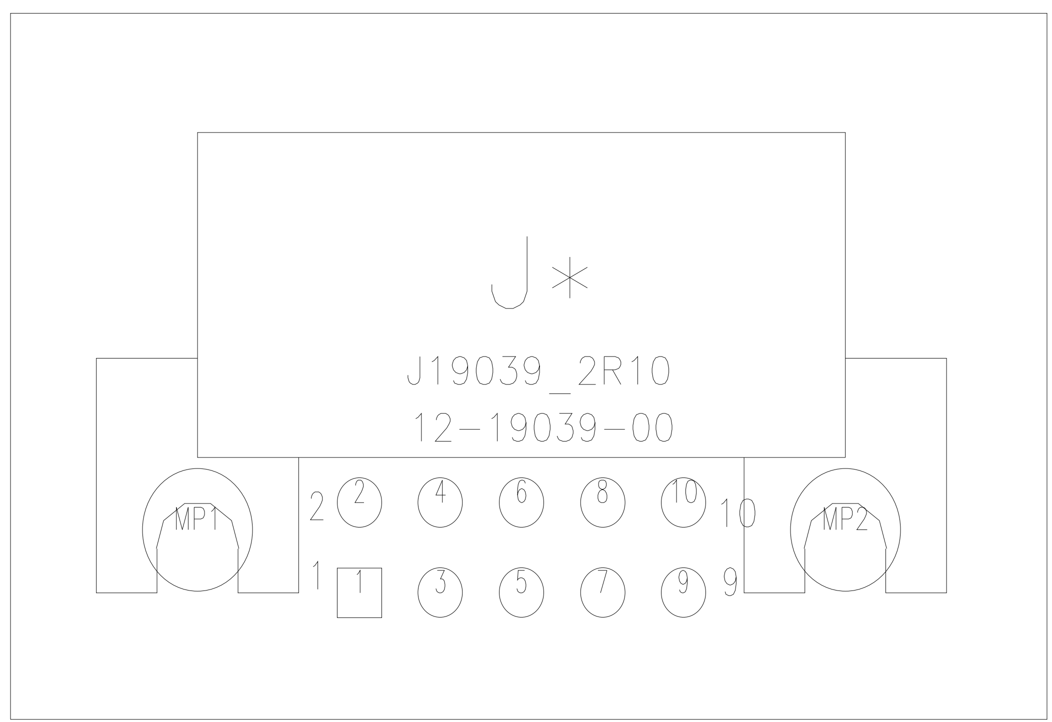


Table 6-23 OCP Connector Pinout

Pin	Signal	Description
1	PWR12	+12.0 Volts
2	PWR5	+5.0 Volts
3	GND	Ground
4	PWR5	+5.0 Volts
5	GND	Ground
6	OCP_HALT_L	System Halt, inverted & sent to EV5
7	OCP_RESET_L	System Reset button, sent to GRU and then to EV5
8	SDA	Serial Data out to OCP display panel
9	SCL	Serial Clock out to OCP display panel
10	DC_ENABLE_L	Power supply enable switch (0=power on)

Floppy Connector

The AlphaStation 600 System Board supports an internal floppy drive, which connects to the system board via a standard 34-pin floppy connector at the lower right corner of the board (J18).
The floppy drive is controlled from a National 87312 Super-I/O controller, which acts as an ISA device in the system.

Figure 6-14 Floppy Connector Pinout

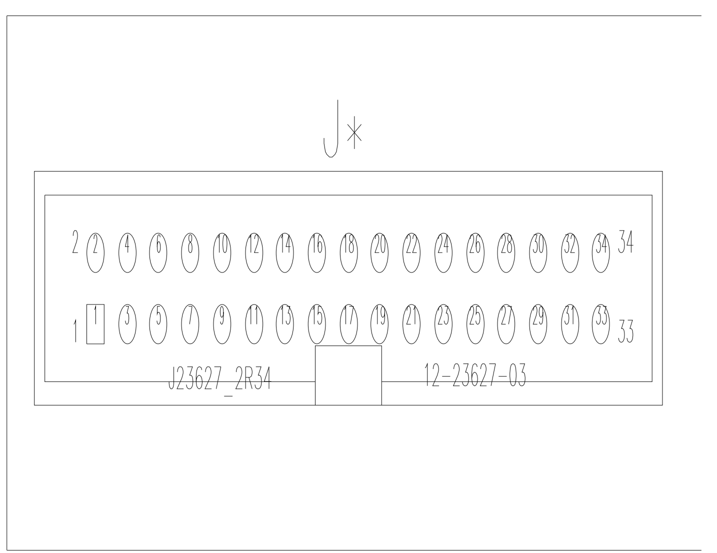


Table 6-24 Floppy Connector Pinout (Signals only)

Pin	Signal
2	DENSEL
6	DRATE0
8	INDEX_L
10	MTR0_L
12	DRSEL1
14	DRSEL0
16	MTR1_L
18	DIR_L
20	STEP_L
22	WRDATA_L
24	WGATE_L
26	TRK0_L
28	WP_L
30	RDDATA_L
32	HDSEL_L
34	DISKCH_L

Pins 1,3,5,7,9,11,13,15,19,21,23,25,29,31,33 are GND (ground)
Pins 4,17,27 are not connected

Serial ROM Connector

A serial ROM port connector is provided to allow downline loading of code directly into EV5 ICache, and also functions as a serial communication port to allow mini-console programs to communicate without using the system's serial I/O features. The SROM port is a "standard" 10 pin interface, which has been used previously on other workstation products. Pinout is shown in Figure 6-15 and listed in Table 6-25.

Figure 6-15 SROM Port Connector Pinout

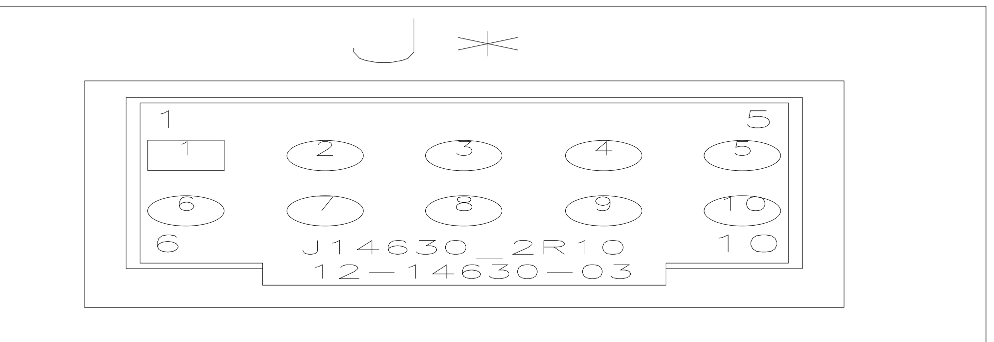


Table 6-25 SROM Port Connector Pinout

Pin	Signal	Description
1	PWR5	+5.0 Volts
2	GND	Ground
3	PWR12	+12.0 Volts
4	GND	Ground
5	GND	Ground
6	GND	Ground
7	VSDETECT	Power supply sense line
8	PWR5	+5.0 Volts
9	SROM_CLK	SROM clock & EV5 Data XMIT
10	SROM_DAT	SROM data & EV5 Data RCV

System Testability Features

The AlphaStation 600 system has a number of built-in test hooks for chip and module level testing. The list that follows provides a brief overview of these features.

- The EV5 SROM interface allows direct communication with EV5, allowing mini-console or diagnostic programs to be run in EV5 with a minimum set of hardware running.
- SROM diagnostics can be run by selecting different positions for the SROM jumpers. The AlphaStation 600 system can select among 8 programs in SROM.
- EV5 contains a JTAG port which can be used in In-Circuit-Test or during chip testing. The AlphaStation 600 system board does not provide a JTAG connector to this port, however.
- The GRU, and CIA have scan capability, and outputs can be tri-stated for ICT testing. TEST_MODE bit values are defined in table 6-26.
- Vendor supplied chips, such as the PCEB, ESC, Super I/O, and 8242 controller, have varying degrees of test features. Output enables on these devices have soft grounds, and test outputs have been brought out to via sites accessible from both sides of the module.
- PCI options slots have JTAG test pins, which are accessible for ICT testing. The AlphaStation 600 system does not chain the JTAG ports together or provide any JTAG support logic.
- PAL devices are tri-state-able. There is only one PAL in the AlphaStation 600 system; it provides PCI arbitration and I2C bus decoding. The PAL has an output enable to allow tri-stating.

Table 6-26 describes the values of the TEST_MODE bits on the CIA, DSW, and GRU ASICs. These pins are tied via resistors to the "Normal Operation" setting (01). They can be pulled up/down during In-Circuit test to select Scan mode or to Tri-State the outputs of the device.

Table 6-26 ASIC Test Mode Settings

T1	T0	Description
0	0	Tri-State all Outputs
0	1	Normal Operation Mode
1	0	SCAN mode
1	1	PLL Test Mode

The AlphaStation 600 system has incorporated a number of features to minimize system noise transmission to the outside environment (in order to comply with FCC regulations), to prevent outside noise from getting in, and to prevent internal noise coupling (crosstalk). Some of these features are outlined below:

- The external I/O area of the board (upper left corner of the system board) has been isolated from the rest of the system with an all-layer void of 100 mils. Signals crossing this void do so over in-line choke devices.
- The AlphaStation 600 system board provides a set of plated slots, connected to the I/O area ground plane, for use with an attachable I/O shield. The shield is used to keep EMI noise from leaking through gaps in the cabinet from the I/O connectors.
- Transorb components and capacitors have been added to external I/O lines to help suppress line transients that may be induced from the outside environment.

The serial, parallel, keyboard, and mouse ports all reside in the plane split area and share this protection.

- MMB modules serve as an EMI barrier between the high speed section (top), and the I/O area (bottom). MMB modules are connected to chassis via a conductive outer edge, and have shunts to ground on the system board at selected locations; spring clips are used to contact EMI landing pads on the system board to close off the gap below the MMB modules in order to reduce EMI noise.
- Selected mounting holes on the system board have been plated to allow the system board ground plane to be attached directly to the chassis for EMI reduction purposes.

7 Control and Status Registers

Register Types

All registers are on naturally aligned 64-byte addresses. There are several categories of AlphaStation 600 system registers.

Table 7-1 AlphaStation 600 Register Categories

Category	Primary User.
PCI Control registers	software
Scatter/Gather Address Translation registers	hardware, software.
Error Reporting Registers	software, firmware/diagnostics
Hardware Configuration Registers	firmware/diagnostics
Diagnostic Registers	hardware debug, diagnostics

Most of the CSRs can be read/written by the software. Some of the Diagnostic registers are reserved for hardware debug and should not be touched by software (that is, these registers should only be manipulated in a well-controlled environment (for example, power-up)).

Register Addressing

The CSRs and Flash ROM are located in the range: 87 4000 0000 to 87 FFFF FFFF:

Table 7-2 Hardware Specific Register Address Map

Start address	Selected Region
87.4000.0000	CIA General Control, Diagnostic, Performance Monitoring, and Error Logging registers
87.5000.0000	CIA Memory Control registers.
87.6000.0000	CIA: PCI Address Translation (S/G, Windows, etc)
87.7000.0000	reserved
87.8000.0000	Flash ROM, CSRs in the GRU ASIC, LEDs

The address space is a hardware-specific variant of sparse space encoding. For the CSRs, CPU address bits <27:6> are used as a longword address and CPU address <5:0> must be zero. All the CIA registers are accessed with a LW granularity.

For the Flash ROM, CPU address <30:6> defines a byte address. The fetched byte is always returned to the CPU in the first byte lane (bits <7:0>).

General Registers

Table 7-3 General CIA CSRs (Base = 87.4000.0000 Hex)

Name	Mnemonic	Offset
CIA Revision	CIA_REV	080
PCI Latency	PCI_LAT	0C0
CIA Control Register	CIA_CTRL	100
Hardware Address Extension Sparse memory	HAE_MEM	400
Hardware Address Extension Sparse I/O space	HAE_IO	440
Configuration Register	CFG	480
CIA Acknowledgment Control Register	CACK_EN	600

Table 7-4 Diagnostic Registers (Base = 87.4000.0000 Hex)

Name	Mnemonic	Offset
CIA Diagnostic Control Register	CIA_DIAG	2000
Diagnostic Check Register	DIAG_CHECK	3000

Table 7-5 Performance Monitoring Registers (Base = 87.4000.0000 Hex)

Name	Mnemonic	Offset
Performance Monitor Register	PERF_MONITOR	4000
Performance Monitor Control Register	PERF_CONTROL	4040

Table 7-6 Error Registers (Base = 87.4000.0000 Hex)

Name	Mnemonic	Offset
CPU Error Information Register 0	CPU_ERR0	8000
CPU Error Information Register 1	CPU_ERR1	8040
CIA Error Register	CIA_ERR	8200
CIA Status Register	CIA_STAT	8240
CIA Error Mask Register	ERR_MASK	8280
CIA Syndrome Register	CIA_SYN	8300
CIA Memory Port Status Register 0	MEM_ERR0	8400
CIA Memory Port Status Register 1	MEM_ERR1	8440
PCI Error Status Register 0	PCI_ERR0	8800
PCI Error Status Register 1	PCI_ERR1	8840
PCI Error Status Register 2	PCI_ERR2	8880

Memory Control Registers

Table 7-7 System Configuration Registers
(Base Address = 87.5000.0000 Hex)

Name	Mnemonic	Offset
Memory Configuration Register	MCR	000
Memory Base Address Register 0	MBA0	600
Memory Base Address Register 2	MBA2	680
Memory Base Address Register 4	MBA4	700
Memory Base Address Register 6	MBA6	780
Memory Base Address Register 8	MBA8	800
Memory Base Address Register 10	MBAA	880
Memory Base Address Register 12	MBAC	900
Memory Base Address Register 14	MBAE	980
Memory Timing Information Register 0	TMG0	B00
Memory Timing Information Register 1	TMG1	B40
Memory Timing Information Register 2	TMG2	B80

PCI Address-related Registers

Table 7-8 PCI Address and Scatter/Gather Registers
(Base Address = 87.6000.0000 Hex)

Name	Mnemonic	Offset
Scatter/Gather Translation Buffer Invalidate	TBIA	100
Window Base0	W0_BASE	400
Window Mask0	W0_MASK	440
Translated Base0	T0_BASE	480
Window Base1	W1_BASE	500
Window Mask1	W1_MASK	540
Translated Base1	T1_BASE	580
Window Base2	W2_BASE	600
Window Mask2	W2_MASK	640
Translated Base2	T2_BASE	680
Window Base3	W3_BASE	700
Window Mask3	W3_MASK	740
Translated Base3	T3_BASE	780
Window DAC Base	DAC	7C0

Scatter/Gather Address Translation Registers

Table 7-9 Address Translation Registers
(Base Address = 87.6000.0000 Hex)

Name	Mnemonic	Offset
Lockable Translation Buffer Tag0	LTB_TAG0	800
Lockable Translation Buffer Tag1	LTB_TAG1	840
Lockable Translation Buffer Tag2	LTB_TAG2	880
Lockable Translation Buffer Tag3	LTB_TAG3	8C0
Translation Buffer Tag0	TB_TAG0	900
Translation Buffer Tag1	TB_TAG1	940
Translation Buffer Tag2	TB_TAG2	980
Translation Buffer Tag3	TB_TAG3	9C0
Translation Buffer 0 Page0	TB0_PAGE0	1000
Translation Buffer 0 Page1	TB0_PAGE1	1040
Translation Buffer 0 Page2	TB0_PAGE2	1080
Translation Buffer 0 Page3	TB0_PAGE3	10C0
Translation Buffer 1 Page0	TB1_PAGE0	1100
Translation Buffer 1 Page1	TB1_PAGE1	1140
Translation Buffer 1 Page2	TB1_PAGE2	1180
Translation Buffer 1 Page3	TB1_PAGE3	11C0
Translation Buffer 2 Page0	TB2_PAGE0	1200
Translation Buffer 2 Page1	TB2_PAGE1	1240
Translation Buffer 2 Page2	TB2_PAGE2	1280
Translation Buffer 2 Page3	TB2_PAGE3	12C0
Translation Buffer 3 Page0	TB3_PAGE0	1300
Translation Buffer 3 Page1	TB3_PAGE1	1340
Translation Buffer 3 Page2	TB3_PAGE2	1380
Translation Buffer 3 Page3	TB3_PAGE3	13C0
Translation Buffer 4 Page0	TB4_PAGE0	1400
Translation Buffer 4 Page1	TB4_PAGE1	1440
Translation Buffer 4 Page2	TB4_PAGE2	1480
Translation Buffer 4 Page3	TB4_PAGE3	14C0
Translation Buffer 5 Page0	TB5_PAGE0	1500
Translation Buffer 5 Page1	TB5_PAGE1	1540
Translation Buffer 5 Page2	TB5_PAGE2	1580
Translation Buffer 5 Page3	TB5_PAGE3	15C0
Translation Buffer 6 Page0	TB6_PAGE0	1600
Translation Buffer 6 Page1	TB6_PAGE1	1640
Translation Buffer 6 Page2	TB6_PAGE2	1680
Translation Buffer 6 Page3	TB6_PAGE3	16C0
Translation Buffer 7 Page0	TB7_PAGE0	1700
Translation Buffer 7 Page1	TB7_PAGE1	1740
Translation Buffer 7 Page2	TB7_PAGE2	1780
Translation Buffer 7 Page3	TB7_PAGE3	17C0

Flash ROM Space

The region 87.8000.0000 to 87.FFFF.FFFF is used to access the GRU ASIC on the IOD bus. The GRU has a number of CSRs for the interrupt logic and the memory/cache presence detect logic. The GRU sources the Lemmon bus on which are attached 1024 KB of Flash ROM, (optional) LEDs and (possibly) some jumpers.

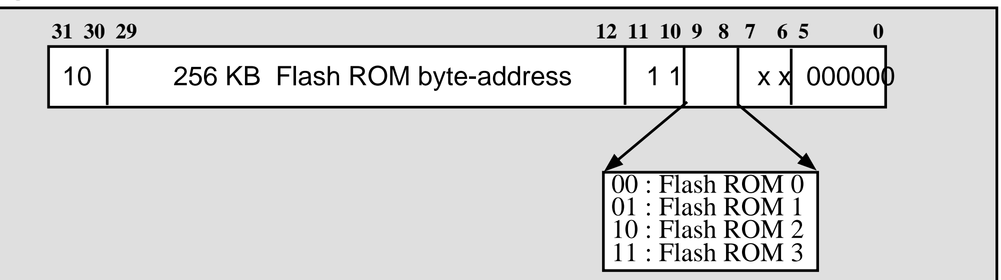
This section describes the software visible features (registers, Flash ROM, etc). For more details on the IOD bus protocol to the GRU, please refer to the GRU ASIC specification.

Addressing

For hardware convenience, a simplified version of sparse-space is used to address this region. CPU address <5:0> must be zero. The remainder of the CPU address space depends on whether the CSRs or the Flash ROM is being accessed:

- **CSRs:** CPU address <29:6> is used as a longword address for the various CSRs (see Table 7-10).
- **FLASH ROM:** The address is defined in Figure 7-1 and is tabulated in Table 7-10.
- Flash ROM address scheme

Figure 1-1 Flash ROM Address Scheme



There is a jumper to enable the VPP pin the Flash ROMs and also the FROM_WRT_EN bit <0> of CIA_DIAG must be set before the FROM can be programmed or erased. For more details of how to program the FROM see the 23000Z4-01 EPROM 256Kx8 CMOS Flash specification.

Table 7-10 GRU Space - (Base Address = 87.8000.0000 Hex)

CPU Address	Selected Region	Mnemonic
87.8000.0000 87.8000.0040 87.8000.0080 87.8000.00C0 87.8000.0100	Interrupt Request register Interrupt Mask register Interrupt Level/edge select register Interrupt High/Low IRQ select register Interrupt Clear register	INT_REQ INT_MASK INT_EDGE INT_HILO INT_CLEAR
87.8000.0140 to 87.8000.01C0	reserved	
87.8000.0200	Cache and memory configuration register	CACHE_CNFG
87.8000.0240 to 87.8000.02C0	reserved	
87.8000.0300	SET Configuration Register	SCR
87.8000.0340 to 87.8000.07C0	reserved	
87.8000.0800	LEDs (not used in current AlphaStation 600 systems)	LED
87.8000.0840 to 87.8000.08C0	reserved	
87.8000.0900	Force System Reset	RESET
87.8000.0940 to 87.8000.0BC0	reserved	
87.8000.0Cxx to 87.BFFF.FCxx	Flash ROM bank 0 256 KB byte-addressed by CPU address<29:12>	
87.8000.0Dxx to 87.BFFF.FDxx	Flash ROM bank 1 256 KB byte-addressed by CPU address<29:12>	
87.8000.0Exx to 87.BFFF.FExx	Flash ROM bank 2 256 KB byte-addressed by CPU address<29:12>	
87.8000.0Fxx to 87.BFFF.FFxx	Flash ROM bank 3 256 KB byte-addressed by CPU address<29:12>	
87.C000.00xx to 87.FFFF.FFxx	reserved	

EV5 Configuration Registers

EV5 Registers

Detailed descriptions of the EV5 registers are available in the EV5 specification[1]. The registers specific to the The AlphaStation 600 system implementation are described in Table 7-11.

Table 7-11 EV5 System Specific Registers

Name	Mnemonic
Scache Control Register	SC_CTL
BCache Control Register	BC_CONTROL
BCache Configuration Register	BC_CONFIG

[1] DECchip 21164-AA (EV5 CPU) Functional Specification

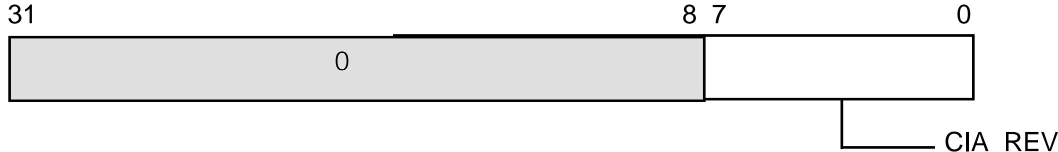
General CIA Registers - Description

CIA Revison Register (CIA_REV)

Access:

Read only, 87.4000.0080

Format:



Description:

The CIA revision Register (CIA_REV) contains the CIA revision.

Table 7-12 CIA Revision Register (CIA_REV)

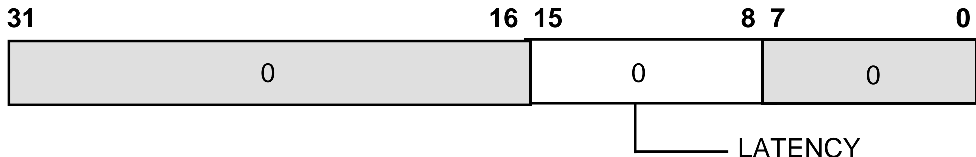
Name	Extent	Access	Init State
CIA_REV	<7:0>	RO	ASIC rev
	CIA_REV specifies the revision of the CIA ASIC • 0000.0000 is pass 1 CIA. Intetnded for early debug and initial software debug • 0000.0001 is pass 2 CIA. Works at speed, fixed bugs found in first pass		
reserved	<31:8>	RO	0

PCI Latency Register (PCI_LAT)

Access:

Read/Write, 87.4000.00C0

Format:



Description:

The PCI Latency Register (PCI_LAT) contains the PCI master latency timeout value.

Table 7-13 CIA Configuration Register (CIA_CNFG)

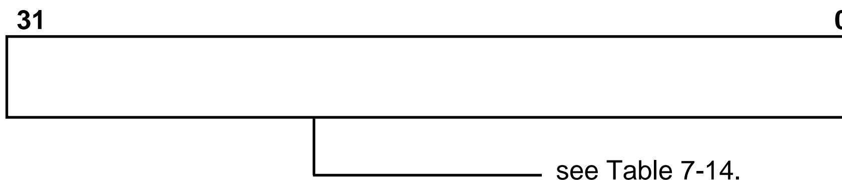
Name	Extent	Access	Init State
reserved	<7:0>	RO	0
LATENCY	<15:8>	RW	0
	PCI master Latency Timer in PCI clock cycles		
reserved	<31:16>	RO	0

CIA Control Register (CIA_CTRL)

Access:

Read/Write, 87.4000.0100

Format:



Description:

The CIA_CTRL register is a general control register for the CIA.

Table 7-14 CIA Control Register (CIA_CTRL)

Name	Extent	Access	Init State
PCI_EN	<0>	RW	0
	0: CIA asserts reset to the PCI 1: CIA does not assert reset to the PCI		
PCI_LOCK_EN	<1>	RW	0
	0: CIA will not lock when the PCI tries to lock it 1: CIA will lock when the PCI tries to lock it		
PCI_LOOP_EN	<2>	RW	0
	0: CIA will not respond as a target when it is the master 1: CIA will respond as a target when it is the master		
FST_BB_EN	<3>	RW	0
	0: CIA will not initiate fast back-to-back PCI transactions 1: CIA will initiate fast back-to-back PCI transactions		
PCI_MST_EN	<4>	RW	0
	0: CIA will not initiate PCI transactions 1: CIA will initiate PCI transactions		
PCI_MEM_EN	<5>	RW	0
	0: CIA will not respond to PCI transactions 1: CIA will respond to PCI transactions		
PCI_REQ64_EN	<6>	RW	0
	0: CIA will not request 64-bit PCI data transactions 1: CIA will request 64-bit PCI data transactions		
PCI_ACK64_EN	<7>	RW	0
	0: CIA will not accept 64-bit PCI data transactions 1: CIA will accept 64-bit PCI data transactions		
ADDR_PE_EN	<8>	RW	0
	0: CIA will not check PCI address parity errors 1: CIA will check PCI address parity errors		
PERR_EN	<9>	RW	0
	0: CIA will not check PCI data parity errors 1: CIA will check PCI data parity errors		
FILL_ERR_EN	<10>	RW	0
	0: CIA will not assert FILL_ERROR 1: CIA will assert FILL_ERROR, if an error occurs during a EV5 read miss		
MCHK_ERR_EN	<11>	RW	0
	0: CIA will not assert the ERROR Pin 1: CIA will assert the ERROR pin to report system machine check conditions		
ECC_CHK_EN	<12>	RW	0
	0: CIA will not check IOD Data 1: CIA will check the IOD Data		
ASSERT_IDLE_BC	<13>	RW	0
	0: CIA will not assert IDLE_BC pin when asserting ADDR_BUS_REQ pin 1: CIA will assert IDLE_BC pin when asserting ADDR_BUS_REQ pin		
CON_IDLE_BC	<14>	RW	0
	0: CIA may generate a non-contiguous IDLE_BC 1: CIA will guanantee that IDLE_BC is contiguous		

Table 7-14 CIA Control Register (CIA_CTRL) (continued)

<table>
<tr><td rowspan="2">CSR_IOA_BYPASS</td><td><15> RW 0</td></tr>
<tr><td>0: CIA will not bypass the I/O address queue
1: CIA will bypass the I/O address queue</td></tr>
<tr><td>IO_FLUSHREQ_EN</td><td><16> RW 0</td></tr>
<tr><td></td><td>Used in combination with CPU_FLUSHREQ_EN. The two bits control the CIA’s response to a PCI master asserting FLUSH_REQ.
<table>
<tr><th>IO_FLUSH_REQ</th><th>CPU_FLUSH_REQ</th><th>Response</th></tr>
<tr><td>0</td><td>0</td><td>Assert MEM_ACK immediately.</td></tr>
<tr><td>0</td><td>1</td><td>Assert MEM_ACK immediately. And do not allow new CPU commads to proceed.</td></tr>
<tr><td>1</td><td>0</td><td>Illegal.</td></tr>
<tr><td>1</td><td>1</td><td>Wait until IOA queue is empty, then assert MEM_ACK. And don’t allow new CPU commads to proceed.</td></tr>
</table></td></tr>
<tr><td rowspan="2">CPU_FLUSHREQ_EN</td><td><17> RW 0</td></tr>
<tr><td>See IO_FLUSHREQ_EN above.</td></tr>
<tr><td rowspan="2">ARB_EV5_EN</td><td><18> RW 0</td></tr>
<tr><td>0: Disable the bypass path from the EV5 into the Memory and IOA queue.
1: Enable the bypass path from the EV5 into the Memory and IOA queue.</td></tr>
<tr><td rowspan="2">EN_ARB_LINK</td><td><19> RW 0</td></tr>
<tr><td>0: Disable CPU Memory Reads from linking (sharing a common RAS strobe)
1: Enable CPU Memory Reads to link (sharing a common RAS strobe)</td></tr>
<tr><td rowspan="2">RD_TYPE</td><td><21:20> RW 0</td></tr>
<tr><td>This field controls the prefech algorithm used for PCI memory read command. See Table 7-15, PCI Read Prefetch Algorithm</td></tr>
<tr><td>reserved</td><td><23:22> RO 0</td></tr>
<tr><td rowspan="2">RL_TYPE</td><td><25:24> RW 0</td></tr>
<tr><td>This field controls the prefech algorithm used for PCI memory read line command. See Table 7-15, PCI Read Prefetch Algorithm</td></tr>
<tr><td>reserved</td><td><27:26> RO 0</td></tr>
<tr><td rowspan="2">RM_TYPE</td><td><29:28> RW 0</td></tr>
<tr><td>This field controls the prefech algorithm used for PCI memory read multiple command. See Table 7-15, PCI Read Prefetch Algorithm</td></tr>
<tr><td>reserved</td><td><30> RO 0</td></tr>
<tr><td rowspan="2">EN_DMA_RD_PERF</td><td><31> RW 1</td></tr>
<tr><td>0: Disable the DMA Read performance logic.
1: Enable the DMA Read performance logic.</td></tr>
</table>

Table 7-15 PCI READ Prefetch Algorithm

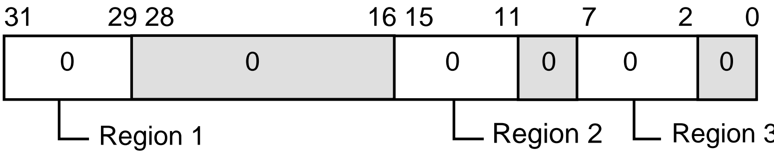
Value	Prefetch algorithm	
0 0	Short	Fetch requested LW/QW and remainder of cache block
0 1	Medium	Prefetch next block and no more. Will not cross 8 KB page boundary
1 0	Long	Keep prefetching until PCI transaction completes. Will not cross 8 KB page.
1 1	*reserved*	

Hardware Address Extension Register (HAE_MEM)

Access:

Read/Write, 87.4000.0400

Format:



Description:

This HAE_MEM Hardware Address Extension Register is used to extend a PCI Sparse-space memory address up to the full 32-bit PCI address. In Sparse Addressing mode, the CPU address provides the low-order PCI addresses bits, while the HAE_MEM provides the higher order bits.

The high-order PCI address bits <31:26> are obtained from either the *Hardware Extension Register (HAE_MEM)* or the CPU address depending on sparse space regions, as shown in Table 7-16. See Chapter 3, AlphaStation 600 Addressing for more details.

Initializing HAE_MEM to 0000.2028_{hex} will make all 3 regions contiguous starting at PCI address 0.

Table 7-16 High-order Sparse Space Bits

CPU address	Region	PCI_ Address					
		31	30	29	28	27	26
80.0000.0000 to 83.FFFF.FFFF	1	HAE_MEM<31>	HAE_MEM<30>	HAE_MEM<29>	CPU<33>	CPU<32>	CPU<31>
84.0000.0000 to 84.FFFF.FFFF	2	HAE_MEM<15>	HAE_MEM<14>	HAE_MEM<13>	HAE_MEM<12>	HAE_MEM<11>	CPU<31>
85.0000.0000 to 85.7FFF.FFFF	3	HAE_MEM<7>	HAE_MEM<6>	HAE_MEM<5>	HAE_MEM<4>	HAE_MEM<3>	HAE_MEM<2>

Table 7-17 Hardware Address Extension Register (HAE_MEM)

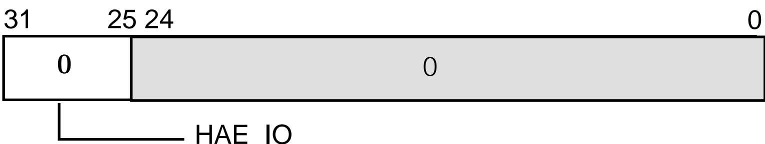
Name	Extent	Access	Init State
Region 1	<31:29>	RW	0
reserved	<28:16>	RO	0
Region 2	<15:11>	RW	0
reserved	<10:8>	RO	0
Region 3	<7:2>	RW	0
reserved	<1:0>	RO	0

Hardware Address Extension Register (HAE_IO)

Access:

Read/Write, 87.4000.0440

Format:



Description:

This HAE_IO Hardware Address Extension Register is used to extend a PCI Sparse-space IO address up to the full 32-bit PCI address. In Sparse Addressing mode, the CPU address provides the PCI addresses up to bit<24> and HAE_IO provides bits<31:25>.

On power-up this register is set to zero. In this case, Sparse I/O region A and region B both map to the lower 32 MB of sparse I/O space. Setting HAE_IO to 200.000 (hex) will make region A and region B consecutive in the lower 64 MB of PCI I/O space.

Table 7-18 Hardware Address Extension Register (HAE_IO)

Name	Extent	Access	Init State
reserved	<24:0>	RO	0
HAE_IO	<31:25>	RW	0

Configuration Type Register (CFG)

Access:

Read/Write, 87.4000.0480

Format:

Description:

The CFG bits are used as the low two address bits during an access to PCI Configuration space.

Table 7-19 CFG Register

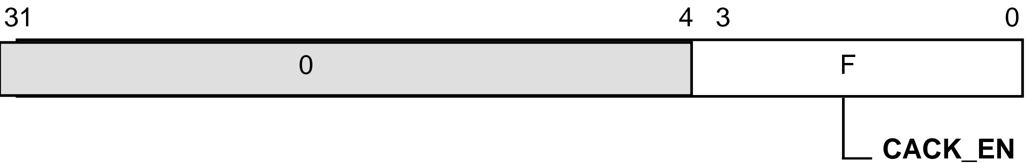
Name	Extent	Access	Init State
CFG	<1:0>	RW	0
	Bits <1:0>: 0 0 = Type 0 configuration cycle; 0 1 = Type 1 configuration cycle; 1 x = *Reserved*		
reserved	<31:2>	RO	0

CIA Acknowledgement Control Register (CACK_EN)

Access:

Read/Write, 87.40000.0600

Format:



Description:

This register has the bits that enable the CIA’s response to EV5 commands. If a bit is set, this enables the CIA to send a CACK to the EV5. If a bit is left cleared ("0"), then the corresponding EV5 command will be ignored by the CIA.

Table 7-20 CIA Acknowledgement Control Register (CACK_EN)

Name	Extent	Access	Init State
CACK_EN	<3:0>	RW	F
	Bits <3:0>	Meaning	
	xxx1	controls LOCK enable	
	xx1x	controls MB enable	
	x1xx	controls SET_DIRTY enable	
	1xxx	controls BC_VICTIM enanble	
reserved	<31:4>	RO	0

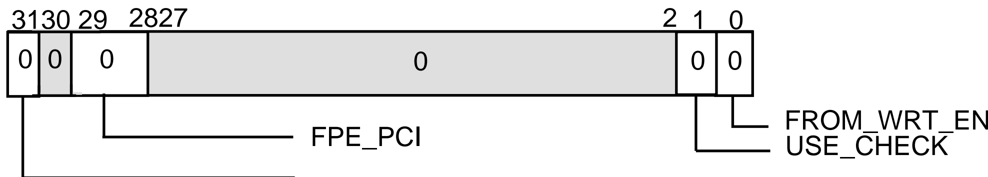
Diagnostic Registers - description

CIA Diagnostic Control Register (CIA_DIAG)

Access:

Read/Write, 87.4000.2000

Format:



Description:

The CIA force error register is a diagnostic/debug register to allow various errors to be tested.

Table 7-21 CIA Diagnostic Control Register (CIA_DIAG)

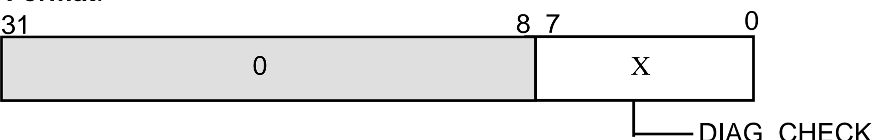
Name	Extent,	Access,	Init State
FROM_WRT_EN	<0>	RW	0
	The FROM can only be programmed when this bit set. A write to the Flash ROM when this bit isn't set, results in a System Machine Check.		
USE_CHECK	<1>	RW	0
	When set, DMA Write cycles use the value in the DIA_CHECK for ECC sent on the IOD bus.		
reserved	<27:2>	RO	0
FPE_PCI	<29:28>	RW	0
	00: Normal parity is output to the PCI 01: Bad parity is forced onto the low 32 bits of the PCI during data cycles 10: Bad parity is forced onto the high 32 bits of the PCI during data cycles 11: Bad parity is forced onto the high and low 32 bits of the PCI during address and data cycles		
reserved	<30>	RO	0
FPE_TO_EV5	<31>	RW	0
	When FPE_CPU_EV5 is set, a parity error is forced on the CPU address/CMD bus when the CIA is the bus master.		

Diagnostic Check Register (DIAG_CHECK)

Access:

Read/Write, 87.4000.3000

Format:



Description:

The DIAG_CHECK is used for diagnostic DMA writes to write a known ECC pattern into memory. This register provides the ECC that gets written to memory if the USE_CHECK bit is set in the CIA_DIAG CSR.

Table 7-22 Diagnostic Check Register (DIAG_CHECK)

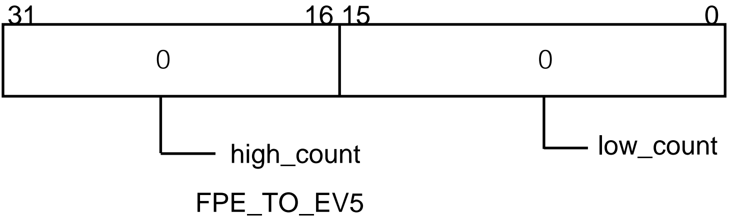
Name	Extent	Access	Init State
DIAG_ CHECK	<7:0>	RW	Undefined
	For diagnostic DMA writes, the DIAG_CHECK register provides the quadword ECC.		
reserved	<31:8>	RO	0

CIA Performance Monitor Register (PERF_MONITOR)

Access:

Read Only, 87.4000.4000

Format:



FPE_TO_EV5

Description:

The PERF_MONITOR CSR is really two 16-bit counters that can be programmed to count a variety of events. Setting up the counters is done via the PERF_CONTROL CSR. Each counter can be programmed to count events such as EV5 Read Misses revceived by CIA or DMA Writes. The PERF_MONITOR can also be set-up as a single 32-bit counter (by telling the high_count to count the low_counter overflow).

Table 7-23 CIA Performance Monitor Register (PERF_MONITOR)

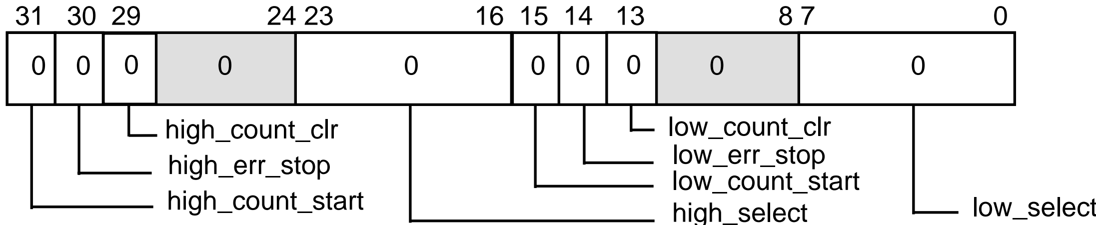
Name	Extent	Access	Init State
low_count	<15:0>	RO	0
	This is the value of the low counter.		
high_count	<31:16>	RO	0
	This is the value of the high counter.		

CIA Performance Control Register (PERF_CONTROL)

Access:

Read/Write, 87.4000.4040

Format:



Description:

Performance Monitor Control register

Table 7-24 CIA Performance Control Register (PERF_CONTROL)

Name	Extent	Access	Init State
low_select	<7:0>	RW	0
	See Table 7-25 for decoding of these bits		
reserved	<12:8>	RO	0
low_count_clr	<13>	WO	0
	write a 1 to clear the low counter		
low_err_stop	<14>	RW	0
	If CIA detects an error and this bit is set, then stop counting.		
low_count_start	<15>	RW	0
	0: don't count, keep current values 1: start counting		
high_select	<23:16>	RW	0
	See Table 7-25 for decoding of these bits		
reserved	<28:24	RO	0
high_count_clr	<29>	WO	0
	write a 1 to clear the high counter		
high_err_stop	<30>	RW	0
	If CIA detects an error and this bit is set, then stop counting.		
high_count_start	<31>	RW	0
	0: don't count, keep current values 1: start counting		

Table 7-25 PERF_CONTROL Register low/high_selects Encoding

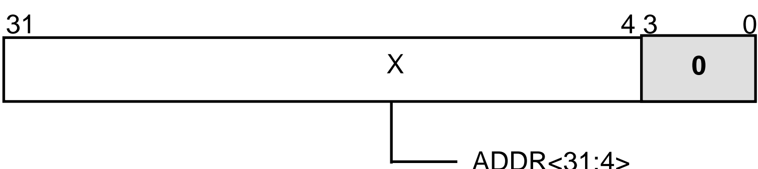
low_select<7:0> and high_select<23:16>	Description
0000 0000	for low_select: *The value of is reserved* for high_select: make 32-bit counter, The high
0000 0001	counting clock cycles, always increment
0000 0010	counting refresh cycles
0001 0000	counting # of EV5 cmd's acknowledged
0001 0001	counting # of EV5 Reads (modify or not)
0001 0010	counting # of EV5 Read miss (not modify)
0001 0011	counting # of EV5 Read Miss Modify
0001 0100	counting # of EV5 BCACHE_VICTIM cmd's that are acknowledged by CIA
0001 0101	counting # of EV5 Locks that are acknowledged by CIA
0001 0110	counting # of EV5 Memory Barrier that are acknowledged by CIA
0001 0111	counting # of EV5 FETCH or FETCH_M
0001 1000	counting # of EV5 Write Blocks (lock or not)
0010 0000	counting # of EV5 memory cmd's
0010 0001	counting # of EV5 I/O cmd's
0010 0010	counting # of EV5 I/O read cmd's
0010 0011	counting # of EV5 I/O write cmd's
0010 0100	counting # of CIA system commands issued (read/flush)
0010 0101	counting # of EV5 system Read commands issued
0010 0110	counting # of EV5 system Flush commands issued
0010 0111	counting # of times CIA received NOACK as a response
0010 1000	counting # of times CIA received Scache ACK as a response
0010 1001	counting # of times CIA received Bcache ACK as a response
0011 0000	counting # of DMA Reads (total)
0011 0001	counting # of DMA Reads (read command)
0011 0010	counting # of DMA Reads (read line command)
0011 0011	counting # of DMA Reads (read multiple command)
0011 0100	counting # of DMA Writes (total)
0011 0101	counting # of DMA Writes (write command)
0011 0110	counting # of DMA Writes (write and invalidate command)
0011 0111	counting # of DMA Dual Address cycle
0011 1000	counting # of DMA cycles that CIA issued a retry to
0011 1001	counting # of I/O cycles that CIA got a retry on
0100 0000	counting # of times PCI bus LOCK was established
0100 0001	counting # of times EV5 tried to access block that was locked
0100 0010	counting # of times DMA (that caused a flush) hit on the victim address
0101 0000	counting # of times CIA had to refill the TLB
0110 0000	counting # of single bit ECC error detected
all others	Reserved/unused, not counting

CPU Error Information Register 0 (CPU_ERR0)

Access:

Read Only, 87.4000.8000

Format:



Description:

The low-order address bits of the CPU Address/CMD bus are locked into this register when the CIA detects an error event. Clearing all the error bits in the CIA_ERR register unlocks this register. When the register is not locked the contents of this register are not defined.

The information in the CPU_ERR0 and CPU_ERR1 registers is only related to EV5 bus parity errors detected by the CIA ASIC (CIA_ERR<2>).

Table 7-26 CPU Error Information Register 0 (CPU_ERR0)

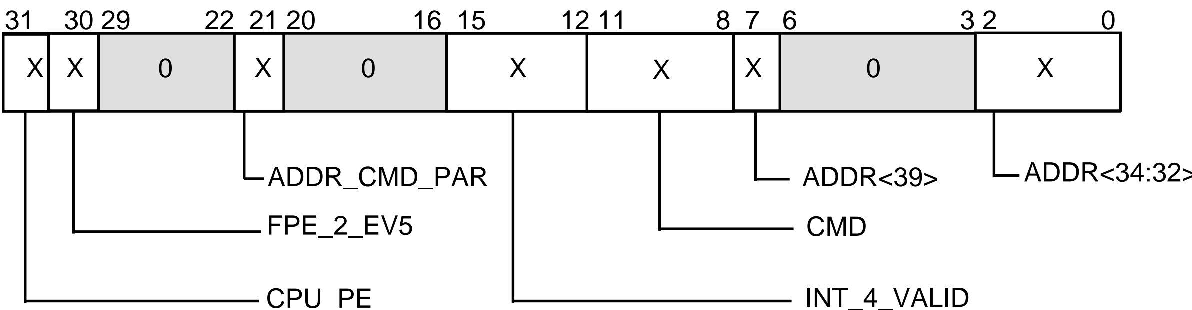
Name	Extent	Access	Init State
reserved	<3:0>	RO	0
ADDR<31:4>	<31:4>	RO	Undefined
	Contains address bits <31:4> of the current address on the CPU address/CMD bus upon an EV5 Interface error.		

CPU Error Information Register 1 (CPU_ERR1)

Access:

Read Only, 87.4000.8040

Format:



Description:

The mask and command field and the remaining address field on the CPU Address/CMD bus are locked into the ERR1 register upon the CIA detecting an error event. Clearing all the error bits in the CIA_ERR register unlocks this register. When the register is not locked, the contents of this register are not defined.

The information in the CPU_ERR0 and CPU_ERR1 registers is only related to EV5 bus parity errors detected by the CIA ASIC (CIA_ERR<2>).

Table 7-27   CPU Error Information Register 1 (CPU_ERR1)

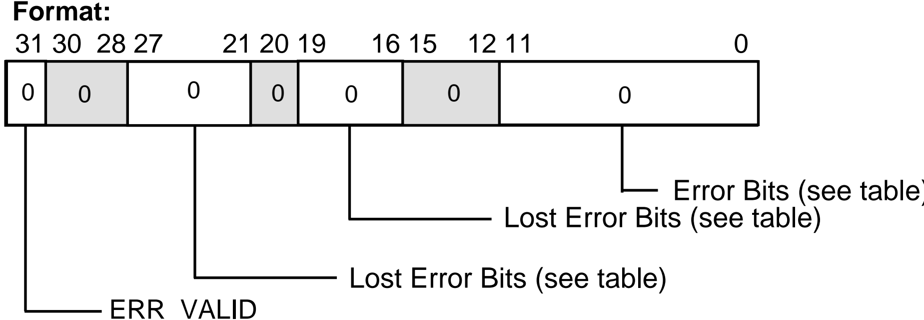
Name	Extent	Access	Init State
ADDR<34:32>	<2:0>	RO	Undefined
	Contains address bits <34:32> from the CPU address/CMD bus.		
reserved	<6:3>	RO	0
ADDR<39>	<7>	RO	Undefined
	Contains address bit <39> from the CPU address/CMD bus.		
CMD	<11:8>	RO	Undefined
	Contains the command from the CPU address/CMD bus.		
INT_4_VALID	<15:12>	RO	Undefined
	Contains the INT_4_VALID bits from the CPU address/CMD bus.		
reserved	<20:16>	RO	0
ADDR_CMD_PAR	<21>	RO	Undefined
	Contains the Parity bit from the CPU address/CMD bus.		
reserved	<29:22>	RO	0
FPE_2_EV5	<30>	RO	Undefined
	Copy of the csr bit to force bad parity to the EV5.		
CPU_PE	<31>	RO	Undefined
	If set, indicates that the CPU interface detected a parity error.		

CIA Error Register (CIA_ERR)

Access:

Read/Write to Clear, 87.4000.8200

Format:



Description:

The CIA_ERR is used by the CIA to log information pertaining to an error condition detected in the CIA ASIC. All bits of the CIA_ERR, except the LOST bits will be locked until the CIA_ERR register is cleared by a software write. The LOST bits will be set whenever the CIA_ERR is already locked and another error is detected. The CIA_ERR register will remain locked until the CIA_ERR is written to and all the individual error bits are cleared (write 1 to clear).

Table 7-28 CIA_Error Register (CIA_ERR)

Name	Extent	Access	Init State
COR_ERR	<0>	RW1C	0
	Correctable (single bit) ECC error detected. This error cannot occur for a CPU to memory read/write (CPU/mem read ECC errors are detected by the EV5; CPU/mem writes are not checked). This error is applicable to a DMA, S/G TLB miss, or an I/O Write from the EV5..		
UN_COR_ERR	<1>	RW1C	0
	Uncorrectable ECC error detected. This error cannot occur for a CPU to memory read/write (CPU/mem read ECC errors are detected by the EV5. CPU/mem writes are not checked). This error is applicable to a DMA, a S/G TLB miss, or an I/O write from the EV5.		
CPU_PE	<2>	RW1C	0
	EV5 bus parity error detected		
MEM_NEM	<3>	RW1C	0
	Access to non-existent memory detected.		
PCI_SERR	<4>	RW1C	0
	PCI bus SERR detected.		
PERR	<5>	RW1C	0
	PCI bus Data Parity error detected.		
PCI_ADDR_PE	<6>	RW1C	0
	PCI bus Address Parity error detected.		
RCVD_MAS_ABT	<7>	RW1C	0
	PCI master state machine generated master abort.		
RCVD_TAR_ABT	<8>	RW1C	0
	PCI master state machine rcvd targert abort.		
PA_PTE_INV	<9>	RW1C	0
	Invalid Page Table entry on Scatter/Gather access.		
FROM_WRT_ERR	<10>	RW1C	0
	Write to Flash ROM attempted without setting FROM_WRT_EN		
IOA_TIMEOUT	<11>	RW1C	0
	I/O timeout occurred. I/O read/write failed to get executed in 1 second.		
reserved	<15:12>	RO	0

Table 7-28 CIA_Error Register (CIA_ERR) (continued)

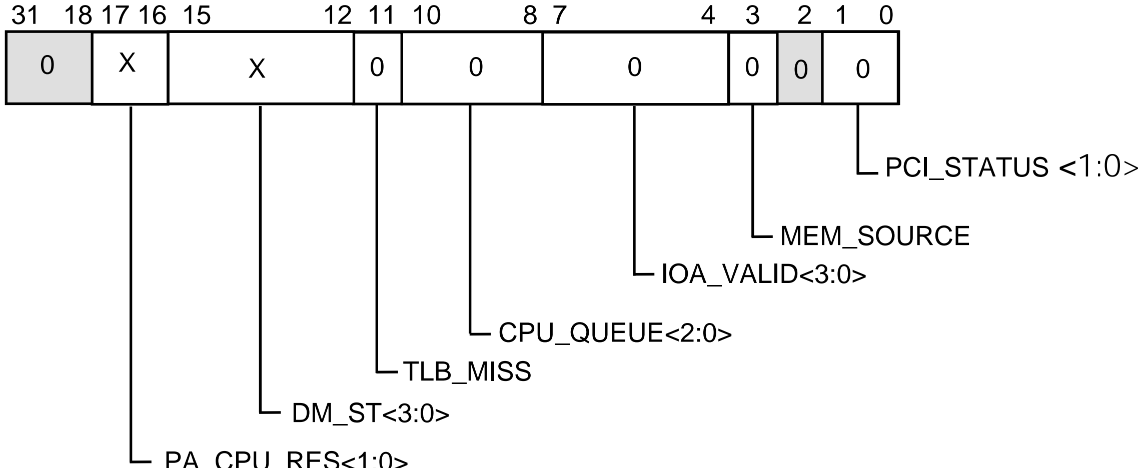
Name	Extent	Access	Init State
lost_COR_ERR	<16>	RO	0
	While CIA_ERR CSR was locked, a correctable ECC error was detected.		
lost_UN_COR_ERR	<17>	RO	0
	While CIA_ERR CSR was locked, an uncorrectable ECC error was detected.		
lost_CPU_PE	<18>	RO	0
	While CIA_ERR CSR was locked, a CPU parity error was detected.		
lost_MEM_NEM	<19>	RO	0
	While CIA_ERR CSR was locked, an access to non-existent memory was detected.		
reserved	<20>	RO	0
lost_PERR	<21>	RO	0
	While CIA_ERR CSR was locked, a PCI Data Parity error was detected.		
lost_PCI_ADDR_PE	<22>	RO	0
	While CIA_ERR CSR was locked, a PCI Address Parity error was detected.		
lost_RCVD_MAS_ABT	<23>	RO	0
	While CIA_ERR CSR was locked, the PCI master state machine generated a Master Abort.		
lost_RCVD_TAR_ABT	<24>	RO	0
	While CIA_ERR CSR was locked, the PCI master state machine recveived a target abort.		
lost_PA_PTE_INV	<25>	RO	0
	While CIA_ERR CSR was locked, an Invalid Page Table entry on Scatter/Gather access occurred.		
lost_FROM_WRT_ERR	<26>	RO	0
	While CIA_ERR CSR was locked, a write to Flash ROM with out setting FROM_WRT_EN was attempted.		
lost_IOA_TIMEOUT	<27>	RO	0
	While CIA_ERR CSR was locked, an I/O timeout occurred. An I/O read/write failed to get executed in 1 second.		
reserved	<30:28>	RO	0
ERR_VALID	<31>	RO	0
	An error has been detected and the CIA error registers are all locked.		

CIA Status Register (CIA_STAT)

Access:

Read Only, 87.4000.8240

Format:



Description:

This register contains status information related to the error stored in the CIA_ERR register. The intent is to provide a snapshot of the status (state) of the CIA when the error was detected.

Table 7-29 CIA Status Register (CIA_STAT)

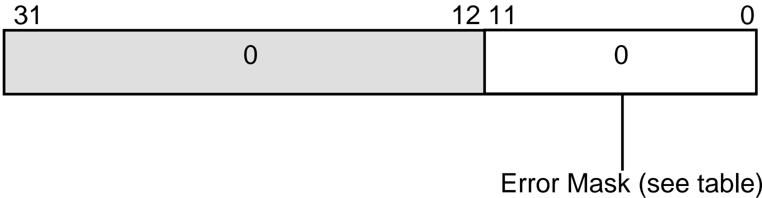
<table>
<tr><th>Name</th><th>Extent</th><th>Access</th><th>Init State</th></tr>
<tr><td rowspan="2">PCI_STATUS<0></td><td><0></td><td>RO</td><td>0</td></tr>
<tr><td colspan="3">If set, then the PCI target state machine is active.</td></tr>
<tr><td rowspan="2">PCI_STATUS<1></td><td><1></td><td>RO</td><td>0</td></tr>
<tr><td colspan="3">If set, then the PCI master state machine is active.</td></tr>
<tr><td>reserved</td><td><2></td><td>RO</td><td>0</td></tr>
<tr><td rowspan="2">MEM_SOURCE</td><td><3></td><td>RO</td><td>0</td></tr>
<tr><td colspan="3">0: source of the memory cycle is the EV5
1: source of the memory cycle is the PCI</td></tr>
<tr><td rowspan="2">IOA_VALID<3:0></td><td><7:4></td><td>RO</td><td>0</td></tr>
<tr><td colspan="3">valid bits for the IO command/address queue</td></tr>
<tr><td rowspan="2">CPU_QUEUE<2:0></td><td><10:8></td><td>RO</td><td>0</td></tr>
<tr><td colspan="3">valid bits for the CPU command/address queue</td></tr>
<tr><td rowspan="2">TLB_MISS</td><td><11></td><td>RO</td><td>0</td></tr>
<tr><td colspan="3">If set, then a TLB MISS refill was in progress when the error occurred.</td></tr>
<tr><td rowspan="2">DM_ST<3:0></td><td><15:12></td><td>RO</td><td>Undefined</td></tr>
<tr><td colspan="3">These bits represent the state of the logic that moves data between the CIA and the DSW ASICs.
<table>
<tr><th>DM_ST<3:0></th><th>Definition</th></tr>
<tr><td>0000</td><td>idle</td></tr>
<tr><td>0001</td><td>restarting DSW IOW buffers</td></tr>
<tr><td>0010</td><td>I/O Write</td></tr>
<tr><td>0110</td><td>DMA Read or TLB miss</td></tr>
<tr><td>0111</td><td>DMA Write</td></tr>
<tr><td>1000</td><td>I/O Write to the GRU ASIC</td></tr>
<tr><td>1001</td><td>I/O Read to the GRU ASIC</td></tr>
<tr><td>1010</td><td>I/O Read to an internal CIA CSR</td></tr>
<tr><td>1011</td><td>I/O Read to the PCI bus (32-bit/64-bit)</td></tr>
<tr><td>others</td><td>reserved</td></tr>
</table></td></tr>
<tr><td rowspan="2">PA_CPU_RES<1:0></td><td><17:16></td><td>RO</td><td>0</td></tr>
<tr><td colspan="3">EV5 response for the DMA. 00 = no response, 01 = no ack, 10 = SCache hit, 11 = BCache hit</td></tr>
<tr><td>reserved</td><td><31:18></td><td>RO</td><td>0</td></tr>
</table>

CIA Error Mask Register (ERR_MASK)

Access:

Read/Write, 87.4000.8280

Format:



Description:

The ERR_MASK is used to disable the logging and reporting of errors. On power-up the default is for error logging to be disabled (ERR_MASK = 0). For each bit a zero (0) disables the logging/reporting of that error, a one (1) enables the corresponding error logging and reporting.

Table 7-30 CIA Error Mask Register (ERR_MASK)

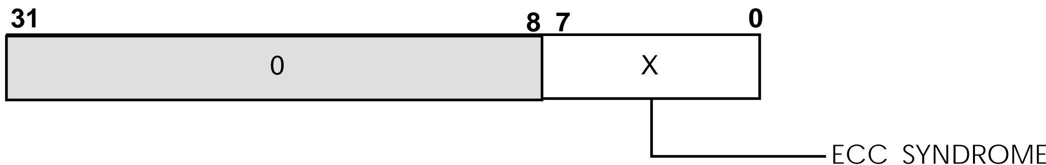
Name	Extent	Access	Init State
COR_ERR	<0>	RW	0
	Disable/enable error logging/reporting for correctable ECC errors.		
UN_COR_ERR	<1>	RW	0
	Disable/enable error logging/reporting for un-correctable ECC errors.		
CPU_PE	<2>	RW	0
	Disable/enable error logging/reporting for CPU Parity errors.		
MEM_NEM	<3>	RW	0
	Disable/enable error logging/reporting for Non-Existant Memory access errors.		
PCI_SERR	<4>	RW	0
	Disable/enable error logging/reporting for PCI SERR errors.		
PERR	<5>	RW	0
	Disable/enable error logging/reporting for PCI Data Parity errors.		
PCI_ADDR_PE	<6>	RW	0
	Disable/enable error logging/reporting for PCI Address Parity errors.		
RCVD_MAS_ABT	<7>	RW	0
	Disable/enable error logging/reporting for PCI Master Abort Errors.		
RCVD_TAR_ABT	<8>	RW	0
	Disable/enable error logging/reporting for PCI Target Abort errors.		
PA_PTE_INV	<9>	RW	0
	Disable/enable error logging/reporting for invald PTE errors.		
FROM_WRT_ERR	<10>	RW	0
	Disable/enable error logging/reporting for FROM write errors.		
WIOA_TIMEOUT	<11>	RW	0
	Disable/enable error logging/reporting for I/O Timeout errors.		

CIA Error Syndrome Register (CIA_SYN)

Access:

Read Only, 87.4000.8300

Format:



Description:

The CIA_SYN register is used by the CIA to log information pertaining to an error detected by the ECC checker. The syndrome is locked into the CIA_SYN register upon a CIA error. Clearing all the error bits in the CIA_ERR register unlocks this register. When the register is not locked the contents of this register are not defined.

Table 7-31 CIA Error Syndrome Register (CIA_SYN)

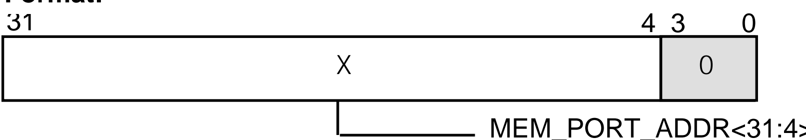
Name	Extent	Access	Init State
ECC_SYNDROME	<7:0>	RO	Undefined
reserved	<31:8>	RO	0

CIA Memory Port Status Register 0 (MEM_ERR0)

Access:

Read Only, 87.4000.8400

Format:



Description:

The low-order address bits of the Memory Port Address bus are locked into this register upon a CIA detected error. The contents are read only. Clearing all the error bits in the CIA_ERR register unlocks this register. When the register is not locked the contents of this register are not defined.

Table 7-32 CIA Memory Port Status Register 0 (MEM_ERR0)

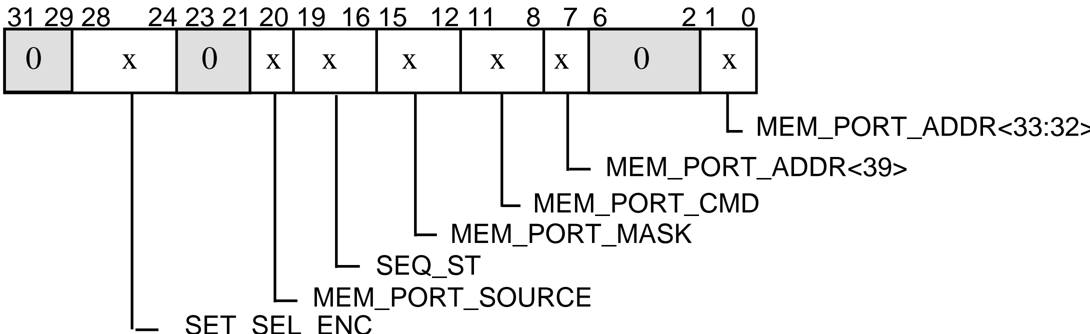
Name	Extent,	Access,	Init State
reserved	<3:0>	RO	undefined
MEM_PORT_ADDR<31:4>	<31:4>	RO	undefined
	Contains address bits <31:4> of the current address in the Memory port when the CIA detects an error.		

CIA Memory Port Status Register 1 (MEM_ERR1)

Access:

Read Only, 87.4000.8440

Format:



Description:

The command, memory mask (INT4_VALID from CPU), Memory sequencer state, the source of the command, an encoded Set Select field, and the remaining address field are locked into the MEM_ERR1 register upon a CIA error. Clearing all the error bits in the CIA_ERR register unlocks this register. When the register is not locked, the contents of this register are not defined.

Table 7-33 CIA Memory Port Status Register 1 (MEM_ERR1)

Name	Extent,	Access,	Init State
MEM_PORT_ADDR<33:32>	<1:0>	RO	Undefined
	Contains address bits <33:32> of the current address in the Memory port when the CIA detects an error.		
reserved	<6:2>	RO	0
MEM_PORT_ADDR<39>	<7>	RW	Undefined
	Address bit <39>		
MEM_PORT_CMD	<11:8>	RO	Undefined
	The memory command when the error occurred.		
MEM_PORT_MASK	<15:12>	RW	Undefined
	The mask bits when the error occurred.		
SEQ_ST	<19:16>	RW	Undefined
	The memory sequencer state when the error occurred.		
MEM_PORT_SOURCE	<20>	RW	Undefined
	Source of the memory command, 0= CPU , 1= DMA.		
reserved	<23:21>	RO	0
SET_SEL_ENC	<28:24>	RW	Undefined
	Encoded set select, indicates which memory set was active when the error occurred.		
reserved	<31:29>	RO	0

Table 7-34 MEM_PORT_CMD Encodings

MEM_PORT_SOURCE	MEM_PORT_CMD	Desription
0	011X	EV5 Write Block or Write Block Lock
0	10XX	EV5 Read Miss, Read Miss Modify
0	1100	EV5 BC_Victim
0	111X	EV5 Read Miss Modify
1	10XX	DMA Read, DMA Read Modify
1	001X	DMA Write
all others are reserved		

Table 7-35 SEQ_ST Encodings

SEQ_ST	Desription
0000	Idle
0001	DMA Read or Write
0010, 0011	EV5 Read Miss (or Read Miss Modify) with Victim
0100, 0101, 0110	EV5 Read Miss (or Read Miss Modify) no Victim
0111, 1000, 1001	Refresh
1100	Idle, waiting for DMA Pending Read
1110, 1111	Idle, RAS Precharge
all others are reserved	

Table 7-36 SET_SEL_ENC Encodings

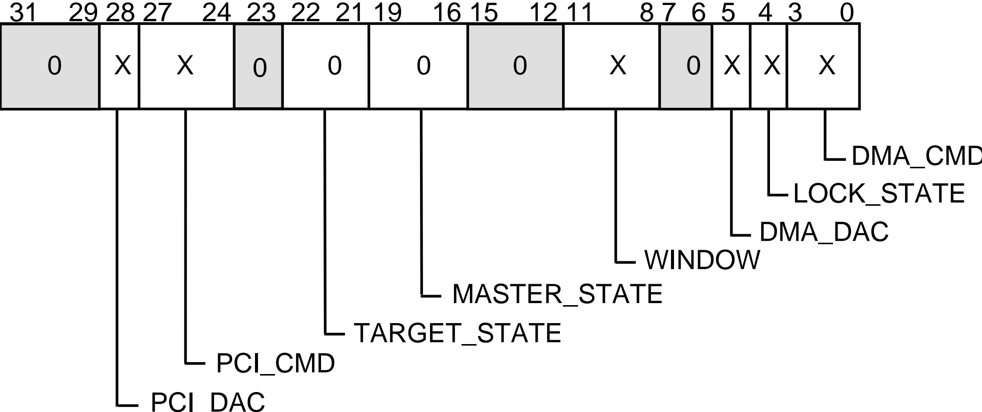
SET_SEL_ENC	Desription
00000	Set 0 Selected
00001	Set 1 Selected
00010	Set 2 Selected
00011	Set 3 Selected
00100	Set 4 Selected
00101	Set 5 Selected
00110	Set 6 Selected
00111	Set 7 Selected
01000	Set 8 Selected
01001	Set 9 Selected
01010	Set A Selected
01011	Set B Selected
01100	Set C Selected
01101	Set D Selected
01110	Set E Selected
01111	Set F Selected
10000	No Set selected
11111	Refresh Cycle
all others are reserved	

PCI Error Register 0 (PCI_ERR0)

Access:

Read Only, 87.4000.8800

Format:



Description:

The PCI_ERR0 register is used by the CIA to log information pertaining to the state of the PCI interface when an error condition is detected by CIA. The CSR is locked, as are all CIA error registers, when the CIA detects an error. And, the CSR is unlocked when the CIA_ERR CSR is cleared. When the register is not locked, the contents are unpredictable.

The data in the WINDOW, DMA_DAC, and DMA_CMD fields is associated with the address stored in the PCI_ERR1 register. This group and PCI_ERR1 hold information related to the following errors:

- Errors associated with the memory while the CIA is handling a DMA
  - Correctable ECC error (CIA_ERR<0>)
  - Uncorrectable ECC error (CIA_ERR<1>)
  - Access to non-existent memory (CIA_ERR<3>)
  - Invalid Page Table entry (CIA_ERR<9>)

The data in the PCI_DAC, PCI_CMD, TARGET_STATE, and MASTER_STATE fields is associated with the address stored in the PCI_ERR2 register. This group and the PCI_ERR2 register hold information related to the following error conditions:

- Errors associated with the PCI bus
  - PCI Data Parity error (CIA_ERR<5>)
  - PCI Address Parity Error (CIA_ERR<6>)
  - PCI Master Abort (CIA_ERR<7>)
  - PCI Target Abort (CIA_ERR<8>)
  - IOA Timeout (CIA_ERR<11>)

The LOCK_STATE field provides general information about the current state of CIA not specifically associated with either PCI_ERR1 or PCI_ERR2.

Table 7-37 PCI Error Register 0 (PCI_ERR0)

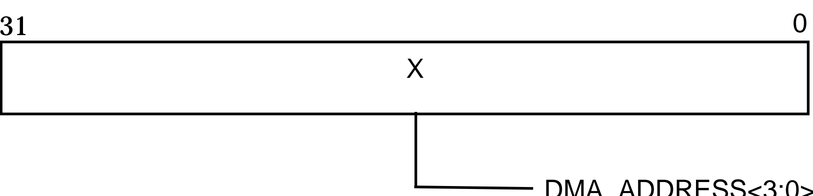
Name	Extent	Access	Init State
DMA_CMD	<3:0>	RO	x
	The PCI command of the current DMA.		
LOCK_STATE	<4>	RO	x
	If set then the CIA was locked when the error was detected.		
DMA_DAC	<5>	RO	x
	If set then the current DMA is a dual address cycle command.		
reserved	<7:6>	RO	0
WINDOW	<11:8>	RO	x
	Indicates which window (if any) was selected by the PCI address. • 0000 No window active • 0001 Window 0 hit • 0010 Window 1 hit • 0100 Window 2 hit • 1000 Window 3 hit		
reserved	<15:12>	RO	0
MASTER_STATE	<19:16>	RO	0
	0: Idle 1: Drive Bus 2: Address Step Cycle 3: Address Cycle 4: Data Cycle 5: Last Read Data Cycle 6: Last Write Data Cycle 7: Read Stop Cycle 8: Write Stop Cycle 9: Read Turnaround Cycle A: Write Turnaround Cycle B: Reserved C: Reserved D: Reserved E: Reserved F: Unknown State		
TARGET_STATE	<22:20>	RO	0
	0: Idle 1: Busy 2: Read Data Cycle 3: Write Data Cycle 4: Read Stop Cycle 5: Write Stop Cycle 6: Read Turnaround Cycle 7: Write Turnaround Cycle		
reserved	<23>	RO	0
PCI_CMD	<27:24>	RO	x
	The current PCI command.		
PCI_DAC	<28>	RO	x
	If set then the current PCI command is a dual address cycle command.		
reserved	<31:29>	RO	0

PCI Error Register 1 (PCI_ERR1)

Access:

Read Only, 87.4000.8840

Format:



Description:

The PCI_ERR1 register is used by the CIA to log PCI address <31:0> for the current DMA pertaining to an error condition logged in PCI_ERR0. This register is locked whenever the CIA detects an error. This register always captures DMA address<31:0>, even for a DMA DAC cycle. The most significant DMA address<39:32> can be obtained from the W_DAC register; DMA address<63:40> had to be zero for the CIA to hit on the DAC cycle. The register is unlocked when the error bits in the CIA_ERR CSR have all been cleared. Contents of this register are unpredictable when not locked.

The PCI_ERR1 register and some fields in PCI_ERR0 (WINDOW, DMA_DAC, and DMA_CMD) is associated hold information related to the following errors:

- Errors associated with the memory while the CIA is handling a DMA
  - Correctable ECC error (CIA_ERR<0>)
  - Uncorrectable ECC error (CIA_ERR<1>)
  - Access to non-existent memory (CIA_ERR<3>)
  - Invalid Page Table entry (CIA_ERR<9>)

Table 7-38 PCI Error Register 1 (PCI_ERR1)

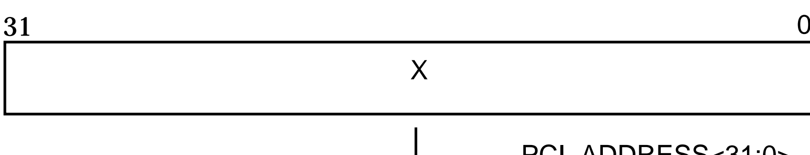
Name	Extent	Access	Init State
DMA_ADDRESS<31:0>	<31:0>	RO	x
	Contains the DMA address <31:0>.		

PCI Error Register 2 (PCI_ERR2)

Access:

Read Only, 87.4000.8880

Format:



Description:

The PCI_ERR2 register is used by the CIA to log PCI address <31:0> pertaining to an error condition logged in PCI_ERR0. This register is locked whenever the CIA detects an error. This register always captures PCI address<31:0>, even for a DMA DAC cycle. The most significant PCI address<39:32> can be obtained from the W_DAC register; PCI address<63:40> had to be zero for the CIA to hit on the DAC cycle. The register is unlocked when the error bits in the CIA_ERR CSR have all been cleared. Contents of this register are unpredictable when not locked.

The PCI_ERR2 register and some fields in PCI_ERR0 (PCI_DAC, PCI_CMD, TARGET_STATE, and MASTER_STATE) hold information related to the following error conditions:

- Errors associated with the PCI bus
  - PCI Data Parity error (CIA_ERR<5>)
  - PCI Address Parity Error (CIA_ERR<6>)
  - PCI Master Abort (CIA_ERR<7>)
  - PCI Target Abort (CIA_ERR<8>)
  - IOA Timeout (CIA_ERR<11>)

Table 7-39 PCI Error Register 2 (PCI_ERR2)

Name	Extent	Access	Init State
PCI_ADDRESS<31:0>	<31:0>	RO	x
	Contains the PCI address <31:0>.		

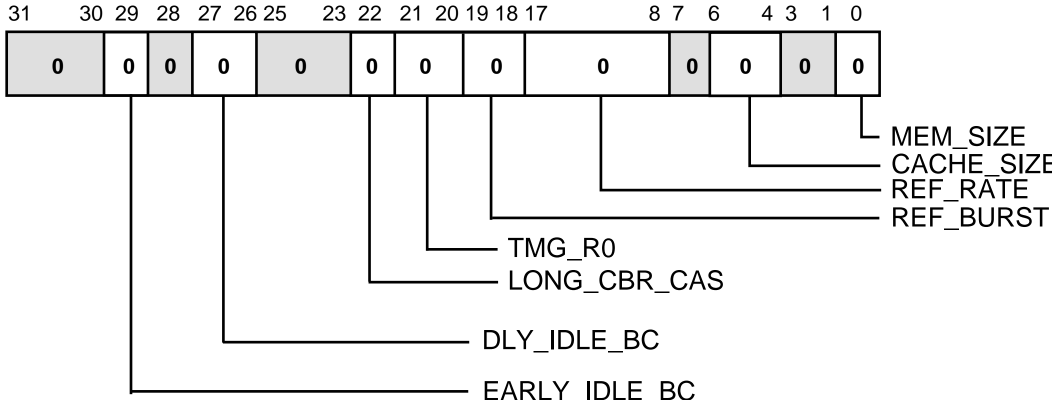
Memory Control Registers - description

Memory Configuration Register (MCR)

Access:

Read/Write, 87.5000.000

Format:



Description:

The MCR register defines the AlphaStation 600 system and memory configuration. The bits within this register are used to configure the CIA memory controller.

Table 7-40 Memory Configuration Register (MCR)

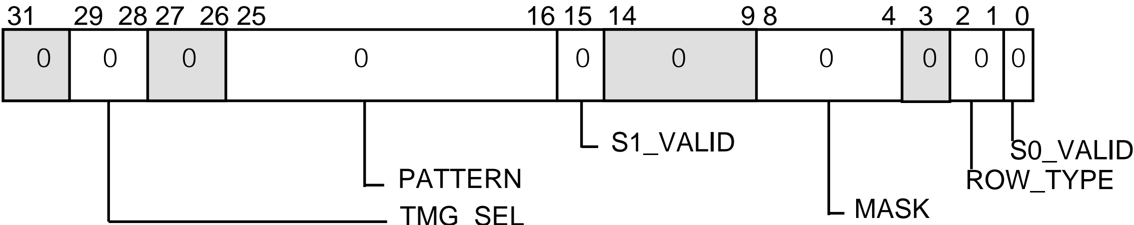
<table>
<tr><th>Name</th><th>Extent</th><th>Access</th><th>Init State</th></tr>
<tr><td rowspan="2">MEM_SIZE</td><td><0></td><td>RW</td><td>0</td></tr>
<tr><td colspan="3">When MEM_SIZE is set (value=1), the memory port is configured to a 256-bit data path and MMB0 and MMB1 must be populated identically. If not, an illegal configuration is flagged. When MEM_SIZE is clear (value=0), the memory port is configured to a 128-bit data path.
Note: This bit is to be set by firmware and must be 1 for AlphaStation 600.</td></tr>
<tr><td>reserved</td><td><3:1></td><td>RO</td><td>0</td></tr>
<tr><td rowspan="2">CACHE_SIZE</td><td><6:4></td><td>RW</td><td>0</td></tr>
<tr><td colspan="3">CACHE_SIZE is determined by firmware using the CACHE_CNFG register and written before any memory cycles are started. For encodings see Table 7-62.</td></tr>
<tr><td>reserved</td><td><7></td><td>RO</td><td>0</td></tr>
<tr><td rowspan="2">REF_RATE</td><td><17:8></td><td>RW</td><td>0</td></tr>
<tr><td colspan="3">The REF_RATE controls the memory refresh rate. An 10-bit free-running counter, starting at 0, counts upto the REF_RATE and resets to zero once the REF_RATE value is reached. Memory is refreshed everytime the REF_RATE value is reached.
Setting the REF_RATE to be 0, will disable refreshes.</td></tr>
<tr><td rowspan="2">REF_BURST</td><td><19:18></td><td>RW</td><td>0</td></tr>
<tr><td colspan="3">The refresh state machine can set up to do all or half of the SIMMs at once and to do one or two refreshes when the REF_RATE counter rolls over.
• 0 0, ras all SIMMs at once, single refresh.
• 0 1, ras all SIMMs at once, double refresh (burst).
• 1 0, ras half of SIMMs at once, single refresh.
• 1 1, ras half of SIMMs at once, double refresh (burst).</td></tr>
<tr><td rowspan="2">TMG_R0</td><td><21:20></td><td>RW</td><td>0</td></tr>
<tr><td colspan="3">Controls the row address set-up.
<table><tr><th>TMG_R0</th><th>Nominal Row Address Set-up</th></tr><tr><td>00</td><td>15 ns</td></tr><tr><td>01</td><td>30 ns</td></tr><tr><td>10</td><td>45 ns</td></tr><tr><td>11</td><td>60 ns</td></tr></table></td></tr>
<tr><td rowspan="2">LONG_CBR_CAS</td><td><22></td><td>RW</td><td>1</td></tr>
<tr><td colspan="3">0: The refresh (CAS-before-RAS) CAS pulse width = 60 ns
1: The refresh (CAS-before-RAS) CAS pulse width = 90 ns</td></tr>
<tr><td>reserved</td><td><25:23></td><td>RO</td><td>0</td></tr>
<tr><td rowspan="2">DLY_IDLE_BC</td><td><27:26></td><td>RW</td><td>0</td></tr>
<tr><td colspan="3">must be 0</td></tr>
<tr><td>reserved</td><td><28></td><td>RO</td><td>0</td></tr>
<tr><td rowspan="2">EARLY_IDLE_BC</td><td><29></td><td>RW</td><td>0</td></tr>
<tr><td colspan="3">muat be 1</td></tr>
<tr><td>reserved</td><td><31:30></td><td>RO</td><td>0</td></tr>
</table>

Memory Base Address Registers 0-E (MBA0-E)

Access:

R/W, address = 87.5000.0600, 87.5000.0680, 87.5000.0700, ... 87.5000.980

Format:



Description:

There are 8 MBA registers (MBA0,2,4,6,8,A,C,E). Each controls two of the 16 (8 in the current AlphaStation 600 system) possible banks of memory which the CIA can support. The bits within the MBA register provide a PATTERN which is compared with the incoming address to determine the bank being accessed. The minimum bank size for an MBA register is 16 Mbytes.

Table 7-41 Memory Base Address Registers 2,4,6,8,A,C,E (MBA0,2,4,6,8,A,C,E)

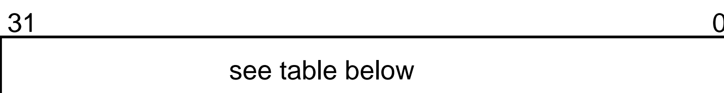
<table>
<tr><th>Name</th><th>Extent</th><th>Access</th><th>Init State</th></tr>
<tr><td rowspan="2">S0_VALID</td><td><0></td><td>RW</td><td>0</td></tr>
<tr><td colspan="3">If set, then side 0 for the bank is valid.</td></tr>
<tr><td rowspan="2">ROW_TYPE</td><td><2:1></td><td>RW</td><td>0</td></tr>
<tr><td colspan="3">ROW_TYPE specifies the row and column configuration of the physical bank being referenced. The ROW_TYPE is used in generating the memory address map.
<table>
<tr><th>ROW_type</th><th>Row / Column</th></tr>
<tr><td>00</td><td>10 row 10 column</td></tr>
<tr><td>01</td><td>12 row 10 column or 11x11</td></tr>
<tr><td>10</td><td>13 row 11 column or 12x12</td></tr>
<tr><td>11</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>reserved</td><td><3></td><td>RO</td><td>0</td></tr>
<tr><td rowspan="2">MASK</td><td><8:4></td><td>RW</td><td>0</td></tr>
<tr><td colspan="3">The MASK indicates which bits are used in comparing the MBA[PATTERN] with the physical address (memory address). The mask field essentially indicates the size of the memory SIMMs in the bank corresponding to the MBA register. For a 256-bit memory data bus, the valid MASK fields are:
<table>
<tr><th>Size of SIMM</th><th>MASK (Hex)</th><th>Comment</th></tr>
<tr><td>4 or 8 Mbytes</td><td>00001</td><td>ADDR[24] is not used in the address comparrison</td></tr>
<tr><td>16 or 32 Mbytes</td><td>00111</td><td>ADDR[26:24] are not used in the address comparrison</td></tr>
<tr><td>64 or 128 Mbytes</td><td>11111</td><td>ADDR[28:24] are not used in the address comparrison</td></tr>
</table></td></tr>
<tr><td>reserved</td><td><14:9></td><td>RO</td><td>0</td></tr>
<tr><td rowspan="2">S1_VALID</td><td><15></td><td>RW</td><td>0</td></tr>
<tr><td colspan="3">If set, then side 1 for the bank is valid. Side 1 can't be valid unless side 0 is also valid.</td></tr>
<tr><td rowspan="2">PATTERN</td><td><25:16></td><td>RW</td><td>0</td></tr>
<tr><td colspan="3">The PATTERN field is compared with the incoming CPU ADDR[33:24] and then ORed with the MASK field to determine if the bank corresponding to this MBA is being accessed. The PATTERN basically indicates the base address of the memory bank, while the MASK indicates the size of the bank.</td></tr>
<tr><td>reserved</td><td><27:26></td><td>RO</td><td>0</td></tr>
<tr><td rowspan="2">TMG_SEL</td><td><29:28></td><td>RW</td><td>0</td></tr>
<tr><td colspan="3">
<table>
<tr><th><29:28></th><th>Use TMG CSR</th><th>Description</th></tr>
<tr><td>00
01
10
11</td><td>TMG0
TMG1
TMG2
reserved</td><td>Used to control memory timing of 50,60 ns SIMMs
Used to control memory timing of 70 ns SIMMs
Used to control memory timing of 80 ns SIMMs
reserved</td></tr>
</table></td></tr>
<tr><td>reserved</td><td><31:30></td><td>RO</td><td>0</td></tr>
</table>

Memory Timing Registers (TMG0-TMG2)

Access:

R/W, 87.5000.0B00, 87.5000.0B40, 87.5000.0B80

Format:



Description:

The TMG CSRs contain the AlphaStation 600 memory timing parameters which control the memory sequencer.

Table 7-42 Memory Timing Parameters, Encoding Values

Parameter	Short Description	Encoded Value							
		000	001	010	011	100	101	110	111
TMG_R1	Read Starting Data Delay	30	-	60	-	-	-	-	-
TMG_R2	Row Address Hold	30	45	60	75	-	-	-	-
TMG_R3	Read, cycle time	60	-	90	-	-	-	-	-
TMG_R4	Read, CAS assertion delay	30	45	60	75	-	-	-	-
TMG_R5	Read CAS pulse width	30	45	60	75	-	-	-	-
TMG_R6	Read, Column address hold	30	45	60	75	-	-	-	-
TMG_W1	Write, data delay	30	-	60	-	-	-	-	-
TMG_W4	Write CAS assertion delay	60	75	90	105	120	135	-	-
TMG_PRE	RAS Pre-charge	0	30	-	-	-	-	-	-
TMG_V3	Wrie/Victim, cycle time	60	-	90	-	-	-	-	-
TMG_V4	Linked Victim, CAS Assertion delay	60	75	90	105	120	135	-	-
TMG_V5	Write/Victim, CAS pulse width	30	45	60	75	-	-	-	-
TMG_V6	Write/Victim Column address hold	30	45	60	75	-	-	-	-
TMG_RV	Delay between a Read and linked Victim	0	0	30	30	-	-	-	-
	Delay to MEM_EN assertion after the read	15	30	45	60				
TMG_RD_DLY	Read data delay info	0	15	30	45	-	-	-	-
a '-' indicates the value is illegal									

Table 7-43 Memory Timing Registers (TMG0-TMG2)

Name	Extent	Access	Init Stat
TMG_R1	<1:0>	RW	0
	read starting, data delay		
TMG_R2	<3:2>	RW	0
	row address hold		
TMG_R3	<5:4>	RW	0
	read, cycle time		
TMG_R4	<7:6>	RW	0
	read, CAS assertion delay		
TMG_R5	<9:8>	RW	0
	read, CAS pulse width		
TMG_R6	<11:10>	RW	0
	read, column address hold		
TMG_W1	<13:12>	RW	0
	write, data delay		
TMG_W4	<16:14>	RW	0
	write, CAS assertion delay		
TMG_PRE	<17>	RW	0
	RAS, pre-charge delay		
TMG_V3	<19:18>	RW	0
	write, cycle time		
TMG_V4	<22:20>	RW	0
	linked victim, CAS assertion delay		
reserved	<23>	RO	0
TMG_V5	<25:24>	RW	0
	victim/write, CAS pulse width		
TMG_V6	<27:26>	RW	0
	victim/write, column address hold		
TMG_RV	<29:28>	RW	0
	read-to-victim start delay		
TMG_RD_DLY	<31:30>	RW	0
	read, data delay (effects DSW)		

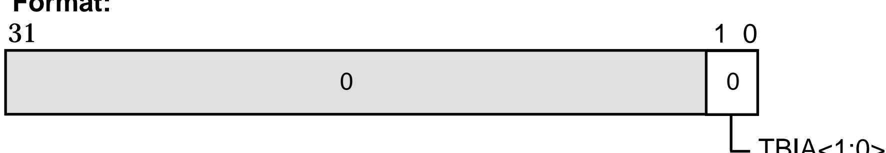
PCI Address-related Registers - Description

Scatter/Gather Translation Buffer Invalidate Register (TBIA)

Access:

Write, 87.6000.0100

Format:



Description:

A write to the TBIA register will result in the specified group of scatter gather TLB TAGs to be marked invalid and unlocked.

Table 7-44 Scatter/Gather Translation Buffer Invalidate Register (TBIA)

Name	Extent	Access	Init State
TBIA<1:0>	<1:0>	WO	0
	A write to this register will invalidate the scatter/gather Translation Buffers.		
reserved	<31:2>	RO	0

Bits <1:0>	Meaning
0 0	*no operation*
0 1	Invalidate and unlock the TLB TAGs that are currently locked
1 0	Invalidate the TLB TAGs that are currently unlocked
1 1	Invalidate and unlock all of the TLB TAGs entries

Window Base Registers (Wx_BASE, x=0-3)

Access:

Read/Write, 87.6000.0400, 87.6000.0500, 87.6000.0600, 87.6000.0700

Format:

31 — 20	19 — 4	3	2	1	0
X	0	X	X	X	X
W_BASE		DAC_enable[2]	MEMCS_enable	Wx_BASE_SG[1]	W_EN

Note: (1) W_BASE[0] only
(2) W_BASE(3) only

Description:

The Window Base register provides the base address for a particular target window. There are 4 Window Base registers: W_BASE[0], W_BASE[1], W_BASE[2], and W_BASE[3]. The W_BASE[x] registers should not be modified unless the software ensures that the no PCI traffic is targetted for the window being modified.

Determining a Hit in the Target Window:

The incoming PCI address bits <31:20> are compared with the each of the four Window Base registers, where the W_MASK register determines which bits are involved in the comparison.

The Target Window is 'Hit' when the masked addresses match a valid Window Base register.

If MEMCS_enable is set then the "Hit" is further qualified by the MEMCS input signal -- this is used if PC compatibility holes are required in the CIA (see Chapter 10, AlphaStation 600 PCI-EISA Bridge).

When DAC_enable is set in Window 3 then W_DAC base register is used to compare PCI address<39:32> of a DAC cycle.

Table 7-45 Window Base Registers (Wx_BASE, x=0-3)

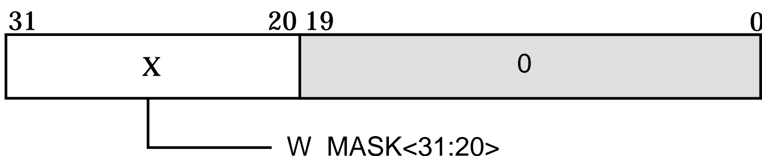
Name	Extent	Access	Init State
W_EN	<0>	RW	X
	When W_EN is cleared, the PCI Target Window is disabled and will not be used to respond to PCI initiated transfers. When W_EN is set, the PCI Target Window is enabled and will be used to respond to PCI initiated transfers that hit in the address range of the Target Window.		
Wx_BASE_SG	<1>	RW	X
	When the SG bit is cleared, the PCI Target Window uses direct mapping to translate a PCI address to a CPU address (Table 7-48) When the SG bit is set, the PCI Target Window uses scatter gather mapping to translate a PCI address to a physical memory address. (See Table 7-49)		
MEMCS_enable **Only in W0_BASE**	<2>	RW	X
	When the MEMCS_enable bit is set then the MEMCS signal from the PCEB (PCI-EISA bridge) is ANDed with the normal window hit.		
DAC_enable **Only in W3_BASE**	<3>	RW	X
	When the DAC_enable bit is set then the W_DAC register is compared against PCI address<39:32> for a PCI DAC cycle. If this compare hit, and the 32-bit portion of the PCI address hit, then a DAC cycle hit occurs.		
reserved	<19:4>	RO	0
W_BASE	<31:20>	RW	X
	W_BASE specifies the PCI base address of the PCI Target Window and is used to determine a hit in the target window. See MEMCS_enable and DAC_enable also.		

Window Mask Registers (Wx_MASK, x=0-3)

Access:

Read/Write, 87.6000.0440, 87.6000.0540, 87.6000.0640, 87.6000.0740

Format:



Description:

The Window Mask register provides a mask corresponding to bits <31:20> of an incoming PCI address. The size of each window can be programmed to be from 1 Mbyte to 4 Gbytes in powers of two by masking bits of the incoming PCI address via the Window Mask register as shown in Table 7-46

There are 4 Window Mask registers: W_MASK[0], W_MASK[1], W_MASK[2], and W_MASK[3]. The W_MASK[x] registers should not be modified unless software insures that the no PCI traffic is targetted for the window being modified.

Determining a Hit in the Target Window:

The incoming PCI address bits <31:20> are compared with the each of the four Window Base registers; where the W_MASK register determines which bits are involved in the comparison. The Target Window is 'Hit' when the masked addresses match a valid Window Base register.

Table 7-46 Window Mask Registers (Wx_MASK, x=0-3)

Name	Extent,	Access,	Init State,
reserved	<19:0>	RO	0
W_MASK<31:20>	<31:20>	RW	X
	W_MASK specifies the size of the PCI Target Window (see table below) and it is also used to mask out address bits not used when determining a PCI Target Window "hit".		

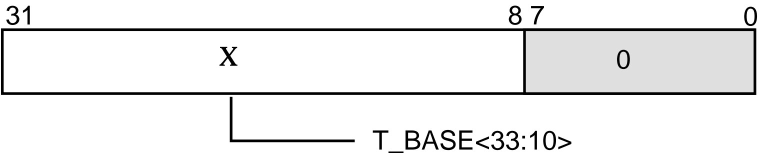
W_MASK<31:20>	Size of Window
0000 0000 0000	1 Megabyte
0000 0000 0001	2 Megabytes
0000 0000 0011	4 Megabytes
0000 0000 0111	8 Megabytes
0000 0000 1111	16 Megabytes
0000 0001 1111	32 Megabytes
0000 0011 1111	64 Megabytes
0000 0111 1111	128 Megabytes
0000 1111 1111	256 Megabytes
0001 1111 1111	512 Megabytes
0011 1111 1111	1 Gigabyte
0111 1111 1111	2 Gigabytes
1111 1111 1111	4 Gigabytes
otherwise	Not supported

Translated Base Registers (Tx_BASE, x=0-3)

Access:

Read/Write, , 87.6000.0480, 87.6000.0580, 87.6000.0680, 87.6000.0780

Format:



Description:

The Translated Base register is used to map PCI addresses into memory. There are 4 Translated Base registers: TBR0, TBR1, TBR2, and TBR3, one for each window. If the Scatter/Gather bit of the Window Base Register is set, then the Translated Base register provides the base address of the Scatter/Gather map for this window. If the Scatter/Gather bit is clear, the Translated Base register provides the base physical address of this window.

The TBR registers should not be modified unless the software ensures that no PCI traffic is targetted for the window being modified.

Table 7-47 Translated Base Registers (Tx_BASE, x=0-3)

Name	Extent	Access	Init State
reserved	<7:0>	RO	0
T_BASE<33:10>	<31:8>	RW	X
	If Scatter/Gather Mapping is disabled, T_BASE<33:10> specifies the base CPU address of the translated PCI address for the PCI Target Window (refer to Table 7-48). If Scatter/Gather Mapping is enabled, T_BASE<33:10> specifies the base CPU address for the Scatter/Gather Map Table for the PCI Target Window (refer to Table 7-49).		

The field W_MASK<31:20> sets the size of the PCI Target window and the number of 8K pages that fall into the window. Every 8K page requires one 8 byte scatter/gather map entry. Table 7-49 shows the relationship of W_MASK to the size of the scatter gather map in memory.

(size of the window in bytes) / (8Kbytes) = number of entries required

(number of entries) * (8 bytes) = size of the scatter/gather table

The quadword address used to index into the table is formed from concatenating the appropriate T_BASE and PCI address bits based on the size of the scatter/gather map. The PCI address forms the index into the table while the T_BASE forms the naturally aligned base of the table.

For example, for a mask of 0000 0000 0000, there are 128 entries in the scatter/gather table and the table size is 1 Kbyte. Entries are quadwords so the lower three bits <2:0> of the address are always zero. Now, mask off PCI bits <31:20> (because of the W_MASK). Then use PCI address<19:13> (7 bits, 2 to the power 7 = 128 entries in the table) as the table index. Use the T_BASE<33:10> to get the other bits of the 34 bit address.

Table 7-48 PCI Address Translation - Scatter/Gather Mapping Disabled

W_MASK<31:20>	Translated Address<33:0>	Unused Translated Base Register Bits
0000 0000 0000	T_BASE<33:20>:PCI_AD_H<19:0>	T_BASE<19:10>
0000 0000 0001	T_BASE<33:21>:PCI_AD_H<20:0>	T_BASE<20:10>
0000 0000 0011	T_BASE<33:22>:PCI_AD_H<21:0>	T_BASE<21:10>
0000 0000 0111	T_BASE<33:23>:PCI_AD_H<22:0>	T_BASE<22:10>
0000 0000 1111	T_BASE<33:24>:PCI_AD_H<23:0>	T_BASE<23:10>
0000 0001 1111	T_BASE<33:25>:PCI_AD_H<24:0>	T_BASE<24:10>
0000 0011 1111	T_BASE<33:26>:PCI_AD_H<25:0>	T_BASE<25:10>
0000 0111 1111	T_BASE<33:27>:PCI_AD_H<26:0>	T_BASE<26:10>
0000 1111 1111	T_BASE<33:28>:PCI_AD_H<27:0>	T_BASE<27:10>
0001 1111 1111	T_BASE<33:29>:PCI_AD_H<28:0>	T_BASE<28:10>
0011 1111 1111	T_BASE<33:30>:PCI_AD_H<29:0>	T_BASE<29:10>
0111 1111 1111	T_BASE<33:31>:PCI_AD_H<30:0>	T_BASE<30:10>
1111 1111 1111	T_BASE<33:32>:PCI_AD_H<31:0>	T_BASE<31:10>

Note: (1) unused Translation Base must be zero for correct operation.
(2) The AlphaStation 600 system is restricted to 6 GB and thus T_BASE<33> = 0.

Table 7-49 PCI Address Translation - Scatter/Gather Mapping Enabled

W_MASK<31:20	SG Map Table Size	Scatter Gather Map Address<33:0> (used to index S/G Table in memory)
0000 0000 0000	1 Kbytes	T_BASE<33:10>:PCI_AD_H<19:13>:000
0000 0000 0001	2 Kbyte	T_BASE<33:11>:PCI_AD_H<20:13>:000
0000 0000 0011	4 Kbytes	T_BASE<33:12>:PCI_AD_H<21:13>:000
0000 0000 0111	8 Kbytes	T_BASE<33:13>:PCI_AD_H<22:13>:000
0000 0000 1111	16 Kbytes	T_BASE<33:14>:PCI_AD_H<23:13>:000
0000 0001 1111	32 Kbytes	T_BASE<33:15>:PCI_AD_H<24:13>:000
0000 0011 1111	64 Kbytes	T_BASE<33:16>:PCI_AD_H<25:13>:000
0000 0111 1111	128 Kbytes	T_BASE<33:17>:PCI_AD_H<26:13>:000
0000 1111 1111	256 Kbytes	T_BASE<33:18>:PCI_AD_H<27:13>:000
0001 1111 1111	512 Kbytes	T_BASE<33:19>:PCI_AD_H<28:13>:000
0011 1111 1111	1 Mbyte	T_BASE<33:20>:PCI_AD_H<29:13>:000
0111 1111 1111	2 Mbyte	T_BASE<33:21>:PCI_AD_H<30:13>:000
1111 1111 1111	4 Mbyte	T_BASE<33:22>:PCI_AD_H<31:13>:000

Window DAC Base Register (W_DAC)

Access:

Read/Write, 87.6000.07C0

Format:

Description:

The Window DAC Base register provides the <7:0> address bits for comparison against PCI address <39:32> during a DAC cycle. PCI address <63:40> has to be zero for a PCI window hit. The DAC BASE register is used in conjunction with the W_BASE register. For more details refer to Chapter 3, AlphaStation 600 Addressing .

The DAC BASE register is only applicable to window 3 and only if enabled by the DAC_enable bit in W_BASE[3].

Determining a Hit in the Target Window:

The Target Window is a ’Hit’ when the following is satisfied:

- The incoming PCI address bits <31:20> matches one of the four Window Base registers; the W_MASK register determines which bits are involved in the comparison.
- PCI address <63:40> is zero.
- PCI address<39:32> match the DAC_BASE<7:0>.

Table 7-50 Window DAC Base Register (W_DAC)

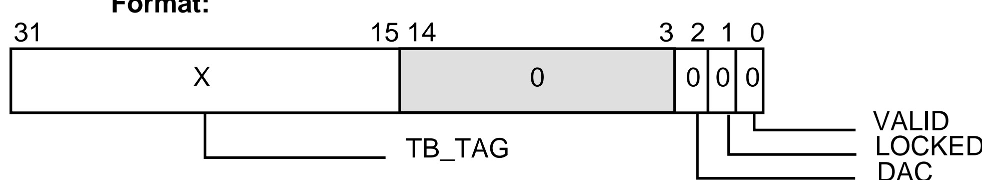
Name	Extent	Access	Init State
DAC_BASE<7:0>	<7:0>	RW	X
	DAC_BASE specifies bits <39:32> of the PCI base address used to determine a hit in the target window for a DAC cycle.		
reserved	<31:8>	RO	0

Lockable Translation Buffer Tag Registers (LTB_TAG0 - LTB_TAG3)

Access:

Read/Write, 87.6000.0800, 87.6000.0840, 87.6000.0880, 87.6000.08C0

Format:



Description:

There are four lockable translation buffer tag registers. Software can write to these LTB_TAG entries. Furthermore, they can be *locked* such that the hardware will not evict the entry on a SG_TLB miss.

Determining a Hit in the Translation Buffer:

After a PCI address hits one of the window registers with SG true, the incoming PCI address bits <31:15> are compared with the each of the eight Translation Buffer Tag Registers. If there is a match, the corresponding Translation Buffer Page register group is indexed by PCI address bits <14:13>. If the page entry is valid then there is a Translation Buffer hit.

Operation on a SG_TLB miss:

A scatter/gather TLB miss is handled by hardware using a round-robin algorithm. An entry is overwritten if it is not locked. The hardware will write all four PTEs on a miss.

Table 7-51 Lockable Translation Buffer Tag Registers (LTB_TAG0 - LTB_TAG3)

Name	Extent	Access	Init State
VALID	<0>	RW	0
	If VALID and CIA_CTLR[SG_TLB_EN] are set, then this entry will be used for address translation.		
LOCKED	<1>	RW	0
	If LOCKED is set the hardware will never evict this entry.		
DAC	<2>	RW	0
	If set then this TAG entry corresponds to a 64-bit PCI address (DAC cycle); otherwise it belongs to a 32-bit PCI address (SAC cycle)		
reserved	<14:3>	RO	0
TB_TAG	<31:15>	RW	X
	TB_TAG<31:15> is the TAG for each translation buffer entry		

Translation Buffer Tag Registers (TB_TAG0 - TB_TAG3)

Access:

Read/Write, 87.6000.0900, 87.6000.09040, 87.6000.0980, 87.6000.09C0

Format:

Description:

There are four Translation Buffer Tag registers which cannot be locked down by the software. Software can write to the TB TAG entries, but they cannot be locked down (and hence may be evicted by the hardware on a SG TLB miss).

Determining a Hit in the Translation Buffer:

The incoming PCI address bits <31:15> are compared with the each of the eight Translation Buffer Tag registers. If there is a match, the corresponding Translation Buffer Page register group is indexed by PCI address bits <14:13>, and if it is valid then there is a Translation Buffer Hit.

Operation on a SG_TLB miss:

A scatter/gather TLB miss is handled by hardware using a round-robin algorithm. An entry is overwritten if it is not locked. The hardware will write all four PTEs on a miss.

Table 7-52 Translation Buffer TAG Registers (TB_TAG0 - TB_TAG3)

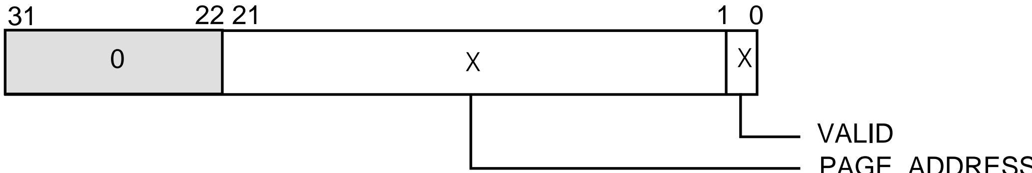
Name	Extent	Access	Init State
VALID	<0>	RW	0
	If VALID and CIA_CTLR[SG_TLB_EN] are set, then this entry will be used for address translation.		
reserved	<1>	RW	0
DAC	<2>	RW	0
	If set then this TAG entry corresponds to a 64-bit PCI address (DAC cycle); otherwise it belongs to a 32-bit PCI address (SAC cycle)		
reserved	<14:3>	RO	0
TB_TAG	<31:15>	RW	X
	TB_TAG<31:15> is the TAG for each translation buffer entry		

Translation Buffer Page Register (TBx_PAGEn)

Access:

Read/Write, 87.6000.1000 thru 87.6000.17C0

Format:



Description:

There are thirty two Translation Buffer Data registers, a group of four for each of the eight translation buffer entries. The TBx_PAGEn registers are automatically updated on a TLB miss (a group of four at a time) by the CIA hardware.

Table 7-53 Translation Buffer Data Register (TBx_PAGEn)

Name	Extent	Access	Init State
VALID	<0>	RW	X
PAGE_ADDRESS	<21:1>	RW	X
	The PAGE_ADDRESS<21:1> forms Physical address<33:13>. PCI_AD_H<12:0> forms physical address<12:0>. For AlphaStation 600 address<33> must be zero.		
reserved	<31:22>	RO	0

Determining a Hit in the Translation Buffer:

The incoming PCI address bits <31:15> are compared with the each of the eight Translation Buffer Tag Registers. If there is a match, the corresponding Translation Buffer Page register group is index by PCI address bits <14:13> and if it is valid then there is a Translation Buffer Hit.

If the Address bits do not match the TAG, or the page entry is invalid then a TLB miss occurs. If the PTE fetched by the hardware TLB-miss handler is still invalid then the CIA_ERROR interrupt is asserted for a DMA write -- see Chapter 8, Hardware Exceptions and Interrupts.

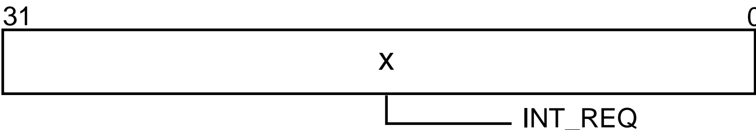
GRU ASIC - related Registers

Interrupt Request Register (INT_REQ)

Access:

Read only, 87.8000.0000

Format:



Description:

This register is used to read the interrupt request lines from the main interrupt logic (in the GRU ASIC). If a bit is set, then it signifies that an interrupt is active.

Table 7-54 INT_REQ Register

Name	Extent	Access	Init State
INT_REQ	<31:0>	RO	Undefined
	0: no Interrupt asserted 1: Interrupt asserted		

Table 7-55 Main Interrupt Logic IRQ Assignment

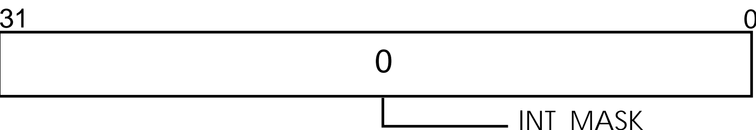
Output	IRQ	Interrupt	Usage (reference designator)
EV5 IRQ<1>	0 1 2 3	int A int B int C int D	PCI Slot 2 -- 64-bit (J11)
	4 5 6 7	int A int B int C int D	PCI Slot 1 -- 64 bit (J10)
	8 9 10 11	int A int B int C int D	PCI Slot 0 -- 64 bit (J9)
	12 13 14 15	int A int B int C int D	PCI Slot 4 -- 32 bit (J8)
	16 17 18 19	int A int B int C int D	PCI Slot 3 -- 32 bit (shared PCI/EISA slot) (J7)
	20-30		reserved
	31	EISA_INT	8259 INT output
All unmasked IRQ inputs have equal priority			

Interrupt Mask Register (INT_MASK)

Access:

Read/Write 87.8000.0040

Format:



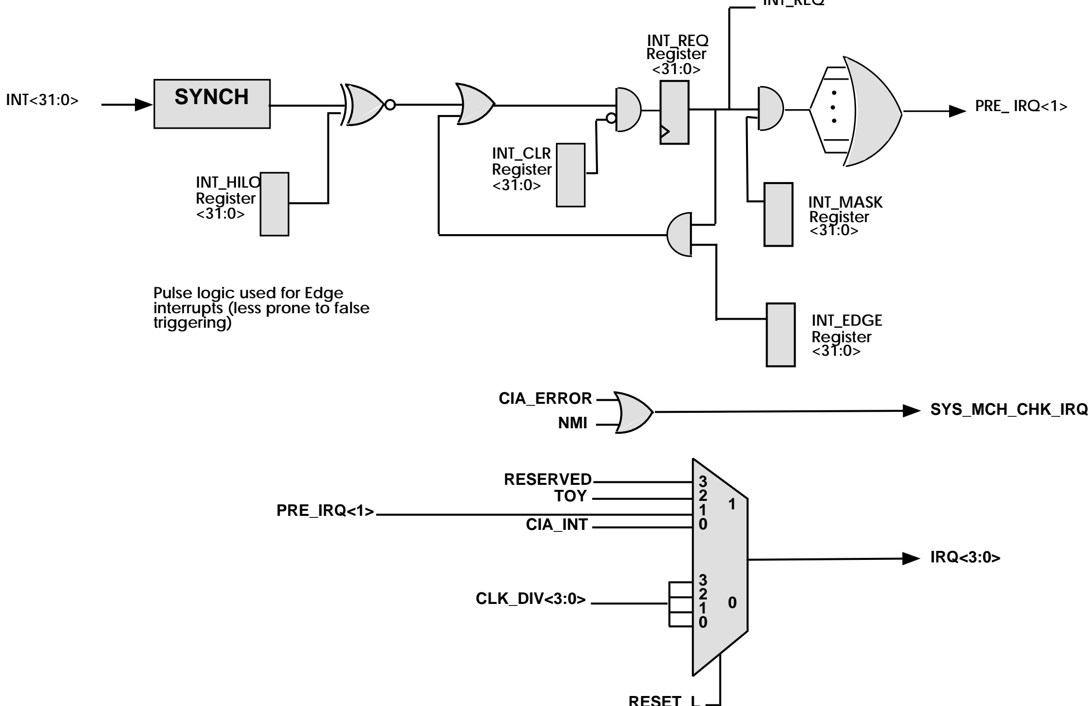
Description:

This register is used to access the interrupt mask register which is physically located in the main interrupt logic (in the GRU ASIC). The main interrupt logic has 32 inputs which are all individually maskable.

Table 7-56 INT_MASK Register

Name	Extent	Access	Init State
INT_MASK	<31:0>	RW	0
	0: Interrupt IRQ input disabled 1: Interrupt IRQ input enabled		

Figure 7-1 GRU Interrupt Logic

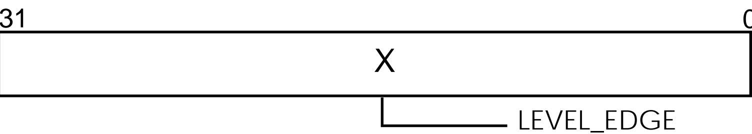


Interrupt Level/Edge Select Register (INT_EDGE)

Access:

Read/write, 87.8000.0080

Format:



Description:

This register is used to control the main interrupt logic (in the GRU asic). If a bit is set, then it signifies that the IRQ line is an edge-type (actually a pulse); otherwise it is a level-type (that is, standard PCI).

The AlphaStation 600 system requires that all irq lines be set to the level-sensitive mode. The edge-mode is not used.

Table 7-57 INT_EDGE Register

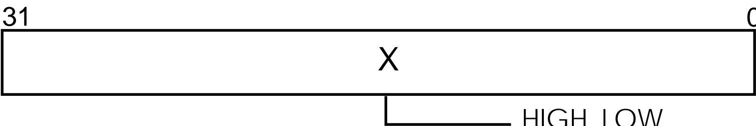
Name	Extent	Access	Init State
LEVEL_EDGE	<31:0>	R/W	undefined
	0: level-sensitive IRQ (PCI type) 1: edge-sensitive IRQ (ISA type)		

Interrupt High/Low select Register (INT_HILO)

Access:

Read/write, 87.8000.00C0

Format:



Description:

This register is used to control the main interrupt logic (in the GRU ASIC). If a bit is set, then it signifies that the IRQ line is active high; otherwise it is active low (that is, standard PCI).

The AlphaStation 600 system requires that all IRQ lines be set to active low, except for bit <31> (PCI_EISA bridge interrupt) which is active high.

Table 7-58 INT_HILO Register

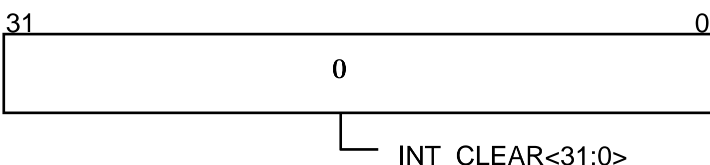
Name	Extent	Access	Init State
HIGH_LOW	<31:0>	RW	Undefined
	0: Active low interrupt (PCI type) 1: Active High interrupt (that is, PCI-EISA bridge interrupt)		

Interrupt Clear Register (INT_CLEAR)

Access:

Read/Write, 87.8000.0100

Format:



Description:

This register is used to access the interrupt mask register which is physically located in the main interrupt logic (in the GRU ASIC). The main interrupt logic has 32 inputs which are all individually cleared.

NOTE: If INT_CLEAR = 1 and an interrupt occurrs, the interrupt will be masked.

Table 7-59 INT_CLEAR Register

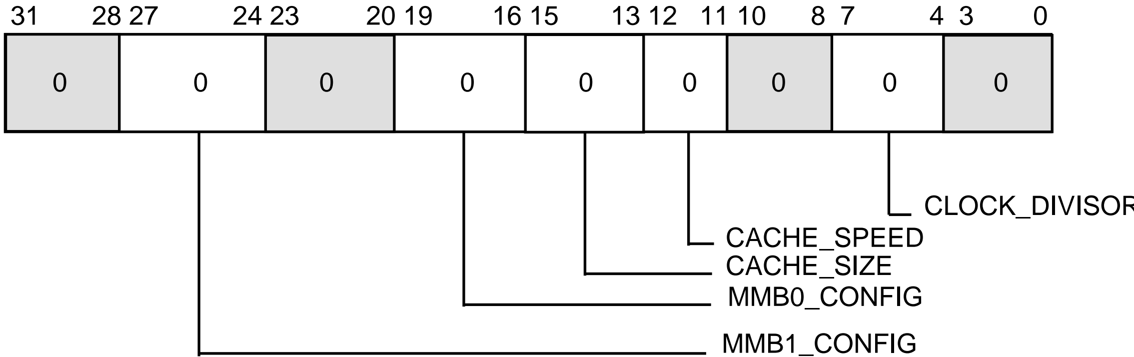
Name	Extent	Access	Init State
INT_CLEAR	<31:0>	RW	0
	0: do not clear interrupt 1: clear interrupt		

Cache & Memory Configuration Register (CACHE_CNFG)

Access:

RO, 87.8000.0200

Format:



Description:

The Cache Configuration Register (CACHE_CNFG) contains the size and speed information for each individual cache SIMM. The information for each SIMM is derived by hardware interrogating presence detect pins. Information from this register is used by firmware to configure the BCACHE

Table 7-60 MMB and Cache Configuration Register (CACHE_CNFG)

Name	Extent	Access	Init State
reserved	<3:0>	RO	0
CLOCK_DIVISOR	<7:4>	RO	0
	The clock divisor is driven onto the IRQ lines during reset to program the EV5 system clock ratio.		
reserved	<10:8>	RO	0
CACHE_SPEED	<12:11>	RO	0
	CACHE_SPEED specifies the access time for the SRAMs on the lowest-order Cache SIMM. See Table 7-61.		
CACHE_SIZE	<15:13>	RO	0
	CACHE_SIZE specifies the size of the SRAMs on the Cache SIMM. See Table 7-62.		
MMB0_CNFG	<19:16>	RO	0
	MMB0_CNFG indicates if MMB0 is present and what type of MMB it is. See Table 7-63.		
reserved	<23:20>	RO	0
MMB1_CNFG	<27:24>	RO	0
	MMB1_CNFG indicates if MMB0 is present and what type of MMB it is. See Table 7-63.		
reserved	<31:28>	RO	0

Table 7-61 Cache Speed

Sx_Speed <1:0>	Cache RAM speed
00	8ns
01	10ns
10	12ns
11	15ns

Table 7-62 Cache Size

Sx_SIZE <2:0>	Cache RAM Size
000	No cache present
001	reserved
010	128KxX - 2 MByte complete cache
011	256KxX - 4 MByte complete cache
100-111	reserved

Table 7-63 MMB Configuration

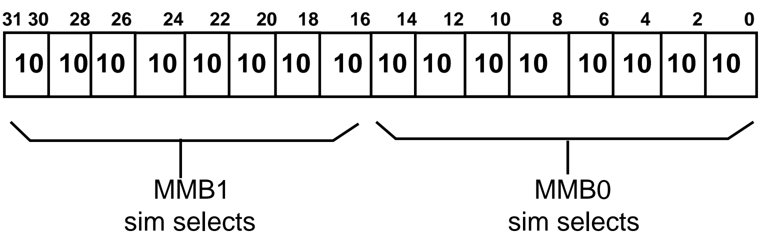
MMBx_CNFG <3:0>	Description
0000	MMB **NOT** present
0001	MMB is present and contains upto 24 SIMMs (12 sets)
0010	MMB is present and contains upto 8 SIMMs (4 sets)
0011 - 1111	reserved

SET Configuration Register (SCR)

Access:

RO, 87.8000.0300

Format:



Description:

The SCR register contains all the access rate (speed) information for each individual SIMM. The information for each set is derived by hardware interrogating individual SIMM's PD3 and PD4 pins. SSx_MMB0/1, as defined in the following tables, specifies the speed of the SIMM. These bits are mapped from the PD3 and PD4 bits coming from this particular SIMM. Information from this register is used by the system designer or the firmware to configure the MCIx and MCR registers.

NOTES: Sets 4, 5, 6 and 7 are not populated on the AlphaStation 600 system.
100ns SIMMs are not supported on the AlphaStation 600 system.

Table 7-64 SIMM PD Speed Select Pins

PD <1:0>	SIMM Speed
00	not defined
01	80 ns
10	70 ns
11	60 ns

Table 7-65 SET Configuration Register (SCR)

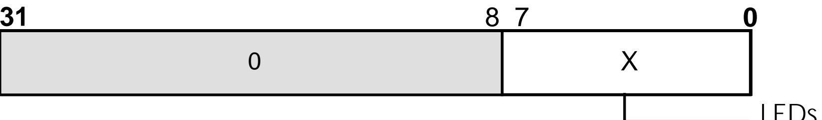
Name	Extent	Access	Init State
SS7_MMB0	<1:0>	RO	10
SS6_MMB0	<3:2>	RO	10
SS5_MMB0	<5:4>	RO	10
SS4_MMB0	<7:6>	RO	10
SS3_MMB0	<9:8>	RO	10
SS2_MMB0	<11:10>	RO	10
SS1_MMB0	<13:12>	RO	10
SS0_MMB0	<15:14>	RO	10
SS7_MMB1	<17:16>	RO	10
SS6_MMB1	<19:18>	RO	10
SS5_MMB1	<19:18>	RO	10
SS4_MMB1	<23:22>	RO	10
SS3_MMB1	<25:24>	RO	10
SS2_MMB1	<27:26>	RO	10
SS1_MMB1	<29:28>	RO	10
SS0_MMB1	<31:30>	RO	10

LED Register (LED)

Access:

Write only, 87.8000.0800

Format:



Description:

This register is used to write to the front panel LEDs. This is not used by current AlphaStation 600 systems. The contents of the LEDs CSR is driven onto the GRU_DAT bus when the Flash ROMs are tri-stated off the bus.

Table 7-66 LED Register

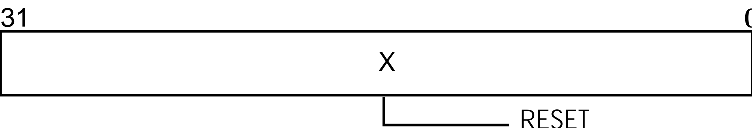
Name	Extent	Access	Init State
LEDs	<7:0>	WO	X
reserved	<31:0>	RO	0

Reset Register (RESET)

Access:

Write only, 87.8000.0900

Format:



Description:

This register is used to reset the AlphaStation 600 system under software control. Writing the value 0000DEAD$_{hex}$ will cause a complete system reset.

Table 7-67 RESET Register

Name	Extent	Access	Init State
RESET	<31:0>	Write only	X
	Writing "0000DEAD" to this register will induce a system reset.		

EV5 Configuration Registers - description

Scache Control Register, SC_CTL

SC_CTL is a read/write register which controls the behavior of the Scache. The only bit of interest for the AlphaStation 600 system initialization is SC_BLK_SIZE. The Scache and Bcache must use the same block size -- the AlphaStation 600 system uses a 64-byte block for the Bcache.

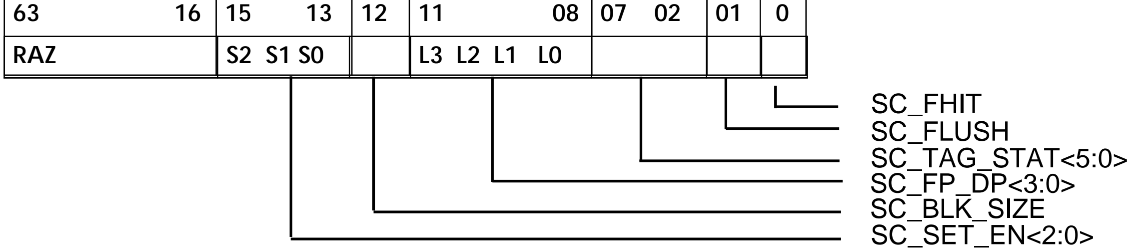


Table 7-68 SC_CTL Field Descriptions

Name	Extent	Type	Init State
SC_FHIT	<0>	RW	0
SC_FLUSH	<1>	RW	0
SC_TAG_STAT<5:0>	<2>	RW	---
SC_FP_DP<3:0>	<3>	RW	---
SC_BLK_SIZE	<12>	RW	1
	AlphaStation 600 and its family of systems will have a 64 Byte block size so this bit must remain set		
SC_SET_EN<2:0>	<15:13>	RW	1
reserved	<63:16>		

Bcache Control Register, BC_CONTROL

BC_CONTROL is a write only register which controls the behavior of the Bcache.

Format:

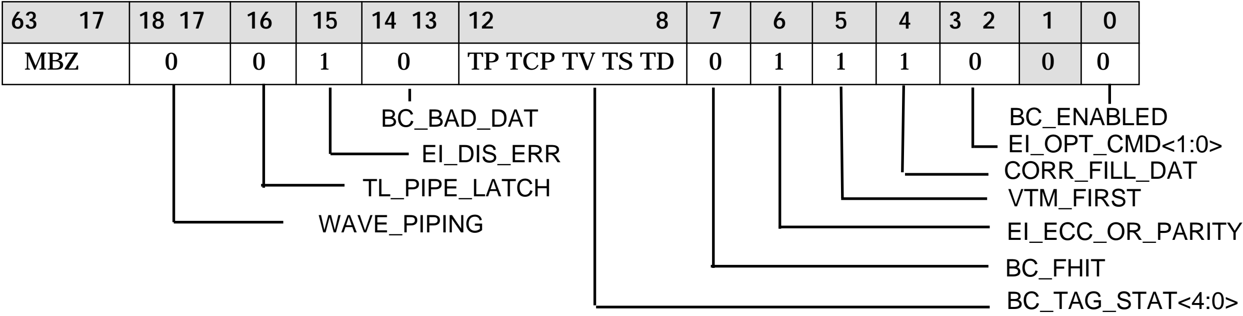


Table 7-69 BC_CONTROL Field Descriptions

Name	Extent,	Type,	Init State
BC_ENABLED	<0>	W	0 When set, the external Bcache is enabled. When clear, the Bcache is disabled. When the Bcache is disabled, the BIU will neither do external cache reads nor writes. This bit will be cleared on reset. **The AlphaStation 600 system will all have a Bcache so this bit must be set before the DECchip 21164-AA does the first cacheable read or write from Bcache or system.** The AlphaStation 600 system will partially function with the Bcache disabled. The AlphaStation 600 system does not check DMA addresses against Write Block commands in the CIA's CPU queue. So, if the Bcache is disabled, there is a potential for a DMA to get stale data from memory if the block being addressed is in the CIA's queue.
reserved	<1>	W	0 **Must be zero.**
EI_OPT_CMD0	<2>	W	0 When set, the optional commands, LOCK, and SET DIRTY, will be driven to the DECchip 21164-AA external interface command pins to be acknowledged by the system interface. When clear, these commands will be internally acknowledged by DECchip 21164-AA and will not be driven off-chip to the system interface. This bit will be cleared on reset. **The AlphaStation 600 system does not use Lock, or SET DIRTY so this bit must be clear before the DECchip 21164-AA does the first cacheable read or write from Bcache or system.**
EI_OPT_CMD1	<3>	W	0 When set, the optional command, MEMORY BARRIER will be driven to the DECchip 21164-AA external interface command pins to be acknowledged by the system interface. When clear, this command will be internally acknowledged by DECchip 21164-AA and will not be driven off-chip to the system interface. This bit will be cleared on reset. **MBs to the system appear not to be required so this bit can remain cleared.**

Table 7-69 BC_CONTROL Field Descriptions (continued)

Name	Extent	Type	Init State
CORR_FILL_DAT	<4>	W	1
	Correct fill data from Bcache or memory, in ECC mode. When this bit is set, fill data from Bcache or memory will first go through error correction logic before being driven to the Scache or Dcache. If the error is correctable, it will be transparent to the machine. When this bit is clear, fill data from Bcache or memory will be directly driven to the Dcache before ECC error is detected. If the error is correctable, corrected data will be returned again, Dcache will be invalidated, and error trap will be taken. This bit will be set on reset. **This is a performance issue (one saves some fill latency). The conservative strategy would be to leave this bit set (especially since the AlphaStation 600 system does not correct fill data).**		
VTM_FIRST	<5>	W	1
	Set for systems without a victim buffer. On a Bcache miss, DECchip 21164-AA will first drive out the victimized block's address on the system address bus, followed by the read miss address and command. CIeared for systems with a victim buffer. If clear, on a Bcache miss with victim, DECchip 21164-AA will first drive out the read miss followed by the victim address and command. This bit will be set on reset. **The AlphaStation 600 system does use a Victim buffer so this bit must be clear for maximum performance benefit.**		
EI_ECC_OR_PARITY	<6>	W	1
	This bit determines whether to operate the external interface in QW ECC or Byte parity mode. When set, DECchip 21164-AA generates/expects QW ECC on the data check pins. When clear, DECchip 21164-AA generates/expects even byte parity on the data check pins. This bit will be set on reset. **The AlphaStation 600 system provides ECC only so this bit must be set before the DECchip 21164-AA does the first cacheable read or write from Bcache or system.**		
BC_FHIT	<7>	W	0
	Bcache force hit. When this bit is set and the Bcache is enabled, all external references in cached space are forced to hit in the Bcache, irrespective of the tag status bits. BC_FHIT bit will be cleared on reset. Software should turn off BC_CONTROL<2> to allow clean to dirty transitions without going to the System This is a diagnostic/self-test feature that is used the same on all systems.		
BC_TAG_STAT	<12:8>	W	?
	This bit field can be only used in BC-FHIT mode to write any combination of tag status and parity bits in the Bcache. The parity bit can be used to write bad tag parity. These bits will be undefined on reset. See Table 7-70 for the encodings. This is a diagnostic/self-test feature that is used the same on all systems.		

Table 7-69 BC_CONTROL Field Descriptions (continued)

<table>
<tr><th>Name</th><th>Extent</th><th>Type</th><th>Init State</th></tr>
<tr><td rowspan="2">BC_BAD_DAT</td><td><14:13></td><td>W</td><td>0</td></tr>
<tr><td colspan="3">When set, this field can be used to write bad data with correctable or uncorrectable error in ECC mode. When bit <13> is set, data bit <0> and <64> are inverted. When bit <14> is set, data bit <1> and <65> are inverted. When the same QW is read from the Bcache, DECchip 21164-AA will detect correctable/ uncorrectable ECC error on both the QWs based on the value of bits <14:13> used when writing. This field will be cleared on reset. This is a diagnostic/self-test feature that is used the same on all systems.</td></tr>
<tr><td rowspan="2">EI_DIS_ERR</td><td><15></td><td>W</td><td>1</td></tr>
<tr><td colspan="3">When set, this bit causes the DECchip 21164-AA to ignore any ECC or parity error on a fill data received from the Bcache or memory and no machine check is taken. It will also ignore any Bcache tag or control parity error. This bit will be set on reset.

The AlphaStation 600 system requires that Bcache and Memory be intialized with good ECC before checking is enabled. So, this bit should not be cleared until after the Bcache and Memory are intialized</td></tr>
<tr><td rowspan="2">TL_PIPE_LATCH</td><td><16></td><td>W</td><td>0</td></tr>
<tr><td colspan="3">When set, this bit causes DECchip 21164-AA to pipe the system control pins (ADDR_BUS_REQ_H, CACK_H, and DACK_H) for one cpu cycle. This bit will be cleared on reset.

The AlphaStation 600 system does not take advantage of this feature so this bit should remain cleared before the DECchip 21164-AA does the first cacheable read or write from Bcache or system.</td></tr>
<tr><td rowspan="2">WAVE_PIPING</td><td><18:17></td><td>W</td><td>0</td></tr>
<tr><td colspan="3">This field is used to indicate to the BIU the number of cycles of wave pipelining that should be used during private reads of the Bcache. Wave piping can be up to four CPU cycles. On power-up, this field will be initialized to a value of zero CPU cycles.

The AlphaStation 600 system takes advantage of this feature and will allow up to M nanoseconds of Wave Piping. This must be converted to cpu cycles. Thus for the AlphaStation 600 system, the following algorithm should be used:-

WAVE_PIPING = M/(CPU cycle time) --- (truncate to integer)

M = minimun cache looptime.</td></tr>
</table>

Table 7-70 BC_TAG_STAT Field Descriptions

Bcache Tag Status<12:8>	Description
BC_TAG_STAT<12>	Parity for Bcache tag
BC_TAG_STAT<11>	Parity for Bcache tag status bits
BC_TAG_STAT<10>	Bcache tag valid bit
BC_TAG_STAT<9>	Bcache tag shared bit
BC_TAG_STAT<8>	Bcache tag dirty bit

Bcache Configuration Register, (BC_CONFIG)

(The Bcache configuration register is write only.)

Table 7-71 BC_CONFIG Field Descriptions

Name	Extent,	Type,	Init State	Description
BC_SIZE	<2:0>	W	1	This field is used to indicate the size of the Bcache. On power-up, this field will be initialized to a value of 1 MByte Bcache. See table 7-72 for the encodings. **The AlphaStation 600 system can have any allowable Bcache size. This is determined by firmware analyzing the Bcache presence detect register. This field should be set up before the Bcache and Memory are intialized**
reserved	<3>	W	0	Must be zero
BC_RD_SPD	<7:4>	W	4	This field is used to indicate to the BIU the read access time of the Bcache, measured in CPU cycles, from the start of a read until data is valid at the input pins. The Bcache read speed must be within four to ten CPU cycles. On power-up, this field will be initialized to a value of four CPU cycles. For systems without a Bcache, the read speed must be equal to SYS clock to CPU clock ratio. **Teh AlphaStation 600 system can have Bcache RAM speeds of 8ns, 10ns, 12ns or 15ns. This is also determined by examining the Bcache presence detect register. See the Firmware chapter. BC_RD_SPD should be set to:-** **BC_RD_SPEED = (10 + SRAM Speed)/(CPU Cycle time)** **(Calculate with real numbers and round-up result to integer value. Instead of "real" numbers can scale -- for example, instead of 22/3.3ns use 220/33).** **This field should be set up before the Bcache and Memory are intialized.**
BC_WR_SPD	<11:8>	W	4	This field is used to indicate to the BIU the write time of the Bcache, measured in CPU cycles. The Bcache write speed must be within four to ten CPU cycles. On power up, this field will be initialized to a value of four CPU cycles. For systems without a Bcache, the write speed must be equal to SYS clock to CPU clock ratio. **The AlphaStation 600 system can have Bcaches RAM speeds of 8ns, 10ns, 12ns or 15ns. This is also determined by examining the Bcache presence detect register. See the Firmware chapter. BC_WR_SPD should be set to:-** **BC_WR_SPEED = (10 + SRAM Speed)/(CPU Cycle time)** **(Calculate with real numbers and round-up result to integer value)** **This field should be set up before the Bcache and Memory are intialized**

Table 7-71 BC_CONFIG Field Descriptions (continued)

Name	Extent	Type	Init State
BC_RD_WR_SPC	<14:12>	W	1
	This field is used to indicate to the BIU the number of CPU cycles to wait when switching from a private read to a private write Bcache transaction. For other data movement commands, such as Read Dirty or Fill from memory, it is up to the system to direct system wide data movement in a way that is safe. On power- up, this field will be initialized to a read/write spacing of one CPU cycle. **The AlphaStation 600 system needs a minimum of 8 ns to turn off the SRAM data before it can be driven by the CPU for a write. Thus BC_RD_WR_SPD should be set to:-** **BC_RD_WR_SPEED = 8/(CPU Cycle time)** **(Calculate with real numbers and round-up result to integer value)** **This field should be set up the DECchip 21164-AA does the first cacheable write to Bcache or system.**		
reserved	<15>	W	0 Must be zero
FILL_WE_OFFSET	<18:16>	W	1
	Bcache write enable pulse offset, from the Sysclock edge, for fills from the system. This field does not affect private writes to Bcache. It is used during fills from the system, when writing the Bcache to determine the number of CPU cycles to wait before driving out the write pulse value as programmed in the BC_WE-CTL field. This field is programmed with a value in the range of one to seven CPU cycles. It must never exceed the sysclock ratio. (for example, if the sysclock ratio is 3, this field must not be larger than 3.) On power-up, this field is initialized to a write offset value of one CPU cycle. **The AlphaStation 600 system needs a minimum of 9 ns to ensure the address is at the Bcache SRAMs before the write enable can be driven by the CPU for a write. Thus FILL_WE_OFFSET should be set to:** **FILL_WE_OFFSET = 9/(CPU Cycle time+1)** **(Calculate with real numbers and round-up result to integer value)** **This field should be set up the DECchip 21164-AA does the first cacheable read from the system.**		
reserved	<19>	W	0 Must be zero

Table 7-71 BC_CONFIG Field Descriptions (continued)

Name	Extent,	Type	Init State	
BC_WE_CTL	<28:20>	W	0	Bcache write enable control. This field is used to control the timing of the write enable during write or fill. If the bit is set the write pulse is asserted. If the bit is clear the write pulse is not asserted. Each bit corresponds to a CPU cycle. At the start of a Bcache write cycle, the write pulse will always be de -asserted for one CPU cycle. After the first cycle, bit <20> of the register is used to assert the write pulse. Each cycle, the next bit will be used to assert the write pulse. On power-up, all bits in this field will be cleared. **The AlphaStation 600 system can have Bcaches RAMS of 8ns, 10ns, 12ns or 15ns.. This is also determined by examining the Bcache presence detect register. See Firmware chapter. The SRAM speed determines the width of the write enable. So this field can be determined as follows:-** **First asserted bit of BC_WE_CTL = 20 + 7/(CPU Cycle time)** **For 8ns SRAMs** **Final asserted bit of BC_WE_CTL = First + 4/(CPU Cycle time)** **For 10ns SRAMs** **Final asserted bit of BC_WE_CTL = First + 5/(CPU Cycle time)** **For 12ns SRAMs** **Final asserted bit of BC_WE_CTL = First + 6/(CPU Cycle time)** **For 15ns SRAMs** **Final asserted bit of BC_WE_CTL = First + 8/(CPU Cycle time)** **This field should be set up before the Bcache and Memory are intialized**
reserved	<33:29>	W	0	Must be zero

Table 7-72 BC_SIZE Field Descriptions

Bcache Size<2:0>	Cache Size
000	Invalid Bcache size
00l	l MByte
010	2 MByte
011	4 MByte[1]
100	8 MByte[1]
101	16 MByte
110	32 MByte
111	64 MByte

[1] Preferred Bcache size for DECchip 21164-AA verification

8
Hardware Exceptions and Interrupts

Introduction

This chapter focuses on the:

- The AlphaStation 600 system interrupt logic
- The AlphaStation 600 system errors
- The AlphaStation 600 system machine check logout frame.

EV5 detected errors are not discussed in detail -- (See the EV5 specification)

System Interrupts

The PCI-EISA bridge chip (ESC) set provides two cascaded 8259s and 14 interrupt lines. The PCI interrupt-acknowledge command is used to read the interrupt request vector out of the 8259s, and to flush the EISA-to_PCI write buffers in the (E)ISA bridge chip (used to ensure coherency between data written to memory, and the ensuing interrupt which signals the completion of the write).

The ESC chip provides four pins which can be used for routing PCI interrupts to the cascaded 8259's. However, this is insufficient to efficiently support the 6 EISA slots (11 interrupts) and the 20 interrupts from the 5 PCI slots[1]. Consequently, the 8259s in the ESC chip are used for the (E)ISA interrupts, and separate interrupt logic is provided (called the Main interrupt logic) for the PCI interrupts. This is shown in Figure 8-1. Note that the EISA interrupt signal passes through the Main interrupt logic block.

At the completion of an (E)ISA interrupt, the software must reset the interrupt latch, wait at least 500ns[2] before issuing the EOI command to the 8259s. Table 8-3 defines the EISA interrupt assignment.

The Main interrupt logic is located in the GRU ASIC. Software has visibility into the interrupt logic via the INT_MASK and INT_REQ registers. Table 8-2 defines the interrupt assignment for the Main Interrupt logic.

Table 8-1 gives the system interrupt assignment for the EV5 interrupt pins. An interrupt is enabled if the current IPL is <u>less than</u> the target IPL of the device. All interrupts are disabled when the processor is executing in PALmode.

[1] Software prefers that the PCI interrupts not be wire-ORed

[2] This is an EISA spec. requirement and a consequence of the weak pull-ups on the IRQ lines.

Figure 8-1 The AlphaStation 600 System Interrupt Scheme

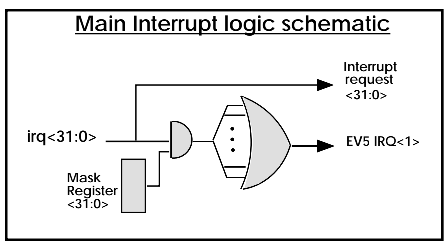
Table 8-1 EV5 Interrupt Assignment

EV5 interrupt pin	IPL	EV5 suggested Usage	AlphaStation 600 Usage
IRQ<0>	20	Corrected system error	Corrected ECC errors detected by CIA
IRQ<1>	21	Medium priority external interrupt	PCI and (E)ISA interrupts
IRQ<2>	22	Timer, interprocessor interrupt.	TOY DALLAS 1287 IRQ pin
IRQ<3>	23	High Priority, Real time devices	Reserved
SYS_MCH_CHK_IRQ	31	Serious System error	PCI/EISA NMI, Most CIA detected errors (CIA_ERROR signal)
MCH_HLT_IRQ	--		Halt button

RESET changes the meaning of the seven EV5 *IRQ inputs, and is described in the EV5 specification and the AlphaStation 600 Reset specification.

Table 8-2 Main Interrupt Logic IRQ Pin Assignment

output	irq	interrupt	Usage (reference designator)
EV5 IRQ<1>	0 1 2 3	int A int B int C int D	PCI Slot 2 -- 64-bit (J9)
	4 5 6 7	int A int B int C int D	PCI Slot 1 -- 64 bit (J10)
	8 9 10 11	int A int B int C int D	PCI Slot 0 -- 64 bit (J11)
	12 13 14 15	int A int B int C int D	PCI Slot 4 -- 32 bit (shared PCI/EISA slot) (J7)
	16 17 18 19	int A int B int C int D	PCI Slot 3 -- 32 bit (J8)
	23-20		Jumpers
	27-24		Module REV
	30-28		Reserved
	31	EISA_INT	8259 INT output
All unmasked irq inputs have equal priority			



The above schematic is simplified; for more information refer to the interrupt section in the CSR chapter.

Each of the irq lines are individually masked by the Mask register. The interrupt output is the OR of the masked interrupts.

The Mask is accessed via the INT_MASK register. The incoming IRQs are sampled before the mask logic and are visible by a read of the INT_REQ register (a synchronizer is provided to ensure no metastability problems when reading the asynchronous PCI interrupts).

Jumpers

Irq	Jumper
20	Alternate console
21	Secure console
22	Reserved
23	Reserved

Module Revision

Systemboard revision:

- Revisions A and B are encoded as 0000
- Revisions C and D are encoded as 0001

Table 8-3 EISA Interrupt Assignment

Priority	Label	Controller	Int/external	Interrupt Source
1	IRQ 0	1	Internal	Internal Timer 1 counter 0 out.
2	IRQ 1		External	Keyboard
3 - 10	IRQ 2		Internal	Interrupt from controller 2
3	IRQ 8#	2	External	Reserved (normally Real Time Clock)
4	IRQ 9			EISA bus pin B04
5	IRQ 10			EISA bus pin D03
6	IRQ 11			EISA bus pin D04
7	IRQ 12			EISA bus pin D05, Mouse interrupt (ABFULL)
8	IRQ 13			Reserved (normally 487 coprocessor error)
9	IRQ 14			EISA bus pin D07
10	IRQ 15			EISA bus pin D06
11	IRQ 3	1		EISA bus pin B25
12	IRQ 4			EISA bus pin B24
13	IRQ 5			EISA bus pin B23
14	IRQ 6			EISA bus pin B22
15	IRQ 7			W EISA bus pin B21

SYS_MCH_CHK_IRQ

The AlphaStation 600 system logically-ORs two signals to generate the SYS_MCH_CHK_IRQ interrupt: the NMI signal from the PCI_EISA bridge chip (ESC) and the CIA_ERROR from the CIA. Hence, all the "serious" system errors (EISA bus time-out, DMA parity error) generate an IPL 31 interrupt.

Table 8-4 defines the hardware and software events which generate the ESC NMI signal[1]. All the NMI sources can be separately enabled/disabled in the ESC chip; and the NMI itself can be enabled/disabled. SERR# and PERR# must be disabled.

[1] This is the standard EISA NMI signal.

Table 8-4 ESC NMI Generation

NMI source	I/O port to Enable/disable	AlphaStation 600 Usage
PCI PERR# PCI data parity error	Port 0061h, bit<2>	DISABLE
PCI SERR# PCI address parity error and other system errors.	Port 0040h, bit<3>	DISABLE
IOCHK# EISA add-in board parity error or some other catastrophic error.	Port 0061h, bit<3>	Allowed
Fail-safe timer time-out. This timer generates an NMI at regular intervals, preventing the system from being tied up in a tight loop. The operating system should reset timer 2 counter 0 at regular intervals to prevent it from causing an NMI under normal operating conditions.	Port 0461h, bit<2>	Allowed
EISA BUS time-out. Set when a bus master uses the bus longer than 8 microseconds.	Port 0461h, bit<3>	Allowed
Software generated NMI Write to I/O port 0462h.	Port 0461h, bit<1>	Allowed

The conditions which generate the CIA_Error are the majority of the error conditions detected by the CIA, and are covered later in this chapter.

The PAL code handler for SYS_MCH_CHK_IRQ requires some care. The PCI specification insists that PERR# is always signaled back to the master of the transaction, allowing the originator of the request, at the hardware or software level, the prerogative of recovery. Thus, when the CIA interrupts the EV5 because of a PERR#, it is possible that the originating PCI master has also set its own interrupt or flag, signaling to its device driver to handle the parity error gracefully.

Halt/Reset Switches

Depressing the Reset button will induce a complete hardware reset of the system (just like a power cycle).

The halt button is debounced and presented to the EV5 MCH_HLT_IRQ. This interrupt is not masked by an IPL level, but is masked if the EV5 is in PAL mode. How the MCH_HLT_IRQ is handled depends on the operating system and is handled uniquely by the O/S specific PALcode. If a "halt" is required then the handler will save information on the stack, and then start running console code. If a "reset" is required[1], then the handler should simply write the value "xxxxDEADh" to the RESET CSR; this will have the same effect as a depressing the reset button.

[1] this "reset" functionality is provided in case some future AlphaStation 600 system packaging variant does not provide a Reset button.

EV5 Error Handling

Please refer to the Error Handling chapter of the EV5 specification for more details. Table 8-5 summarizes the EV5 features.

Table 8-5 EV5 Error Detection Features

Region/fault	Description
Memory/Bcache/uncached access	ECC checked. A maskable, *corrected ECC error* interrupt is provided at IPL 31
Bcache Tags	Parity checked.
Ibox -- No progress time-out	This is intended to prevent a hang. For the AlphaStation 600 system this should be programmed to the largest time-out period. *One possible cause for this trap is a (bad) PCI device that has held a memory lock for a long, long time, and the lock has stalled the EV5.*
Outgoing Bcache data	EV5 does not check outgoing Bcache data. Neither does the Data Switch check this data (for example, no ECC checking on Bcache victims)
Address/cmd from system	Odd Parity checked.

PCI Error Handling

The PCI specification allows for parity error detection on both the address and data cycles. The parity check encompasses the complete AD<31:0> field, the byte enables as well as the 64-bit PCI extension. All PCI options must generate parity, and nearly all PCI options check parity (the exceptions are devices, such as video frame buffers, which cannot create system integrity problems in the event of undetected errors).

Two PCI signals are provided for error notification:

- **PERR#** -- used to report DATA parity errors (except for the Special cycle since these are broadcast writes and PCI devices are not required to listen to special cycles). The master reports read parity errors; and the targets reports write parity errors.
- **SERR#** -- typically used to report ADDRESS parity errors and Special cycle DATA parity errors. However, SERR# is also used to report catastrophic system errors (such as certain master[1] or target aborts). SERR# is asynchronous with respect to a PCI transaction. SERR# is a one clock cycle pulse; but it may linger for a few cycles because of the modest pull-up.

  A selected agent that detects an address parity error will do one of the following:

  (1) claim the cycle and terminate cleanly as though the address was correct;
  (2) claim the cycle and terminate with a target abort; or
  (3) not claim the cycle and let it terminate with a master abort.

  For both PERR# and SERR#, the PCI specification suggests that the transaction should complete gracefully, preserving normal PCI operation.

[1] Not all Master Aborts are catastrophic -- for instance a special cycle always enjoys a master abort.

PERR# Implications

A PCI device detecting a PERR# is expected to gracefully complete the transaction and to inform the device driver by posting an interrupt (or setting a flag). Any PCI-PCI bridge will transparently, propagate bad parity to the target/master, allowing the receiving device to act accordingly by posting interrupts, etc.

Problems arise with the Intel PCI-EISA bridge which does not propagate bad parity; it simply asserts PERR# and expects the system hardware (that is, the CIA) to notify the driver. Consequently, the CIA is compelled to always sample the PERR# line[1] and assert CIA_ERROR anytime PERR# is asserted. The ramification of this is that occasionally two interrupts may be generated; table 8-6 shows this could occur if two PCI devices are communicating on the PCI bus and a data parity error is observed (in this case the CIA will post a SYS_MCH_CHK_IRQ and the PCI device will probably assert an interrupt).

Table 8-6 AlphaStation 600 Handling of PCI Data Parity Errors

Data Parity error scenario	Device detecting parity error	Interrupt by PCI-device?	SYS_MCH_CHK_IRQ
PCI-device Write to PCI-EISA bridge	PCI-EISA	No	Yes
PCI-device Write to CIA	CIA	No	Yes
CIA Read of any PCI-device	CIA	No	Yes
PCI-EISA bridge Read of any PCI device	PCI-EISA	No	Yes
PCI-device Read of PCI-device (excluding PCI-EISA bridge)	PCI device & CIA	Probable	Yes

SERR# implications

CIA Target Abort:

Should the CIA receive a target abort for its own PCI transaction then it will assert SERR#, and it will assert CIA_ERROR which in turn will assert the SYS_MCH_CHK_IRQ input.

PCI Address Parity Error -- CIA Master:

As a master device, the CIA will become aware of an address parity error either via a master abort, target abort or SERR#. In all three cases CIA_ERROR will result.

However, since the PCI-EISA bridge subtractively decodes the PCI address space, and thus absorbs all unclaimed addresses, it is clearly the most-likely target for a bad PCI address. Unfortunately, the PCI-EISA Bridge chip does **not** report address parity errors; and consequently, the majority of PCI memory address parity errors will not be detected (and instead will access/corrupt some EISA/ISA device).

PCI Address Parity Error -- CIA Target:

There are three possible options:

- If the (erroneous) address hits in one of the four PCI Windows in the CIA then the CIA will claim the cycle. This is because the CIA hardware is optimized to accept a PCI transfer (that is, DEVSEL) as soon as possible, and requiring DEVSEL to be qualified by an address parity check would delay the acceptance by one cycle.

[1] and not just when the CIA is the target/master of a transaction.

There are two scenarios to consider:

— **DMA Read case:** The CIA will have already set the wheels in motion to fetch the read data before the address parity error is detected. In this case the fetched data is discarded; but note that this error can quite easily spawn off numerous other errors, including: invalid-TLB-entry error; ECC error on the fetched data (or TLB miss); or a non-existent memory access. For these secondary errors, the CIA_ERR register will already be locked by the address parity error, and thus the CIA_ERR<LOST> bit will be set.

  The CIA is now left with two means of terminating this PCI transaction: either it could disconnect, or it could Target Abort. A disconnect is not really a viable option - it will result in the PCI master resuming the transaction after a few cycles[1], and causing perhaps another address parity error; and thus hogging the PCI-bus ad infinitum. The CIA will Target Abort the offending PCI transfer, and assert SERR# and CIA_ERROR.

— **DMA Write case:** By the time that the CIA had detected the address parity error it could have already accepted a Dword of data, and the PCI master may no longer be on the bus (for example, a one word DMA). If the DMA is still active, the CIA will continue accepting the data (and discarding it) until the DMA write completes. No data will be written to memory. As before, numerous secondary errors are possible (invalid TLB entry, non-existent memory, ECC error, etc.), which would result in the CIA_ERR<LOST> flag being set. The CIA will assert CIA_ERROR.

— **DMA Write/Read complication:** The PCI specification states that any agent can check and signal address parity errors on SERR#, regardless of intended master and target. Consequently, it is possible (but probably, very unlikely) that while the CIA has detected the address parity error, some other PCI agent will signal SERR# for this same address parity error. Unfortunately, the assertion of SERR# has no timing relation to any PCI transaction, and thus this PCI agent can assert SERR# at some indeterminate, future time, which may or may not coincide with the CIA's detection of the address parity error. If the PCI agent is very late is sending the SERR# then a second CIA_ERROR is possible.

- The address does not hit in any of the four PCI windows in the CIA, but it does hit in some other PCI devices window. In this case, the other PCI device will claim the cycle and generate a SERR#. The CIA will induce SYS_MCH_CHK_IRQ on detecting SERR#.
- The address does not hit in any of the four PCI windows in the CIA, and nor does any other PCI device claim the cycle[2]. In this case the originator will Master Abort and again a SERR# will be generated, and CIA will induce SYS_MCH_CHK_IRQ.

Address Parity Error and LOCK Bit Set.

This scenario can only occur if the master which holds the lock has an address parity error which fortuitously hits in one of the CIA PCI-address windows. It has to be the lock-master since all other PCI transactions will be retried (that is, the PCI lock protocol). In this case the PCI lock will be cleared (or not set if this is the start of a lock sequence) and the CIA lock-register will be invalidated. Otherwise the behavior is as for a standard address parity error as described above.

[1] Assuming that the device does not listen to SERR# (which is true of the PCI-EISA bridge for example).

[2] Only possible for addresses which are not subtractively-decode by the PCI-EISA bridge (that is PCI I/O-address exceeding 64-KBytes).

DAC Cycle Address Parity Error:

The handling for this case is the same as for a standard (SAC) PCI address parity error, as described above. The main difference is that the PCI_ERRx registers will contain a DAC indication, the least-significant 32 address bits, and the command field for the subsequent command (of the DAC pair). The MS bits of the address can be inferred by reading the PCI DAC BASE register in the CIA.

Special Cycle Data Parity Error:

The CIA will not detect this error. The handling of this error is as follows:

When a special cycle is issued there are two situations to consider. First, all the PCI devices are deaf to the special cycle, in which case a parity error is immaterial; and second, if a device is listening for the special cycle then it will almost certainly report the data parity error as a SERR#. The CIA will intercept the SERR# and report it to the EV5 via the CIA_ERROR.

AlphaStation 600 Error Handling

The CIA ASIC together with the EV5 processor detect the vast majority of AlphaStation 600 system errors; the remaining few errors are detected by the PCI-EISA bridge chip-set and by the PCI option devices (SERR# and PERR#).

When an error is detected, information about the failure is latched and the failing command is completed or terminated as gracefully as possible. The error condition is signaled to the EV5 via an interrupt which wakes the AlphaStation 600 system-specific PAL code handler. This handler reads the error information and initiates the appropriate action.

Figure 8-2 AlphaStation 600 Error Logic

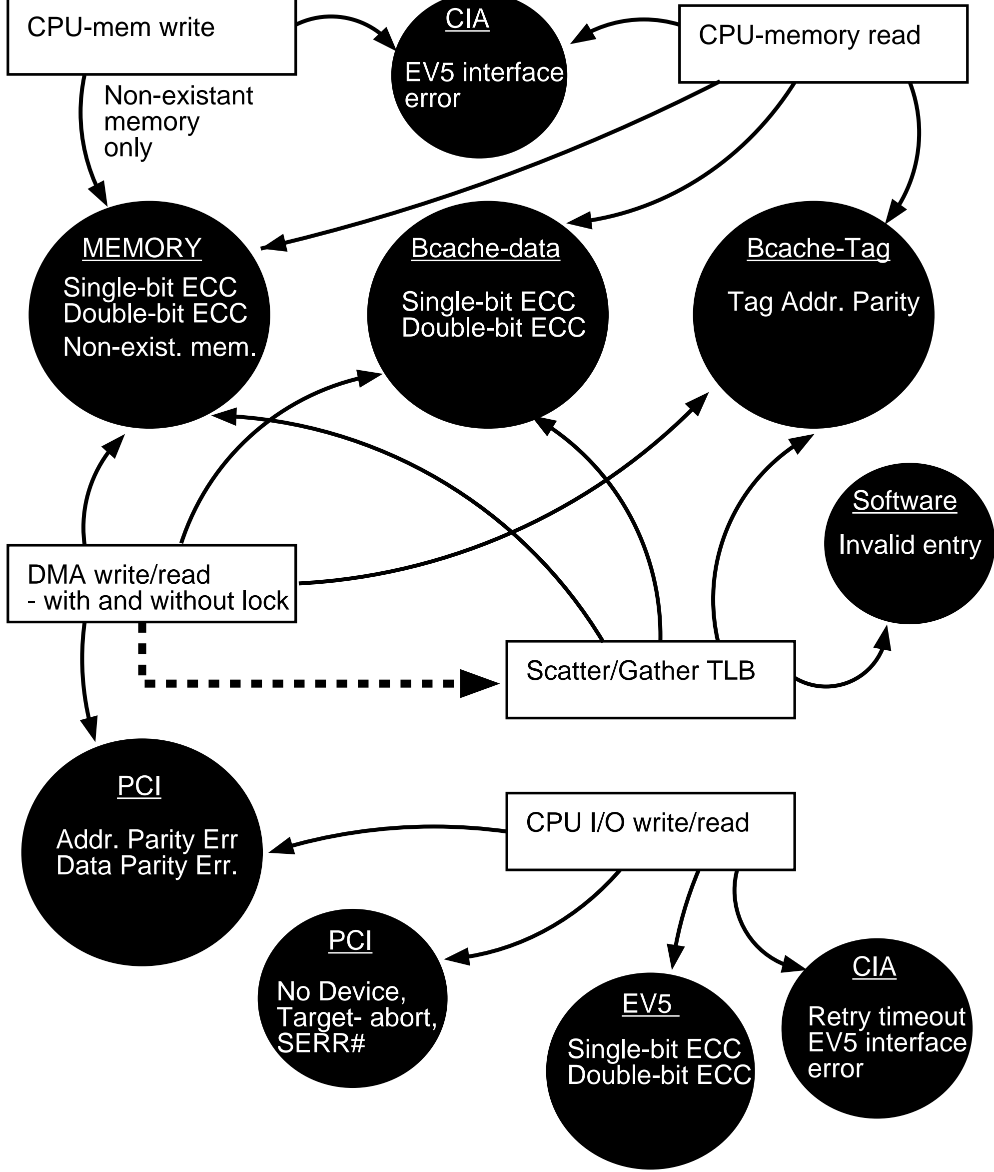
Figure 8-2 shows the AlphaStation 600 system block diagram with the major error detection/generation logic.

- **Memory:** ECC protected.
- **Bcache:** data is ECC protected; tag is parity protected.
- **Data Switch ASICs:** no ECC or parity logic. Data from memory is passed un-modified to the EV5/Bcache or the CIA ASIC.
- **IOD bus:** ECC checked/generated. Data from either the EV5/Bcache or memory is ECC protected, and is checked/corrected by the CIA; data from the CIA to the Data Switch (for example, PCI DMA write data or an internal CIA CSR) has ECC generated for it by the CIA.
- **Address/command bus:** parity checked at both the EV5 and the CIA.

- **PCI bus:** parity protected. PERR# (Data parity error) and SERR# (address parity and system error) are detected by the CIA.
- **(E)ISA bus:** Parity for I/O board memory and other catastrophic errors via the ISA IOCHK signal.
- **Scatter/gather TLB in the CIA:** No checking.
- **Flash ROM:** No parity/ECC checking. Checksum if implemented by software.
- **Lemmon Bus:** no checking.

Figure 8-3 shows the three classes of transactions (CPU memory, CPU I/O and DMA) and their possible errors -- for example a DMA read can suffer either a memory or Bcache ECC error.

Figure 8-3 Possible Errors



CIA ASIC Error Registers

Table 8-7 describes the CIA error registers. More details are to be found in the CSR chapter. Most of the error registers preserve the address and command of the offending transaction.

Table 8-7 CIA Error Registers

Description	CIA error register		Details
Error indication	CIA Error register	CIA_ERR	This registers is the most useful in that it indicates which error occurred: • Bcache/memory ECC error • S/G map invalid entry • etc.
CIA status	CIA status register	CIA_STAT	What the CIA was doing at the time of the error. However, since the CIA can do numerous concurrent operations then the status may have nothing to do with the error condition.
EV5 Interface error	EV5 Interface-error Register 0	CPU_ERR0	Address and Command field for: • Interface parity error • EV5 read/write to non-existent memory
	EV5 Interface-error Register 1	CPU_ERR1	
ECC syndrome	ECC error syndrome Register	CIA_SYN	ECC syndrome
Memory Port errors	Memory Port Status reg 0	MEM_ERR0	Address, command and memory timing parameters when error occurred for following reasons: • EV5 read/write to non-existent memory
	Memory Port Status reg 1	MEM_ERR1	
	Memory Port Status reg 2	MEM_ERR2	
PCI detected errors	PCI Error Register 0	PCI_ERR0	PCI Address and Command pertaining to error. Also indicates if the CIA was the master or target of the PCI transaction.
	PCI Error Register 1	PCI_ERR1	
	PCI Error Register 2	PCI_ERR2	
Error Mask	CIA Error Mask Register	ERR_MASK	Mask the detection and reporting of CIA errors

All the error registers are frozen (locked) on the first detection of an error condition, thus preserving the error state until the error handler clears the CIA_ERR register. If subsequent error conditions occur (for example, a PCI address parity error can induce a non-existent memory error) then the "lost" fields in the error registers are set.

Not all the error registers contain valid information for all errors: for instance, the PCI error registers are irrelevant for a CPU memory fill error. To help determine the valid error registers the error-analyzing code should first direct its attention to the CIA_ERR register.

Table 8-8 summarizes the CIA_ERR definitions. Bits <11:0> define the error condition when bit 31 is set; the remaining bits indicate what other errors occurred in the shadow of this first error condition. The fourth column of this table indicates which operation could have suffered the error condition. For example, bit 6 is set when a PCI agent attempts a DMA read/write of memory and the CIA detects an address parity error; this is the only way that this bit can be set. However, most of the bits are set for a number of reasons.

Table 8-8 CIA_ERR Register

Extent	Mnemonic	Description	Possible Cause	Valid Registers
<0>	COR_ERR	Corrected single bit ECC error.	This error cannot occur for a CPU-to-memory read/write (CPU-mem reads are checked by the EV5 and CPU-mem writes are not checked by AlphaStation 600 system). This error is applicable to: • DMA read/write data ECC error • S/G TLB miss PTE ECC error • ECC error on CPU-IO write data from EV5	CIA_SYN (PCI_ERR0,1,2)
<1>	UN_COR_ERR	Uncorrectable ECC error		
<2>	CPU_PE	EV5 to CIA addr/command bus parity error	• CPU-IO read/write • CPU-mem read/write	CPU_ERR0,1
<3>	MEM_NEM	Non-existent memory	• DMA read/write -- bad address mapping from PCI to memory; bad PTE • CPU-MEM read/write to bad address.	MEM_ERR0,1 (PCI_ERR0,1,2)
<4>	PCI_SERR	PCI SERR line asserted	• CPU-IO write address parity error • PCI device detected internal problem	PCI_ERR0,1,2
<5>	PCI_PERR	PCI PERR line asserted	• DMA read/write data parity error • CPU-IO write/read data parity error	
<6>	PCI_ADDR_PE	PCI address parity error.	• DMA read/write address parity error	
<7>	RCVD_MAS_ABT	CIA received a PCI Master abort error	• CPU-IO read/write address parity error (special cycle master abort is not an error)	
<8>	RCVD_TAR_ABT	CIA received a PCI Target abort error	• CPU-IO read/write address parity error	
<9>	PA_PTE_INV	Scatter/gather TLB invalid entry	• DMA read/write	
<10>	FROM_WRT_ERR	Flash ROM write error	• CPU write attempted while Flash ROM write is disabled (FROM_WR_EN is not enabled)	None
<11>	IOA_TIMEOUT	PCI IO timeout occurred	• CPU-IO read/write	(PCI_ERR0,1,2)
<15:12>		Reserved		
<16> <17> <18> <19>	Lost_COR_ERR Lost_UN_COR_ERR Lost_CPU_PE Lost_MEM_NEM	These lost errors occur in the shadow of a prior error, and correspond to errors normally logged in bits <3:0> of this register		
<20>		Reserved		
<21> <22> <23> <24> <25> <26> <27>	Lost_PCI_PERR Lost_PCI_ADDR_PE Lost_RCVD_MAS_ABT Lost_RCVD_TAR_ABT Lost_PA_PTE_INV Lost_FROM_WRT_ERR Lost_IOA_TIMEOUT	These lost errors occur in the shadow of a prior error, and correspond to errors normally logged in bits <11:5> of this register		
<30:28>		Reserved		
<31>	ERR_VALID	Error occurred		

The last column of table 8-8 shows which error registers are applicable to the various error condition. When a register is enclosed in parentheses (for example-- (PCI_ERR0,1,2)), then this register may have valid information; to determine if it does contain valid data then the CIA_STAT register must be analyzed in conjunction with the status information in PCI_ERR0,1.

The CIA_STAT register is frozen at the time of an error, and provides information on what the CIA was fetching from the DSW asic at the time of the error. Whenever, this register indicates (via the DM_ST field) that a DMA or I/O operation was in effect, and the PCI_ERR0 register (command field) indicates the same operation, then the PCI_ERR0 and PCI_ERR1 registers are valid. The "MEM_SOURCE" bit of CIA_STAT is also useful to differentiate between a CPU and a DMA operation (for example, to help determine if a DMA or CPU operation suffered an ECC error when CIA_ERR<0> is set).

Note that the CIA can perform multiple transactions (for example, a CPU fill concurrently with a CPU-I/O write). Thus it is possible for the CIA_STAT to reflect the status of an operation which has no relevance to the error condition (for example, the CIA_STAT could reflect a DMA which was ongoing while an EV5-CIA interface error occurred). Consequently, the CIA_STAT is not an infallible indicator of the error condition.

Table 8-9 is an extension of table 8-8 and shows the most likely cause of a hardware error. Once the systemboard is suspected of causing the error then the AlphaStation 600 system is effectively useless and recovery is impossible. If the memory or PCI option card are suspected, then diagnostics should be run to help further isolate the problem.

Table 8-9 CIA_ERR Register Fault Indication

Extent	Mnemonic	Possible Cause	Possible Faulty hardware (in order of likelihood)
<0> <1>	COR_ERR UN_COR_ERR	DMA Read/write data ECC error	(1) Memory (2) Systemboard (IOD-bus, Memory-buses, DSW, CIA)
		S/G TLB miss PTE ECC error	
		ECC error on CPU-IO write data from EV5	(1) Systemboard (EV5-data bus, IOD-bus, DSW, CIA)
<2>	CPU_PE	CPU-IO read/write	(1) Systemboard (EV5-addr/command bus, CIA)
		CPU-mem read/write	
<3>	MEM_NEM	DMA read/write -- bad address mapping from PCI to memory; bad PTE	(1) Software problem (for example, bad PTE) (2) Systemboard (CIA, address bus to memory)
		CPU-MEM read/write to bad address.	
<4>	PCI_SERR	CPU-IO write address parity error	(1) PCI option failure (2) Systemboard (PCI bus, CIA)
		PCI device detected internal problem	(1) PCI option failure
<5>	PCI_PERR	DMA read/write data parity error	(1) Systemboard (PCI bus, CIA) (2) PCI option failure
		CPU-IO write/read data parity error	
<6>	PCI_ADDR_PE	DMA read/write address parity error	
<7>	RCVD_MAS_ABT	CPU-IO read/write address parity error	
<8>	RCVD_TAR_ABT	CPU-IO read/write address parity error	
<9>	PA_PTE_INV	DMA read/write	(1) Software problem with PCI window CSRs or PTEs (2) Systemboard (CIA)
<10>	FROM_WRT_ERR	CPU write attempted while Flash ROM write is disabled (FROM_WR_EN is not enabled)	
<11>	IOA_TIMEOUT	CPU-IO read/write	(1) PCI option failure

CIA Error Mask Register

This register is used to disable the detection and reporting of certain errors. Most of the CIA detected errors can be disabled (see Table 8-10). This is mainly a debug feature to help isolate faulty error detection logic. The default on power-up is for error logging and reporting to be off.

Table 8-10 CIA Error Mask Register

Extent	Error Mnemonic	Enabled error
<0>	COR_ERR	Corrected single bit ECC error.
<1>	UN_COR_ERR	Un-corrected ECC error
<2>	CPU_PE	EV5 to CIA address/command bus parity error
<3>	MEM_NEM	Non-existent memory
<4>	PCI_SERR	PCI SERR line asserted
<5>	PCI_PERR	PCI PERR line asserted
<6>	PCI_ADDR_PE	PCI address parity error.
<7>	RCVD_MAS_ABT	Master abort error
<8>	RCVD_TAR_ABT	Target abort error
<9>	PA_PTE_INV	Scatter/gather TLB invalid entry
<10>	FROM_WRT_ERR	Flash ROM write error
<11>	IOA_TIMEOUT	CPU-IO operation timeout on the PCI.
<32:12>		Reserved

The error logic is disabled when the mask bit is cleared -- in which case it is as if the error never existed (no logging or reporting of the error).

CIA Error Reporting

Table 8-11 lists the three ways the AlphaStation 600 system reports errors to the EV5. Most of the errors detected by the CIA fall into the Class 2 category.

Table 8-11 Error Reporting to EV5

Error	EV5 input signal	Comments
Class 1	IRQ<0>	Low Priority.
		Examples: Correctable ECC Errors.
Class 2	SYS_MCH_CHK_IRQ	High priority, fatal errors. Can be masked at IPL 31, in PALmode, and by CIA_MASK CSR.
		Examples: Uncorrectable ECC errors, Invalid S/G map, time-out, Target and Master abort, PCI parity error, Invalid address, Parity error on the address/command bus from the EV5 to the CIA.
Class 3	FILL_ERROR_H	The EV5 takes a PALcode trap to the MCHK entry point.
		Examples: Any error associated with a CPU read must be reported this way in order to un-stall the CPU. Examples are: reads to non-existent memory (physical or PCI) or to non-existent CSRs.

CIA Detected Errors

The following tables tabulate the error conditions associated with an operation. These tables assume a single error occurred. This is consistent with the error registers which only log the information for the first error detected. There is a small possibility that multiple unrelated errors (for example, DMA read and CPU read errors) occur concurrently; this situation is not covered.

The following tables are to a large extent the same as table 8-8 just re-arranged in a different format with more detail. Tables 8-8 and 8-9 are better suited for the software error-handler, whereas the following tables are better suited for the hardware group (to help ensure that the hardware has covered all the possible error cases, and secondly to help isolate which functional verification tests are needed). Finally, note that Tables 8-8 and 8-9 do not include class 3 errors.

Table 8-12 DMA Read Associated Errors

Error type	CIA_ERR bit set	Relevant CIA Error Registers	What Detected The Error	Action (see table 8-11)
Corrected ECC errors: • DMA data from memory • DMA data from cache • S/G TLB miss from memory • S/G TLB miss from cache	COR_ERR	CIA_SYN PCI_ERR0,1 if status bits indicate that DMA Read operation in progress.	CIA	Class 1
Uncorrectable ECC errors: • DMA data from memory • DMA data from cache • S/G TLB miss from memory • S/G TLB miss from cache	UN_COR_ERR			**Class 2** The PCI agent which instigated the DMA read has the transaction Target aborted by the CIA.
S/G TLB invalid entry *(Note 1)*	PA_PTE_INV	PCI_ERR0,1		
Non-existent memory access: • DMA tries to access NEM • PTE access to NEM	MEM_NEM	MEM_ERR0,1 PCI_ERR0,1 if doing DMA		
PCI address bad parity (PCI parity encompasses the address and command fields)	PCI_ADDR_PE	PCI_ERR0,1,2	CIA and possibly a PCI device	**Class 2** The PCI agent which instigated the DMA read has the transaction Target aborted by the CIA. It is possible for some PCI agent to concurrently detect the address parity error and then assert SERR#. The CIA does not capture this as a lost bit.
PCI data bad parity	PERR		PCI device	**Class 2** The PCI agent which instigated the DMA read will detect the data parity error, and will assert PERR# and possibly post an interrupt.
Address/cmd bad parity -- used for READ/FLUSH	Not applicable.	See EV5 spec.	EV5	Machine check trap
Bcache tag bad parity				

Notes:

1 The scatter/gather TLB is assumed to be coherent with memory -- whenever software change a PTE they must flush the S/G TLB. The hardware reports an invalid entry when it is servicing a TLB miss and fetches an invalid entry from memory.

Table 8-13 DMA Write Associated Errors

Error type	CIA_ERR bit set	Relevant CIA Error Registers	What Detected The Error	Action (see table 8-11)
Corrected ECC errors: • DMA data to memory • DMA data to cache • S/G TLB miss from memory • S/G TLB miss from cache	COR_ERR	CIA_SYN PCI_ERR0,1 if status bits indicate that DMA write operation in progress.	CIA	Class 1
<u>Uncorrectable ECC errors</u>: • DMA data to memory • DMA data to cache • S/G TLB miss from memory • S/G TLB miss from cache	UN_COR_ERR			**Class 2** The PCI agent which instigated the DMA write is allowed to complete normally, except that the write data is not written to memory. The CIA does **NOT** issue a Target abort.
Non-existent memory access: • DMA tries to access NEM • PTE access to NEM	MEM_NEM	MEM_ERR0, 1 PCI_ERR0,1 if doing DMA		
<u>S/G TLB invalid entry</u> *(Note 1)*	PA_PTE_INV	PCI_ERR0,1		As above, but note following: For a long DMA write (many blocks), the CIA will take the first block of data and disconnect the DMA write (because of a TLB miss -- invalid TLB entries always start with a TLB miss). The CIA will discard the data in its write buffer, and the CIA will accept the next PCI transaction. The retried (bad) DMA write may get in again the whole procedure repeats, on a block-by-block basis. Thus, the interrupt handler should be aware that a bad DMA device may still be active when servicing the error condition.
PCI address bad parity (PCI parity encompasses the address and command fields)	PCI_ADDR_PE		CIA and possibly a PCI device	**Class 2** The PCI agent which instigated the DMA write is allowed to complete normally, except that the write data is not written to memory. The CIA does **NOT** issue a Target abort.
PCI data bad parity	PERR		CIA	**Class 2** The PCI agent which instigated the DMA write is allowed to complete normally, but the write data is discarded by the CIA.
<u>Address/cmd bad parity</u> -- used for READ/FLUSH	Not applicable.	See EV5 spec.	EV5	Machine check trap
Bcache tag bad parity				

<u>Notes:</u>

1 The scatter/gather TLB is assumed to be coherent with memory -- whenever software change a PTE they must flush the S/G TLB. The hardware reports an invalid entry when it is servicing a TLB miss and fetches an invalid entry from memory.

Table 8-14 I/O Write and Special Cycle Errors

Error detected	CIA_ERR bit set	Relevant CIA Error Registers	What Detected The Error	Action (see table 8-11)
Corrected ECC errors: EV5 I/O write data had ECC error	COR_ERR	CIA_SYN PCI_ERR0,2 if status bits indicate that IO operation in progress.	CIA	**Class 1**
Uncorrectable ECC errors: EV5 I/O write data had ECC error	UN_COR_ERR		CIA	**Class 2** The CIA cleanly terminates the PCI transaction when it detects bad CPU-IO write data. The PCI agent does not receive any bad data.
Bad PCI address No PCI device responds (see Note 1)	RCVD_MAS_ABT	PCI_ERR0,2	CIA	**Class 2** This is a No-DEVSEL error - software has somehow generated a bad address.
EV5 System Interface bad parity	CPU_PE	CPU_ERR0,1	CIA	**Class 2** No PCI transaction will occur.
PCI address bad parity -- 1 CIA received a PCI target abort	RCVD_TAR_ABT	PCI_ERR0,2	CIA and PCI device possibly	**Class 2** The PCI agent detected the address PE and terminated the transaction with a target abort.
PCI address bad parity --2 PCI device asserted SERR#	PCI_SERR		PCI device and CIA (via the SERR#)	**Class 2** The PCI agent allowed the CPU-IO write to complete cleanly, and asserted SERR.
PCI address bad parity -- 3 CIA received a PCI master abort (see note 1)	RCVD_MAS_ABT		CIA	Class 2 No PCI agent responded.
PCI data bad parity	PERR		PCI device	Class 2 PCI device will detect the data PE and assert PERR#, and also the PCI device may set iots own interrupt.
IO timeout	IOA_TIMEOUT	PCI_ERR0,2 if this reg status indicates CPU-IO operation	CIA	Class 2 The CIA has an I/O timeout register to detect fatal PCI bus hangs.

Notes:

1 This will only occur for a PCI I/O address which accesses beyond 64 KB. The PCI-EISA bridge will claim all other unclaimed addresses (that is, subtractive decode). Any problems on the EISA bus will result in a PCI-EISA NMI (for example, bus timeout). Note that the PCI-EISA Bridge chip does not detect address parity errors.

Table 8-15 CPU I/O Read and PCI Interrupt-ACK Errors

Error detected	CIA_ERR bit set	Relevant CIA Error Registers	What Detected The Error	Action (see table 8-11)
Invalid PCI address No PCI device responds	RCVD_MAS_ABT	PCI_ERR0,2	CIA	**Class 2** This is a No-DEVSEL error - software has somehow generated a bad address.
EV5 System Interface bad parity	CPU_PE	CPU_ERR0,1	CIA	**Class 2** No PCI transaction will occur.
PCI address bad parity -- 1 CIA received a PCI target abort	RCVD_TAR_ABT	PCI_ERR0,2	CIA and PCI device possibly	**Class 2** The PCI agent detected the address PE and terminated the transaction with a target abort.
PCI address bad parity -- 2 CIA received a PCI master abort (see note 1)	RCVD_MAS_ABT		CIA	Class 2 No PCI agent responded.
PCI data parity error	PERR		CIA	Class 2 CIA detects data PE and asserts PERR# line
IO read timeout	IOA_TIMEOUT	PCI_ERR0,2 if this reg status indicates CPU-IO operation	CIA	Class 2 The CIA has an I/O timeout register to detect fatal PCI bus hangs.
EV5 detects ECC data error	Not applicable		EV5	Machine Check

Notes:

1 This will only occur for a PCI I/O address which accesses beyond 64 KB -- the PCI-EISA bridge will claim all other unclaimed addresses (that is, subtractive decode). Any problems on the EISA bus will result in a PCI-EISA NMI (for example, bus timeout). Note that the PCI-EISA Bridge chip does not detect address parity errors.

Table 8-16 CPU/Memory Read Associated Errors

Error detected	CIA_ERR bit set	Relevant CIA Error Registers	What Detected The Error	Action (see table 8-11)
ECC corrected fill error	Not applicable.		EV5	System MCHK interrupt (maskable)
ECC uncorrected fill error				Machine check
Non-existent memory address	NEM	MEM_ERR0,1	CIA	Class 3 Garbage data with good ECC is returned to the EV5
EV5 Address/cmd bad parity	CPU_PE	CPU_ERR0,1	CIA	

Table 8-17 CPU/Memory Victim/Write Associated Errors

Error detected	CIA_ERR bit set	Relevant CIA error registers	Who detected error	Action (see table 8-11)
Non-existent memory address	NEM	MEM_ERR0,1	CIA	Class 2 Data is not written to memory
EV5 Address/cmd bad parity	CPU_PE	CPU_ERR0,1	CIA	

Error Insertion

In general there will be two mechanisms for inserting errors:

Force Error Registers

The CIA contains a CSR that contains force error bits. If the bit in the register is asserted then a parity error will be forced on the next transaction, the bit will be cleared and a forced error bit will be set in a status register.

The CIA will contain force error bits for:

- The EV5 to CIA address/command interface
- CIA to PCI (drive bad parity -- check the PCI-EISA bridge parity detect logic)

Accessing Main Memory Via CPU Uncached Space

The AlphaStation 600 system can be configured to do a "bounce around" path which converts a CPU IO read/write to a pseudo-DMA read/write into memory.

The trick here is to configure the PCI window to a certain address space, and then to perform CPU uncached IO read/writes to the same address space. The CPU IO read/write will be sent out on the PCI bus by the CIA ASIC, and will also be accepted by the CIA since the transaction will hit in the PCI window. The CPU IO read/write will thus be transferred to a PCI memory space read/write (that is, a DMA read/write).

Writing Bad ECC Into Memory

The CIA_DIAG CSR can be used instead of the ECC generation logic in the CIA to load bad ECC into memory. Thus diagnostics can load bad ECC into memory using the "bounce-around" path described above. This bad ECC can be accessed via a CPU fill or via a pseudo-DMA using the "bounce-around" path.

With the "bounce-around" path, the complete DMA path inside the CIA and DSW is tested. Moreover, Read and Flush operations to the EV5/Bcache are always performed (since the width of a CPU uncached operation is less than a cache block in size); and the scatter/gather or direct addressing scheme can be used.

Machine Check Logout Data Structure

The Machine check logout data structure consists of :

- A Correctable Error (small) logout frame
  - This is used to log correctable errors such as Single bit ECC errors.
  - These errors can be detected by either the processor or the system (the CIA ASIC in the case of AlphaStation 600 system).
- An Uncorrectable Error (large) logout frame
  - This is used to log uncorrectable errors such a double bit ECC errors
  - These errors can also be detected by either the processor or the system

The PALcode will fill in these data structures when it handles the interrupt generated by either a correctable or uncorrectable error. The system software will have a handler that will read the information in these logs and save the data into their appropriate error log area.

EV5 generated machine checks, such as single bit or double bit ECC errors, will be vectored to the machine check entry point (0x400 off of the current PAL_Base) in the current PALcode.

System generated machine checks will assert the SYS_MCH_CHK_IRQ line on the EV5. These interrupts will occur at IPL level 31, thus they can be masked out when running at EV5 internal IPL = 31.

Correctable Error Logout Frame

The figure below shows the format of a correctable Error Logout Frame on the AlphaStation 600 system.

Figure 8-4 Correctable Error Machine Check Logout Frame

63 — 0	Offset
FLAGS	0000
OFFSETS	0008
MACHINE CHECK CODE	0010
EI_ADDR	0018
FILL_SYN	0020
EI_STAT	0028
ISR	0030
CIA_SYN	0038
MEM_ERR0	0040
MEM_ERR1	0048
CIA_ERR_STAT	0050
CIA_ERR	0058

The description of the fields in the correctable machine check logout frame are as follows:

- FLAGS
  - — This quadword is broken up into two longword fields. The low longword will contain the size of the frame in bytes, while the high longword is used for flags where:
    - * bit 64 = 1 for retryable error
    - * bit 63 = 1 for second error occurred
- OFFSETS
  - — This quadword is broken up into two longword fields:
    - * longword 0 contains the byte offset to the EV5 specific information.
    - * longword 1 contains the byte offset to the System (AlphaStation 600) Specific information. The offset to the AlphaStation 600 system specific information begins at offset 38h.
- MACHINE CHECK CODE
  - — This is the type of machine check that has caused the interrupt to occur. These fall into two classes, processor detected and system detected. The following table lists all possible correctable machine check codes on the AlphaStation 600 system.

Table 8-18 Correctable Machine Check Error Codes

Description	Code
EV5 detected Correctable ECC Error	0x86
CIA detected Correctable ECC Error	0x201

- EI_ADDR

  — This is the contents of the EV5 EI_ADDR register at the time of the failure. The contents of this field is meaningful for processor detected correctable errors

- FILL_SYN

  — This is the contents of the EV5 FILL_SYN register at the time of the failure. The contents of this field is meaningful when the error is a processor detected single bit ECC error. The value in this register will isolate the failure to a data bit or check bit within the failing quadword.
  NOTE:
  This is the register that will contain the failing syndrome for processor detected single bit ECC errors. The data located in the CIA CIA_SYN register should NOT be used to isolate the failing bit.

- EI_STAT

  — This is the contents of the EV5 EI_STAT register at the time of the failure. Where,

  * bit 31 = 1 on a processor detected single bit error
  * bit 30 = 0 if the source of the error came from the B-Cache
    = 1 if the source of the error came from the memory
  * bit 34 = 0 if the error occurred during a D-stream fill
    = 1 if the error occurred during an I-stream fill

- ISR

  — This is the contents of the EV5 ISR register at the time of the failure. This register will tell you what interrupts were pending. The bits of interest for correctable errors are the MCK bit (bit 21) and the CRD bit (bit 30). If you see the CRD bit set then, the machine check was probably caused by a processor detected correctable error. If the CRD bit is clear and the MCK bit is set then , the machine check was probably caused by a system detected correctable error. If it was a system correctable error then, the CIA CSR's that follow will be of interest.

- CIA_SYN

  — This is the contents of the CIA Error syndrome register. This will isolate the failing bit in a quadword for system detected correctable errors.
  NOTE:
  This register will contain the failing ECC syndrome for system detected ECC errors. The data located in the EV5 FILL_SYN register should NOT be used to isolate the single bit error.

- MEM_ERR0

  — This is the contents of the CIA Memory Port Status Register 0. This will tell you low order address bits for the location in memory that contained the single bit error.

- MEM_ERR1

  — This is the contents of the CIA Memory Port Status Register 1. This will contain the high order address bits, the current command, and the current INT4_VALID mask.

- CIA_ERR_STAT

  — This is the contents of the CIA Error status register. This will contain status information such as the transaction type, current state of the CIA queues, etc

- CIA_ERR
  - — This is the contents of the CIA Error register. This register will have bit 0 set to a 1 on a system detected correctable error. The PALcode will dismiss the error by writing a 1 to bit0 of this register after it has logged all the appropriate information. This will unlock the CIA error registers so they can be updated on a future error.

Deciphering a Correctable Error Machine Check Logout Frame

If you want to decipher the cause of a correctable single bit ECC error, You should perform the following steps.

- Check the machine check code type
- IF the machine check code == 0x86 THEN (processor detected)
  - — FILL_SYN register contains the failing syndrome
  - — EI_ADDR register contains the physical address where failure occurred
  - — EI_STAT register contains the following information
    - * bit 31 = 1 for correctable ECC error
    - * bit 30 = 0 if data came from the BCache
      bit 30 = 1 if data came from Memory
    - * bit 34 = 1 if error occurred during ICache fill
      bit 34 = 0 if error occurred during DCache fill
- ELSE IF the machine check code = 0x201 THEN (System detected)
  - — CIA_SYN register contains the failing syndrome.
  - — CIA_ERR register bit <0> = 1 for correctable ECC error.
  - — MEM_ST0,1 contains NO applicable information.
  - — To determine if the cause of the error was a DMA, Scatter/gather TLB miss or a CPU IO write then CIA_STAT must be analyzed as follows:

CIA STATUS register CIA_STAT	DMA Read		DMA Write		IO write
	TLB miss	No TLB miss	TLB miss	No TLB miss	
PCI_STATUS<1:0>	1	1	1	1	X
TLB_MISS	1	0	1	0	0
DM_ST<3:0>	6	6	6	7	2, 3, 8
PA_CPU_RES<1:0>	01: Memory data 10: Scache data 11: BCache data		Not relevant		
MEM_SOURCE IOA_VALID<3:0> CPU_QUEUE<2:0>	Not relevant				

  - * For a DMA read/write (no TLB miss), the PCI address is saved in PCI_ERR1. PCI_ERR0 should indicate an active target state (that is, MASTER_STATE=0 and TARGET_STATE <> 0). PCI_ERR0<WINDOW> will indicate which PCI window hit the PCI address, allowing software to access the appropriate T_BASE CSR, and thus determine the physical memory address which suffered the ECC error. The PCI_ERR0<CMD> will indicate if the PCI command was a READ or a WRITE. The CIA_STAT<PA_CPU_RES> will indicate if the DMA data originated from memory or the EV5 caches.
  - * For a Scatter/Gather TLB miss, the PCI address is saved in PCI_ERR1. PCI_ERR0 should indicate an active target state (that is, MASTER_STATE=0 and TARGET_STATE <> 0). PCI_ERR0<WINDOW> will in-

dicate which PCI window hit the PCI address, allowing software to access the appropriate T_BASE CSR, and thus determine the physical memory address which suffered the ECC error. The selected PCI window must be a scatter/gather window. The PCI_ERR0<CMD> will indicate if the PCI command was a READ or a WRITE. The CIA_STAT<PA_CPU_RES> will indicate if the DMA data originated from memory or the EV5 caches.

  * If the error was a CPU-IO write then the offending address is NOT saved in any register. However, this bug is a serious systemboard problem indicating a failure of either: (1) the EV5-DSW data bus; (2) the IOD bus; (3) or the CIA or the DSW ASIC.

- Once the physical address is obtained, using a show memory command from the console with a MAP of the SIMMS you can isolate the failing FRU. If the bad data came from cache you will have to use a different map to isolate to the Cache SIMM. These maps will be available at a later date.
- An uncorrectable ECC error is deciphered in a similar manner.

ECC Syndromes for Single-Bit Errors

The following table lists the Syndromes for any single bit error whether it is detected by the CIA or the EV5. This data was taken from the DECchip 21164-AA (EV5 CPU) Specification, Revision 1.9.

Table 8-19 EV5 Single Bit Error Syndromes

Data Bit	Syndrome	Data Bit	Syndrome	Check Bit	Syndrome
00	0xCE	32	0x4F	00	0x01
01	0xCB	33	0x4A	01	0x02
02	0xD3	34	0x52	02	0x04
03	0xD5	35	0x54	03	0x08
04	0xD6	36	0x57	04	0x10
05	0xD9	37	0x58	05	0x20
06	0xDA	38	0x5B	06	0x40
07	0xDC	39	0x5D	07	0x80
08	0x23	40	0xA2		
09	0x25	41	0xA4		
10	0x26	42	0xA7		
11	0x29	43	0xA8		
12	0x2A	44	0xAB		
13	0x2C	45	0xAD		
14	0x31	46	0xB0		
15	0x34	47	0xB5		
16	0x0E	48	0x8F		
17	0x0B	49	0x8A		
18	0x13	50	0x92		
19	0x15	51	0x94		
20	0x16	52	0x97		
21	0x19	53	0x98		
22	0x1A	54	0x9B		
23	0x1C	55	0x9D		
24	0xE3	56	0x62		
25	0xE5	57	0x64		
26	0xE6	58	0x67		
27	0xE9	59	0x68		
28	0xEA	60	0x6B		
29	0xEC	61	0x6D		
30	0xF1	62	0x70		
31	0xF4	63	0x75		

Uncorrectable Error Logout Frame

The figure below shows the format of a uncorrectable Error Logout Frame

Field	Offset
FLAGS	0000
OFFSETS	0008
MACHINE CHECK CODE	0010
SHADOW REGISTERS 8-14, 25	0018
PALTEMP REGISTERS 0-23	0058
EXC_ADDR	0118
EXC_SUM	0120
EXC_MASK	0128
PAL_BASE	0130
ISR	0138
ICSR	0140
IC_PERR_STAT	0148
DC_PERR_STAT	0150
VA	0158
MM_STAT	0160
SC_ADDR	0168
SC_STAT	0170
BC_TAG_ADDR	0178
EI_ADDR	0180
FILL_SYN	0188
EI_STAT	0190
LD_LOCK	0198
For spcific error information, refer to Table 8-20	01A0

The figure below shows the AlphaStation 600 system's specific error information in the Uncorrectable Error Log frame.

Figure 8-5 AlphaStation 600 Specific Error Information

Field	Offset
CPU_ERR0	1A0
CPU_ERR1	1A8
CIA_ERR	1B0
CIA_ERR_STAT	1B8
CIA_ERR_MASK	1C0
CIA_SYN	1C8
MEM_ERR0	1D0
MEM_ERR1	1D8
PCI_ERR0	1E0
PCI_ERR1	1E8
NMI_INFO	1F0
PCI_ERR2	1F8

The description of the fields in the uncorrectable error machine check log are as follows:

- FLAGS
  - — This quadword consists of two longword fields where:
    - * Longword 0 is the size of the logout frame in bytes
    - * Longword 1 contains the following flags
      - x bit 63 = 1 means the error is retryable
      - x bit 62 = 1 means a second error has occurred
- OFFSETS
  - — This quadword consists of two longword fields where:
    - * Longword 0 is the offset in bytes to the EV5 specific information
    - * Longword 1 is the offset in bytes to the System (AlphaStation 600 system) specific information
- MACHINE CHECK CODE
  - — This will tell you the type of error that has caused the uncorrectable error.

Table 8-20 Uncorrectable Machine Check Error Codes

Description	Error Code
* Tag Parity Error	0x80
* Tag Control Parity Error	0x82
* Generic Hard Error	0x84
* Processor Detected Uncorrectable ECC error	0x88
* Bugcheck generated by OS specific PALcode	0x8A
* Bugcheck generated by PALcode	0x90
* I-Cache Read retryable Error	0x96
* Processor Detected Hard error	0x98
System Detected Uncorrectable ECC error	0x203
Parity error detected by CIA	0x205
Non-existent Memory Error	0x207
PCI SERR detected	0x209
PCI Data Parity Error detected	0x20b
PCI Address Parity Error detected	0x20d
PCI Master Abort error	0x20f
PCI Target Abort error	0x211
Scatter/Gather PTE invalid error	0x213
Flash ROM Write Error	0x215
IOA Timeout detected	0x217
IOCHK#, EISA add-in board parity error or other catastrophic	0x219
EISA Fail-safe timer time-out	0x21b
EISA Bus time-out	0x21d
EISA Software generated NMI	0x21f
Unexpected EV5 IRQ[3] interrupt	0x221

NOTE:

This table is subject to change as new PALcode is released. All the errors marked by a '*' are included in the current generic EV5 PALcode.

You should also note that all errors that enter PALcode as an interrupt have the low bit set while errors that enter that PALcode as a hardware machine check have the low bit clear. This is done to comply with the standard set by the EV5 PALcode released by Hudson

- Shadow Registers 8-14, 25
  - — This area in the logout frame contains the state of the Shadow registers at the time of error. The EV5 has 8 shadow registers that are available only when in PAL-mode. These will be used rather than the user's registers to reduce the amount of registers that the PALcode needs to save before handling a PALcode entry.
- PALTemp 0 - 23
  - — This area in the logout frame contains the contents of PT0 -PT23 at the time of the error.
- EXC_ADDR
  - — This has the contents of the EV5 EXC_ADDR register at the time of the error. This will tell you where you were executing from at the time of the error. Refer to the DEC Chip 21164-AA specification for details.
- EXC_SUM
  - — This has the contents of the EV5 EXC_SUM register at the time of error. This will tell you the type of arithmetic trap occurred. This will be valid on a machine check due to a arithmetic Trap. Refer to the DEC Chip 21164-AA specification for details.
- EXC_MASK
  - — This has the contents of the EV5 EXC_MASK register at the time of error. This register will log the destination register of an operation that has caused a arithmetic trap. This will be valid on a machine check due to a arithmetic Trap. Refer to the DEC Chip 21164-AA specification for details.
- PAL_BASE
  - — This has the contents of the EV5 PAL_BASE register at the time of error. Refer to the DEC Chip 21164-AA specification for details.
- ISR
  - — This has the contents of the EV5 ISR register at the time of error. This register will give you a summary of all pending interrupts at the time of error. Refer to the DEC Chip 21164-AA specification for details. In the case of a uncorrectable machine check, bit 31 will equal a 1 when it is a system detected uncorrectable error. This bit contains the state of the SYS_MCH_CHK_IRQ_H at the time of the error. In the AlphaStation 600 system, the CIA ASIC will drive this signal on system detected uncorrectable errors.
- ICSR
  - — This has the contents of the EV5 ICSR register at the time of error. This register will give you the current setup of the EV5's IBOX. Refer to the DEC Chip 21164-AA specification for details.

- IC_PERR_STAT
  - This has the contents of the EV5 IC_PERR_STAT register at the time of error. This register will have:
    - bit 11 set to a 1 on a I-cache data parity error
    - bit 12 set to a 1 on a I-cache tag parity error
    - bit 13 set to a 1 on a Timeout reset error or CFAIL_H/no CACK_H error occurred
- DC_PERR_STAT
  - This has the contents of the EV5 DC_PERR_STAT register at the time of error. This register will have:
    - bit 0 set to a 1 if a second error has occurred in a cycle after the register was locked
    - bit 1 set to a 1 once the error bits 2:5 are locked into this register. If this bit is clear then the state of bits 2:5 are meaningless.
    - bit 2 set to a 1 on a data parity error in D-cache bank 0
    - bit 3 set to a 1 on a data parity error in D-cache bank 1
    - bit 4 set to a 1 on a tag parity error in D-cache bank 0
    - bit 5 set to a 1 on a tag parity error in D-cache bank 1
- VA
  - This has the contents of the EV5 VA register at the time of error. This will contain the effective virtual address associated with D-stream faults, DTB misses, or D-cache parity errors.
- MM_STAT
  - This has the contents of the EV5 MM_STAT register at the time of error. This register will help determine the reason for a D-stream Fault or D-Cache parity error. This register will have:
    - bit 0 set to a 1 if the failing reference was a write
    - bit 1 set to a 1 if the reference caused an Access violation
    - bit 2 set to a 1 if the reference was a read and the PTE's Fault on Read bit was set
    - bit 3 set to a 1 if the reference was a write and the PTE's Fault of Write bit was set
    - bit 4 set to a 1 if the reference resulted in a DTB Miss
    - bit 5 set to a 1 if the reference had a bad virtual address
    - bits 10:6 contain the Ra field of the faulting instruction
    - bits 16:11 contain the Opcode field of the failing instruction
- SC_ADDR
  - This has the Address that was being accessed when a failure was detected in the EV5's Secondary cache.

- SC_STAT
  - This has the contents of the EV5 SC_STAT register at the time of error. This register will help determine whether the error was caused by a tag or data parity error in the secondary cache. This register contains the following:
    - bits 2:0 will contain a 1 in the bit corresponding to the set that was being probed when a Scache Tag parity error occurred
    - bits 10:3 will tell which longword within the two octawords in a SCache block has an error ... bit 3 denotes Longword 0, bit 4 denotes Longword 1 and so on.
    - bits 15:11 indicates the SCache transaction which caused the error, where:
      - 1x110 means Set Shared from system
      - 1x101 means Read Dirty from system
      - 1x100 means Invalidate from system
      - 1x001 means SCache victim
      - 00001 means SCache I-Read
      - 01001 means SCache D-Read
      - 01011 means SCache D-Write
    - bit 16 will be set to a 1 if another error occurs after this register is locked.
- BC_TAG_ADDR
  - This has the contents of the EV5 BC_TAG_ADDR register at the time of error. This register contains the following:
    - bit 12 is set to a 1 on a BCache hit
    - bit 13 contains the Tag Control parity
    - bit 14 contains the Tag Control dirty bit
    - bit 15 contains the Tag Control shared bit
    - bit 16 contains the Tag Control valid bit
    - bit 17 contains the Tag parity
    - bits 38:20 contain the BCache Tag
- EI_ADDR
  - This has the contents of the EV5 EI_ADDR register at the time of error. This will contain the physical address of any transfer that is logged in the EV5 EI_STAT register. This register contains the following data:
    - bits 3:0 are always read as a one
    - bits 39:4 contain bits 39:4 of the failing address
- FILL_SYN
  - This has the contents of the EV5 FILL_SYN register at the time of error. This register should not contain a syndrome for a single bit error in a large frame, due to the fact that only double bit errors should be logged in a uncorrectable error logout frame.

- EI_STAT

  — This has the contents of the EV5 EI_STAT register at the time of error. This will help identify the reason for any type of processor detected uncorrectable errors at its external interface. This register will contain the following:

  * bits 27:0 are always read as a one
  * bit 28 is set to a 1 on a BCache Tag parity error
  * bit 29 is set to a 1 on a BCache Tag control parity error
  * bit 30 = 1 means the error source is the memory subsystem
    = 0 means the error source is the BCache subsystem
  * bit 31 is set to a 1 on a Correctable ECC error. **This should not be set in a uncorrectable machine check logout frame unless a single bit error is detected at the same time as a uncorrectable error occurs**
  * bit 32 is set to a 1 on a Uncorrectable ECC error.
  * bit 33 is set to a 1 when an address or command received by the EV5 has a parity error.
  * bit 34 = 1 when the error occurred during an ICache fill
    = 0 when the error occurred during a DCache fill
  * bit 35 is set to a 1 if an second error occurs after this register has been locked because of a previous error.

- LD_LOCK

  — This has the contents of the EV5 LD_LOCK register at the time of error.

- CPU_ERR0

  — This has the contents of the CIA ASIC's CPU_ERR0 register at the time of error. This will contain the low order address bits (EV5 ADDR_H <31:4>) at the time of error.

- CPU_ERR1

  — This has the contents of the CIA ASIC's CPU_ERR1 register at the time of error. This will contain the high address bits (EV5 ADDR_H <39:32>) at the time of error as well as other information. Refer to the chapter on Control Registers for details.

- CIA_ERR

  — This has the contents of the CIA ASIC's CIA_ERR register at the time of error. Refer to the chapter on Control Registers for details.

- CIA_ERR_STAT

  — This has the contents of the CIA ASIC's CIA_ERR_STAT register at the time of error. Refer to the chapter on Control Registers for details.

- CIA_ERR_MASK

  — This has the contents of the CIA ASIC's CIA_ERR_MASK register at the time of error. Refer to the chapter on Control Registers for details.

- CIA_SYN

  — This has the contents of the CIA ASIC's CIA_SYN register at the time of error. Refer to the chapter on Control Registers for details.

- MEM_ERR0
  — This has the contents of the CIA ASIC's CIA_MEM0 register at the time of error. This will contain the low address bits (MEM_ADDR_H<31:4>) at the time of error. Refer to the chapter on Control Registers for details.
- MEM_ERR1
  — This has the contents of the CIA ASIC's CIA_MEM1 register at the time of error. Refer to the chapter on Control Registers for details.
- PCI_ERR0
  — This has the contents of the CIA ASIC's PCI_ERR0 register at the time of error. Refer to the chapter on Control Registers for details.
- PCI_ERR1
  — This has the contents of the CIA ASIC's PCI_ERR1 register at the time of error. Refer to the chapter on Control Registers for details.
- NMI INFO
  — This has the contents of the EISA bridge NMI status and control register (61h) in the low longword and the contents of the EISA bridge NMI Extended status and control register (461h) in the high longword.

Deciphering an Uncorrectable Error Logout Frame

If you want to determine the cause of an uncorrectable error logout frame , you should use the following flow.

- Get the Machine Check Error Code
- IF the Error code > 0x201 THEN (system detected uncorrectable error)
  — Use the Section called "**CIA detected errors"** in this chapter to help isolate the failure. It contains the information needed to help isolate the error such as:
    - * Which bits should be set in the CIA_ERR register for the particular error
    - * What the relevant CIA error registers corresponding the the error
    - * Who Detected the Error?
    - * How the error is reported to the CPU?
- ELSE IF the Error code < 0x200 THEN (Processor detected uncorrectable error)
  — Use the machine check code to determine the type of error that has occurred.
  — The following are the internal registers that are used to determine the type of error.
    - * The EI_STAT register contains bits for:
      - x B-Cache Tag Parity Error
      - x B-Cache Tag Control Parity Error
      - x External interface source , used to isolate ECC errors between B-cache and memory
      - x Uncorrectable ECC error
      - x External Interface parity error
      - x FILL IRD bit that is used to determine whether the ECC error was detected on an I-stream or D-stream fill
      - x Second Error Occurred bit
    - * The EI_ADDR register contains the physical address associated with errors reported in the EI_STAT register

* The SC_STAT register contains bits that will allow you to:
  - x isolate the set on a S-cache Tag parity error
  - x isolate the failing longword on a S-Cache Data Parity Error
  - x log the S-Cache transaction at the time of error
* The SC_ADDR will contain the physical address associated with errors reported in the SC_STAT register.
* The IC_PERR_STAT contains information about an I-Cache parity error
* The DC_PERR_STAT contains information about a D-cache parity error.

Please Refer to the PALcode/IPR and Error Handling chapters in the DEC chip 21164-AA Specification for more details on the using the EV5 registers that are logged in the machine check logout frame.

9

AlphaStation 600 System Initialization

Introduction

This chapter will describe the initialization performed by the system firmware at power-up. The firmware will be split between two types of ROMS. A serial ROM that interfaces directly with the EV5 CPU and One Megabyte of Flash ROM that resides in the GRU ASIC's CSR space. The One Megabyte of flash ROM is split up so 512 Kbytes are used for the SRM Console and 512 Kbytes are used for the ARC console. This chapter discusses the initialization performed by the Serial ROM and the SRM console.

This chapter is broken up into two sections:

- Serial ROM performed initialization
- System ROM (Flash ROM) performed initialization

Serial ROM Performed Initialization

The Serial ROM firmware is responsible for the following initialization steps:

- Initialize the EV5 CPU
- Test the interface to the EISA Bridge chipset
  — The Serial ROM code will test the interface to the EISA Bridge chipset by first determining that the device has been found. Then, a datapath test will be performed by longword writes and reads using PCI Configuration address space.
- Test and Initialize to the OCP device
- Size and Configure the Third Level Cache
  — The Serial ROM code will handle any supported third level cache size. This will provide a single Serial ROM image that supports many different system variations..
  — The Serial ROM code will configure a third level cache size equal to 0, if it finds Cache SIMMS of different sizes.
- Size and Configure the MMB types
  — The Serial ROM code will handle any supported MMB type. The MMB type is needed so the SROM code can determine the data path width to memory (128/256 bit). This flexibility will allow the AlphaStation 600 system to work with either:
    * One 128 bit MMB (128 bit data path)
    * Two 128 bit MMB (256 bit data path)

* Two 64 bit MMB (128 bit data path)

- Size and Configure the Main Memory
  - — The Serial ROM code will handle any combination of memory sizes and speeds as long as a SIMM is present in all slots within a bank.
  - — The Serial ROM code will use the smallest SIMM size and the slowest speed to determine memory size and the speed of the memory for the bank.
  - — A normally configured bank should have all SIMMS of equal size and speed, but we are building this flexibility in to allow for a better chance of having a fail-safe power-up.
- Test and Initialize Cache and Memory
  - — The Serial ROM code will initialize and test 32 megabytes of memory.
  - — The Serial ROM code will initialize the memory tested with 0's
- Determine which console has to be loaded by reading the console type variable from the TOY/NVR chip.
- Test the interface to the system Flash ROMS
  - — The Serial ROM code will test the interface to the System Flash ROMS by trying to read the manufacturing header data from the ROMs.
- Load the Desired Console firmware from Flash ROM to RAM
  - — The code will be loaded to address 0x100000
  - — The Serial ROM will then program the CIA_CTRL register to disable the capability to write to the Flash ROM. This will be done to protect the flash ROM from any inadvertent writes to its address space that may corrupt its contents.

The Serial ROM code will not initialize any hardware that resides off the PCI interface on the CIA ASIC other than the PCI/EISA bridge chipset and the datapath to the OCP device. PCI initialization will be performed by the Console firmware. The following table is a summary of the EV5 registers and the AlphaStation 600 system CSR's that are initialized by the Serial ROM code.

Table 9-1 Summary of SROM Initialization

Name	Value	Description
IBOX_CSR (EV5)		EV5 IBOX CSR ...SROM code will disable floating point, enable PAL shadow registers, and allow PALRES instructions to be issued from kernel mode
HWINT_CLR (EV5)	0x0	clear the edge sensitive interrupts for performance counters,serial line, and CRD's
DC_MODE (EV5)	0x1	Enable the D-Cache with parity checking enabled.
SC_CTL (EV5)	0xF000	Set the secondary cache block size to 64 bytes and enable all three sets in the second level cache.
BC_CONTROL (EV5)	0x61	The Third level cache will be enabled (if found), ECC mode selected, no Victim buffer enabled, and machine checks enabled
BC_CONFIG (EV5)	Config Specific	The state of this register is totally dependent on the size and speed of the cache found.
MBA Registers 0-15 (CIA)	Config Specific	The Serial ROM code will program in the base address of the equivalent bank. The banks with larger SIMM sizes will be placed in low memory. If no memory is found in that bank a value of 0 will be written to the register to make that bank invalid.
Timing Registers 0-2 (CIA)	Config Specific	The Serial ROM code will program these register so they correspond with the slow, medium and fast timing.
MCR Register (CIA)	Config Specific	The Serial ROM code will program this register to the appropriate data path size, third level cache size, and refresh state. The Serial ROM code will set the INVALID MEM bits if it finds any mismatched SIMMS within a set.

System Firmware Performed Initialization

The code that resides in the Flash ROMS consist of :

- PALcode
- SRM console code, which consists of:
  - a Kernel
  - Console Terminal drivers
  - Console Boot drivers
  - Various Port drivers
  - ALPHA SRM compliant console functionality
  - Minimal Power-up Self Test
- X86 Emulator

This code will perform most of the initialization and device configuration for devices that sit off the PCI bus on the AlphaStation 600 system. The initialization is broken up into the following tasks:

- Sizing the PCI bus
- Configuring the PCI bus
- Initialization of the following devices ...
  - PCI / EISA Bridge Chipset
  - 87312 devices
    - COM1 port (ISA)
    - COM2 port (ISA)
    - Centronix Parallel Port (ISA)
    - Floppy controller (ISA)
  - 8242 Keyboard/Mouse chip (ISA)
  - 1225 Battery Backed-up SRAM chip (ISA)
  - 1287 TOY clock chip (ISA)
  - Any NCR 53C810 SCSI chips (PCI)
  - Any QLOGIC ISP1020 SCSI Chips (PCI)
  - Any Tulip Ethernet chips (PCI)
  - Graphics device used for Console (PCI)
  - Any other Device that has a dynamically linked-in driver present
  - Initialization of any option with X86 code, if it is being used as a console device or a boot device.

Overview of Sizing the PCI Bus

The PCI bus will be sized by the PCI bus driver's initialize function. This driver will be one of the first driver's started up on console entry. This has to be done before we start any of the port driver's on the PCI or EISA bus. The most important results of the configuration are :

- a PCI-EISA bridge chip at ID 0
  - It is a **Fatal** error, if we don't find this chip. Our console input device lives on the other side of this chip on an ISA controller.
- at least one I/O Subsystem board in one of the remaining PCI ID's
  - It is a **Fatal** error, if we don't find one of these modules. We can still attempt to bring up the console program, but we will not be able to perform any boots through the embedded NCR 53C810, QLOGIC ISP1020 or TULIP device drivers.
- Display device(s), Base Class 03, in one or more PCI slots
  - A base class 3 device is a display device, the lack of a display device on the PCI, as well as the EISA, will cause the console to use the COM1 port as the console device.

Sizing/Configuring the PCI Bus

The following work will be performed by the PCI bus driver initialization code:

- Size the primary bus
  - — This is done by performing Type 0 configuration cycles to ID 0-5 on the primary bus.
  - — If a PCI to PCI bridge chip is found, its appropriate secondary bus is sized-down before proceeding to the next ID on the primary bus.
    - * This will be done by performing Type 1 configuration cycles with the bus equal to the secondary bus number corresponding to the bridge.
    - * If a subsequent bridge chip is found on the secondary bus, it its secondary bus is sized-down before we proceed down the initial secondary bus. The PCI-PCI bridge chip that is being used on the AlphaStation 600 system will support, at most, two levels of bridging.
  - — The following activities must be performed on all devices that have been found:
    - * Read and save vendor ID
    - * Read and save device ID
    - * Read and save version ID
    - * Set up any associated I/O mapped Base addresses with current Free I/O mapped address
    - * Update current free I/O mapped address to allow for enough address space for this device
    - * Set up any associated Memory mapped Base addresses with current Free Memory mapped address
    - * Update current free Memory mapped Base address with current Free Memory mapped address

Accessing PCI Bus Configuration

The result of this PCI sizing will be a configured PCI bus.

The current configuration can be determined by reading the current setup of the configuration registers for each device on the bus. This is similar to the mechanism used on other platforms.

PCI-PCI Bridge Configuration

The PCI bus driver will configure any PCI-PCI bridge found in the following way:

Table 9-2 PCI-PCI Bridge Initialization

Name	Value	Description
cache_line_size	4	number of 32 bit cache entities in a cache line
latency_timer		Controls time-out of P_FRAME_L signal
Primary Bus	0	Used to allow TYPE 1 configuration cycles initiated by the secondary bus
Secondary bus	x	The bus number of the secondary bus on this bridge
Subordinate bus	y	This is used in case there are PCI-bridge chips behind this bridge chip where y is the highest bus number that resides behind this bridge
Secondary MLT		Master Latency value for the secondary bus
Memory Base Address		Low address in memory space that this bridge will decode. Value must be 1 MB aligned,. This value is totally dependent on what devices are in the system.
Memory Limit Address		High Address in Memory space that this bridge will decode. Value must be 1 MB aligned. This value is totally dependent on what devices are in the system.
I/O base Address		Low address in I/O space that this bridge will decode. This value is totally dependent on what devices are in the system.
I/O Limit Address		High Address in I/O space that this bridge will decode. This value is totally dependent on the devices that are in the system.

Driver Initiated PCI Configuration

Once the PCI bus sizing and configuration is complete, the device drivers for the known PCI, EISA and ISA devices,as well as the EISA bus driver, can be started. It will be up to each driver to determine if one or more applicable device(s) exist in the current system configuration.

NOTE:

This will allow us to support up to 5 PCI I/O boards in the system at the same time using the same set of drivers.

The console will support this driver based sizing by providing routines to scan the system configuration table for specific **device** and **vendor ids**.

The device drivers will have the ability to fill in the non-Base Address portion of the PCI configuration space. The system firmware will contain at least the following PCI drivers that will configure their respective controllers.

- TULIP DRIVER
  - — This will not need to do any initialization above what is done by the PCI bus driver.

- NCR 53C810 DRIVER
  - This will initialize any non-standard field in the NCR810 configuration register space
- QLOGIC ISP1020 DRIVER
  - This will initialize any non-standard field in the ISP1020's configuration register space
- VGA DRIVER
  - This will initialize any non-standard field in a VGA type device's configuration space.
- Dynamically linked-in PCI drivers
  - These drivers need to initialize their respective configuration registers with the appropriate data.

PCI-EISA Bridge Configuration

The PCI-EISA Bridge chip will be configured by the AlphaStation 600 system initialization code. It will be initialized as shown in Table 9-3.

Table 9-3 PCI-EISA Bridge Chip Initialization

Name	Value	Description
Mode Select	0x40	SERR enabled, 8 EISA masters, Configuration RAM access enabled
BIOS Chip Select	0x0000	Disable Access to motherboard BIOS
Peripheral Chip Select A	0x07	Enable RTC decode, Keyboard Controller decode, and Floppy Disk decode
Peripheral Chip Select B	0xc4	CRAM decode, Port 92 enabled, COM1 at 3f8-F, COM2 at 2f8-f, LPT at 3bc-f
EISA Motherboard ID		The AlphaStation 600 system value for EISA motherboard ID
General Purpose Chip Select 0	0x530	Set up IIC controllers to live at address 530h
General Purpose Chip Select Mask0	0x1	Select address 530-1h as legal addresses for the IIC controller
General Purpose X-bus Control	0x1	General Purpose Chip select 0 is enabled, GPCS 1-2 are disabled
NMI status and control register	0x4	Clear any system board NMI errors
PCEB Master LAntency Timer	0xf8	Sets up MLT to be a its maximium count value. This corresponds to 248 PCI clocks
PCEB EISA to Mem attributes	0x1	Sets memory region 1 to be buffered
EISA-PCI MEMORY region 1	0xffff0000	Setup memory region 1 to map to full 4 Gigabyte PCI memory space
Internal 8259's		Setup up to enable IRQ 0, 1, 2, 5, 9 and 14. The two controllers are cascaded together

COM1 (87312) Device Initialization

The device driver will put the COM1 port in the following state:

- 9600 baud
- 8 bit character,1 stop bit, No parity
- Initialize the modem bits

COM2 (87312) Device Initialization

The device driver will put the COM1 port in the following state:

- 9600 baud
- 8 bit character,1 stop bit, No parity
- Initialize the modem bits

Parallel Port (87312) Device Initialization

The device driver will put this port in the following state:

- Line printer will be reset
- Printer will be put on-line with line feed at end of line enabled

Floppy Controller (87312) Device Initialization

The device driver will perform the following initialization, if the device is opened for access:

- Initializes the Floppy controller
- Sets up the data rate
- Spins up the floppy device
- Recalibrate the drive to track 0
- Attempts to read sector 0
- IF a successful read is performed THEN
  - — a unit block for this device is initialized and put onto the queue
- ELSE
  - — The floppy is spun down

If the floppy device is not opened, then No initialization is performed.

Keyboard/Mouse Device Initialization

The Device driver will perform the following initialization:

- Flush the input buffer
- Select PS/2 mode
- Enable and test the keyboard interface chip
- Determine whether a 84 key or 101 key keyboard is present

Battery Backed SRAM Device Initialization

The Device driver will perform the following initialization.

- Update any changes that are necessary to ARC configuration tree. This will only need to be done if the configuration has changed since the last power-up.

The battery backed up SRAM will be used for:

- Environment variables (2Kbyte)
  - — The AlphaStation 600 system will use the same utilization as the MORGAN console
- the ARC configuration tree (6Kbyte)

TOY Driver Device Initialization

The Device Driver will perform the following initialization:

- Initialize CSRA to:
  - — Time Divisor base = 32.768Khz
  - — Rate Select = 976.562 us
- Initialize CSRB to:
  - — 24 hour mode enabled
  - — Periodic Interrupts enabled
  - — DM bit enabled

NCR810 Driver Device Initialization

The Device Driver will perform the following initialization:

- TBD.

TULIP Driver Device Initialization

The Device Driver will perform the following initialization:

- Read and check the station address ROM
- Initialize Transmit Descriptors
- Initialize Receive Descriptors
- Initialize Setup,which consists of:
  - — Building a perfect filter
  - — Starting up a Transmit process
  - — Send out the startup frame
- Start up a Receive Process (packets can now be received off the wire)

TGA Driver Device Initialization

The Device Driver will perform the following initialization:

- TBD.

VGA Driver Device Initialization

The Device Driver will perform the following initialization:

- TBD.

Memory Initialization

The Console program will start up a task that runs in the background process to initialize the system memory. This process will be stopped once control is passed to the loaded code or the initialization is complete (If you are auto-booting, you cannot depend on all of memory to be initialized).

- All memory will be initialized to 0.

Figure 9-1 is a memory map after the console is started up.

Figure 9-1 AlphaStation 600 Memory Map After Initialization

0x0000
Unused
0x1C00
SROM Mailbox
0x2000
HWRPB
0x5000
PAL Impure Area
0x6000
Machine Check Logout Frame(s)
0xC000
Flash ROM based code
Rest of Tested memory
0x2000000
Initialized /Untested Memory
Top of Memory

Miscellaneous CIA Related Initialization

The console program will initialize miscellaneous CSR's in the CIA during the initial entry into console that were not initialized by the serial ROM. The serial ROM will mainly initialize the Memory controller registers while the console code is initializing the rest of the CIA's CSR's. The following table lists the CSR's that are initialized by the Flash ROM-based code as a result of a powerup or a "init" command.

Table 9-4 CIA Main CSR Register Initialization

Name	Value	Description
SG_TBI	0x3	Invalidate the Scatter Gather Translation buffers
PCI_CNFG	0x0	Force next PCI configuration access to be a type 0 access
HAE_MEM	0x2028	All three PCI memory regions will be contiguous to each other
HAE_IO	0x2000000	The two PCI I/O regions will be contiguous to each other, so that the 1st address of region b follows the last address of region a

Table 9-5 CIA Memory CSR initialization

Name	Value	Description
MCR	Not written	Relies on value written by the SROM
MBA 0-15	Not written	Relies on value written by the SROM
MEM_TMG 0-2	Not written	Relies on value written by the SROM

Table 9-6 CIA Physical Address Translation CSR Initialization

Name	Value	Description
Window Base Register 0	0x800000	Window 0 enabled with a base address of 0x800000 and scatter/gather enabled. This window will be used by an ISA device so it can access all of configured memory.
Window Base Register 1	0x40000000	Window 1 enabled with a base address 0f 0x40000000 (1 gigabyte) scatter/gather disabled. This window can be used by EISA and PCI devices to DMA into memory
Window Base Register 2-3	0x0	The windows will be disabled
Window Mask Register 0	0x700000	This will set up a 8 Mbyte DMA window that can be used for ISA devices
Window Mask Register 1	0x3ff00000	This will setup a window size of 1 Gigabyte
Window Mask Register 2-3	0x0	These registers will not be used since they are disabled
Translation Base register 0	0xa000	This will map window 0's Scatter/Gather table into address 0xA000. **This is in console space ... This might need to be moved if this is an issue**
Translation Base register 1	0x0	This will map PCI Window 1 to start at address 0 in memory and it will continue up to address 1 gigabyte.
Translation Base register 2-3	0x0	These registers will not be used since they are disabled

Table 9-7 CIA Error CSR initialization

Name	Value	Description
CIA_ERR	0xffff	This will dismiss any pending Error as well as unlock all the error registers in the CIA
ERR_MASK	0xffff	All Error logic will be ENABLED

Address Map

Beyond any ISA devices that reside on the EISA bus, there are no fixed addresses that are assigned on a per ID basis. The console configuration must be used to determine:

- The type of devices in the system
- The location of devices in the system

Certain addresses could possibly be assigned for each ID, but this is not recommended. It could become unsupportable, with the possibility of bridges and devices that take an unspecified amount of I/O and memory space behind the bridge.

The location of a device in the CPU's address space is also a function of the type of access that you want to make to that device. Please refer to Chapter 3, AlphaStation 600 Addressing, for a complete description.

ISA Devices Address Map

Table 9-8 describes where some of the I/O devices will reside in PCI Sparse I/O Space region A. These are the address that you would use to do byte accesses to these registers.

Table 9-8 ISA device Address Map (Sparse IO Space)

Name	ISA Address	PCI Address	Description
PRI	0x3f0-0x3f7	0x85.80007e00- 0x85.80007ee0	Primary Floppy Device
COM1	0x3f8-0x3ff	0x85.80007f00- 0x85.80007fe0	Serial Port #1
COM2	0x2f8-0x2ff	0x85.80005f00- 0x85.80005fe0	Serial Port #2
LPT1	0x3bc-0x3b8	0x85.80007780- 0x85.800077e0	Parallel Port
KBD_DATA KBD_CMD/STAT	0x60 0x64	0x85.80000c00 0x85.80000c80	Keyboard/Mouse controller
RTC_OFFSET, RTC_DATA	0x70 0x71	0x85.80000e00 0x85.80000e20	Real Time Clock Chip

The following algorithm was used to convert the ISA address into the correct EV5 generated address for byte accesses to PCI sparse IO space.

```
#define pci_sparse_io_rega_byte(x) (0x8580000000 | (x<<5))
```

The address is basically shifted left by 5 bits and OR'd with the base address of PCI sparse memory space. See the Chapter 3, AlphaStation 600 Addressing, for a complete description on address mapping.

Software Considerations

The following conditions need to be considered due to the flexibility of the placement of PCI addresses

- If software modifies any of the configuration base addresses set up by the console, this will cause the console callbacks as well as the console program itself to have unpredictable results. This is due to the fact that the console software expects to find the particular controllers where they were last left.
- The powerup initialization code is used to give the hardware to the operating systems in a known/friendly state. This does not mean that the Operating System Driver's don't need to initialize the hardware. It means that you can expect to get the hardware in a controlled state. The Operating system code should not be written in such a way where it assumes:
  - — specifics on how the controller is initialized after leaving console, that is, If a particular bug exists in the hardware ,we may have to run the controller in a special way on early hardware. This will be changed once the hardware works as expected.
  - — memory is initialized to a particular value.

10

AlphaStation 600 PCI-EISA Bridge

Introduction

The PCI-EISA chip bridge set consists of the PCI to EISA bridge chip, the PCEB (82375EB); and the EISA system controller, the ESC (82374EB). These chips are essentially a collection of peripheral chip designs (such as interrupt logic, timers, DMA control, arbiters, etc.) coerced into two packages. The objective of this chapter is to define which of these component parts the AlphaStation 600 system design uses, and to provide suggestions and requirements in the programming of the internal registers.

Figure 10-1 shows the AlphaStation 600 system standard I/O busses and devices, and indicates how the PCI-EISA bridge is used and what devices are attached to the X-bus. Tables 10-2 and 10-3 tabulate the functional components of the two chips and define whether this logic is used in the AlphaStation 600 system.

Related Documentation

For more detailed information on the PCI-EISA bridge chips, the following specifications are mandatory.

Table 10-1 Documentation

Documentation		
82420/82430 PCI set ISA and EISA Bridges	Intel	1993
82375EB(PCEB) A-2 and 82374EB (ESC) A-2 Stepping Info. Rev 2.0 *This document describes changes and bug fixes.*	Intel	Jan 31, 1994

Figure 10-1 AlphaStation 600 Standard I/O Busses

Speaker
EISA slots
Floppy Drive
COMBO Chip 87312
Serial Port
Serial Port
Parallel Port
EISA Interrupts
PCI-EISA Bridge
PCI arb.
EISA-bus
buffer
X-bus
PCI bus
EISA Config RAM
Real Time Clock DS1287
Keyboard & mouse 8242
Operator Ctrl Panel PCD8584
CIA
Memory Presence Detect Logic
PCI Interrupt Logic
756 KB FLASH RAM
IOD bus
buffer
Lemmon Bus
GRU asic
Data Switch

Table 10-2 ESC chip -- AlphaStation 600 System Requirements

PCEB functionality	Associated signal pins		AlphaStation 600 usage
PCI Bus interface	PCICLK	PCI clock	Yes
	PERR# SERR#	PCI parity and system error	**No.** See Chapter 8, AlphaStation 600 Hardware Exceptions and Interrupts, for more information
	RESET#	PCI reset	Yes
ESC-PCEB interface	EISAHOLD EISAHLDA PEREQ# NMFLUSH SDCPYEN SDCPYUP SDOE# SDLE# INTCHPIO	Private interface	Yes
Timers	SPKR SLOWH#	Speaker Slow CPU	**Yes** **No** -- 486 signal
Interrupt Controller	IRQ's INT NMI	EISA IRQ's Interrupt out NMI	**Yes.** See Chapter 8, AlphaStation 600 Hardware Exceptions and Interrupts, for more information
EISA bus interface	BCLKOUT, BCLK LA[31:2] BE[3:0]# M/IO# W/R EX32# EX16# START# CMD# EXRDY SLBURST# MSBURST# MASTER16# SD[7:0]	Standard EISA bus signals	Yes
ISA bus interface	BALE SA[1:0] SBHE# M16# IO16# MRDC# MWTC# SMRDC# SMWTC# IORC# IOWC# CHRDY IOCHK# NOWS# OSC RFRESH# RSTDRV AEN#	Standard ISA bus signals	Yes

Table 10-2 ESC chip -- AlphaStation 600 Requirements (continued)

PCEB functionality	Associated signal pins		AlphaStation 600 usage
EISA arbiter	MREQ[7:0] MACK[3:0] EMACK[3:0]	Master Req. Master Ack Encoded Ack	**Yes** The AlphaStation 600 system supports 4 EISA slots
DMA controller	DREQ[7:5,3:0] DACK[7:5,3:0] EOP	DMA request DMA ack End of Process	**Yes** This is standard EISA
EISA address buffer control	SALE# LASAOE# SALAOE#	SA latch en. LA to SA en. SA to LA en.	**Yes** This is standard EISA
Coprocessor interface	FERR# IGNNE#	Coproc. err Ignore err	**No** This is for the x86 CPU.
BIOS interface	LBIOSCS#	Latched BIOS Chip select	**No** The AlphaStation 600 system does not support a Flash RAM on the X-bus. Instead, it is on the Lemmon bus.
Keyboard Controller	KYBDCS#	Kybd chip sel.	**Yes** The AlphaStation 600 keyboard controller (8242) is on the system board.
	ALTRST# ALTA20	Alternate reset Alternate A20	**No** This is for the x86 CPU.
	ABFULL	Aux Buffer full	**No** This signal indicates that the 8242 Mouse buffer is full. In the AlphaStation 600 system, the mouse interrupt is connected to IRQ<12> externally (software must clear reg. 4Dh bit 4).
Real Time Clock	RTCALE RTCRD# RTCWR#	RTC addr EN. RTC read RTC write	**Yes** The AlphaStation 600 TOY (Dallas DS1287) is on the system board.
Floppy Disk controller	FDCCS# DSKCHG DLIGHT	Floppy chip sel Disk change Disk act. light	**No** The AlphaStation 600 floppy disk controller is on the SIO chip
Configuration RAM	CRAMRD# CRAMWR#	Conf. RAM read and write	**Yes** AlphaStation 600 system supports the EISA non-volatile configuration RAM on the X-bus.
Xbus control	XBUSTR# XBUSOE#	transmit/receive Data output en.	**Yes** The AlphaStation 600 system uses the X-bus
General purpose chip select	GPCS[2:0]#	Gen. Purpose chip select	**Yes (GPCS[0} only).** The Operator Control panel chip is selected by GPCS[0].
Test	TEST	Chip test	**No**

Table 10-3 PCEB chip - AlphaStation 600 Requirements

PCEB functionality	Associated signal pins		AlphaStation 600 usage
PCI bus Interface	PCICLK PCIRST# AD[31:0] C/BE[3:0]# FRAME# TRDY# IRDY# STOP# PLOCK# IDSEL DEVSEL# PAR PERR#	Standard PCI bus signals	**Yes**
ESC-PCEB interface	EISAHOLD EISAHLDA PEREQ# NMFLUSH SDCPYEN SDCPYUP SDOE# SDLE# INTCHPIO	Private interface	**Yes**
PCI Address Decode	MEMCS# PIODEC#	Mem chip sel PCI IO space	**Yes** Used by the CIA for PC hole decode **No**
PCI Arbiter	CPUREQ# CPUGNT# REQ[3:0]# GNT[3:0]#	CPU req CPU ack Slot request Slot ack	Yes The PCEB does NOT support an external PCI arbiter. Since the AlphaStation 600 sytem has one more PCI slot than the PCEB can handle, a sub-arbiter is provided on the systemboard.
	FLSHREQ# MEMACK#	Flush req Mem ackn	**Yes** **Yes** See the Coherency section of this chapter for more details.
	MEMREQ#	Mem request	**No**
(E)ISA interface	BCLK START CMD# M/IO# W/R EX32# EX16# SLBURST# MSBURST# LOCK# BE[3:0]# LA[31:2] REFRESH# SD[7:0] IO16#	Standard EISA bus signals	**Yes**
Test	TEST		**No**

ESC Functionality

The stepping (revision) of the chip which the AlphaStation 600 system will use at power-up is A-2. The AlphaStation 600 system's use of the ESC chip is almost a standard implementation. Some of the AlphaStation 600 system's specific details/features are:

- **Keyboard controller:** This is based on the standard 8242. The mouse interrupt (the so called ABFULL# signal) is **not** wired to the ABFULL# input. Software should disable its path to IRQ<12> via the ESC register CLKDIV bit<4>.
- **SERR and PERR:** Software must **NOT** enable these inputs to the NMI logic in the ESC.
- **Interrupt logic:** This logic is used for the EISA interrupts. The ESC should disable the connection of ABFULL to IRQ<12>. The Real Time Clock is **not** wired to this interrupt logic; instead it is wired directly to an EV5 interrupt pin. See Chapter 8, Hardware Exceptions and Interrupts, for more details.
- **TOY - Real time clock:** This is based on the Dallas 1287 and is wired to the X-bus in the conventional manner.
- **Configuration RAM:** 8 KB of non-volatile RAM is provided on the X-bus for the EISA configuration space.
- **BIOS:** There is **NO** BIOS ROM on the X-bus. Instead, the AlphaStation 600 system provides Flash ROM on the Lemmon bus. The Lemmon bus is "closer" to the CPU and thus a preferred location for this RAM (that is, serial-ROM code need check less of the system before accessing the Flash RAM code).
- **Speaker:** This is provided in addition to the Audio card for two reasons: first, some O/S may not support the Audio card; and second, the user may have headphones connected to the audio card, which are not currently being worn.
- **General Purpose device**. The Operator Control Panel interface chip (PCD8584) is wired to GPCS[0] -- the GPCS[0] signal is gated with CMD_L to allow consecutive access of the PCD8584 -- that is, software does not need to do anything special.

ESC Registers

Table 10-4 lists the ESC registers and, where applicable, describes the AlphaStation 600 system programming requirement.

Table 10-4 ESC Registers

<table>
<tr><th>Category</th><th colspan="2">ESC register</th><th>AlphaStation 600 requirements</th></tr>
<tr><td rowspan="14">ESC Configuration Registers</td><td>ESCID</td><td>ESC Identity</td><td rowspan="2">No special AlphaStation 600 system requirement</td></tr>
<tr><td>RID</td><td>Revision ID</td></tr>
<tr><td>MS</td><td>Mode select</td><td>
<table>
<tr><th>Bit</th><th>Value</th><th>AlphaStation 600 requirement</th></tr>
<tr><td>1:0</td><td>11</td><td>8 EISA masters (AEN encoded) (AEN unencoded does not work -- undocumented bug in ESC chip)</td></tr>
<tr><td>2</td><td>1</td><td>PIRQ mapped to MREQ (new encoding)</td></tr>
<tr><td>3</td><td>0</td><td>SERR# disabled</td></tr>
<tr><td>4</td><td>0</td><td>GPCS pins enabled</td></tr>
<tr><td>5</td><td>1</td><td>Config. RAM enabled</td></tr>
<tr><td>6</td><td>1</td><td>MREQ enabled (new encoding)</td></tr>
<tr><td>7</td><td></td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>BIOSCSA</td><td rowspan="2">BIOS chip sel registers</td><td rowspan="2">Set both registers = 0
AlphaStation 600 does not support BIOS.</td></tr>
<tr><td>BIOSCSB</td></tr>
<tr><td>CLKDIV</td><td>Clk divisor</td><td>
<table>
<tr><th>Bit</th><th>Value</th><th>AlphaStation 600 requirement</th></tr>
<tr><td>2:0</td><td>0</td><td>Clock Divisor = 4 (33.3 MHz)</td></tr>
<tr><td>3</td><td>0</td><td>KBFULL</td></tr>
<tr><td>4</td><td>0</td><td>ABFULL</td></tr>
<tr><td>5</td><td>0</td><td>Coprocessor Error disabled</td></tr>
<tr><td>7:6</td><td></td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>PCSA</td><td rowspan="2">Peripheral Chip select</td><td>
<table>
<tr><th>Bit</th><th>Value</th><th>AlphaStation 600 requirement</th></tr>
<tr><td>0</td><td>1</td><td>RTC enabled</td></tr>
<tr><td>1</td><td>1</td><td>Keyboard enabled</td></tr>
<tr><td>5:2</td><td>0</td><td>Floppy and IDE disabled (Floppy is on the SIO chip)</td></tr>
<tr><td>6</td><td>0</td><td>Keyboard on X-bus (new encoding)</td></tr>
<tr><td>7</td><td></td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>PCSB</td><td>
<table>
<tr><th>Bit</th><th>Value</th><th>AlphaStation 600 requirement</th></tr>
<tr><td>6:0</td><td>0</td><td>Serial and Parallel Port disabled (this logic is on the SIO chip)</td></tr>
<tr><td>7</td><td>1</td><td>Configuration RAM enabled</td></tr>
</table>
</td></tr>
<tr><td>EISAID</td><td>EISA ID</td><td rowspan="2">No special AlphaStation 600 requirement</td></tr>
<tr><td>SGRBA</td><td>Scatter/gather base address</td></tr>
<tr><td>PIRQ[3:0]</td><td>PCI irq route</td><td>Bit <7> = 0 (disable). The PIRQ bits are not used by AlphaStation 600</td></tr>
<tr><td>GPCSLA[2:0]
GPCSHA[2:0]
GPCSM[2:0]</td><td>Gen Purp chip select Low High Address & Mask</td><td>GPCS[0] is used in the AlphaStation 600 system for the Operator Control Panel interface chip. GPSCS[2:1] are not used.</td></tr>
<tr><td colspan="4">The correct procedure for changing a configuration register value is to first read the register, change only the required bits (do not modify the reserved bits) and write the new value out to the register.</td></tr>
</table>

Table 10-4 ESC Registers (continued)

<table>
<tr><th>Category</th><th colspan="2">ESC register</th><th>AlphaStation 600 requirements</th></tr>
<tr><td>ESC DMA Registers</td><td colspan="2">See ESC spec for details</td><td>No special AlphaStation 600 hardware requirements</td></tr>
<tr><td>ESC TIMER registers</td><td></td><td></td><td>No special AlphaStation 600 hardware requirements</td></tr>
<tr><td>Interrupt Controller Registers</td><td></td><td></td><td>The AlphaStation 600 system uses this ESC interrupt logic. See the Hardware Exceptions and Interrupt chapter for the irq assignment.
The standard cascaded scheme is used.</td></tr>
<tr><td rowspan="4">NMI status & control registers</td><td>NMI status and Control</td><td rowspan="4"></td><td>
<table>
<tr><th>Bit</th><th>Value</th><th>AlphaStation 600 requirement</th></tr>
<tr><td>0</td><td></td><td>Software can use as they choose</td></tr>
<tr><td>1</td><td>1</td><td>Speaker enabled</td></tr>
<tr><td>2</td><td>0</td><td>PERR# disabled</td></tr>
<tr><td>7:3</td><td></td><td>Software can use as they choose</td></tr>
</table>
</td></tr>
<tr><td>NMI Cntrl and RTC address</td><td rowspan="3">No special AlphaStation 600 hardware requirements</td></tr>
<tr><td>NMI ext Status & cntrl reg</td></tr>
<tr><td>Software NMI</td></tr>
<tr><td rowspan="3">EISA Config. Floppy, and Port 92</td><td>Configuration RAM register</td><td></td><td>The AlphaStation 600 system provides the Configuration RAM. All Page address bits are used.</td></tr>
<tr><td>Digital output Register</td><td></td><td>Since there is no floppy on the AlphaStation 600 X-bus then bit<3> should be 0.</td></tr>
<tr><td>Port 92 Reg</td><td></td><td>
<table>
<tr><th>Bit</th><th>Value</th><th>AlphaStation 600 requirement</th></tr>
<tr><td>0</td><td>0</td><td>ALTRST# not used by AlphaStation 600</td></tr>
<tr><td>1</td><td>0</td><td>ALT20 not used by AlphaStation 600</td></tr>
<tr><td>2</td><td></td><td>Reserved</td></tr>
<tr><td>3</td><td></td><td>RTL Password protection
Software can use as they choose</td></tr>
<tr><td>4</td><td>0</td><td>Not used</td></tr>
<tr><td>5</td><td></td><td>Reserved</td></tr>
<tr><td>7:6</td><td>0</td><td>Fixed Disk activity Light not used by AlphaStation 600</td></tr>
</table>
</td></tr>
</table>

PCEB Functionality

The AlphaStation 600 system's use of the PCEB is fairly standard. The following sections highlight specific AlphaStation 600 system requirements.

PCI-to-EISA Address Decode

This section is concerned with PCI-to-EISA transactions. Subtractive decode must be used for the PCEB (negative decode does not work).

PC Compatibility Addressing and Holes

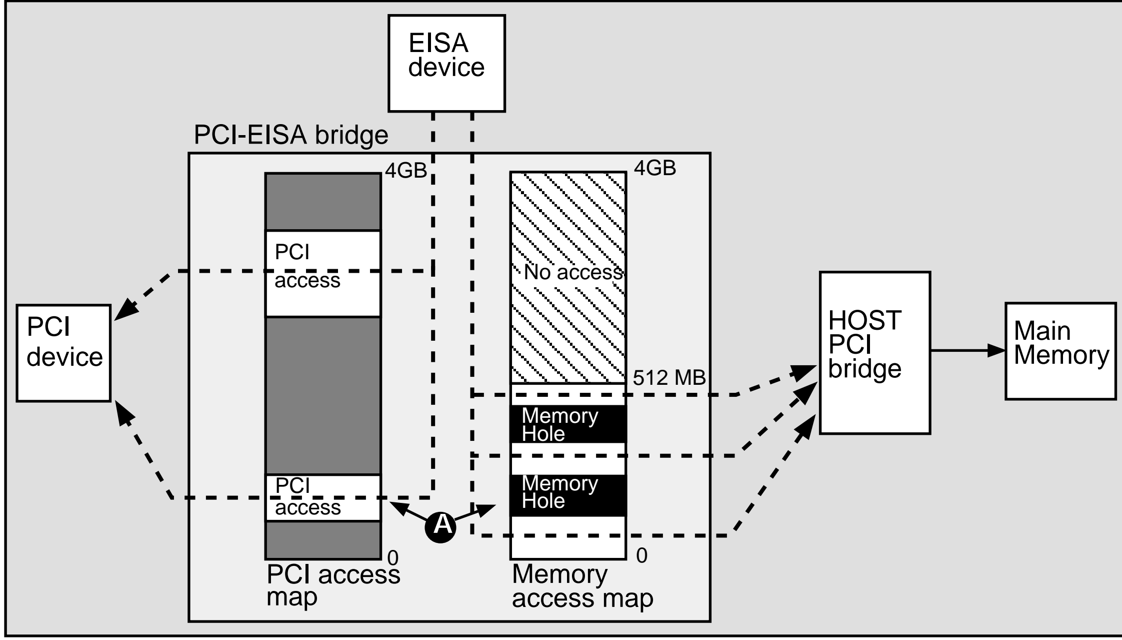
This section is concerned with EISA-to-PCI transactions.

Software note: *Although PC compatibility holes are required in the PCI-EISA bridge to support certain memory-mapped ISA devices; it is hard to imagine of any useful scenario where this memory-mapped ISA device is directly accessed by a PCI device. Consequently, the CIA will probably never need to support any PC compatibility holes (that is, the MEMCS# logic will not be enabled in the CIA). Since the "cost" of this logic to the CIA is only one pin and a few gates, the functionality is provided "just in case".*

The PC architecture allows certain (E)ISA devices to respond to hardwired memory addresses: an example is a VGA graphics devices which has its frame buffer located in memory address region A0000-BFFFF. Main memory must be made inaccessible for such memory-mapped regions, and this inaccessible region is called a PC compatibility hole (or "hole" for short).

The EISA-PCI bridge provides access for (E)ISA devices to main memory (which is normally behind a HOST-PCI bridge) via positive address decode. The lower 512 MB of EISA address range is partitioned into many sub-segments which can be enabled by the MCSTOM, MCSTOH, MCSBOH, EADC1, EADC2 registers. These registers allow main memory "holes" to be created. The right-hand-side of figure 10-2 shows an access path from an EISA device through to main memory.

Figure 10-2 EISA Access to PCI and Memory



With this address-hole logic, a PCI-EISA bridge chip can be configured to allow EISA-to-EISA operations as well as EISA-to-memory transfers. For example, if a VGA graphics device resides on the EISA bus, then other EISA device can access the VGA frame-buffer directly (eg a multimedia video device) without having the access go incorrectly to main memory.

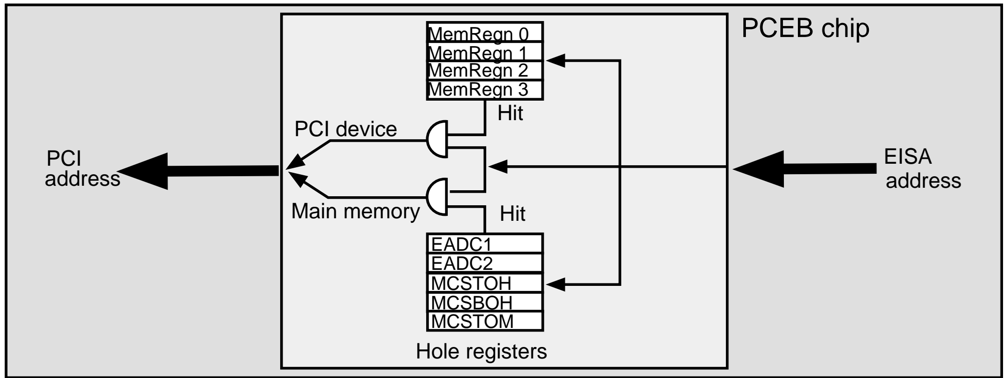
A hole is not a bi-directional blockage: for example, a PCI-EISA bridge can have a hole preventing access by EISA devices to main memory; but a PCI device can happily reach through the "hole" to the memory-mapped EISA device.

The PCI-EISA bridge chip can also direct EISA addresses to a PCI device using the MEMREGN registers. This path is shown on the left-hand-side of Figure 10-2. This path is not intended for access to main memory via a Host-bridge device (that is, the CIA) on the PCI bus; but with care[1] main memory access is possible.

Note that accesses to the PCI devices and access to main memory must be mutually exclusive[2] -- thus access to a PCI device occurs either above the main-memory window or in a main-memory hole (See A in Figure 2).

Figure 10-3 shows the relationship between the PCEB registers and the EISA access to main memory or to a PCI device.

Figure 10-3 PCI-EISA Bridge: EISA Address Decode



Once EISA-to PCI or PCI-to-EISA access is allowed then "holes" are required in the CIA to block the PCI transfer from incorrectly accessing main memory. Although, the PCI window BASE and MASK registers in the CIA provide a means for controlling access to main memory; their granularity and resolution is poor. EISA access to the PCI devices is through Base and Limit registers with a 64 KB resolution; the CIA's BASE and MASK registers only provide a 1 MB resolution.

Now, a typical PCI device has no better resolution than the CIA. Thus transfers from a EISA device to a "typical" PCI device are easily controlled with the CIAs and the PCI devices window logic. However, the transfer in the opposite direction (PCI to EISA) requires much finer 64 KB resolution (ie EISA devices are memory-mapped on 64 KB segments). This is depicted in Figure 10-4.

[1] In this path, the MEMCS# is not asserted. Hence PCI window 0 in the CIA will not accept the transaction if the MEMCS path is enabled. However, other windows can be configured to accept this command.

[2] that is, EISA transactions can only go to one place. This is also the PCI requirement of non-overlapping windows (note that main memory is behind a PCI device (the host bridge).).

Figure 10-4 PCI-EISA and CIA Hole Example

The example in Figure 10-4 is as follows: An ISA VGA graphics device resides on the (E)ISA bus, and a multi-media device resides on the PCI bus. The frame-buffer is mapped in the ISA memory space. If other (E)ISA devices require access to the ISA frame buffer, then a "hole" is required in the PCI-EISA bridge's window to main memory (but at this point, no corresponding hole is required in the CIA's PCI-address window -- the need for this is explained next). Note that for the EISA devices to access main memory, a PCI window is required in the CIA to allow the EISA devices a path through to memory.

Suppose the PCI device wishes to write to the VGA frame-buffer. Without a "hole" in the CIA, the PCI access to the ISA frame buffer would instead go to main memory (that is, the access would "hit" in the CIA window --- and the PCI-EISA bridge would ignore the transaction for it subtractively decodes the PCI addresses). Hence, a "hole" is required in the CIA's PCI window, and this hole is located in the same address region as the hole in the PCI-EISA bridge.

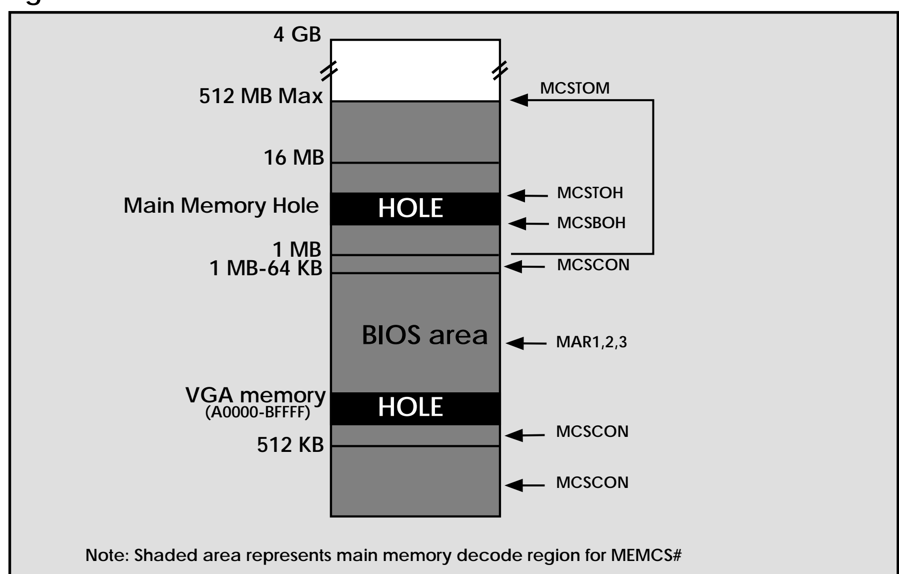
The PCEB chip provides an address decoder which takes into account memory holes and asserts a signal, MEMCS#, whenever a PCI or EISA device wishes to access main memory. The CIA qualifies its PCI window hit logic with MEMCS#, thus preventing the CIA from accepting any transactions which fall into a hole.

The CIA is NOT always required to match every hole in the PCI-EISA bridge (although there is no harm in doing so). The CIA must have "holes" punched into its PCI window only if PCI traffic is directed to an EISA memory-mapped device which is within the address space of a PCI window.

MEMCS# Details

The PCEB chip of the PCI-EISA bridge provides address decode logic with considerable attributes and features (eg read only, write only, VGA frame buffer, memory holes, BIOS shadowing) to help manage the EISA memory map and PC compatibility holes. This is known as *main memory decoding* in the PCEB chip, and results in the generation of the MEMCS# (**MEM**ory **C**hip **S**elect) signal. The CIA uses MEMCS# if it is enabled in the PCI BASE register for window 0.

Figure 10-5 MEMCS# Decode Area



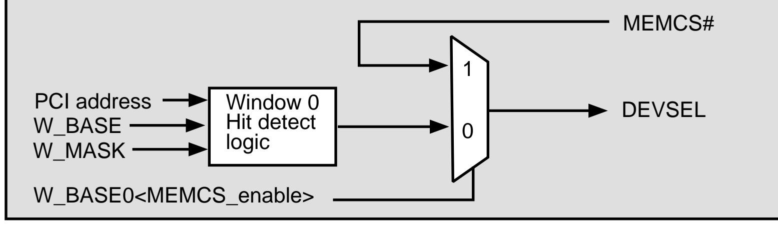
In Figure 10-5, the MEMCS# range is shown shaded lightly; the two main holes are shown shaded darkly. This range is subdivided into numerous portions (for example, BIOS areas) which are individually enabled/disabled using various registers:

- MCSTOM (top of memory) register. This has a 2 MB granularity and can be programmed to select the regions from 1 MB up to 512 MBs.
- MCSTOH (top of hole) and MCSBOH (bottom of hole) registers define a memory hole region where MEMCS# is not selected. The granularity of the hole is 64 KB.
- MAR1,2,3 registers. These enable various BIOS regions.
- MCSCON (control) register. This register enables the MEMCS# decode logic, and in addition selects a number of regions (eg 0-512 KB).
- The VGA memory hole region never asserts MEMCS#.

For more detail, please refer to the Intel 82375EB specification.

PCI window 0 in the CIA can be enabled to accept the MEMCS# signal as the PCI memory decode signal. With this path enabled, the PCI window hit logic simply uses the MEMCS# signal (ie if MEMCS# is asserted then a PCI window 0 hit occurs and the PCI DEVSEL signal is asserted).

Figure 10-6 MEMCS# Logic



Consequently, the PCI BASE address must be large enough to encompass the MEMCS region programmed into the PCI-EISA bridge. The remaining window attributes are still applicable/required:

- The SG bit in the PCI BASE determines if scatter/gather or direct-mapping is applicable

- The MASK register size information must match the MEMCS# size (in order for the S/G and direct mapping algorithms to correctly use the Translated Base register).
- The MEMCS_Enable bit in the W_BASE 0 CSR takes precedence over the PCI window enable bit (i,e., W_BASE<W_EN>).

Note: *The CIAs does "typical" DEVSEL except when the MEMCS# path is enabled, in which case it runs in "slow mode". This is done automatically by the hardware. However, software must configure the PCI-EISA bridge chips to "slow-mode" by writing to the appropriate CSRs.*

PCI Arbitration

The PCEB does not allow the use of an external arbiter; the internal PCI arbiter must be used. The AlphaStation 600 system has one PCI slot more than the PCEB can handle. This problem is solved by providing a sub-arbiter on the system module for the 32 bit sllots in conjunction with the PCEBs arbiter.

PCI Arbitration - Power-up

The PCEB arbiter is itiialized to provide round-robin arbitration but parks the host bridge (CIA) on the PCI. This means that the CIA is driving AD[31:0], C/BE[3:0] and PAR. The remaining 64-bit PCI signals are pulled-up by resistors and need not be driven.

Figure 10-7 AlphaStation 600 PCI Arbiter Scheme

Table 10-5 Round-Robin PCI Arbitration

Current Least Recently Used State					Highest Priority	Next Least Recently Used State				
Bank 2	Bank 0	Bank 3	Bank 1	Sub-arb		Bank 2	Bank 0	Bank 3	Bank 1	Sub-arb
Bank 0	X	Slot 1	X	X	Slot 2	Bank 3	same	Slot 2	same	same
Bank 0	X	Slot 2	X	X	Slot 1	Bank 3	same	Slot 2	same	same
Bank 3	X	X	Sub-arb	X	CPU	Bank 1	same	same	CPU	same
Bank 3	X	X	CPU	Slot 3	Slot 4	Bank 1	same	same	Sub-arb	Slot 4
Bank 3	X	X	CPU	Slot 4	Slot 3	Bank 1	same	same	Sub-arb	Slot 3
Bank 1	PCEB	X	X	X	Slot 0	Bank 0	Slot 0	same	same	same
Bank 1	Slot 0	X	X	X	PCEB	Bank 0	PCEB	same	same	same

If a bank or slot is not requesting, the priority is passed on to the next slot or bank that is requesting.

Potential for PCI-EISA Bridge Starvation

Extract from the PCEB stepping information: If two or more PCI masters have higher priority than the PCEB in a fixed arbitration mode (modes 4,5,6,7,8,9,A,B) then there is the possibility that the PCEB will never grant itself the bus due to heavy PCI traffic by two or more PCI masters, and the PCEB may starve itself.

The current recommended arbitration scheme is mode 2.

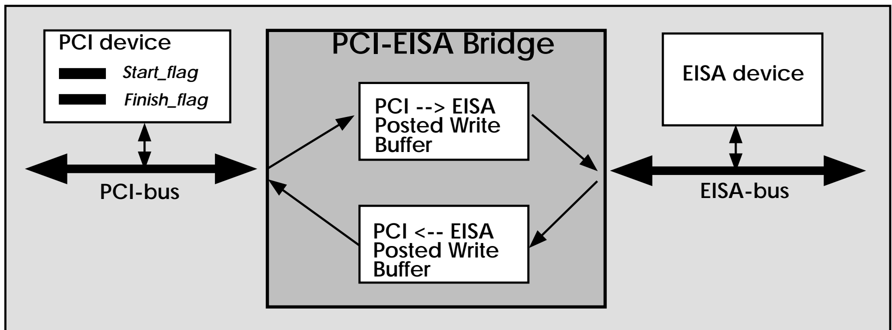
Coherency Implications

Once an EISA Bus owner begins a cycle on the EISA bus, the cycle cannot be backed-off. It can only be held in wait states via EXRDY. In order to know the destination of an EISA bus request, the request must be granted and the cycle needs to begin. Once a cycle is started, no other device can intervene and gain ownership of the EISA bus until the committed cycle completes.

The above EISA constraint, coupled with the write buffering and the need for strict read/write ordering can lead to a deadlock. Consider Figure 10-8 and the following scenario:

The EISA device wakes-up the PCI device by a write to its *start_flag;* this induces the PCI to dump data over to the EISA device. The EISA device polls the *finish_flag* within the PCI device to determine when the data dump has completed[1,2].

Figure 10-8 EISA Deadlock Example



The problem arises when the PCI to EISA data is stored temporarily in a Posted Write Buffer inside the PCI-EISA bridge chip (see Figure 10-8). Once the EISA device samples the *Finish_flag* set in the PCI device, then, in accordance with strict write-ordering, it believes that the PCI device has completed the data dump, and that the data is in the EISA device (that is, it does not expect any of the data to be in the Posted Write Buffer in the Bridge chip).

To ensure this strict-ordering constraint, the Bridge chip must flush the Posted Write Buffers when it detects an EISA master wishing to read a PCI device. However, this would lead to a deadlock: the flush cannot proceed because the bus is now committed to another EISA master, (which cannot back-off); and yet, the bridge cannot proceed with the EISA transaction as this would violate the strict-ordering rules.

The solution adopted by Intel is drastic -- the PCI-to-EISA write buffer must be permanently disabled.

[1] This is effectively Litmus test 7 of the Alpha SRM (section 5.6.2.7), except now there are no explicit MB instructions; they are implicit in the Intel architecture. This is important because the Intel architecture is the implicit model behind the PCI specification, and is the one most PCI drivers will be based on.

[2] The flags could be in either PCI memory or PCI I/O-space.

Coherency: Posted Write Buffer in the PCI Device

The above deadlock scenario is still applicable if a PCI device contains a posted write buffer. In this case, the PCI write buffer must be flushed before the EISA device is granted the bus. The PCEB chip provides a signal called FLSHREQ# to achieve this objective.

How all this is achieved[1] in the PCI-EISA bridge chip is somewhat more convoluted:

- When the ESC chip detects an EISA request, it first makes the PCEB perform the following tasks:

  1: Check to see if it itself is locked as a PCI resource. If so, wait until the lock is cleared.
  2: Stop accepting PCI requests (that is, all PCI requests to the PCEB are retried).
  3: Hand the EISA bus to the ESC (but note that the EISA bus is still owned by the PCI-EISA bridge).

  Note that no PCI write buffers have yet been flushed since the FLSHREQ# signal has not yet been asserted.

- EISA arbitration is "frozen" -- that is, no more EISA requests are accepted and the highest priority request is determined (BUT not granted - the PCI-EISA bridge still owns the EISA bus).
- If the EISA request is either an EISA-master, ISA-master or DMA, then FLSHREQ# is asserted. All PCI write buffers pointing towards the PCI-bus (and thus possibly destined for the EISA bus) are flushed and disabled.

  The PCEB directs any PCI flushed-writes requests which it receives to the EISA bus -- remember, the PCI-EISA bridge is still master of the EISA bus. Note, that since the PCI-to-EISA write buffers are (always) disconnected, and the PCI bandwidth exceeds the EISA bandwidth, then the PCEB will disconnect after every Dword.

- Once all the system write buffers have been flushed, then the system asserts MEMACK#.
- Only now does the PCI-EISA bridge relinquish the EISA-bus and pass control over to the requesting (E)ISA-master or DMA.

The problem with FLSHREQ# is that it is not a PCI signal -- it is a sideband signal. This means that the signal is not available on a PCI slot-connector. Consequently, there is no way of informing the PCI options to flush their buffers. Most, if not all PCI posted write-buffers adhere to strict-ordering rules and would force a flush of their write buffer for a read transaction. Since software cannot rely on the hardware FLSHREQ# signal then it must handle deadlock avoidance itself. Here are a number of options:

- Disable all Posted Write Buffers on the PCI option card. If the option card has a PCI-PCI bridge then the write buffers in this bridge will need to be disabled. This is a poor solution as it will seriously impact performance.
- After the PCI device dumps its data to the EISA bus, it must read back the last item written before asserting the *finish_flag*[2].
- Instead of going directly from the PCI device to the EISA device, send the data via memory: viz, the PCI device writes the data to system memory, and the EISA then copies from memory. The *start_flag* and *finish_flag* can still be on the PCI device. The reason this works is because the EISA-devices read of the *finish_flag* will cause the PCI device to flush its write buffers; and since this flush is directed towards main

[1] Actually, no Intel documentation is fully explicit on this. But, based on Intel's track record with the PCI-EISA bridge it is quite likely that there are bugs yet to be found in their implementation of FLUSHREQ#.
[2] In general, this works because most PCI write buffers preserve write order (at least for Dwords)

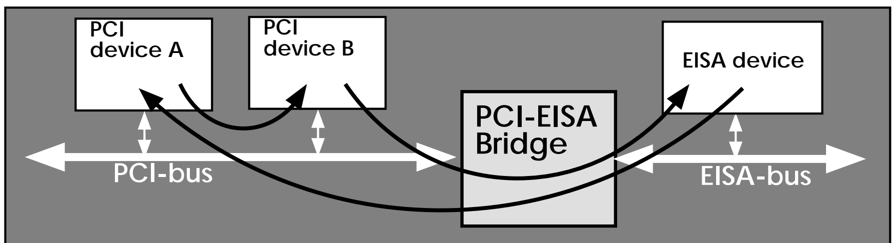
memory, and not the EISA device, then deadlock cannot occur (that is, there is no resource on the path to memory which cannot be backed-off).

- Place the *finish_flag* in main memory: the PCI device will write to a location in main memory when it has completed the transfer, and the EISA device reads this memory location. This works for all PCI write-buffers which maintain write-ordering (that is, if the write of the flag to main memory occurred, then all prior writes from the PCI device to the EISA device must also have completed).
- Locate the *finish_flag* in the EISA device.
- etc.

PCI Deadlock Avoidance Rule

If a PCI option device can write directly to any EISA device, and the PCI device has a posted write buffer, then no EISA device must ever read directly (or indirectly -- see next paragraph) from that same PCI device. Otherwise, a deadlock is possible.

Figure 10-9 Interacting Deadlock Example



Note that interacting events could yield a deadlock. For instance, two PCI devices labeled A and B both have write buffers. If PCI device A contains write data destined for device B, and device B has write data destined for EISA, then a read from an EISA device to device A will create a deadlock.

Coherency: CIA and FLSHREQ#

The issue is what to do with the FLSHREQ signal and the one-and-only posted-write buffer (the I/O write buffer) inside the CIA. If one observes that the CPU-CIA entity is really a PCI device, then the example discussed earlier is applicable; and since the CIA is on the system module then it can access the FLSHREQ# signal. The obvious answer is to flush and disable the CIA posted write buffer on the assertion of FLSHREQ#; especially since EISA-to-CIA and CIA-to-EISA transactions are common. Unfortunately, it is not quite that "obvious".

What makes the solution less "obvious" is that the data-coherency that is being preserved is really an uncommon, almost trite case -- and it has to do with preserving ordering between CPU I/O writes and DMA memory accesses.

The only posted write buffer in the CIA is for I/O writes. Furthermore, the AlphaStation 600 system uses a *posted write and run* approach for I/O writes: that is, the CIA does not preserve ordering between CPU memory writes and I/O writes. Applying this information to the afore-described deadlock example results in the following scenario between the CPU and the EISA device:

- The EISA device "wakes-up" the CPU by an interrupt (this is equivalent to setting a *start_flag* in main memory).
- The CPU copies memory data to the EISA device. Notice that this is done via uncached (viz CPU I/O space) writes. When completed the CPU sets the *finish_flag* in main memory (and this is why this example is so artificial and unlikely[1] -- in "real" life the *finish_flag* would be an I/O write and there would be no ordering problems).

[1] It transpires that graphics devices use programmed I/O for frame-buffer writes and could use memory flags for flow-control. This is not quite the same situation, but who knows what tricks are being planned by bizarre minds.

- The EISA device polls the *finish_flag* in main memory. Once it has successfully read the flag, then it believes that the data transferred by the CPU is in the EISA device (and not in any write buffers in the CIA, PCI-EISA bridge, etc).

As before, the FLSHREQ# signal can be used to make this scenario work; however, this will entail flushing the I/O write buffers (and stalling the CPU from writing to these buffers) for **every** (non-refresh) EISA transaction -- that is, including EISA transactions which are not destined for main memory -- and furthermore, the CPU will be stalled for the duration of the EISA transfer.

Consequently, the default AlphaStation 600 system approach is to not bother flushing the Posted write buffer (the I/O write buffer) for a FLSHREQ#; although we do allow the option of software enabling the flushing via the CIA_CTRL CSR.

Software requirement: The AlphaStation 600 system I/O write buffers are intended to be used in a post-and-run manner[1]. Software must ensure data coherency with respect to a PCI or EISA device by software techniques, such as:

- The I/O write buffer is flushed by performing an IO read (reading a CIA CSR is sufficient and fast).
- Completing the I/O write sequence by a I/O write to a *finish_flag* on the PCI/EISA device.
- etc.

Guaranteed Access Time Mode

EISA and ISA cycles can be extended (that is, wait states) with the EXRDY and CHRDY signals. However, the wait states are not indefinite, and 2.1 uS is the longest delay possible which will satisfy both bus constraints.

The PCI-EISA bridge set can be configured into a mode called Guaranteed Access Time (GAT) mode, where the bridge issues MEMREQ# when it wishes an unimpeded path to system memory; and the system responds with MEMACK# once it has completed the preparation to guarantee a 2.1 uS maximum latency. The AlphaStation 600 system is designed to run with GAT mode enabled.

The simplest approach for the system is to "crow-bar" the path to memory -- that is, stall the CPU and keep the CPU stalled until the EISA device has completed its transaction (note that the CPU is not just stalled for EISA-to-PCI transactions, but also for EISA-to-EISA transfers since the PCEB asserts MEMREQ# for any non-Refresh EISA cycle). Table 10-6 itemizes the total delay for an ISA memory read.

[1] Intel architecture (and thus PCI architecture) also assumes a post-and-run IO write design.

Table 10-6 AlphaStation 600 GAT Latency Delay

Operation	Delay With Scatter/Gather & CPU stalled	Delay No Scatter/Gather CPU not stalled
MEMREQ received: -- Flush I/O write buffers (optional) -- schedule memory refresh MEMACK# sent	 200ns (refresh occurs in parallel with issuing MEMACK)	 200ns (refresh occurs in parallel with issuing MEMACK)
ISA bus delay - 1 cycle	120 nS	120 nS
PCI_EISA bridge - 2 PCI cycles	60 nS	60 nS
PCI transaction - 4 cycles	240 nS	240 nS
CIA pipe delay - 2 cycles	60 nS	60 nS
CPU memory read with victim	----	400 nS
Scatter/Gather miss (+ PCI retry)	400 nS	--
Memory read for EISA data + CIA pipe delay	270 nS	270 nS
PCI transfer	30 nS	30 nS
PCI-EISA bridge - 2 PCI cycles	60 nS	60 nS
Align to ISA clock	90 nS	90 nS
ISA bus - 1 cycle	120 nS	120 nS
Total GAT delay	1.650 uS	1.650 uS
Note: the figures in this table are only a quick estimate. This table remains to be refined		

Table 10-6 tabulates two variants. The first delay column represents the "crow-bar" approach where the CPU has been stalled. There is sufficient time in this case to allow scatter/gather to be enabled for the PCI-EISA transactions. The last column, represents the case where scatter/gather is disabled (that is, direct mapping), or the TLB entry has been locked. Now there is sufficient time to allow the CPU to perform memory transactions: in this case the EISA transaction may have to wait as long as 400ns for a CPU memory fill with victim to complete.

Gat-Mode Software Notes

Table 10-6 assumes only one scatter/gather miss. There is not enough time to support two misses and provide a reasonable margin of error in the calculation. Hence, if it is possible for an EISA transfer to cross a 32 KB-aligned PCI address boundary (each scatter/gather TLB entry is 4 consecutive 8 KB PTEs), then scatter/gather mapping should not be used if GAT mode is required, unless the appropriate TLB entries are locked.

Software can enable or disable the stalling of the CPU during EISA transactions via the CIA_CTRL CSR. If the CPU is in the non-stalled mode then the AlphaStation 600 system will allow CPU to memory operations to proceed (but not I/O transactions) while in GAT mode. Clearly, for performance reasons it is preferable to avoid stalling the CPU. However, this will mean that scatter/gather must be disabled for the EISA transfers; or that the TLB entry has been locked. Otherwise, the CPU will be stalled for the duration of most EISA transactions.

The AlphaStation 600 hardware is designed to run in GAT mode. In this mode FLSHREQ# and MEMREQ# are synonyms. Hence, the AlphaStation 600 system will only use one of these signals (probably FLSHREQ#). Software needs to consider the implications of this if they choose to run with GAT-mode disabled.

Data Buffering in the PCEB

The AlphaStation 600 system expects that the Line Buffer is enabled in the PCEB chip.

The latest errata from Intel requires that the PCEB Posted Write Buffer is disabled. This means that a PCI master requesting a PCI-to-EISA transfer is retried until the PCEB owns the EISA bus. Each PCI-to-EISA transfer must complete all the way to the EISA destination before the next transfer may begin. In other words, performance through to EISA will be abysmal (and the fewer EISA options installed the better).

11
System Coherency

Introduction

This chapter defines the rules that will insure that data coherency is maintained across the AlphaStation 600 system. All types of data movement are considered, including I/O reads and writes, DMA transfers and peer to peer transfers and their relationships to interrupts. This chapter also describes how the AlphaStation 600 system maintains coherency if the rules are followed.

Referenced Documents

Alpha System Reference Manual

Software must observe all requirements of the latest version of the **Alpha System Reference Manual**, and the AlphaStation 600 system conforms to it. Chapter 5 is of particular interest. In regard to Chapter 8, all of the I/O system built into the AlphaStation 600 system is "Local I/O Space." The AlphaStation 600 system contains no "Remote I/O Space".

PCI Local Bus Specification

The I/O bus system in the AlphaStation 600 system begins with a 64 bit PCI emerging from the CIA chip. Software must comply with requirements in the latest version (Rev 2.0 or later) of the **PCI Local Bus Specification, Production Version**, as published by the PCI Special Interest Group, 5200 N.E. Elam Young Parkway, Hillsboro, Oregon 97124.

NCR 53C820

At least initially, the internal Fast Wide SCSI bus and the external Fast Wide SCSI bus will both be generated with NCR 53C820 chips. These parts appear on a PCI option module that will be standard in AlphaStation 600 systems. This does, however, make it easy, from a hardware prospective, to change this to a different chip type if and when that becomes desirable. In operations with the SCSI busses and devices on the SCSI busses, software must observe the requirements of the NCR 53C820 data book.

Other Devices

See data books or specifications for requirements for operating other devices.

Coherency Summary

The AlphaStation 600 chip set treats an MB as a NOP, and the AlphaStation 600 system should be configured so that MBs do not leave the EV5 chip. A recent ECO to the Alpha SRM allows "posted" or "buffered" writes to I/O devices. This ECO allows designs to buffer writes from the CPU to a PCI device, and allows reads from the same PCI-devices to main memory to complete while writes to this same device are buffered. The ECO states that the usual way for a programmer to be sure the write data has reached the device is to read a register on the same device. This is explained in more detail in this chapter.

The following AlphaStation 600 hardware features are required for the ECO:

- Before a read of an I/O location is processed, all preceding writes by the EV5 to devices are flushed out of the buffers.
- Before a read by a PCI-device to main memory is processed, all preceding writes by the device to main memory are flushed out of their write buffers.

Classification of Coherency Situations

Generic Event Types in the AlphaStation 600 System

The possible generic events in the AlphaStation 600 system are:

1. A Read
2. A Write
3. An Interrupt
4. I/O Page Table Modification

1. The possible things that can go wrong with a read are:

A. Fails to return the latest written data

B. Data changed before read

2. The possible things that can go wrong with a write are:

A. Data fails to make it to a later read (same as 1 A)

B. Data overwrites some later write

C. Data is overwritten by some earlier write (same as 2B)

D. Data is overwritten before readers are finished with it (same as 1B)

E. Data fails to make it to destination when needed to cause some side effect

F. Side Effects Happen out of order

3. The possible things that can go wrong with an interrupt are:

A. Preceding reads or writes are not completed when interrupt is recognized (these failures should all show up under 1 or 2 above)

4. The possible things that can go wrong with the I/O page table modification are:

A. Failure to use the latest Page Table State

B. Page Table changed before use of it is completed

I conclude from above that all failures can be accounted for under items 2 or 4, that is all failures may be considered to be either Write failures or I/O page table modification failures. Further It can be seen that 2B and 2C are the same thing.

The complete list of possible failures may be restated:

1. The possible things that can go wrong with a write are:

A. Data fails to make it to a later read

B. Data overwrites some later write

C. Data is overwritten before readers are finished with it

D. Data fails to make it to destination when needed to cause some side effect

E. Side Effects Happen out of order

2. The possible things that can go wrong with the I/O page table modification are:

A. Failure to use the latest Page Table State

B. Page Table changed before use of it is completed

Possible Write - Read and Write - Write interactions

To aid in this study we construct exhaustive decompositions:

* A Read is either:

    EV5 initiated

    device initiated

* An EV5 initiated read is either:

    A memory read

    An I/O read

* An EV5 initiated I/O read is either:

    An AlphaStation 600 system CSR read

    A device read

* A device initiated read is either:

    A memory read

    A peer device read

* A Write is either:

    EV5 initiated

    device initiated

* An EV5 initiated write is either:

    A memory write

    An I/O write

* An EV5 initiated I/O write is either:

    An AlphaStation 600 system CSR write

    A device write

* A device initiated write is either:

    A memory write

    A peer device write

It will be noted that a complete decomposition of reads above yields an exhaustive list of 5 different types of reads. An exhaustive decomposition of writes above yields an exhaustive list of 5 different writes. The two lists are the same.

If one considers the data interactions under item 1, then either for write-read interactions or for write-write interactions, the cartesian product yields 25 different cases, in principle. However, in most of the 25 cases there can be no data interaction. You cannot have a problem with a read to one place not returning data that was written to a different device. Nor can you have data written to one device overwrite data written to another device. The following table gives all significant cases:

TRANSACTION 1	TRANSACTION 2
EV5 to memory	EV5 to memory
EV5 to memory	device to memory
EV5 I/O to AlphaStation 600 CSR	EV5 I/O to AlphaStation 600 CSR
EV5 I/O to device	EV5 I/O to device
EV5 I/O to device	device peer
device to memory	EV5 to memory
device to memory	device to memory
device peer	EV5 I/O to device
device peer	device peer

There are therefore 9 different significant cases of interaction of Writes with Reads or Writes with Writes.

The issue now is to describe the software rules for each situation and then to demonstrate that if the rules are followed, there can be no failure in an AlphaStation 600 system.

Basic Properties of the AlphaStation 600 System

We will use a few facts about the AlphaStation 600 system design:

1. The DRAM memory does only one thing at a time.

2. A read or a write to memory is atomic. I am particularly interested here in writes that actually require read modify write cycles in the DRAM. No other use of the DRAM can get between the read and write of the read modify write. This being so, we can ignore the fact that some writes require read modify write. All other accesses to DRAM occurred either completely before a write (whether or not it was a read modify write), or completely after the write.

3. The memory sequence for an EV5 cache miss with victim is atomic in the memory. A cache miss with victim first does a read in memory for the block required for the cache miss, and then it writes the victim data to memory. No other access to memory can come between the read and the write.

4. When a transaction has won arbitration for the memory it is said to "own memory." Once a transaction owns memory, there will be no memory access for any other transaction until this transaction has completed all of its memory activity.

5. In the event of an I/O TLB miss, a Read command is given to the EV5 for the data for the TLB entry and a memory access is also made for this data. The Read command is not issued to EV5 until memory is owned for this access.

6. When a device read to memory is processed, a Read Command is issued to EV5 for the required data and also a memory access is made for the required data. The Read command is not issued to EV5 until memory is owned for the memory cycle.

7. When a device write to memory is processed, a Flush Command is issued to EV5 for the required data. A memory access is also made. This memory access serves to write data obtained from EV5 in response to the Flush Command to memory and also serves to

write the data from the device to memory. The Flush Command is not issued to EV5 until memory is owned for the memory write.

8. We can conclude that the two-step sequences described in Basic Properties 5, 6, and 7 above are all mutually atomic. In each case the two-step sequence is a probe of EV5 and a memory cycle. For any pair of sequences of the types described in 5, 6, and 7, both steps of one of the sequences happened entirely before either of the two steps of the other sequence.

Analysis of Interactions with Writes

Data fails to make it to a later read

The possible decomposed cases are:

1. Write, EV5 to memory; Read, EV5 to memory

SOFTWARE RULES

For an EV5 memory read to a location that was previously modified with an EV5 memory write to work correctly the following sequence must be used:

STORE

LOAD

WHY IT WORKS IN the AlphaStation 600 system

We assume EV5 internally does the right thing. The only time the AlphaStation 600 system is involved is if the written location became a victim and was written to memory, or was evicted from cache with a Flush Command and then it was read back in a miss.

First consider the case in which the data was a victim. EV5 will issue the cash miss with victim command first and then the cache miss to the victim address. The AlphaStation 600 system processes EV5 commands in order. The cache miss with victim will own the memory first and do both the read and the write of the victim. The read write sequence for a miss with victim is atomic in the memory (Basic Property 3). The memory access for the cache miss will be after the victim write.

Next consider evicting data with a Flush and then an EV5 cache miss hits that data. If the miss is on the EV5 pins after the Flush Command went to EV5, then the memory is already owned to write the Flush Data to memory (Basic Property 7) before the miss command is issued by EV5. The memory cycle for the miss is after the flushed data goes to memory.

2. Write, EV5 to memory; Read, device to memory

SOFTWARE RULES

In order to insure that a device will correctly access data written to memory by EV5, software must use the following sequence:

EV5 STORE TO MEMORY

WEAK MEMORY BARRIER OR MEMORY BARRIER

EV5 SIGNALS DEVICE

DEVICE RECEIVES SIGNAL

DEVICE READS MEMORY

The EV5 can signal the device only by writing something to memory or by writing something to a device or by reading a device. In the case of signaling by reading a device, Memory Barrier must be used, Weak Memory Barrier is not enough.

WHY IT WORKS IN the AlphaStation 600 system

Under these conditions, by the time the SIGNAL leaves the EV5 chip, a condition will have been established in which either EV5 will supply the data in response to a Read or a Flush, or the data has been sent to memory in a miss with victim or in response to a previous flush.

Data from EV5 in a flush buffer or en route to a flush buffer is as good as in memory, relative to a device read by Basic Property 8. If the device read sequence is before the flush sequence then it will get the data from the Read command to EV5 on its behalf. If the device read sequence is after the flush sequence, then it will get the data from memory.

Data in the victim buffer is as good as in memory. A device read to memory could be for the block that is in the victim buffer or is en route to the victim buffer and that such a device read must get the value that is in the victim buffer or en route to the victim buffer. The details are as follows: the EV5 retains (and still owns) the victim data, even though a copy may currently reside in the victim buffer, until the EV5 memory read is completed. Hence a device read will obtain the correct data from the EV5 if the EV5 read has not yet completed; and if the read has completed, then the data is in memory since the AlphaStation 600 system ensures that the victim write is atomic with the associated EV5 read fill.

3. Write, EV5 I/O to the AlphaStation 600 CSR; Read, EV5 I/O to the AlphaStation 600 CSR

SOFTWARE RULES

To insure that a read from a location, a CSR in this case, will return the result of a previous write to that location, software must use the following sequence:

EV5 STORE TO the AlphaStation 600 CSR

EV5 READ TO the AlphaStation 600 CSR

WHY IT WORKS IN the AlphaStation 600 system

Due to fact that the read and write addresses match, and the read and write are to non cached space, EV5 will emit the write first before the read. The AlphaStation 600 system will do only one operation on any given CSR at a time and in the order in which the EV5 emitted them.

4. Write, EV5 I/O to device; Read, EV5 I/O to device

SOFTWARE RULES

To insure that a read from a location, a device register in this case, will return the result of a previous write to that location, software must use the following sequence:

EV5 STORE I/O TO DEVICE

EV5 LOAD I/O TO DEVICE

WHY IT WORKS IN the AlphaStation 600 system

Due to the fact that addresses for the store and load match and the store and load are to noncached space, EV5 will emit the write first before the read. I/O commands from EV5 go to the PCI strictly in the order in which they are emitted by the EV5. PCI transactions from EV5 to any given target stay strictly in order through bridges and what have you.

5. Write, EV5 I/O to device; Read, device peer

SOFTWARE RULES

To insure that device B, doing a peer to peer read of device A will get the result of an EV5 write to device A, software must use the following sequence:

EV5 STORE I/O TO DEVICE A

MEMORY BARRIER[1]

EV5 LOAD I/O TO DEVICE A

MEMORY BARRIER

EV5 SIGNALS DEVICE B

DEVICE B RECEIVES SIGNAL

DEVICE B READ PEER DEVICE A

The only possibilities for the signal are EV5 writes memory, EV5 writes to device B, or EV5 reads to device B.

EV5 Load I/O to device A need not be to any particular register or location. It can be to anything in device A. The general principle is that the one and only mechanism that EV5 has to determine that an I/O write has arrived at a device is to read something (anything) from that same device. EV5 cannot signal device B that the data is at device A until it knows itself that the data is at device A.

WHY IT WORKS IN the AlphaStation 600 system

Due to the first Memory Barrier[2], the EV5 write and read to device A will be emitted in order. They will go to the PCI in order and arrive at device A in order. Hence we do not get past the second memory barrier until the EV5 write to device A has actually reached the device. Hence device B could not receive the signal, no matter what it is, until after the EV5 write to device A has reached device A. The device B read to device A then certainly reaches device A after the EV5 write to device A does. After that it is up to device A to do the right thing.

SPECIAL CASE:

In the special case that the signal is either EV5 writes to device B, or EV5 reads to device B, but specifically excluding the use of memory to signal, the following sequence works:

[1] this MB is not required if the EV5 store to the Device A and the EV5 load to device A are to the same address -- see 4 above.

[2] this MB is not required if the EV5 store to the Device A and the EV5 load to device A are to the same address -- see 4 above.

EV5 STORE I/O TO DEVICE A

MEMORY BARRIER OR WEAK MEMORY BARRIER

EV5 SIGNALS DEVICE B

DEVICE B RECEIVES SIGNAL

DEVICE B READ PEER DEVICE A

The signal must be EV5 writes to device B, or EV5 reads to device B. If the signal is EV5 reads to device B then Memory Barrier must be used; Weak Memory Barrier is not enough.

WHY IT WORKS IN the AlphaStation 600 system

Because of the Memory Barrier (or Weak Memory Barrier in the case of two I/O writes) EV5 emits the write to device A before the I/O reference that constitutes the signal. The AlphaStation 600 system processes the two EV5 I/O references in order so the write to device A goes on the PCI before the signal.

By the Triangle Inequality for I/O busses, the write to device A arrives at device A before the read command from device B arrives at device A, that read command being in response to the signal arriving at device B.

6. Write, device to memory; Read, EV5 to memory

SOFTWARE RULES

To insure that an EV5 read to memory will obtain the data written to memory by a device, software must use the following sequence:

DEVICE WRITES TO MEMORY

DEVICE SIGNALS EV5

EV5 RECEIVES SIGNAL

MEMORY BARRIER

EV5 LOADS FROM MEMORY

The device may signal EV5 by interrupting, writing a location in memory, or by EV5 reading to the device.

In the case that the device signals EV5 by interrupting, WE REQUIRE THAT EV5 READS A REGISTER IN THE DEVICE to "receive the signal". This device register read can be a read to verify the source of an interrupt or to determine the cause of the interrupt or the device status. If none of these things are necessary, then EV5 must read any register in the device anyway.

The device read will most likely be performed by the low-level interrupt handler: (1) for an EISA interrupt, the PCI Int-Ack command will be issued to read the EISA interrupt vector out of the PCI-EISA bridge chip. This Int-Ack cycle implicitly flushes the the EISA-to-PCI line buffers in the bridge chip; (2) For a PCI interrupt then the device must be read if it contains posted write buffers (or if it is located behind a PCI-to-PCI bridge). The interrupt handler may do this read when determining the source of the interrupt (in case the interrupts are wire-ORed).

For the purposes of this analysis, this makes the case of the device signaling by means of interrupting the same as the case of signaling by means of EV5 reading a register. That is, for purposes of analysis, it makes no difference if EV5 read a register because of an interrupt or for any other reason.

WHY IT WORKS IN the AlphaStation 600 system

First consider signaling by EV5 reading a register in the device, including the case of signaling with an interrupt.

The device will not signal until the device memory write has finished on the Bus segment connected to the device. This means that the write data is at least buffered in the first bridge, if it is not any further. Here a bridge could be the EISA to PCI bridge or the PCI to PCI bridge. With either type of bridge, the write data from the device and the read return data following it stay in order. Either bridge will retry a read if the write buffers are not empty, thereby keeping the order without causing a deadlock.

We can therefore be sure that the device write to memory will arrive at the CIA in an earlier PCI cycle than the returning data to complete the signal to the EV5.

Since the device write to memory has arrived at the CIA before the read return data, the device write to memory will be processed first. This means that the device write to memory will cause an EV5 flush command and it will own memory before the read return data is returned to EV5 (Basic Properties 7). This means it is also before EV5 can complete the Memory Barrier Instruction and hence before EV5 can probe its cache for the following memory read, or issue a miss for that memory read.

Any EV5 miss after the Flush will get the result of this device write since memory was owned for the device write before the Flush command was issued to EV5.

Next consider the device signaling EV5 by writing a location in memory. In this case the device does two writes to memory. These two writes will stay in order through PCI's and bridges and the AlphaStation 600 logic, all the way to memory. EV5 flushes will be in order. In fact EV5 will get the first Flush, then the first memory write will happen atomically, then EV5 will get the second flush, then the second memory write will happen atomically (Basic Properties 7).

7. Write, device to memory; Read, device to memory

FROM SAME DEVICE

The write and the read from the same device to memory stay in order through the whole system. So this works for any single device.

FROM DIFFERENT DEVICES

SOFTWARE RULES

For one device to correctly read data from memory that was written to memory by another device, software must use the following sequence:

DEVICE A WRITES TO MEMORY

DEVICE A SIGNALS DEVICE B

DEVICE B RECEIVES SIGNAL

DEVICE B READS MEMORY

We consider here only the case in which the signal is device A writing memory, or device A doing a peer to peer read or write to device B through the AlphaStation 600 I/O system.

The case that is NOT considered is that there might be some private interconnect, not part of the AlphaStation 600 design, between the devices. If there is such a thing, then coherency is the responsibility of the provider of such private interconnect.

Of course there is also the case in which device A signals a 3rd party, and the 3rd party signals device B. The case where the 3rd party is EV5 is the most important such case but not the only possible case. This works provided EV5 (or other device) receives the signal in a manner that would allow it itself to read the data from device A, and then signals device B in a manner that would work if it itself had written the data.

WHY IT WORKS IN the AlphaStation 600 system

In the case of signaling through memory, this works because two writes to memory from the same device stay in order.

Signaling via peer reads or peer writes works because of the Triangle Inequality for I/O busses (see Appendix A).

Suppose device A writes memory, then device A writes device B to signal it and then device B reads memory. By the triangle inequality device A write to memory arrives at the CIA before the read command from B induced by the message from A can get to the CIA. The memory commands at the CIA are processed in order, so the read gets the right data.

Now suppose device A writes memory, then device B reads device A and determines that the write was done, then device B reads memory. By the triangle inequality, the device A's write to memory arrives at the CIA before the read command from B where that read was a result of read return data from A to B. Memory commands arriving at the CIA are processed in order so you get the right result.

8. Write, device peer; Read, EV5 I/O to device

SOFTWARE RULES

For EV5 to read data from device B that was written to device B by device A, the following sequence must be used:

DEVICE A WRITES TO DEVICE B

DEVICE A SIGNALS EV5

EV5 RECEIVES SIGNAL

MEMORY BARRIER[1]

EV5 READS DEVICE B

Device A may signal EV5 with an interrupt, by writing a location in memory, or by EV5 reading device A. In the event that an interrupt is the signal WE REQUIRE THAT EV5 READS A REGISTER IN DEVICE A for EV5 to "receive the signal". This read may be to determine the source of the interrupt or the reason for the interrupt or to get status. If none of these things are necessary then EV5 must read some register in device A anyway. This makes signaling via interrupt and signaling via EV5 reading device A essentially the same.

[1] MB is required to keep the EV5 read for the "EV5 receives signal" and the "EV5 reads Device B" ordered.

WHY IT WORKS IN the AlphaStation 600 system

This works because of the Triangle Inequality for I/O busses.

Assume the signal is EV5 reading device A (this includes interrupt). By the triangle inequality device A's write to B arrives at B before the EV5 read command arrives at B, the EV5 read command being in response to read return data from A to EV5.

Assume the signal is writing a location in memory. By the triangle inequality, device A's write arrives at B before EV5's read, that read being in response to write data arriving at CIA from device A.

9. Write, device peer; Read, device peer

SAME DEVICE

If device A writes device B and then device A reads back from B, this works because only one command goes on the bus at a time and the bus system maintains ordering.

DIFFERENT DEVICES

SOFTWARE RULES

To insure that device C correctly reads data from device B that was written to device B by device A, the following sequence must be used:

DEVICE A WRITES TO DEVICE B

DEVICE A SIGNALS DEVICE C

DEVICE C RECEIVES SIGNAL

DEVICE C READS DEVICE B

We consider the case of signaling by writing a location in memory or through reads or writes on the AlphaStation 600 I/O system.

The case that we do not deal with is that there could be a private interconnect, not part of the AlphaStation 600 I/O system, between devices A and C. The signal could go on this private interconnect. If this is the case then coherency is the responsibility of the provider of such interconnect.

WHY IT WORKS IN the AlphaStation 600 system

This can be shown to work by using the Triangle Inequality for I/O busses. Examples of this appear above.

Data overwrites some later write.

The possible decomposed cases are:

1. Write, EV5 to memory; Write, EV5 to memory (same location)

SOFTWARE RULES

In order to insure that writes to memory stay in order, the following sequence must be used:

EV5 STORE TO MEMORY

EV5 STORE TO MEMORY

WHY IT WORKS IN the AlphaStation 600 system

We assume internally EV5 does the right thing. The AlphaStation 600 system gets involved only if the block goes to memory as a victim, or is extracted with a Flush Command. the only interesting case is where that happens between the two stores. In this case EV5 must get the block back again with a miss in order to update it for the second write. The issue becomes simply, can the AlphaStation 600 system correctly process a miss with victim and a following miss requesting that victim data. This was dealt with under 1. in the write - read interactions.

2. Write, EV5 to memory; Write, device to memory (same location)

SOFTWARE RULES

In order to insure that a device will correctly overwrite data written to memory by EV5, software must use the following sequence:

EV5 STORE TO MEMORY

WEAK MEMORY BARRIER OR MEMORY BARRIER

EV5 SIGNALS DEVICE

DEVICE RECEIVES SIGNAL

DEVICE WRITES MEMORY

The EV5 can signal the device only by writing something to memory or by writing something to a device or by reading a device. In the case of signaling by reading a device, Memory Barrier must be used, Weak Memory Barrier is not enough.

WHY IT WORKS IN the AlphaStation 600 system

Under these conditions, by the time the signal leaves the EV5 chip, the write is completed. The result of the write is either in the cache and will be extracted by a Flush command, or the data has been sent to memory in a miss with victim or in response to a previous flush.

Data from EV5 in a flush buffer or en route to a flush buffer is as good as in memory, relative to another device write. Two device write sequences are mutually atomic (Basic Properties 8). If the device write in question is before another device write sequence, then its Flush command will extract the result of the EV5 write from EV5 and the device data will merge on top of it. If it is after another device write sequence, the result of the EV5 write will be obtained from memory and the device data will be merged on top of it.

Data in the victim buffer is as good as in memory. A device write to memory could be for the block that is in the victim buffer or is en route to the victim buffer and that such a device write must merge on top of the value that is in the victim buffer or en route to the victim buffer. The details are as follows: the EV5 retains (and still owns) the victim data, even though a copy may currently reside in the victim buffer, until the EV5 memory read is completed. Hence a device write will FLUSH the data out of the EV5 if the EV5 read has not yet completed, and the data in the victim buffer will be invalidated; and if the EV5 read has completed, then the victim data is in memory since the AlphaStation 600 system ensures that the victim write is atomic with the associated EV5 read fill.

It is ,however, NOT true that data in the victim buffer is as good as in memory. A device write to memory could be for the block that is in the victim buffer or is en route to the victim buffer and that such a device write must merge on top of the value that is in the victim buffer or en route to the victim buffer.

3. Write, EV5 I/O to the AlphaStation 600 CSR; Write, EV5 I/O to AlphaStation 600 CSR

SOFTWARE RULES

To correctly sequence two writes to the same AlphaStation 600 CSR, software must use the following sequence:

EV5 STORE I/O TO CSR

EV5 STORE I/O TO CSR

Due to the fact that the addresses of the two stores match, EV5 will either correctly merge them into a single write block or else EV5 will emit the writes in order. The AlphaStation 600 system will do only 1 operation on any given CSR at a time and in the order in which the EV5 emitted them.

4. Write, EV5 I/O to device; Write, EV5 I/O to device

SOFTWARE RULES

In order to correctly sequence two writes to the same device register, software must use the following sequence:

EV5 STORE I/O TO DEVICE

EV5 STORE I/O TO DEVICE

Due to the fact that the addresses of the two stores match, EV5 will either correctly merge them into the same write block or else EV5 will emit the writes in order. I/O commands from EV5 go to the PCI strictly in the order in which they are emitted by the EV5. PCI transactions from EV5 to any given target stay strictly in order through bridges and what have you.

5. Write, EV5 I/O to device; Write, device peer

SOFTWARE RULES

To insure that an EV5 write to a device is properly sequenced with a write to that device by a peer, the following sequence must be used:

EV5 STORE I/O TO DEVICE A

MEMORY BARRIER[1]

EV5 LOAD I/O TO DEVICE A

MEMORY BARRIER

EV5 SIGNALS DEVICE B

DEVICE B RECEIVES SIGNAL

DEVICE B WRITES DEVICE A

The only possibilities for the signal are EV5 writes memory, EV5 writes to device B, or EV5 reads to device B.

[1] this MB is not required if the EV5 store to the Device A and the EV5 load to device A are to the same address.

EV5 Load I/O to device A need not be to any particular register or location. It can be to anything in device A. The general principle is that the one and only mechanism that EV5 has to determine that an I/O write has arrived at a device is to read something (anything) from that same device. EV5 cannot signal device B that the data is at device A until it knows itself that the data is at device A.

WHY IT WORKS IN the AlphaStation 600 system

The EV5 write and read to device A will be emitted in the order shown by EV5 because of the Memory Barrier between them. They will go to the PCI in order and arrive at device A in order. Hence we do not get past the second memory barrier until the EV5 write to device A has actually reached the device. Hence device B could not receive the signal, no matter what it is, until after the EV5 write to device A has reached device A. The device B write to device A then certainly reaches device A after the EV5 write to device A does. After that it is up to device A to do the right thing.

SPECIAL CASES:

SOFTWARE RULES

In the special case that the signal is either EV5 writes to device B, or EV5 reads to device B, but specifically excluding the use of memory to signal, the following sequence works:

EV5 STORE I/O TO DEVICE A

MEMORY BARRIER OR WEAK MEMORY BARRIER

EV5 SIGNALS DEVICE B

DEVICE B RECEIVES SIGNAL

DEVICE B WRITES DEVICE A

The signal must be EV5 writes to device B, or EV5 reads to device B. If the signal is EV5 reads to device B then Memory Barrier must be used; Weak Memory Barrier is not enough.

WHY IT WORKS IN the AlphaStation 600 system

Because of the Memory Barrier (or Weak Memory Barrier in the case of two I/O writes) EV5 emits the write to device A before the I/O reference that constitutes the signal. The AlphaStation 600 system processes the two EV5 I/O references in order so the write to device A goes on the PCI before the signal.

By the Triangle Inequality for I/O busses, the write to device A arrives at device A before the write command from device B arrives at device A, the write from B being in response to the signal arriving at device B.

6. Write, device to memory; Write, EV5 to memory

SOFTWARE RULES

To insure the correct sequencing of a device writing to memory and EV5 writing to the same location the following sequence must be used:

DEVICE WRITES MEMORY

DEVICE SIGNALS EV5

EV5 RECEIVES SIGNAL

MEMORY BARRIER

EV5 WRITES MEMORY

The device may signal EV5 by interrupting, writing a location in memory, or by EV5 reading to the device.

In the case that the signal is an interrupt WE REQUIRE THAT EV5 READS A REGISTER IN THE DEVICE to "receive the signal". This device register read can be a read to verify the source of an interrupt or to determine the cause of the interrupt or the device status. If none of these things are necessary, then EV5 must read any register in the device anyway.

For the purposes of this analysis, this means that the case of signaling with an interrupt is the same as the as the case of signaling by EV5 reading the device. For this analysis it makes no difference if EV5 read the device in response to an interrupt or for any other reason.

WHY IT WORKS IN the AlphaStation 600 system

First, consider the case of signaling by EV5 reading the device, including the interrupt case.

The device will not signal until the device memory write has finished on the Bus segment connected to the device. This means that the write data is at least buffered in the first bridge, if it is not any further. Here a bridge could be the EISA to PCI bridge or the PCI to PCI bridge. The read return data from a device cannot get ahead of write data from the device that started ahead of it, with either bridge. Either bridge will retry a read if the write buffers are not empty, thereby keeping the order without causing a deadlock.

We can therefore be sure that the device write to memory will arrive at the CIA in an earlier PCI cycle than the returning data to complete the signal to the EV5.

Since the device write to memory has arrived at the CIA before the read return data, the device write to memory will be processed first. This means that the device write to memory will cause an EV5 flush command and it will own memory before the read return data is returned to EV5. This means it is also before EV5 can complete the Memory Barrier Instruction and hence before EV5 can execute its write to memory.

Next consider the device signaling EV5 by writing a location in memory.

In this case the device does two writes to memory. These two writes will stay in order through PCI’s and bridges and the AlphaStation 600 system logic, all the way to memory. EV5 flushes will be in order. In fact EV5 will get the first Flush, then the first memory write will happen atomically, then EV5 will get the second flush, then the second memory write will happen atomically.

7. Write, device to memory; Write, device to memory

SAME DEVICE

Two writes from the same device to memory stay in order through the whole system. So this works for any single device.

DIFFERENT DEVICES

SOFTWARE RULES

To insure proper sequencing of two devices writing to the same location in memory, the following sequence must be used:

DEVICE A WRITES MEMORY

DEVICE A SIGNALS DEVICE B

DEVICE B RECEIVES SIGNAL

DEVICE B WRITES MEMORY

We will consider only the case where the signaling mechanism is another write to memory or a peer read or write through the AlphaStation 600 I/O system.

There is also the possibility that device A signals a 3rd party and then the 3rd party signals device B. The most important case of this is when the 3rd party is EV5, but this is not the only such case. This works if EV5 (or other device) receives the signal in such a way that it could itself properly write over the data, and then signals in such a way that device B could correctly overwrite the data if it itself had written it.

The case we will NOT consider is that there might be some private interconnect, not part of the AlphaStation 600 design, between the devices. If there is such a thing, then coherency is the responsibility of the provider of such private interconnect.

WHY IT WORKS IN the AlphaStation 600 system

In the case of signaling through memory, this works because two writes to memory from the same device stay in order.

Signaling via peer reads or peer writes works because of the Triangle Inequality for I/O busses.

Suppose device A writes memory, then device A writes device B to signal it and then device B writes memory. By the triangle inequality device A write to memory arrives at the CIA before the write command from B induced by the message from A can get to the CIA. The memory commands at the CIA are processed in order.

Now suppose device A writes memory, then device B reads device A and determines that the write was done, then device B writes memory. By the triangle inequality, the device A's write to memory arrives at the CIA before the write from B where B's write was a result of read return data from A to B. Memory commands arriving at the CIA are processed in order so you get the right result.

8. Write, device peer; Write, EV5 I/O to device

SOFTWARE RULES

To insure correct sequencing of device A write to device B and EV5 write to device B, the following sequence must be used:

DEVICE A WRITES TO DEVICE B

DEVICE A SIGNALS EV5

EV5 RECEIVES SIGNAL

MEMORY BARRIER

EV5 WRITES DEVICE B

Device A may signal EV5 with an interrupt, by writing a location in memory, or by EV5 reading device A. In the event that an interrupt is the signal WE REQUIRE THAT EV5 READS A REGISTER IN DEVICE A for EV5 to "receive the signal". This read may be to determine the source of the interrupt or the reason for the interrupt or to get status. If none of these things are necessary then EV5 must read some register in device A anyway. This makes signaling via interrupt and signaling via EV5 reading device A essentially the same.

WHY IT WORKS IN the AlphaStation 600 system

This works because of the Triangle Inequality for I/O busses.

Assume the signal is EV5 reading device A (this includes interrupt). By the triangle inequality device A's write to B arrives at B before the EV5 write arrives at B, the EV5 write being in response to read return data from A to EV5.

Assume the signal is writing a location in memory. By the triangle inequality, device A's write arrives at B before EV5's write, EV5's write being in response to write data arriving at CIA from device A.

9. Write, device peer; Write, device peer

SAME DEVICE

If device A writes device B and then device A writes device B again, this works because only one command goes on the bus at a time and the bus system maintains ordering.

DIFFERENT DEVICES

SOFTWARE RULES

To insure that writes from two different devices are sequenced properly at device B, the following sequence must be used:

DEVICE A WRITES DEVICE B

DEVICE A SIGNALS DEVICE C

DEVICE C RECEIVES SIGNAL

DEVICE C WRITES DEVICE B

We consider only cases where the signaling mechanism is writing a location in memory or through reads or writes on the AlphaStation 600 I/O system.

There is also the possibility that device A signals a 3rd party and then the 3rd party signals device C. The most important case of this is when the 3rd party is EV5, but this is not the only such case. This works if EV5 (or other device) receives the signal in such a way that it could itself properly write over the data, and then signals in such a way that device B could correctly overwrite the data if it itself had written it.

The case that we do not deal with is that there could be a private interconnect, not part of the AlphaStation 600 I/O system, between devices A and C. The signal could go on this private interconnect. If this is the case then coherency is the responsibility of the provider of such interconnect.

WHY IT WORKS IN the AlphaStation 600 system

This can be shown to work by using the Triangle Inequality for I/O busses. Examples of this appear above.

Data is overwritten before readers are finished with it

Here we have a situation in which a reader signals when he is done with some memory data and then a writer gets the signal and then writes the location. Assuming the reader does not signal until it actually has all the data it wants, it is pretty obvious that this will work regardless of the design details. The only tiny detail is, if EV5 is the reader, to be sure the EV5 actually has the data before signalling. A Memory Barrier between the last load and the signal will always do it. There may well be cases when the Memory Barrier is not needed.

Data fails to make it to destination

SOFTWARE RULES

In the AlphaStation 600 System[1], the only way for a program to determine that an I/O write has made it to a device, is for the program to read a register in that device. The following sequence may be used:

EV5 STORE I/O TO DEVICE ADDRESS A

EV5 LOAD I/O FROM DEVICE ADDRESS A

the program can now be certain the write has reached device A

Here the store and the load must be to the same address.

The following sequence may also be used:

EV5 STORE I/O TO DEVICE A

MEMORY BARRIER

EV5 LOAD I/O FROM DEVICE A

the program can now be certain the write has reached device A

[1] and any PCI-based system for that matter

It is NOT necessary to read the same register that was written but it is required that a register in the same DEVICE be read. This brings up the issue of what is a device?

The AlphaStation 600 ASIC chipset is considered one device for these purposes. This chipset consists of 4 DSW chips and one CIA chip. These chips contain certain CSRs. If any CSR in the chipset is written, the only way to be sure that the write is effective is to do one of the above sequences. In particular, if you write a CSR with a side effect of interacting with following read or write I/O's, (the HAE register which affects addressing of following I/O references is an example) you can neither be sure that it will be effective nor be sure that it won't be effective on an I/O reference until after a read is done to some CSR in the AlphaStation 600 chipset.

Normally, one need not follow every write by a read; the read can terminate a long sequence of CSR writes, since all write are kept in order by the AlphaStation 600 system, and a read for the last write will ensure that all prior writes completed. For an example of when an IO read is required see the section "Failure to use latest IO page table state" a few pages forward.

Each PCI to PCI bridge is considered a device, for these purposes. This includes any bridges on the system board and also any bridges that are on PCI option modules. Currently the plan is to have an SCSI-Ethernet PCI option module that will appear on most AlphaStation 600 units. This module has a PCI to PCI bridge followed by SCSI adapters and an Ethernet adapter. The bridge on this PCI option is considered a device by itself.

The PCI to EISA/ISA bridge is considered a device, for these purposes.

Each load on a PCI or EISA bus is considered a device.

There is some flexibility in what is defined to be a device but you get what you deserve. An example is in order and the AlphaStation 600 SCSI-Ethernet PCI module is ideal for an example.

This module contains three independent PCI to SCSI adapters, each one being a separate chip, and one Tulip ethernet chip, behind a PCI to PCI bridge. If you want to, you may define that the entire module is one device. According to the rules, you write a CSR in one of the chips, for example, in the Tulip chip. Then, after a Memory Barrier you read any CSR anywhere on the module to verify delivery of the write data. The result of doing this is that your read only insures that the write data was delivered to what you defined as the device, in this case the whole module. You can be sure that your write data reached the module, but you cannot be sure it has reached the Tulip chip in particular. If you want the write data to reach the Tulip chip specifically, you must read some register in the Tulip chip. This effectively means you are defining the Tulip chip to be the device.

To a large extent, for the purposes of this discussion, you may define what ever you want to be devices. Then the rule that to insure delivery of write data to the device, read any register in the device will work exactly as you define it. If you define something to be the device then you cannot distinguish features inside the device.

WHY IT WORKS IN the AlphaStation 600 system

In the first case, the store and the load are emitted in order by EV5 because their addresses match and they are in non cached space. In the second case, the store and the load are emitted by EV5 because of the Memory Barrier between them. Commands go through the I/O system in order. By the time the Read reaches a device, the Write has already arrived first.

Side Effects happen out of order

1. Read and write induced side effects

For Read and Write induced side effects, keeping side effects in order comes down to making sure the reads and writes arrive at the device in order. This is the same problem that was extensively treated by the preceding analysis of write - read and write - write interactions.

2. EV5 Writes I/O to a device, then writes memory as a signal.

There is, however, the case of a write to a device that is signaled via a write to memory, later examined by the device. This can be thought of as a strange case of two side effects interacting, in that both events do not fit the memory model of only worrying that something that was written can be correctly retrieved. Had this fit the memory model this case would have showed up among the previously presented memory model cases.

Signaling with reads or writes to the device is not interesting because this again simply reduces to keeping reads and writes in order. Only signaling through memory is new.

SOFTWARE RULES

To insure that write data is at a device when the signal that it is there arrives, when that signal is through memory, one of the two following sequence must be used. Sequence 1:

EV5 STORE I/O TO DEVICE ADDRESS A of DATA X

EV5 LOAD I/O FROM DEVICE ADDRESS A

MEMORY BARRIER

EV5 WRITES MEMORY

DEVICE READS MEMORY

DEVICE REQUIRES PRESENCE OF WRITTEN DATA X

In this case the store and load must be to the same address.

Sequence 2:

EV5 STORE I/O TO DEVICE of DATA X

MEMORY BARRIER

EV5 LOAD I/O FROM DEVICE

MEMORY BARRIER

EV5 WRITES MEMORY

DEVICE READS MEMORY

DEVICE REQUIRES PRESENCE OF WRITTEN DATA X.

This follows the general principle that the only way EV5 can be sure that write data has reached the device is to read back a register in the same device. It is not necessary to read the same register that was written. The read can be to any register in the same device. EV5 cannot signal the device that the data is present until it itself knows this.

WHY THIS WORKS IN the AlphaStation 600 system

EV5 emits the store and the load in order, either because their addresses match and they are in noncached space, or because of the Memory Barrier between them. By the time the Memory Barrier after the load completes, the write data will be at the device. EV5 write to memory is after this memory barrier.

Analysis of I/O Page Table Modification Interactions

Failure to use the latest I/O Page Table State

SOFTWARE RULES

The required sequence to ensure that new I/O page table state is used is:

EV5 STORE TO I/O PAGE TABLE

WEAK MEMORY BARRIER

EV5 STORE TO I/O TBIA

MEMORY BARRIER

EV5 LOAD FROM ANY AlphaStation 600 CSR

MEMORY BARRIER

SIGNAL DEVICE

DEVICE RECEIVES SIGNAL

DEVICE DOES READ OR WRITE TO MEMORY USING NEW PAGE TABLE STATE

WHY IT WORKS IN the AlphaStation 600 system

By the time the write to TBIA has been emitted from EV5, a condition has been established in which either the new page table data will be obtained in any Flush or READ command to EV5, or else the new page table data has already moved out of EV5 in a miss with victim or in response to a previous Flush.

Any TB miss occurring after the Write to TBIA has been emitted from EV5 will do a Read to EV5 after the Write to TBIA was emitted. Either the Read to EV5 will obtain the new page table data or that data is in memory or that data is in a flush buffer or that data is in the victim buffer.

If the data is in memory the TB miss memory cycle will get it.

Data in a Flush buffer is as good as in memory, relative to the TB miss. The two-step TB miss sequence and the two-step Flush sequence are mutually atomic by Basic property 8. Either the TB miss sequence will be entirely before the flush sequence, in which case the data will come from the Read Command to EV5, or the TB miss sequence will be entirely after the Flush sequence, in which case the data will come from memory.

As explained in earlier section, data in the victim buffer is as good as in memory. A TLB miss will obtain the correct value, even if the data is in the victim buffer.

At this point we have shown that a TB miss occurring after the write to TBIA was emitted from EV5 will retrieve the new page table state. There is still the issue of will a TB miss happen when appropriate?

The write to TBIA invalidates the TB but a TB miss can result in validating an entry again, depending on the page table state that was found. It must be shown that it is not possible that a TB entry gets validated again after a TBIA without getting the new page table state written to it.

In a TB miss, I/O writes will not be processed from the Read command going to EV5 for the page table data until after the new state is put in the page table entry and the page table entry is marked valid. Hence any write to TBIA is processed before the TB miss issues a Read command to EV5 (so that new page table state will be obtained), or else it is after the TB entry has been set valid so that the TB entry will be left invalid.

There is then a Memory Barrier and a read to an AlphaStation 600 CSR, and a following Memory Barrier. This sequence insures that by the time the last Memory Barrier completes, the TB has been invalidated. Hence, whatever the signal is, by the time the signal is emitted from EV5, the TB has been invalidated and any device memory reference will either hit an invalid TB entry or the new page table state in that entry.

SPECIAL CASES:

SOFTWARE RULES

In the special case that the signal to the device is an EV5 Read I/O to the device or an EV5 Write I/O to the device, but specifically excluding the use of memory to signal, the following faster sequence can be used:

EV5 STORE TO I/O PAGE TABLE

WEAK MEMORY BARRIER

EV5 STORE TO I/O TBIA

MEMORY BARRIER OR WEAK MEMORY BARRIER

EV5 SIGNALS DEVICE

DEVICE RECEIVES SIGNAL

DEVICE DOES READ OR WRITE TO MEMORY USING NEW PAGE TABLE STATE

The signal must be EV5 reds or writes to the device. If the signal is a read to the device then Memory Barrier must be used; Weak Memory Barrier is not enough.

WHY IT WORKS IN the AlphaStation 600 system

The argument that the TB will get invalidated and conditions will be such that if the TB entry becomes valid again, then it will represent the new page table state is the same as in the general case above. the difference here is that we must show that if the device references memory in response to the signal, then it will effectively see the new page table state.

The signal is an EV5 I/O read or write to the device. The AlphaStation 600 system processes EV5 I/O reads and writes in the order received from the EV5, which will be Write to TBIA and then Signal Device because of the Weak (or full) Memory Barrier between them. By the time the PCI transaction to do the signal completes, the TB has been invalidated. Since there is only one PCI transaction to the CIA at a time, the device read or write to memory follows the signal PCI transaction.

I/O Page Table Changed While In Use

SOFTWARE RULES

To insure that an I/O page table entry is no longer being used when its state is changed, one of the following two sequences must be used. Sequence 1:

DEVICE SIGNALS COMPLETION OF USE OF A PAGE TABLE ENTRY

EV5 RECEIVES SIGNAL

MEMORY BARRIER

EV5 STORE TO I/O PAGE TABLE ENTRY

The signal may be an interrupt, a device write to memory, or EV5 reading the device. In the event that the signal is an interrupt, WE REQUIRE THAT EV5 READS A REGISTER IN THE DEVICE to "receive the signal". Hence the case of an interrupt and of signaling by EV5 reading the device are the same.

WHY IT WORKS IN the AlphaStation 600 system

Assume the signal was EV5 reading the device (including interrupt). Then any PCI transaction using the page table entry would have gone on the PCI before the read data to EV5. Hence it would have arrived at the CIA before any read data to EV5. The device read or write to memory would be processed first because it arrived first. The device read or write to memory would go through the TB before it does the Read or Flush to EV5. If there is a TB miss this will be resolved before the Read or Flush to EV5. The read return data to EV5 will be after the Read or Flush for the device read or write to memory. All TB activity is therefore complete before the Memory Barrier can complete.

Assume the signal is that the device writes to memory. EV5 must read the signal from memory to receive the signal. We now have two device memory references, the device read or write using the page table entry, and the device write to memory constituting the signal. Each of these memory references is a two-step sequence in which the EV5 is probed and a memory cycle happens. They are atomic relative to each other (Basic Property 8).

The device references are processed in order so the device memory reference sequence will be entirely before the device write to memory for the signal sequence.

Therefore the preceding memory references will have gone through the TB and be in memory before the signal reference either Flushes EV5 or writes memory. Hence before EV5 could receive the signal and before the following Memory Barrier could finish, so before the page table is modified.

Note that the signal write itself might use the page table entry that is about to be changed. The signal write will go through the TB and this will be resolved before the signal write does a Flush to EV5 or writes memory. As above, this must be before the page table is modified.

SOFTWARE RULES

Sequence 2:

EV5 STORE I/O TO DEVICE TO DISABLE USE OF PAGE TABLE ENTRY

MEMORY BARRIER

EV5 LOAD I/O FROM DEVICE

MEMORY BARRIER

EV5 STORE TO THE PAGE TABLE ENTRY

The device spec must say that the device will not do a PCI transaction referencing the interesting virtual address following the PCI transaction that wrote a disabling state to it.

WHY IT WORKS IN the AlphaStation 600 system

As has already been argued above, by the time the second Memory Barrier completes, the EV5 write to the device has been delivered. By the assumed device properties there will be no following memory reference using the interesting page table entry. Any device memory reference preceding delivery of the EV5 write to the device will arrive at the CIA before the read return data to EV5. Hence it will be processed first. It will pass the TB and any TB miss will be fully resolved before the read return data is given to EV5. This is before the second Memory Barrier could finish and so before the page table entry is modified.

Triangle Inequality for I/O Busses

We assume here that we have a set of PCI and EISA busses connected by two port bridges. The net is acyclic; that is, there are no closed loops. This implies that there is a single direct path from any device to any other device. A direct path is a path that does not go on any single bus more than once, or through any given bridge more than once.

Define a message from X to Y to be any command, read or write, initiated by X with Y as target, and in addition, if Y initiates a read from X as a target, the read return data from X to Y is considered a message.

We assume each bridge has the property that messages flowing through it in a given direction are kept strictly in order. This is in fact true of the PCI to PCI bridge being developed in DEC. This is only partially true in the EISA/ISA bridge: for writes greater than a Dword in size the writes are ordered; all smaller writes may get merged together (merging will only occur if the Dword in which the data will merge is still in the buffer, and has not been expelled by some other system/chip event).

Let A, B and C be any three devices where each device connects to one and only one bus. A, B, and C can connect to any busses in the net.

Suppose device A sends a message to device C, and after that device A sends a message to device B. After device B receives the message from A it sends a message to device C. Then, exactly as you might expect, device C receives the message directly from A before it receives the message from B from A.

Proof:

There is a single direct path from A to C. This is the path that the direct message from A to C takes. This direct path starts with the bus A is on and goes through 0 or more bridges and winds up with the bus C is on. Let the sequence of bridges be B1, B2, B3, ...

Considering the combined path of the messages from A to B and then to C, on one leg or the other, this path must go through the same bridges B1, B2, B3, ... This path might however contain a closed loop; That is, part of the path from A to B to C might start at a bridge z and eventually get back to bridge z. If there are any such closed loops, remove them, so no bridge appears more than once. The remaining sequence not only goes through B1, B2, B3, ..., but in fact goes through these bridges in the same order as the direct path.

Observe that the path from A to B to C must be fully connected, of course. If a loop starting at Bz and ending at Bz is removed, leaving just the single visitation of bridge Bz then the path is still connected. Now however the path is a direct path because it doesn't hit the same bridge twice. And it goes from A to C. But there is only one unique direct path from A to C, so this must be it.

Hence the message from A to B to C must follow the exact same path B1, B2, B3, ... that the direct path from A to C followed, except that there may have been some circular side trips inserted somewhere.

The direct message from A to C must get to bridge B1 before the indirect message does. The reason is that it goes on the bus first so in fact gets to B1 before the indirect message even gets on the bus to go anywhere.

A

CIA - DSW Command Fields (CMC and IOC)

Introduction

The DSW is relatively simple and only reacts to commands from the CIA. This approach helped keep the complexity in one chip. There are two independent command fields that control DSW.

- IOC<6:0> - this bus mostly controls data movement on the IOD bus.
- CMC<8:0> - this bus mostly controls the data moving between EV5 and DSW and Memory and DSW.

IOC<6:0> - IOD control bus. The CIA uses IOC to control the data buffers in the DSW and the direction of data flow between CIA and DSW. The encoding of the IOC<6:0> bits are as follows:

IOC[6:4]	IOC<3>	IOC<2>	IOC<1>	IOC<0>	EXPLANATION
000	Buffer n	X	X	X	Clear valid bits in PCI buffer n
001	X	X	X	X	NOP
010	Buffer n	Addr m2	Addr m1	Addr m0	Write PCI buffer n, addr m
011	Mask3	Mask2	Mask1	Mask0	Write IOR (QW mask)
100	Buffer n	Addr m2	Addr m1	Addr m0	Read DMA buffer n, addr m
101	Buffer n1	Buffer n0	Addr m1	Addr m0	Read IOW buffer n, addr m
110	X	X	X	X	Reserved
111	Mask3	Mask2	Mask1	Mask0	Start IOW (buffer mask)

IOC[6] is used by the Data Switch to control the tri-state drivers on the IOD bus. When IOC[6] = 1, the CIA is driving the IOD bus (or the bus may be tri-state). When IOC[6] = 0, the DSW is driving the IOD bus.

When idle, the CIA drives the command Read IOW buffer n, onto the IOC bus.

CMC<8:0> - The CMC is used to control data movement between EV5 DSW and Memory.

Each of the CMC Commands is described below:

- **Clear.** CMC[8:5] = 0000. The Clear command is used to start, stop, or clear buffers when no data movement is required. The variable specifies the action to be taken.

— CMC[4:3] specify optional actions as follows:

CMC[4:3]	Action
00	None
01	Start the victim buffer
10	Start flush buffer 0 and Clear PCI buffer 0
11	Start flush buffer 1 and Clear PCI buffer 1

— CMC[2:0] specify optional actions as follows:

Var<2:0>	Action
000	None
001	Stop Victim
010	Stop Flush Buffer 0
011	Stop Flush Buffer 1
100	Stop IOW 0
101	Stop IOW 1
110	Stop IOW 2
111	Stop IOW 3

- **Fill from Memory**. CMC[8:5] = 0010. The Fill from Memory command instructs the DSW to latch data on the memory bus and drive it onto the CPU data bus.
  - — CMC[3] is used to indicate which half of the 256-bit memory data bus onto the 128 bit CPU data bus. It reflects the wrap order (address A4) of the Read Miss command from the CPU.

CMC[3]	Action
0	drive MEM_DAT[127:0] followed by MEM_DAT[256:128] next cycle
1	drive MEM_DAT[256:128] followed by MEM_DAT[127:0] next cycle

— CMC[2] is used to indicate whether the memory data should be loaded in DSW on the rising of the SYS CLK or delayed by 1/2 SYS CLK cycle. If CMC[2] = 0, the memory data is loaded on the rising edge of SYS CLK. Otherwise, DSW delays loading the data for 1/2 SYS CLK cycle.

— CMC[1:0] specify optional actions as follows:

CMC[1:0]	Action
00	None
01	Stop Victim buffer
10	Stop Flush buffer 0
11	Stop Flush buffer 1

- **Fill from IOR**. CMC[8:5] = 0011. The Fill from IOR command instructs the DSW to drive the CPU data bus with data from the I/O Read buffer.
  - — CMC[3] is used to indicate whether to send quadwords 0,1 or 2,3 first. It reflects the wrap order (address A4) of the Read Miss command from the CPU.

CMC[3]	Action
0	drive quadwords 0,1 first, then quadwords 2,3 next cycle
1	drive quadwords 2,3 first, then quadwords 0,1 next cycle

  - — CMC[2] is not used.
  - — CMC[1:0] specify optional actions as follows:

CMC[1:0]	Action
00	None
01	Stop Victim buffer
10	Stop Flush buffer 0
11	Stop Flush buffer 1

- **DMA Read 0**. CMC[8:5] = 0100. The DMA Read 0 command is used to move memory data into memory buffer 0.
  - — CMC[4:3] are used to specify which buffer location(s) to begin loading the memory data into.
  - — CMC[2] is used to indicate whether the memory data should be loaded in DSW on the rising of the SYS CLK or delayed by 1/2 SYS CLK cycle. If CMC[2] = 0, the memory data is loaded on the rising edge of SYS CLK. Otherwise, DSW delays loading the data for 1/2 SYS CLK cycle.
  - — CMC[1:0] specify optional actions as follows:

CMC[1:0]	Action
00	None
01	Stop Victim buffer
10	Stop Flush buffer 0
11	Stop Flush buffer 1

- **DMA Read 1**. CMC[8:5] = 0101. The DMA Read 1 command is used to move memory data into memory buffer 1. The variable field (CMC[4:0]) behave the same as the **DMA Read 0** command above.
- **DMA Read 0 and Stop IOW**. CMC[8:5] = 0110. This command is similar to the DMA Read 0 command. The difference being that CMC[1:0] specify which IOW buffer to stop.

— CMC[1:0] specify which IOW buffer to stop:

CMC[1:0]	Action
00	Stop IOW buffer 0
01	Stop IOW buffer 1
10	Stop IOW buffer 2
11	Stop IOW buffer 3

- **DMA Read 1 and Stop IOW**. CMC[8:5] = 0111. This command is similar to the DMA Read 0 and Stop IOW command. The difference being that data is loaded into memory buffer 1.
- **Write Victim**. CMC[8:5] = 1000. This command is used to move the Victim buffer contents to memory.
  — CMC[4:3] are used to specify which buffer location to begin loading the memory from. It reflects the wrap order (address[5:4]) of the Read Miss with Victim command from the CPU.
  — CMC[2:0] specify optional actions as follows:

Var<2:0>	Action
000	None
001	Stop Victim
010	Stop Flush Buffer 0
011	Stop Flush Buffer 1
100	Stop IOW 0
101	Stop IOW 1
110	Stop IOW 2
111	Stop IOW 3

- **DMA Write 0**. CMC[8:5] = 1010. DMA Write 0 instructs the DSW to drive the memory data bus with data from buffer 0. The DSW will pick the correct data from either the Flush, PCI, or Memory buffer based on the valid bits set in those buffers.
  — CMC[4:3] specify the data to be written as follows:

CMC[4:3]	Action
00	Write quadwords 0,1,2,3 to memory
01	reserved for 128-bit memory systems
10	Write quadwords 4,5,6,7 to memory
11	reserved for 128-bit memory systems

  — CMC[2:0] have the same encodings as the Write Victim command above.
- **DMA Write 1**. CMC[8:5] = 1011. DMA Write 1 instructs the DSW to drive the memory data bus with data from buffer 1. The DSW will pick the correct data from either the Flush, PCI, or Memory buffer based on the valid bits set in those buffers.
  — CMC[4:0] have the same encodings as the DMA Write 0 command.