ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Pentek 4205* at our website: *Click HERE*

OPERATING MANUAL

MODEL 4205

MPC7457 PowerPC Processor
VIM/PMC Carrier VMEbus Board

Pentek, Inc.
One Park Way
Upper Saddle River, NJ 07458
(201) 818–5900
http://www.pentek.com/

Copyright © 2007

Manual Part No: 800.42050 Rev: A.6 – March 8, 2007

Manual Revision History

Date	Revision	Comments
9/4/03	Preliminary	Preliminary manual
11/29/04	A	Removed Sect. 1.11, SwiftNet Software. Updated CPU Speed options, Sect 3.7.6. Changed jumper JB4 default state, Sect 2.2.4. Updated Monitor, Chapter 9, for new RedBoot commands. Added Option 007 Gigabit Ethernet Interface, Sects. 1.9, 1.10, 1.13, 2.5, 3.5.4, 4.5.10, 9.3, & 9 4.1. Updated Power Specifications, Sect 1.13. Corrected Jumper JB7 settings, Sect 2.2.6. Added VIM installation instructions for new heat sink, Sect 2.4.1. Corrected Reset Register to show PCI bus reset bits, Sect 3.7.2. Removed FPGA Configuration ID Registers, Sect 6.9 & 6.11. Added Option 150/151/152 PMC Extender, Sects. 1.10, 2.5.10, 3.5.7, 4.5.13, 8.3, 8.5, & Appendix C. Removed Option 055 (733 MHz) Processors, and made Option 57 (1 GHz MPC7457) standard.
12/6/04	A.1	Corrected definition of Jumper JB3, Sect 2.2.2, per KBCase 1236.
2/1/05	A.2	Separated front panel LED descriptions for each option, Sect 2.5.5. Added WatchDog Timer Register, Sect 3.7.8 (only for Rev 'P' PCB assemblies).
8/25/05	A.3	Updated Power ratings, Sect 1.14, per KBCase 1220.
1/31/06	A.4	Removed VME access to VIM BIFOs, Chapters 5 and 6.
11/9/06	A.5	Updated PCB Assembly drawing, Fig 2–1, for changes to jumper blocks in PCB Rev B and up.
3/8/07	A.6	Corrected PMC connector designations from P1n to J1n, Fig 2–1 and Sect 2.3.2.

Warranties

Pentek warrants that all products manufactured by Pentek conform to published Pentek specifications and are free from defects in materials and workmanship for a period of one year from the date of delivery when used under normal operating conditions and within the service conditions for which they were furnished. The obligation of Pentek arising from a warranty claim shall be limited to repairing or at its option, replacing without charge, any product that in Pentek's sole opinion proves to be defective within the scope of the warranty. Pentek must be notified in writing of the defect or nonconformity within the warranty period and the affected product returned to Pentek within thirty days after discovery of such defect or nonconformity. Buyer shall prepay shipping charges, taxes, duties and insurance for products returned to Pentek for warranty service. Pentek shall pay for the return of products to buyer except for products returned from another country.

Pentek shall have no responsibility for any defect or damage caused by improper installation, unauthorized modification, misuse, neglect, inadequate maintenance, accident, or for any product which has been repaired or altered by anyone other than Pentek or its authorized representatives.

The warranty described above is buyer's sole and exclusive remedy and no other warranty, whether written or oral, is expressed or implied. Pentek specifically disclaims fitness for a particular purpose. Under no circumstances shall Pentek be liable for any direct, indirect, special, incidental, or consequential damages, expenses, losses, or delays (including loss of profits) based on contract, tort, or any other legal theory.

The software in this product was in part provided by Red Hat and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the author be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

Copyrights

With the exception of those items listed below, the contents of this publication are copyright © 2007, Pentek, Inc. All Rights Reserved. Contents of this publication may not be reproduced in any form without written permission.

Part of the software embedded in this product is eCos, embedded Configurable operating system, from Red Hat, Inc. Portions created by Red Hat are Copyright © 1998, 1999, 2000 by Red Hat, Inc. (http://www.redhat.com/). All Rights Reserved. Appendix E, RedBoot User's Guide, is extracted from the eCos Reference Manual, Copyright © 1998, 1999, 2000, 2001, 2002, 2003 by Red Hat, Inc. Nick Garnett (eCosCentric) Jonathan Larmour (eCosCentric) Andrew Lunn (Ascom) Gary Thomas (MLB Associates) Bart Veer (eCosCentric).

Trademarks

Pentek, ReadyFlow, and VIM are trademarks or registered trademarks of Pentek, Inc.

DuPont is a trademark of E.I. du Pont de Nemours and Company. Red Hat, eCos, and RedBoot are trademarks of Red Hat, Inc. Intel is a registered trademark of Intel Corporation. Marvell and Discovery are registered trademarks or trademarks of Marvell. Motorola and AtliVec are registered trademarks or trademarks of Motorola, Inc. PLX Technology and the PLX logo are registered trademarks of PLX Technology, Inc. PowerPC is a registered trademark of International Business Machines Corporation. RACE++ and RACEway++ are registered trademarks or trademarks of Mercury Computer Systems, Inc. Tundra and Universe II are registered trademarks or trademarks of Tundra Semiconductor Corporation. Tornado and VxWorks are registered trademarks of Wind River Systems, Inc. Xilinx and Virtex are registered trademarks of Xilinx, Inc.

Printed in the United States of America.

Table of Contents

Page

Chapter 1: Introduction

1.1 General Description....21
1.2 Processor....21
Figure 1–1: Model 4205 Block Diagram....22
1.3 Controller/Bus Interface....23
1.3.1 SDRAM....23
1.3.2 FLASH Memory....23
1.3.3 RS–232/Ethernet Interface....23
1.3.4 TWSI Bus....23
1.3.5 PCI Buses....23
1.4 VIM Interfaces....24
1.5 VME64 Interface....24
1.6 PMC Interface....24
1.7 Optional RACEway Interfaces (Option 005 or 010)....24
1.8 Optional Fibre Channel Interface (Option 006)....24
1.9 Optional Gigabit Ethernet Interface (Option 007)....24
1.10 Optional PMC Extender Board (Option 150/151/152)....25
Figure 1–2: Option 150/151/152 Extender Block Diagram....25
1.11 Theory of Operation....26
1.12 Board Support Software....27
1.12.1 VxWorks....27
1.12.2 eCos....27
1.13 Supporting Documentation....28
1.14 Specifications....29

Table of Contents

Page

Chapter 2: Installation and Connections

2.1 Inspection 33
2.2 Jumper Settings 33
Figure 2–1: Model 4205 PCB, Component Side 34
2.2.1 VMEbus A16/A24 Slave Select Jumper 35
Table 2–1: VMEbus A16/A24 Slave Select Jumper – JB1 35
2.2.2 VMEbus System Reset Monitor Jumper 35
Table 2–2: VMEbus System Reset Monitor Jumper – JB3 35
2.2.3 VMEbus A16/A24 Slave Base Address Jumpers 36
Table 2–3: VMEbus A16/A24 Slave Base Address Jumpers – JB2 36
2.2.4 FLASH Memory Write Enable Jumper 37
Table 2–4: FLASH Write Enable Jumper – JB4 37
2.2.5 GT–64260 Serial EEPROM Write Enable Jumper 37
Table 2–5: GT–64260 EEPROM Write Enable Jumper – JB6 37
2.2.6 Power–Up Mode Jumpers 38
Table 2–6: Board Power–Up Mode Jumpers – JB7 38
2.2.7 Secondary PCI Bus Speed Jumper 38
Table 2–7: Secondary PCI Bus Speed Jumper – JB8 38
2.2.8 FLASH Boot Parameters Restore Jumper 39
Table 2–8: FLASH Boot Parameters Restore Jumper – JB9 39
2.2.9 General Purpose Jumper 39
Table 2–9: General Purpose Jumper – JB10 39
2.2.10 FPGA Configuration Download Jumper 40
Table 2–10: FPGA Configuration Download Jumper – JB12 40
2.2.11 FPGA General Purpose Jumpers 40
Table 2–11: FPGA General Purpose Jumpers – JB13, JB14 40
2.2.12 Factory Default Jumpers 41
Table 2–12: Factory Default Jumpers 41
2.3 Signal Connectors 42
2.3.1 VIM Connectors 42
2.3.2 PMC Connectors 42
2.3.3 MPC745X COP/JTAG Connector 43
Table 2–13: MPC745X COP/JTAG Connector – J1 43
2.3.4 JTAG In–Circuit Program Chain Connector 43
Table 2–14: JTAG ISP Connector – J6 43
Figure 2–2: Model 4205 PCB, Solder Side 44

Table of Contents

Page

Chapter 2: Installation and Connections *(continued)*

2.4 Installation 45
2.4.1 Installing VIM Modules 45
Figure 2–3: Model 4205 PCB Heat Sink – Style "B" 45
2.4.2 Installing a PMC Module 47
2.4.3 Installing the Model 4205 in the VME Card Cage 47
2.4.4 Removing the Model 4205 from the VME Card Cage 47
2.5 Front Panel 48
Figure 2–4: Model 4205 Front Panel 48
2.5.1 Power LED 48
2.5.2 Temperature LED 48
2.5.3 Reset Button 48
2.5.4 Processor LEDs 49
2.5.5 I/O LEDs 49
2.5.5.1 Standard Models 49
Figure 2–5: Standard Model 4205 Front Panel LEDs 49
2.5.5.2 Option 006 50
Figure 2–6: Model 4205 Option 006 Front Panel LEDs 50
2.5.5.3 Option 007 51
Figure 2–7: Model 4205 Option 007 Front Panel LEDs 51
2.5.6 RS–232/Ethernet Connector 52
Table 2–15: RS–232/Ethernet Connector 52
Table 2–16: RS–232/Ethernet Cable Pin Connections 52
Figure 2–8: RS–232/Ethernet Cable 53
2.5.7 Fibre Channel Connector (Option 006) 53
Table 2–17: Fibre Channel Connector 53
2.5.8 Gigabit Ethernet Connector (Option 007) 54
Table 2–18: Gigabit Ethernet Connector 54
2.5.9 Front Panel Test Points 54
Figure 2–9: Front Panel Test Points 54
2.5.10 Option 150/151/152 PMC Front Panel 55
Figure 2–10: Option 15x Front Panel 55
2.6 Power On Reset 56
Figure 2–11: Model 4205 Resets 56

Table of Contents

Page

Chapter 3: Processor Memory Maps and Resources

3.1 Overview 57
3.2 MPC745X Processor 57
3.2.1 MPC745X Processor Memory Map 58
Table 3–1: MPC745X Processor Memory Map 58
3.2.2 MPC745X PCI Bus I/O Map 59
Table 3–2: MPC745X PCI Bus I/O Address Map 59
3.2.3 GT–64260 TWSI Bus Address Map 59
Table 3–3: GT–64260 TWSI Bus Address Map 59
3.3 SDRAM 60
Table 3–4: SDRAM Memory Map 60
3.4 FLASH Memory 60
3.5 MPC745X Resources Operation 61
3.5.1 Universe II Registers 61
3.5.2 VMEbus Master Access 61
Table 3–5: VMEbus Master Addresses 61
3.5.3 PLX 9656 Registers 62
Table 3–6: PLX 9656 Registers Addresses 62
3.5.4 Serial Mezzanine Module Access 62
3.5.4.1 Fibre Channel Access (Option 006) 62
3.5.4.2 Gigabit Ethernet Access (Option 007) 62
3.5.5 PMC Module Access 63
3.5.6 RACEway Access (Option 005 or 010) 63
3.5.7 Option 150/151/152 PMC Module Access 64
Figure 3–1: Option 15x Front Panel 64
Table 3–7: Option 150/151/152 PMC Addresses 64
3.5.8 VIM Module Access 65
Table 3–8: VIM Module Access Addresses 65
3.5.9 VIM Interface Configuration Registers 66
Table 3–9: VIM Interface Configuration Register Addresses 66
3.5.10 VIM BIFO/FPGA Access 67
3.5.11 Voltage/Temperature Monitors 68
3.5.12 GT–64260 EEPROM 68
3.5.13 Board ID EEPROM 69
3.5.14 Real–Time Clock 69
3.5.15 Programmable Clock Generator 69

Table of Contents

Page

Chapter 3: Processor Memory Maps and Resources *(continued)*

3.6 GT−64260 Controller 70
3.6.1 GT−64260 Internal Registers 70
3.6.2 Messaging Unit (MU) 71
3.6.3 IDMA Controller 71
3.6.4 General Purpose Port (GPP) 71
3.6.5 Communication Unit 71
3.6.6 Interrupt Controller 71
3.6.7 TWSI Interface 71
3.6.8 Multi−Purpose Pins (MPP) 72
Table 3−10: GT−64260 MPP Pin Default Functions 72
3.6.8.1 DMA Requests 73
3.6.8.2 VIM FPGA PLX 9656 Interrupts 73
3.6.8.3 VMEbus Interrupt 73
3.6.8.4 Serial Mezzanine Module Interrupt 73
3.6.8.5 PXB Interrupt (Option 005 or 010) 74
3.6.8.6 PMC Module Interrupt 74
3.6.8.7 Front Panel Test Point Interrupt 74
3.6.8.8 GT−64260 Initialization Active 74
3.6.8.9 VIM Incoming FIFO Not Almost Empty Interrupts 74
3.6.8.10 VIM Outgoing FIFO Almost Empty Interrupts 74
3.6.8.11 VIM Incoming FIFO Almost Full Interrupts 74
3.6.8.12 VIM Incoming FIFO Not Empty Interrupt 75
3.6.8.13 VIM Outgoing FIFO Empty Interrupt 75
3.6.8.14 VIM Incoming/Outgoing FIFO Full Interrupt 75
3.6.8.15 VIM Module DBINT/MBIN/XMIT/RCV Interrupt 75

Table of Contents

Page

Chapter 3: Processor Memory Maps and Resources *(continued)*

3.7 Board Control/Status Registers 76
Table 3–11: Board Control/Status Register Addresses 76
3.7.1 LED Control Register 77
Table 3–12: LED Control Register 77
3.7.1.1 TRIGGER 77
3.7.1.2 LED3/2/1/0 77
3.7.2 Reset Register 78
Table 3–13: Reset Register 78
3.7.2.1 PCI_1/PCI_0 78
3.7.2.2 VIM D/C/B/A 78
3.7.2.3 BOARD 78
Figure 3–2: Model 4205 Resets 79
3.7.3 Fault Register 80
Table 3–14: Fault Register 80
3.7.4 BIST Register 80
Table 3–15: BIST Register 80
3.7.5 Module Status Register 81
Table 3–16: Module Status Register 81
3.7.5.1 PMC PRESENT 81
3.7.5.2 VIM D/C/B/A PRESENT 81
3.7.6 Jumper Status Register 82
Table 3–17: Jumper Status Register 82
3.7.6.1 GENERAL PURPOSE 82
3.7.6.2 FLASH RESTORE 82
3.7.6.3 CPU SPEED 83
3.7.6.4 SEC PCI BUS SPEED 83
3.7.6.5 FLASH WRITE ENABLE 83
3.7.6.6 SELF–TEST MODE 83
3.7.7 FPGA Jumper Status Register 84
Table 3–18: FPGA Jumper Status Register 84
3.7.8 Watchdog Timer Control Register 85
Table 3–19: Watchdog Timer Control Register 85
3.7.8.1 RESET ENABLE 85
3.7.8.2 TIMER STATUS 86
3.7.8.3 COUNTER 86

Table of Contents

Page

Chapter 3: Processor Memory Maps and Resources *(continued)*

3.8 Board Interrupt Registers 87
Table 3–20: Board Interrupt Register Addresses 87
3.8.1 VIM BIFO Full Interrupt Mask Register 88
Table 3–21: VIM BIFO Full Interrupt Mask Register 88
3.8.2 VIM BIFO Empty Interrupt Mask Register 89
Table 3–22: VIM BIFO Empty Interrupt Mask Register 89
3.8.3 PMC Interrupt Mask Register 90
Table 3–23: PMC Interrupt Mask Register 90
3.8.4 VIM Module Interrupt Mask Register 91
Table 3–24: VIM Module Interrupt Mask Register 91
3.8.5 VIM BIFO Full Interrupt Flag Register 92
Table 3–25: VIM BIFO Full Interrupt Flag Register 92
3.8.6 VIM BIFO Empty Interrupt Flag Register 93
Table 3–26: VIM BIFO Empty Interrupt Flag Register 93
3.8.7 PMC Interrupt Flag Register 94
Table 3–27: PMC Interrupt Flag Register 94
3.8.8 VIM Module Interrupt Flag Register 95
Table 3–28: VIM Module Interrupt Flag Register 95
3.8.9 VIM Serial Port Interrupt Flag Register 96
Table 3–29: VIM Serial Port Interrupt Flag Register 96
3.8.10 VIM BIFO Full Interrupt Status Register 97
Table 3–30: VIM BIFO Full Interrupt Status Register 97
3.8.11 VIM BIFO Empty Interrupt Status Register 98
Table 3–31: VIM BIFO Empty Interrupt Status Register 98
3.8.12 PMC Interrupt Status Register 99
Table 3–32: PMC Interrupt Status Register 99
3.8.13 VIM Module Interrupt Status Register 100
Table 3–33: VIM Module Interrupt Status Register 100
3.8.14 VIM Serial Port Interrupt Status Register 101
Table 3–34: VIM Serial Port Interrupt Status Register 101
3.8.15 Temperature/Voltage Fault Interrupt Mask Register 102
Table 3–35: Temperature/Voltage Fault Interrupt Mask Register 102

Table of Contents

Page

Chapter 4: PCI Bus Memory Maps and Resources

4.1 Overview 103
4.2 PCI Configuration Space 103
Table 4–1: PCI Configuration Space Addresses 103
4.3 PCI Bus Memory Map 104
Table 4–2: PCI Bus Memory Address Map 104
4.4 PCI Bus I/O Map 105
Table 4–3: PCI Bus I/O Address Map 105
4.5 PCI Bus Resources Operation 105
4.5.1 SDRAM 105
4.5.2 FLASH Memory 106
4.5.3 GT–64260 Internal Registers 106
4.5.4 GT–64260 Messaging Unit 106
4.5.5 Board Control/Status Registers 107
Table 4–4: Board Control/Status Register PCI Bus Addresses 107
4.5.6 Board Interrupt Registers 107
Table 4–5: Board Interrupt Register PCI Bus Addresses 107
4.5.7 Universe II VME Registers 108
4.5.8 VMEbus Master Access 108
Table 4–6: VMEbus Master PCI Bus Addresses 108
4.5.9 PLX 9656 Registers 109
Table 4–7: PLX 9656 Registers PCI Bus Addresses 109
4.5.10 Serial Mezzanine Module Access 109
4.5.10.1 Fibre Channel Access (Option 006) 109
4.5.10.2 Gigabit Ethernet Access (Option 007) 109
4.5.11 PMC Module Access 110
4.5.12 RACEway Access (Option 005 or 010) 110
4.5.13 Option 150/151/152 PMC Module Access 111
Figure 4–1: Option 15x Front Panel 111
Table 4–8: Option 150/151/152 PCI Bus Addresses 111
4.5.14 VIM Module Access 112
Table 4–9: VIM Module Access PCI Bus Addresses 112
4.5.15 VIM Interface Configuration Registers 113
Table 4–10: VIM Interface Configuration Register PCI Bus Addresses 113
4.5.16 VIM BIFO/FPGA Access 114

Table of Contents

Page

Chapter 5: VME Interface Memory Maps and Resources

5.1 Overview....115
5.2 VMEbus Master Access....115
Table 5–1: VMEbus Master Memory Map....115
5.3 VMEbus Slave Access....116
5.3.1 Using the Monitor to Configure Slave Access....116
5.3.2 Default VMEbus A16/A24 Slave Memory Map....117
Table 5–2: Default VMEbus A16/A24 Slave Memory Map....117
5.3.3 Extended VMEbus A24 Slave Memory Map....118
Table 5–3: Extended VMEbus A24 Slave Memory Map....118
5.3.4 Extended VMEbus A32 Slave Memory Map....119
Table 5–4: Extended VMEbus A32 Slave Memory Map....119
5.4 VMEbus Slave Resources Operation....120
5.4.1 Universe II Registers....120
5.4.2 GT–64260 Internal Registers....120
5.4.3 GT–64260 Messaging Unit....121
5.4.4 SDRAM Access....121
Table 5–5: VMEbus SDRAM Access Addresses....121
5.4.5 Board Control/Status Registers....122
Table 5–6: Board Control/Status Register VMEbus Addresses....122
5.4.6 Board Interrupt Registers....122
Table 5–7: Board Interrupt Register VMEbus Addresses....122
5.4.7 PLX 9656 Registers....123
Table 5–8: PLX 9656 Registers VMEbus Addresses....123
5.4.8 PMC Module Access....123
5.4.9 RACEway PXB++ Registers (Option 005 or 010)....123
5.4.10 VIM Module Access....124
Table 5–9: VIM Module Control/Status VMEbus Addresses....124
5.4.11 VIM Interface Configuration Registers....125
Table 5–10: VIM Interface Configuration Register VMEbus Addresses....125

Table of Contents

Page

Chapter 6: VIM Operation

6.1 Overview 127
6.2 VIM BIFO/FPGA Configurations 127
Figure 6–1: VIM BIFO/FPGA Configuration – VIM to Processor 128
Figure 6–2: VIM BIFO/FPGA Configuration – VIM to VIM 128
6.3 VIM Module Access 129
Table 6–1: VIM Module Access Addresses 129
6.4 VIM Interface Configuration Registers 130
Table 6–2: VIM Interface Configuration Register Addresses 130
6.4.1 VIM FPGA Configuration Control Registers 131
Table 6–3: VIM FPGA Configuration Control Register 131
6.4.1.1 WRITE 131
6.4.1.2 INIT 131
6.4.1.3 DONE 132
6.4.1.4 PRGM 132
6.4.1.5 CONFIG MODE 132
6.4.2 VIM FPGA Configuration Data Registers 133
Table 6–4: VIM FPGA Configuration Data Register 133
6.4.3 VIM Interface BIFO Control Register 134
Table 6–5: VIM Interface BIFO Control Register 134
6.4.3.1 VIM x BIFO RESET 134
6.4.3.2 VIM x BIFO PATH SELECT 134
6.4.4 VIM Interface BIFO Reset Status Register 135
Table 6–6: VIM Interface BIFO Reset Status Register 135
6.4.5 VIM FPGA DCM Reset Register 136
Table 6–7: VIM FPGA DCM Reset Register 136
6.5 VIM BIFO/FPGA Resources 137
6.5.1 VIM to Processor Configuration Memory Maps 137
Table 6–8: VIM BIFO/FPGA PLX 9656 Access Memory Map, VIM–Proc 137
Table 6–9: VIM BIFO/FPGA PCI Bus Access Memory Map, VIM–Proc 138
6.5.2 VIM to VIM Configuration Memory Maps 139
Table 6–10: VIM BIFO/FPGA PLX 9656 Access Memory Map, VIM–VIM 139
Table 6–11: VIM BIFO/FPGA PCI Bus Access Memory Map, VIM–VIM 140

Table of Contents

Page

Chapter 6: VIM Operation *(continued)*

6.6 VIM BIFO Direct Access 141
6.6.1 Initializing VIM BIFOs 141
6.6.2 VIM BIFO Data Access 142
Table 6–12: VIM BIFO Data Addresses 142
6.6.3 VIM BIFO Demand Mode DMA Operation – GT–64260 143
Table 6–13: VIM BIFO GT–64260 IDMA Channels 143
Figure 6–3: VIM BIFO Demand Mode DMA Timing (2 cycles) 144
6.6.4 VIM BIFO Demand Mode DMA Operation – PLX 9656 145
Table 6–14: VIM BIFO DMA Addresses 145
6.6.5 VIM BIFO Mailbox Access 146
Table 6–15: VIM BIFO Mailbox Addresses 146
6.7 VIM Serial Ports 147
Table 6–16: VIM Serial Port FIFO Addresses 147
Table 6–17: VIM Serial Port FIFO Flags 147
6.8 VIM BIFO FPGA Access (VIM to Processor) 148
6.8.1 VIM FPGA DPRAM Swinging Buffer Operation 148
Figure 6–4: DPRAM Swinging Buffers 148
Table 6–18: VIM FPGA Swinging Buffer Addresses 148
6.8.2 VIM FPGA DMA Link List RAM Operation 150
Figure 6–5: Example DMA Chain 150
Table 6–19: VIM FPGA DMA RAM Addresses 150
6.9 VIM FPGA Control/Status Registers (VIM to Processor) 151
Table 6–20: VIM FPGA Control/Status Registers Base Address 151
Table 6–21: VIM FPGA Control/Status Register Addresses, VIM–Proc 152
6.9.1 Swinging Buffer Control Registers 153
Table 6–22: Swinging Buffer Control Register 153
6.9.1.1 BUFFER WR EN 153
6.9.1.2 ADDR MODEn 153
6.9.2 Swinging Buffer Length Registers 154
Table 6–23: Swinging Buffer Length Register 154
6.9.3 Swinging Buffer Full Mask Registers 155
Table 6–24: Swinging Buffer Full Mask Register 155
6.9.3.1 DMA REQUEST 155
6.9.3.2 BUFFER FULL INTERRUPT 155
6.9.4 Swinging Buffer Full Flag Registers 156
Table 6–25: Swinging Buffer Full Flag Register 156
6.9.5 Swinging Buffer Count Registers 157
Table 6–26: Swinging Buffer Count Register 157

Table of Contents

Page

Chapter 6: VIM Operation *(continued)*

6.9 VIM FPGA Control/Status Registers (VIM to Processor) (continued)
- 6.9.6 FPGA Interrupt Mask Registers 158
  - **Table 6–27: FPGA Interrupt Mask Register 158**
  - **Table 6–28: FPGA Interrupt Registers Bits, VIM to Proc 159**
- 6.9.7 FPGA Interrupt Flag Registers 160
  - **Table 6–29: FPGA Interrupt Flag Register 160**
- 6.9.8 FPGA Interrupt Status Registers 162
  - **Table 6–30: FPGA Interrupt Status Register 162**
- 6.9.9 FPGA Serial Port Control Registers 163
  - **Table 6–31: FPGA Serial Port Control Register 163**
  - 6.9.9.1 RCV ENABLE B(D) 163
  - 6.9.9.2 XMIT ENABLE B(D) 163
  - 6.9.9.3 WORD SIZE B(D) 164
  - 6.9.9.4 XMIT CLK SRC B(D) 164
  - 6.9.9.5 RCV RESET B(D) 164
  - 6.9.9.6 XMIT RESET B(D) 164
  - 6.9.9.7 RCV ENABLE A(C) 164
  - 6.9.9.8 XMIT ENABLE A(C) 164
  - 6.9.9.9 WORD SIZE A(C) 165
  - 6.9.9.10 XMIT CLK SRC A(C) 165
  - 6.9.9.11 RCV RESET A(C) 165
  - 6.9.9.12 XMIT RESET A(C) 165

6.10 FPGA Buffer Format Modes 166
- 6.10.1 Normal Buffer Mode (code 0000) 166
- 6.10.2 Split Buffer Mode x2 (code 0001) 166
- 6.10.3 Split Buffer Mode x4 (code 0010) 167
- 6.10.4 Split Buffer Mode x8 (code 0011) 167
- 6.10.5 Split Buffer Mode x16 (code 0100) 168
- 6.10.6 Split Buffer Mode x32 (code 0101) 168
- 6.10.7 Data Unpacking Mode, Split Buffer x2 (code 0110) 169
- 6.10.8 Data Unpacking Mode, Split Buffer x4 (code 0111) 169
- 6.10.9 Data Unpacking Mode, Split Buffer x8 (code 1000) 170
- 6.10.10 Data Unpacking Mode, Split Buffer x16 (code 1001) 170

Table of Contents

Page

Chapter 6: VIM Operation *(continued)*

6.11 VIM FPGA Control/Status Registers (VIM to VIM) 171
Table 6–32: VIM FPGA Control/Status Registers Base Address....171
Table 6–33: VIM FPGA Control/Status Register Addresses, VIM–VIM 172
6.11.1 Data Transfer Control Registers 173
Table 6–34: Data Transfer Control Register....173
6.11.1.1 XFER FLOW EN 173
6.11.1.2 SUSP RESPONSE 173
6.11.1.3 MODE 174
6.11.1.4 XFER DIR 174
6.11.2 Data Transfer Status Registers 175
Table 6–35: Data Transfer Status Register....175
6.11.2.1 DATA VALID 175
6.11.2.2 SUSP ACK IN 176
6.11.2.3 SUSP ACK OUT 176
6.11.2.4 SUSP IN 176
6.11.2.5 SUSP OUT 176
6.11.2.6 DIR OUT 177
6.11.2.7 BIFO DIR 177
6.11.2.8 DIR MASTER 177
6.11.2.9 PRESENT STATE 177
6.11.3 FPGA Interrupt Mask Registers 178
Table 6–36: FPGA Interrupt Mask Register 178
Table 6–37: FPGA Interrupt Registers Bits, VIM to VIM....179
6.11.4 FPGA Interrupt Flag Registers 180
Table 6–38: FPGA Interrupt Flag Register....180
6.11.5 FPGA Interrupt Status Registers 181
Table 6–39: FPGA Interrupt Status Register 181

Table of Contents

Page

Chapter 6: VIM Operation *(continued)*

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)
6.11.6 FPGA Serial Port Control Registers 182
Table 6–40: FPGA Serial Port Control Register 182
6.11.6.1 RCV ENABLE B(D) 182
6.11.6.2 XMIT ENABLE B(D) 182
6.11.6.3 WORD SIZE B(D) 183
6.11.6.4 XMIT CLK SRC B(D) 183
6.11.6.5 RCV RESET B(D) 183
6.11.6.6 XMIT RESET B(D) 183
6.11.6.7 RCV ENABLE A(C) 183
6.11.6.8 XMIT ENABLE A(C) 183
6.11.6.9 WORD SIZE A(C) 184
6.11.6.10 XMIT CLK SRC A(C) 184
6.11.6.11 RCV RESET A(C) 184
6.11.6.12 XMIT RESET A(C) 184
6.12 VIM to VIM Data Transfer Operation 185
6.12.1 Transmit 185
6.12.2 Receive 185
Figure 6–6: VIM to VIM Data Transfer Timing 186

Table of Contents

Page

Chapter 7: Interrupt Operation

7.1 Overview 187
7.2 Interrupt Operation 187
7.3 Interrupt Routing 188
Figure 7–1: Model 4205 Interrupt Connections 188
Figure 7–2: VIM BIFO/FPGA Interrupt Routing 189
Table 7–1: Interrupt Routing 190
7.4 MPC745X Processor Interrupts 191
7.4.1 MPC745X Incoming Interrupts 191
7.4.2 MPC745X Outgoing Interrupts 191
7.5 GT–64260 Doorbell Interrupts 192
7.6 VIM BIFO Interrupts 193
7.7 VIM Module Interrupts 194
7.7.1 VIM Module DB/MB Interrupts 194
7.7.2 VIM Serial Port Interrupts 194
7.8 VMEbus Interrupts 195
7.8.1 VMEbus Interrupt Handling 195
7.8.2 VMEbus Interrupt Generation 196
7.9 PXL 9656 Interrupts 197
Figure 7–3: PLX PCI 9656 Interrupt Connections 197
7.10 Voltage/Temperature Monitor Interrupts 198
7.11 PMC Module Interrupts 198
7.12 RACEway PXB++ Interrupts (Option 005 or 010) 199
7.13 Serial Mezzanine Module Interrupts 199

Table of Contents

Page

Chapter 8: FLASH Memory and Boot Firmware

8.1 Overview 201
8.2 FLASH Memory 202
8.3 Firmware Boot Operation 202
Table 8–1: FLASH Memory Map 202
8.3.1 Use of SDRAM During Boot 203
Table 8–2: SDRAM Boot Map 203
Table 8–3: Current Temperature/Voltage SDRAM Boot Map 204
Table 8–4: Peak Voltage/Temperature SDRAM Boot Map 205
Table 8–5: VIM ID EEPROM Example Code (Model 6230 shown) 206
8.3.2 MPC745X Boot Sequence 207
Table 8–6: LED Boot Codes 207
Figure 8–1: MPC745X Firmware Boot Sequence 209
8.4 MPC745X Firmware Functions 210
Table 8–7: MPC745X Firmware Function Summary 210
Table 8–8: SDRAM Command Table 210
8.5 Firmware Test Failure Codes 211
Figure 8–2: Front Panel LEDs 211
Table 8–9: Boot Firmware Test Failure Codes 211

Table of Contents

Page

Chapter 9: The Monitor Program

9.1 Overview 213
9.2 Connecting to the Serial Port 213
9.3 Connecting to Ethernet Interface 214
9.4 RedBoot Monitor Commands 214
9.4.1 appnetconfig 214
9.4.2 cconfig 218
9.4.3 displayTempVoltage 220
9.4.4 fpgaconfig 221
9.4.5 fpgaload 222
9.4.6 help 223
9.4.7 pdump 225
9.4.8 pedit 226
9.4.9 time 227
9.4.10 vmemasterconfig 228
9.4.11 vmeslaveconfig 231

Table of Contents

Page

Appendix A: Model 4205 ID EEPROM

A.1 Introduction A–1
A.2 ID EEPROM Format and Contents A–1
Table A–1: EEPROM Example Code A–2

Appendix B: Tundra® Universe II™ Registers

Table B–1: Universe II Control and Status Registers B–1

Appendix C: PCI Configuration Space Registers

C.1 Introduction C–1
C.2 Primary and Secondary PCI Buses C–1
Table C–1: PCI Configuration Space Header (Type 00h) C–1
C.3 Option 150/151/152 PCI Buses C–2
Figure C–1: Option 150/151/152 Extender Block Diagram C–2

Appendix D: Option 005/010 RACEway Operation

D.1 Overview D–1
D.2 General Operation D–1
D.3 RACEway Transfer Rates D–2

Appendix E: RedBoot™ User's Guide

E.1 Introduction E–1

Chapter 1: Introduction

1.1 General Description

The Model 4205 processor board is a VIM/PMC carrier VMEbus baseboard with a single PowerPC® processor. It provides a low−cost solution for introducing PCI Mezzanine Card (PMC) and Velocity Interface Mezzanine (VIM®) I/O into VME systems, and with optional front panel high−speed serial I/O can also be used for recording applications. The Model 4205 contains both VIM and PMC module sites; features RS−232/Ethernet, VME64, (optional) RACEway®, and (optional) Fibre Channel or (optional) Gigabit Ethernet serial interfaces; and can act as VMEbus Slot 1 Controller.

A block diagram of the Model 4205 is presented in Figure 1−1 on the next page.

1.2 Processor

The Model 4205 includes one 1−GHz MPC7457 PowerPC® processor. The processor has access to each VIM and PMC site and to global resources on the board through a Marvell® Discovery™ GT−64260 system controller. The global resources that can be accessed by the processor include the RS−232/Ethernet, VMEbus, optional Fibre Channel or Gigabit Ethernet, and optional RACEway interfaces.

The Motorola MPC7457 processor is an implementation of the PowerPC microprocessor family of reduced instruction set computer (RISC) microprocessors. This processor utilizes the AltiVec™ vector engine to perform parallel processing of multiple data elements (SIMD) with 128−bit operations. The MPC7457 is a high−performance, superscalar microprocessor that can dispatch and complete two instructions simultaneously. The AltiVec processor executes both fixed and floating−point instructions, augmenting the separate integer and floating point units also on the chip.

NOTE: The standard Model 4205 now includes Option 057, the 1−GHz MPC7457 PowerPC processor. Earlier models provided a 600−MHz MPC7455 PowerPC processor, which is currently obsolete. This manual uses the term 'MPC745X' to refer to either processor, depending on which you have received.

The MPC7457 processor includes a 512−kByte on−chip L2 cache memory, and a 4−MByte L3 cache memory.

1.2 **Processor** (continued)

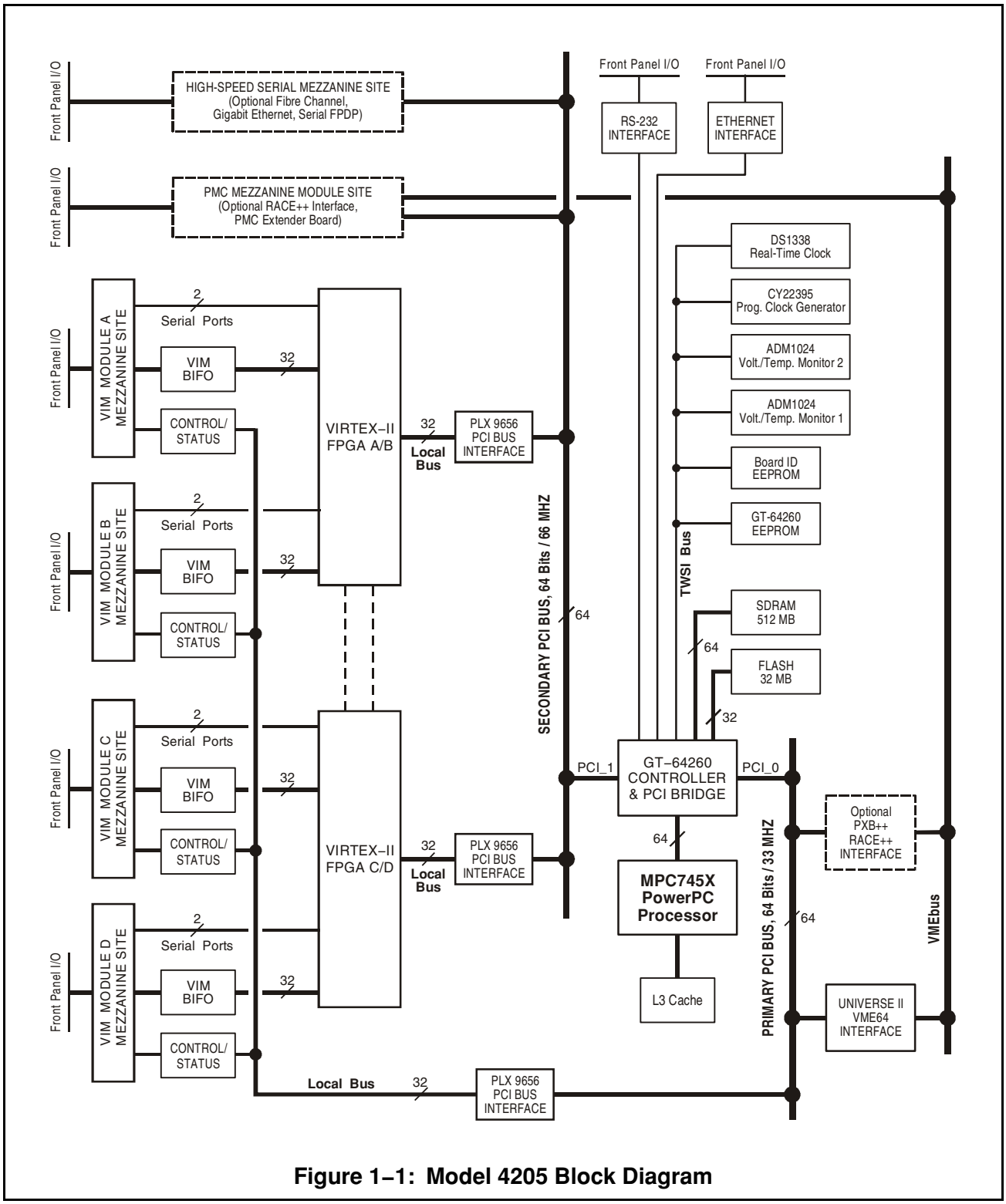


Figure 1–1: Model 4205 Block Diagram

1.3 Controller/Bus Interface

The Marvell® Discovery™ GT−64260 is an integrated system controller for the MPC745X PowerPC processor. This controller acts as the interface and device controller for access to the following Model 4205 resources.

1.3.1 SDRAM

A 512−MByte SDRAM (64M x 64) provides a convenient shared resource for staging data. (With Option 340, the size of SDRAM is 1 GByte.) The SRDAM is used for boot operation at power−up or reset of the Model 4205.

1.3.2 FLASH Memory

A 32−MByte FLASH memory (32M x 8) provides nonvolatile storage of initialization, self−test, and boot code for the MPC745X. When the Model 4205 is reset or power is applied, the MPC745X copies the boot code from FLASH to SDRAM (above) and then executes this boot code from SDRAM.

1.3.3 RS−232/Ethernet Interface

The GT−64260 controller supports input and output of serial data using two RS−232 and one 10/100 BaseT Ethernet LAN ports. This interface is provided on a 15−pin Micro−D connector on the Model 4205 front panel.

1.3.4 TWSI Bus

The GT−64260 two−wire serial interface (TWSI) provides access to the board's voltage and temperature monitors (two Analog Devices ADM1024s), the programmable clock generator (Cypress CY22395), a real−time clock (Dallas Semiconductor DS1338), the board ID EEPROM, and the GT−64260 configuration EEPROM.

1.3.5 PCI Buses

The Model 4205 has two PCI buses: the Primary PCI Bus and the Secondary PCI Bus. The GT−64260 controller acts as the PCI bridge between the buses, and provides the interface from the MPC745X to all PCI resources.

- The Primary PCI Bus (GT−64260 PCI_0) provides the MPC745X processor access to each VIM module's control and status registers, the VME64 VMEbus interface, and the optional primary RACEway interface.

- The Secondary PCI Bus (GT−64260 PCI_1) provides the MPC745X access to the VIM data interfaces, the PMC site, the optional second RACEway interface, and the optional high−speed serial mezzanine site.

1.4 VIM Interfaces

The Model 4205 provides four Pentek Velocity Interface Mezzanine (VIM) interface connections. Each VIM data interface includes a 2 x 1k x 32 high−speed synchronous bi−directional FIFO (BIFO), which buffers 32−bit parallel data transfers between the VIM module and the PCI interface at rates up to 200 MBytes/second. Serial ports provide an additional data streaming connection between each VIM module and the Secondary PCI Bus. Each VIM interface also provides a high−speed streaming data interface through one of two on−board Xilinx® Virtex®−II FPGAs. A portion of each FPGA can be configured as two 2 x 4k x 32 dual−ported random access memory (DPRAM) swinging buffers, one from each VIM BIFO interface.

1.5 VME64 Interface

The Model 4205 Master/Slave VME64 interface (Tundra® Universe II™) allows the MPC745X processor to act as a VMEbus Master for data transfers to or from any VME−bus device. It also allows VMEbus Slave access to SDRAM, VIM, PMC, and RACEway interfaces from an external VMEbus Master.

1.6 PMC Interface

The 64−bit, 66−MHz PCI Mezzanine Card (PMC) site accepts industry−standard PMC modules for a wide variety of interface and communication functions (such as the Pentek Model 7131 16−channel Digital Receiver PMC Module). The Model 4205 Option 010 RACE++ interface, below, uses the PMC site.

1.7 Optional RACEway Interfaces (Option 005 or 010)

Model 4205 Option 005 provides a RACE++® interface to the Primary PCI Bus that allows data transfers up to 267 MBytes/second with other RACE++ VME boards. Option 010 includes Option 005, adds a second RACE++ interface to the Secondary PCI Bus using the PMC site, and allows data transfers up to 533 MBytes/second.

1.8 Optional Fibre Channel Interface (Option 006)

The Option 006 Fibre Channel Copper Interface provides up to 2 Gigabits/second data transfers with the Secondary PCI Bus using a front panel serial I/O connector.

1.9 Optional Gigabit Ethernet Interface (Option 007)

The Option 007 Gigabit Ethernet Interface provides a 10/100/1000BaseT Ethernet I/O interface to the Secondary PCI Bus through a front panel serial I/O connector.

NOTE: Option 007 and Option 006 cannot be installed together since they both occupy the Serial Mezzanine site.

1.10 Optional PMC Extender Board (Option 150/151/152)

The Option 150, Option 151, or Option 152 PMC Extender Board for the Model 4205 provides additional PCI Mezzanine Card (PMC) sites. It extends the baseboard PMC connector into a second VMEbus slot, where up to three PMC modules can be mounted. This additional PMC support is provided without interfering with base–board VIM mezzanine sites, so that up three PMC modules and two VIM–2 modules can be mounted on a single baseboard simultaneously.

Option 150/151/152 adds two more PCI buses to the Model 4205. The following is a block diagram of the Option 150/151/152 PMC Extender, showing these additional Ter–tiary and Quaternary PCI buses.

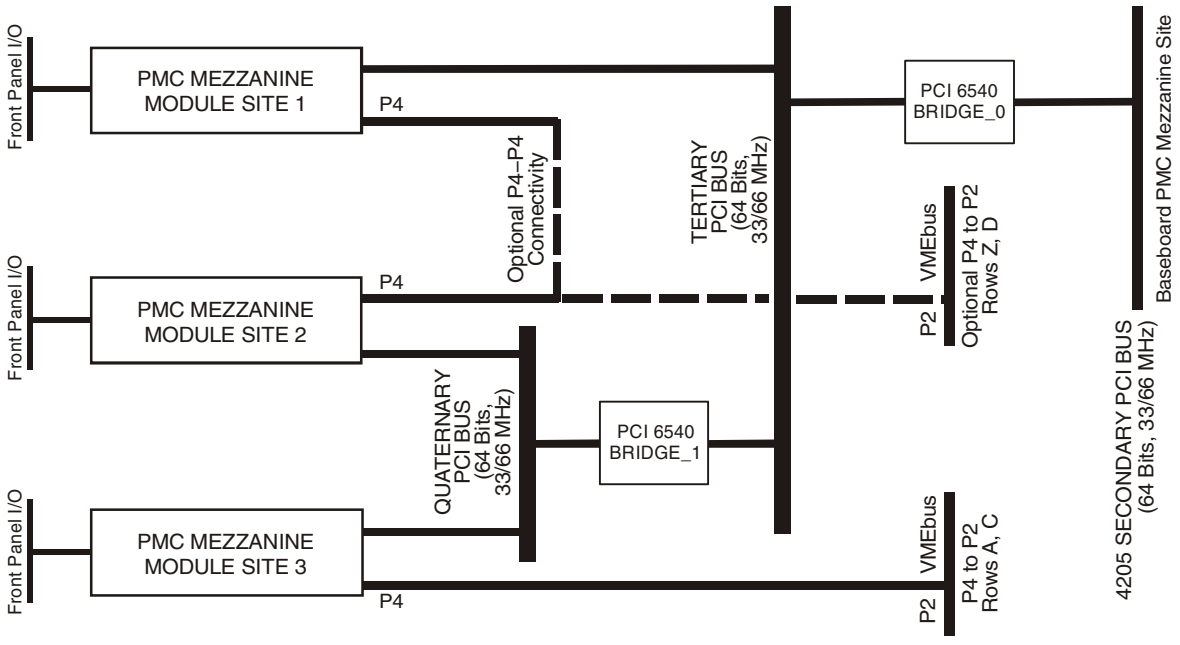


Figure 1–2: Option 150/151/152 Extender Block Diagram

NOTE: The Option 150/151/152 PMC Extender Board is documented separately from the Model 4205, as the Model 9220 PMC Extender Board. Refer to the Model 9220 Operating Manual, part # 800.92200 (which is supplied with your Model 4205 Option 150/151/152), for detailed description of the Extender Board setup and operation.

1.11 Theory of Operation

The Model 4205 VMEbus processor board targets embedded applications requiring high performance interfaces. Featuring one Motorola MPC745X PowerPC processor, the Model 4205 supports numerous high−speed interfaces including VIM, PMC, VME64, RACE++ (Option 005 or 010), and either Fibre Channel (Option 006) or Gigabit Ethernet (Option 007).

The 1−GHz Motorola MPC7457 utilizes the AltiVec vector engine to perform parallel processing of multiple data elements (SIMD) with 128−bit operations. The AltiVec processor executes both fixed and floating−point instructions, augmenting the separate integer and floating point units also on the chip. Four types of memory resources are provided on the board for the MPC7457: a fast 512−kByte level 2 cache, a 4−MByte level 3 cache, a large 512−MByte SDRAM (1 GByte with Option 340), and a nonvolatile 32−MByte FLASH boot memory.

The Model 4205 is equipped with four Velocity Interface Mezzanine (VIM) connectors. The VIM architecture allows simultaneous I/O data transfer from all four VIM interfaces with no contention for shared resources. Each VIM connection is through a bi−directional FIFO (BIFO) that buffers 32−bit parallel data transfers between the VIM module and the Model 4205 PCI interface. Each VIM BIFO interface is also routed to one of two Xilinx Virtex−II FPGAs, which provide each VIM module a private high−speed synchronous bi−directional streaming data interface. Serial ports provide an additional data streaming interface for each VIM module. Control and status registers on the VIM modules can be read and written to as memory−mapped resources.

In addition to accommodating Pentek's broad range of VIM I/O peripherals, the Model 4205 incorporates a 64−bit, 66−MHz PCI Mezzanine Card (PMC) site, which accepts industry−standard modules for a wide variety of interface and communication functions.

The Model 4205 includes two global 64−bit PCI buses. The Primary PCI bus provides access to the VMEbus and the optional primary RACE++ interface. A PCI bridge joins the Primary PCI Bus to the Secondary PCI Bus, which features interfaces for the VIM data, the PMC site, the optional second RACE++ interface, and the optional Fibre Channel or Gigabit Ethernet interfaces.

The MPC745X can master the VMEbus using the industry standard Universe II VME64 interface chip. As a VMEbus slave, the Model 4205 presents all global board resources for convenient read/write access from any external VMEbus master.

The Model 4205 features either an optional copper Fibre Channel interface for high−speed data transfer to and from fibre channel storage devices, or an optional 1000BaseT Gigabit Ethernet I/O. When installed, this interface is located on the front panel along with 10/100 BaseT Ethernet and RS−232 interfaces.

Delivering up to 267 or 533 MBytes/second between VME boards, the optional RACE−way interfaces allow the processor to send and receive RACE++ packets. Both RACE−way options maintain the board's one VMEbus slot configuration while accommodating VIM and PMC modules.

1.12 Board Support Software

Pentek's board support software allows high−level programming of board resources to speed development tasks. The Model 4205 uses the RedBoot™ kernel in all configurations for processor booting and initial self−test. RedBoot allows the processor to boot VxWorks® or eCos™ for an operating system (OS) development platform.

1.12.1 VxWorks

The VxWorks scenario requires Wind River licensing. The user needs an OEM agreement from Wind River, and requires a node license and Tornado® II code generation tools, also from Wind River, to develop an application.

Pentek provides VxWorks BSPs and drivers for the Model 4205, available as Pentek Model 4996 VxWorks Board Support Package, Option 205. This package provides support for MPC745X, GT−64260, VIM interfaces, VMEbus interface, and the optional RACEway, Fibre Channel, and Gigabit Ethernet interfaces. Additional software drivers are provided in this package for each Pentek VIM module used on the Model 4205. This VxWorks Board Support Package also includes a Fibre Channel driver to enable the Model 4205 to interface to SCSI−based storage devices.

ReadyFlow® examples, compiled to run on the Model 4205 under VxWorks, are provided for the Model 4205 baseboard, available as Pentek Model 4999 ReadyFlow Board Support Libraries, Option 205. Model 4999 ReadyFlow Board Support Libraries are also available for each Pentek VIM and PMC module that can be used on the Model 4205.

In order to achieve plug and play operation, users who choose to use the PMC site must operate in this VxWorks development scenario and must use VxWorks drivers provided by the PMC manufacturer. The Model 4205 provides interrupts in compliance with the PMC specification to support plug and play operation.

1.12.2 eCos

eCos is a license−free OS that provides many of the functions of an OS development environment. The user can operate the Model 4205 with eCos in a development scenario similar to the VxWorks environment (Section 1.12.1, above). Code development tools required by the user are the GNU compiler and debug tools (available from Pentek).

The Pentek Model 4999 ReadyFlow Board Support Libraries, Option 205, provide baseboard and VIM support within the eCos OS environment. The ReadyFlow libraries include a Fibre Channel driver to enable the Model 4205 to interface to SCSI−based storage devices.

1.13 Supporting Documentation

In addition to the operating instructions provided in this manual, copies of relevant user documents for the programmable devices on the Model 4205 are available by separate request from the respective companies. Refer to each company's Website for the latest versions.

- Motorola Inc. — http://www.motorola.com

  Motorola MPC7450 RISC Microprocessor Family User's Manual, Reference Manual, MPC7450UM/D

- Marvell — http://www.marvell.com

  Marvell GT–64260A/B Datasheet, MV–S100414–00, Rev. B

- Tundra Semiconductor Corporation — http://www.tundra.com

  Universe II User Manual, 80A3010_MA001_03

- PLX Technology, Inc. — http://www.plxtech.com

  PCI 9565 Data Book, 9565–SIL–DB–P1–0.90

  PCI 9540 Data Book, 6540CB–SIL–DB–P1–2.01 (Option 150/151/152)

- Analog Devices, Inc. — http://www.analog.com

  Analog Devices ADM1024 Datasheet, C00059–0–2/01, Rev. A

- Cypress Semiconductor Corporation — http://www.cypress.com

  Cypress CY22395 Datasheet, 38–07186

- Dallas Semiconductor Corporation — http://www.maxim–ic.com

  Dallas DS1338 Datasheet, 010803

NOTE: All references in this Operating Manual to the user documents listed above refer to the document versions available at the time of this manual printing. If you use different versions of these documents, these references may not be the same.

1.14 Specifications

Processor:	Motorola	Clock Rate	L2 Cache	L3 Cache
Option 057:	MPC7457	1 GHz	512 kBytes	4 MBytes
(Obsolete)	MPC7455	600 MHz	256 kBytes	2 MBytes

Controller

Device:	Marvell Discovery GT−64260
Interfaces:	64−bit, 133−MHz MPC745X bus (CPU) 64−bit, 33−MHz Primary PCI Bus (PCI_0) 64−bit, 66−MHz Secondary PCI Bus (PCI_1) 64−bit, 133−MHz SDRAM 32−bit, 133−MHz FLASH Memory 8−channel DMA Ethernet and RS−232 Communication Ports Two−Wire Serial Interface (TWSI) 32 Multi−Purpose Pins (MPP)

SDRAM

Size	
Standard:	512 MBytes (64M x 64)
Option 340:	1 GByte (128M x 64)
Speed:	133 MHz
Access:	MPC745X, VMEbus, PCI Bus

FLASH Memory

Size:	32 MBytes (32M x 8)
Access:	MPC745X, PCI Bus

Communication Ports

Device:	GT−64260 Communication Unit
External Connection:	Front panel 15−pin Micro−D connector
Ethernet Data Rate:	10/100 Mbps
RS−232 Data Rate:	9600/19200/38400/57600/115200 Baud (two ports)
Access:	MPC745X, VMEbus, PCI Bus

TWSI Bus

Devices	
Board ID EEPROM:	1 kByte (1k x 8)
GT−64260 EEPROM:	1 kByte (1k x 8)
Voltage/Temperature Monitors:	Two Analog Devices ADM1024s
Programmable Clock Generator:	Cypress CY22395
Real−Time Clock:	Dallas DS1338
Access:	MPC745X, VMEbus, PCI Bus

1.14 Specifications (continued)

PCI Buses

Primary PCI Bus
- Speed: 33 MHz
- Data: 64 bits

Secondary PCI Bus
- Speed: 66/33 MHz (jumper selectable)
- Data: 64 bits

Tertiary PCI Bus (Option 150/151/152 only)
- Speed: 66/33 MHz (jumper selectable on Option PCB)
- Data: 64 bits

Quaternary PCI Bus (Option 150/151/152 only)
- Speed: 66/33 MHz (jumper selectable on Option PCB)
- Data: 64 bits

VIM Control/Status Interface
- **Device:** PLX Technology Model PCI 9656
- **PCI Bus (Primary):** 64 bits, 33 MHz
- **Local Bus:** 32 bits, 66 MHz
- **Data Transfer Modes:** Direct Slave Mode and DMA Mode
- **Access:** MPC745X, VMEbus, PCI Bus

VIM BIFOs: Four, one for each VIM site
- **Size**
  - Standard: 8 kBytes (two 1k x 32 FIFO buffers)
  - Option 320: 128 kBytes (two 16k x 32 FIFO buffers)
- **BIFO data path:** Direct to PCI interface or through FPGA
- **Access:** MPC745X, VMEbus, PCI Bus

VIM FPGAs: Two FPGAs, one for VIM A/B, one for VIM C/D
- **Device**
  - Standard: Xilinx Virtex–II XC2V1000
  - Option 300: Xilinx Virtex–II XC2V3000
- **VIM Streaming Data Interface**
  - Configuration: Two streaming data VIM interfaces per FPGA
  - Data Size: 32 bits
  - Data Interface: User configurable—factory programmed by Pentek for several standard configurations (see Chapter 6)
  - Speed
    - VIM side: 420 MBytes/second
    - Processor side: 264 MBytes/second (528 MBytes/second burst rate)
    - VIM to VIM: 420 MBytes/second
- **VIM Serial Interface**
  - Configuration: Two serial ports per FPGA, one for each VIM site (FPGA configuration dependent)
  - Data Size: 8, 16, or 32 bits
  - Data Interface: One 1–MByte FIFO for each serial port, with FIFO flags

1.14 Specifications (continued)

VIM/FPGA PCI Interface: Two, one for VIM A/B FPGA, one for VIM C/D FPGA
- **Device:** One PLX Technology Model PCI 9656 per FPGA
- **PCI Bus (Secondary):** 64 bits, 66 MHz
- **Local Bus:** 32 bits, 66 MHz
- **Data Transfer Modes:** Direct Slave Mode and DMA Mode
- **Access:** MPC745X, VMEbus, PCI Bus

VMEbus Interface
- **Device:** Tundra Universe II
- **Compliance:** VME64, A32/D64
- **Master Access:** MPC745X, PCI Bus
- **Slave Access:** VMEbus

PMC Interface
- **Quantity:** Standard: One PMC Site
  Option 150/151/152: Three PMC Sites
- **Data Width:** 64 bits
- **Clock Rate:** 66 MHz
- **Compliance:** PMC Standard IEEE P1386.1
- **Access:** MPC745X, VMEbus, PCI Bus

RACE++ Interfaces
- **Devices**
  - Primary (Option 005/010): Mercury Computer Systems PXB++ PCI Bridge
  - Second (Option 010): Pentek PMC module (Option 010 requires Mercury DLKP modules to connect to the VME backplane)
- **Speed**
  - Option 005: 267 MByte/second, peak
  - Option 010: 533 MByte/second, peak
- **Compliance:** RACEway Interlink Standard ANSI/VITA 5.1–1999
- **Access:** MPC745X, VMEbus, PCI Bus

Fibre Channel Interface (Option 006 only)
- **Device:** QLogic Model ISP2310
- **External Connection:** Front panel 7–pin HSSDC2 receptacle
- **Type:** Copper serial interface
- **Speed:** 2 Gigabits/second, peak
- **Access:** MPC745X, PCI Bus

Gigabit Ethernet Interface (Option 007 only)
- **Device:** Intel Model 82545GM Gigabit Ethernet Controller
- **External Connection:** Front panel 8–pin HSSDC receptacle
- **Type:** Ethernet interface
- **Speed:** Three speeds available:
  10 Mbps (10BaseT)
  100 Mbps (100BaseT)
  1Gbps (1000BaseT)
- **Access:** MPC745X, PCI Bus

1.14 Specifications (continued)

Physical

Dimensions:	Standard 6U VMEbus board
Depth:	160.0 mm (6.3 in)
Height:	233.5 mm (9.2 in)
Panel Width:	20.3 mm (0.8 in)
Weight:	425 grams (15 oz)

Estimated Total Power

XC2V1000 FPGAs w/base circuitry:	38.0 W
XC2V1000 FPGAs w/heavy use:	42.2 W
XC2V3000 FPGAs w/heavy use:	48.2 W

NOTE: Power consumption estimates are typical, executing code after boot, and will vary based on FPGA usage.
These power estimates are drawing from 5V power; the 4205 itself does not draw any 12V power though installed VIM/PMC mezzanine modules might.

Option 005 RACE++:	Adds 1.25 W
Option 006 Fibre Channel:	Adds 2.3 W
Option 007 Gigabit Ethernet:	Adds 2.06 W

Environmental

Operating Temperature:	0° to 50°C
Storage Temperature:	−20° to 90°C
Relative Humidity:	95% non−condensing
VME Rack Exhaust Temp:	0° to 50°C

Chapter 2: Installation and Connections

2.1 Inspection

After unpacking the Model 4205, inspect it carefully for any damage to connectors or components. If any damage is discovered, please contact Pentek immediately at (201) 818−5900. Please save the original shipping container and packing material, in case return shipment is required.

2.2 Jumper Settings

The following subsections describe user operating parameters that are set by shorting jumpers on the Model 4205 PCB. The user should not change jumpers that are not described in the following sections—those are reserved for factory test and setup purposes only.

The term jumper (or jumper block) refers to a group of two or more pins on a circuit board that may be connected in pairs by a shorting jumper to set or change an operating characteristic of the board. Pins connected in this manner are referred to as installed, or shorted. Pins that are not connected are referred to as removed, or open. The shorting jumpers used on the Model 4205 are for 0.020" (0.51 mm) square pins spaced on 0.079" (2.00 mm) centers. These jumpers are Pentek part number 356.00010 (DuPont™ part number 86730−001, or equivalent).

As shipped from the factory, several jumpers are installed in default positions on your board. The default operating parameters they select (for example, the VMEbus address of the board) may or may not meet your requirements. Before installing your Model 4205 into the VMEbus card cage, please review the following subsections to determine whether you need to change any of these settings.

Figure 2−1, on the next page, shows the locations of jumpers that can be changed for your application requirements. As shown in this illustration, a heat sink is installed on the 4205 PCB. All user jumpers can be accessed without removing this heat sink



The user should not change jumpers that are not identified in this chapter—all other jumpers are reserved for factory test and setup purposes only.

2.2 **Jumper Settings** (continued)

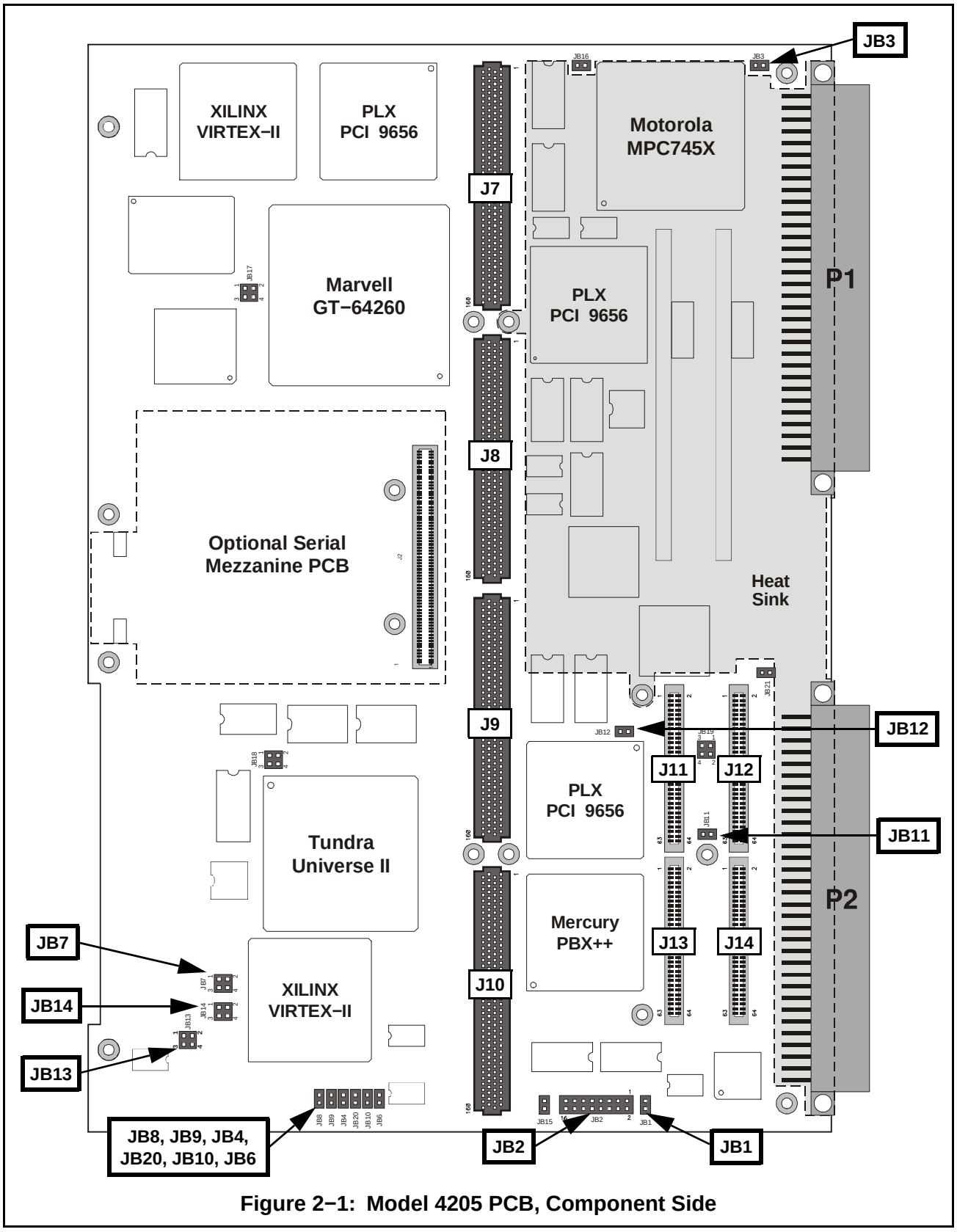


Figure 2−1: Model 4205 PCB, Component Side

2.2 Jumper Settings (continued)

2.2.1 VMEbus A16/A24 Slave Select Jumper

Jumper **JB1** configures the Model 4205 to respond as a VMEbus Slave in either A16 or A24 address space. When this jumper is removed (the factory default setting), the Model 4205 responds as a Slave in A16 space; when this jumper is installed, the Model 4205 responds as a Slave in A24 space. The following table shows the settings for this jumper. See Section 2.2.3 to set the VMEbus A16/A24 slave base address for the board.

Table 2–1: VMEbus A16/A24 Slave Select Jumper – JB1

Jumper	Jumper State	VMEbus Slave Function
JB1	Removed *	A16 Address Space Enabled
	Installed	A24 Address Space Enabled
* Factory Default Setting		

The Model 4205 is shipped with VMEbus A32 space slave access disabled. The user can enable and configure A32 slave access to the board either from the GT–64260 serial port or from VME A16/A24 space (see Section 5.3).

2.2.2 VMEbus System Reset Monitor Jumper

Jumper **JB3** enables the Model 4205 to respond to or ignore a VME System Reset. With this jumper removed (the factory default setting), the 4205 responds to a VME System Reset issued by a VMEbus system controller. This is typically the mode needed when the 4205 is NOT a Slot 1 Controller. With this jumper installed, the 4205 ignores a VME System Reset. This is typically the mode needed as Slot 1 Controller. The following table shows the settings for this jumper.

Table 2–2: VMEbus System Reset Monitor Jumper – JB3

Jumper	Jumper State	VMEbus System Reset Monitor Function
JB3	Removed *	4205 responds to VME System Reset
	Installed	4205 ignores VME System Reset
* Factory Default Setting		

NOTE: Install a jumper across **JB3** when the 4205 is installed in slot 1 as the VMEbus system controller, and remove this jumper all other times.

2.2 Jumper Settings (continued)

2.2.3 VMEbus A16/A24 Slave Base Address Jumpers

Jumper block **JB2** has eight pairs of pins that set the Model 4205 slave base address (the lowest address that this board responds to when addressed as a slave) in A16 and A24 VMEbus address space. Use Jumper **JB1** (Section 2.2.1) to select which slave address mode the board responds to, A16 or A24.

JB2 pins **1–2** correspond to the least significant VMEbus address bit A16 in A24 space, and pins **9–10** correspond to the least significant VMEbus address bit A12 in A16 space. Each higher numbered pair of jumper pins corresponds to the next more significant address bit, such that pins **15–16** correspond to address bit A23 in A24 space and to bit A15 in A16 space. Shorting a pair of jumper pins ('installed') sets the corresponding address bit to '0', and an open pair of jumper pins ('removed') sets the corresponding address bit to '1'.

The following table shows which VMEbus address bit corresponds to each pair of jumper block pins, and gives the hexadecimal address weight of each jumper position in both address spaces.

Table 2–3: VMEbus A16/A24 Slave Base Address Jumpers – JB2

Jumper JB2 Pins	VMEbus Address Bit		Hex Address Weight	
	A16	A24	A16	A24
1 – 2	–	A16	–	**0x01 0000**
3 – 4	–	A17	–	**0x02 0000**
5 – 6	–	A18	–	**0x04 0000**
7 – 8	–	A19	–	**0x08 0000**
9 –10	A12	A20	**0x1000**	**0x10 0000**
11 – 12	A13	A21	**0x2000**	**0x20 0000**
13 – 14	A14	A22	**0x4000**	**0x40 0000**
15 – 16	A15	A23	**0x8000**	**0x80 0000**

Jumper installed, bit = '0'; Jumper removed, bit = '1'
Factory Default Setting (all jumpers installed):
A16VME_base = **0x0000**
A24VME_base = **0x00 0000**

NOTE: The Model 4205 is shipped with A16/A24 slave access enabled only to the Universe II registers, at the slave base address set by **JB2**. You can configure these registers from the board's serial port or from the VMEbus, as described in Section 5.3, to enable extended A24 or A32 slave access and set the slave base address for other board resources.

2.2 Jumper Settings (continued)

2.2.4 FLASH Memory Write Enable Jumper

Jumper **JB4** enables writing to or erasing the 32–MByte FLASH memory. When this jumper is installed (the factory default setting), FLASH is write enabled. This jumper is also connected to data bit D2 of the Board Jumper Status Register, Section 3.7.6.5. The following table shows the FLASH Write Enable jumper settings.

Table 2–4: FLASH Write Enable Jumper – JB4

Jumper	Jumper State	Write Enable Function
JB4	Removed	FLASH Write Protected
	Installed *	FLASH Write Enabled
* Factory Default Setting		



FLASH is shipped with a default boot program installed by Pentek.
Users who tamper with the Pentek boot code do so at their own risk.

2.2.5 GT–64260 Serial EEPROM Write Enable Jumper

Jumper **JB6** enables writing to or erasing the GT–64260 configuration serial EEPROM. When this jumper is removed (the factory default setting), this EEPROM is write protected. The following table shows the serial EEPROM write enable jumper settings.

Table 2–5: GT–64260 EEPROM Write Enable Jumper – JB6

Jumper	Jumper State	Write Enable Function
JB6	Removed *	EEPROM Write Protected
	Installed	EEPROM Write Enabled
* Factory Default Setting		



The GT–64260 Serial EEPROM is shipped with programming information preset by Pentek, and should **NOT** be modified by the user.

2.2 Jumper Settings (continued)

2.2.6 Power–Up Mode Jumpers

Jumper block **JB7** determines the factory built–in self–test (BIST) operation at power–up or reset of the board. When both jumpers are installed (the factory default setting), the factory diagnostic tests are run at power–up or reset. These jumper pins are also connected to data bits D0 and D1 of the Board Jumper Status Register, Section 3.7.6.6. The following table shows the jumper settings for the power–up modes that can be selected by these jumper pins.

Table 2–6: Board Power–Up Mode Jumpers – JB7

JB7 Pins 1–2 Jumper State	JB7 Pins 3–4 Jumper State	Power–Up Function
Installed *	Installed *	Run BIST at power on
Removed	Installed	Skip factory BIST
Installed	Removed	Run BIST continuously
Removed	Removed	Reserved
* Factory Default Settings		

2.2.7 Secondary PCI Bus Speed Jumper

Jumper **JB8** selects the speed of the Secondary PCI Bus. When this jumper is removed (the factory default setting), the Secondary PCI Bus is clocked to run at 66 MHz. When the jumper is installed, bus speed is reduced to 33 MHz for troubleshooting purposes. This jumper is also connected to data bit D3 of the Board Jumper Status Register, Section 3.7.6.4. The following table shows the jumper settings for selecting the speed of the Secondary PCI Bus.

Table 2–7: Secondary PCI Bus Speed Jumper – JB8

Jumper	Jumper State	Secondary PCI Bus Speed
JB8	Removed *	66 MHz
	Installed	33 MHz
* Factory Default Setting		

NOTE: If you leave jumper **JB8** off, the bus defaults to 66 MHz, but if you install a 33–MHz PMC with the jumper off, the bus will switch to 33 MHz operation.

2.2 Jumper Settings (continued)

2.2.8 FLASH Boot Parameters Restore Jumper

Jumper **JB9** determines whether or not to restore the default boot parameters in FLASH memory during boot operation at power–up or reset of the board. When this jumper is removed (the factory default setting), the Pentek default boot parameters are not restored in FLASH. This jumper is also connected to data bit D6 of the Board Jumper Status Register, Section 3.7.6.2. The following table shows the jumper settings.

Table 2–8: FLASH Boot Parameters Restore Jumper – JB9

Jumper	Jumper State	Power–Up FLASH Function
JB9	Removed *	Do Not Restore boot parameters
	Installed	Restore default boot parameters in FLASH
* Factory Default Setting		



If you install jumper **JB9** for the 'Restore default boot parameters in FLASH' option, you must also install jumper **JB4** to enable writing to FLASH memory, see Section 2.2.4.

2.2.9 General Purpose Jumper

Jumper **JB10** is an undefined, general–purpose logic jumper for user applications. This jumper is connected to data bit D7 of the Board Jumper Status Register, Section 3.7.6.1. The following table shows the jumper settings.

Table 2–9: General Purpose Jumper – JB10

Jumper	Jumper State	Description
JB10	Installed *	Grounded to Logic '0'
	Removed	Pulled up to Logic '1'
* Factory Default Setting		

2.2 Jumper Settings (continued)

2.2.10 FPGA Configuration Download Jumper

Jumper **JB12** selects the source of the Xilinx Virtex–II FPGA configuration data download for both FPGAs. Depending on the setting of this jumper, the FPGA can select its configuration data either from an on–board memory (the factory default setting), or from a serial download using a Xilinx cable. Refer to VIM FPGA Configuration Control Register, Section 6.4.1, for further information on reconfiguring the Virtex FPGAs. The following table shows the jumper settings for selecting the source of the FPGA configuration data.

Table 2–10: FPGA Configuration Download Jumper – JB12

Jumper	Jumper State	FPGA Configuration Source
JB12	Removed *	On–board Configuration PROM
	Installed	Serial Download (Xilinx cable)
* Factory Default Setting		

2.2.11 FPGA General Purpose Jumpers

Jumper blocks **JB13** and **JB14** are 4–pin general–purpose jumpers for user applications. These jumpers are connected to data bits D0 through D3 of the FPGA Jumper Status Register, Section 3.7.7. The following table shows the jumper settings.

Table 2–11: FPGA General Purpose Jumpers – JB13, JB14

Jumper	Jumper Pins	Jumper State	Description
JB13	1 – 2	Installed *	Grounded to Logic '0'
		Removed	Pulled up to Logic '1'
	3 – 4	Installed *	Grounded to Logic '0'
		Removed	Pulled up to Logic '1'
JB14	1 – 2	Installed *	Grounded to Logic '0'
		Removed	Pulled up to Logic '1'
	3 – 4	Installed *	Grounded to Logic '0'
		Removed	Pulled up to Logic '1'
* Factory Default Setting			

2.2 Jumper Settings (continued)

2.2.12 Factory Default Jumpers

The following jumpers are reserved for factory test and setup purposes only.

Table 2−12: Factory Default Jumpers

Jumper	Jumper State	Default Function	
JB11	Removed *	PLX 9656 Configuration EEPROM Write Enable	
JB15	Removed *	PXB++ JTAG Enable	
JB16	Removed *	MPC745X JTAG Enable	
JB20	Removed *	Power Management	
JB21	Removed *	Reserved	
Jumper	**Jumper Pins**	**Jumper State**	**Default Function**
JB17	**1 − 2**	Installed *	Program Parts (Bypass JTAG)
	3 − 4	Installed *	
JB18	**1 − 3**	Do not care	Bypass Serial Mezzanine Board JTAG
	2 − 4	Installed *	
JB19	**1 − 3**	Do not care	Bypass PMC JTAG
	2 − 4	Installed *	
* Factory Default Settings			



DO NOT change these jumper settings!

2.3 Signal Connectors

The following four subsections describe the signal connectors on the Model 4205 PCB.

2.3.1 VIM Connectors

Four 160−pin connectors on the Model 4205 PCB, designated **J7**, **J8**, **J9**, and **J10**, provide connections for Velocity Interface Mezzanine (VIM) modules. **J7** and **J8** comprise the upper module site (VIM A and VIM B), and **J9** and **J10** comprise the lower module site (VIM C and VIM D). See Figure 2−1 for location of these connectors on the component side of the PCB. The Model 4205 can support one or two VIM−2 modules (in either site), or one VIM−4 module (which occupies both sites).

Each connector provides an identical VIM signal interface from one of the VIM nodes (A, B, C, or D). For detailed descriptions of each signal on this connector, please refer to Pentek's VIM Specification, Pentek part number 800.00002.

2.3.2 PMC Connectors

The PCI Mezzanine Card (PMC) site accepts industry−standard PMC modules for a wide variety of interface and communication functions (such as Pentek's Model 7131 Digital Receiver PMC module). The PMC is connected to the Model 4205 PCB using four 64−pin connectors designated **J11**, **J12**, **J13**, and **J14**. See Figure 2−1 for location of these connectors on the component side of the PCB.

These connectors are defined by the PMC standard, as follows:

- **J11** and **J12** contain the 32−bit PCI bus interface, plus associated power, ground, reserved pins, and other necessary signals.
- **J13** supports expansion to the 64−bit PCI bus interface.
- **J14** supports user−defined inputs/outputs. The Model 4205 routes I/O signals from the **J14** connector to the VMEbus backplane **P2** connector.

Refer to the IEEE P1386.1 PMC Standard for information concerning the PCI Mezzanine Card standards and pin definitions.

NOTE: Generally, if you install a PMC on the Model 4205, you can install only one VIM−2 module, on the upper VIM site (A and B). However, if you install the Option 010 (RACE++) PMC or the Option 150/151/152 PMC Extender Board, you can still install two VIM−2 modules or a VIM−4 module on the board.

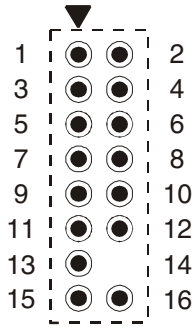
2.3 Signal Connectors (continued)

2.3.3 MPC745X COP/JTAG Connector

The MPC745X COP/JTAG connector, designated **J1** on the solder side of the PCB (see Figure 2–2, next page), provides a connection to download programs to the MPC745X, and to control the execution of those programs. This connector is reserved for Pentek factory use only. The pinout for this 16–pin header is given in the following table.

Table 2–13: MPC745X COP/JTAG Connector – J1

Signal	Pin Number		Pin Number	Signal
PPC_TDO	1		2	QACK_L
PPC_TDI	3		4	TRST_L
+ 3.3 V	5		6	+ 3.3 V
TCK	7		8	CKSTP_IN_L
TMS	9		10	No Connection
COP_SRST_L	11		12	No Connection
COP_HRST_L	13		14	(key)
CKSTP_OUT_L	15		16	GND

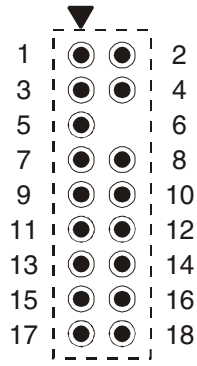


2.3.4 JTAG In–Circuit Program Chain Connector

The JTAG In–Circuit Program Chain (ISP) connector, designated **J6** on the solder side of the PCB (see Figure 2–2, next page), provides a connection to download programs and control their execution. This connector is reserved for Pentek factory use only. The pinout for this 18–pin connector is given in the following table.

Table 2–14: JTAG ISP Connector – J6

Signal	Pin Number		Pin Number	Signal
TMS	1		2	$\overline{\text{TRST}}$
TDI	3		4	GND
+ 3.3 V	5		6	(key)
TDO	7		8	GND
TCK_RET	9		10	No Connection
TCK	11		12	GND
No Connection	13		14	$\overline{\text{VIRTEX PROG}}$
VIRTEX CNFG D0	15		16	VIRTEX CCLK
VIRTEX DONE	17		18	$\overline{\text{VIRTEX INIT}}$



2.3 **Signal Connectors** (continued)

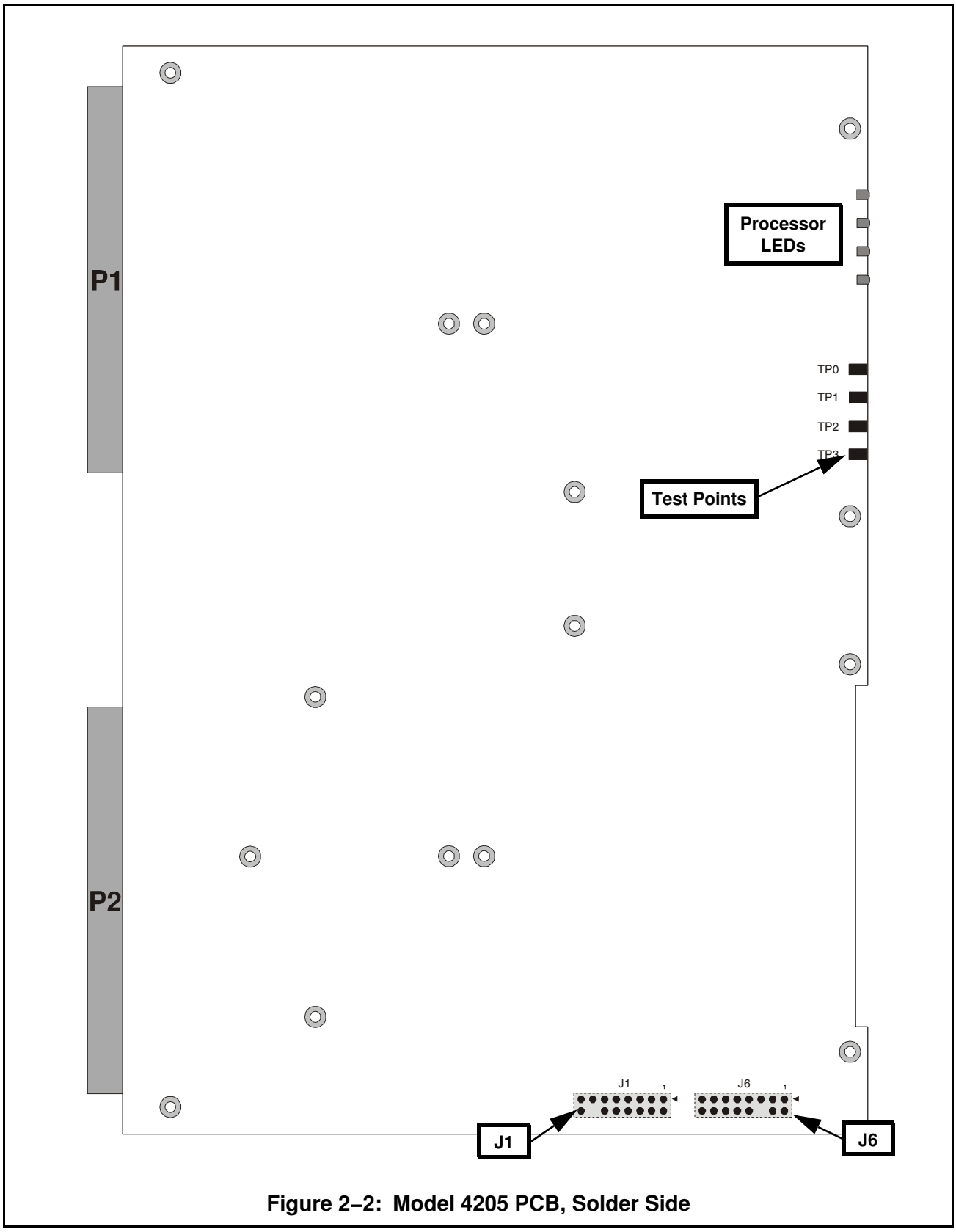


Figure 2–2: Model 4205 PCB, Solder Side

2.4 Installation

CAUTION

TURN OFF ALL POWER TO THE VME CARD CAGE BEFORE INSERTING OR REMOVING ANY BOARD.

Before you install the Model 4205 board into a VMEbus card cage, you must install any expansion VIM and/or PMC module(s) for your application. Generally, you can install one VIM−4, two VIM−2's, one VIM−2 plus one PMC, one VIM−2, or one PMC on the Model 4205 baseboard.

NOTE: If you install a PMC that has a front panel, you can install only one VIM−2 module, on the upper VIM site. However, if you install the Option 010 (RACE++) PMC, you can still install two VIM−2 modules or one VIM−4 module on the board.

Each Pentek VIM and PMC module is supplied with an Operating Manual that provides setup and installation instructions, such as jumper settings, baseboard connection, and front panel connections. Refer to the manual supplied with your module(s) for the proper installation procedures, and to Sections 2.4.1 (VIM) and 2.4.2 (PMC).

2.4.1 Installing VIM Modules

The Model 4205 board has a heat sink that uses the upper VIM site mounting hole to secure it to the PCB (see Figure 2−1). There are two different styles of heat sinks, and VIM module installation depends on the heat sink style.

- If your heat sink ***does NOT*** have the four threaded mounting holes illustrated in Figure 2−3 below (Style "A"), refer to Step a on the next page.
- If your heat sink ***does*** have the four threaded mounting holes illustrated below (Style "B", required for use with the Model 9220 PMC Extender), refer to Step b on the next page.

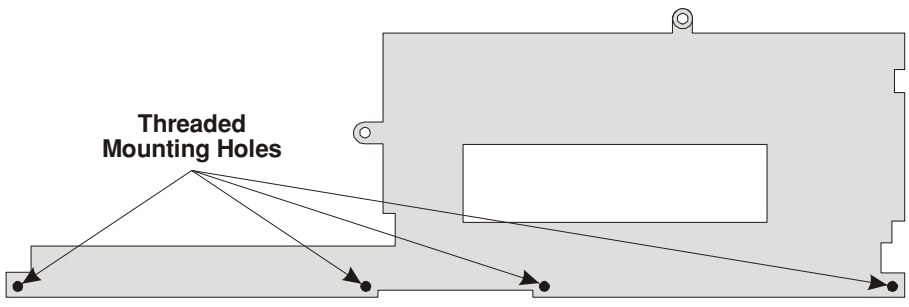


Figure 2−3: Model 4205 PCB Heat Sink − Style "B"

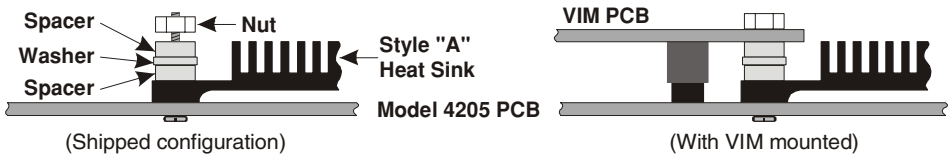
2.4 Installation (continued)

2.4.1 Installing VIM Modules (continued)

a) Mounting a VIM module on Style A Heat Sink

This style heat sink is mounted to the 4205 PCB with several spacers for attaching a VIM module. These are shown at the left (shipped) in the illustration below.

To mount a standard VIM module, remove the nylon spacer (near the VIM connectors) from the VIM module, remove the nut above the heat sink spacers, then attach the VIM to the 4205 PCB with this nut, over these spacers, as shown at the right in the illustration below.

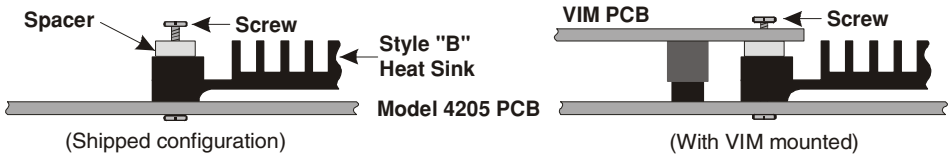


To mount a VIM module that has an I/O personality module, such as the Model 6251, also remove the top spacer and washer, leaving only one spacer, then attach the VIM PCB to the 4205 PCB with the nut.

b) Mounting a VIM module on Style B Heat Sink

This style heat sink provides a threaded hole and a spacer for attaching a VIM module to the heat sink. This is shown at the left (shipped) in the illustration below.

To mount a standard VIM module, remove the nylon spacer (near the VIM connectors) from the VIM module, then attach the VIM to the 4205 PCB heat sink, over the spacer, with this screw, as shown at the right in the illustration below.



To mount a VIM module that has an I/O personality module, such as the Model 6251, remove the spacer and attach the VIM PCB directly to the 4205 PCB heat sink with the screw.

The lower VIM site uses the VIM mounting nylon spacer and screws supplied with your VIM module.

2.4 Installation (continued)

2.4.2 Installing a PMC Module

The Model 4205 board heat sink uses one of the PMC standoff mounting holes to secure it to the PCB (see Figure 2–1). This mounting bracket on the heat sink has a threaded hole for attaching a PMC module, which takes the place of one of the PMC standoffs. The 4205 is also shipped with a PMC standoff installed in the other PMC mounting hole.

To mount your PMC module onto the 4205, use the two short Phillips screws (provided with the 4205) to secure the PMC PCB to the heat sink and to the other PMC standoff, through the PMC standoff holes.

NOTE: The Option 010 PMC module requires Mercury DLKP modules to connect to the VME backplane P2 connectors.

2.4.3 Installing the Model 4205 in the VME Card Cage

After the necessary jumpers on the Model 4205 have been set and the assembly of the board and expansion VIM and/or PMC module(s) (if any) has been completed, install the complete assembly into the VMEbus card cage.

When inserting the Model 4205 assembly, use only the front panel ejector handles of the board. After inserting the assembly into the cage card guides, push in on the top and bottom ejector handles to fully seat the connectors. Once the assembly is seated, screw the captive panel screws at the top and bottom of the front panel into the top and bottom rails of the card cage.

2.4.4 Removing the Model 4205 from the VME Card Cage

Before removing the Model 4205 assembly, first loosen the captive screws on the top and the bottom of the front panel. Then, push both ejector handles of the board away from the center of the panel to eject the assembly from the cage connectors. Once the assembly is disengaged, pull outward on the ejector handles to remove the assembly from the card cage guides.

CAUTION

TURN OFF ALL POWER TO THE VME CARD CAGE BEFORE INSERTING OR REMOVING ANY BOARD.

2.5 Front Panel

The Model 4205 front panel has five sections: three fixed plates at the top, center, and bottom of the panel, and two removable blank panels. The top fixed plate includes two LEDs labeled **Pwr** and **Temp**. The center fixed plate contains a connector for RS–232 and optional high–speed serial communications, plus four LEDs labeled **Link**, **Actv**, **Fault**, and **100**. The bottom fixed plate provides a push button labeled **Reset**. Four PCB LEDs associated with the MPC745X processor, labeled **0**, **1**, **2**, and **3**, are visible to the left of the top blank panel.

The figure at the right depicts the Model 4205 front panel, ***without*** Option 006 or 007, as it appears with no VIM or PMC module installed. See Figure 2–6 or Figure 2–7 on the following pages for the front panel center plate with Option 006 or Option 007. The following subsections describe the front panel controls, indicators, and connectors.

Before installing a VIM module, you must remove one or both blank panels for installation of the module's panel(s). The bottom blank panel has a blank cutout for a PMC module front panel. Refer to the manual supplied with your module(s) for panel installation instructions.

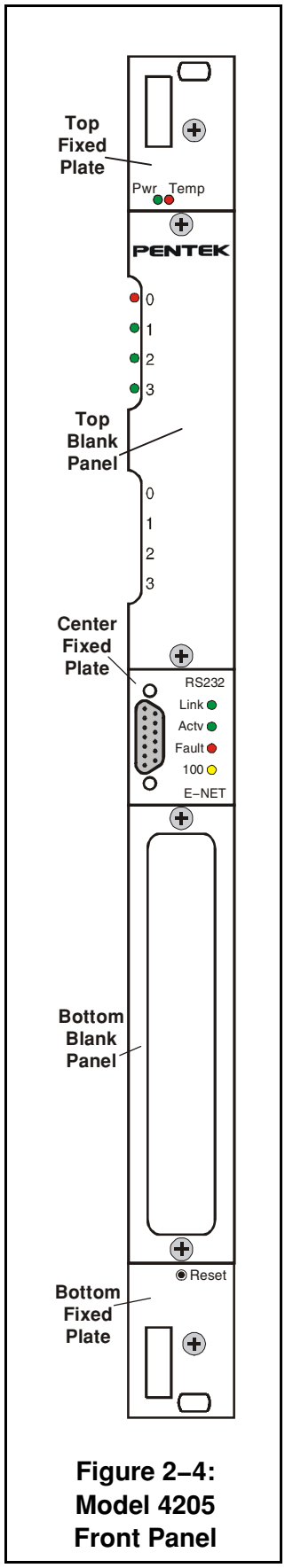


Figure 2–4: Model 4205 Front Panel

2.5.1 Power LED

At the lower left of the top fixed plate is a green LED labeled **Pwr** that illuminates when +5 V_{DC} power is applied to the Model 4205.

2.5.2 Temperature LED

At the lower center of the top fixed plate is a red LED labeled **Temp** that illuminates when either of the Model 4205 voltage/temperature monitors detects an out–of–limit temperature on the PCB. The MPC745X processor controls the sensor limits for these monitors (see Section 3.5.11).

2.5.3 Reset Button

Press the **Reset** push–button on the bottom fixed plate to start an initialization cycle that returns the Model 4205 to its power–up state. Note that this button is recessed in the panel—you must use a small probe to depress it.

2.5 Front Panel (continued)

2.5.4 Processor LEDs

At the upper left of the top blank panel are four PCB LEDs, one red and three green, labeled **0**, **1**, **2**, and **3**, that are associated with the MPC745X processor. These LEDs indicate the results of the MPC745X power−up self−tests. If all tests pass, the green LED **1** will blink at a rate of about five times per second. If any test fails, the red LED **0** will repetitively blink until power is removed or the **Reset** button is pressed (the Reset cycle does not include the power−up self−tests). See Section 8.5 for the test failure red LED blink codes.

2.5.5 I/O LEDs

The Model 4205 has several I/O LEDs on the front panel center plate. There are three different configurations of I/O LEDs, as described in the following subsections, for the Standard Model 4205, Option 006, or Option 007.

2.5.5.1 Standard Models

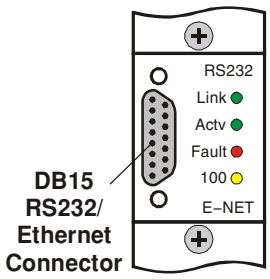


Figure 2−5: Standard Model 4205 Front Panel LEDs

The Standard Model 4205 front panel center plate, illustrated above, has four LEDs at the right of the DB15 connector.

- Green **Link** indicates a good DB15 Ethernet connection.
- Green **Actv** indicates DB15 Ethernet active.
- Red **Fault** indicates a boot fault (see Sections 8.3.2 and 8.5).
- Yellow **100** indicates DB15 Ethernet 100BaseT speed operation.

2.5 **Front Panel** (continued)

2.5.5 **I/O LEDs** (continued)

2.5.5.2 Option 006

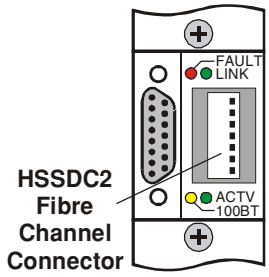


Figure 2–6: Model 4205 Option 006 Front Panel LEDs

The Model 4205 Option 006 (Fibre Channel) front panel center plate, illustrated above, has four LEDs.

Two LEDs above the Fibre Channel connector:

- Red **FAULT** indicates a boot fault (see Sections 8.3.2 and 8.5).
- Green **LINK** indicates a good DB15 Ethernet connection.

Two LEDs below the Fibre Channel connector:

- Green **ACTV** indicates DB15 Ethernet active.
- Yellow **100BT** indicates DB15 Ethernet 100BaseT speed operation.

2.5 **Front Panel** (continued)

2.5.5 **I/O LEDs** (continued)

2.5.5.3 Option 007

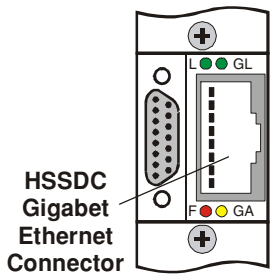


Figure 2–7: Model 4205 Option 007 Front Panel LEDs

The Model 4205 Option 007 (Gigabit Ethernet) front panel center plate, illustrated above, has four LEDs.

Two LEDs above the Gigabit Ethernet connector:

- Green **L** indicates a good DB15 Ethernet connection.
- Green **GL** indicates good HSSDC Gigabit Ethernet connection.

Two LEDs below the Gigabit Ethernet connector:

- Red **F** indicates a boot fault (see Sections 8.3.2 and 8.5).
- Yellow **GA** indicates HSSDC Gigabit Ethernet active.

Four LEDs are located on the Option 007 mezzanine PCB and are not visible on the front panel. These are labeled on the mezzanine PCB as follows:

- **ACTIVE** (**D5**) indicates DB15 Ethernet active.
- **100BT** (**D7**) indicates DB15 Ethernet 100BaseT speed.
- **LINK100** (**D3**) indicates HSSDC Gigabit Ethernet 100BaseT speed.
- **LINK1000** (**D4**) indicates HSSDC Gigabit Ethernet 1000BaseT speed.

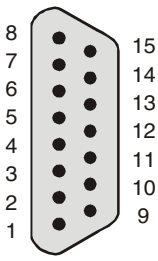
2.5 Front Panel (continued)

2.5.6 RS–232/Ethernet Connector

An RS–232 and Ethernet 15–pin Micro–DB15 connector is located on the center plate of the front panel. The GT–64260 controller (see Section 3.6.5) provides one 10/100 BaseT Ethernet LAN port and two MPSC serial ports at this connector (MPSC0 is routed to the RS–232 0 pins and MPSC1 is routed to the RS–232 1 pins). The following table shows the pinouts of this connector.

Table 2–15: RS–232/Ethernet Connector

Signal	Pin Number		Pin Number	Signal
E–net, TXP	8		15	Gnd (75Ω)
Gnd	7		14	Gnd (75Ω)
Gnd	6		13	E–net, RXN
RS–232, RCV 1	5		12	E–net, RXP
RS–232, RCV 0	4		11	Gnd (75Ω)
RS–232, XMIT 1	3		10	Gnd (75Ω)
RS–232, XMIT 0	2		9	E–net, TXN
Gnd	1			



Pentek provides a mating RS–232/Ethernet cable, Pentek Model 2142. This cable has three breakout connectors (see Figure 2–8 on the next page): one Ethernet RJ45 connector and two RS–232 DB9 connectors. The cable length from the front panel Micro–DB15 connector to the DB9 (1) connector is 18 inches; the cable length from the DB9 (1) connector to each of the other two breakout connectors is 6 inches. The following table shows the pin–to–pin signal wiring from the front panel RS–232/Ethernet DB15 connector to each breakout connector.

Table 2–16: RS–232/Ethernet Cable Pin Connections

Signal	DB15 Pin	RJ45 Pin	DB9 (1) Pin	DB9 (2) Pin
RS–232, XMIT 0	2	NC	2	NC
RS–232, XMIT 1	3		NC	2
RS–232, RCV 0	4		3	NC
RS–232, RCV 1	5		NC	3
GND	6		5	5
E–net, TXP	8	3	NC	NC
E–net, TXN	9	6		
E–net, RXP	12	1		
E–net, RXN	13	2		

2.5 Front Panel (continued)

2.5.6 RS–232/Ethernet Connector (continued)

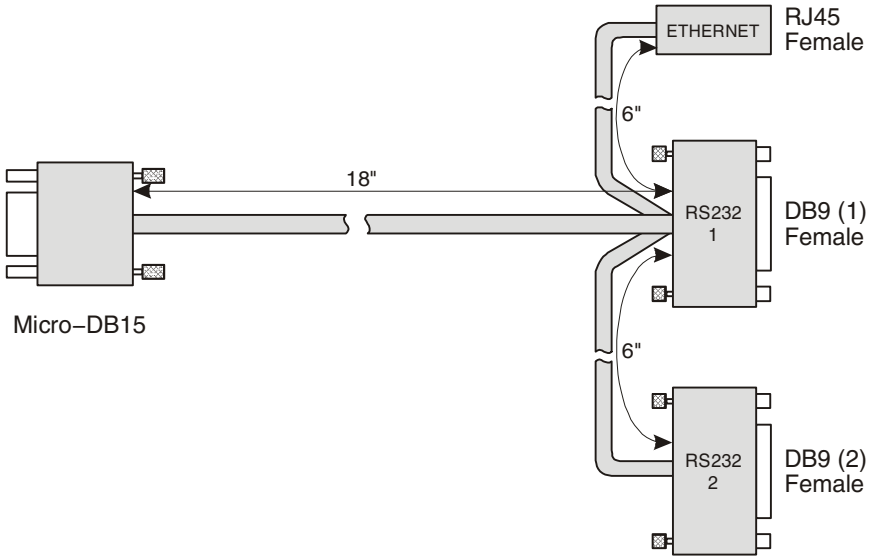


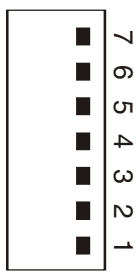
Figure 2–8: RS–232/Ethernet Cable
(Pentek Model 2142)

2.5.7 Fibre Channel Connector (Option 006)

With Option 006, a 7–pin copper Fibre Channel connector is provided on the center plate of the front panel (see Figure 2–5), Tyco/AMP part # 1364927–1 HSSDC2 receptacle. This connector is compatible with most RAID/JBOD storage device fibre–channel connectors. The following table shows the pinouts of this connector.

Table 2–17: Fibre Channel Connector

Signal	Pin Number
GND	7
TRANS–	6
TRANS+	5
GND	4
RCV+	3
RCV–	2
GND	1



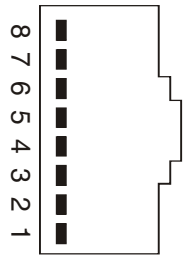
2.5 Front Panel (continued)

2.5.8 Gigabit Ethernet Connector (Option 007)

With Option 007, an 8–pin Gigabit Ethernet connector is provided on the center plate of the front panel (see Figure 2–5), Tyco/AMP part # 636180–1 HSSDC receptacle. The following table shows the pinouts of this connector.

Table 2–18: Gigabit Ethernet Connector

Signal	Pin Number
TP3–	8
TP3+	7
E–net TXN	6
TP2–	5
TP2+	4
E–net TXP	3
E–net RXN	2
E–net RXP	1



Pentek provides a mating cable, Pentek Model 2144 (available in 3, 6, or 10 foot lengths), which routes the Gigabit Ethernet connector E–net signals (indicated in the table above) to an Ethernet RJ45 plug that can be used with standard Ethernet interfaces.

2.5.9 Front Panel Test Points

To the left of the top blank panel, below the four Processor LEDs, you can access four test points located on the front edge, solder side, of the Model 4205 PCB. These are identified as **TP0**, **TP1**, **TP2**, and **TP3** on the PCB, as indicated in the figure at the right, and in Figure 2–2 on page 44. These test points provide two general–purpose signals and two grounds, as follows.

TP0 – Trigger Output
TP1 – Ground
TP2 – Ground
TP3 – Interrupt Input

The Trigger Output, **TP0**, is a logic output signal that is controlled by the board's LED Control Register, see Section 3.7.1. The Interrupt Input, **TP3**, is a logic input signal that is routed to the GT–64260 MPP13 pin, see Section 3.6.8.7.

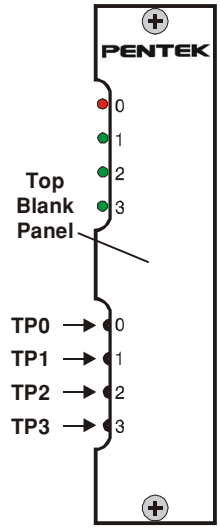


Figure 2–9: Front Panel Test Points

2.5 Front Panel (continued)

2.5.10 Option 150/151/152 PMC Front Panel

The Option 150/151/152 PMC Extender Board for VMEbus provides additional PCI Mezzanine Card (PMC) sites for the Model 4205. This board extends the baseboard PMC connector into a second VME–bus slot, where up to three PMC modules can be mounted. These three PMC sites are identified as **PMC 1**, **PMC 2**, and **PMC 3**, as illustrated in the figure at the right. This additional PMC support is pro–vided without interfering with baseboard VIM mez–zanine sites, so that up three PMC modules and two VIM–2 modules can be mounted on a single base–board simultaneously.

NOTE: The Option 150/151/152 PMC Extender Board is documented separately from the Model 4205, as the Model 9220 PMC Extender Board. Refer to the Model 9220 Operating Manual, part # 800.92200 (which is supplied with Model 4205 Option 150/151/152), for detailed description of the Extender Board setup and use.

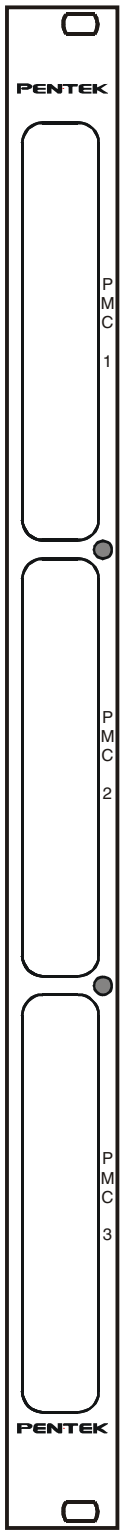


Figure 2–10: Option 15x Front Panel

2.6 Power On Reset

The following diagram shows the reset action for the Model 4205 following an operator power on or reset of the board, or a VMEbus System Reset. The resets issued by the Board Reset Register (Section 3.7.2) and the Watchdog Timer logic (Section 3.7.8) are also shown. Note that the Power On Reset resets the Temperature Monitors and BIST registers, but the Reset Button, VMEbus Reset, and Register Resets do not.

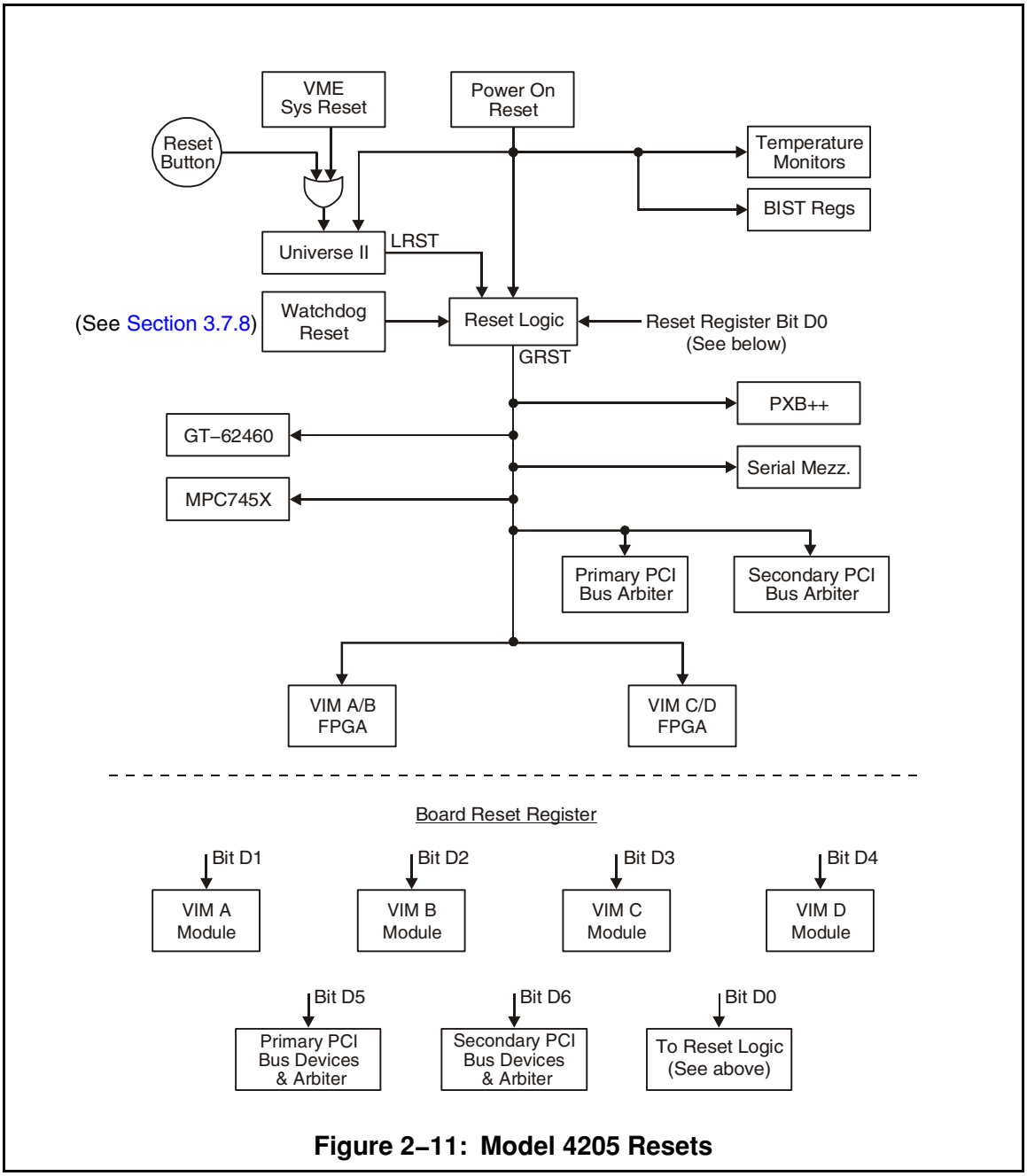


Figure 2–11: Model 4205 Resets

Chapter 3: Processor Memory Maps and Resources

3.1 Overview

This chapter presents memory maps and operation of Model 4205 resources available to the MPC745X processor, through the GT−64260 controller. The MPC745X provides high−performance processing for your embedded applications, and the GT−64260 controller supports the MPC745X with PCI interfaces, local buses, and communications ports. The GT−64260 also acts as the PCI bus bridge between the Primary and Secondary PCI Buses. The following sections describe MPC745X operation of these resources. Pentek board support software routines (see Section 1.12) are available to control the operation of these resources.

In addition to the resource descriptions in this chapter, refer to Chapter 6 for VIM BIFO and FPGA operation, Chapter 7 for interrupt operation, and Chapter 8 for FLASH memory and firmware operation.

3.2 MPC745X Processor

The Motorola MPC745X processor is an implementation of the PowerPC microprocessor family of reduced instruction set computer (RISC) microprocessors. The processor implements the 32−bit portion of the PowerPC architecture, which provides 32−bit effective addresses, integer data types of 8, 16, and 32 bits, and floating−point data types of 32 and 64 bits. The MPC745X is a high−performance, superscalar microprocessor that can dispatch and complete two instructions simultaneously. It incorporates eight independent execution units and three register files, and provides an external Level 2 (L2) cache interface. The MPC745X utilizes the AltiVec vector engine to perform parallel processing of multiple data elements (SIMD) with 128−bit operations. The AltiVec processor executes both fixed and floating−point instructions, augmenting the separate integer and floating point units also on the chip.

Model 4205 PowerPC processor specifications:

Processor	Clock Rate	Level 2 Cache	Level 3 Cache	
MPC7457	1 GHz	512 kBytes	4 MBytes	
MPC7455	600 MHz	256 kBytes	2 MBytes	(Obsolete)

Detailed description of this processor is provided in Motorola MPC7450 RISC Microprocessor Family User's Manual, MPC7450UM/D.

The MPC745X processor has access to several resources through the Marvell GT−64260 64−bit, 133−MHz CPU bus, plus access to other board resources using the GT−64260 controller's PCI interface. See Section 3.6 for description of the GT−64260 Controller. The three tables on the next two pages provide memory maps of the resources available to the MPC745X.

3.2 MPC745X Processor (continued)

3.2.1 MPC745X Processor Memory Map

The following table shows the resources accessible from the MPC745X, the memory addresses of that resource, the path to each resource, and a reference to MPC745X operation of each resource. Many of these resources are also accessible from the PCI bus, Chapter 4, and the VMEbus, Chapter 5.

Table 3–1: MPC745X Processor Memory Map

MPC745X Memory Address	Resource	Path	Information
0x0000 0000 – 0x3FFF FFFF	SDRAM	GT–64260	Section 3.3
0x4000 0000 – 0x4000 0FFF	Board Control/Status Registers		Section 3.7
0x4000 1000 – 0x401F FFFF	Unused		–
0x4020 0000 – 0x4020 0FFF	Board Interrupt Registers		Section 3.8
0x4020 1000 – 0x402F FFFF	Unused		–
0x4030 0000 – 0x4030 0FFF	Board Interrupt Registers		Section 3.8
0x4030 1000 – 0x403F FFFF	Unused		–
0x4040 0000 – 0x405F FFFF	PCI Bus I/O Access	PCI Buses	Section 3.2.2
0x4060 0000 – 0x4060 0FFF	Universe II Registers	Primary PCI Bus	Section 3.5.1
0x4061 1000 – 0x406F FFFF	Unused		–
0x4070 0000 – 0x4070 00FF	PLX 9656 Registers, VIM Control/Status		Section 3.5.3
0x4070 0100 – 0x407F FFFF	Unused		–
0x4080 0000 – 0x408F FFFF	VIM Module Access		Section 3.5.8
0x4090 0000 – 0x409F FFFF	VIM Interface Configuration Registers		Section 3.5.9
0x40A0 0000 – 0x40AF FFFF	RACEway PXB++ Registers (Option 005/010)		Section 3.5.6
0x40B0 0000 – 0x40BF FFFF	PXB Reserved	–	–
0x40C0 0000 – 0x40DF FFFF	Unused	–	–
0x40E0 0000 – 0x40E0 00FF	PLX 9656 Registers, VIM A/B FPGA	Secondary PCI Bus	Section 3.5.3
0x40E0 0100 – 0x40EF FFFF	Unused		–
0x40F0 0000 – 0x40F0 00FF	PLX 9656 Registers, VIM C/D FPGA		Section 3.5.3
0x40F0 0100 – 0x40FF FFFF	Unused		–
0x4100 0000 – 0x41FF FFFF	VIM A/B BIFO/FPGA Access		Section 3.5.10
0x4200 0000 – 0x42FF FFFF	VIM C/D BIFO/FPGA Access		
0x4300 0000 – 0x43FF FFFF	Unused		–
0x4400 0000 – 0x5FFF FFFF	PMC Module Access		Section 3.5.5
	Second RACEway Access (Option 010)		Section 3.5.6
	Option 150/151/152 PMC Module Access		Section 3.5.7
0x6000 0000 – 0x7FFF FFFF	Serial Mezzanine Module Access (Fibre Channel or Gigabit Ethernet)		Section 3.5.4
0x8000 0000 – 0xDFFF FFFF	VME Master Access	Primary PCI Bus	Section 3.5.2
0xE000 0000 – 0xEFFF FFFF	Primary RACEway Access (Option 005/010)		Section 3.5.6
0xF000 0000 – 0xF0FF FFFF	Unused	–	–
0xF100 0000 – 0xF100 FFFF	GT–64260 Internal Registers	GT–64260	Section 3.6
0xF101 0000 – 0xFDFF FFFF	Unused		–
0xFE00 0000 – 0xFFFF FFFF	FLASH Memory		Section 3.4

3.2 MPC745X Processor (continued)

3.2.2 MPC745X PCI Bus I/O Map

The MPC745X can also access certain resources using PCI I/O–mapped addresses. The following table shows the MPC745X memory addresses for these resources and a reference to information on each resource.

Table 3–2: MPC745X PCI Bus I/O Address Map

MPC745X Memory Address	Resource	Path	Information
0x4040 1000 – 0x4040 1FFF	Universe II Registers	Primary PCI Bus	Section 3.5.1
0x4040 2000 – 0x4040 20FF	PLX 9656 Registers, VIM Control/Status		Section 3.5.3
0x4050 8000 – 0x4050 80FF	PLX 9656 Registers, VIM A/B FPGA	Secondary PCI Bus	Section 3.5.3
0x4050 9000 – 0x4050 90FF	PLX 9656 Registers, VIM C/D FPGA		
0x4050 A000 – 0x4050 A0FF	PMC Module *		Section 3.5.5
0x4050 B000 – 0x4050 BFFF	Serial Mezzanine Module *		Section 3.5.4
0x4050 C000 – 0x4050 EFFF	Option 150/151/152 PMC Module		Section 3.5.7

* Module dependent—I/O–mapped addresses may not be present on all PCI devices.

3.2.3 GT–64260 TWSI Bus Address Map

Using the GT–64260 Two–wire Serial Interface (TWSI) (see Section 3.6.7), the MPC745X can access the board's voltage and temperature monitors, programmable clock generator, real–time clock, GT–64260 EEPROM, and board ID EEPROM. The following table shows the TWSI bus address for each resource and a reference to information on each resource.

Table 3–3: GT–64260 TWSI Bus Address Map

Binary Bus Address	Resource	Information
0101100b	Voltage/Temperature Monitor 1	Section 3.5.11
0101101b	Voltage/Temperature Monitor 2	
1010000b	GT–64260 EEPROM	Section 3.5.12
1010100b	Board ID EEPROM	Section 3.5.13
1101000b	Real–Time Clock	Section 3.5.14
1101001b	Programmable Clock Generator	Section 3.5.15

Note: Each device on the GT–64260 TWSI bus has a 7–bit address.

NOTE: These resources are programmed by Pentek for proper board operation and should not be modified by the user.

3.3 SDRAM

The MPC745X has access to a 512–MByte SDRAM (1–GByte SDRAM with Option 340), which is used for boot operation at reset or power–up of the board. The Pentek–supplied default boot firmware uses fixed areas of SDRAM for controlling and storing results of boot initialization. The following table shows the general SDRAM memory map. See Section 8.3 for detailed memory maps of the SDRAM Boot Area.

Table 3–4: SDRAM Memory Map

MPC745X Memory Address	SDRAM Memory Area
0x0000 0000 – 0x000F FFFF	Boot Area (see Table 8–2, Section 8.3.1)
0x0010 0000 – 0x3FFF FFFF	User Area



The first 1 MByte of SDRAM (**0x0000 0000** to **0x000F FFFF**) is reserved for use by the default boot firmware (Section 8.3.1).

Loading user code into this area or accessing data in this area can cause unpredictable behavior.

In addition, the default MPC745X firmware includes a set of user functions that can be called using a Command Table in SDRAM (see Section 8.4).

3.4 FLASH Memory

A 32–MByte FLASH memory is provided for nonvolatile storage of initialization, self–test, and boot code. User boot code may be added to FLASH for individual application requirements. When the Model 4205 is reset or power is applied, the MPC745X copies the boot code from FLASH to SDRAM and then executes the boot code from SDRAM. Using the Pentek–supplied default boot firmware, at the end of the boot sequence the MPC745X checks for user boot code in FLASH, and then loads and executes the user code if it is present. See Chapter 8 for a detailed memory map of FLASH and description of the Pentek–supplied default boot firmware operation.

The MPC745X processor can access FLASH as a memory–mapped resource starting at memory address **0xFE00 0000**. FLASH memory is write–protected using PCB jumpers. See Section 2.2.4 for these jumper settings.



Pentek ships FLASH memory with a default boot program (Section 8.3) and MPC745X firmware functions (Section 8.4) installed. If you write–enable FLASH, this default Pentek code is not protected.

Users who tamper with the Pentek boot code do so at their own risk.

3.5 MPC745X Resources Operation

The following subsections describe operation of board resources available to the MPC745X processor. Pentek board support software (see Section 1.12) is available for operation of these resources.

3.5.1 Universe II Registers

The Tundra Universe II VME interface allows the MPC745X to act as VMEbus Master for data transfers to or from a VMEbus device (see Section 3.5.2), and provides VMEbus Slave access to board resources from an external VMEbus Master. Refer to Chapter 5 for description of Model 4205 VME operation.

The Universe II has programmable registers for VME system configuration and control: PCI Configuration Space, VMEbus Configuration and Status Reg–isters, and Universe II Device Specific Status Registers. A reference list of these registers is provided in Appendix B. Refer to the Tundra Universe II User Manual for description of these registers. The MPC745X can access the Universe II registers starting at memory address **0x4060 4000**, or starting at memory address **0x4040 1000** using PCI I/O–mapped addresses.

3.5.2 VMEbus Master Access

When the Model 4205 is a VMEbus Master, the MPC745X has Master access to the VMEbus and can perform D16, D32, and D64 block transfers in all relevant VMEbus address spaces. See Section 5.2 for further information on VMEbus Master operation. For VMEbus Master access, the following table shows the MPC745X memory addresses available in each VMEbus address space and the VMEbus addresses generated for each MPC745X memory address.

Table 3–5: VMEbus Master Addresses

VMEbus Address Space (Address Modifiers)	MPC745X Memory Address	Generated VME Address
A16/D16 (Non–Privileged, 0x29)	**0x8001 0000 – 0x8001 FFFF**	**0x0000 – 0xFFFF**
A16/D32 (Non–Privileged, 0x29)	**0x8002 0000 – 0x8002 FFFF**	**0x0000 – 0xFFFF**
A24/D16 (Standard, Non–Privileged, Data, 0x39)	**0x8100 0000 – 0x81FF FFFF**	**0x00 0000 – 0xFF FFFF**
A24/D32 (Standard, Non–Privileged, Data, 0x39)	**0x8200 0000 – 0x82FF FFFF**	**0x00 0000 – 0xFF FFFF**
A32/D16 (Extended, Non–Privileged, Data, 0x09)	**0x9000 0000 – 0x9FFF FFFF**	**0x0000 0000 – 0x0FFF FFFF**
A32/D32 (Extended, Non–Privileged, Data, 0x09)	**0xA000 0000 – 0xDFFF FFFF**	**0x0000 0000 – 0x3FFF FFFF**
A32/D64 (Extended, Non–Privileged, Data, 0x09)	**0x8400 0000 – 0x8FFF FFFF**	**0x0000 0000 – 0x0BFF FFFF**

The default boot sequence (Chapter 8) configures the Universe II registers for the VMEbus addresses shown above. The Monitor program (Chapter 9) pro–vides commands for reconfiguring VMEbus Master access at boot time.

3.5 MPC745X Resources Operation (continued)

3.5.3 PLX 9656 Registers

The Model 4205 has three PCI bus interfaces, each using a dedicated PLX Technology PCI 9656 Bus Master. One PLX 9656 is on the Primary PCI Bus for the VIM control and status interfaces of all four VIM sites, and the other two are on the Secondary PCI Bus for the VIM A/B FPGA data interfaces and the VIM C/D FPGA data interfaces.

Each PLX 9656 has programmable registers to control its PCI interface. These registers are described in the PLX PCI 9656 Data Book. See Appendix C for a reference list of the PCI Configuration Space Registers. The MPC745X can access the registers for each PLX 9656 using the PCI memory−mapped or PCI I/O−mapped addresses listed in the following table.

Table 3−6: PLX 9656 Registers Addresses

PLX 9656	MPC745X Memory Address	MPC745X (PCI I/O) Address
VIM Control/Status	0x4070 0000 – 0x4070 FFFF	0x4040 2000 – 0x4040 20FF
VIM A/B FPGA	0x40E0 0000 – 0x40E0 FFFF	0x4050 8000 – 0x4050 80FF
VIM C/D FPGA	0x40F0 0000 – 0x40F0 FFFF	0x4050 9000 – 0x4050 90FF

3.5.4 Serial Mezzanine Module Access

The serial mezzanine site provides an optional high−speed serial data interface to the Model 4205 front panel from the Secondary PCI Bus. If a serial mezzanine module is installed, the MPC745X can access the module's local address space resources (module I/O, memory, registers, DMA, etc.) starting at memory address **0x6000 0000**, or starting at memory address **0x4050 B000** if PCI I/O−mapped access is implemented on the module.

3.5.4.1 Fibre Channel Access (Option 006)

When the Option 006 Fibre Channel module is installed on the serial mezzanine site, the Model 4205 allows up to 2 Gigabits/second serial transfers using a front panel copper Fibre Channel serial I/O connector.

3.5.4.2 Gigabit Ethernet Access (Option 007)

When the Option 007 Gigabit Ethernet module is installed on the serial mezzanine site, the Model 4205 allows 1000BaseT Gigabit Ethernet I/O transfers using a front panel Ethernet connector.

3.5 MPC745X Resources Operation (continued)

3.5.5 PMC Module Access

The PCI Mezzanine Card (PMC) site accepts industry−standard PMC modules, interfaced to the Secondary PCI Bus. If a PMC module is installed, the MPC745X can access PMC local address space resources (module I/O, memory, registers, DMA, etc.) starting at memory address **0x4400 0000**, or starting at memory address **0x4050 A000** if PCI I/O−mapped access is implemented on the PMC module.

Refer to the Operating Manual supplied with your PMC module for description of its operation and programmable resources. Refer to Section 3.5.6, below, for the Pentek Model 4205 Option 010 RACE++ PMC module.

3.5.6 RACEway Access (Option 005 or 010)

Model 4205 Option 005 provides a single RACEway RACE++ interface to the Primary PCI Bus. Option 010 includes the Option 005 interface and adds a second RACE++ interface to the Secondary PCI Bus. Pentek board support software (Section 1.12) is available for operation of RACEway.

When either Option 005 or 010 is installed, the Model 4205 supports outgoing (Master) and incoming (Slave) RACEway operations using the PCI bus. With Option 005, the Model 4205 allows data transfers up to 267 MBytes/second with other RACEway−equipped VME boards. Option 010 allows data transfers up to 533 MBytes/second with other RACEway−equipped VME boards (Mercury RACE++ DLKP interlink modules are required with Option 010). Refer to Appendix D for description of Raceway operation.

NOTE: The optional RACEway interfaces can generate Broadcast/Multicast writes, but cannot receive them.

Options 005 and 010 both include the primary RACE++ interface to the Primary PCI Bus. This interface uses a PXB++ PCI Bridge chip from Mercury Computer Systems. The MPC745X can access the primary RACEway address space starting at memory address **0xE000 0000**, and can access the PXB++ internal registers starting at memory address **0x40A0 0000**.

Option 010 provides a second RACE++ interface to the Secondary PCI Bus. Option 010 occupies the PMC site (Section 3.5.5, above), and uses a PXB++ PCI Bridge chip. The MPC745X can access the second RACEway address space starting at memory address **0x5000 0000**, and can access the PXB++ internal registers starting at memory address **0x4400 0000**. (PCI I/O−mapped access is not implemented on the Option 010 PMC module.)

NOTE: The Pentek default boot sequence (Chapter 8) configures the PXB++ internal registers for proper operation with Pentek's board support software—you do not need to access these registers.

3.5 MPC745X Resources Operation (continued)

3.5.7 Option 150/151/152 PMC Module Access

The Option 150/151/152 PMC Extender Board for VMEbus provides additional PCI Mezzanine Card (PMC) sites for the Model 4205. This board extends the baseboard PMC connector into a second VME–bus slot, where up to three PMC modules can be mounted. These three PMC sites are identified as **PMC 1**, **PMC 2**, and **PMC 3**, as illustrated in the figure at the right.

NOTE: The Option 150/151/152 PMC Extender Board is documented separately from the Model 4205, as the Model 9220 PMC Extender Board. Refer to the Model 9220 Operating Manual, part # 800.92200 (which is supplied with Model 4205 Option 150/151/152), for detailed description of the Extender Board setup and use.

If a PMC module is installed on Option 150/151/152 PMC Extender Board, depending on the Extender Board site the PMC is mounted on (see figure at the right), the MPC745X can access PMC local address space resources (module I/O, memory, registers, DMA, etc.) at the PCI memory–mapped or PCI I/O–mapped addresses listed in the following table.

Table 3–7: Option 150/151/152 PMC Addresses

Option 15x PMC Site	MPC745X Memory Address	MPC745X (PCI I/O) Address
PMC 1	0x4400 0000 – 0x4FFF FFFF	0x4050 C000 – 0x4050 CFFF
PMC 2	0x5000 0000 – 0x57FF FFFF	0x4050 D000 – 0x4050 DFFF
PMC 3	0x5800 0000 – 0x5FFF FFFF	0x4050 E000 – 0x4050 EFFF

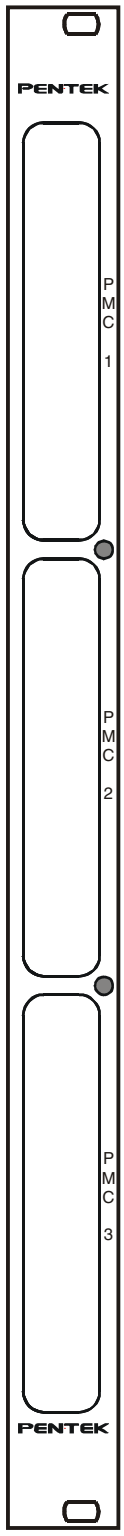


Figure 3–1: Option 15x Front Panel

3.5 MPC745X Resources Operation (continued)

3.5.8 VIM Module Access

Each Pentek VIM module has resources that can be accessed by processors on the VME baseboard, including programmable registers that configure and control its operation. Each VIM module is supplied with an Operating Manual that describes its operation and any programmable resources on that module. Refer to that manual for information on operation and programming these resources, including any on–board control registers and data storage.

When a VIM module is installed on the Model 4205, the MPC745X can access each VIM site's on–board resources at the addresses listed in the following table.

Table 3–8: VIM Module Access Addresses

VIM Interface	MPC745X Memory Address	VIM Module Information
VIM A	**0x4080 0000 – 0x4083 FFFF**	Refer to the Operating Manual supplied with your VIM module
VIM B	**0x4084 0000 – 0x4087 FFFF**	
VIM C	**0x4088 0000 – 0x408B FFFF**	
VIM D	**0x408C 0000 – 0x408F FFFF**	

Each VIM interface has a unique MPC745X base address. This allows the MPC745X to address the VIM resources that are identified for each baseboard processor: A, B, C, or D (most Pentek VIM modules identify access to their resources in terms of baseboard processors). For example, if a particular VIM resource can be accessed by Processor C on the baseboard, the MPC745X accesses that resource starting at base address **0x4088 0000**.

Pentek has reserved the first 128 kBytes of each VIM memory for I/O space, and the remaining region for memory space. At the time of this publication, there are no VIM modules available from Pentek that utilize the I/O space. VIM module control registers are generally mapped into the memory space, at a fixed memory offset from the starting address—this offset is **0x20000**.

For example, to access the first register of a Pentek VIM C module from the Processor C baseboard interface, the MPC745X would access that register at address **0x408A 0000** (VIM C Module base address [**0x4088 0000**] + VIM module register offset [**0x20000**]).

Refer to Section 6.3 for additional description of VIM module operation.

3.5 MPC745X Resources Operation (continued)

3.5.9 VIM Interface Configuration Registers

Each VIM interface has a bi−directional FIFO (BIFO) that buffers data transfers between the VIM module and the MPC745X. Each VIM BIFO has a streaming data interface through one of two on−board Xilinx Virtex−II FPGAs. One of these FPGAs is dedicated to VIM A and B interfaces and the other is dedicated to VIM C and D.

The Model 4205 has several 32−bit registers that configure and control the operation of these VIM BIFO and FPGA interfaces. The MPC745X can access these registers at the memory addresses listed in the following table. The referenced subsections provide information about the use and programming of each register. Refer to Section 6.4 for detailed descriptions of these VIM interface configuration registers.

Table 3−9: VIM Interface Configuration Register Addresses

MPC745X Memory Address	Register	Access	Register Information
0x4090 0000	VIM A/B FPGA Configuration Control Register	R/W	Section 6.4.1
0x4090 0004	VIM A/B FPGA Configuration Data Register	W.O.	Section 6.4.2
0x4090 0008	VIM C/D FPGA Configuration Control Register	R/W	Section 6.4.1
0x4090 000C	VIM C/D FPGA Configuration Data Register	W.O.	Section 6.4.2
0x4090 0010	VIM Interface BIFO Control Register	R/W	Section 6.4.3
0x4090 0014	VIM Interface BIFO Reset Status Register	R.O.	Section 6.4.4
0x4090 0018	VIM FPGA DCM Reset Register	R/W	Section 6.4.5

NOTE: Several of the VIM BIFO/FPGA resources described in this manual are dependent on the configuration determined by the above registers. In particular, the VIM Interface BIFO Control Register (Section 6.4.3) selects the data routing from the VIM BIFO to other board resources. Some of these resources are only available for a configuration that routes data from the VIM BIFOs to the FPGA (VIM BIFO PATH SELECT = FPGA), and others are only available for a configuration that routes data from the VIM BIFOs to the PCI interface (VIM BIFO PATH SELECT = PCI Interface).

3.5 MPC745X Resources Operation (continued)

3.5.10 VIM BIFO/FPGA Access

Each VIM interface has a bi−directional FIFO (BIFO) that buffers data transfers between the VIM module and the MPC745X. Each VIM BIFO can be routed through a streaming data interface in one of two on−board Xilinx Virtex−II FPGAs. One FPGA is dedicated to VIM A and B interfaces and the other is dedicated to VIM C and D. Serial ports provide an additional data streaming connection between the VIM module and the MPC745X.

The VIM BIFO and FPGA interfaces can be configured in different ways using the VIM Interface Configuration Registers, Section 3.5.9. Two default VIM BIFO/FPGA configurations are software selectable at boot time: VIM to Processor, and VIM to VIM. Different memory maps are provided for the VIM BIFO/FPGA resources available to the MPC745X for each configuration, as described in Section 6.5. Pentek board support software (Section 1.12) is available for controlling VIM BIFO and FPGA access.

- **VIM to Processor**: Each FPGA is configured with two 2 x 4k x 32 dual−ported random access memory (DPRAM) swinging buffers. Data from each VIM BIFO can be fed to an individual DPRAM swinging buffer. The DPRAM buffers can transfer data to the MPC745X by master or slave DMA using a 2k x 32 FPGA DMA RAM memory.

  This configuration also allows direct MPC745X read/write access to the VIM BIFO Data and Mailbox resources, and to the VIM Serial Ports.

  See Section 6.5.1 for the memory maps of the VIM BIFO/FPGA resources available to the MPC745X for the VIM to Processor configuration. Also, see Figure 6−1 for illustration of this configuration.

- **VIM to VIM**: Each FPGA is configured to allow direct 105−MHz data transfers between VIM sites A and C, and between VIM sites B and D, respectively.

  This configuration does not include the FPGA DPRAM swinging buffers or the FPGA DMA RAM memory—these are available only in the VIM to Processor configuration above. However, access to the VIM Serial Ports, VIM BIFO Data, and VIM BIFO Mailboxes from the MPC745X is the same in both configurations.

  See Section 6.5.2 for the memory maps of the VIM BIFO/FPGA resources available to the MPC745X for the VIM to VIM configuration. Also, see Figure 6−2 for illustration of this configuration.

3.5 MPC745X Resources Operation (continued)

3.5.11 Voltage/Temperature Monitors

The Model 4205 has two Analog Devices ADM1024 voltage/temperature monitors that provide constant monitoring of critical voltages and temperatures on the PCB. Each ADM1024 controls several temperature and voltage sensors using programmable limits. The MPC745X sets the voltage and temperature limits using ADM1024 internal registers. These ADM1024 registers can be accessed through the GT−64260 TWSI interface, Section 3.6.7.



Each ADM1024 is programmed with default limits by the Pentek boot firmware, and should **NOT** be modified by the user.

During normal operation the MPC745X reads the current ADM1024 voltages and temperatures at regular intervals, and stores the new values into SDRAM (see Section 8.3.1). If a monitored voltage or temperature falls outside of the limits, the ADM1024s assert an interrupt to the MPC745X and the VMEbus (see Section 7.10). For an over−temperature, the ADM1024s also illuminate the red **Temp** LED (Section 2.5.2) on the front panel.

3.5.12 GT−64260 EEPROM

The 1−kByte GT−64260 serial EEPROM contains GT−64260 Controller configuration information, and is used at boot time to initialize the GT−64260 for the memory maps and operation shown in this chapter (see Section 3.6). This GT−64260 serial EEPROM can be accessed through the GT−64260 TWSI interface, Section 3.6.7.



The GT−64260 EEPROM is shipped with the GT−64260 boot configuration preset by Pentek, and should **NOT** be modified by the user.

3.5 MPC745X Resources Operation (continued)

3.5.13 Board ID EEPROM

The 1–kByte serial ID EEPROM contains Model 4205 board identification and configuration information. Refer to Appendix A for the format and contents of this EEPROM. This EEPROM can be accessed through the GT–64260 TWSI interface, Section 3.6.7.



The Board ID EEPROM is shipped with board identification preset by Pentek, and should **NOT** be modified by the user.

During boot operation, the contents of this EEPROM are copied into SDRAM, as described in Section 8.3.1.

3.5.14 Real–Time Clock

The Dallas Semiconductor DS1338 Real–Time Clock provides a full binary–coded decimal (BCD) clock/calendar for your applications, in either 24–hour or 12–hour format. The DS1338 has a built–in power–sense circuit that detects power failures and automatically switches to a battery backup. The DS1338 can be accessed through the GT–64260 TWSI interface, Section 3.6.7.

At boot time, the RedBoot Monitor provides a user option to reprogram the Real–Time Clock, as described in Chapter 9.

3.5.15 Programmable Clock Generator

The Cypress CY22395 Programmable Clock Generator generates the on–board clocks. The CY22395 has three phase–locked loops, which, when combined with the reference clock input, allow up to four independent frequencies to be output. The CY22395 can be accessed through the GT–64260 TWSI interface, Section 3.6.7.



The CY22395 is programmed with default clock values by the Pentek boot firmware, and should **NOT** be modified by the user.

3.6 GT–64260 Controller

The Marvell GT–64260 acts as the system controller for the MPC745X processor, and is the PCI bus controller for MPC745X access to the two PCI buses: the Primary PCI Bus and the Secondary PCI Bus.

- The Primary PCI Bus provides the interface between the processor and the VMEbus (Tundra Universe II), the VIM module control and status registers, and the optional primary RACE++ interface (Mercury Computer Systems PXB++ PCI Bridge chip).

- The Secondary PCI Bus provides the interface between the processor and the VIM data interfaces, the PMC site, the optional second RACE++ interface, and the optional serial mezzanine site.

The following subsections (Section 3.6.1 through Section 3.6.8) identify the internal registers and major functional units of the GT–64260 that are used in the Model 4205. Refer to the Marvell GT–64260 Datasheet for description and programming of these functions. Pentek board support software routines (see Section 1.12) are available to control the operation of these GT–64260 functions.

3.6.1 GT–64260 Internal Registers

Internal GT–64260 registers set up the communications and data paths between the MPC745X and the SDRAM, FLASH memory, VIM interfaces, and other Model 4205 global resources. These registers are described in the Marvell GT–64260 Datasheet for each of the functional units identified in the following subsections. The MPC745X can access the GT–64260 internal registers starting at memory address **0xF100 0000**.



1) The Pentek default boot sequence (Section 8.3.2) configures these registers for the memory maps and operation shown in this chapter—except where noted, you do not need to access the GT–64260 internal registers.

2) The GT–64260 internal registers are programmed in Little Endian. If you access a register from an MPC745X, PCI device, or external VMEbus master that is set for Big Endian, you must perform byte swapping.

To access a GT–64260 internal register, first determine the desired register's offset from the Marvell GT–64260 Datasheet, and then add that offset to the MPC745X starting memory address **0xF100 0000**.

3.6 GT−64260 Controller (continued)

3.6.2 Messaging Unit (MU)

The GT−64260 Messaging Unit (MU) allows the MPC745X to communicate with PCI resources through command passing and interrupt generation, using Message Registers, Doorbell Registers, and Circular Queues. The MU is implemented in both PCI interfaces, and allows for messaging between the MPC745X and PCI, and inter−PCI interfaces messaging. To access the MU reg−isters from the MPC745X, use the GT−64260 'internal register space' offsets listed in the GT−64260 Datasheet.

3.6.3 IDMA Controller

The GT−64260 IDMA Controller can move data between any source and any destination, such as SDRAM, PCI Buses, or CPU bus. In the Model 4205 Demand Mode DMA operation, IDMA Channels 0 through 3 are used for reads from each Incoming VIM FIFO, and IDMA Channels 4 through 7 are used for writes to each Outgoing VIM FIFO (see Section 6.6.3).

3.6.4 General Purpose Port (GPP)

The GT−64260 contains a 32−bit General Purpose Port (GPP). Each GPP pin can be assigned to act as a general purpose input or output pin, and can be used to register external interrupts (when the pin is assigned as an input). The GPP is multiplexed on the Multi−Purpose Pins (MPP) pins (see Section 3.6.8).

3.6.5 Communication Unit

The GT−64260 Communication Unit supports input and output of RS−232 and 10/100 BaseT Ethernet LAN signals using the Communication Unit, Multi Protocol Serial Controller (MPSC), and Fast Ethernet Unit Registers. These signals are routed to a 15−pin front panel connector (see Section 2.5.6). Two MPSC serial ports are provided: MPSC0 is connected to the RS−232 0 pins and MPSC1 is connected to the RS−232 1 pins.

3.6.6 Interrupt Controller

The GT−64260 Interrupt Controller routes internal interrupt requests (and optionally external interrupt requests) to both the MPC745X and the PCI bus. Model 4205 interrupt operation is described in Chapter 7.

3.6.7 TWSI Interface

The GT−64260 Two−wire Serial Interface (TWSI) can initiate read and write transactions using the TWSI Registers. The resources that can be accessed using the GT−64260 TWSI interface are identified in Section 3.2.3.

3.6 GT–64260 Controller (continued)

3.6.8 Multi–Purpose Pins (MPP)

The 32 Multi–Purpose Pins (MPP) on the GT–64260 are used for peripheral interrupt and DMA I/O functions for the MPC745X. The Model 4205 default boot sequence configures these pins as shown in the following table.

Table 3–10: GT–64260 MPP Pin Default Functions

Pin	Function	Description	Int. Active	Information
MPP0	DMAReq[0]	DMA Request 0 (VIM A Incoming FIFO)	–	Section 3.6.8.1
MPP1	GPP[1]	VIM A/B FPGA PLX 9656 Interrupt	High	Section 3.6.8.2
MPP2	DMAReq[1]	DMA Request 1 (VIM B Incoming FIFO)	–	Section 3.6.8.1
MPP3	GPP[3]	VIM C/D FPGA PLX 9656 Interrupt	High	Section 3.6.8.2
MPP4	DMAReq[2]	DMA Request 2 (VIM C Incoming FIFO)	–	Section 3.6.8.1
MPP5	GPP[5]	Universe II (VME) Interrupt	High	Section 3.6.8.3
MPP6	DMAReq[3]	DMA Request 3 (VIM D Incoming FIFO)	–	Section 3.6.8.1
MPP7	GPP[7]	Serial Mezzanine Module Interrupt	High	Section 3.6.8.4
MPP8	DMAReq[4]	DMA Request 4 (VIM A Outgoing FIFO)	–	Section 3.6.8.1
MPP9	GPP[9]	PXB++ (RACEway) Interrupt (Option 005 or 010)	High	Section 3.6.8.5
MPP10	DMAReq[5]	DMA Request 5 (VIM B Outgoing FIFO)	–	Section 3.6.8.1
MPP11	GPP[11]	PMC Module Interrupt	High	Section 3.6.8.6
MPP12	DMAReq[6]	DMA Request 6 (VIM C Outgoing FIFO)	–	Section 3.6.8.1
MPP13	GPP[13]	Front Panel Test Point Interrupt, **TP3**	High	Section 3.6.8.7
MPP14	DMAReq[7]	DMA Request 7 (VIM D Outgoing FIFO)	–	Section 3.6.8.1
MPP15	InitAct	GT–64260 Initialization Active	–	Section 3.6.8.8
MPP16	GPP[16]	VIM A Incoming FIFO Not Almost Empty Interrupt	High	Section 3.6.8.9
MPP17	GPP[17]	VIM B Incoming FIFO Not Almost Empty Interrupt	High	
MPP18	GPP[18]	VIM C Incoming FIFO Not Almost Empty Interrupt	High	
MPP19	GPP[19]	VIM D Incoming FIFO Not Almost Empty Interrupt	High	
MPP20	GPP[20]	VIM A Outgoing FIFO Almost Empty Interrupt	High	Section 3.6.8.10
MPP21	GPP[21]	VIM B Outgoing FIFO Almost Empty Interrupt	High	
MPP22	GPP[22]	VIM C Outgoing FIFO Almost Empty Interrupt	High	
MPP23	GPP[23]	VIM D Outgoing FIFO Almost Empty Interrupt	High	
MPP24	GPP[24]	VIM A Incoming FIFO Almost Full Interrupt	High	Section 3.6.8.11
MPP25	GPP[25]	VIM B Incoming FIFO Almost Full Interrupt	High	
MPP26	GPP[26]	VIM C Incoming FIFO Almost Full Interrupt	High	
MPP27	GPP[27]	VIM D Incoming FIFO Almost Full Interrupt	High	
MPP28	GPP[28]	VIM Incoming FIFO Not Empty Interrupt	High	Section 3.6.8.12
MPP29	GPP[29]	VIM Outgoing FIFO Empty Interrupt	High	Section 3.6.8.13
MPP30	GPP[30]	VIM Incoming/Outgoing FIFO Full Interrupt	High	Section 3.6.8.14
MPP31	GPP[31]	VIM Module DBINT/MBIN/XMIT/RCV Interrupt	High	Section 3.6.8.15

3.6 GT–64260 Controller (continued)

3.6.8 Multi–Purpose Pins (continued)

Note that many of the MPP pins are configured as GT–64260 General Purpose Port (GPP) functions (see Section 3.6.4). Each GPP pin can act as a general purpose input or output pin and can be used to register external interrupts. When configured as interrupts, GPP pin interrupts are edge triggered.

The following subsections provide brief descriptions of the default use of the MPP pins. Pentek board support software routines (see Section 1.12) are available to control the operation of these MPP functions.



If you use Pentek's Model 4996 VxWorks BSP, some of these resources are used by the BSP software and MPC745X access to their capabilities is limited. Refer to the VxWorks BSP documentation, Pentek manual 803.42050, for description of these limitations.

3.6.8.1 DMA Requests MPP0/2/4/6/8/10/12/14

These eight MPP pins provide VIM Incoming FIFO and Outgoing FIFO signals to the GT–64260 IDMA Controller. These signals are based on the FIFO boundary flag conditions, and are used by the GT–64260 to control Demand Mode DMA operation of each FIFO. See Section 6.6.3 for description of Demand Mode DMA operation.

Note that the VIM BIFO boundary flags are also routed directly to the MPC745X as interrupts (see Sections 3.6.8.10 and 3.6.8.11).

3.6.8.2 VIM FPGA PLX 9656 Interrupts MPP1/3

MPP pin 1 routes the PCI interrupt signal INTA from VIM A/B FPGA PLX 9656 to the MPC745X; MPP pin 3 routes the PCI interrupt signal INTA from VIM C/D FPGA PLX 9656 to the MPC745X (see Section 7.9). See Figure 7–3 in Section 7.9 for illustration of the routing of these interrupts to MPP pins 1 and 3.

3.6.8.3 VMEbus Interrupt MPP5

This MPP pin routes the interrupt signal LINT4 from the Universe II to the MPC745X (see Section 7.8.1).

3.6.8.4 Serial Mezzanine Module Interrupt MPP7

This MPP pin routes the PCI interrupt signal INTA from the Serial Mezzanine Module to the MPC745X (see Section 7.13).

3.6 GT–64260 Controller (continued)

3.6.8 Multi–Purpose Pins (continued)

3.6.8.5 PXB Interrupt (Option 005 or 010) MPP9

This MPP pin routes interrupt signal INTA from the PXB++ PCI Bridge (Option 005 or 010) to the MPC745X (see Section 7.12).

3.6.8.6 PMC Module Interrupt MPP11

This MPP pin routes the four PMC interrupt status bits from the PMC Interrupt Status Register (see Section 3.8.12) to the MPC745X (see Section 7.11). There are four PMC interrupts, INTA through INTD. Pin MPP11 is an OR'ed result of these four status bits, as enabled by the PMC Interrupt Mask Register (see Section 3.8.3).

3.6.8.7 Front Panel Test Point Interrupt MPP13

Pin MPP13 routes the input signal from the front panel test point, **TP3**, as an interrupt to the MPC745X (see Section 7.4.1). This test point is located on the front edge of the PCB (see Section 2.5.9).

3.6.8.8 GT–64260 Initialization Active MPP15

Pin MPP15 provides a signal when the GT–64260 is in its initialization cycle. This signal is for internal use only.

3.6.8.9 VIM Incoming FIFO Not Almost Empty Interrupts MPP16–19

These four MPP pins route the Incoming FIFO Not Almost Empty flag from each node's VIM BIFO to the MPC745X (see Section 7.6).

3.6.8.10 VIM Outgoing FIFO Almost Empty Interrupts MPP20–23

These four MPP pins route the Outgoing FIFO Almost Empty flag from each node's VIM BIFO to the MPC745X (see Section 7.6).

These four FIFO flags are also used for Demand Mode DMA operation of the FIFO (see Section 3.6.8.1).

3.6.8.11 VIM Incoming FIFO Almost Full Interrupts MPP24–27

These four MPP pins route the Incoming FIFO Almost Full flag from each node's VIM BIFO to the MPC745X (see Section 7.6).

These four FIFO flags are also used for Demand Mode DMA operation of the FIFO (see Section 3.6.8.1).

3.6 GT–64260 Controller (continued)

3.6.8 Multi–Purpose Pins (continued)

3.6.8.12 VIM Incoming FIFO Not Empty Interrupt MPP28

This MPP pin routes the four VIM Incoming FIFO Not Empty status bits from the VIM BIFO Empty Interrupt Status Register (see Section 3.8.11) to the MPC745X (see Section 7.6).

There are eight status bits in this register, two for each VIM FIFO. Pin MPP28 is an OR'ed result of only the four Incoming FIFO Not Empty status bits for each VIM, as enabled by the VIM BIFO Empty Interrupt Mask Register (see Section 3.8.2).

3.6.8.13 VIM Outgoing FIFO Empty Interrupt MPP29

This MPP pin routes the four VIM Outgoing FIFO Empty bits from the VIM BIFO Empty Interrupt Status Register (see Section 3.8.11) to the MPC745X (see Section 7.6).

There are eight status bits in this register, two for each VIM FIFO. Pin MPP29 is an OR'ed result of only the four Outgoing FIFO Empty status bits for each VIM, as enabled by the VIM BIFO Empty Interrupt Mask Register (see Section 3.8.2).

3.6.8.14 VIM Incoming/Outgoing FIFO Full Interrupt MPP30

This MPP pin routes the eight VIM FIFO Full status bits from the VIM BIFO Full Interrupt Status Register (see Section 3.8.10) to the MPC745X (see Section 7.6).

There are eight bits in this status register, one for each VIM BIFO's Incoming FIFO Full flag and Outgoing FIFO Full flag. Pin MPP30 is an OR'ed result of all eight interrupt status bits, as enabled by the VIM BIFO Full Interrupt Mask Register (see Section 3.8.1).

3.6.8.15 VIM Module DBINT/MBIN/XMIT/RCV Interrupt MPP31

This MPP pin routes the eight bits from the VIM Module Interrupt Status Register see (Section 3.8.13) and the eight status bits from the VIM Serial Port Interrupt Status Register (see Section 3.8.14) to the MPC745X (see Section 7.7).

Pin MPP31 is an OR'ed result of all 16 status bits from both registers. The eight VIM Module Interrupt Status bits must be enabled by the VIM Module Interrupt Mask Register (see Section 3.8.4). The eight VIM Serial Port Interrupt Status bits must be enabled by the FPGA Interrupt Mask Registers (see Section 6.9.6).

3.7 Board Control/Status Registers

The MPC745X has access to a set of 32−bit registers that control several resources on the Model 4205 board. The following table presents a memory map of these registers. The referenced subsections following this table provide information about the use and programming of each register.

Table 3−11: Board Control/Status Register Addresses

MPC745X Memory Address	Register	Access	Register Information
0x4000 0000	LED Control Register	R/W	Section 3.7.1
0x4000 0004	Reset Register	R/W	Section 3.7.2
0x4000 0008	Fault Register	R/W	Section 3.7.3
0x4000 000C	BIST Register	R/W	Section 3.7.4
0x4000 0010	Module Status Register	R.O.	Section 3.7.5
0x4000 0014	Jumper Status Register	R.O.	Section 3.7.6
0x4000 0018	FPGA Jumper Status Register	R.O.	Section 3.7.7
0x4000 001C	Watchdog Timer Control Register	R/W	Section 3.7.8

Pentek board support software routines (see Section 1.12) are available to control the operation of these register functions.

3.7 Board Control/Status Registers (continued)

3.7.1 LED Control Register

This register contains control bits for the Model 4205 front panel test points and LEDs.

The following table shows the format of this register. The subsections below this table describe each register bit.

Table 3–12: LED Control Register
R/W @ **0x4000 0000**

	D31 – D5	D4	D3	D2	D1	D0
Bit Name	Reserved	TRIGGER	LED3	LED2	LED1	LED0
Function	Write zeros, Mask read	0 = Logic '0' 1 = Logic '1'	0 = LED off 1 = LED on			
All bits default to the logic '0' state at power up and reset.						

3.7.1.1 TRIGGER Bit D4

This bit sets a logic signal to test point **TP0** on the front edge of the PCB (see Section 2.5.9). Set this bit to logic '1' to set **TP0** to logic '1'. Clear the bit to logic '0' (its default state) to set **TP0** to logic '0'.

3.7.1.2 LED3/2/1/0 Bits D5/D4/D3/D2

These four bits control the four processor LEDs on the front panel (see Section 2.5.4). Set a bit to logic '1' to illuminate the respective LED. Clear the bit to logic '0' (its default state) to turn the LED off. These LEDs also indicate the results of the power–up self–tests.

3.7 Board Control/Status Registers (continued)

3.7.2 Reset Register

The Reset Register contains reset bits for various elements on the board.

The following table shows this register's bit layout. The subsections following this table describe each register bit.

Table 3–13: Reset Register
R/W @ **0x4000 0004**

	D31 – D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	PCI_1	PCI_0	VIM D	VIM C	VIM B	VIM A	BOARD
Function	Write 0s, Mask read	0 = Normal 1 = Reset						
All bits default to the logic '0' state at power up and reset.								

3.7.2.1 PCI_1/PCI_0 Bits D6/D5

These bits are the resets for the Primary PCI Bus (PCI_0) and the Secondary PCI Bus (PCI_1). This reset will create a reset to only the devices on the respective PCI bus and the PCI bus arbiter. Set a bit to logic '1' to hold the PCI bus in its reset state. Clear a bit to logic '0' (its default state) to return the bus to normal operation.

3.7.2.2 VIM D/C/B/A Bits D4/D3/D2/D1

These bits are the board resets for each VIM module (DBRST) installed on the respective VIM interfaces. Set a bit to logic '1' to hold the VIM module in its reset state. Clear a bit to logic '0' (its default state) to return the module to normal operation.

These bits do not reset the VIM BIFO—see the VIM Interface BIFO Control Register, Section 6.4.3 for the VIM BIFO resets.

3.7.2.3 BOARD Bit D0

This bit is the board reset. This resets all board resources, including the four VIM modules (DBRSTA/B/C/D, same as bits D4 through D1 above) and both PCI buses (PCI RST, same as bits D6 and D5 above). Set this bit to logic '1' to hold the board in its reset state. Clear this bit to logic '0' (its default state) to return the board to normal operation.

3.7 Board Control/Status Registers (continued)

3.7.2 Reset Register (continued)

The following diagram shows the Reset Register actions on the board. The 'Bit Dn' labels in this diagram refer to the Reset Register bits, Table 3–13. The resets from board power on, front panel Reset button, VMEbus Sys Reset, and the Watchdog Timer Register (Section 3.7.8) are also shown.

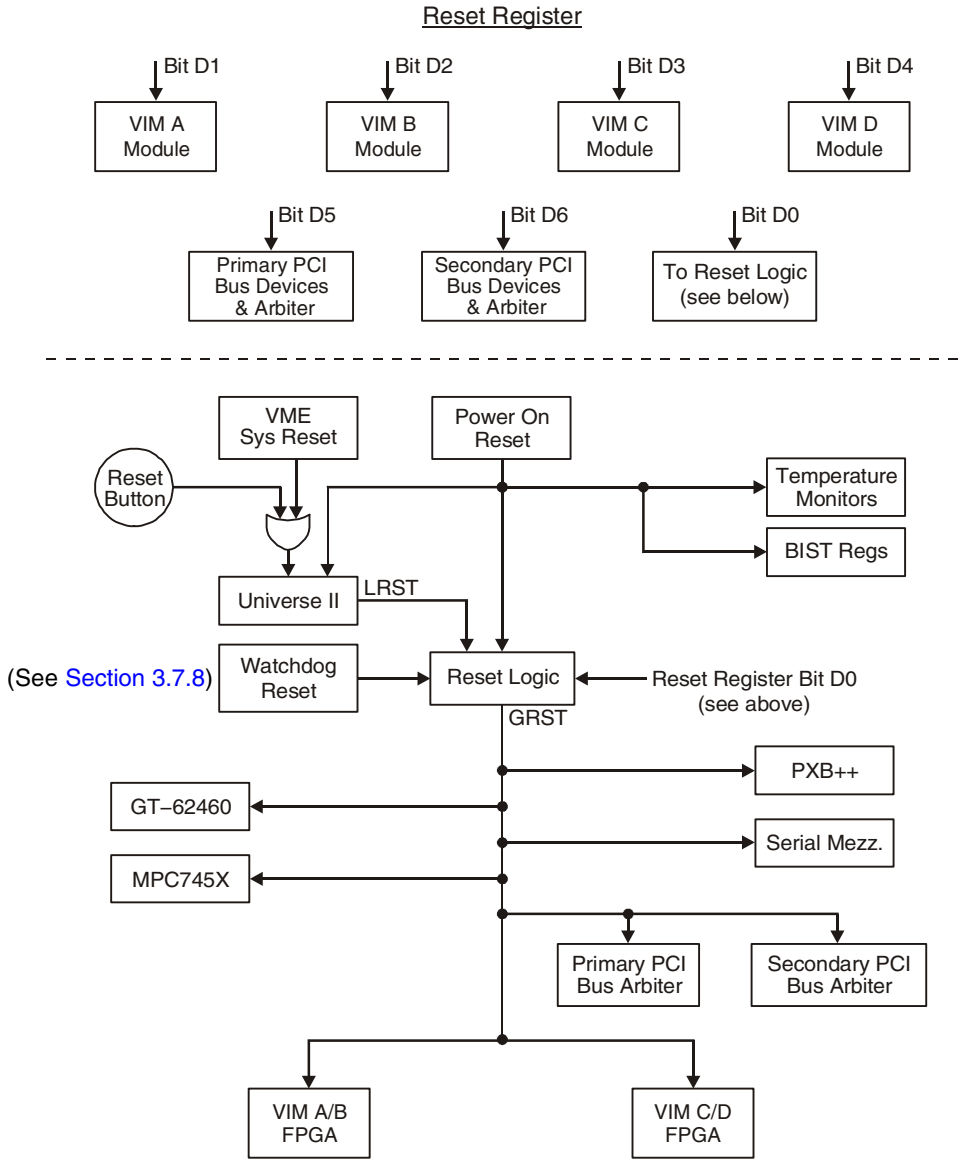


Figure 3–2: Model 4205 Resets

3.7 Board Control/Status Registers (continued)

3.7.3 Fault Register

The Fault Register is used by the built–in self–test (BIST) function to indicate a fault condition during power–up or reset (see Section 8.3). There is only one bit in this register, which controls the red **Fault** LED on the Model 4205 front panel (see Section 2.5.5).

The following table shows this register's bit layout. The paragraphs following this table describe the register bit.

Table 3–14: Fault Register
R/W @ **0x4000 0008**

	D31 – D1	D0
Bit Name	Reserved	FAULT LED
Function	Write zeros, Mask read	0 = Off 1 = On
See paragraph below for state of register at power up and reset.		

Set this bit to logic '1' to illuminate the **Fault** LED. Clear this bit to logic '0' to turn the **Fault** LED off.

At power–up or reset, the FAULT LED bit initializes at logic '1'. At the start of BIST (Section 8.3), this bit is cleared to logic '0' by the BIST routine. If a failure is detected during BIST, the bit is set to logic '1' to illuminate the **Fault** LED.

3.7.4 BIST Register

This register contains status bits for the board's built–in self–tests (BIST). These bits indicate the results of the self–test on parts of the Model 4205. These bits are used internally only for the board's start–up self–tests, and data written to or read from this register is meaningless.

Table 3–15: BIST Register
R/W @ **0x4000 000C**

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	BIST7	BIST6	BIST5	BIST4	BIST3	BIST2	BIST1	BIST0
Function	Write zeros, Mask read	BIST Results							BIST Completed
All bits default to the logic '0' state at power up and reset.									

3.7 Board Control/Status Registers (continued)

3.7.5 Module Status Register

The Module Status Register contains read–only status bits that report the presence of each VIM module and a PMC module on the Model 4205.

The following table shows this register's bit layout. The subsections following this table describe each status bit.

Table 3–16: Module Status Register
R.O. @ **0x4000 0010**

	D31 – D5	D4	D3	D2	D1	D0
Bit Name	Reserved	PMC PRESENT	VIM D PRESENT	VIM C PRESENT	VIM B PRESENT	VIM A PRESENT
Function	Mask read	0 = Module not present 1 = Module present				
All bits default to the presence state of the respective module at power up and reset.						

3.7.5.1 PMC PRESENT Bit D4

This bit indicates whether or not a PMC module is present on the board's PMC site. When this bit is read as logic '1', a PMC module is installed on the board. When read as logic '0', no PMC module is present.

3.7.5.2 VIM D/C/B/A PRESENT Bits D3/D2/D1/D0

This bit indicates whether or not a VIM module is present on the respective VIM interface. When a bit is read as logic '1', a corresponding VIM module is installed on the board. When read as logic '0', no VIM module is present on this interface.

3.7 Board Control/Status Registers (continued)

3.7.6 Jumper Status Register

This register contains read–only status bits that indicate the state of various PCB jumpers.

The following table shows this register's bit layout. The subsections following this table describe each status bit.

Table 3–17: Jumper Status Register
R.O. @ **0x4000 0014**

	D31 – D8							
Bit Name	Reserved							
	D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Jumper	**JB10**	**JB9**	–		**JB8**	**JB4**	**JB7, 3–4**	**JB7, 1–2**
Bit Name	GENERAL PURPOSE	FLASH RESTORE	CPU SPEED		SEC PCI BUS SPEED	FLASH WRITE ENABLE	SELF–TEST MODE	
Function	0 = Jumper On * 1 = Jumper Off	0 = Do Not Restore * 1 = Restore Boot Param.	00 = 600 MHz 01 = 667 MHz 10 = 1 GHz 11 = 1.3 GHz		0 = 33 MHz 1 = 66 MHz *	0 = Write Protected 1 = Write Enabled *	00 = BIST at power on * 01 = Skip BIST 10 = Run BIST continuous 11 = Reserved	

* Factory default state
All bits default to the jumper state at power up and reset.

3.7.6.1 GENERAL PURPOSE Bit D7

This bit indicates the status of the general–purpose jumper **JB10**, Section 2.2.9. When this bit is read as logic '0', the jumper is installed on the PCB (the factory default setting). When read as logic '1', the jumper is not installed.

3.7.6.2 FLASH RESTORE Bit D6

This bit indicates the status of the FLASH Boot Parameter Restore jumper **JB9**, Section 2.2.8, which determines whether or not to restore the default boot parameters in FLASH during power–up of the board. When read as logic '0', the jumper is not installed (the factory default setting), and the boot sequence will not restore the default parameters. When read as logic '1', the jumper is installed, and the boot will restore the default parameters in FLASH.

3.7 Board Control/Status Registers (continued)

3.7.6 Jumper Status Register (continued)

3.7.6.3 CPU SPEED — Bits D5 & D4

These two bits indicate the processor clock speed (factory set via PCB wiring, depending on the CPU option you receive), and are provided to accommodate optional higher speed PowerPC proces–sors. The settings for these two bits are:

00 – 600 MHz (MPC7455)
01 – 667 MHz
10 – 1 GHz (MPC7457)
11 – 1.3 GHz

3.7.6.4 SEC PCI BUS SPEED — Bit D3

This bit indicates the status of the Secondary PCI Bus speed jumper **JB8**, Section 2.2.7. When read as logic '0', the jumper is installed, and the Secondary PCI Bus speed is 33 MHz. When read as logic '1', the jumper is not installed (the factory default setting), and the Secondary PCI Bus is clocked to run at 66 MHz.

3.7.6.5 FLASH WRITE ENABLE — Bit D2

This bit indicates the status of the FLASH Memory Write Enable jumper **JB4**, Section 2.2.4. When read as logic '0', the jumper is not installed, and the FLASH is write–protected. When read as logic '1', the jumper is installed (the factory default setting), and the FLASH is write–enabled and can be erased and programmed.

3.7.6.6 SELF–TEST MODE — Bits D1 & D0

These two bits indicate the self–test mode at power–up or reset of the board. Each bit corresponds to the setting of jumpers on PCB jumper block **JB7**, Section 2.2.6, where pins **1–2** correspond to bit D0 and pins **3–4** correspond to bit D1. When read as logic '0', the jumper is installed. When read as logic '1', the jumper is not installed. The settings for these two bits are:

00 – Run BIST at power on (Factory Default)
01 – Skip BIST at power on
10 – Run BIST continuously
11 – Reserved

3.7 Board Control/Status Registers (continued)

3.7.7 FPGA Jumper Status Register

The FPGA Jumper Status Register contains read–only bits that report the status of the FPGA general–purpose jumpers **JB13** and **JB14**, Section 2.2.11.

The following table shows this register's bit layout. The paragraph following this table describes these status bits.

Table 3–18: FPGA Jumper Status Register
R.O. @ **0x4000 0018**

	D31 – D4	D3	D2	D1	D0
Jumper	–	**JB14, 3–4**	**JB14, 1–2**	**JB13, 3–4**	**JB13, 1–2**
Bit Name	Reserved	FPGA_3	FPGA_2	FPGA_1	FPGA_0
Function	Mask read	0 = Jumper On * 1 = Jumper Off			

* Factory default jumper state
All bits default to the jumper state at power up and reset.

Each read–only bit indicates whether the respective jumper is present on the PCB. When read as logic '1', no jumper is installed on the corresponding pin position on the board. When read as logic '0', a jumper is installed (the factory default state).

3.7 Board Control/Status Registers (continued)

3.7.8 Watchdog Timer Control Register

The Watchdog Timer Control Register contains bits that control a watchdog timer on the board. The watchdog timer uses an internal counter and the Dallas DS1338 Real−Time Clock to provide a fail−safe monitoring capability to reset the board in the event that the processor has stopped.

NOTE: This register is available only on Model 4205 boards with PCB assembly revision **'P'** or greater (a stick−on label **'PC.xxxxxxxP'** on the PCB).

The following table shows this register's bit layout. The subsections following this table describes these control bits.

Table 3−19: Watchdog Timer Control Register
R/W @ **0x4000 001C**

	D31 – D8		
Bit Name	Reserved		
	D7	**D6**	**D5 – D0**
Bit Name	RESET ENABLE	TIMER STATUS	COUNTER
Function	0 = Disable 1 = Enable	0 = Clear 1 = Tripped	Timeout counter value (LSB = 250 msec)
All bits default to the logic '0' state at power up and reset.			

3.7.8.1 RESET ENABLE Bit D7

This bit enables the board's watchdog logic to reset the board if the COUNTER value has counted down to zero. Set this bit to logic '1' to enable a board reset when the COUNTER value has counted down to zero. Clear the bit to logic '0' (its default state) to disable the board's watchdog reset.

The reset from the watchdog logic resets all board resources, similar to the BOARD Reset bit D0 of the Reset Register, Section 3.7.2.3. See Figure 3−2 for illustration of the board Reset actions.

3.7 Board Control/Status Registers (continued)

3.7.8 Watchdog Timer Control Register (continued)

3.7.8.2 TIMER STATUS Bit D6

This bit indicates the status of the board's watchdog counter. When read as logic '0', the counter has not counted down to zero. When read as logic '1', the counter has counted down to zero (Tripped), and the watchdog logic will reset the board if the reset is enabled by bit D7, Section 3.7.8.1 above.

When you write any value to this register, the watchdog logic will clear this status bit to zero.

3.7.8.3 COUNTER Bits D5 – D0

These six bits specify the counter timeout value used by the board's watchdog logic. After you write a non–zero value to this field, when this value counts down to zero the TIMER STATUS bit D6, Section 3.7.8.2 above, is set to one and the board is reset if enabled by bit D7, Section 3.7.8.1.

The counter timeout value is a 6–bit integer with bit D0 equal to 250 milliseconds.



For this watchdog counter to operate properly, the Dallas DS1338 RTC must be programmed to provide a square–wave output at 4.096 kHz.

This output is programmed by default at boot time by the RedBoot Monitor.

3.8 Board Interrupt Registers

The MPC745X processor has access to a set of 32−bit interrupt control registers that configure its response to interrupts from board resources. The following table presents a memory map of these registers. The referenced subsections following this table provide information about the use and programming of each register.

Table 3−20: Board Interrupt Register Addresses

MPC745X Memory Address	Register	Access	Register Info.
0x4020 0000	VIM BIFO Full Interrupt Mask Register	R/W	Section 3.8.1
0x4020 0004	VIM BIFO Empty Interrupt Mask Register	R/W	Section 3.8.2
0x4020 0008	PMC Interrupt Mask Register	R/W	Section 3.8.3
0x4020 000C	VIM Module Interrupt Mask Register	R/W	Section 3.8.4
0x4020 0010	VIM BIFO Full Interrupt Flag Register	R/Clr	Section 3.8.5
0x4020 0014	VIM BIFO Empty Interrupt Flag Register	R/Clr	Section 3.8.6
0x4020 0018	PMC Interrupt Flag Register	R/Clr	Section 3.8.7
0x4020 001C	VIM Module Interrupt Flag Register	R/Clr	Section 3.8.8
0x4030 0000	VIM Serial Port Interrupt Flag Register	R/Clr	Section 3.8.9
0x4030 0004	VIM BIFO Full Interrupt Status Register	R.O.	Section 3.8.10
0x4030 0008	VIM BIFO Empty Interrupt Status Register	R.O.	Section 3.8.11
0x4030 000C	PMC Interrupt Status Register	R.O.	Section 3.8.12
0x4030 0010	VIM Module Interrupt Status Register	R.O.	Section 3.8.13
0x4030 0014	VIM Serial Port Interrupt Status Register	R.O.	Section 3.8.14
0x4030 0018	Temperature/Voltage Fault Interrupt Mask Register	R/W	Section 3.8.15

The VIM and PMC Interrupt Registers control several interrupts that can be asserted to the MPC745X using GT−64260 MPP pins (see Section 3.6.8). The Temperature/Voltage Fault Interrupt Mask Register enables the interrupt from the board's temperature/voltage monitors to the MPC745X SMI interrupt input.

NOTE: In the following subsections, the VIM FIFO direction, Incoming or Outgoing, is labeled in reference to the MPC745X processor. That is, 'Incoming' refers to data from the VIM to the MPC745X.

Refer to Chapter 7 for further description of Model 4205 interrupt operation.

Pentek board support software routines (see Section 1.12) are available to control the operation of these interrupt register functions.

3.8 Board Interrupt Registers (continued)

3.8.1 VIM BIFO Full Interrupt Mask Register

This register contains interrupt enable bits that allow each VIM FIFO Full flag to assert an interrupt to the MPC745X processor. Refer to Section 6.6 for information on VIM BIFO flag initialization and operation.

There are two interrupt enable bits in this register for each VIM interface (A, B, C, and D), one for the Incoming FIFO Full flag condition (VIM x IN FULL) and one for the Outgoing FIFO Full flag condition (VIM x OUT FULL).

The following table shows this register's bit layout. The paragraphs following this table describe the use of these bits.

Table 3–21: VIM BIFO Full Interrupt Mask Register
R/W @ **0x4020 0000**

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	VIM D OUT FULL	VIM C OUT FULL	VIM B OUT FULL	VIM A OUT FULL	VIM D IN FULL	VIM C IN FULL	VIM B IN FULL	VIM A IN FULL
Function	Write zeros, Mask read	0 = Disable Interrupt 1 = Enable Interrupt							
All bits default to the logic '0' state at power up and reset.									

Set a mask bit to logic '1' to enable an interrupt from the specified VIM FIFO Full flag condition. When a mask bit is set to logic '1' and the associated VIM FIFO Full status bit transitions to active, an interrupt is routed to the MPC745X using GT–64260 MPP pin 30 (Section 3.6.8.14).

Clear a mask bit to logic '0' to disable the interrupt (its default state).

3.8 Board Interrupt Registers (continued)

3.8.2 VIM BIFO Empty Interrupt Mask Register

This register contains interrupt enable bits that allow each VIM FIFO Empty/ Not Empty flag to assert an interrupt to the MPC745X. See Section 6.6 for information on VIM BIFO flag initialization and operation.

There are two interrupt enable bits in this register for each VIM interface (A, B, C, and D), one for the Incoming FIFO Not Empty flag condition (VIM x IN NOT EMPTY) and one for the Outgoing FIFO Empty flag condition (VIM x OUT EMPTY).

The following table shows this register's bit layout. The paragraphs following this table describe the use of these bits.

Table 3–22: VIM BIFO Empty Interrupt Mask Register
R/W @ **0x4020 0004**

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	VIM D OUT EMPTY	VIM C OUT EMPTY	VIM B OUT EMPTY	VIM A OUT EMPTY	VIM D IN NOT EMPTY	VIM C IN NOT EMPTY	VIM B IN NOT EMPTY	VIM A IN NOT EMPTY
Function	Write zeros, Mask read	0 = Disable Interrupt 1 = Enable Interrupt							
All bits default to the logic '0' state at power up and reset.									

Set a mask bit to logic '1' to enable an interrupt from the specified VIM FIFO Empty/Not Empty flag condition. When a mask bit is set to logic '1' and the associated VIM FIFO status bit transitions to active (Section 3.8.11), an interrupt is routed to the MPC745X using GT–64260 MPP pin 28 (Section 3.6.8.12) for the Incoming FIFOs, or MPP pin 29 (Section 3.6.8.13) for the Outgoing FIFOs.

Clear a mask bit to logic '0' to disable the interrupt (its default state).

3.8 Board Interrupt Registers (continued)

3.8.3 PMC Interrupt Mask Register

This register contains interrupt enable bits that allow the PMC module interrupts to assert an interrupt to the MPC745X.

There are four PMC interrupt enable bits in this register, one bit for each PMC module interrupt INTA through INTD (see Section 7.11). Refer to the Operating Manual supplied with your PMC module for description of the respective interrupt sources. When the Model 4205 Option 010 RACE++ PMC is installed, the module's RACEway controller (PXB++ PCI Bridge chip) provides an interrupt on its PCI INTA interrupt output.

The following table shows this register's bit layout. The paragraphs following this table describe the use of these bits.

Table 3–23: PMC Interrupt Mask Register
R/W @ **0x4020 0008**

	D31 – D4	D3	D2	D1	D0
Bit Name	Reserved	PMC INTD	PMC INTC	PMC INTB	PMC INTA
Function	Write zeros, Mask read	0 = Disable Interrupt 1 = Enable Interrupt			
All bits default to the logic '0' state at power up and reset.					

Set a mask bit to logic '1' to enable an interrupt from the specified PMC interrupt. When a mask bit is set to logic '1' and the associated PMC interrupt transitions to active (Section 3.8.12), an interrupt is routed to the MPC745X using GT–64260 MPP pin 11 (Section 3.6.8.6), and to the Universe II PCI bus LINT1 input (Section 7.8.2).

Clear the bit to logic '0' to disable the interrupt (its default state).

3.8 Board Interrupt Registers (continued)

3.8.4 VIM Module Interrupt Mask Register

This register contains interrupt enable bits that allow VIM module interrupts to assert an interrupt to the MPC745X processor.

There are two interrupt enable bits in this register for each VIM interface (A, B, C, and D), one for the Daughterboard DBINT interrupt (VIM x DBINT) and one for the Mailbox MBIN interrupt (VIM x MBIN). Refer to the Operating Manual supplied with your VIM module for description of the respective interrupt sources.

The following table shows this register's bit layout. The paragraphs following this table describe the use of these bits.

Table 3–24: VIM Module Interrupt Mask Register
R/W @ 0x4020 000C

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	VIM D MBIN	VIM C MBIN	VIM B MBIN	VIM A MBIN	VIM D DBINT	VIM C DBINT	VIM B DBINT	VIM A DBINT
Function	Write zeros, Mask read	0 = Disable Interrupt 1 = Enable Interrupt							
All bits default to the logic '0' state at power up and reset.									

Set a mask bit to logic '1' to enable an interrupt from the specified VIM module interrupt. When a mask bit is set to logic '1' and the associated VIM interrupt transitions to active (Section 3.8.13), an interrupt is routed to the MPC745X using GT–64260 MPP pin 31 (Section 3.6.8.15).

Clear a mask bit to logic '0' to disable the interrupt (its default state).

3.8 Board Interrupt Registers (continued)

3.8.5 VIM BIFO Full Interrupt Flag Register

This register contains latched flag bits for the associated VIM FIFO Full flags. See Section 6.6 for information on VIM BIFO flag initialization and operation.

There are two flag bits in this register for each VIM interface (A, B, C, and D), one for the Incoming FIFO Full flag (VIM x IN FULL) and one for the Outgoing FIFO Full flag (VIM x OUT FULL). Each flag bit is a latched version of the associated VIM FIFO Full flag (see Section 3.8.10), and is set when that flag transitions from inactive (logic '0') to active (logic '1'). Since FIFO flag conditions may be transient, they are latched in this register so that the processor can determine the cause.

The following table shows this register's bit layout. The paragraphs following this table describe the use of these bits.

Table 3–25: VIM BIFO Full Interrupt Flag Register
R/Clr @ **0x4020 0010**

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	VIM D OUT FULL	VIM C OUT FULL	VIM B OUT FULL	VIM A OUT FULL	VIM D IN FULL	VIM C IN FULL	VIM B IN FULL	VIM A IN FULL
Function	Write zeros, Mask read	**Read:** 0 = No VIM FIFO Full condition 1 = VIM FIFO Full latched **Clear:** 0 = Normal operation 1 = Clear latch							

IMPORTANT! When reset, including power–up, the state of this register is unknown. You should clear this register before using it, by writing '1's into all bits.

Each flag bit is a read/clear bit.

Read – When read as logic '1', the associated VIM FIFO Full flag condition has been latched. When read as logic '0', the associated VIM FIFO Full flag is not active. A flag bit remains latched at '1' until you clear it (see **Clear** below).

Clear – Since these bits latch in response to an interrupt, to detect subsequent interrupts, you must clear the latched bits in this register. To clear any bit that is set to logic '1', you must write a '1' to that bit to clear the bit.

3.8 Board Interrupt Registers (continued)

3.8.6 VIM BIFO Empty Interrupt Flag Register

This register contains latched flag bits for the associated VIM FIFO Empty/ Not Empty flags. See Section 6.6 for information on VIM FIFO flag initialization and operation.

There are two flag bits in this register for each VIM interface (A, B, C, and D), one for the Incoming FIFO Not Empty flag (VIM x IN NOT EMPTY) and one for the Outgoing FIFO Empty flag (VIM x OUT EMPTY). Each flag bit is a latched version of the associated VIM FIFO interrupt status bit (Section 3.8.11), and is set when that status bit transitions from inactive (logic '0') to active (logic '1'). Since FIFO flag conditions may be transient, they are latched in this register so that the processor can determine the cause.

The following table shows this register's bit layout. The paragraphs following this table describe the use of these bits.

Table 3–26: VIM BIFO Empty Interrupt Flag Register
R/Clr @ **0x4020 0014**

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	VIM D OUT EMPTY	VIM C OUT EMPTY	VIM B OUT EMPTY	VIM A OUT EMPTY	VIM D IN NOT EMPTY	VIM C IN NOT EMPTY	VIM B IN NOT EMPTY	VIM A IN NOT EMPTY
Function	Write zeros, Mask read	**Read:** 0 = No FIFO Empty condition 1 = VIM FIFO Empty latched **Clear:** 0 = Normal operation 1 = Clear latch				**Read:** 0 = No FIFO Not Empty condition 1 = VIM FIFO Not Empty latched **Clear:** 0 = Normal operation 1 = Clear latch			

IMPORTANT! When reset, including power–up, the state of this register is unknown. You should clear this register before using it, by writing '1's into all bits.

Each flag bit is a read/clear bit.

Read – When read as logic '1', the associated VIM FIFO Empty/Not Empty flag condition has been latched. When read as logic '0', the associated VIM FIFO flag is not active. A flag bit remains latched until you clear it (see **Clear** below).

Clear – Since these bits latch in response to an interrupt, to detect subsequent interrupts, you must clear the latched bits in this register. To clear any bit that is set to logic '1', you must write a '1' to that bit to clear the bit.

3.8 Board Interrupt Registers (continued)

3.8.7 PMC Interrupt Flag Register

This register contains latched flag bits for the PMC module interrupts.

There are four PMC interrupt flag bits in this register, one bit for each PMC module interrupt INTA through INTD (see Section 7.11). Each flag bit is a latched version of the associated PMC interrupt status bit (Section 3.8.12), and is set when that interrupt transitions from inactive (logic '0') to active (logic '1'). Since PMC interrupt conditions may be transient, they are latched in this register so that the processor can determine the cause.

Refer to the Operating Manual supplied with your PMC module for description of the interrupt sources. When the Model 4205 Option 010 RACE++ PMC is installed, the module's RACEway controller (PXB++ PCI Bridge chip) provides an interrupt on its PCI INTA interrupt output.

The following table shows this register's bit layout. The paragraph following this table describes the use of these bits.

Table 3–27: PMC Interrupt Flag Register
R/Clr @ **0x4020 0018**

	D31 – D4	D3	D2	D1	D0
Bit Name	Reserved	PMC INTD	PMC INTC	PMC INTB	PMC INTA
Function	Write zeros, Mask read	**Read:** 0 = No interrupt 1 = PMC interrupt latched **Clear:** 0 = Normal operation 1 = Clear latch			

IMPORTANT! When reset, including power–up, the state of this register is unknown. You should clear this register before using it, by writing '1's into all bits.

Each flag bit is a read/clear bit.

Read – When read as logic '1', the associated PMC interrupt has been latched. When read as logic '0', the associated PMC interrupt is not active. A flag bit remains latched until you clear it (see **Clear** below).

Clear – Since these bits latch in response to an interrupt, to detect subsequent interrupts, you must clear the latched bits. To clear any bit that is set to logic '1', you must write a '1' to that bit to clear the bit.

3.8 Board Interrupt Registers (continued)

3.8.8 VIM Module Interrupt Flag Register

This register contains latched flag bits for the VIM module interrupts.

There are two flag bits in this register for each VIM interface (A, B, C, and D), one for the Daughterboard DBINT interrupt (VIM x DBINT) and one for the Mailbox MBIN interrupt (VIM x MBIN). Each flag bit is a latched version of the associated VIM interrupt status bit (Section 3.8.13), and is set when that interrupt transitions from inactive (logic '0') to active (logic '1'). Since VIM module interrupt conditions may be transient, they are latched in this register so that the processor can determine the cause.

Refer to the Operating Manual supplied with your VIM module for description of the respective interrupt sources.

The following table shows this register's bit layout. The paragraph following this table describes the use of these bits.

Table 3–28: VIM Module Interrupt Flag Register
R/Clr @ **0x4020 001C**

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	VIM D MBIN	VIM C MBIN	VIM B MBIN	VIM A MBIN	VIM D DBINT	VIM C DBINT	VIM B DBINT	VIM A DBINT
Function	Write zeros, Mask read	**Read:** 0 = No interrupt 1 = VIM module interrupt latched **Clear:** 0 = Normal operation 1 = Clear latch							

IMPORTANT! When reset, including power–up, the state of this register is unknown. You should clear this register before using it, by writing '1's into all bits.

Each flag bit is a read/clear bit.

Read – When read as logic '1', the associated VIM interrupt has been latched. When read as logic '0', the associated VIM interrupt is not active. A flag bit remains latched until you clear it (see **Clear** below).

Clear – Since these bits latch in response to an interrupt, to detect subsequent interrupts, you must clear the latched bits in this register. To clear any bit that is set to logic '1', you must write a '1' to that bit to clear the bit.

3.8 Board Interrupt Registers (continued)

3.8.9 VIM Serial Port Interrupt Flag Register

This register contains latched flag bits for the VIM module serial port interrupts. Each VIM interface has one serial port routed to the associated FPGA, which are identified as ports A0, B0, C0, and D0. Serial port data is routed through a 1–MByte FIFO in the FPGA. Refer to the Operating Manual supplied with your VIM module for description of the serial ports.

There are two flag bits in this register for each VIM interface, one for the serial port FIFO transmit interrupt (VIM x XMIT) and one for the serial port FIFO receive interrupt (VIM x RCV). Each flag bit is a latched version of the associated VIM serial port RCV or XMIT FIFO flag conditions from the FPGA Interrupt Status Registers, Section 6.9.8, and is set when that interrupt transitions from inactive (logic '0') to active (logic '1'). Since VIM serial port interrupt conditions may be transient, they are latched in this register so that the processor can determine the cause.

The following table shows this register's bit layout. The paragraph following this table describes the use of these bits.

Table 3–29: VIM Serial Port Interrupt Flag Register
R/Clr @ **0x4030 0000**

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	VIM D RCV	VIM D XMIT	VIM C RCV	VIM C XMIT	VIM B RCV	VIM B XMIT	VIM A RCV	VIM A XMIT
Function	Write zeros, Mask read	**Read:** 0 = No interrupt 1 = VIM serial port interrupt latched **Clear:** 0 = Normal operation 1 = Clear latch							
IMPORTANT! When reset, including power–up, the state of this register is unknown. You should clear this register before using it, by writing '1's into all bits.									

Each flag bit is a read/clear bit.

Read – When read as logic '1', the associated VIM serial port interrupt has been latched. When read as logic '0', the associated VIM serial port interrupt is not active. A flag bit remains latched until you clear it (see **Clear** below).

Clear – Since these bits latch in response to an interrupt, to detect subsequent interrupts, you must clear the latched bits in this register. To clear any bit that is set to logic '1', you must write a '1' to that bit to clear the bit.

Refer to Section 6.7 for further information of VIM serial port operation.

3.8 Board Interrupt Registers (continued)

3.8.10 VIM BIFO Full Interrupt Status Register

This register contains status bits for the associated VIM FIFO Full flags. See Section 6.6 for information on VIM BIFO flag initialization and operation.

There are two status bits in this register for each VIM interface (A, B, C, and D), one for the Incoming FIFO Full status (VIM x IN FULL) and one for the Out–going FIFO Full status (VIM x OUT FULL). Each status bit is a buffered version of the associated VIM FIFO Full flag condition, and may be monitored by soft–ware to verify that the flag condition has been properly cleared.

The following table shows this register's bit layout. The paragraph following this table describes the use of these bits.

Table 3–30: VIM BIFO Full Interrupt Status Register
R.O. @ **0x4030 0004**

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	VIM D OUT FULL	VIM C OUT FULL	VIM B OUT FULL	VIM A OUT FULL	VIM D IN FULL	VIM C IN FULL	VIM B IN FULL	VIM A IN FULL
Function	Mask read	0 = FIFO Full flag not active 1 = FIFO Full flag active (FIFO is full)							
All bits default to the logic '0' state at power up and reset.									

Each status bit is read only. When read as logic '1', the associated FIFO Full flag condition is active (true). When read as logic '0', the associated FIFO Full flag condition is inactive.

When a status bit transitions to logic '1' and the associated VIM BIFO Full Interrupt Mask Register bit is enabled (Section 3.8.1), an interrupt is routed to the MPC745X using GT–64260 MPP pin 30 (Section 3.6.8.14).

3.8 Board Interrupt Registers (continued)

3.8.11 VIM BIFO Empty Interrupt Status Register

This register contains status bits for the associated VIM FIFO Empty/Not Empty flags. See Section 6.6 for information on VIM BIFO flag initialization and operation.

There are two status bits in this register for each VIM interface (A, B, C, and D), one for the Incoming FIFO Not Empty status (VIM x IN NOT EMPTY) and one for the Outgoing FIFO Empty status (VIM x OUT EMPTY). Each status bit is a buffered version of the associated VIM FIFO Empty/Not Empty flag condition, and may be monitored by software to verify that the flag condition has been properly cleared.

The following table shows this register's bit layout. The paragraph following this table describes the use of these bits.

Table 3–31: VIM BIFO Empty Interrupt Status Register
R.O. @ **0x4030 0008**

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	VIM D OUT EMPTY	VIM C OUT EMPTY	VIM B OUT EMPTY	VIM A OUT EMPTY	VIM D IN NOT EMPTY	VIM C IN NOT EMPTY	VIM B IN NOT EMPTY	VIM A IN NOT EMPTY
Function	Mask read	0 = FIFO Empty flag not active 1 = FIFO Empty flag active (FIFO is empty)				0 = FIFO flag not active 1 = FIFO Not Empty flag active (FIFO is not empty)			
All bits default to the logic '0' state at power up and reset.									

Each status bit is read only. When read as logic '1', the associated FIFO Empty/Not Empty flag condition is active (true)—for bits D7 through D4, this indicates that the associated VIM Outgoing FIFO is empty; for bits D3 through D0, this indicates that the associated VIM Incoming FIFO is not empty. When read as logic '0', the associated FIFO flag condition is inactive.

When a status bit transitions to logic '1' and the associated VIM BIFO Empty Interrupt Mask Register bit is enabled (Section 3.8.2), an interrupt is routed to the MPC745X using GT–64260 MPP pin 28 (Section 3.6.8.12) for the Incoming FIFOs, or MPP pin 29 (Section 3.6.8.13) for the Outgoing FIFOs.

3.8 Board Interrupt Registers (continued)

3.8.12 PMC Interrupt Status Register

This register contains status bits for the PMC module interrupts.

There are four PMC interrupt status bits in this register, one bit for each PMC module interrupt INTA through INTD (see Section 7.11). Each status bit is a buffered version of the associated PMC interrupt, and may be monitored by software to verify that the interrupting condition has been properly cleared.

Refer to the Operating Manual supplied with your PMC module for description of the interrupt sources. When the Model 4205 Option 010 RACE++ PMC is installed, the module's RACEway controller (PXB++ PCI Bridge chip) provides an interrupt on its PCI INTA interrupt output.

The following table shows this register's bit layout. The paragraph following this table describes the use of these bits.

Table 3–32: PMC Interrupt Status Register
R.O. @ **0x4030 000C**

	D31 – D4	D3	D2	D1	D0
Bit Name	Reserved	PMC INTD	PMC INTC	PMC INTB	PMC INTA
Function	Mask read	0 = Interrupt not asserted 1 = PMC interrupt asserted			
All bits default to the logic '0' state at power up and reset.					

Each status bit is read only. When read as logic '1', the associated PMC interrupt is active (asserted). When read as logic '0', the associated PMC interrupt is inactive.

When a status bit transitions to logic '1' and the associated PMC Interrupt Mask Register bit is enabled (Section 3.8.3), an interrupt is routed to the MPC745X using GT–64260 MPP pin 11 (Section 3.6.8.6), and to the Universe II PCI bus LINT1 input (Section 7.8.2).

3.8 Board Interrupt Registers (continued)

3.8.13 VIM Module Interrupt Status Register

This register contains status bits for the VIM module interrupts. Refer to the Operating Manual supplied with your VIM module for description of the respective interrupt sources.

There are two interrupt status bits in this register for each VIM interface (A, B, C, and D), one for the Daughterboard DBINT interrupt (VIM x DBINT) and one for the Mailbox MBIN interrupt (VIM x MBIN). Each status bit is a buffered version of the associated VIM module interrupt, and may be monitored by software to verify that the interrupting condition has been properly cleared.

The following table shows this register's bit layout. The paragraph following this table describes the use of these bits.

Table 3–33: VIM Module Interrupt Status Register
R.O. @ **0x4030 0010**

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	VIM D MBIN	VIM C MBIN	VIM B MBIN	VIM A MBIN	VIM D DBINT	VIM C DBINT	VIM B DBINT	VIM A DBINT
Function	Mask read	0 = Interrupt not asserted 1 = VIM module interrupt asserted							
All bits default to the logic '0' state at power up and reset.									

Each status bit is read only. When read as logic '1', the associated VIM module interrupt is active (asserted). When read as logic '0', the associated VIM interrupt is inactive.

When a status bit transitions to logic '1' and the associated VIM Module Interrupt Mask Register bit is enabled (Section 3.8.4), an interrupt is routed to the MPC745X using GT–64260 MPP pin 31 (Section 3.6.8.15).

Refer to Section 6.6 for further information of VIM BIFO operation.

3.8 Board Interrupt Registers (continued)

3.8.14 VIM Serial Port Interrupt Status Register

This register contains status bits for the VIM module serial port interrupts. Each VIM interface has one serial port routed to the associated FPGA, which are identified as ports A0, B0, C0, and D0. (Serial port data is routed through a 1−MByte FIFO in the FPGA.) Refer to the Operating Manual supplied with your VIM module for description of the serial ports.

There are two interrupt status bits in this register for each VIM interface, one for the serial port FIFO transmit interrupt (VIM x XMIT) and one for the serial port FIFO receive interrupt (VIM x RCV). Each status bit is an OR'ed result of the associated serial port RCV or XMIT FIFO flag conditions from the FPGA Interrupt Status Registers, Section 6.9.8, for those bits enabled by the FPGA Interrupt Mask Registers, Section 6.9.6.

The following table shows this register's bit layout. The paragraph following this table describes the use of these bits.

Table 3–34: VIM Serial Port Interrupt Status Register
R.O. @ **0x4030 0014**

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	VIM D RCV	VIM D XMIT	VIM C RCV	VIM C XMIT	VIM B RCV	VIM B XMIT	VIM A RCV	VIM A XMIT
Function	Mask read	0 = Interrupt not asserted 1 = VIM serial port interrupt asserted							
All bits default to the logic '0' state at power up and reset.									

Each status bit is read only. When read as logic '1', the associated VIM serial port interrupt is active (asserted). When read as logic '0', the associated VIM serial port interrupt is not active.

When a status bit transitions to logic '1', an interrupt is routed to the MPC745X using GT−64260 MPP pin 31 (Section 3.6.8.15). Note that this interrupt can be masked only using the FPGA Interrupt Mask Registers, Section 6.9.6.

Refer to Section 6.7 for further information of VIM serial port operation.

3.8 Board Interrupt Registers (continued)

3.8.15 Temperature/Voltage Fault Interrupt Mask Register

This register contains the interrupt enable bit that allows the temperature/voltage interrupt output from the ADM1024 Voltage/Temperature Monitors (see Sections 3.5.11 and 7.10) to assert an interrupt to the MPC745X on its SMI interrupt input.

There is only one bit in this register for the temperature/voltage interrupts. There are two ADM1024s on the Model 4205 and the interrupt output lines (INT) of both ADM1024s are tied together for this interrupt.

The following table shows this register's bit layout. The paragraphs following this table describe the use of this bit.

Table 3–35: Temperature/Voltage Fault Interrupt Mask Register
R/W @ **0x4030 0018**

	D31 – D1	D0
Bit Name	Reserved	TMP/VLT
Function	Write zeros, Mask read	0 = Disable 1 = Enable
All bits default to the logic '0' state at power up and reset.		

Set the mask bit to logic '1' to enable a temperature/voltage interrupt from either ADM1024. When the mask bit is set to logic '1' and either ADM1024 interrupt output line, INT, is active, an interrupt can occur to the MPC745X SMI input.

Clear the bit to logic '0' to disable the interrupt (its default state).

NOTE: When an over–temperature occurs, the ADM1024s also illuminate the red **Temp** LED on the front panel (see Section 2.5.2).

Chapter 4: PCI Bus Memory Maps and Resources

4.1 Overview

The Model 4205 supports two PCI bus structures, the Primary and Secondary PCI Buses. The GT−64260 controller acts as the bridge between the two PCI buses. The Primary PCI Bus provides access to the VIM control and status registers, and to the VMEbus and optional RACEway interfaces. The Secondary PCI Bus provides access to the VIM I/O interfaces, the PMC site, and the optional serial mezzanine site.

This chapter presents the memory maps and operation of board resources available as PCI memory−mapped and I/O resources to a device that masters either PCI bus. In addition, when accessed by an external VMEbus Master, the Universe II provides access to PCI bus resources in PCI Configuration Space (Section 4.2, below).

4.2 PCI Configuration Space

PCI bus devices can be programmed, through PCI Configuration Space or VMEbus, to make the resource available in PCI memory space and/or in PCI I/O space. PCI Configuration Cycles can be generated by accessing a VMEbus Slave image whose Local Address Space field (LAS) is set for Configuration Space. Refer to the Tundra Universe II User Manual for description of PCI Configuration Cycles. See Appendix C for a reference list of the PCI Configuration Space Registers. The following table shows the resources available using PCI Configuration Cycles.

Table 4−1: PCI Configuration Space Addresses

Resource	PCI Address Line	VMEbus Address (VA15:11)
Universe II	12	**00010**
GT−64260	13	**00011**
PLX 9656, VIM Control/Status	14	**00100**
RACEway PXB++ (Option 005 or 010)	15	**00101**
Unused	16	**00110**
Serial Mezzanine Module	17	**00111**
PLX 9656, VIM A/B FPGA	18	**01000**
PLX 9656, VIM C/D FPGA	19	**01001**
PMC Module	20	**01010**

NOTE: This PCI Configuration Space map is provided for reference only. The default boot sequence (Section 8.3.2) configures these PCI devices for the PCI memory and I/O maps shown in this chapter—the user does not need to configure these resources using PCI Configuration Space.

4.3 PCI Bus Memory Map

The following PCI bus memory map shows the resources available to any device that masters the Primary or Secondary PCI Buses, the path to each resource, the PCI memory–mapped address of that resource, and a reference to information on each resource.

Table 4–2: PCI Bus Memory Address Map

PCI Memory Address	Resource	Path	Information
0x0000 0000 – 0x0000 0FFF	GT–64260 Messaging Unit	GT–64260	Section 4.5.4
0x0000 1000 – 0x3FFF FFFF	SDRAM		Section 4.5.1
0x4000 0000 – 0x4000 0FFF	Board Control/Status Registers		Section 4.5.5
0x4000 1000 – 0x401F FFFF	Unused		–
0x4020 0000 – 0x4020 0FFF	Board Interrupt Registers		Section 4.5.6
0x4020 1000 – 0x402F FFFF	Unused		–
0x4030 0000 – 0x4030 0FFF	Board Interrupt Registers		Section 4.5.6
0x4030 1000 – 0x405F FFFF	Unused		–
0x4060 0000 – 0x4060 0FFF	Universe II Registers	Primary PCI Bus	Section 4.5.7
0x4060 1000 – 0x406F FFFF	Unused		–
0x4070 0000 – 0x4070 00FF	PLX 9656 Registers, VIM Control/Status		Section 4.5.9
0x4070 0100 – 0x407F FFFF	Unused		–
0x4080 0000 – 0x408F FFFF	VIM Module Access		Section 4.5.14
0x4090 0000 – 0x409F FFFF	VIM Interface Configuration Registers		Section 4.5.15
0x40A0 0000 – 0x40AF FFFF	RACEway PXB++ Registers (Option 005/010)		Section 4.5.12
0x40B0 0000 – 0x40BF FFFF	PXB Reserved	–	–
0x40C0 0000 – 0x40DF FFFF	Unused	–	–
0x40E0 0000 – 0x40E0 00FF	PLX 9656 Registers, VIM A/B FPGA	Secondary PCI Bus	Section 4.5.9
0x40F0 0100 – 0x40FF FFFF	Unused		–
0x40F0 0000 – 0x40F0 00FF	PLX 9656 Registers, VIM C/D FPGA		Section 4.5.9
0x40F0 0100 – 0x40FF FFFF	Unused		–
0x4100 0000 – 0x41FF FFFF	VIM A/B BIFO/FPGA Access		Section 4.5.16
0x4200 0000 – 0x42FF FFFF	VIM C/D BIFO/FPGA Access		
0x4300 0000 – 0x43FF FFFF	Unused		–
0x4400 0000 – 0x5FFF FFFF	PMC Module Access		Section 4.5.11
	Second RACEway Access (Option 010)		Section 4.5.12
	Option 150/151/152 PMC Module Access		Section 4.5.13
0x6000 0000 – 0x7FFF FFFF	Serial Mezzanine Module Access (Fibre Channel or Gigabit Ethernet)		Section 4.5.10
0x8000 0000 – 0xDFFF FFFF	VME Master Access	Primary PCI Bus	Section 4.5.8
0xE000 0000 – 0xEFFF FFFF	Primary RACEway Access (Option 005/010)		Section 4.5.12
0xF000 0000 – 0xF0FF FFFF	Unused	–	–
0xF100 0000 – 0xF100 FFFF	GT–64260 Internal Registers	GT–64260	Section 4.5.3
0xF101 0000 – 0xFDFF FFFF	Unused		–
0xFE00 0000 – 0xFFFF FFFF	FLASH Memory		Section 4.5.2

4.4 PCI Bus I/O Map

In addition to PCI memory−mapped access to resources shown in Table 4−2, a device that masters either PCI bus can access certain resources using PCI I/O space. The following table shows the PCI I/O−mapped addresses for resources on each PCI bus and a reference to information on each resource.

Table 4−3: PCI Bus I/O Address Map

PCI I/O Address Range	Resource	Path	Information
0x1000 – 0x1FFF	Universe II Registers	Primary PCI Bus	Section 4.5.7
0x2000 – 0x20FF	PLX 9656 Registers, VIM Control/Status		Section 4.5.9
0x8000 – 0x80FF	PLX 9656 Registers, VIM A/B FPGA	Secondary PCI Bus	Section 4.5.9
0x9000 – 0x90FF	PLX 9656 Registers, VIM C/D FPGA		
0xA000 – 0xA0FF	PMC Module *		Section 4.5.11
0xB000 – 0xBFFF	Serial Mezzanine Module *		Section 4.5.10
0xC000 – 0xEFFF	Option 150/151/152 PMC Module		Section 4.5.13

* Module dependent—I/O−mapped addresses may not be present on all PCI devices.

4.5 PCI Bus Resources Operation

The following subsections describe operation of the Model 4205 resources available to a device that masters either PCI bus.

4.5.1 SDRAM

The PCI bus provides access to the 512−MByte SDRAM (1−GByte SDRAM with Option 340), which is used for boot operation at reset or power−up of the board. See Section 8.3 for detailed memory maps of SDRAM use during boot operation. A PCI bus device can access SDRAM starting at PCI memory address **0x0000 1000**.



The first 1 MByte of SDRAM (**0x0000 1000** to **0x000F FFFF**) is reserved for use by the default boot firmware (Section 8.3.1).

Loading user code into this area or accessing data in this area can cause unpredictable behavior.

The first 4 kBytes of SDRAM address space, **0x0000 0000** to **0x0000 1000**, are reserved for accessing the GT−64260 Messaging Unit (see Section 4.5.4).

4.5 PCI Bus Resources Operation (continued)

4.5.2 FLASH Memory

A 32−MByte FLASH memory is provided for nonvolatile storage of initialization, self−test, and boot code. User boot code may be added to FLASH for individual application requirements.

A PCI bus device can access FLASH as a memory−mapped resource starting at PCI memory address **0xFE00 0000**. FLASH memory is write−protected using PCB jumpers. See Section 2.2.4 for these jumper settings.



FLASH is shipped with a default boot program (Section 8.3) and a set of MPC745X firmware functions (Section 8.4) installed by Pentek. If you write−enable FLASH, this default Pentek code is not protected.

Users who tamper with the Pentek boot code do so at their own risk.

4.5.3 GT−64260 Internal Registers

The GT−64260 internal registers set up the communications and data paths between the MPC745X processor and PCI bus resources, including PMC, VIM, RACE++, and VMEbus. See Section 3.6 for description of these registers. A PCI bus device can access the GT−64260 internal registers starting at PCI memory address **0xF100 0000**.



The GT−64260 internal registers are programmed in Little Endian. If you access them from a PCI bus device that is Big Endian, you must perform byte swapping.

4.5.4 GT−64260 Messaging Unit

The GT−64260 Messaging Unit (MU) allows PCI devices to communicate with the MPC745X processor through command passing and interrupt generation, using Message Registers, Doorbell Registers, and Circular Queues. Refer to the Marvell GT−64260 Datasheet for description of the Messaging Unit.

A PCI bus device can access the MU registers from GT−64260 internal register space (Section 4.5.3) starting at PCI memory address **0xF100 0000**, or from SDRAM space (Section 4.5.1) starting at PCI memory address **0x0000 0000**. Note that the address offsets of the MU registers (in the Marvell GT−64260 Datasheet) are different when accessed from these two memory spaces.

4.5 PCI Bus Resources Operation (continued)

4.5.5 Board Control/Status Registers

A PCI bus device can access the board control registers at the PCI memory addresses shown in the following table (see Section 3.7 for more information).

Table 4–4: Board Control/Status Register PCI Bus Addresses

PCI Memory Address	Register	Access	Register Info.
0x4000 0000	LED Control Register	R/W	Section 3.7.1
0x4000 0004	Reset Register	R/W	Section 3.7.2
0x4000 0008	Fault Register	R/W	Section 3.7.3
0x4000 000C	BIST Register	R/W	Section 3.7.4
0x4000 0010	Module Status Register	R.O.	Section 3.7.5
0x4000 0014	Jumper Status Register	R.O.	Section 3.7.6
0x4000 0018	FPGA Jumper Status Register	R.O.	Section 3.7.7

4.5.6 Board Interrupt Registers

A PCI bus device can access the board interrupt registers at the PCI memory addresses shown in the following table (see Section 3.8 for more information).

Table 4–5: Board Interrupt Register PCI Bus Addresses

PCI Memory Address	Register	Access	Register Info.
0x4020 0000	VIM BIFO Full Interrupt Mask Register	R/W	Section 3.8.1
0x4020 0004	VIM BIFO Empty Interrupt Mask Register	R/W	Section 3.8.2
0x4020 0008	PMC Interrupt Mask Register	R/W	Section 3.8.3
0x4020 000C	VIM Module Interrupt Mask Register	R/W	Section 3.8.4
0x4020 0010	VIM BIFO Full Interrupt Flag Register	R/Clr	Section 3.8.5
0x4020 0014	VIM BIFO Empty Interrupt Flag Register	R/Clr	Section 3.8.6
0x4020 0018	PMC Interrupt Flag Register	R/Clr	Section 3.8.7
0x4020 001C	VIM Module Interrupt Flag Register	R/Clr	Section 3.8.8
0x4030 0000	VIM Serial Port Interrupt Flag Register	R/Clr	Section 3.8.9
0x4030 0004	VIM BIFO Full Interrupt Status Register	R.O.	Section 3.8.10
0x4030 0008	VIM BIFO Empty Interrupt Status Register	R.O.	Section 3.8.11
0x4030 000C	PMC Interrupt Status Register	R.O.	Section 3.8.12
0x4030 0010	VIM Module Interrupt Status Register	R.O.	Section 3.8.13
0x4030 0014	VIM Serial Port Interrupt Status Register	R.O.	Section 3.8.14
0x4030 0018	Temp/Volt Fault Interrupt Mask Register	R/W	Section 3.8.15

4.5 PCI Bus Resources Operation (continued)

4.5.7 Universe II VME Registers

The Universe II allows a PCI Bus Master to act as a VMEbus Master for data transfers to or from any VMEbus device (see Section 4.5.8, below). It also provides VMEbus Slave access from an external VMEbus Master to certain PCI bus resources. Refer to Chapter 5 for further description of Universe II VMEbus operation on the Model 4205.

The Universe II provides programmable registers for VME system configuration. These registers are divided into three groups: PCI Configuration Space, VMEbus Configuration and Status Registers, and Universe II Device Specific Status Registers. A reference list of these registers is provided in Appendix B. Refer to the Tundra Universe II User Manual for detailed description of these registers. A PCI bus device can access these registers starting at PCI memory address **0x4060 0000**, or starting at PCI I/O address **0x1000**.

4.5.8 VMEbus Master Access

When the Model 4205 is a VMEbus Master, any device that masters either PCI bus also has Master access to the VMEbus and can perform D16, D32, and D64 block transfers in all relevant VMEbus address spaces. See Section 5.2 for more information on VMEbus Master operation. The VMEbus Master Memory Map for a PCI bus device is provided in the following table, showing the VMEbus addresses generated for each PCI address range.

Table 4–6: VMEbus Master PCI Bus Addresses

PCI Memory Addresses	VMEbus Address Space (Address Modifiers)	Generated VME Address
0x8001 0000 – 0x8001 FFFF	A16/D16 (Non–Privileged, 0x29)	**0x0000 – 0xFFFF**
0x8002 0000 – 0x8002 FFFF	A16/D32 (Non–Privileged, 0x29)	**0x0000 – 0xFFFF**
0x8100 0000 – 0x81FF FFFF	A24/D16 (Standard, Non–Privileged, Data, 0x39)	**0x00 0000 – 0xFF FFFF**
0x8200 0000 – 0x82FF FFFF	A24/D32 (Standard, Non–Privileged, Data, 0x39)	**0x00 0000 – 0xFF FFFF**
0x9000 0000 – 0x9FFF FFFF	A32/D16 (Extended, Non–Privileged, Data, 0x09)	**0x0000 0000 – 0x0FFF FFFF**
0xA000 0000 – 0xDFFF FFFF	A32/D32 (Extended, Non–Privileged, Data, 0x09)	**0x0000 0000 – 0x3FFF FFFF**
0x8400 0000 – 0x8FFF FFFF	A32/D64 (Extended, Non–Privileged, Data, 0x09)	**0x0000 0000 – 0x0BFF FFFF**

The default boot sequence (Chapter 8) configures the Universe II registers for the VMEbus addresses shown above. The Monitor program (Chapter 9) provides commands for reconfiguring VMEbus Master access at boot time.

4.5 PCI Bus Resources Operation (continued)

4.5.9 PLX 9656 Registers

There are three PLX PCI 9656 Bus Master chips on the Model 4205, one for the VIM control and status interfaces of all four VIM sites, one for the VIM A/B FPGA data interface, and one for the VIM C/D FPGA data interface.

Each 9656 has several programmable registers to control the PCI interface. These registers are described in the PLX PCI 9656 Data Book. A PCI bus device can access the internal registers for each 9656 using PCI memory–mapped or PCI I/O addresses listed in the following table.

Table 4–7: PLX 9656 Registers PCI Bus Addresses

PLX 9656	PCI Memory Address	PCI I/O Address
VIM Control/Status	0x4070 0000 – 0x4070 FFFF	0x2000 – 0x20FF
VIM A/B FPGA	0x40E0 0000 – 0x40E0 FFFF	0x8000 – 0x80FF
VIM C/D FPGA	0x40F0 0000 – 0x40F0 FFFF	0x9000 – 0x90FF

4.5.10 Serial Mezzanine Module Access

The serial mezzanine site provides an optional high–speed serial data interface to the Model 4205 front panel. If a serial mezzanine module is installed, a PCI bus device can access the module's local address space resources (local module I/O, memory, registers, DMA, etc.) starting at PCI memory address **0x6000 0000**, or starting at PCI I/O address **0xB000** if PCI I/O access is implemented on the serial module.

4.5.10.1 Fibre Channel Access (Option 006)

When the Option 006 Fibre Channel module is installed on the serial mezzanine site, the Model 4205 allows up to 2 Gigabits/second data transfers using a front panel copper Fibre Channel serial I/O connector.

4.5.10.2 Gigabit Ethernet Access (Option 007)

When the Option 007 Gigabit Ethernet module is installed on the serial mezzanine site, the Model 4205 allows 1000BaseT Gigabit Ethernet I/O transfers using a front panel Ethernet connector.

4.5 PCI Bus Resources Operation (continued)

4.5.11 PMC Module Access

The Model 4205 PCI Mezzanine Card (PMC) site accepts industry−standard PMC modules. If a PMC module is installed, a PCI bus device can access PMC local address space resources (I/O, memory, registers, DMA, etc.) starting at PCI memory address **0x4400 0000**, or starting at PCI I/O address **0xA000** if PCI I/O access is implemented on the PMC module.

Refer to the Operating Manual supplied with your PMC module for description of its operation and programmable resources. Refer to Section 4.5.12, below, for description of Pentek's RACE++ PMC module, Option 010.

4.5.12 RACEway Access (Option 005 or 010)

Option 005 for the Model 4205 provides a single RACEway RACE++ interface to the Primary PCI Bus. Option 010 includes the Option 005 interface and adds a second RACE++ interface to the Secondary PCI Bus.

When either Option 005 or 010 is installed, the Model 4205 supports outgoing (Master) and incoming (Slave) RACEway operations using the PCI bus. With Option 005, the Model 4205 allows data transfers up to 267 MBytes/second with other RACEway−equipped VME boards. Option 010 allows data transfers up to 533 MBytes/second with other RACEway−equipped VME boards. Refer to Appendix D for description of Raceway operation.

NOTE: The optional RACEway interfaces can generate Broadcast/Multicast writes, but cannot receive them.

Options 005 and 010 both include the primary RACE++ interface to the Primary PCI Bus. This interface uses a PXB++ PCI Bridge chip from Mercury Computer Systems. A PCI bus device can access primary RACEway address space starting at PCI memory address **0xE000 0000**, and can access the PXB++ chip internal configuration registers starting at memory address **0x40A0 0000**.

NOTE: The Pentek default boot sequence (Chapter 8) configures the PXB++ internal registers for proper operation with Pentek's board support software—you do not need to access these registers.

Option 010 provides a second RACE++ interface to the Secondary PCI Bus. Option 010 occupies the PMC site (Section 4.5.11, above), and uses a PXB++ PCI Bridge chip. A PCI bus device can access the second RACEway address space starting at PCI memory address **0x5000 0000**, and can access the PXB++ chip internal configuration registers starting at memory address **0x4400 0000**. (There is no PCI I/O−mapped access implemented for the Option 010 PMC module.)

4.5 PCI Bus Resources Operation (continued)

4.5.13 Option 150/151/152 PMC Module Access

The Option 150/151/152 PMC Extender Board for VMEbus provides additional PCI Mezzanine Card (PMC) sites for the Model 4205. This board extends the baseboard PMC connector into a second VME–bus slot, where up to three PMC modules can be mounted. These three PMC sites are identified as **PMC 1**, **PMC 2**, and **PMC 3**, as illustrated in the figure at the right.

NOTE: The Option 150/151/152 PMC Extender Board is documented separately from the Model 4205, as the Model 9220 PMC Extender Board. Refer to the Model 9220 Operating Manual, part # 800.92200 (which is supplied with Model 4205 Option 150/151/152), for detailed description of the Extender Board setup and use.

If a PMC module is installed on Option 150/151/152 PMC Extender Board, depending on the Extender Board site the PMC is mounted on (see figure at the right), a PCI bus device can access PMC local address space resources (module I/O, memory, reg–isters, DMA, etc.) at the addresses listed in the fol–lowing table.

Table 4–8: Option 150/151/152 PCI Bus Addresses

Option 15x PMC Site	PCI Memory Address	PCI I/O Address
PMC 1	0x4400 0000 – 0x4FFF FFFF	0xC000 – 0xCFFF
PMC 2	0x5000 0000 – 0x57FF FFFF	0xD000 – 0xDFFF
PMC 3	0x5800 0000 – 0x5FFF FFFF	0xE000 – 0xEFFF

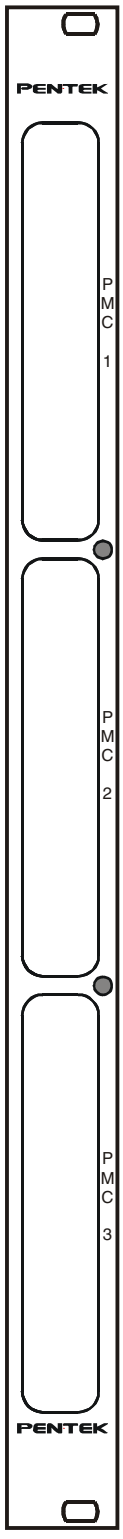


Figure 4–1: Option 15x Front Panel

4.5 PCI Bus Resources Operation (continued)

4.5.14 VIM Module Access

Each Pentek VIM module has resources that can be accessed by processors on the VME baseboard, including programmable registers that configure and control its operation. Each VIM module is supplied with an Operating Manual that describes its operation and any programmable resources on that module. Refer to that manual for information on operation and programming these resources, including any on−board control registers and data storage.

When a VIM module is installed on the Model 4205, a PCI bus device can access each VIM site's on−board resources at the addresses listed in the following table.

Table 4−9: VIM Module Access PCI Bus Addresses

VIM Interface	PCI Bus Memory Address	VIM Module Information
VIM A	**0x4080 0000 − 0x4083 FFFF**	Refer to the Operating Manual supplied with your VIM module
VIM B	**0x4084 0000 − 0x4087 FFFF**	
VIM C	**0x4088 0000 − 0x408B FFFF**	
VIM D	**0x408C 0000 − 0x408F FFFF**	

Each VIM interface has a unique base address in this table. This allows a PCI device to address the VIM resources that are identified for each baseboard processor, A, B, C, or D (most Pentek VIM modules identify access to their resources in terms of baseboard processors). For example, to access a Processor C VIM resource on the baseboard, the PCI device would access that resource starting at base address **0x4088 0000**.

In general, Pentek VIM module control registers are accessed at a fixed memory offset from the module's starting address—this offset is **0x20000**.

For example, to access the first register of a Pentek VIM C module from the Processor C baseboard interface, a PCI device would access that register at address **0x408A 0000** (VIM C Module base address [**0x4088 0000**] + VIM module register offset [**0x20000**]).

Refer to Section 6.3 for additional description of VIM module operation.

4.5 PCI Bus Resources Operation (continued)

4.5.15 VIM Interface Configuration Registers

Each VIM interface has a bi−directional FIFO (BIFO) that buffers data transfers between the VIM module and the PCI interface. Each VIM BIFO has a streaming data interface through one of two on−board Xilinx Virtex−II FPGAs. One of these FPGAs is dedicated to VIM A and B interfaces and the other is dedicated to VIM C and D.

The Model 4205 has several 32−bit registers that configure and control the operation of these VIM BIFO and FPGA interfaces, as described in Section 6.4. A PCI bus device can access these registers at the memory addresses listed in the following table. The referenced subsections provide information about the use and programming of each register.

Table 4−10: VIM Interface Configuration Register PCI Bus Addresses

PCI Bus Memory Address	Register	Access	Register Information
0x4090 0000	VIM A/B FPGA Configuration Control Register	R/W	Section 6.4.1
0x4090 0004	VIM A/B FPGA Configuration Data Register	W.O.	Section 6.4.2
0x4090 0008	VIM C/D FPGA Configuration Control Register	R/W	Section 6.4.1
0x4090 000C	VIM C/D FPGA Configuration Data Register	W.O.	Section 6.4.2
0x4090 0010	VIM Interface BIFO Control Register	R/W	Section 6.4.3
0x4090 0014	VIM Interface BIFO Reset Status Register	R.O.	Section 6.4.4
0x4090 0018	VIM FPGA DCM Reset Register	R/W	Section 6.4.5

NOTE: Several of the VIM BIFO/FPGA resources described in this manual are dependent on the configuration determined by the above registers. In particular, the VIM Interface BIFO Control Register (Section 6.4.3) selects the data routing from the VIM BIFO to other board resources. Some resources are only available for a configuration that routes data from the VIM BIFOs to the FPGA (VIM BIFO PATH SELECT = FPGA), and others are only available for a configuration that routes data from the VIM BIFOs to the PCI interface (VIM BIFO PATH SELECT = PCI Interface).

4.5 PCI Bus Resources Operation (continued)

4.5.16 VIM BIFO/FPGA Access

Each VIM interface has a bi−directional FIFO (BIFO) that buffers data transfers between the VIM module and the PCI interface, and a streaming data interface through one of two on−board Xilinx Virtex−II FPGAs. One of these FPGAs is dedicated to VIM A and B interfaces and the other is dedicated to VIM C and D. Serial ports provide an additional data streaming connection between the processor each VIM module.

The VIM BIFO and FPGA interfaces can be configured in different ways using the VIM Interface Configuration Registers, Section 4.5.15. Two default VIM BIFO/FPGA configurations are software selectable at boot time: VIM to Processor, and VIM to VIM. Different memory maps are provided for the VIM BIFO/FPGA resources available on the PCI bus for each configuration, as described in Section 6.5.

- **VIM to Processor**: Each FPGA is configured with two 2 x 4k x 32 dual−ported random access memory (DPRAM) swinging buffers. Data from each VIM BIFO can be fed to one DPRAM swinging buffer. The DPRAM buffers can transfer data by master or slave DMA using a 2k x 32 FPGA DMA RAM memory.

  This configuration also allows direct PCI bus read/write access to the VIM BIFO Data and Mailbox resources, and to the VIM Serial Ports.

  See Section 6.5.1 for the memory maps of VIM BIFO/FPGA resources available to a PCI bus device for the VIM to Processor configuration. Also, see Figure 6−1 for illustration of this configuration.

- **VIM to VIM**: Each FPGA is configured to allow direct 105−MHz data transfers between VIM sites A and C, and between VIM sites B and D, respectively.

  This configuration does not include the FPGA DPRAM swinging buffers or the FPGA DMA RAM memory—these are available only in the VIM to Processor configuration. However, access to the VIM Serial Ports, VIM BIFO Data, and VIM BIFO Mailboxes from the PCI bus is the same in both configurations.

  See Section 6.5.2 for the memory maps of VIM BIFO/FPGA resources available to a PCI bus device for the VIM to VIM configuration. Also, see Figure 6−2 for illustration of this configuration.

Chapter 5: VME Interface Memory Maps and Resources

5.1 Overview

The Model 4205 provides a full−featured Master/Slave VME64 interface. As a VMEbus Master, the board can access the entire VME address space for A32, A24, and A16 modes of operation. As a VMEbus Slave, the board presents A32, A24, and A16 memory space to any external master. Using the Tundra Universe II as the interface, the board supports 64−bit master/slave block transfers in addition to 32−bit and 16−bit block transfers. The board also supports 32−bit, 16−bit, and 8−bit single cycle transfers as a VMEbus Master and Slave. As a VMEbus system controller, the board can act as a bus arbiter for multiple master systems.

5.2 VMEbus Master Access

As a VMEbus Master, the Model 4205 can access 32−bit, 16−bit, or 8−bit data and perform D32 and D64 block transfers in all relevant address spaces. The user can configure the 4205's VMEbus master interface from the GT−64260 serial port or using VME A16 Space. The Model 4205 Monitor program (see Chapter 9) provides configuration menus for configuring a custom VMEbus Master Memory Map. The default VMEbus Master Memory Map is provided in the following table, showing the VMEbus address generated for the MPC745X and PCI bus addresses in each address region.

Table 5−1: VMEbus Master Memory Map

VMEbus Address Space (Address Modifiers)	MPC745X Address	PCI Bus Address	VMEbus Address
A16/D16 (Non−Privileged, 0x29)	0x8001 0000 − 0x8001 FFFF	0x8001 0000 − 0x8001 FFFF	0x0000 − 0xFFFF
A16/D32 (Non−Privileged, 0x29)	0x8002 0000 − 0x8002 FFFF	0x8002 0000 − 0x8002 FFFF	0x0000 − 0xFFFF
A24/D16 (Standard, Non−Privileged, Data, 0x39)	0x8100 0000 − 0x81FF FFFF	0x8100 0000 − 0x81FF FFFF	0x00 0000 − 0xFF FFFF
A24/D32 (Standard, Non−Privileged, Data, 0x39)	0x8200 0000 − 0x82FF FFFF	0x8200 0000 − 0x82FF FFFF	0x00 0000 − 0xFF FFFF
A32/D16 (Extended, Non−Privileged, Data, 0x09)	0x9000 0000 − 0x9FFF FFFF	0x9000 0000 − 0x9FFF FFFF	0x0000 0000 − 0x0FFF FFFF
A32/D32 (Extended, Non−Privileged, Data, 0x09)	0xA000 0000 − 0xDFFF FFFF	0xA000 0000 − 0xDFFF FFFF	0x0000 0000 − 0x3FFF FFFF
A32/D64 (Extended, Non−Privileged, Data, 0x09)	0x8400 0000 − 0x8FFF FFFF	0x8400 0000 − 0x8FFF FFFF	0x0000 0000 − 0x0BFF FFFF

5.3 VMEbus Slave Access

As a VMEbus Slave, the Universe II is responsible for decoding the address and controlling VMEbus access from an external VMEbus Master to various on–board resources. The SDRAM, some PCI bus resources, and certain Processor resources are accessible from the VMEbus. The board's slave interface supports single accesses and block transfers of D16, D32, and D64 data sizes.

The Model 4205 is shipped from the factory with VMEbus A16 address space slave access enabled. Jumpers on the PCB configure the board to respond as a Slave in A16 or A24 address space (see Section 2.2.1), and set the A16/A24 slave base address (see Section 2.2.3). The factory default setting of these jumpers is for A16 address space, starting at A16 slave base address **0x0000**. The Model 4205 memory map occupies 4 kBytes of VME space for the default configuration in either address mode.



If you want an external VMEbus Master to access the board's A24 memory space in A24 slave address mode, you must set the A24 slave base address (jumper block **JB2**, Section 2.2.3) to either **0x00 0000** or **0x80 0000**.

The Model 4205 is shipped with A16 VME slave access enabled to the Universe II registers only (see Default VMEbus A16/A24 Slave Memory Map, Section 5.3.2). Using the Model 4205 RedBoot Monitor program through the GT–64260 serial port, or writing to the Universe II registers from an external VMEbus Master, you can enable VME slave access to other board resources in A24 or A32 address spaces.

5.3.1 Using the Monitor to Configure Slave Access

The Model 4205 RedBoot Monitor program provides commands for configuring A24 or A32 VME slave access to various board resources at boot time. Refer to Chapter 9 for description of the operation of the RedBoot Monitor.

When you enable extended A24 or A32 VME slave access using the default Monitor program parameters, an Extended VMEbus Slave Memory Map is enabled that provides access to several Model 4205 resources from an external VMEbus Master (see the Extended VMEbus A24 or A32 Slave Memory Maps, Section 5.3.3 or 5.3.4).

You can specify a custom VMEbus slave memory map using the Monitor program commands. If you create a custom memory map from the Monitor, the custom memory map is stored in FLASH memory and restored after a board reset. The Monitor program also provides an option to restore the factory default VMEbus configuration parameters if needed.

5.3 VMEbus Slave Access (continued)

5.3.2 Default VMEbus A16/A24 Slave Memory Map

At board power on, the Model 4205 provides a default A16/A24 Slave Memory Map that allows access only to the Universe II registers from an external VMEbus Master. The factory default access is for VMEbus A16 address space, starting at A16 slave base address **0x0000**.

The following table shows the default VMEbus A16/A24 Slave Memory Map.

Table 5–2: Default VMEbus A16/A24 Slave Memory Map

VMEbus Address Space	Universe II Default VMEbus Address
A16	A16VME_base + **0x0000 – 0x0FFF**
A24	A24VME_base + **0x00 0000 – 0x00 0FFF**

The Address Space, A16 or A24, is determined by the setting of PCB jumper **JB1**, Section 2.2.1. The factory default setting is A16.

The A16/A24 slave base address 'AnnVME_base' is set by PCB jumper block **JB2**, Section 2.2.3. The factory default setting is **0x0000**.

The Universe II provides several user–programmable registers that facilitate VME system configuration and allow the user to control Universe II operational characteristics. The registers are divided into three groups: PCI Configuration Space, VMEbus Configuration and Status Registers, and Universe II Device Specific Status Registers. A quick reference list of these registers is provided in Appendix B. Refer to the Tundra Universe II User Manual for detailed descriptions of these registers.

The user can configure A24 or A32 slave access to various resources on the Model 4205 from VMEbus A16 or A24 space using these Universe II registers. An external VMEbus Master can access the Universe II registers starting at the default VMEbus A16 slave address 'A16VME_base' + **0x0000**, or starting at the default VMEbus A24 slave address 'A24VME_base' + **0x00 0000**.

When you enable either A24 or A32 Extended VME slave access with the Monitor program, Section 5.3.1, an Extended VMEbus Slave Memory Map is enabled that provides access to additional Model 4205 resources from an external VMEbus Master (see Section 5.3.3 or 5.3.4).



You can enable only one of the two Extended VMEbus Slave Memory Maps, either A24 (Section 5.3.3) or A32 (Section 5.3.4).

5.3 VMEbus Slave Access (continued)

5.3.3 Extended VMEbus A24 Slave Memory Map

After board power–up, you can enable VMEbus A24 slave access to various board resources from an external VMEbus Master using the Universe II regis–ters (see Default A16/A24 Slave Memory Map, Section 5.3.2).

Alternately, if you enable the VME Slave Extended Memory Map for A24 access using the Monitor program (Section 9.4), an extended VMEbus A24 slave memory map is enabled, as shown in Table 5–3 below. The Universe II Registers addresses in this table are shown as offsets relative to the A24 slave base address 'A24VME_base' set by PCB jumper block **JB2**, Section 2.2.3. The remaining VMEbus addresses in this table are offset from the A24 slave base address set by the Monitor (the default is **0x0**).

Table 5–3: Extended VMEbus A24 Slave Memory Map

VMEbus A24 Address	Resource	Information
A24VME_base + **0x00 0000 – 0x00 0FFF**	Universe II Registers [1]	Section 5.4.1
A24VME_base + **0x00 1000 – 0x00 1FFF**	GT–64260 Message Unit	Section 5.4.3
A24VME_base + **0x00 2000 – 0x00 5FFF**	SDRAM Boot Area [2]	Section 5.4.4
A24VME_base + **0x00 6000 – 0x00 FFFF**	Unused	–
A24VME_base + **0x01 0000 – 0x01 FFFF**	GT–64260 Internal Registers	Section 5.4.2
A32VME_base + **0x02 0000 – 0x1F FFFF**	Unused	–
A24VME_base + **0x20 0000 – 0x7F FFFF**	SDRAM User Area [2]	Section 5.4.4

1 – The Universe II registers are available in this memory map only when PCB jumper **JB1** is set for A24 address space. The base address of these registers (A24VME_base) is set by PCB jumper block **JB2**.

2 – An External VMEbus Master can access only certain areas of SDRAM. See Table 5–5 in Section 5.4.4 for the SDRAM addresses accessible.

5.3 VMEbus Slave Access (continued)

5.3.4 Extended VMEbus A32 Slave Memory Map

After board power–up, you can enable VMEbus A32 slave access to various board resources from an external VMEbus Master using the Universe II registers (see Default A16/A24 Slave Memory Map, Section 5.3.2).

Alternately, if you enable the VME Slave Extended Memory Map for A32 access using the Monitor program (Section 9.4), an extended VMEbus A32 slave memory map is enabled, as shown in Table 5–4 below. VMEbus addresses in this table are offset from the A32 slave base address 'A32VME_base' set by the Monitor (the default is **0x0**).

Table 5–4: Extended VMEbus A32 Slave Memory Map

VMEbus A32 Address	Resource	Information
A32VME_base + **0x0000 0000 – 0x0000 0FFF**	Reserved	–
A32VME_base + **0x0000 1000 – 0x0000 1FFF**	GT–64260 Message Unit	Section 5.4.3
A32VME_base + **0x0000 2000 – 0x0000 5FFF**	SDRAM Boot Area *	Section 5.4.4
A32VME_base + **0x0000 6000 – 0x0000 FFFF**	Unused	–
A32VME_base + **0x0001 0000 – 0x0001 FFFF**	GT–64260 Internal Registers	Section 5.4.2
A32VME_base + **0x0002 0000 – 0x000F FFFF**	Unused	–
A32VME_base + **0x0010 0000 – 0x005F FFFF**	PMC Module Access	Section 5.4.8
A32VME_base + **0x0060 0000 – 0x0060 0FFF**	Board Control/Status Registers	Section 5.4.5
A32VME_base + **0x0060 1000 – 0x007F FFFF**	Unused	–
A32VME_base + **0x0080 0000 – 0x0080 0FFF**	Board Interrupt Registers	Section 5.4.6
A32VME_base + **0x0080 1000 – 0x008F FFFF**	Unused	–
A32VME_base + **0x0090 0000 – 0x0090 0FFF**	Board Interrupt Registers	Section 5.4.6
A32VME_base + **0x0090 1000 – 0x0090 FFFF**	Unused	–
A32VME_base + **0x00A0 0000 – 0x00AF 00FF**	PLX 9656 Registers, VIM Control/Status	Section 5.4.7
A32VME_base + **0x00AF 0100 – 0x00AF FFFF**	Unused	–
A32VME_base + **0x00B0 0000 – 0x00BF FFFF**	VIM Module Access	Section 5.4.10
A32VME_base + **0x00C0 0000 – 0x00CF FFFF**	VIM Interface Configuration Registers	Section 5.4.11
A32VME_base + **0x00D0 0000 – 0x00DF FFFF**	RACEway PXB++ Registers (Option 005/010)	Section 5.4.9
A32VME_base + **0x00E0 0000 – 0x00E0 00FF**	PLX 9656 Registers, VIM A/B FPGA	Section 5.4.7
A32VME_base + **0x00E0 0100 – 0x00EF FFFF**	Unused	–
A32VME_base + **0x00F0 0000 – 0x00F0 00FF**	PLX 9656 Registers, VIM C/D FPGA	Section 5.4.7
A32VME_base + **0x00F0 0100 – 0x00FF FFFF**	Unused	–
A32VME_base + **0x0100 0000 – 0x02FF FFFF**	Unused	–
A32VME_base + **0x0300 0000 – 0x03FF FFFF**	SDRAM User Area *	Section 5.4.4

* An External VMEbus Master can access only certain areas of SDRAM. See Table 5–5 in Section 5.4.4 for the SDRAM addresses accessible.

5.4 VMEbus Slave Resources Operation

The following sections describe the operation of Model 4205 resources from an external VMEbus Master using the default Extended VMEbus Slave Memory Maps. All VME–bus addresses in the following sections are shown as offsets relative to the base address 'AnnVME_base' in the Extended VMEbus A24 Slave Memory Map, Section 5.3.3, or Extended VMEbus A32 Slave Memory Map, Section 5.3.4. Note that several of these resources are available only in VMEbus A32 address space.

5.4.1 Universe II Registers

The Tundra Universe II provides several user–programmable registers that facilitate VME system configuration and allow the user to control Universe II operational characteristics. The registers are divided into three groups: PCI Configuration Space, VMEbus Configuration and Status Registers, and Uni–verse II Device Specific Status Registers. A reference list of these registers is provided in Appendix B. Refer to the Tundra Universe II User Manual for detailed description of these registers.

These registers are available only in the Extended VMEbus A24 Slave Memory Map, when the PCB jumper block **JB1** is set to respond as a Slave in A24 address space (see Section 2.2.1). The Universe II Register addresses are offset relative to the A24 slave base address 'A24VME_base' set by PCB jumper block **JB2** (Section 2.2.3). An external VMEbus Master can access these registers starting at the VMEbus A24 slave address 'A24VME_base' + **0x00 0000**.

5.4.2 GT–64260 Internal Registers

The GT–64260 internal registers set up the communications and data paths between the PCI buses and the SDRAM, FLASH memory, VIM interfaces, and other Model 4205 global resources. See Section 3.6 for description of the use of these registers in the Model 4205. Refer to the Marvell GT–64260 Datasheet for description and programming of all internal registers.

Using the Extended VMEbus Slave Memory Maps, an external VMEbus Mas–ter can access the GT–64260's internal registers, starting at the VMEbus A24 slave address 'A24VME_base' + **0x01 0000**, or starting at the VMEbus A32 slave address 'A32VME_base' + **0x0001 0000**.



The GT–64260 internal registers are programmed in Little Endian. If you access them from an external VMEbus Master that is Big Endian, you must perform byte swapping.

5.4 VMEbus Slave Resources Operation (continued)

5.4.3 GT–64260 Messaging Unit

The GT–64260 Messaging Unit (MU) allows an external VMEbus Master to communicate with PCI resources through command passing and interrupt generation, using Message and Doorbell Registers, and Circular Queues.

Using the Extended VMEbus Slave Memory Maps, an external VMEbus Master can access the GT–64260 MU registers from the Primary PCI Bus, starting at the VMEbus A24 slave address 'A24VME_base' + **0x00 1000**, or starting at the VMEbus A32 slave address 'A32VME_base' + **0x0000 1000**.

In addition, an external VMEbus Master can access some of the GT–64260 MU registers from the GT–64260 internal register space, starting at the VMEbus A24 slave address 'A24VME_base' + **0x01 0000**, or starting at the VMEbus A32 slave address 'A32VME_base' + **0x0001 0000** (see Section 5.4.2).

5.4.4 SDRAM Access

Using the Extended VMEbus Slave Memory Maps, an external VMEbus Master has limited access to the following two areas of SDRAM.

- Access to 32 kBytes of the Boot Area of SDRAM, starting at PCI bus address **0x0000 1000** (see Section 8.3.1 for details of the SDRAM Boot Area). (The first 4 kBytes of SDRAM address space are used for the GT–64260 Messaging Unit, Section 5.4.3 above.)

- Access to the User Area of SDRAM, starting at PCI bus address **0x0010 0000**. The size of SDRAM User Area that can be accessed by an external VMEbus Master is 6 MBytes when addressed from A24 address space or 16 MBytes when addressed from A32 address space.

The following table shows the VMEbus addresses that access each area of SDRAM, the size of each area that can be accessed, and the equivalent MPC745X or PCI bus addresses for these VMEbus addresses.

Table 5–5: VMEbus SDRAM Access Addresses

SDRAM Area	A24/A32 VMEbus Address	Available SDRAM Size	Equivalent MPC745X or PCI Bus Address
Boot Area	A24VME_base + **0x00 2000** – **0x00 5FFF**	32 kBytes	**0x0000 1000** – **0x0000 4FFF**
	A32VME_base + **0x0000 2000** – **0x0000 5FFF**		
User Area	A24VME_base + **0x20 0000** – **0x7F FFFF**	6 MBytes	**0x0010 0000** – **0x006F FFFF**
	A32VME_base + **0x0300 0000** – **0x03FF FFFF**	16 MBytes	**0x0010 0000** – **0x010F FFFF**

5.4 VMEbus Slave Resources Operation (continued)

5.4.5 Board Control/Status Registers

Using the Extended VMEbus A32 Slave Memory Maps, an external VMEbus Master can access the board control registers at the VMEbus A32 slave address shown in the following table (see Section 3.7 for more information).

Table 5–6: Board Control/Status Register VMEbus Addresses

Register	VME A32 Address	Access	Information
LED Control Register	A32VME_base + **0x0060 0000**	R/W	Section 3.7.1
Reset Register	A32VME_base + **0x0060 0004**	R/W	Section 3.7.2
Fault Register	A32VME_base + **0x0060 0008**	R/W	Section 3.7.3
BIST Register	A32VME_base + **0x0060 000C**	R/W	Section 3.7.4
Module Status Register	A32VME_base + **0x0060 0010**	R.O.	Section 3.7.5
Jumper Status Register	A32VME_base + **0x0060 0014**	R.O.	Section 3.7.6
FPGA Jumper Status Register	A32VME_base + **0x0060 0018**	R.O.	Section 3.7.7

5.4.6 Board Interrupt Registers

Using the Extended VMEbus A32 Slave Memory Maps, an external VMEbus Master can access the board interrupt registers at the VMEbus A32 slave address shown in the following table (see Section 3.8 for more information).

Table 5–7: Board Interrupt Register VMEbus Addresses

Register	VME A32 Address	Access	Information
VIM BIFO Full Interrupt Mask	A32VME_base + **0x0080 0000**	R/W	Section 3.8.1
VIM BIFO Empty Interrupt Mask	A32VME_base + **0x0080 0004**	R/W	Section 3.8.2
PMC Interrupt Mask	A32VME_base + **0x0080 0008**	R/W	Section 3.8.3
VIM Module Interrupt Mask	A32VME_base + **0x0080 000C**	R/W	Section 3.8.4
VIM BIFO Full Interrupt Flag	A32VME_base + **0x0080 0010**	R/Clr	Section 3.8.5
VIM BIFO Empty Interrupt Flag	A32VME_base + **0x0080 0014**	R/Clr	Section 3.8.6
PMC Interrupt Flag	A32VME_base + **0x0080 0018**	R/Clr	Section 3.8.7
VIM Module Interrupt Flag	A32VME_base + **0x0080 001C**	R/Clr	Section 3.8.8
VIM Serial Port Interrupt Flag	A32VME_base + **0x0090 0000**	R/Clr	Section 3.8.9
VIM BIFO Full Interrupt Status	A32VME_base + **0x0090 0004**	R.O.	Section 3.8.10
VIM BIFO Empty Interrupt Status	A32VME_base + **0x0090 0008**	R.O.	Section 3.8.11
PMC Interrupt Status	A32VME_base + **0x0090 000C**	R.O.	Section 3.8.12
VIM Module Interrupt Status	A32VME_base + **0x0090 0010**	R.O.	Section 3.8.13
VIM Serial Port Interrupt Status	A32VME_base + **0x0090 0014**	R.O.	Section 3.8.14
Temp/Volt Fault Interrupt Mask	A32VME_base + **0x0090 0018**	R/W	Section 3.8.15

5.4 VMEbus Slave Resources Operation (continued)

5.4.7 PLX 9656 Registers

There are three PLX PCI 9656 Bus Master chips on the Model 4205, one for the VIM control and status interfaces of all four VIM sites, one for the VIM A/B FPGA data interface, and one for the VIM C/D FPGA data interface.

Each 9656 has several programmable registers. These registers are described in the PLX PCI 9656 Data Book. Using the Extended VMEbus A32 Slave Memory Maps, an external VMEbus Master can access the internal registers for each 9656 at the VMEbus A32 slave address shown in the following table.

Table 5–8: PLX 9656 Registers VMEbus Addresses

PLX 9656	VME A32 Address
VIM Control/Status	A32VME_base + **0x00A0 0000 – 0x00A0 00FF**
VIM A/B FPGA	A32VME_base + **0x00E0 0000 – 0x00E0 00FF**
VIM C/D FPGA	A32VME_base + **0x00F0 0000 – 0x00F0 00FF**

5.4.8 PMC Module Access

Using the Extended VMEbus Slave Memory Maps, an external VMEbus Master can access the PMC local address space (local module I/O, memory, etc.) starting at the VMEbus A32 slave address 'A32VME_base' + **0x0010 0000**.

Refer to the Operating Manual supplied with your PMC module for description of its operation and programmable resources.

5.4.9 RACEway PXB++ Registers (Option 005 or 010)

When either Option 005 or 010 is installed, the Model 4205 supports both outgoing (Master) and incoming (Slave) RACEway operations using the PCI bus. Refer to Appendix D for description of Raceway operation.

Options 005 and 010 both include the primary RACE++ interface to the Primary PCI Bus. This interface uses the PXB++ PCI Bridge chip from Mercury Computer Systems. An external VMEbus Master can access the PXB++ chip internal configuration registers starting at the VMEbus A32 slave address 'A32VME_base' + **0x00D0 0000**.

5.4 VMEbus Slave Resources Operation (continued)

5.4.10 VIM Module Access

Each Pentek VIM module has resources that can be accessed by processors on the VME baseboard, including programmable registers that configure and control its operation. Each VIM module is supplied with an Operating Manual that describes its operation and any programmable resources on that module. Refer to that manual for information on operation and programming these resources, including any on−board control registers and data storage.

When a VIM module is installed on the Model 4205, an external VMEbus Master can access each VIM site's on−board resources at the addresses listed in the following table, using the Extended VMEbus Slave Memory Maps.

Table 5−9: VIM Module Control/Status VMEbus Addresses

VIM Interface	VME A32 Address	VIM Module Information
VIM A	A32VME_base + **0x00B0 0000 − 0x00B3 FFFF**	Refer to the Operating Manual supplied with your VIM module
VIM B	A32VME_base + **0x00B4 0000 − 0x00B7 FFFF**	
VIM C	A32VME_base + **0x00B8 0000 − 0x00BB FFFF**	
VIM D	A32VME_base + **0x00BC 0000 − 0x00BF FFFF**	

Each VIM interface has a unique base address in this table. This allows an external VMEbus Master to address the VIM resources that are identified for each baseboard processor, A, B, C, or D (most Pentek VIM modules identify access to their resources in terms of baseboard processors). For example, to access a Processor C VIM resource on the baseboard, an external VMEbus Master would access that resource starting at the VMEbus A32 slave address 'A32VME_base' + **0x00B8 0000**.

In general, Pentek VIM module control registers are accessed at a fixed memory offset from the module's starting address—this offset is **0x20000**.

Refer to Section 6.3 for further description of the VIM module access.

5.4 VMEbus Slave Resources Operation (continued)

5.4.11 VIM Interface Configuration Registers

Each VIM interface has a bi−directional FIFO (BIFO) that buffers data transfers between the VIM module and the PCI interface. Each VIM BIFO has a streaming data interface through one of two on−board Xilinx Virtex−II FPGAs. One of these FPGAs is dedicated to VIM A and B interfaces and the other is dedicated to VIM C and D.

The Model 4205 has several 32−bit registers that configure and control the operation of these VIM BIFO and FPGA interfaces. Using the Extended VMEbus Slave Memory Maps, an external VMEbus Master can access these registers at the memory addresses listed in the following table. The referenced subsections provide information about the use and programming of each register.

Table 5−10: VIM Interface Configuration Register VMEbus Addresses

Register	VME A32 Address	Access	Information
VIM A/B FPGA Configuration Control	A32VME_base + **0x00C0 0000**	R/W	Section 6.4.1
VIM A/B FPGA Configuration Data	A32VME_base + **0x00C0 0004**	W.O.	Section 6.4.2
VIM C/D FPGA Configuration Control	A32VME_base + **0x00C0 0008**	R/W	Section 6.4.1
VIM C/D FPGA Configuration Data	A32VME_base + **0x00C0 000C**	W.O.	Section 6.4.2
VIM Interface BIFO Control	A32VME_base + **0x00C0 0010**	R/W	Section 6.4.3
VIM Interface BIFO Reset Status	A32VME_base + **0x00C0 0014**	R.O.	Section 6.4.4
VIM FPGA DCM Reset	A32VME_base + **0x00C0 0018**	R/W	Section 6.4.5

Refer to Section 6.4 for further description of the VIM FPGA and BIFO interface configuration registers.

This page is intentionally blank

Chapter 6: VIM Operation

6.1 Overview

The Model 4205 provides four Velocity Interface Mezzanine (VIM) interfaces. Each VIM module's Bi−FIFO Streaming Data Interface is connected to a bi−directional FIFO (BIFO) on the Model 4205, which buffers 32−bit parallel data transfers between the VIM module and the PCI interface. Each VIM BIFO can be routed directly to the PCI interface, or through one of two on−board Xilinx Virtex−II FPGAs. One of these FPGAs is dedicated to VIM A and B interfaces and the other is dedicated to VIM C and D. Each VIM module's Serial Port Interface provides an additional data streaming connection between the VIM module and the PCI interface, through the associated FPGAs.

6.2 VIM BIFO/FPGA Configurations

The VIM BIFO and FPGA interfaces can be configured in different ways using the VIM Interface Configuration Registers, Section 6.4. Two default configurations (illustrated on the next page) are software selectable, allowing the board to boot up to one of these mappings. Configurations are stored in nonvolatile FLASH memory.

In addition, for any configuration, each VIM BIFO data bus can be routed directly to the associated PCI interface (the default path) or through the associated FPGA. This BIFO data path choice determines the availability of several VIM BIFO/FPGA resources.

- **Direct to PCI Interface**, Section 6.6 — The default PCI Interface path provides direct VIM BIFO Data Access and Mailbox capabilities. When the VIM BIFO is routed to the PCI interface, prefetching is not possible. Note that with this routing, the FPGA is bypassed and its swinging buffers do not have access to the VIM data.

- **Through the FPGA**, Section 6.8 — You can use the FPGA routing path to pass VIM BIFO data to a swinging buffer, to pass the data directly to another VIM interface, or to perform user−defined processing on the VIM data stream before presenting it to the PCI interface. Note that when the VIM BIFO is routed to the FPGA, you cannot access the BIFO Data Access and Mailbox capabilities.

  Each FPGA can be configured with two 2 x 4k x 32 dual−ported random access memory (DPRAM) swinging buffers (the default 'Vim to Processor' configuration shown in Figure 6−1 on the following page). Data from each VIM BIFO can be fed to one DPRAM swinging buffer. The local bus side of each FPGA−PCI interface has access to both of these memories. The DPRAM buffers can transfer data by master or slave DMA. Each has its own DMA channel, and can create PCI interrupts.

The Serial Ports data path between each VIM module and the PCI interface, Section 6.7, is not affected by the VIM BIFO data bus path choice.

6.2 VIM BIFO/FPGA Configurations (continued)

The following two default VIM BIFO/FPGA configurations are selectable at boot time using the RedBoot Monitor (see Chapter 9, **`fpgaconfig`** command, Section 9.4.4). See Section 6.4 for description of the registers that select and control these configurations.

You can route VIM BIFO data through the FPGA DPRAM, or directly to the PCI Interface. Use the VIM Interface BIFO Control Register (Section 6.4.3) to select BIFO data routing.

Each FPGA operates independently.

See Section 6.5.1 for memory maps.

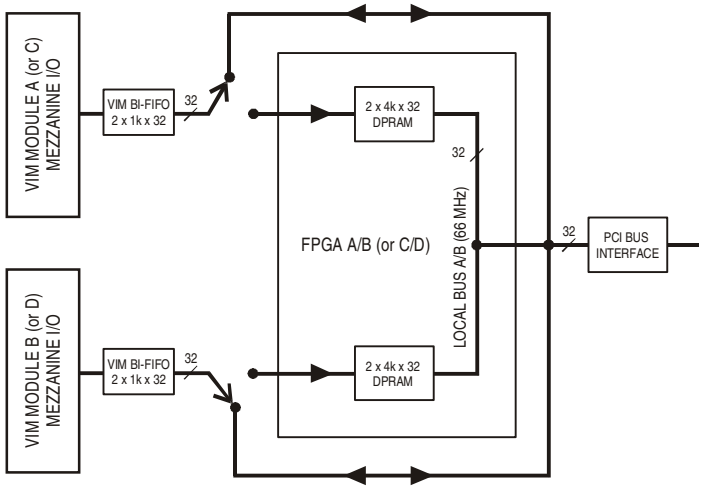


Figure 6–1: VIM BIFO/FPGA Configuration – VIM to Processor

You can route VIM BIFO data through the FPGA to the other FPGA's VIM BIFOs, or directly to the PCI Interface. Use the VIM Interface BIFO Control Register (Section 6.4.3) to select BIFO data routing.

See Section 6.5.2 for memory maps.

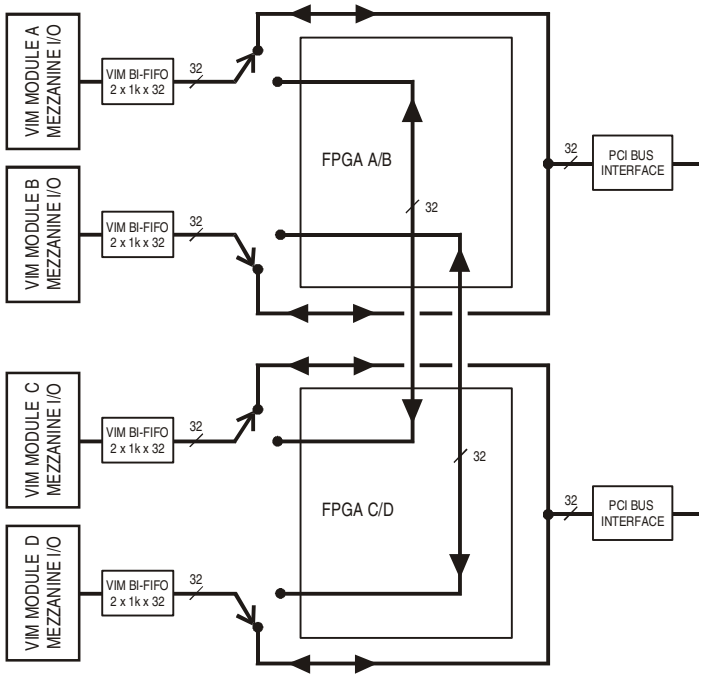


Figure 6–2: VIM BIFO/FPGA Configuration – VIM to VIM

6.3 VIM Module Access

Each Pentek VIM module has on–board resources that can be accessed by processors on the VME baseboard, including programmable registers that configure and control the operation of the VIM module. Each VIM module is supplied with an Operating Manual that describes its operation and any programmable resources on that module. Refer to that manual for information on operation and programming these resources, including any on–board control registers and data storage.

When a VIM module is installed, the MPC745X, PCI bus, or VMEbus can access each VIM site's on–board resources at the addresses listed in the following table.

Table 6–1: VIM Module Access Addresses

VIM Interface	Starting Memory Address		
	MPC745X	PCI Bus	VMEbus A32 Slave *
VIM A	**0x4080 0000**	**0x4080 0000**	A32VME_base + **0x00B0 0000**
VIM B	**0x4084 0000**	**0x4084 0000**	A32VME_base + **0x00B4 0000**
VIM C	**0x4088 0000**	**0x4088 0000**	A32VME_base + **0x00B8 0000**
VIM D	**0x408C 0000**	**0x408C 0000**	A32VME_base + **0x00BC 0000**

* Offset relative to the base address 'A32VME_base' in the Extended VMEbus A32 Slave Memory Map, Section 5.3.4 (there is no access to the VIM modules from VME A24 space)

Each VIM interface has a unique base address. This allows any device on the Model 4205 (MPC745X, PCI device, external VMEbus Master) to address unique VIM resources that are identified for each baseboard processor (most Pentek VIM modules identify their resources in terms of baseboard processors A, B, C, or D). For example, to access a Processor C VIM resource from the MPC745X, you would access that resource starting at MPC745X memory address **0x4088 0000**.

Pentek VIM modules reserve the first 128 kBytes of each VIM memory for I/O space, and the remaining region for memory space. At the time of this publication, there are no VIM modules available from Pentek that utilize the I/O space. VIM module control registers are generally mapped into the memory space, at a fixed memory offset from the module's base address—this offset is **0x20000**.

For example, to access the first register of a Pentek VIM C module from the Processor C interface, the MPC745X would access that register at address **0x408A 0000** (VIM C Module base address [**0x4088 0000**] + VIM module register offset [**0x20000**]).

6.4 VIM Interface Configuration Registers

Each VIM BIFO can buffer data transfers directly between the VIM module and the PCI interface. Alternately, each VIM BIFO can use a streaming data interface through one of the two on−board FPGAs. One FPGA is dedicated to VIM A and B interfaces and the other is dedicated to VIM C and D.

The Model 4205 has a set of 32−bit registers that configure and control the operation of these VIM BIFO and FPGA interfaces. The following table lists the addresses of these VIM interface configuration registers. Each register address is shown as offset relative to the base address of the register set (VIMConfig_base) from different board resources (MPC745X, PCI bus, or VMEbus). The referenced subsections following this table provide information about the format and programming of each register.

Table 6−2: VIM Interface Configuration Register Addresses

Base Address	MPC745X	PCI Bus	VMEbus A32 Slave *	
VIMConfig_base =	**0x4090 0000**	**0x4090 0000**	A32VME_base + **0x00C0 0000**	
Address Offset	**Register**		**Access**	**Information**
VIMConfig_base + **0x0000**	VIM A/B FPGA Configuration Control Register		R/W	Section 6.4.1
VIMConfig_base + **0x0004**	VIM A/B FPGA Configuration Data Register		W.O.	Section 6.4.2
VIMConfig_base + **0x0008**	VIM C/D FPGA Configuration Control Register		R/W	Section 6.4.1
VIMConfig_base + **0x000C**	VIM C/D FPGA Configuration Data Register		W.O.	Section 6.4.2
VIMConfig_base + **0x0010**	VIM Interface BIFO Control Register		R/W	Section 6.4.3
VIMConfig_base + **0x0014**	VIM Interface BIFO Reset Status Register		R.O.	Section 6.4.4
VIMConfig_base + **0x0018**	VIM FPGA DCM Reset Register		R/W	Section 6.4.5

* Offset relative to the base address 'A32VME_base' in the Extended VMEbus A32 Slave Memory Map, Section 5.3.4 (there is no access to these VIM registers from VME A24 space)

Several of the VIM BIFO/FPGA resources described in this chapter are dependent on the configuration determined by the above registers. These BIFO path dependencies are noted where applicable in the following sections.

Two default VIM BIFO/FPGA configurations are selectable at boot time using the Red−Boot Monitor program (see Chapter 9).

- **VIM to Processor**: See Section 6.5.1 for the memory map of VIM BIFO/FPGA resources available for the VIM to Processor configuration. See Figure 6−1 for illustration of this configuration.

- **VIM to VIM**: See Section 6.5.2 for the memory map of VIM BIFO/FPGA resources available for the VIM to VIM configuration. See Figure 6−2 for illustration of this configuration.

6.4 VIM Interface Configuration Registers (continued)

6.4.1 VIM FPGA Configuration Control Registers

There are two VIM FPGA Configuration Control Registers, one for the VIM A/B FPGA and one for the VIM C/D FPGA. These registers contain control and status bits to control the configuration of the FPGA associated with the respective VIM interfaces.

The following table shows this register's bit layout. The subsections following this table describe each register bit.

Table 6–3: VIM FPGA Configuration Control Register

For VIM A/B FPGA: from MPC745X or PCI Bus: R/W @ **0x4090 0000**
from VMEbus A32 space: R/W @ A32VME_base + **0x00C0 0000**
For VIM C/D FPGA: from MPC745X or PCI Bus: R/W @ **0x4090 0008**
from VMEbus A32 space: R/W @ A32VME_base + **0x00C0 0008**

	D31 – D5	D5	D4	D3 *	D2 *	D1	D0
Bit Name	Reserved	WRITE	Reserved	$\overline{\text{INIT}}$	DONE	PRGM	CONFIG MODE
Function	Write zeros, Mask read	0 = Disable 1 = Enable	Write zeros, Mask read	0 = Initializing 1 = Initialized	0 = Configuring 1 = Completed	0 = Disable 1 = Reprogram	0 = Onboard 1 = Upload

* These bits are Read Only
All bits default to the logic '0' state at power up and reset.

6.4.1.1 WRITE Bit D5

This bit is used in the parallel processor upload mode (CONFIG MODE bit D0 = 1) to enable the associated FPGA to accept parallel configuration data from the VIM FPGA Configuration Data Register (Section 6.4.2). Set this bit to logic '1' to enable writing configuration data to the FPGA with the VIM FPGA Configuration Data Register. Clear the bit to logic '0' (its default state) to disable configuration data from being accepted by the FPGA.

6.4.1.2 $\overline{\text{INIT}}$ Bit D3

This read–only bit reflects the state of the associated FPGA's INIT pin. When read as logic '0', the FPGA is either in the process of initializing, or a configuration attempt has failed or found errors. When read as logic '1', the FPGA power–up initialization is complete and the FPGA is ready to be configured.

6.4 VIM Interface Configuration Registers (continued)

6.4.1 VIM FPGA Configuration Control Registers (continued)

6.4.1.3 DONE Bit D2

This read−only bit reflects the state of the associated FPGA's DONE pin. When read as logic '0', the FPGA is not yet configured or the configuration failed. When read as logic '1', the FPGA configuration is complete and without error.

6.4.1.4 PRGM Bit D1

This bit forces a re−configuration of the associated Virtex FPGA. When you transition this bit from logic '0' (its default state) to logic '1', the FPGA begins a configuration reprogramming cycle when in the on−board serial configuration PROM or parallel processor upload modes (see CONFIG MODE bit D0, below). This bit has no effect in the serial download mode (see FPGA Configuration Download jumper, Section 2.2.10), as the Xilinx cable has control of PRGM in that mode.

6.4.1.5 CONFIG MODE Bit D0

This bit selects the source of the Virtex FPGA configuration data. When you clear this bit to logic '0' (its default state), the FPGA selects its configuration data either from the on−board configuration PROMs or from a serial download (depending on the setting of the FPGA Configuration Download jumper, Section 2.2.10). When you set this bit to logic '1', the FPGA is in parallel processor upload mode, and you can upload the configuration from the processor using the associated VIM FPGA Configuration Data Register, as described in Section 6.4.2.

6.4 VIM Interface Configuration Registers (continued)

6.4.2 VIM FPGA Configuration Data Registers

There are two VIM FPGA Configuration Data Registers, one for the VIM A/B FPGA and one for the VIM C/D FPGA. These registers are used to write the configuration data to the FPGA associated with the respective VIM sites when the CONFIG MODE bit (D0) is set to logic '1' in the associated VIM FPGA Configuration Control Register, Section 6.4.1.

The following table shows this register's bit layout. The paragraphs following this table describe these bits.

Table 6–4: VIM FPGA Configuration Data Register

For VIM A/B FPGA: from MPC745X or PCI Bus: W.O. @ **0x4090 0004**
from VMEbus A32 space: W.O. @ A32VME_base + **0x00C0 0004**
For VIM C/D FPGA: from MPC745X or PCI Bus: W.O. @ **0x4090 000C**
from VMEbus A32 space: W.O. @ A32VME_base + **0x00C0 000C**

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0
Function	Write zeros	Data Bytes							

All bits default to the logic '0' state at power up and reset.

Each data byte (DATA7 : DATA0) written to this register is, in turn, written to the associated FPGA configuration data bus.

Writing is only allowed when the CONFIG MODE bit (D0) is set to logic '1' (Upload) and the WRITE bit (D4) is set to logic '1' (Enable) in the associated VIM FPGA Configuration Control Register, Section 6.4.1.



The Pentek default boot sequence (Section 8.3) uses the FPGA Configuration Data Bus to define the two default VIM FPGA configurations that can be selected: VIM to Processor, and VIM to VIM.

If you access this register, the memory maps and resources identified in this chapter may not be available.

6.4 VIM Interface Configuration Registers (continued)

6.4.3 VIM Interface BIFO Control Register

This register contains control bits for the four VIM interface BIFOs.

The following table shows this register's bit layout. The subsections following this table describe each register bit.

Table 6–5: VIM Interface BIFO Control Register

From MPC745X or PCI Bus: R/W @ **0x4090 0010**
From VMEbus A32 space: R/W @ A32VME_base + **0x00C0 0010**

	D31 – D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	Reserved	VIM D BIFO RESET	VIM C BIFO RESET	VIM B BIFO RESET	VIM A BIFO RESET	VIM D BIFO PATH SELECT	VIM C BIFO PATH SELECT	VIM B BIFO PATH SELECT	VIM A BIFO PATH SELECT
Function	Write zeros, Mask read	0 = Normal operation 1 = Reset				0 = PCI Interface 1 = FPGA			

All bits default to the logic '0' state at power up and reset.

6.4.3.1 VIM x BIFO RESET Bits D7–D4

Each of these bits forces a reset to the respective VIM BIFO. Set a bit to logic '1' to hold the BIFO in its reset state. Clear a bit to logic '0' (its default state) to return the BIFO to normal operation.

6.4.3.2 VIM x BIFO PATH SELECT Bits D3–D0

Each of these bits controls the respective VIM BIFO data bus connection. Clear a bit to logic '0' (its default state) to connect the VIM BIFO data bus directly to the PCI interface (PLX 9656 Local Bus). Set a bit to logic '1' to connect the VIM BIFO to the FPGA.

The default PCI Interface path provides direct VIM BIFO Data Access and Mailbox capabilities. You must choose this path to program the BIFO boundary flags. Note that when the BIFO is connected to the PCI interface, the swinging buffers do not have access to the VIM data, and VIM data prefetching is not possible.

Use the FPGA path to pass VIM BIFO data to a DPRAM swinging buffer that is interfaced to the PCI interface, to pass the data directly to another VIM interface, or to perform user defined processing on the data stream before presenting it to the PCI bus. Note that when the VIM BIFO is connected to the FPGA path, you cannot use the direct BIFO Data Access and Mailbox capabilities.

6.4 VIM Interface Configuration Registers (continued)

6.4.4 VIM Interface BIFO Reset Status Register

This register indicates whether a minimum number of BIFO clock cycles have occurred for a VIM BIFO since the assertion of a VIM BIFO RESET (from the VIM Interface BIFO Control Register, Section 6.4.3) for the reset to take affect.

The following table shows this register's bit layout. The paragraphs following this table describe these bits.

Table 6–6: VIM Interface BIFO Reset Status Register

From MPC745X or PCI Bus: R.O. @ **0x4090 0014**
From VMEbus A32 space: R.O. @ A32VME_base + **0x00C0 0014**

	D31 – D8	D3	D2	D1	D0
Bit Name	Reserved	VIM D BIFO RESET STATUS	VIM C BIFO RESET STATUS	VIM B BIFO RESET STATUS	VIM A BIFO RESET STATUS
Function	Mask read	0 = Minimum Reset time not elapsed 1 = Minimum Reset time elapsed			

All bits default to the logic '0' state at power up and reset.

Each of these read–only status bits indicates whether a minimum number of BIFO clock cycles have occurred since the assertion of a RESET to a VIM BIFO (Section 6.4.3). The respective status bit is cleared to logic '0' when the VIM BIFO RESET is not asserted. Once the VIM BIFO RESET has been asserted, the status bit changes to logic '1' after a valid reset period has elapsed. When the VIM BIFO RESET is de–asserted (cleared to logic '0'), the status bit is again cleared to logic '0'.

Each status bit can be monitored to guarantee a valid BIFO RESET when the VIM FCLK clock frequency or presence is not known. A minimum of four clock cycles of both BIFO clocks must occur during assertion of the reset in order to guarantee the BIFO is reset.

6.4 VIM Interface Configuration Registers (continued)

6.4.5 VIM FPGA DCM Reset Register

This register forces a reset of the FPGA internal clock (DCM) for each VIM FPGA. This register contains two reset bits, one for VIM A/B FPGA DCM and one for VIM C/D FPGA DCM.

The following table shows this register's bit layout. The paragraphs following this table describe these bits.

Table 6–7: VIM FPGA DCM Reset Register
From MPC745X or PCI Bus: R/W @ **0x4090 0018**
From VMEbus A32 space: R/W @ A32VME_base + **0x00D0 0018**

	D31 – D2	D1	D0
Bit Name	Reserved	VIM C/D DCM RESET	VIM A/B DCM RESET
Function	Write zeros, Mask read	0 = Normal operation 1 = Reset	
All bits default to the logic '0' state at power up and reset.			

Each of these bits forces a reset to the associated VIM FPGA internal clock DCM. Set this bit to logic '1' to hold the DCM in its reset state. Clear this bit to logic '0' (its default state) to return the DCM to normal operation.

NOTE: After you reset a clock, you must wait until the clock DCM interrupt bit in the FPGA Interrupt Status Register (Section 6.9.8) indicates that the phase–locked loop is locked (DCM interrupt status bit D4 = 0) before you use the associated VIM BIFOs.

6.5 VIM BIFO/FPGA Resources

Two default VIM BIFO/FPGA configurations are selectable at boot time (Chapter 9): VIM to Processor (illustrated in Figure 6–1), and VIM to VIM (illustrated in Figure 6–2). Separate memory maps for each configuration are provided in the following two sub–sections. You can access these resources from the MPC745X or PCI bus, or you can set up the associated PLX 9656 for DMA access to its FPGA resources.

6.5.1 VIM to Processor Configuration Memory Maps

The following two tables show the memory maps for the VIM to Processor configuration selectable at boot time (see Figure 6–1). Each FPGA is config–ured with two 2 x 4k x 32 dual–ported random access memory (DPRAM) swinging buffers. Data from each VIM BIFO can be fed to one DPRAM swinging buffer. The DPRAM buffers can transfer data by master or slave DMA using a 2k x 32 FPGA DMA RAM memory. This configuration also allows direct read/write access to the VIM BIFO Data and Mailboxes, and to the VIM Serial Ports. Each FPGA can operate independently.

For PLX 9656 DMA operation, use the PLX 9656 local bus addresses provided in the following table to access the VIM BIFO/FPGA resources associated with that PLX PCI interface. (Note that the addresses are the same for each PLX 9656—the selection of which VIM resource you access is determined by the PLX 9656 you use.) The VIM BIFO/FPGA resource availability is configu–ration dependent, as identified by the BIFO Path column (set by the VIM Interface BIFO Control Register, Section 6.4.3). The referenced sections pro–vide information about the operation of each resource.

Table 6–8: VIM BIFO/FPGA PLX 9656 Access Memory Map, VIM–Proc

VIM BIFO/FPGA Resource	PLX 9656 Local Bus Address	BIFO Path	Reference
VIM A (C) FPGA Swinging Buffer *	0x00 0000 – 0x00 3FFF	FPGA	Section 6.8.1
VIM B (D) FPGA Swinging Buffer *	0x08 0000 – 0x08 3FFF		
VIM A/B (C/D) FPGA DMA RAM *	0x10 0000 – 0x10 1FFF		Section 6.8.2
VIM A (C) Serial Port	0x18 0000 – 0x18 03FF	Do not care	Section 6.7
VIM B (D) Serial Port	0x1C 0000 – 0x1C 03FF		
VIM A/B (C/D) FPGA Registers	0x80 0000 – 0x8F FFFF		Section 6.9
VIM A (C) BIFO Data	0x90 0000 – 0x93 FFFF	PCI Interface	Section 6.6
VIM A (C) BIFO Mailbox	0x94 0000		Section 6.6.5
VIM B (D) BIFO Data	0x98 0000 – 0x9B FFFF		Section 6.6
VIM B (D) BIFO Mailbox	0x9C 0000		Section 6.6.5
* Prefetching enabled for these addresses			

To set up the associated PLX PCI 9656 for DMA access, see Section 3.5.3 for PLX 9656 register addresses from the MPC745X processor, Section 4.5.9 from a PCI bus device, or Section 5.4.7 from an external VMEbus Master.

6.5 VIM BIFO/FPGA Resources (continued)

6.5.1 VIM to Processor Configuration Memory Maps (continued)

The following table presents a memory map for access to VIM BIFO/FPGA resources from the MPC745X or a PCI bus device. The VIM BIFO/FPGA resource availability is configuration dependent, as identified by the BIFO Path column (set by the VIM Interface BIFO Control Register, Section 6.4.3). The referenced sections provide information about the operation of each resource.

Table 6–9: VIM BIFO/FPGA PCI Bus Access Memory Map, VIM–Proc

VIM BIFO/FPGA Resource	MPC745X or PCI Bus Memory Address	BIFO Path	Reference
VIM A/B FPGA			
VIM A FPGA Swinging Buffer *	0x4100 0000 – 0x4100 3FFF	FPGA	Section 6.8.1
VIM B FPGA Swinging Buffer *	0x4108 0000 – 0x4108 3FFF		
VIM A/B FPGA DMA RAM *	0x4110 0000 – 0x4110 1FFF		Section 6.8.2
VIM A Serial Port	0x4118 0000 – 0x4118 03FF	Do not care	Section 6.7
VIM B Serial Port	0x411C 0000 – 0x411C 03FF		
VIM A/B FPGA Registers	0x4180 0000 – 0x418F FFFF		Section 6.9
VIM A BIFO Data	0x4190 0000 – 0x4193 FFFF	PCI Interface	Section 6.6
VIM A BIFO Mailbox	0x4194 0000		Section 6.6.5
VIM B BIFO Data	0x4198 0000 – 0x419B FFFF		Section 6.6
VIM B BIFO Mailbox	0x419C 0000		Section 6.6.5
VIM C/D FPGA			
VIM C FPGA Swinging Buffer *	0x4200 0000 – 0x4200 3FFF	FPGA	Section 6.8.1
VIM D FPGA Swinging Buffer *	0x4208 0000 – 0x4208 3FFF		
VIM C/D FPGA DMA RAM *	0x4210 0000 – 0x4210 1FFF		Section 6.8.2
VIM C Serial Port	0x4218 0000 – 0x4218 03FF	Do not care	Section 6.7
VIM D Serial Port	0x421C 0000 – 0x421C 03FF		
VIM C/D FPGA Registers	0x4280 0000 – 0x428F FFFF		Section 6.9
VIM C BIFO Data	0x4290 0000 – 0x4293 FFFF	PCI Interface	Section 6.6
VIM C BIFO Mailbox	0x4294 0000		Section 6.6.5
VIM D BIFO Data	0x4298 0000 – 0x429B FFFF		Section 6.6
VIM D BIFO Mailbox	0x429C 0000		Section 6.6.5
* Prefetching enabled for these addresses			

6.5 VIM BIFO/FPGA Resources (continued)

6.5.2 VIM to VIM Configuration Memory Maps

The following two tables show the memory maps for the VIM to VIM configuration selectable at boot time (see Figure 6–2). In this configuration, each FPGA is set up to allow direct 105–MHz data transfers between VIM sites A and C, and between VIM sites B and D.

The VIM to VIM configuration does not include the FPGA DPRAM swinging buffers or the FPGA DMA RAM memory—these are available only in the VIM to Processor configuration, Section 6.5.1. In addition, the FPGA Control/Status Registers are different from the VIM to Processor configuration, as described in Section 6.11. However, direct access to the VIM Serial Ports, VIM BIFO Data, and VIM BIFO Mailboxes from the PCI bus is the same in both configurations.

For direct data transfers between associated VIM sites, both FPGA ends of a VIM to VIM pair (A and C, or B and D) must be set up with complementary functions, as described in VIM to VIM Data Transfer Operation, Section 6.12.

For PLX 9656 DMA operation, use the PLX 9656 local bus addresses provided in the following table to access the VIM BIFO/FPGA resources associated with that PLX PCI interface. (Note that the addresses are the same for each PLX 9656—the selection of which VIM resource you access is determined by the PLX 9656 you use.) The VIM BIFO/FPGA resource availability is configuration dependent, as identified by the BIFO Path column (set by the VIM Interface BIFO Control Register, Section 6.4.3). The referenced sections provide information about the operation of each resource.

Table 6–10: VIM BIFO/FPGA PLX 9656 Access Memory Map, VIM–VIM

VIM BIFO/FPGA Resource	PLX 9656 Local Bus Address	BIFO Path	Reference
VIM A (C) Serial Port	0x18 0000 – 0x18 03FF	Do not care	Section 6.7
VIM B (D) Serial Port	0x1C 0000 – 0x1C 03FF		
VIM A/B (C/D) FPGA Registers	0x80 0000 – 0x8F FFFF		Section 6.11
VIM A (C) BIFO Data	0x90 0000 – 0x93 FFFF	PCI Interface	Section 6.6
VIM A (C) BIFO Mailbox	0x94 0000		Section 6.6.5
VIM B (D) BIFO Data	0x98 0000 – 0x9B FFFF		Section 6.6
VIM B (D) BIFO Mailbox	0x9C 0000		Section 6.6.5

To set up the associated PLX PCI 9656 for DMA access, see Section 3.5.3 for PLX 9656 register addresses from the MPC745X processor, Section 4.5.9 from a PCI bus device, or Section 5.4.7 from an external VMEbus Master.

6.5 VIM BIFO/FPGA Resources (continued)

6.5.2 VIM to VIM Configuration Memory Maps (continued)

The following table presents a memory map for access to VIM BIFO/FPGA resources from the MPC745X or a PCI bus device. The VIM BIFO/FPGA resource availability is configuration dependent, as identified by the BIFO Path column (set by the VIM Interface BIFO Control Register, Section 6.4.3). The referenced sections provide information about the operation of each resource.

Table 6–11: VIM BIFO/FPGA PCI Bus Access Memory Map, VIM–VIM

VIM BIFO/FPGA Resource	MPC745X or PCI Bus Memory Address	BIFO Path	Reference
VIM A/B FPGA			
VIM A Serial Port	0x4118 0000 – 0x4118 03FF	Do not care	Section 6.7
VIM B Serial Port	0x411C 0000 – 0x411C 03FF		
VIM A/B FPGA Registers	0x4180 0000 – 0x418F FFFF		Section 6.11
VIM A BIFO Data	0x4190 0000 – 0x4193 FFFF	PCI Interface	Section 6.6
VIM A BIFO Mailbox	0x4194 0000		Section 6.6.5
VIM B BIFO Data	0x4198 0000 – 0x419B FFFF		Section 6.6
VIM B BIFO Mailbox	0x419C 0000		Section 6.6.5
VIM C/D FPGA			
VIM C Serial Port	0x4218 0000 – 0x4218 03FF	Do not care	Section 6.7
VIM D Serial Port	0x421C 0000 – 0x421C 03FF		
VIM C/D FPGA Registers	0x4280 0000 – 0x428F FFFF		Section 6.11
VIM C BIFO Data	0x4290 0000 – 0x4293 FFFF	PCI Interface	Section 6.6
VIM C BIFO Mailbox	0x4294 0000		Section 6.6.5
VIM D BIFO Data	0x4298 0000 – 0x429B FFFF		Section 6.6
VIM D BIFO Mailbox	0x429C 0000		Section 6.6.5

6.6 VIM BIFO Direct Access

Both default boot FPGA configurations (see Section 6.5) allow direct access to the VIM BIFO data and mailboxes from the PCI buses as described in this section.

NOTE: To access VIM BIFO data and mailboxes as described in this section, you must choose the PCI interface path in the VIM Interface BIFO Control Register, Section 6.4.3 (set VIM BIFO PATH SELECT = PCI Interface).

6.6.1 Initializing VIM BIFOs

To initialize a VIM BIFO and program the BIFO flags, first set the BIFO path to 'PCI Interface' with the VIM Interface BIFO Control Register, Section 6.4.3. Then, reset the BIFO using this VIM Interface BIFO Control Register.



While the VIM BIFO is in reset, you cannot perform reads or writes to the BIFO. You must clear the BIFO RESET bit (Section 6.4.3) and then wait four VIM module clock cycles before you are able to write to the BIFO, as indicated by the VIM Interface BIFO Reset Status Register (Section 6.4.4).

After the reset is cleared, initialize the VIM BIFO programmable boundary flag levels (FIFO Almost Full and FIFO Almost Empty) using the first four writes to the BIFO. Even if the flags are not used, four dummy writes must occur before data transfers, since following a reset the first four writes to the BIFO are always loaded into its flag registers. The VIM BIFO data direction, Incoming or Outgoing, is labeled in reference to the MPC745X processor.

To program the flags, write the following four words first to the VIM BIFO:

Word 1 = 1024-w (Outgoing FIFO Almost Full)
Word 2 = x (Outgoing FIFO Almost Empty)
Word 3 = 1024-y (Incoming FIFO Almost Full)
Word 4 = z (Incoming FIFO Almost Empty)

Where: **1024** = size of each FIFO buffer (32–bit words)
(**16384** for Option 320)
w = Outgoing FIFO Almost Full flag setting
x = Outgoing FIFO Almost Empty flag setting
y = Incoming FIFO Almost Full flag setting
z = Incoming FIFO Almost Empty flag setting

The valid range for each flag is from (decimal) 1 to 1020 (1 to 16,380 for Option 320) 32–bit words. Write each value into the lowest bits of each word, where bit D0 is the least significant bit of the word. See Table 6–12, on the following page, for the starting data address of each VIM BIFO. After the first four writes to establish the flag levels, the VIM BIFO is ready for data transfers.

6.6 VIM BIFO Direct Access (continued)

6.6.2 VIM BIFO Data Access

To read or write data directly from or to a VIM BIFO from the MPC745X or from a PCI bus device, access the BIFO at the memory addresses shown in the following table. To access VIM BIFO data at these addresses, you must choose the PCI interface path in the VIM Interface BIFO Control Register, Section 6.4.3.

Table 6–12: VIM BIFO Data Addresses

VIM BIFO	MPC745X or PCI Bus Memory Address	PLX 9656 Local Bus Address
VIM A	0x4190 0000 – 0x4193 FFFF	0x90 0000 – 0x93 FFFF
VIM B	0x4198 0000 – 0x419B FFFF	0x98 0000 – 0x9B FFFF
VIM C	0x4290 0000 – 0x4293 FFFF	0x90 0000 – 0x93 FFFF
VIM D	0x4298 0000 – 0x429B FFFF	0x98 0000 – 0x9B FFFF

VIM BIFO reads and writes can also be performed using DMA Demand Mode of operation either in the GT–64260, as described in Section 6.6.3, or in the respective FPGA's PLX 9656, as described in Section 6.6.4.

Data reads occur from the Incoming FIFO and writes occur to the Outgoing FIFO. Read and write operations cause the appropriate Incoming/Outgoing FIFO status flags to be set and cleared when the data reaches the programmed boundary levels (Section 6.6.1).

- The VIM Incoming/Outgoing FIFO Full and Empty flags are routed to the board VIM BIFO Interrupt Status Registers (Sections 3.8.10 and 3.8.11).

- GT–64260 Multi–Purpose Pins (MPP) (Section 3.6.8.1) route the VIM Incoming FIFO Almost Full and Outgoing FIFO Almost Empty flags to the GT–64260 IDMA Controller. These are used for VIM BIFO DMA Demand Mode operation in the GT–64260 (Section 6.6.3).

- The VIM Incoming FIFO Almost Full and Outgoing FIFO Almost Empty flags are also routed to the associated PLX 9656. These are used for VIM BIFO DMA Demand Mode operation in the PLX 9656 (Section 6.6.4).

6.6 VIM BIFO Direct Access (continued)

6.6.3 VIM BIFO Demand Mode DMA Operation – GT–64260

In GT–64260 Demand Mode operation, IDMA Channels 0 through 7 are used for writes to each Outgoing VIM FIFO and for reads from each Incoming VIM FIFO as indicated in the following table.

Table 6–13: VIM BIFO GT–64260 IDMA Channels

VIM BIFO	Incoming VIM FIFO	Outgoing VIM FIFO
VIM A BIFO	IDMA Channel 0	IDMA Channel 4
VIM B BIFO	IDMA Channel 1	IDMA Channel 5
VIM C BIFO	IDMA Channel 2	IDMA Channel 6
VIM D BIFO	IDMA Channel 3	IDMA Channel 7

GT–64260 DMA Demand Mode operates as follows:

1) Set up the GT–64260 IDMA Controller for DMA Demand Mode, set the DMA data count and source/destination address, and then activate the appropriate IDMA Channel. Refer to the Marvell GT–64260 Datasheet for IDMA Controller operation and programming.

2) In the VIM module, start or enable the flow of data at its FIFO interface. The FIFO Outgoing and Incoming signals are routed to the GT–64260 MPP pins (Section 3.6.8.1) and will trigger a DMA transfer when the appropriate FIFO boundary condition is met (Figure 6–3 on the following page illustrates the Demand Mode DMA signal timing).

   a) Writing to the Outgoing FIFO—When the Outgoing FIFO Almost Empty condition occurs, a DMA Request is asserted to fill the VIM FIFO from the 4205 source address. A DMA transfer will run until either the DMA completes or the FIFO Almost Full condition exists. When the Outgoing FIFO Almost Full condition occurs, the DMA Request is cleared and the DMA transfer will stop. The DMA transfer will resume when another DMA Request is issued, which will occur when the Outgoing Almost Empty condition is reestablished.

   b) Reading from the Incoming FIFO—When the Incoming FIFO Almost Full condition occurs, a DMA Request is asserted to empty the VIM FIFO to the 4205 destination address. A DMA transfer will run until either the DMA completes or the FIFO Almost Empty condition exists. When the Incoming FIFO Almost Empty condition occurs, the DMA Request is cleared and the DMA transfer will stop. The DMA transfer will resume when another DMA Request is issued, which will occur when the Incoming Almost Full condition is reestablished.

6.6 VIM BIFO Direct Access (continued)

6.6.3 VIM BIFO Demand Mode DMA Operation (continued)

The following figure illustrates the signal timing for Demand Mode DMA operation.

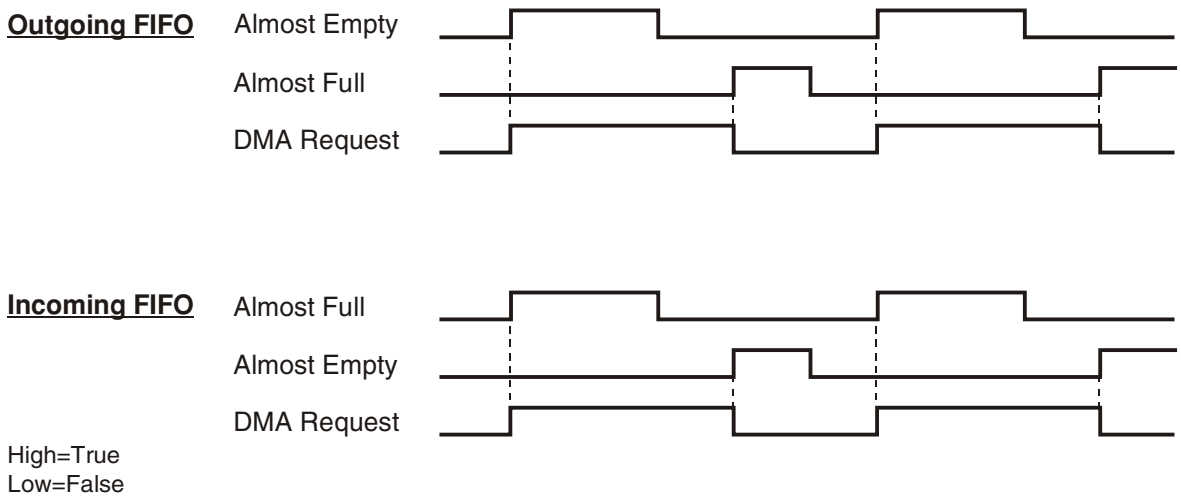


Figure 6–3: VIM BIFO Demand Mode DMA Timing (2 cycles)

Note that the GT–64260 IDMA Controller allows channels to run indepen–dently and concurrently, thus you may initiate and perform both Outgoing and Incoming FIFO DMA operations at the same time.

6.6 VIM BIFO Direct Access (continued)

6.6.4 VIM BIFO Demand Mode DMA Operation – PLX 9656

The PLX 9656 can perform direct DMA transfers between a VIM BIFO and the PCI bus. The following table gives the VIM BIFO addresses to use for PLX 9656 Demand Mode DMA operation.

Table 6–14: VIM BIFO DMA Addresses

VIM BIFO	PLX 9656 Local Bus Address
VIM A BIFO Data	0x90 0000 – 0x93 FFFF
VIM B BIFO Data	0x98 0000 – 0x9B FFFF
VIM C BIFO Data	0x90 0000 – 0x93 FFFF
VIM D BIFO Data	0x98 0000 – 0x9B FFFF

PLX9656 DMA Demand Mode operates as follows:

1) To perform VIM BIFO Demand Mode DMA transfers by the PLX 9656, first enable Direct BIFO Read or Write access (DMA REQUEST = 10 or 11, in the Swinging Buffer Full Mask Register, Section 6.9.3) for the appropriate direction of transfer.

2) Set up the PLX 9656 Demand Mode DMA Channel 0/1, set the DMA data count and source/destination address, and then activate Demand Mode DMA operation in the PLX 9656. Refer to the PLX 9656 Datasheet for DMA operation and programming.

3) In the VIM module, start or enable the flow of data at its FIFO interface. The FIFO Outgoing and Incoming signals will trigger a DMA transfer when the appropriate FIFO boundary condition is met. (Figure 6–3 on the prior page illustrates the Demand Mode DMA signal timing—for PLX 9656 operation, the 'DMA Request' legend refers to the DMA Demand Mode Request 'DREQ0/1' signal.)

   a) Writing to the Outgoing FIFO—When the Outgoing FIFO Almost Empty condition occurs, a DMA transfer starts to fill the VIM FIFO from the 4205 source address. The DMA transfer will run until either the DMA completes or the FIFO Almost Full condition exists.

   b) Reading from the Incoming FIFO—When the Incoming FIFO Almost Full condition occurs, a DMA transfer starts to empty the VIM FIFO to the 4205 destination address. The DMA transfer will run until either the DMA completes or the FIFO Almost Empty condition exists.

6.6 VIM BIFO Direct Access (continued)

6.6.5 VIM BIFO Mailbox Access

The MPC745X or a PCI bus device can access the VIM BIFO mailboxes at the memory addresses shown in the following table. To access VIM Mailbox data at these addresses, you must choose the PCI interface path in the VIM Interface BIFO Control Register, Section 6.4.3.

Table 6–15: VIM BIFO Mailbox Addresses

VIM BIFO	MPC745X or PCI Bus Memory Address
VIM A BIFO	0x4194 0000
VIM B BIFO	0x419C 0000
VIM C BIFO	0x4294 0000
VIM D BIFO	0x429C 0000

A write to a VIM BIFO Mailbox address, above, sends the data word to the respective VIM Mailbox, and asserts the VIM MBOUT interrupt flag to the VIM module.

When the VIM module writes to its mailbox, a VIM MBIN interrupt status is asserted for that VIM module in the Board VIM Module Interrupt Status Register (Section 3.8.13). A read from a VIM Mailbox address, above, reads the data word from the respective VIM Mailbox, and clears the VIM MBIN interrupt from the VIM module.

6.7 VIM Serial Ports

Both default boot FPGA configurations (see Section 6.5) allow access to the VIM Serial Ports. Each VIM interface has one serial port routed to the associated FPGA (A0, B0, C0, and D0). (With Option 300, XC2V3000 FPGA, serial ports A1, B1, C1, and D1 are also connected to FPGA spare pins, but no interface logic is implemented in the default configurations.) You can access the serial ports when either the FPGA or PCI Interface VIM BIFO path is selected by the VIM Interface BIFO Control Register, Section 6.4.3. The serial ports operate with a subset of TI BSP serial port functionality. Several FPGA control/status registers are provided for VIM Serial Port operation—depending on the FPGA configuration, see Section 6.9 or 6.11 for description of these registers.

When you read or write to a serial port, data is routed through a 1–MByte FIFO in the FPGA. The following table shows the starting address of each VIM serial port FIFO.

Table 6–16: VIM Serial Port FIFO Addresses

Serial Port	MPC745X or PCI Bus Memory Address	PLX 9656 Local Bus Address
VIM A (A0)	0x4118 0000 – 0x4118 03FF	0x18 0000 – 0x18 03FF
VIM B (B0)	0x411C 0000 – 0x411C 03FF	0x1C 0000 – 0x1C 03FF
VIM C (C0)	0x4218 0000 – 0x4218 03FF	0x18 0000 – 0x18 03FF
VIM D (D0)	0x421C 0000 – 0x421C 03FF	0x1C 0000 – 0x1C 03FF

Note: Data prefetching must be disabled in the associated PLX 9656 for these addresses.

The following FIFO boundary flags are provided for controlling the serial data flow.

Table 6–17: VIM Serial Port FIFO Flags

Serial Port FIFO Flag	Description
Transmit FIFO Empty (XMIT E)	All data written to the FIFO has been read by the VIM module
Transmit FIFO Not Half Full (XMIT NHF)	Data being read by the VIM from the FIFO is at or below 512 kBytes
Receive FIFO Not Empty (RCV NE)	Data has been written by the VIM to an empty FIFO
Receive FIFO Half Full (RCV HF)	Data being written by the VIM to the FIFO is at or above 512 kBytes
Receive FIFO Full (RCV F)	Data being written by the VIM to the FIFO is at the 1–MByte limit

These flags are routed to the FPGA Interrupt Flag and Status Registers (see Section 6.9 or 6.11). The flags can be monitored individually to control the serial ports, or can be configured to interrupt the MPC745X processor. To interrupt the MPC745X, use the FPGA Interrupt Mask Register (Section 6.9 or 6.11) to specify which flags are routed to the board VIM Serial Port Interrupt Status Register (Section 3.8.14).

6.8 VIM BIFO FPGA Access (VIM to Processor)

The default VIM to Processor FPGA configuration, Section 6.5.1, allows read access to each VIM BIFO through a dual–ported random access memory (DPRAM) in the FPGA.

NOTE: To access a VIM BIFO through the FPGA as described in this section, you must choose the FPGA path (VIM BIFO PATH SELECT = FPGA) in the VIM Interface BIFO Control Register, Section 6.4.3.

6.8.1 VIM FPGA DPRAM Swinging Buffer Operation

In this configuration each FPGA is configured with two 2 x 4k x 32 DPRAM Swinging Buffers, one for each VIM BIFO. These swinging buffers provide a streaming data interface from the associated VIM BIFOs to the PCI interface. The following table shows the memory addresses of the four swinging buff–ers. Data prefetching is enabled for these addresses.

Table 6–18: VIM FPGA Swinging Buffer Addresses

Swinging Buffer	MPC745X or PCI Bus Memory Address	PLX 9656 Local Bus Address
VIM A	0x4100 0000 – 0x4100 3FFF	0x00 0000 – 0x00 3FFF
VIM B	0x4108 0000 – 0x4108 3FFF	0x08 0000 – 0x08 3FFF
VIM C	0x4200 0000 – 0x4200 3FFF	0x00 0000 – 0x00 3FFF
VIM D	0x4208 0000 – 0x4208 3FFF	0x08 0000 – 0x08 3FFF

Each swinging buffer is organized as two 4k x 32 halves of each DPRAM, as illustrated below. Swinging buffer operation is described on the next page.

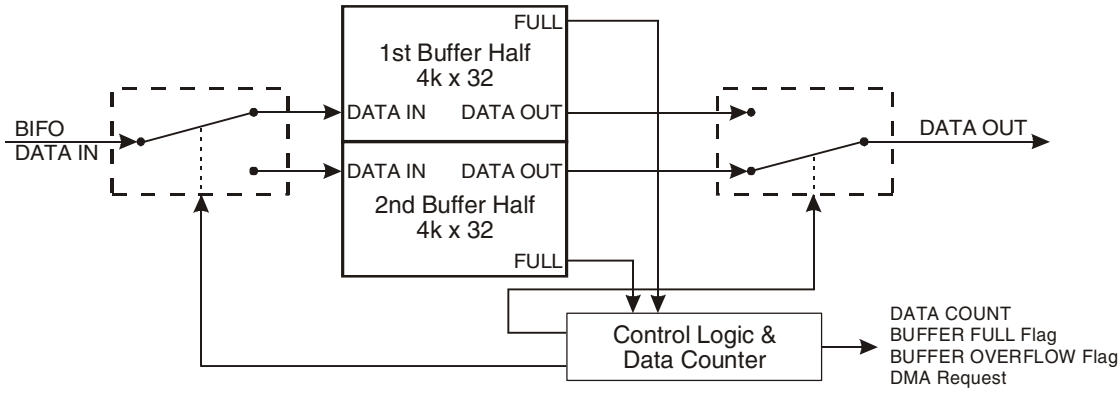


Figure 6–4: DPRAM Swinging Buffers

6.8 **VIM BIFO FPGA Access** (VIM to Proc.) (continued)

6.8.1 **VIM FPGA DPRAM Swinging Buffer Operation** (continued)

VIM BIFO data is written into one half of the swinging buffer while the buffer is write enabled (BUFFER WR EN = Enabled, in the Swinging Buffer Control Register, Section 6.9.1). When the buffer fills to the amount programmed by the Swinging Buffer Length Register (Section 6.9.2), the buffer write will swap and data will automatically continue to write into the other half without any loss of data. When BUFFER WR EN is disabled, buffer writing stops and the buffer write pointer will swap for the next writing operation.

When the buffer goes full, the FPGA will set the BUFFER FULL flag in the Swinging Buffer Full Flag Register (Section 6.9.4), which can assert an interrupt if enabled by the Swinging Buffer Full Mask Register (Section 6.9.3). The Swinging Buffer Count Register (Section 6.9.5) indicates the number of data words available to read in the full buffer. This count will be the amount programmed by the Swinging Buffer Length Register (Section 6.9.2), unless BUFFER WR EN is disabled before the buffer had this length of words written to it. The buffer data can then be read from the first buffer half even as data is simultaneously being written into the second half.

Data can be read in random access fashion. You may read data any number of times since reading does not destroy the data, as in a FIFO. It is not necessary to read all data words if not desired. Once the data has been read, to set up to read the next buffer half you must clear the BUFFER FULL flag (as described in Section 6.9.4). Clearing this bit serves two purposes: first it clears the interrupt, and then it toggles the buffer half pointer in the swinging buffer to provide read access to the next buffer half. The selection of the correct buffer half is automatic, and as far as the user is concerned, each is accessed in the same 4k–address range. If the second buffer half fills before the first half has been cleared, a BUFFER OVFL indication (Section 6.9.8) occurs signaling that both buffer halves are full and there is no more room for data.

NOTE: Since the BUFFER FULL flag controls the buffer swap, do not clear this flag until you have read the buffer data or you will access the incorrect buffer half.

When a buffer overflow occurs, you must read both buffer halves. After the first one is read, write logic '1' to clear the BUFFER FULL flag (Section 6.9.4). Since the other buffer half is also full, the flag will not clear, but the buffer pointer will swap to the second buffer half. After reading the second half, write logic '1' to clear the BUFFER FULL flag again. If the first half has not filled again, this time the flag will clear.

Note that you may also write to the swinging buffer half that is currently selected by the read access buffer pointer. Writing to the buffer half does not set any buffer flags or toggle the pointer.

6.8 VIM BIFO FPGA Access (VIM to Proc.) (continued)

6.8.2 VIM FPGA DMA Link List RAM Operation

To perform chained DMA transfers of the swinging buffer memory, enable the Swinging Buffer Full flag to issue a DMA request to the PLX 9656 (DMA REQUEST = 01, Section 6.9.3). DMA chain definitions are stored in a 2k x 32 FPGA DMA RAM memory, at the addresses indicated below. Data prefetching is enabled for these addresses.

Table 6–19: VIM FPGA DMA RAM Addresses

FPGA RAM	MPC745X or PCI Bus	PLX 9656 Local Bus
VIM A/B	0x4110 0000 – 0x4110 1FFF	0x10 0000 – 0x10 1FFF
VIM C/D	0x4210 0000 – 0x4210 1FFF	

An example DMA chain is illustrated below, in which the buffer will perform DMA transfers alternately to two memory addresses. Interrupts can be created for the completion of any part of the chain, or for the end of the chain. If the DMA chain includes clearing the Swinging Buffer Full flag (Section 6.9.4), as illustrated below, the transfer can run without processor intervention.

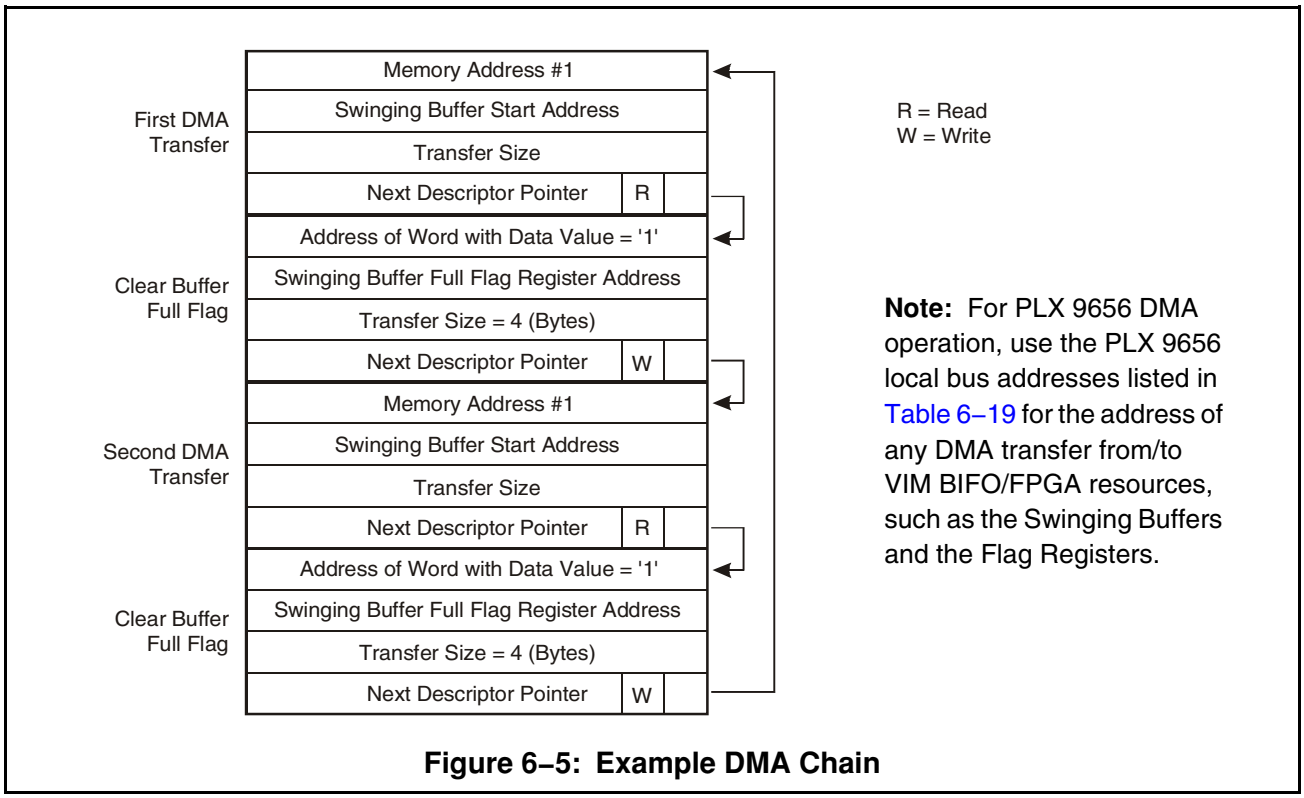


Figure 6–5: Example DMA Chain

6.9 **VIM FPGA Control/Status Registers** (VIM to Processor)

The default VIM to Processor configuration has two sets of 32–bit registers to control the VIM BIFO/FPGA data interface, one set for the VIM A/B FPGA and one set for the VIM C/D FPGA. You can access these control/status registers when you select either the FPGA or PCI Interface VIM BIFO Path with the VIM Interface BIFO Control Register (Section 6.4.3). However, if you choose the PCI Interface path, the Swinging Buffer registers and interrupts have no effect.

Table 6–21, on the next page, lists the addresses of these control/status registers for the two FPGAs. The address of each register is shown as offset relative to a base address for each FPGA ('VIMRegs_base') from the MPC745X or PCI bus, as shown in Table 6–20, below. In addition, when using PLX 9656 DMA operation (such as in DMA Link List, Section 6.8.2), an address is listed for access from the associated PLX local bus. The referenced subsections following these tables provide information about the format and programming of each register.

Table 6–20: VIM FPGA Control/Status Registers Base Address

FPGA		MPC745X or PCI Bus	PLX 9656 Local Bus
VIM A/B	VIMRegs_base =	**0x4180 0000**	**0x80 0000**
VIM C/D		**0x4280 0000**	**0x80 0000**

To access a VIM FPGA control/status register, first determine the base address of the appropriate VIM FPGA for the control device you are using from the above table, then add the offset for the specific register you wish to use from Table 6–21 on the next page.

For example: To access the VIM A Swinging Buffer Control Register from the MPC745X, you would access that register at MPC745X memory address **0x4180 0000** (MPC745X VIM A/B FPGA base address [**0x4180 0000**] + register offset [**0x0000**]).

NOTE: In the following memory map, some registers apply to a specific VIM module data interface (A, B, C, or D)—there are four separate registers for each of these functions, one register per VIM interface. Other registers apply to both VIM interfaces in a specific FPGA (A and B, or C and D)—there are two registers for each of these functions, one register per FPGA.

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

Table 6–21: VIM FPGA Control/Status Register Addresses, VIM–Proc			
VIM A/B FPGA			
Base Address	**MPC745X or PCI Bus**	**PLX 9656**	
VIMRegs_base =	**0x4180 0000**	**0x80 0000**	
Address Offset	**Register**	**Access**	**Information**
VIMRegs_base + **0x0000**	VIM A Swinging Buffer Control Register	R/W	Section 6.9.1
VIMRegs_base + **0x0004**	VIM A Swinging Buffer Length Register	R/W	Section 6.9.2
VIMRegs_base + **0x0008**	VIM A Swinging Buffer Full Mask Register	R/W	Section 6.9.3
VIMRegs_base + **0x000C**	VIM A Swinging Buffer Full Flag Register	R/Clr	Section 6.9.4
VIMRegs_base + **0x0010**	VIM A Swinging Buffer Count Register	R.O.	Section 6.9.5
VIMRegs_base + **0x0014**	Reserved	–	–
VIMRegs_base + **0x0018**	VIM B Swinging Buffer Control Register	R/W	Section 6.9.1
VIMRegs_base + **0x001C**	VIM B Swinging Buffer Length Register	R/W	Section 6.9.2
VIMRegs_base + **0x0020**	VIM B Swinging Buffer Full Mask Register	R/W	Section 6.9.3
VIMRegs_base + **0x0024**	VIM B Swinging Buffer Full Flag Register	R/Clr	Section 6.9.4
VIMRegs_base + **0x0028**	VIM B Swinging Buffer Count Register	R.O.	Section 6.9.5
VIMRegs_base + **0x002C**	Reserved	–	–
VIMRegs_base + **0x0030**	FPGA A/B Interrupt Mask Register	R/W	Section 6.9.6
VIMRegs_base + **0x0034**	FPGA A/B Interrupt Flag Register	R/Clr	Section 6.9.7
VIMRegs_base + **0x0038**	FPGA A/B Interrupt Status Register	R.O.	Section 6.9.8
VIMRegs_base + **0x003C**	FPGA A/B Serial Port Control Register	R/W	Section 6.9.9
VIM C/D FPGA			
Base Address	**MPC745X or PCI Bus**	**PLX 9656**	
VIMRegs_base	**0x4280 0000**	**0x80 0000**	
Address Offset	**Register**	**Access**	**Information**
VIMRegs_base + **0x0000**	VIM C Swinging Buffer Control Register	R/W	Section 6.9.1
VIMRegs_base + **0x0004**	VIM C Swinging Buffer Length Register	R/W	Section 6.9.2
VIMRegs_base + **0x0008**	VIM C Swinging Buffer Full Mask Register	R/W	Section 6.9.3
VIMRegs_base + **0x000C**	VIM C Swinging Buffer Full Flag Register	R/Clr	Section 6.9.4
VIMRegs_base + **0x0010**	VIM C Swinging Buffer Count Register	R.O.	Section 6.9.5
VIMRegs_base + **0x0014**	Reserved	–	–
VIMRegs_base + **0x0018**	VIM D Swinging Buffer Control Register	R/W	Section 6.9.1
VIMRegs_base + **0x001C**	VIM D Swinging Buffer Length Register	R/W	Section 6.9.2
VIMRegs_base + **0x0020**	VIM D Swinging Buffer Full Mask Register	R/W	Section 6.9.3
VIMRegs_base + **0x0024**	VIM D Swinging Buffer Full Flag Register	R/Clr	Section 6.9.4
VIMRegs_base + **0x0028**	VIM D Swinging Buffer Count Register	R.O.	Section 6.9.5
VIMRegs_base + **0x002C**	Reserved	–	–
VIMRegs_base + **0x0030**	FPGA C/D Interrupt Mask Register	R/W	Section 6.9.6
VIMRegs_base + **0x0034**	FPGA C/D Interrupt Flag Register	R/Clr	Section 6.9.7
VIMRegs_base + **0x0038**	FPGA C/D Interrupt Status Register	R.O.	Section 6.9.8
VIMRegs_base + **0x003C**	FPGA C/D Serial Port Control Register	R/W	Section 6.9.9

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

6.9.1 Swinging Buffer Control Registers

There are four Swinging Buffer Control Registers, one for each VIM interface, A, B, C, and D. This register selects the format of each VIM DPRAM Swinging Buffer and enables writing to this buffer. See Section 6.8.1 for description of Swinging Buffer operation. The following table shows the contents of this register. The subsections following the table describe the register bits.

Table 6–22: Swinging Buffer Control Register

For VIM A: R/W @ VIMRegs_base + **0x0000**
For VIM B: R/W @ VIMRegs_base + **0x0018**
For VIM C: R/W @ VIMRegs_base + **0x0000**
For VIM D: R/W @ VIMRegs_base + **0x0018**
See Table 6–20 for the register base address, VIMRegs_base

	D31 – D5	D4	D3	D2	D1	D0
Bit Name	Reserved	BUFFER WR EN	ADDR MODE3	ADDR MODE2	ADDR MODE1	ADDR MODE0
Function	Write with zeros, Mask when reading	0 = Disabled 1 = Enabled	Buffer Format Mode (see Section 6.9.1.2)			
All bits default to the logic '0' state at power on and reset.						

6.9.1.1 BUFFER WR EN Bit D4

This bit enables writing to the swinging buffer from the associated VIM BIFO. When this bit is cleared to logic '0' (its default state), writes are disabled. When this bit is set to logic '1', writes are enabled to this data buffer.

6.9.1.2 ADDR MODEn Bits D3 to D0

These four bits select the sequence that the swinging buffer is filled by incoming data. See Section 6.10 for description of the associated Buffer Format Modes. The settings for these four bits are:

0000 – Normal	address order: 11 10 9 8 7 6 5 4 3 2 1 0
0001 – Split halves	address order: 0 11 10 9 8 7 6 5 4 3 2 1
0010 – Split 4ths	address order: 1 0 11 10 9 8 7 6 5 4 3 2
0011 – Split 8ths	address order: 2 1 0 11 10 9 8 7 6 5 4 3
0100 – Split 16ths	address order: 3 2 1 0 11 10 9 8 7 6 5 4
0101 – Split 32ths	address order: 4 3 2 1 0 11 10 9 8 7 6 5
0110 – Split halves	address order: 1 11 10 9 8 7 6 5 4 3 2 0
0111 – Split 4ths	address order: 2 1 11 10 9 8 7 6 5 4 3 0
1000 – Split 8ths	address order: 3 2 1 11 10 9 8 7 6 5 4 0
1001 – Split 16ths	address order: 4 3 2 1 11 10 9 8 7 6 5 0
1010 to **1111** – (reserved)	

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

6.9.2 Swinging Buffer Length Registers

There are four Swinging Buffer Length Registers, one for each VIM interface, A, B, C, and D. This register programs the data length of each VIM DPRAM Swinging Buffer.

The following table shows the contents of this register. The paragraphs following this table describe these bits.

Table 6–23: Swinging Buffer Length Register

For VIM A: R/W @ VIMRegs_base + **0x0004**
For VIM B: R/W @ VIMRegs_base + **0x001C**
For VIM C: R/W @ VIMRegs_base + **0x0004**
For VIM D: R/W @ VIMRegs_base + **0x001C**
See Table 6–20 for the register base address, VIMRegs_base

	D31 – D12	D11 – D0
Bit Name	Reserved	BUFFER LENGTH [11:0]
Function	Write with zeros, Mask when reading	Swinging Buffer Length = BUFFER LENGTH [11:0] + 1
All bits default to the logic '0' state at power on and reset.		

The value written to this register programs the length of the associated VIM swinging buffer (swinging buffer length = BUFFER LENGTH [11:0] +1), in number of 32–bit words. Its value is less than or equal to 4096.

Once the swinging buffer fills to this point, the buffer automatically swaps, a BUFFER FULL Flag is asserted, and an interrupt or DMA request is generated to the PLX 9656 associated with this FPGA (A/B or C/D), depending on the settings in the associated Swinging Buffer Full Mask Register (Section 6.9.3).

See Section 6.8.1 for description of swinging buffer operation.

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

6.9.3 Swinging Buffer Full Mask Registers

There are four Swinging Buffer Full Mask Registers, one for each VIM interface, A, B, C, and D. Each register contains the mask bit for the VIM Swinging Buffer Full interrupt and two enable bits for PLX 9656 DMA request operation. The following table shows the contents of this register. The subsections following the table provide descriptions of the register bits.

Table 6–24: Swinging Buffer Full Mask Register

For VIM A: R/W @ VIMRegs_base + **0x0008**
For VIM B: R/W @ VIMRegs_base + **0x0020**
For VIM C: R/W @ VIMRegs_base + **0x0008**
For VIM D: R/W @ VIMRegs_base + **0x0020**
See Table 6–20 for the register base address, VIMRegs_base

	D31 – D3	D2 – D1	D0
Bit Name	Reserved	DMA REQUEST	BUFFER FULL INTERRUPT
Function	Write with zeros, Mask when reading	00 = Disable 01 = Buffer Full 10 = Direct BIFO Read 11 = Direct BIFO Write	0 = Disabled 1 = Enabled
All bits default to the logic '0' state at power on and reset.			

6.9.3.1 DMA REQUEST Bits D2 & D1

This bit enables DMA request operation for the associated PLX 9656. The settings for these two bits are:

00 – Disable
01 – Buffer Full (when buffer fills to progr. length, **Section 6.9.2**)
10 – Direct BIFO Read (when Almost Full, **Section 6.6.4**)
11 – Direct BIFO Write (when Almost Empty, **Section 6.6.4**)

6.9.3.2 BUFFER FULL INTERRUPT Bit D0

This bit enables an interrupt from the associated Swinging Buffer (A, B, C, or D) when that buffer is full. When this bit is cleared to logic '0' (its default state), an interrupt will not be asserted. When bit is set to logic '1', a local bus interrupt is asserted to the PLX 9656 associated with this VIM FPGA, Section 7.9, when the associated Swinging Buffer is full.

See Section 6.8.1 for description of swinging buffer operation.

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

6.9.4 Swinging Buffer Full Flag Registers

There are four Swinging Buffer Full Flag Registers, one each for VIM A, B, C, and D. Each register contains the flag bit for the VIM Swinging Buffer Full condition. Note that the Buffer Full Flag bit of each register is the same logic bit as the corresponding BUFFER FULL (A, B, C, or D) flag bit of the applica–ble FPGA Interrupt Flag Register, Section 6.9.7. The following table shows the contents of this register. The paragraphs following the table describe this bit.

Table 6–25: Swinging Buffer Full Flag Register

For VIM A: R/Clr @ VIMRegs_base + **0x000C**
For VIM B: R/Clr @ VIMRegs_base + **0x0024**
For VIM C: R/Clr @ VIMRegs_base + **0x000C**
For VIM D: R/Clr @ VIMRegs_base + **0x0024**
See Table 6–20 for the register base address, VIMRegs_base

	D31 – D1	D0
Bit Name	Reserved	BUFFER FULL FLAG
Function	Write with zeros, Mask when reading	Read/Clear (see below)

IMPORTANT! When reset, including power–up, the state of this register is unknown. You should clear this register before using it, by writing '1' to bit D0.

Read: When this bit reads logic '0' (its default state), there is no Buffer Full condition from the associated VIM Swinging Buffer. When this bit reads logic '1', a Buffer Full condition has been latched. A Buffer Full condition means that a swinging buffer half is full, or the buffer half pointer has swapped. The associated Swinging Buffer Count Register, Section 6.9.5, indicates the number of data samples in the full buffer.

Clear: This bit latches in response to a Buffer Full condition, and you must clear it to detect a subsequent Buffer Full. To clear this bit, write logic '1' to it. Clearing this bit also toggles the buffer half pointer in the swinging buffer, and clears the BUFFER FULL flag bit (D0 or D1) in the FPGA Interrupt Flag Regis–ter (Section 6.9.7). After writing a '1', the Buffer Full flag bit, however, remains set at '1' if the other buffer half is full. It only clears after the buffer is empty and no more data is available for reading. Note that if both halves of a swing–ing buffer are full, writing a '1' toggles the swinging buffer pointer, but the Buffer Full flag bit remains a '1' indicating that there is still another buffer half to read. After reading the second half, write a '1' to clear the flag bit.



Since this bit controls which half of the swinging buffer is accessible, do not clear it until the data in the buffer is read.

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

6.9.5 Swinging Buffer Count Registers

There are four Swinging Buffer Count Registers, one each for VIM A, B, C, and D. This read–only register indicates the actual number of data samples contained in the buffer half that is ready to be read.

The following table shows the contents of this register. The paragraphs following the table describe the register bits.

Table 6–26: Swinging Buffer Count Register

For VIM A: R.O. @ VIMRegs_base + **0x0010**
For VIM B: R.O. @ VIMRegs_base + **0x0028**
For VIM C: R.O. @ VIMRegs_base + **0x0010**
For VIM D: R.O. @ VIMRegs_base + **0x0028**
See Table 6–20 for the register base address, VIMRegs_base

	D31 – D13	D12 – D0
Bit Name	Reserved	COUNT [12:0]
Function	Mask when reading	Number of data sample in buffer = COUNT [12:0]

All bits default to the logic '0' state at power on and reset.

These bits are an indication of the actual number of 32–bit data samples contained in the associated VIM swinging buffer half that is ready to be read. It is valid only when the BUFFER FULL FLAG reads logic '1' (Section 6.9.4). Its value is less than or equal to 4096.

See Section 6.8.1 for description of swinging buffer operation.

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

6.9.6 FPGA Interrupt Mask Registers

There are two FPGA Interrupt Mask Registers, one for FPGA A/B and one for FPGA C/D. Each bit of this register enables or disables the generation of an interrupt for one of the FPGA functions.

The following table shows the contents of this register. The paragraphs following the table describe the use of these bits. Table 6–28, on the next page, describes the interrupt condition associated with each bit.

Table 6–27: FPGA Interrupt Mask Register
R/W @ VIMRegs_base + **0x0030**
See Table 6–20 for the register base address, VIMRegs_base

	D31 – D16							
Bit Name	Reserved							
	D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Bit Name	SP B(D) RCV F	SP B(D) RCV HF	SP B(D) RCV NE	SP B(D) XMIT NHF	SP B(D) XMIT E	SP A(C) RCV F	SP A(C) RCV HF	SP A(C) RCV NE
Function	0 = Disable Interrupt 1 = Enable Interrupt							
	D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1 – D0**	
Bit Name	SP A(C) XMIT NHF	SP A(C) XMIT E	Reserved	DCM	BUFFER OVFL B(D)	BUFFER OVFL A(C)	Reserved	
Function	0 = Disable Interrupt 1 = Enable Interrupt		Write zeros, Mask read	0 = Disable Interrupt 1 = Enable Interrupt			Write zeros, Mask read	
All bits default to the logic '0' state at power on and reset.								

Set a mask bit to logic '1' to enable an interrupt for the specified condition. When a mask bit is set to logic '1' and the corresponding status bit changes to active in the FPGA Interrupt Status Register, Section 6.9.8, an interrupt is asserted to the MPC745X or PLX 9656, depending on the type of interrupt.

- The Serial Port interrupts (D15 – D6) are routed to the MPC745X using the Board VIM Serial Port Interrupt Status Register, Section 3.8.14.

- The DCM and Buffer interrupts (D4 – D2) are routed to the PLX 9656 controlling this FPGA, Section 7.9.

Clear the mask bit to logic '0' to disable the interrupt (its default state).

The following two sections, FPGA Interrupt Flag Register, Section 6.9.7, and FPGA Interrupt Status Register, Section 6.9.8, contain related functions for these interrupts.

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

6.9.6 FPGA Interrupt Mask Registers (continued)

Table 6–28: FPGA Interrupt Registers Bits, VIM to Proc

Bit Name	Bit Position	Interrupt Function
SP B(D) RCV F	D15	This bit is associated with the VIM B (VIM D for FPGA C/D) Serial Port Receive FIFO Full flag, Section 6.7, Table 6–17.
SP B(D) RCV HF	D14	This bit is associated with the VIM B (VIM D for FPGA C/D) Serial Port Receive FIFO Half Full flag, Section 6.7, Table 6–17.
SP B(D) RCV NE	D13	This bit is associated with the VIM B (VIM D for FPGA C/D) Serial Port Receive FIFO Not Empty flag, Section 6.7, Table 6–17.
SP B(D) XMIT NHF	D12	This bit is associated with the VIM B (VIM D for FPGA C/D) Serial Port Transmit FIFO Not Half Full flag, Section 6.7, Table 6–17.
SP B(D) XMIT E	D11	This bit is associated with the VIM B (VIM D for FPGA C/D) Serial Port Transmit FIFO Empty flag, Section 6.7, Table 6–17.
SP A(C) RCV F	D10	This bit is associated with the VIM A (VIM C for FPGA C/D) Serial Port Receive FIFO Full flag, Section 6.7, Table 6–17.
SP A(C) RCV HF	D9	This bit is associated with the VIM A (VIM C for FPGA C/D) Serial Port Receive FIFO Half Full flag, Section 6.7, Table 6–17.
SP A(C) RCV NE	D8	This bit is associated with the VIM A (VIM C for FPGA C/D) Serial Port Receive FIFO Not Empty flag, Section 6.7, Table 6–17.
SP A(C) XMIT NHF	D7	This bit is associated with the VIM A (VIM C for FPGA C/D) Serial Port Transmit FIFO Not Half Full flag, Section 6.7, Table 6–17.
SP A(C) XMIT E	D6	This bit is associated with the VIM A (VIM C for FPGA C/D) Serial Port Transmit FIFO Empty flag, Section 6.7, Table 6–17.
DCM	D4	This bit is associated with the VIM A/B (or C/D) FPGA clock (DCM) not locked condition.
BUFFER OVFL B(D) BUFFER OVFL A(C)	D3 D2	These bits are associated with a data overflow interrupt from the associated VIM interface Swinging Buffer. The interrupt indicates that both halves of the respective buffer are full.
BUFFER FULL B(D) BUFFER FULL A(C)	D1 D0	These bits are associated with a Swinging Buffer Full interrupt from the associated VIM interface Swinging Buffer. Note that the interrupt mask bit for these functions are located in the associated Swinging Buffer Full Mask Register, Section 6.9.3. Note also that the flag bits are the same logic as the BUFFER FULL FLAG bits in the Swinging Buffer Full Flag Register, Section 6.9.4.
Note: These bit assignments and definitions apply to FPGA Interrupt Mask Register, Section 6.9.6, FPGA Interrupt Flag Register, Section 6.9.7, and FPGA Interrupt Status Register, Section 6.9.8		

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

6.9.7 FPGA Interrupt Flag Registers

There are two FPGA Interrupt Flag Registers, one for FPGA A/B and one for FPGA C/D. Each bit in this register is a read/clear bit that is latched in response to an interrupt occurrence.

The following table shows the contents of this register. The paragraphs following the table describe the use of these bits. Table 6–28, on the prior page, describes the interrupt condition associated with each bit.

Table 6–29: FPGA Interrupt Flag Register
R/Clr @ VIMRegs_base + **0x0034**
See Table 6–20 for the register base address, VIMRegs_base

	D31 – D16							
Bit Name	Reserved							
	D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Bit Name	SP B(D) RCV F	SP B(D) RCV HF	SP B(D) RCV NE	SP B(D) XMIT NHF	SP B(D) XMIT E	SP A(C) RCV F	SP A(C) RCV HF	SP A(C) RCV NE
Function	**Read:** 0 = No interrupt 1 = Interrupt latched **Clear:** 0 = Normal operation 1 = Clear latch							
	D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Bit Name	SP A(C) XMIT NHF	SP A(C) XMIT E	Reserved	DCM	BUFFER OVFL B(D)	BUFFER OVFL A(C)	BUFFER FULL B(D)	BUFFER FULL A(C)
Function	**Read:** 0 = No inter 1 = Interrupt **Clear:** 0 = Operate 1 = Clear		Write zeros, Mask read	**Read:** 0 = No interrupt 1 = Interrupt latched **Clear:** 0 = Normal operation 1 = Clear latch			**Read Only:** 0 = No interrupt 1 = Interrupt latched (See **Note** next page)	

IMPORTANT! When reset, including power–up, the state of this register is unknown. You should clear this register before using it, by writing '1's into all bits.

When any bit in the FPGA Interrupt Status Register (Section 6.9.8) changes to logic '1', the corresponding bit in this register also changes to logic '1'. However, when a bit in the FPGA Interrupt Status Register clears (changes from logic '1' to logic '0'), the corresponding latched bit in this register does not clear, but remains at the logic '1' state.

Each bit in this register is a read/clear bit. The Read and Clear functions are described on the following page.

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

6.9.7 FPGA Interrupt Flag Registers (continued)

Read – When read as logic '1', an interrupt has occurred from the associated FPGA interrupt source. When read as logic '0', the associated FPGA interrupt source is not active. A flag bit remains latched until you clear it (see **Clear** on the following page).

Clear – Since these bits latch in response to an interrupt, to detect subsequent interrupts, you must clear the bits in this register. To clear any bit in this register that is set to logic '1', you must write a '1' to that bit to clear the bit (except bits D0 and D1, see **Note** below).



1) The BUFFER FULL flag bits (D0 and D1) are the same logic bits as the BUFFER FULL FLAG bits of the associated Swinging Buffer Full Interrupt Flag Registers, Section 6.9.4 (that is, BUFFER FULL A in this register is the same logic condition as BUFFER FULL FLAG of Swinging Buffer Full Interrupt Flag Register for VIM A).

2) The BUFFER FULL latched bits cannot be cleared by this register—you must use the Swinging Buffer Full Interrupt Flag Registers, Section 6.9.4, to clear these latched bits.

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

6.9.8 FPGA Interrupt Status Registers

There are two FPGA Interrupt Status Registers, one for FPGA A/B and one for FPGA C/D. Each bit in this read–only register changes whenever the associated FPGA interrupt source indication changes.

The following table shows the contents of this register. The paragraphs following the table describe the use of these bits. Table 6–28 describes the interrupt condition associated with each bit.

Table 6–30: FPGA Interrupt Status Register
R.O. @ VIMRegs_base + **0x0038**
See Table 6–20 for the register base address, VIMRegs_base

	D31 – D16							
Bit Name	Reserved							
	D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Bit Name	SP B(D) RCV F	SP B(D) RCV HF	SP B(D) RCV NE	SP B(D) XMIT NHF	SP B(D) XMIT E	SP A(C) RCV F	SP A(C) RCV HF	SP A(C) RCV NE
Function	0 = No Interrupt 1 = Interrupt							
	D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Bit Name	SP A(C) XMIT NHF	SP A(C) XMIT E	Reserved	DCM	BUFFER OVFL B(D)	BUFFER OVFL A(C)	BUFFER FULL B(D)	BUFFER FULL A(C)
Function	0 = No Interrupt 1 = Interrupt		Mask read	0 = No Interrupt 1 = Interrupt				
All bits default to the logic '0' state at power on and reset.								

A status bit changes to logic '1' whenever the interrupt indication occurs. When any bit in this register changes to logic '1', the corresponding bit in the FPGA Interrupt Flag Register, Section 6.9.7, is set to logic '1'. In addition, when either BUFFER FULL status bit changes to logic '1', the BUFFER FULL FLAG bit in the corresponding Swinging Buffer Full Interrupt Flag Register, Section 6.9.4, is also set to logic '1'.

If the corresponding bit in the FPGA Interrupt Mask Register, Section 6.9.6, or the corresponding Swinging Buffer Full Mask Register, Section 6.9.3, is set to logic '1', then a status bit transition from logic '0' to logic '1' asserts an interrupt. For bits D15 to D6, this interrupt is routed to the MPC745X processor using the VIM Serial Port Interrupt Status Register, Section 3.8.14. For bits D4 to D0, the interrupt signals are routed to this FPGA's PLX 9656, Section 7.9. See Figure 7–3 in Section 7.9 for illustration of the PLX 9656 interrupt routing.

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

6.9.9 FPGA Serial Port Control Registers

There are two FPGA Serial Port Control Registers, one for FPGA A/B and one for FPGA C/D. Each register contains controls bits for the associated VIM module's serial ports.

The following table shows the contents of the registers. The subsections following the table provide descriptions of the register bits.

Table 6–31: FPGA Serial Port Control Register
R/W @ VIMRegs_base + **0x003C**
See Table 6–20 for the register base address, VIMRegs_base

	D31 – D15	D14	D13	D12 – D11	D10	D9	D8
Bit Name	Reserved	RCV ENABLE B(D)	XMIT ENABLE B(D)	WORD SIZE B(D)	XMIT CLK SRC B(D)	RCV RESET B(D)	XMIT RESET B(D)
Function	Write zeros, Mask read	0 = Disabled 1 = Enabled		00 = 8-bit word 01 = 16-bit word 10 or 11 = 32-bit word	0 = 33 MHz 1 = VIM SCLK	0 = Normal operation 1 = Reset	
	D7	**D6**	**D5**	**D4 – D3**	**D2**	**D1**	**D0**
Bit Name	Reserved	RCV ENABLE A(C)	XMIT ENABLE A(C)	WORD SIZE A(C)	XMIT CLK SRC A(C)	RCV RESET A(C)	XMIT RESET A(C)
Function	Write zeros, Mask read	0 = Disabled 1 = Enabled		00 = 8-bit word 01 = 16-bit word 10 or 11 = 32-bit word	0 = 33 MHz 1 = VIM SCLK	0 = Normal operation 1 = Reset	
All bits default to the logic '0' state at power on and reset.							

6.9.9.1 RCV ENABLE B(D) Bit D14

This bit enables receive operation for Serial Port B (Serial Port D for FPGA C/D). Set this bit to logic '1' to enable receives. Clear this bit to logic '0' (its default state) to disable receives.

6.9.9.2 XMIT ENABLE B(D) Bit D13

This bit enables transmit operation for Serial Port B (Serial Port D for FPGA C/D). Set this bit to logic '1' to enable transmits. Clear this bit to logic '0' (its default state) to disable transmits.

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

6.9.9 FPGA Serial Port Control Registers (continued)

6.9.9.3 WORD SIZE B(D) Bits D12–D11

These two bits set the word size for Serial Port B (Serial Port D for FPGA C/D). The settings for these two bits are:

00 – 8–bit word
01 – 16–bit word
10 or **11** – 32–bit word

6.9.9.4 XMIT CLK SRC B(D) Bit D10

This bit selects the clock for Serial Port B (Serial Port D for FPGA C/D). Set this bit to logic '1' to use the associated VIM interface SCLK. Clear this bit to logic '0' (its default state) to set the clock to 33 MHz.

6.9.9.5 RCV RESET B(D) Bit D9

This bit resets Serial Port B (Serial Port D for FPGA C/D) receive operation. Set this bit to logic '1' to hold the receive port in reset. Clear this bit to logic '0' (its default state) to resume normal operation of the receive port.

6.9.9.6 XMIT RESET B(D) Bit D8

This bit resets Serial Port B (Serial Port D for FPGA C/D) transmit operation. Set this bit to logic '1' to hold the transmit port in reset. Clear this bit to logic '0' (its default state) to resume normal operation of the transmit port.

6.9.9.7 RCV ENABLE A(C) Bit D6

This bit enables receive operation for Serial Port A (Serial Port C for FPGA C/D). Set this bit to logic '1' to enable receives. Clear this bit to logic '0' (its default state) to disable receives.

6.9.9.8 XMIT ENABLE A(C) Bit D5

This bit enables transmit operation for Serial Port A (Serial Port C for FPGA C/D). Set this bit to logic '1' to enable transmits. Clear this bit to logic '0' (its default state) to disable transmits.

6.9 VIM FPGA Control/Status Registers (VIM to Proc.) (continued)

6.9.9 FPGA Serial Port Control Registers (continued)

6.9.9.9 WORD SIZE A(C) Bits D4–D3

These two bits set the word size for Serial Port A (Serial Port C for FPGA C/D). The settings for these two bits are:

00 – 8–bit word
01 – 16–bit word
10 or **11** – 32–bit word

6.9.9.10 XMIT CLK SRC A(C) Bit D2

This bit selects the clock for Serial Port A (Serial Port C for FPGA C/D). Set this bit to logic '1' to use the associated VIM interface SCLK. Clear this bit to logic '0' (its default state) to set the clock to 33 MHz.

6.9.9.11 RCV RESET A(C) Bit D1

This bit resets Serial Port A (Serial Port C for FPGA C/D) receive operation. Set this bit to logic '1' to hold the receive port in reset. Clear this bit to logic '0' (its default state) to resume normal operation of the receive port.

6.9.9.12 XMIT RESET A(C) Bit D0

This bit resets Serial Port A (Serial Port C for FPGA C/D) transmit operation. Set this bit to logic '1' to hold the transmit port in reset. Clear this bit to logic '0' (its default state) to resume normal operation of the transmit port.

6.10 FPGA Buffer Format Modes

In the VIM to Processor FPGA configuration, a portion of each FPGA can be configured as dual−ported random access memory (DPRAM). Under this configuration, there are four 2 x 4k x 32 DPRAM Swinging Buffers, one for each VIM interface (A, B, C, and D). Refer to Section 6.8.1 for description of the swinging buffer operation.

The Model 4205 has ten Buffer Format Modes for filling the DPRAM Swinging Buffers. The Swinging Buffer Control Register, Section 6.9.1.2, sets the Buffer Format Mode, and enables writing to the associated FPGA DPRAM Swinging Buffer (A, B, C, or D). Each Swinging Buffer (A, B, C, or D) can be set individually.

6.10.1 Normal Buffer Mode (code 0000)

This is the default Buffer Format Mode. In this mode, data flows incremen−tally into the DPRAM buffers in their exact format—the samples are not divided into any blocks as in the other buffer modes. The data configuration for each buffer appears as follows:

Buffer
Sample T_0
Sample T_1
Sample T_2
Sample T_3
.
.
.

6.10.2 Split Buffer Mode x2 (code 0001)

In this mode, each DPRAM buffer is split into two blocks. Input data is inter−leaved between the two blocks, one data sample at a time, as follows:

Block 1	Block 2
Sample T_0	Sample T_1
Sample T_2	Sample T_3
Sample T_4	Sample T_5
Sample T_6	Sample T_7
.	.
.	.

6.10 FPGA Buffer Format Modes (continued)

6.10.3 Split Buffer Mode x4 (code 0010)

In this mode, each DPRAM buffer is split into four blocks. Input data is inter–leaved between the four blocks, one data sample at a time, as follows:

Block 1	Block 2	Block 3	Block 4
Sample T_0	Sample T_1	Sample T_2	Sample T_3
Sample T_4	Sample T_5	Sample T_6	Sample T_7
Sample T_8	Sample T_9	Sample T_{10}	Sample T_{11}
Sample T_{12}	Sample T_{13}	Sample T_{14}	Sample T_{15}
.	.	.	.
.	.	.	.

6.10.4 Split Buffer Mode x8 (code 0011)

In this mode, each DPRAM buffer is split into eight blocks. Input data is inter–leaved between the eight blocks, one data sample at a time, as follows:

Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7	Block 8
Sample T_0	Sample T_1	Sample T_2	Sample T_3	Sample T_4	Sample T_5	Sample T_6	Sample T_7
Sample T_8	Sample T_9	Sample T_{10}	Sample T_{11}	Sample T_{12}	Sample T_{13}	Sample T_{14}	Sample T_{15}
Sample T_{16}	Sample T_{17}	Sample T_{18}	Sample T_{19}	Sample T_{20}	Sample T_{21}	Sample T_{22}	Sample T_{23}
Sample T_{24}	Sample T_{25}	Sample T_{26}	Sample T_{27}	Sample T_{28}	Sample T_{29}	Sample T_{30}	Sample T_{31}
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

6.10 FPGA Buffer Format Modes (continued)

6.10.5 Split Buffer Mode x16 (code 0100)

In this mode, each DPRAM buffer is split into 16 blocks. Input data is inter–leaved between the blocks, one data sample at a time, as follows:

Bk 1	Bk 2	Bk 3	Bk 4	Bk 5	Bk 6	Bk 7	Bk 8	Bk 9	Bk 10	Bk 11	Bk 12	Bk 13	Bk 14	Bk 15	Bk 16
T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}
T_{16}	T_{17}	T_{18}	T_{19}	T_{20}	T_{21}	T_{22}	T_{23}	T_{24}	T_{25}	T_{26}	T_{27}	T_{28}	T_{29}	T_{30}	T_{31}
T_{32}	T_{33}	T_{34}	T_{35}	T_{36}	T_{37}	T_{38}	T_{39}	T_{40}	T_{41}	T_{42}	T_{43}	T_{44}	T_{45}	T_{46}	T_{47}
T_{48}	T_{49}	T_{50}	T_{51}	T_{52}	T_{53}	T_{54}	T_{55}	T_{56}	T_{57}	T_{58}	T_{59}	T_{60}	T_{61}	T_{62}	T_{63}
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

6.10.6 Split Buffer Mode x32 (code 0101)

In this mode, each DPRAM buffer is split into 32 blocks. Input data is inter–leaved between the blocks, one data sample at a time, as follows:

Bk 1	Bk 2	Bk 3	Bk 4	Bk 5	Bk 6	Bk 7	Bk 8	Bk 9	Bk 10	Bk 11	Bk 12	Bk 13	Bk 14	Bk 15	Bk 16	Bk 17	Bk 18	Bk 19	Bk 20	Bk 21	Bk 22	Bk 23	Bk 24	Bk 25	Bk 26	Bk 27	Bk 28	Bk 29	Bk 30	Bk 31	Bk 32
T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}	T_{19}	T_{20}	T_{21}	T_{22}	T_{23}	T_{24}	T_{25}	T_{26}	T_{27}	T_{28}	T_{29}	T_{30}	T_{31}
T_{32}	T_{33}	T_{34}	T_{35}	T_{36}	T_{37}	T_{38}	T_{39}	T_{40}	T_{41}	T_{42}	T_{43}	T_{44}	T_{45}	T_{46}	T_{47}	T_{48}	T_{49}	T_{50}	T_{51}	T_{52}	T_{53}	T_{54}	T_{55}	T_{56}	T_{57}	T_{58}	T_{59}	T_{60}	T_{61}	T_{62}	T_{63}
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

6.10 FPGA Buffer Format Modes (continued)

6.10.7 Data Unpacking Mode, Split Buffer x2 (code 0110)

In this mode, two consecutive data samples enter an individual buffer block. Each DPRAM buffer is split into two blocks. Input data is interleaved between the two buffer blocks as follows:

Block 1	Block 2
Sample T_0	Sample T_2
Sample T_1	Sample T_3
Sample T_4	Sample T_6
Sample T_5	Sample T_7
.	.
.	.

6.10.8 Data Unpacking Mode, Split Buffer x4 (code 0111)

In this mode, two consecutive data samples enter an individual buffer block. Each DPRAM buffer is split into four blocks. Input data is interleaved between the four buffer blocks as follows:

Block 1	Block 2	Block 3	Block 4
Sample T_0	Sample T_2	Sample T_4	Sample T_6
Sample T_1	Sample T_3	Sample T_5	Sample T_7
Sample T_8	Sample T_{10}	Sample T_{12}	Sample T_{14}
Sample T_9	Sample T_{11}	Sample T_{13}	Sample T_{15}
.	.	.	.
.	.	.	.

6.10 FPGA Buffer Format Modes (continued)

6.10.9 Data Unpacking Mode, Split Buffer x8 (code 1000)

In this mode, two consecutive data samples enter an individual buffer block. Each DPRAM buffer is split into eight blocks. Input data is interleaved between the eight buffer blocks as follows:

Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7	Block 8
Sample T_0	Sample T_2	Sample T_4	Sample T_6	Sample T_8	Sample T_{10}	Sample T_{12}	Sample T_{14}
Sample T_1	Sample T_3	Sample T_5	Sample T_7	Sample T_9	Sample T_{11}	Sample T_{13}	Sample T_{15}
Sample T_{16}	Sample T_{18}	Sample T_{20}	Sample T_{22}	Sample T_{24}	Sample T_{26}	Sample T_{28}	Sample T_{30}
Sample T_{17}	Sample T_{19}	Sample T_{21}	Sample T_{23}	Sample T_{25}	Sample T_{27}	Sample T_{29}	Sample T_{31}
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

6.10.10 Data Unpacking Mode, Split Buffer x16 (code 1001)

In this mode, two consecutive data samples enter an individual buffer block. Each DPRAM buffer is split into 16 blocks. Input data is interleaved between the 16 buffer blocks as follows:

Bk 1	Bk 2	Bk 3	Bk 4	Bk 5	Bk 6	Bk 7	Bk 8	Bk 9	Bk 10	Bk 11	Bk 12	Bk 13	Bk 14	Bk 15	Bk 16
T_0	T_2	T_4	T_6	T_8	T_{10}	T_{12}	T_{14}	T_{16}	T_{18}	T_{20}	T_{22}	T_{23}	T_{26}	T_{28}	T_{30}
T_1	T_3	T_5	T_7	T_9	T_{11}	T_{13}	T_{15}	T_{17}	T_{19}	T_{21}	T_{23}	T_{25}	T_{27}	T_{29}	T_{31}
T_{32}	T_{34}	T_{36}	T_{38}	T_{40}	T_{42}	T_{44}	T_{46}	T_{47}	T_{50}	T_{52}	T_{54}	T_{56}	T_{58}	T_{60}	T_{62}
T_{33}	T_{35}	T_{37}	T_{39}	T_{41}	T_{43}	T_{45}	T_{47}	T_{49}	T_{51}	T_{53}	T_{55}	T_{57}	T_{59}	T_{61}	T_{63}
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

6.11 **VIM FPGA Control/Status Registers** (VIM to VIM)

The VIM to VIM FPGA configuration has two sets of 32−bit registers to control the VIM BIFO/FPGA data interface, one set for the VIM A/B FPGA and one for the VIM C/D FPGA. You can access these registers when you select either the FPGA or PCI Interface VIM BIFO Path in the VIM Interface BIFO Control Register (Section 6.4.3). However, you must select the FPGA path before starting any VIM to VIM data transfers.

Table 6−33, on the next page, lists the addresses of these registers for both FPGAs. The address of each register is shown as offset relative to a base address for each FPGA ('VIMRegs_base') from the MPC745X or PCI bus, indicated in Table 6−32, below. In addition, when using PLX 9656 DMA operation, an address is provided for access from the associated PLX local bus. The referenced subsections following these tables provide information about the format and programming of each register.

Table 6−32: VIM FPGA Control/Status Registers Base Address

FPGA		MPC745X or PCI Bus	PLX 9656 Local Bus
VIM A/B	VIMRegs_base =	**0x4180 0000**	**0x80 0000**
VIM C/D		**0x4280 0000**	**0x80 0000**

To access a VIM FPGA control/status register, first determine the base address of the appropriate VIM FPGA for the control device you are using from the above table, then add the offset for the specific register you wish to use from Table 6−33 on the next page.

For example: To access the VIM A Data Transfer Control Register from the MPC745X, you would access that register at MPC745X memory address **0x4180 0000** (MPC745X VIM A/B FPGA base address [**0x4180 0000**] + register offset [**0x0000**]).

NOTE: In the following memory map, some registers apply to a specific VIM module data interface (A, B, C, or D)—there are four separate registers for each of these functions, one register per VIM interface. Other registers apply to both VIM interfaces in a specific FPGA (A and B, or C and D)—there are two registers for each of these functions, one register per FPGA.

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)

Table 6–33: VIM FPGA Control/Status Register Addresses, VIM–VIM			
VIM A/B FPGA			
Base Address	**MPC745X or PCI Bus**	**PLX 9656**	
VIMRegs_base =	**0x4180 0000**	**0x80 0000**	
Address Offset	**Register**	**Access**	**Information**
VIMRegs_base + **0x0000**	VIM A Data Transfer Control Register	R/W	Section 6.11.1
VIMRegs_base + **0x0004**	VIM A Data Transfer Status Register	R.O.	Section 6.11.2
VIMRegs_base + **0x0008** to + **0x0017**	Reserved	–	–
VIMRegs_base + **0x0018**	VIM B Data Transfer Control Register	R/W	Section 6.11.1
VIMRegs_base + **0x001C**	VIM B Data Transfer Status Register	R.O.	Section 6.11.2
VIMRegs_base + **0x0020** to + **0x002F**	Reserved	–	–
VIMRegs_base + **0x0030**	FPGA A/B Interrupt Mask Register	R/W	Section 6.11.3
VIMRegs_base + **0x0034**	FPGA A/B Interrupt Flag Register	R/Clr	Section 6.11.4
VIMRegs_base + **0x0038**	FPGA A/B Interrupt Status Register	R.O.	Section 6.11.5
VIMRegs_base + **0x003C**	FPGA A/B Serial Port Control Register	R/W	Section 6.11.6
VIM C/D FPGA			
Base Address	**MPC745X or PCI Bus**	**PLX 9656**	
VIMRegs_base	**0x4280 0000**	**0x80 0000**	
Address Offset	**Register**	**Access**	**Information**
VIMRegs_base + **0x0000**	VIM C Data Transfer Control Register	R/W	Section 6.11.1
VIMRegs_base + **0x0004**	VIM C Data Transfer Status Register	R.O.	Section 6.11.2
VIMRegs_base + **0x0008** to + **0x0017**	Reserved	–	–
VIMRegs_base + **0x0018**	VIM D Data Transfer Control Register	R/W	Section 6.11.1
VIMRegs_base + **0x001C**	VIM D Data Transfer Status Register	R.O.	Section 6.11.2
VIMRegs_base + **0x0020** to + **0x002F**	Reserved	–	–
VIMRegs_base + **0x0030**	FPGA C/D Interrupt Mask Register	R/W	Section 6.11.3
VIMRegs_base + **0x0034**	FPGA C/D Interrupt Flag Register	R/Clr	Section 6.11.4
VIMRegs_base + **0x0038**	FPGA C/D Interrupt Status Register	R.O.	Section 6.11.5
VIMRegs_base + **0x003C**	FPGA C/D Serial Port Control Register	R/W	Section 6.11.6

NOTE: The VIM to VIM interfaces are configured as fixed pairs—that is, VIM A BIFO is paired with VIM C BIFO, and VIM B BIFO is paired with VIM D BIFO. For each pair of VIM to VIM transfers, FPGA A/B is the Master—only the Master can set the data transfer direction (XFER DIR). See VIM to VIM Data Transfer Operation, Section 6.12, for description of these transfers.

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)

6.11.1 Data Transfer Control Registers

There are four Data Transfer Control Registers, one for each VIM interface, A, B, C, and D. This register sets the direction and mode of operation of the data transfer between the selected VIM BIFOs. See VIM to VIM Data Transfer Operation, Section 6.12, for description of VIM to VIM data transfers.

The following table shows the contents of this register. The subsections following the table describe the register bits.

Table 6–34: Data Transfer Control Register
For VIM A: R/W @ VIMRegs_base + **0x0000**
For VIM B: R/W @ VIMRegs_base + **0x0018**
For VIM C: R/W @ VIMRegs_base + **0x0000**
For VIM D: R/W @ VIMRegs_base + **0x0018**
See Table 6–32 for the register base address, VIMRegs_base

	D31 – D4	D3	D2	D1	D0
Bit Name	Reserved	XFER FLOW EN	SUSP RESPONSE	MODE	XFER DIR
Function	Write with zeros, Mask when reading	0 = Disabled 1 = Enabled	0 = Ignore 1 = Suspend	0 = Mode 0 1 = Mode 1	0 = Transmit 1 = Receive
All bits default to the logic '0' state at power on and reset.					

6.11.1.1 XFER FLOW EN Bit D4

This bit enables data transfers from or to the selected VIM BIFO. Set this bit to logic '1' to enable a data transfer from or to the selected VIM BIFO, in accordance with the direction and mode specified in the other bits of this register. Clear this bit to logic '0' (its default state) to disable data transfers.

6.11.1.2 SUSP RESPONSE Bit D3

This bit enables a Suspend of the data transfer for the selected VIM BIFO. This bit applies only to the FPGA of the pair that is set for Transmit by the XFER DIR bit (Section 6.11.1.4). Set this bit to logic '1' to enable response to a Suspend signal from the other FPGA. When Enabled, the Transmit FPGA will suspend the VIM BIFO data transfer when a Suspend signal is received from the other (Receive) FPGA and will resume the transfer when that Suspend clears. Clear this bit to logic '0' (its default state) to ignore a Suspend signal from the other FPGA. See VIM to VIM Data Transfer Operation, Section 6.12, for description of the Suspend operation in each operating mode.

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)

6.11.1 Data Transfer Control Registers (continued)

6.11.1.3 MODE Bit D2

This bit selects the mode of data transfer for the selected VIM BIFO. Clear this bit to logic '0' (its default state) for Mode 0, or set it to logic '1' for Mode 1. Each mode operates differently depending on the setting of XFER DIR, below. See Section 6.12 for further description of these modes.

XFER DIR	MODE	Operation
Transmit	Mode 0	The FPGA starts reading data from the VIM BIFO when the BIFO is Not Almost Empty, and stops reading data when the BIFO is Almost Empty. The FPGA transmits the data to the other FPGA as long as there is no Suspend signal from that FPGA. If a Suspend occurs and SUSP EN is enabled, the FPGA stops the transfer until the Suspend clears, and resumes when Suspend is clear.
	Mode 1	The FPGA starts reading data from the VIM BIFO when the BIFO is Almost Full, and stops reading data when the BIFO is Almost Empty. The FPGA transmits the data to the other FPGA as long as there is no Suspend signal from that FPGA. If a Suspend occurs and SUSP EN is enabled, the FPGA stops the transfer until the Suspend clears, and resumes when Suspend is clear.
Receive	Mode 0	The FPGA receives data from the other FPGA. The FPGA starts writing data to the VIM BIFO when the BIFO is Not Almost Full. When the BIFO is Almost Full, the FPGA issues a Suspend and stops writing data. After a Suspend, the FPGA resumes writing and clears Suspend when the BIFO is Not Almost Full.
	Mode 1	The FPGA receives data from the other FPGA. The FPGA starts writing data to the VIM BIFO when the BIFO is Almost Empty. When the BIFO is Almost Full, the FPGA issues a Suspend and stops writing data. After a Suspend, the FPGA resumes writing and clears Suspend when the BIFO is Almost Empty.

6.11.1.4 XFER DIR Bit D1

This bit selects the direction of the data transfer between FPGAs for the selected VIM BIFO. This bit can only be set by the FPGA Master (FPGA A/B) for each VIM BIFO pair (that is, use the VIM A register for the VIM A – VIM C pair, and use the VIM B register for the VIM B – VIM D pair).

Set this bit to logic '1' to receive data from the other FPGA (C/D) and write to the selected VIM BIFO (A or B). Clear this bit to logic '0' (its default state) to read the selected VIM BIFO (A or B) and transmit the data to the other FPGA (C/D).

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)

6.11.2 Data Transfer Status Registers

There are four Data Transfer Status Registers, one for each VIM interface, A, B, C, and D. This read–only register reports the current status of the data transfer between the VIM BIFO pairs. Most of the bits in this register depend on the direction of the data transfer between FPGAs for the selected VIM BIFO, as set by XFER DIR in the Data Transfer Control Register, Section 6.11.1.4. See VIM to VIM Data Transfer Operation, Section 6.12, for description of these data transfers.

The following table shows the contents of this register. The paragraphs following this table describe these bits.

Table 6–35: Data Transfer Status Register

For VIM A: R.O. @ VIMRegs_base + **0x0004**
For VIM B: R.O. @ VIMRegs_base + **0x001C**
For VIM C: R.O. @ VIMRegs_base + **0x0004**
For VIM D: R.O. @ VIMRegs_base + **0x001C**
See Table 6–20 for the register base address, VIMRegs_base

	D31 – D12	D11	D10	D9	D8
Bit Name	Reserved	$\overline{\text{DATA VALID}}$	$\overline{\text{SUSP ACK IN}}$	$\overline{\text{SUSP ACK OUT}}$	$\overline{\text{SUSP IN}}$
Function	Write with zeros, Mask when reading	0 = Valid 1 = No Data	0 = Acknowl. 1 = No Ack	0 = Acknowl. 1 = No Ack	0 = Suspend 1 = No Susp.

	D7	D6	D5	D4	D3 – D0
Bit Name	$\overline{\text{SUSP OUT}}$	DIR OUT	BIFO DIR	DIR MASTER	PRESENT STATE
Function	0 = Suspend 1 = No Susp.	0 = Receive 1 = Transmit	0 = Read 1 = Write	0 = Slave 1 = Master	Current state machine state

When reset, including power–up, the state of this register is unknown.

NOTE: Most of these status bits are used only for factory test purposes. In addition, many of these signals are highly transient, and may not be detectable by the processor in normal operation.

6.11.2.1 $\overline{\text{DATA VALID}}$ Bit D11

This bit indicates whether data is valid for writing to the selected VIM BIFO. This bit applies only to the FPGA that is set as the receiver of a VIM pair (see XFER DIR, Section 6.11.1.4). When this bit is read as logic '1', there is no data available from the transmitter. When read as logic '0', data from the transmit FPGA is valid and can be written to the selected VIM BIFO.

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)

6.11.2 Data Transfer Status Registers (continued)

6.11.2.2 $\overline{\text{SUSP ACK IN}}$ Bit D10

This bit indicates whether or not the receiver has received an acknowledge to its Suspend signal from the transmitter. This bit applies only for the FPGA that is set as the receiver of a VIM pair (see XFER DIR, Section 6.11.1.4). When this bit is read as logic '1', the receiver has no acknowledge. When read as logic '0', the receiver has received an acknowledge for its Suspend.

6.11.2.3 $\overline{\text{SUSP ACK OUT}}$ Bit D9

This bit indicates whether the transmitter has acknowledged a Suspend signal from the receiver for the VIM BIFO transfer. This bit applies only for the FPGA that is set as the transmitter of a VIM pair (see XFER DIR, Section 6.11.1.4). When this bit is read as logic '1', the transmitter has not acknowledged a Suspend. When read as logic '0', the transmitter has acknowledged a Suspend.

6.11.2.4 $\overline{\text{SUSP IN}}$ Bit D8

This bit indicates whether or not the transmitter has received a Suspend signal from the receiver for the VIM BIFO transfer. This bit applies only for the FPGA that is set as the transmitter of a VIM pair (see XFER DIR, Section 6.11.1.4). When this bit is read as logic '1', the transmitter has received no Suspend. When read as logic '0', the transmitter has received a Suspend.

6.11.2.5 $\overline{\text{SUSP OUT}}$ Bit D7

This bit indicates whether the receiver has sent a Suspend signal for the VIM BIFO transfer. This bit applies only for the FPGA that is set as the receiver of a VIM pair (see XFER DIR, Section 6.11.1.4). When this bit is read as logic '1', the receiver has not sent a Suspend. When read as logic '0', the receiver has sent a Suspend.

NOTE: For the above four status bits, the typical Suspend sequence would result in each status being activated in the following order (see Section 6.12 for description of data transfer and suspend):

1) If a Suspend is needed, the receiver sets SUSP OUT and, in turn, the transmitter receives the SUSP IN.

2) After receiving the Suspend, the transmitter sets SUSP ACK OUT and, in turn, the receiver receives the SUSP ACK IN.

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)

6.11.2 Data Transfer Status Registers (continued)

6.11.2.6 DIR OUT Bit D6

This bit indicates the direction of data transfer (XFER DIR) set by FPGA A/B, which is being sent to FPGA C/D for the selected VIM BIFO. This bit is used only as an internal check of the transfer direction setting (see XFER DIR, Section 6.11.1.4). The DIR OUT bit value is inverted from the XFER DIR bit value. When this bit is read as logic '1', the associated FPGA is set to read the selected VIM BIFO and transmit the data to the other FPGA's paired VIM BIFO. When it is read as logic '0', the associated FPGA is set to receive data from the other FPGA's paired VIM BIFO and write to the selected VIM BIFO.

For example, if XFER DIR for VIM A is set to logic '1' (VIM A Receive), DIR OUT for VIM A will read logic '0' and DIR OUT for VIM C will read logic '1'.

6.11.2.7 BIFO DIR Bit D5

This bit indicates the current direction of the data read/write for the selected VIM BIFO. When this bit is read as logic '0', the associated FPGA is set to read the selected VIM BIFO and transmit data to the other FPGA. When this bit is logic '1', the associated FPGA is set to write to the selected VIM BIFO from data received from the other FPGA.

6.11.2.8 DIR MASTER Bit D4

This bit indicates the FPGA that is controlling this BIFO data transfer. (For each pair of VIM to VIM transfers, FPGA A/B is the Master.) When this bit is read as logic '1', this FPGA is controlling (Master) the selected VIM transfers. When read as logic '0', this FPGA is the Slave for the selected VIM transfers.

6.11.2.9 PRESENT STATE Bits D3 – 0

These four bits indicate the state that the FPGA State Machine is currently in for the selected VIM BIFO data transfer. The values/states indicated by these bits are used only internally for factory test and debugging purposes.

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)

6.11.3 FPGA Interrupt Mask Registers

There are two FPGA Interrupt Mask Registers, one for FPGA A/B and one for FPGA C/D. Each bit of this register enables or disables the generation of an interrupt for various FPGA functions.

The following table shows the contents of this register. The paragraphs following the table describe the use of these bits. Table 6–37, on the next page, describes the interrupt condition associated with each bit.

Table 6–36: FPGA Interrupt Mask Register
R/W @ VIMRegs_base + **0x0030**
See Table 6–32 for the register base address, VIMRegs_base

	D31 – D16							
Bit Name	Reserved							
	D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Bit Name	SP B(D) RCV F	SP B(D) RCV HF	SP B(D) RCV NE	SP B(D) XMIT NHF	SP B(D) XMIT E	SP A(C) RCV F	SP A(C) RCV HF	SP A(C) RCV NE
Function	0 = Disable Interrupt 1 = Enable Interrupt							
	D7	**D6**	**D5**	**D4**	**D3 – D0**			
Bit Name	SP A(C) XMIT NHF	SP A(C) XMIT E	Reserved	DCM	Reserved			
Function	0 = Disable Interrupt 1 = Enable Interrupt							
All bits default to the logic '0' state at power on and reset.								

Set a mask bit to logic '1' to enable an interrupt for the specified condition. When a mask bit is set to logic '1' and the corresponding status bit changes to active in the FPGA Interrupt Status Register, Section 6.11.5, an interrupt is asserted to the MPC745X or PLX 9656, depending on the type of interrupt.

- The Serial Port interrupts (D15 – D6) are routed to the MPC745X using the Board VIM Serial Port Interrupt Status Register, Section 3.8.14.
- The DCM interrupt (D4) is routed to the PLX 9656 controlling this FPGA, Section 7.9.

Clear the mask bit to logic '0' to disable the interrupt (its default state).

The following two sections, FPGA Interrupt Flag Register, Section 6.11.4, and FPGA Interrupt Status Register, Section 6.11.5, contain related functions for these interrupts.

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)

6.11.3 FPGA Interrupt Mask Registers (continued)

Table 6–37: FPGA Interrupt Registers Bits, VIM to VIM

Bit Name	Bit Position	Interrupt Function
SP B(D) RCV F	D15	This bit is associated with the VIM B (VIM D for FPGA C/D) Serial Port Receive FIFO Full flag, Section 6.7, Table 6–17.
SP B(D) RCV HF	D14	This bit is associated with the VIM B (VIM D for FPGA C/D) Serial Port Receive FIFO Half Full flag, Section 6.7, Table 6–17.
SP B(D) RCV NE	D13	This bit is associated with the VIM B (VIM D for FPGA C/D) Serial Port Receive FIFO Not Empty flag, Section 6.7, Table 6–17.
SP B(D) XMIT NHF	D12	This bit is associated with the VIM B (VIM D for FPGA C/D) Serial Port Transmit FIFO Not Half Full flag, Section 6.7, Table 6–17.
SP B(D) XMIT E	D11	This bit is associated with the VIM B (VIM D for FPGA C/D) Serial Port Transmit FIFO Empty flag, Section 6.7, Table 6–17.
SP A(C) RCV F	D10	This bit is associated with the VIM A (VIM C for FPGA C/D) Serial Port Receive FIFO Full flag, Section 6.7, Table 6–17.
SP A(C) RCV HF	D9	This bit is associated with the VIM A (VIM C for FPGA C/D) Serial Port Receive FIFO Half Full flag, Section 6.7, Table 6–17.
SP A(C) RCV NE	D8	This bit is associated with the VIM A (VIM C for FPGA C/D) Serial Port Receive FIFO Not Empty flag, Section 6.7, Table 6–17.
SP A(C) XMIT NHF	D7	This bit is associated with the VIM A (VIM C for FPGA C/D) Serial Port Transmit FIFO Not Half Full flag, Section 6.7, Table 6–17.
SP A(C) XMIT E	D6	This bit is associated with the VIM A (VIM C for FPGA C/D) Serial Port Transmit FIFO Empty flag, Section 6.7, Table 6–17.
DCM	D4	This bit is associated with the VIM A/B (or C/D) FPGA clock (DCM) not locked condition.

Note: These bit assignments and definitions apply to FPGA Interrupt Mask Register, Section 6.11.3, FPGA Interrupt Flag Register, Section 6.11.4, and FPGA Interrupt Status Register, Section 6.11.5

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)

6.11.4 FPGA Interrupt Flag Registers

There are two FPGA Interrupt Flag Registers, one for FPGA A/B and one for FPGA C/D. Each bit in this register is a read/clear bit that is latched in response to an interrupt occurrence. The following table shows the contents of this register. The paragraphs following this table describe the use of these bits. Table 6–37 describes the interrupt condition associated with each bit.

Table 6–38: FPGA Interrupt Flag Register

R/Clr @ VIMRegs_base + **0x0034**

See Table 6–32 for the register base address, VIMRegs_base

	D31 – D16							
Bit Name	Reserved							
	D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Bit Name	SP B(D) RCV F	SP B(D) RCV HF	SP B(D) RCV NE	SP B(D) XMIT NHF	SP B(D) XMIT E	SP A(C) RCV F	SP A(C) RCV HF	SP A(C) RCV NE
Function	**Read:** 0 = No interrupt 1 = Interrupt latched **Clear:** 0 = Normal operation 1 = Clear latch							
	D7	**D6**	**D5**	**D4**	**D3 – D0**			
Bit Name	SP A(C) XMIT NHF	SP A(C) XMIT E	Reserved	DCM	Reserved			
Function	**Read:** 0 = No interrupt 1 = Interrupt latched **Clear:** 0 = Normal operation 1 = Clear latch							

IMPORTANT! When reset, including power–up, the state of this register is unknown. You should clear this register before using it, by writing '1's into all bits.

When any bit in the FPGA Interrupt Status Register (Section 6.11.5) changes to logic '1', the corresponding bit in this register also changes to logic '1'. However, when a bit in the FPGA Interrupt Status Register clears (changes from logic '1' to logic '0'), the corresponding latched bit in this register does not clear, but remains at the logic '1' state.

Read – When read as logic '1', an interrupt has occurred from the associated FPGA interrupt source. When read as logic '0', the associated FPGA interrupt source is not active. A flag bit remains latched until you clear it (see **Clear**).

Clear – Since these bits latch in response to an interrupt, to detect subsequent interrupts, you must clear the bits in this register. To clear any bit in this register that is set to logic '1', you must write a '1' to that bit to clear the bit.

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)

6.11.5 FPGA Interrupt Status Registers

There are two FPGA Interrupt Status Registers, one for FPGA A/B and one for FPGA C/D. Each bit in this read–only register changes whenever the associated FPGA interrupt source indication changes.

The following table shows the contents of this register. The paragraphs following the table describe the use of these bits. Table 6–37 describes the interrupt condition associated with each bit.

Table 6–39: FPGA Interrupt Status Register
R.O. @ VIMRegs_base + **0x0038**
See Table 6–32 for the register base address, VIMRegs_base

	D31 – D16							
Bit Name	Reserved							
	D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Bit Name	SP B(D) RCV F	SP B(D) RCV HF	SP B(D) RCV NE	SP B(D) XMIT NHF	SP B(D) XMIT E	SP A(C) RCV F	SP A(C) RCV HF	SP A(C) RCV NE
Function	0 = No Interrupt 1 = Interrupt							
	D7	**D6**	**D5**	**D4**	**D3 – D0**			
Bit Name	SP A(C) XMIT NHF	SP A(C) XMIT E	Reserved	DCM	Reserved			
Function	0 = No Interrupt 1 = Interrupt							
All bits default to the logic '0' state at power on and reset.								

A status bit changes to logic '1' whenever the interrupt indication occurs. When any bit in this register changes to logic '1', the corresponding bit in the FPGA Interrupt Flag Register, Section 6.11.4, is set to logic '1'.

If the corresponding bit in the FPGA Interrupt Mask Register, Section 6.11.3 is set to logic '1', then a status bit transition from logic '0' to logic '1' asserts an interrupt. This interrupt is routed to the MPC745X processor using the VIM Serial Port Interrupt Status Register, Section 3.8.14, for bits D15 to D6, or to this FPGA's PLX 9656, Section 7.9, for bit D4. See Figure 7–3 in Section 7.9 for illustration of the PLX 9656 interrupt routing.

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)

6.11.6 FPGA Serial Port Control Registers

There are two FPGA Serial Port Control Registers, one for FPGA A/B and one for FPGA C/D. Each register contains controls bits for the associated VIM module serial ports.

The following table shows the contents of the registers. The subsections following the table provide descriptions of the register bits.

Table 6–40: FPGA Serial Port Control Register

R/W @ VIMRegs_base + **0x003C**

See Table 6–32 for the register base address, VIMRegs_base

	D31 – D15	D14	D13	D12 – D11	D10	D9	D8
Bit Name	Reserved	RCV ENABLE B(D)	XMIT ENABLE B(D)	WORD SIZE B(D)	XMIT CLK SRC B(D)	RCV RESET B(D)	XMIT RESET B(D)
Function	Write zeros, Mask read	0 = Disabled 1 = Enabled		00 = 8-bit word 01 = 16-bit word 10 or 11 = 32-bit word	0 = 33 MHz 1 = VIM SCLK	0 = Normal operation 1 = Reset	
	D7	**D6**	**D5**	**D4 – D3**	**D2**	**D1**	**D0**
Bit Name	Reserved	RCV ENABLE A(C)	XMIT ENABLE A(C)	WORD SIZE A(C)	XMIT CLK SRC A(C)	RCV RESET A(C)	XMIT RESET A(C)
Function	Write zeros, Mask read	0 = Disabled 1 = Enabled		00 = 8-bit word 01 = 16-bit word 10 or 11 = 32-bit word	0 = 33 MHz 1 = VIM SCLK	0 = Normal operation 1 = Reset	

All bits default to the logic '0' state at power on and reset.

6.11.6.1 RCV ENABLE B(D) Bit D14

This bit enables receive operation for Serial Port B (Serial Port D for FPGA C/D). Set this bit to logic '1' to enable receives. Clear this bit to logic '0' (its default state) to disable receives.

6.11.6.2 XMIT ENABLE B(D) Bit D13

This bit enables transmit operation for Serial Port B (Serial Port D for FPGA C/D). Set this bit to logic '1' to enable transmits. Clear this bit to logic '0' (its default state) to disable transmits.

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)

6.11.6 FPGA Serial Port Control Registers (continued)

6.11.6.3 WORD SIZE B(D) Bits D12–D11

These two bits set the word size for Serial Port B (Serial Port D for FPGA C/D). The settings for these two bits are:

00 – 8–bit word
01 – 16–bit word
10 or **11** – 32–bit word

6.11.6.4 XMIT CLK SRC B(D) Bit D10

This bit selects the clock for Serial Port B (Serial Port D for FPGA C/D). Set this bit to logic '1' to use the associated VIM interface SCLK. Clear this bit to logic '0' (its default state) to set the clock to 33 MHz.

6.11.6.5 RCV RESET B(D) Bit D9

This bit resets Serial Port B (Serial Port D for FPGA C/D) receive operation. Set this bit to logic '1' to hold the receive port in reset. Clear this bit to logic '0' (its default state) to resume normal operation of the receive port.

6.11.6.6 XMIT RESET B(D) Bit D8

This bit resets Serial Port B (Serial Port D for FPGA C/D) transmit operation. Set this bit to logic '1' to hold the transmit port in reset. Clear this bit to logic '0' (its default state) to resume normal operation of the transmit port.

6.11.6.7 RCV ENABLE A(C) Bit D6

This bit enables receive operation for Serial Port A (Serial Port C for FPGA C/D). Set this bit to logic '1' to enable receives. Clear this bit to logic '0' (its default state) to disable receives.

6.11.6.8 XMIT ENABLE A(C) Bit D5

This bit enables transmit operation for Serial Port A (Serial Port C for FPGA C/D). Set this bit to logic '1' to enable transmits. Clear this bit to logic '0' (its default state) to disable transmits.

6.11 VIM FPGA Control/Status Registers (VIM to VIM) (continued)

6.11.6 FPGA Serial Port Control Registers (continued)

6.11.6.9 WORD SIZE A(C) Bits D4–D3

These two bits set the word size for Serial Port A (Serial Port C for FPGA C/D). The settings for these two bits are:

00 – 8–bit word
01 – 16–bit word
10 or **11** – 32–bit word

6.11.6.10 XMIT CLK SRC A(C) Bit D2

This bit selects the clock for Serial Port A (Serial Port C for FPGA C/D). Set this bit to logic '1' to use the associated VIM interface SCLK. Clear this bit to logic '0' (its default state) to set the clock to 33 MHz.

6.11.6.11 RCV RESET A(C) Bit D1

This bit resets Serial Port A (Serial Port C for FPGA C/D) receive operation. Set this bit to logic '1' to hold the receive port in reset. Clear this bit to logic '0' (its default state) to resume normal operation of the receive port.

6.11.6.12 XMIT RESET A(C) Bit D0

This bit resets Serial Port A (Serial Port C for FPGA C/D) transmit operation. Set this bit to logic '1' to hold the transmit port in reset. Clear this bit to logic '0' (its default state) to resume normal operation of the transmit port.

6.12 VIM to VIM Data Transfer Operation

VIM to VIM data transfers are configured as fixed pairs; that is, VIM A BIFO is paired with VIM C BIFO, and VIM B BIFO is paired with VIM D BIFO. Use the Data Transfer Control Registers, Section 6.11.1, to set the direction and mode of operation of the data transfer. FPGA A/B is the data transfer Master—only the Master can set the data transfer direction (XFER DIR). Each VIM to VIM pair can be set independently, however, both FPGA ends of each pair must be set for complementary functions.

There are two modes of operation for each transfer between a pair of VIM BIFOs, Mode 0 or Mode 1. Mode 0 starts and stops the transfer using the same boundary flag signal, and thus has lower latency. Mode 1 starts the transfer at one boundary flag level and stops the transfer at the other boundary flag level, thus providing higher latency. Each FPGA end of a VIM to VIM pair can operate in different modes.

Each mode operates differently depending on the setting of XFER DIR in the Data Transfer Control Register. Figure 6–6 on the following page illustrates the VIM to VIM data transfer timing for two of these modes (Transmit in Mode 0, and Receive in Mode 1).

6.12.1 Transmit

Mode 0 – The Transmit FPGA starts reading data from the VIM Incoming FIFO when the FIFO is Not Almost Empty, and stops reading data when the FIFO is Almost Empty. This FPGA transmits the FIFO data to the other (Receive) FPGA as long as there is no Suspend signal from that FPGA. If a Suspend occurs and SUSP EN is enabled, the Transmit FPGA stops the transfer until the Suspend clears, and resumes when Suspend clears.

Mode 1 – The Transmit FPGA starts reading data from the VIM Incoming FIFO when the FIFO is Almost Full, and stops reading data when the FIFO is Almost Empty. This FPGA transmits the FIFO data to the other (Receive) FPGA as long as there is no Suspend signal from that FPGA. If a Suspend occurs and SUSP EN is enabled, the Transmit FPGA stops the transfer until the Suspend clears, and resumes when Suspend clears.

6.12.2 Receive

Mode 0 – The Receive FPGA receives data from the other (Transmit) FPGA. The FPGA starts writing the data received to the VIM Outgoing FIFO when the FIFO is Not Almost Full. When the Outgoing FIFO is Almost Full, the Receive FPGA issues a Suspend and stops writing data. After a Suspend, the FPGA resumes writing and clears Suspend when the FIFO is Not Almost Full.

Mode 1 – The Receive FPGA receives data from the other (Transmit) FPGA. The FPGA starts writing the data received to the VIM Outgoing FIFO when the FIFO is Almost Empty. When the Outgoing FIFO is Almost Full, the Receive FPGA issues a Suspend and stops writing data. After a Suspend, the FPGA resumes writing and clears Suspend when the FIFO is Almost Empty.

6.12 VIM to VIM Data Transfer Operation (continued)

As indicated in the mode descriptions on the preceding page, all VIM to VIM data transfers use the VIM BIFO Almost Empty and Almost Full boundary flags. Thus, before starting any transfer, you must initialize the Incoming FIFO and Outgoing FIFO boundary flags, as described in Section 6.6.1. After initializing these flags, set the VIM BIFO path to 'FPGA' in the VIM Interface BIFO Control Register, Section 6.4.3, before starting a VIM to VIM data transfer.

The following figure illustrates the signal timing for a sample VIM A to VIM C data transfer, using different modes at either end, and with Suspend enabled.

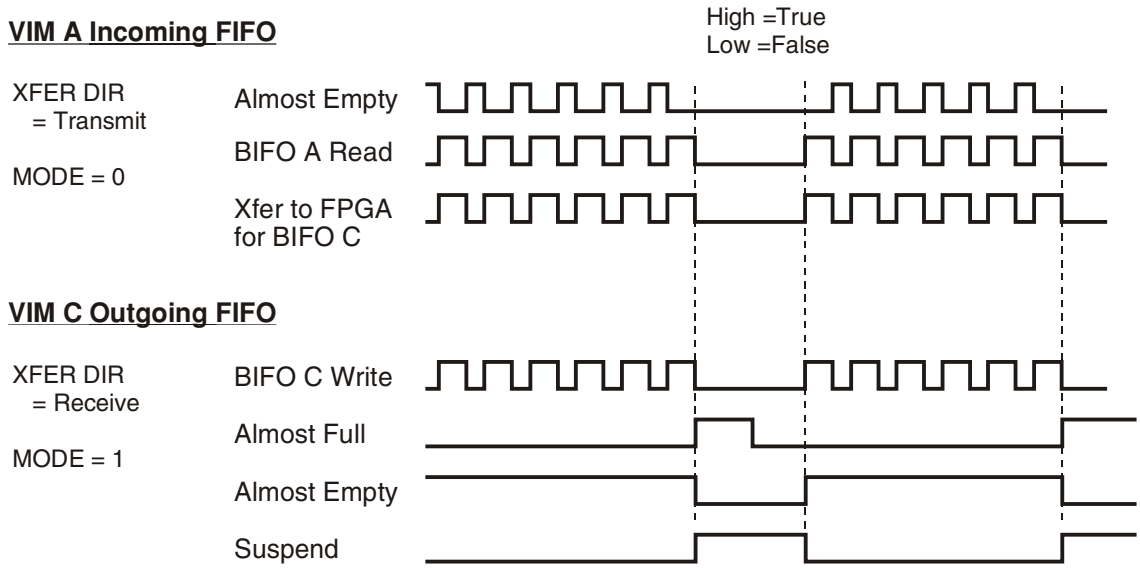


Figure 6–6: VIM to VIM Data Transfer Timing

Note that, as illustrated in this example, if VIM C cannot empty the Outgoing FIFO at the same or higher rate as VIM A is filling the Incoming FIFO, there may be a loss of real–time data from VIM A during the Suspend period. If VIM C can empty the Outgoing FIFO equal to or faster than the rate that VIM A is filling the Incoming FIFO, then the above example would show the VIM C FIFO Almost Full signal staying low (False), and the Suspend would not be issued.

Alternately, if you disable Suspend, during the period that the VIM C Outgoing FIFO is Almost Full, the VIM A FPGA would still transmit data to the VIM C FPGA, but that data may not be able to be written to the VIM C Outgoing FIFO and would be lost.

The Suspend capability is of use when a VIM module can stop filling its Incoming FIFO without loss of data.

Chapter 7: Interrupt Operation

7.1 Overview

This chapter describes the operation of interrupts on the Model 4205. Interrupts can be asserted from the MPC745X processor, each VIM site, the VMEbus interface, the PMC site, the RS−232/Ethernet interface, the ADM1024 Voltage/Temperature monitors, and the optional RACEway interfaces. In addition, the MPC745X processor, and the VME−bus interface can respond to and clear interrupts. Pentek board support software rou−tines (see Section 1.12) are available to control the operation of these interrupt functions.

Section 7.3 provides illustration and identification of all interrupt signal routing and interconnections on the Model 4205.

7.2 Interrupt Operation

The MPC745X processor can respond to two interrupt inputs, SMI and INT. The ADM1024 voltage/temperature monitors are connected to the SMI input and the GT−64260 is connected to the INT input. Any Model 4205 resource, other than the ADM1024, that needs to interrupt the MPC745X is connected to the GT−64260, which in turn interrupts and routes the source interrupt signal to the MPC745X. The MPC745X can assert interrupts to certain board resources by writing to the device registers through the PCI bus. Section 7.4 provides a summary of all interrupt sources and desti−nations for the MPC745X processor.

The GT−64260 controller has Messaging Unit Inbound Doorbell Registers that can be used to generate MPC745X interrupts. Section 7.5 describes the use of the GT−64260 Doorbell Registers for interrupts.

Each VIM module provides three types of interrupts to the MPC745X processor. These interrupts are from the VIM BIFO interface, the VIM serial port interface, and the VIM random access interface. Sections 7.6 and 7.7 describe the operation of these VIM inter−face interrupts.

The Model 4205 is capable of acting as a VMEbus interrupt handler. The seven VME−bus Interrupt Request signals (IRQ1 to IRQ7) are routed to the Universe II, which in turn can map an interrupt to the MPC745X processor, through the GT−64260, from these signals. Also, several resources on the Model 4205 can assert a VMEbus interrupt through the Universe II. Section 7.8 describes these VMEbus interrupts.

Each FPGA's PLX PCI 9656, the Voltage/Temperature monitors, the PMC module, the optional RACEway interfaces, and the optional Serial Mezzanine module can assert interrupts to the MPC745X processor and the VMEbus. These interrupts are described in Sections 7.9, 7.10, 7.11, 7.12, and 7.13, respectively.

7.3 Interrupt Routing

The Model 4205 interrupt interconnections are illustrated below. A detailed diagram of the VIM BIFO/FPGA interrupt logic is provided in Figure 7–2 on the following page. See Table 7–1 for identification of each interrupt route.

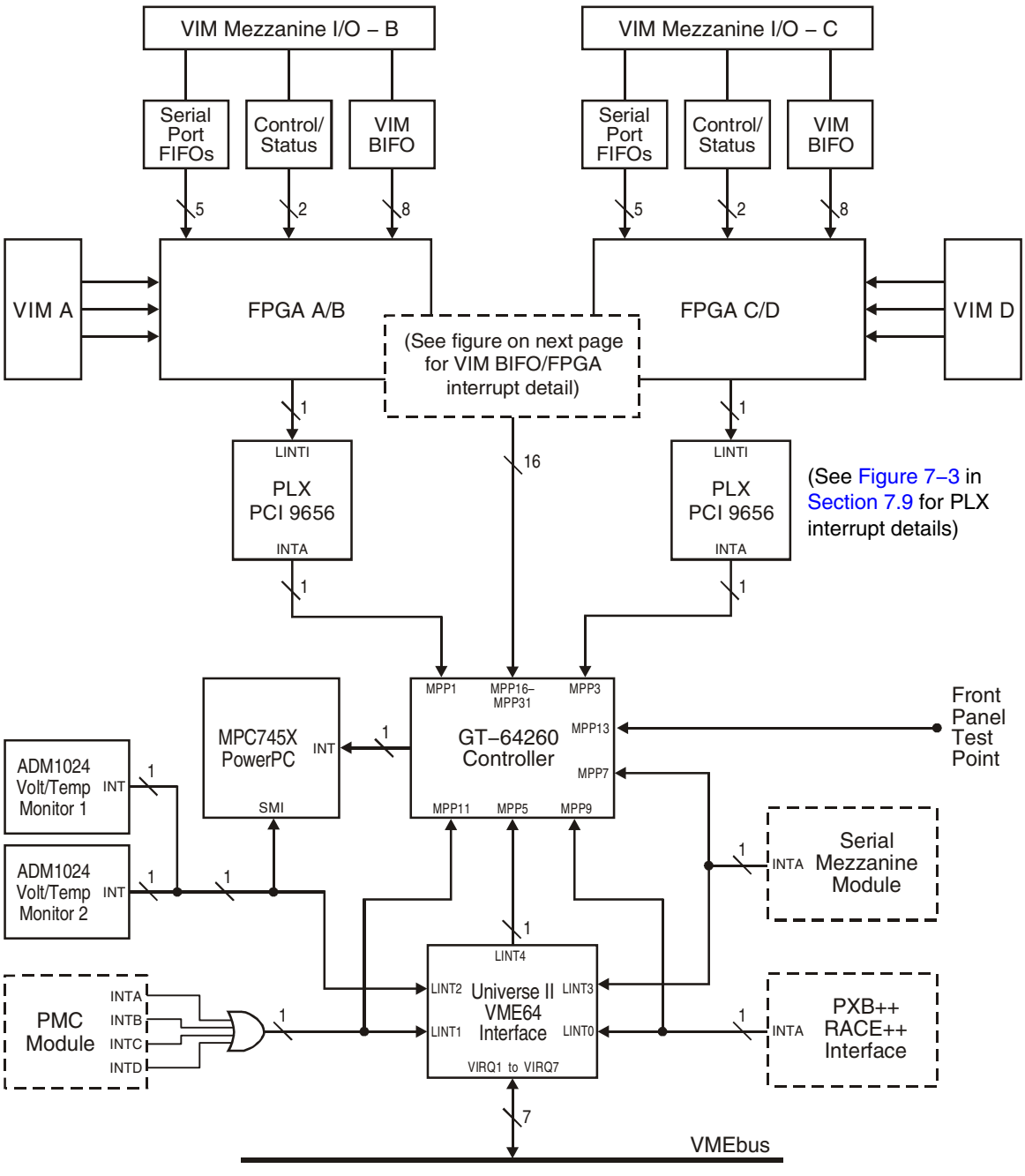


Figure 7–1: Model 4205 Interrupt Connections
(VIMs A and D not detailed)

7.3 Interrupt Routing (continued)

A detailed diagram of the VIM BIFO/FPGA interrupt logic is provided below. The overall interrupt interconnections are illustrated in Figure 7–1 on the preceding page.

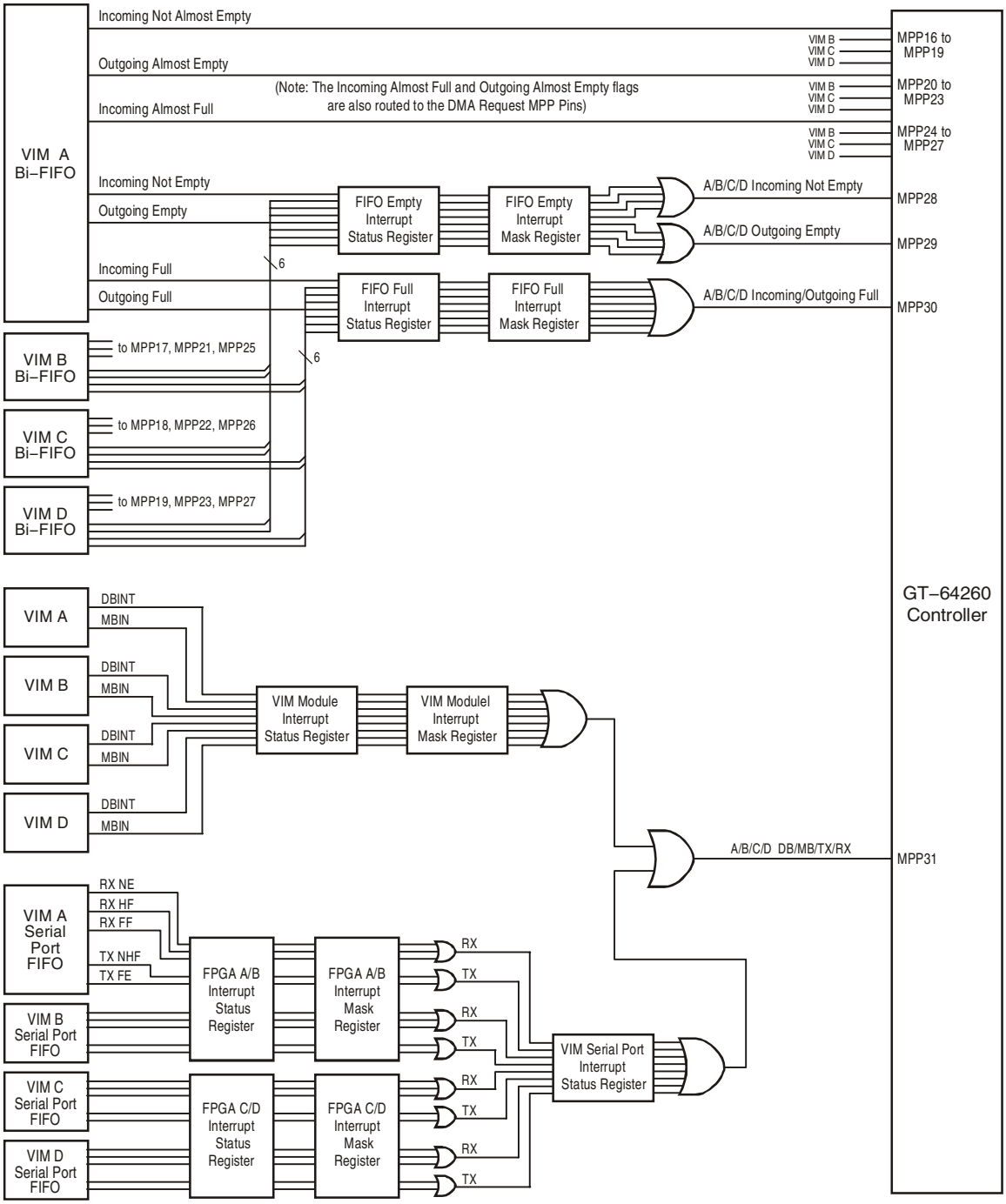


Figure 7–2: VIM BIFO/FPGA Interrupt Routing

7.3 Interrupt Routing (continued)

The following table identifies the source, destination, path, and basic operation of each interrupt. This table includes interrupts that are asserted by register writes over the interconnecting buses as well as the interrupt signals shown in Figure 7–1. The referenced sections provide information on the operation of these interrupts.

Table 7–1: Interrupt Routing

Source	Destination	Path	Operation	Information
Front Panel Test Point	MPC745X	GT–64260 MPP13	External signal applied to Test Point TP3	Section 2.5.9
MPC745X	VMEbus	PCI Bus to Universe II Mailbox Registers	MPC745X write to Universe II Registers	Section 7.4.2
	VIM Module	FPGA to VIM BIFO Mailboxes	MPC745X write to VIM BIFO Mailboxes	
PCI and VMEbus devices	MPC745X	PCI Bus to GT–64260 MU Doorbell Register	PCI write to GT–64260 MU Doorbell Register	Section 7.5
VIM BIFO	MPC745X	VIM FIFO flags to MPC745X registers to MPP16–MPP30	VIM FIFO sets flag conditions	Section 7.6
VIM Module	MPC745X	VIM DBINT/MBIN to MPC745X register to MPP31	VIM asserts interrupt on DBINT or MBIN line	Section 7.7
VIM Serial Ports	MPC745X	VIM serial port FIFO to MPC745X register to MPP31	VIM XMIT/RCV FIFO sets flag conditions	
VMEbus	MPC745X	Universe II LINT4 to GT–64260 MPP5	Universe II asserts interrupt line LINT4	Section 7.8.1
Either FPGA	Associated PLX 9656	FPGA to MPC745X interrupt register to PLX 9656 LINTI	DCM loss or Swinging Buffer conditions	Section 7.9
FPGA A/B PLX	MPC745X	GT–64260 MPP1	PLX 9656 asserts interrupt line LINT0	
FPGA C/D PLX	MPC745X	GT–64260 MPP3		
ADM1024s Temp/Volt Limit	MPC745X	MPC745X SMI (maskable)	Either ADM1024 asserts interrupt line INT	Section 7.10
	VMEbus	Universe II LINT2		
PMC Site	MPC745X	GT–64260 MPP11	PMC module asserts interrupt lines INTA/B/C/D	Section 7.11
	VMEbus	Universe II LINT1		
RACEway PXB++	MPC745X	GT–64260 MPP9	PXB++ asserts interrupt line INTA	Section 7.12
	VMEbus	Universe II LINT0		
Serial Mezzanine Module	MPC745X	GT–64260 MPP7	Serial module asserts interrupt line INTA	Section 7.13
	VMEbus	Universe II LINT3		

7.4 MPC745X Processor Interrupts

The MPC745X processor can respond to specific interrupt signals (Section 7.4.1) and can assert interrupts (Section 7.4.2) to board resources. The GT−64260 also has Doorbell Registers that can be used to interrupt the MPC745X (Section 7.5).

7.4.1 MPC745X Incoming Interrupts

The MPC745X processor can respond to two interrupt signals, SMI and INT. The voltage/temperature monitor interrupt outputs (Section 7.10) are connected to the MPC745X system management interrupt input, SMI. The GT−64260 controller is connected to the MPC745X external interrupt input, INT.

Interrupts to the MPC745X from all resources except the ADM1024 are connected to the GT−64260 using MPP pins. When the GT−64260 receives an interrupt on one of its MPP pins, it in turn interrupts the MPC745X on its INT pin. The MPC745X can then read the MPP pins using its GPP port registers (see Section 3.6.4) to determine the interrupt source.

The following MPP pins are configured as interrupt inputs to the GT−64260. See Section 3.6.8 for description of these MPP pins. When configured as interrupts, MPP pin interrupts are edge triggered.

MPP Pin	From Resource
MPP1	FPGA A/B PLX Interrupt (Section 7.9)
MPP3	FPGA C/D PLX Interrupt (Section 7.9)
MPP5	Universe II VMEbus Interrupt (Section 7.8.1)
MPP7	Serial Mezzanine Module Interrupt (Section 7.13)
MPP9	PXB++ RACEway Interrupt (Section 7.12)
MPP11	PMC Interrupt (Section 7.11)
MPP13	Front Panel Test Point (Section 2.5.9)
MPP16 – MPP30	VIM BIFO Interrupts (Section 7.6)
MPP31	VIM Module DB/MB/XMIT/RCV Interrupt (Section 7.7)

7.4.2 MPC745X Outgoing Interrupts

The MPC745X can assert interrupts to the VMEbus by writing to Universe II Mailbox Registers, MBOX0 to MBOX3 (Section 7.8.2). A reference list of the Universe II registers is provided in Appendix B of this manual. Based on the Universe II Registers memory addresses in the MPC745X Processor Memory Map, Section 3.2.1, MBOX0 to MBOX3 are located at MPC745X memory addresses **0x4060 0348** to **0x4060 0354**.

In addition, the MPC745X can assert a Mailbox Interrupt, MBOUT, to each VIM BIFO mailbox interface by writing to the associated VIM BIFO Mailbox, as described in Section 6.6.5.

7.5 GT−64260 Doorbell Interrupts

The GT−64260 Messaging Unit provides Inbound Doorbell Registers that can be used to generate MPC745X interrupts. There are two 32−bit registers, one on the Primary PCI Bus and one on the Secondary PCI Bus. The Inbound Doorbell Register on the Primary PCI Bus allows the VMEbus or RACEway to assert an interrupt to the MPC745X with a register write. The Inbound Doorbell Register on the Secondary PCI Bus allows any PCI device to assert an interrupt to the MPC745X with a register write.

A device can assert an interrupt to the MPC745X by writing any number other than all zeroes to the associated GT−64260 Inbound Doorbell Register. When the GT−64260 receives one of these doorbell register writes, it interrupts the MPC745X on the INT pin. The interrupt remains asserted until the MPC745X writes logic '1' to the bit, which clears the bit and deasserts the interrupt.

Each Inbound Doorbell Register can generate one of two interrupt causes in the GT−64260 when a bit is set in the register. The upper 16 bits of each register assert a ‘High’ doorbell interrupt cause, and the lower 16 bits assert a ‘Low’ doorbell interrupt cause. All register bits are available for the user.

The GT−64260 Messaging Unit provides Outbound Doorbell Registers that can be used by the MPC745X to generate interrupts. There are two 32−bit registers, one on the Primary PCI Bus and one on the Secondary PCI Bus. The Outbound Doorbell Register on the Primary PCI Bus allows the MPC745X to assert an interrupt to the VMEbus or RACEway with a register write. The Inbound Doorbell Register on the Secondary PCI Bus allows the MPC745X to assert an interrupt to any PCI device with a register write.

See Section 3.6.1 for the address of the GT−64260 Internal Registers from the MPC745X processor, see Section 5.4.3 for the address of the Messaging Unit Registers from an external VMEbus master, and see Section 4.5.4 for the address of these registers from the PCI bus. The Messaging Unit Registers are described in the Marvell GT−64260 Datasheet. Note that the memory address offset of each register is different when accessed from the GT−64260 internal register space or from PCI bus memory−mapped space.

Pentek board support software routines (see Section 1.12) are available to control the operation of these interrupt functions.

7.6 VIM BIFO Interrupts

Each VIM module's Bi–FIFO Streaming Data Interface is connected to a VIM BIFO on the Model 4205. With respect to Model 4205 devices (the MPC745X, a PCI bus device, or an external VMEbus Master), VIM BIFO data reads occur from the Incoming FIFO and writes occur to the Outgoing FIFO. Read and write operations to each VIM BIFO cause the appropriate FIFO status flags to be set and cleared when the data reaches the programmed boundary levels described in Section 6.6.1. See Figure 7–2 for illustration of the VIM BIFO interrupt routing.

Several VIM Incoming and Outgoing FIFO flags are routed directly to the following GT–64260 MPP pins (Section 3.6.8), which in turn route the flags to the MPC745X as an interrupt (Section 7.4.1).

MPP Pin	VIM BIFO Interrupt
MPP16	VIM A Incoming FIFO Not Almost Empty Interrupt
MPP17	VIM B Incoming FIFO Not Almost Empty Interrupt
MPP18	VIM C Incoming FIFO Not Almost Empty Interrupt
MPP19	VIM D Incoming FIFO Not Almost Empty Interrupt
MPP20	VIM A Outgoing FIFO Almost Empty Interrupt
MPP21	VIM B Outgoing FIFO Almost Empty Interrupt
MPP22	VIM C Outgoing FIFO Almost Empty Interrupt
MPP23	VIM D Outgoing FIFO Almost Empty Interrupt
MPP24	VIM A Incoming FIFO Almost Full Interrupt
MPP25	VIM B Incoming FIFO Almost Full Interrupt
MPP26	VIM C Incoming FIFO Almost Full Interrupt
MPP27	VIM D Incoming FIFO Almost Full Interrupt

In addition to the above interrupts, the four Outgoing FIFO Almost Empty and four Incoming FIFO Almost Full flags are also routed to eight DMA Request GT–64260 MPP pins for use in Demand Mode DMA operation, as described in Section 6.6.3.

The VIM BIFO Empty/Not Empty flags are routed to the VIM BIFO Empty Interrupt Status Register (Section 3.8.11) and the VIM BIFO Full flags are routed to the VIM BIFO Full Interrupt Status Register (Section 3.8.10). When a status bit transitions to logic '1' and the corresponding VIM BIFO Empty or Full Interrupt Mask Register bit is enabled (Section 3.8.2 or Section 3.8.1), an interrupt is routed the MPC745X using the following three GT–64260 MPP pins (Section 7.4.1). These MPP pins are OR'ed signals from the respective status bits in the two interrupt status registers (see Figure 7–2).

MPP Pin	VIM BIFO Interrupt
MPP28	VIM Incoming FIFO Not Empty Interrupt
MPP29	VIM Outgoing FIFO Empty Interrupt
MPP30	VIM Incoming/Outgoing FIFO Full Interrupt

7.7 VIM Module Interrupts

Each VIM interface provides several interrupts to the MPC745X processor from the VIM Daughterboard and Mailbox interrupts, and from the VIM serial port interface. Refer to Pentek Manual 800.0002, Pentek Velocity Interface Mezzanine Specification, for details on these interfaces and signals.

7.7.1 VIM Module DB/MB Interrupts

Each VIM site provides a Daughterboard interrupt (DBINT) and a Mailbox interrupt (MBIN) to the baseboard. Each VIM module may assert an interrupt on these lines depending on the capabilities of that module. Refer to the applicable VIM Operating Manual for description of any interrupts that can be generated on these lines by the module.

The DBINT and MBIN interrupts from all four VIM interfaces are routed to the VIM Module Interrupt Status Register (Section 3.8.13). When a VIM module asserts an interrupt on one of these lines, the associated status bit in this register transitions to logic '1', and if the corresponding VIM Module Interrupt Mask Register bit is enabled (Section 3.8.4), an interrupt is routed to the MPC745X using GT−64260 MPP pin MPP31 (Section 7.4).

7.7.2 VIM Serial Port Interrupts

Each VIM site has one serial port connected to the associated FPGA, which are identified as ports A0, B0, C0, and D0. Data from each VIM serial port is routed through a separate FIFO in the FPGA, as described in Section 6.7, and several FIFO boundary flags indicate the data levels in each FIFO.

When you read and write to a serial port FIFO, three RCV (receive) and two XMIT (transmit) FIFO flags are routed to the appropriate bits in the FPGA Interrupt Status Register (Section 6.9.8). For each corresponding bit enabled in the FPGA Interrupt Mask Register (Section 6.9.6), OR'ed results of the three RCV and two XMIT FIFO status bits are routed to the RCV and XMIT status bits of the VIM Serial Port Interrupt Status Register (Section 3.8.14). When any of the enabled RCV or XMIT status bits transitions to logic '1', an interrupt is routed to the MPC745X using GT−64260 MPP pin MPP31 (Section 7.4).

NOTE: Pin MPP31 is an OR'ed signal from the eight status signals of the VIM Module Interrupt Status Register (Section 7.7.1, above), plus the eight status signals of the VIM Serial Port Interrupt Status Register (Section 7.7.2, above).
See Figure 7−2 for illustration of this OR'ed signal routing.

7.8 VMEbus Interrupts

The Model 4205 is capable of acting as a VMEbus interrupt handler. The seven VME–bus Interrupt Request signals (IRQ1 to IRQ7) are routed to the Universe II, which in turn can map an interrupt to the MPC745X processor from these signals. Also, the MPC745X processor can generate a VMEbus interrupt, on any of the seven IRQ levels, either from PCI interrupts or by setting appropriate bits in Universe II registers.

Refer to the Tundra Universe II User Manual for description of Universe II interrupt handling and interrupt generation. A reference list of the Universe II registers is pro–vided in Appendix B of this manual.

7.8.1 VMEbus Interrupt Handling

As a VMEbus interrupt handler, the Universe II can monitor any or all of the VMEbus interrupt levels IRQ1 to IRQ7. Each interrupt is enabled using the Universe II LINT_EN register. Once enabled, assertion of any of the VMEbus interrupt level causes the internal interrupt handler circuitry to request own–ership of the Universe II VMEbus Master Interface. The Universe II latches the first interrupt that appears on the VMEbus and performs an interrupt acknowledge.

Once the acknowledge cycle is complete, the Universe II can assert an inter–rupt to the PCI bus on one of the interrupt lines LINT0 to LINT7, depending on the mapping for that VMEbus level in the LINT_MAPx registers. The interrupt is then cleared and the VMEbus interrupt level is re–armed.

On the Model 4205, the following Universe II PCI bus LINT interrupts are asserted to specific board resources. The VMEbus interrupt levels that gener–ate these interrupts must be mapped by the user (Pentek BSP board support software routines, Section 1.12, are available to map these interrupts).

PCI LINT	To Resource
LINT4	MPC745X, via GT–64260 MPP Pin 5 (Section 7.4.1)

7.8 VMEbus Interrupts (continued)

7.8.2 VMEbus Interrupt Generation

The Universe II can generate interrupts to the VMEbus on interrupt levels IRQ1 to IRQ7 from PCI bus interrupts LINT0 to LINT7. Each of these PCI interrupt sources is individually enabled using the VINT_EN register, and mapped to a particular VMEbus interrupt level using the VINT_MAPx registers. Multiple sources may be mapped to any VMEbus level.

On the Model 4205, the following Universe II PCI bus LINT interrupts are received from specific board resources. The VMEbus interrupt levels that are generated from these interrupts must be mapped by the user (Pentek BSP board support software routines, Section 1.12, are available to map these interrupts).

PCI LINT	From Resource
LINT0	PXB++ RACEWay Interrupt (Section 7.12)
LINT1	PMC Interrupt (Section 7.11)
LINT2	ADM1024 Voltage/Temperature Monitor Interrupt (Section 7.10)
LINT3	Serial Mezzanine Module Interrupt (Section 7.13)
LINT5 – LINT7	Reserved

The Universe II also can generate an interrupt on the VMEbus when one of its Mailbox Registers is accessed, MBOX0 to MBOX3. The mailboxes support read and write access, and may be enabled to generate interrupts on the VMEbus when they are written to. Each Mailbox Register must be enabled by the VINT_EN register and mapped to a particular VMEbus Interrupt level using the VINT_MAPx registers. When enabled, once the mailbox interrupt has been received, the Universe II asserts the appropriate VMEbus interrupt output signal.

Pentek BSP board support software routines, Section 1.12, are available to generate a VMEbus interrupt on any of the seven IRQ levels using the VMEbus Software Interrupts, SW_INT1 to SW_INT7.

7.9 PXL 9656 Interrupts

Each FPGA is controlled by a dedicated PLX PCI 9656 PCI Bus Master, identified as FPGA A/B PLX and FPGA C/D PLX for the respective FPGAs. Each of these two PLX 9656s can respond to and assert interrupts.

For each FPGA, the DCM, two BUFFER OVERFLOW, and two BUFFER FULL interrupt status bits from the FPGA Interrupt Status Register (Section 6.9.8) are OR'ed together and routed to the associated PLX 9656. When any of these status bits transitions to logic '1', and the associated FPGA Interrupt Mask Register bit (Section 6.9.6) or Swinging Buffer Full Mask Register bit (Section 6.9.3) is enabled, an interrupt is asserted to the 9656 local interrupt input, LINTI.

The 9656 responds to this interrupt and, in turn, can assert an interrupt on its PCI interrupt output, INTA, to the following resource interrupt inputs.

PLX 9656	To Resource	Interrupt Input
FPGA A/B PLX	**MPC745X**	GT–64260 MPP pin 1 (Section 7.4)
FPGA C/D PLX	**MPC745X**	GT–64260 MPP pin 3 (Section 7.4)

The DCM, BUFFER OVERFLOW, and BUFFER FULL interrupt interconnections to the PLX 9656 for each FPGA are illustrated below.

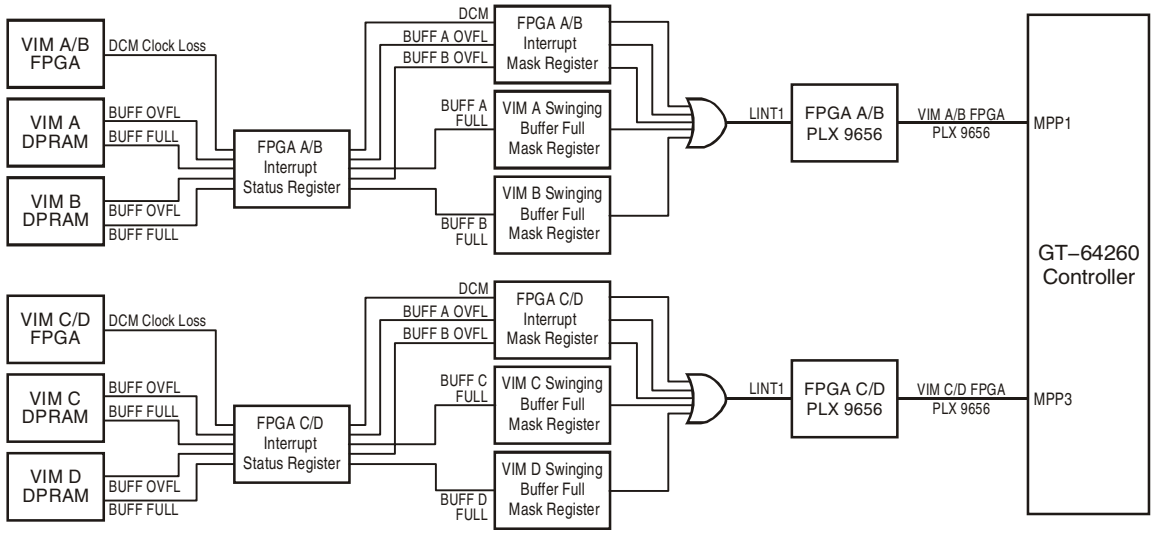


Figure 7–3: PLX PCI 9656 Interrupt Connections

7.10 Voltage/Temperature Monitor Interrupts

The two ADM1024 Voltage/Temperature Monitors provide constant monitoring of critical voltages and temperatures on the Model 4205 PCB, using programmable voltage and temperature limits (see Section 3.5.11). If any measured voltage or temperature fall outside the limits, the ADM1024 asserts an interrupt on its interrupt output line, INT, to the following resource interrupt inputs (the INT outputs of both ADM1024s are tied together).

To Resource	Interrupt Input
MPC745X	MPC745X SMI Interrupt (Section 7.4) – See **Note** below
VMEbus	Universe II LINT2 (Section 7.8.2)



The MPC745X SMI interrupt is enabled/disabled using the Temperature/ Voltage Fault Interrupt Mask Register (Section 3.8.15). The default state of this interrupt is disabled.

When an over–temperature occurs, the ADM1024s also illuminate the red **Temp** LED on the front panel (Section 2.5.2).

7.11 PMC Module Interrupts

A PMC module can generate four PCI interrupt outputs, INTA through INTD. Refer to the Operating Manual supplied with your PMC for description of the respective interrupt sources. The four PMC interrupts are routed to the PMC Interrupt Status Register (Section 3.8.12). If the associated bit in the PMC Interrupt Mask Register (Section 3.8.3) is enabled, an interrupt is asserted, from the OR'ed result of the four status bits, to the following resource interrupt inputs.

To Resource	Interrupt Input
MPC745X	GT–64260 MPP pin 11 (Section 7.4)
VMEbus	Universe II LINT1 (Section 7.8.2)

When Option 010 RACE++ PMC is installed, the module's RACEway controller (PXB++ PCI Bridge) provides an interrupt on its PCI INTA interrupt output.

7.12 RACEway PXB++ Interrupts (Option 005 or 010)

The optional RACE++ interfaces provide data transfers with other RACE++ VME boards. When Option 005 or 010 is installed, the primary RACEway controller (PXB++ PCI Bridge chip) provides an interrupt on its PCI INTA interrupt output. The PXB++ INTA output is routed to the following resource interrupt inputs.

To Resource	Interrupt Input
MPC745X	GT−64260 MPP pin 9 (Section 7.4)
VMEbus	Universe II LINT0 (Section 7.8.2)

7.13 Serial Mezzanine Module Interrupts

The optional serial mezzanine module provides a high−speed serial data interface to the front panel. When external data is received by the interface, or at completion of the cur−rent data transmission, the module controller may generate an interrupt to the follow−ing resource interrupt inputs.

To Resource	Interrupt Input
MPC745X	GT−64260 MPP pin 7 (Section 7.4)
VMEbus	Universe II LINT3 (Section 7.8.2)

When Option 006 Fibre Channel is installed, the module provides an interrupt from the QLogic ISP2310 INTA output.

This page is intentionally blank

Chapter 8: FLASH Memory and Boot Firmware

8.1 Overview

The Model 4205 contains a FLASH memory that acts as the board's nonvolatile storage and a boot resource for the MPC745X processor. When power is applied to the board or when the board is reset, a boot sequence is initiated in the MPC745X for initialization and self−test of the board. With the default board setup, the boot code stored in the FLASH is loaded into the SDRAM and executed to initialize the board.

FLASH memory is shipped with default boot firmware, installed by Pentek. User boot code may be added to the FLASH for individual application requirements.

The following sections of this chapter describe the FLASH memory and Pentek−supplied default boot firmware operation:

- Section 8.2 describes the use of FLASH memory by the MPC745X processor.
- Section 8.3 describes the default boot operation of the MPC745X processor, using the Pentek−supplied boot firmware, when the Model 4205 is reset or power is applied to the board.
- Section 8.4 describes the Pentek−supplied firmware functions in FLASH. This firmware provides a variety of user functions that can be called by any device that has access to SDRAM.
- Section 8.5 identifies the firmware failure codes that may be illuminated on the front panel LEDs or stored in SDRAM if failures occur during boot firmware operation.

See also the reset logic diagram in Section 2.6 for illustration of the reset logic at power−up or reset of the board.

8.2 FLASH Memory

A 32–MByte FLASH memory is provided for nonvolatile storage of initialization, self–test, and boot code. User code may be added to FLASH for individual application requirements. The following table shows the FLASH memory map. FLASH memory is write–protected using PCB jumpers. See Section 2.2.4 for these jumper settings.

Table 8–1: FLASH Memory Map

MPC745X Memory Address	FLASH Memory Area
0xFE00 0000 – 0xFF7F FFFF	User Area
0xFF80 0000 – 0xFFDF FFFF	Reserved for Pentek FPGA Images
0xFFE0 0000 – 0xFFE7 FFFF	VxWorks Boot Parameters
0xFFE8 0000 – 0xFFEF FFFF	VxWorks BootRom Code
0xFFF0 0000 – 0xFFF7 FFFF	RedBoot (see Chapter 9)
0xFFF8 0000 – 0xFFBF FFFF	RedBoot Configuration Parameters
0xFFFC 0000 – 0xFFFF FFFF	RedBoot FIS Directory



FLASH is shipped with the above default boot firmware installed by Pentek.

Users who tamper with the Pentek boot code do so at their own risk.

8.3 Firmware Boot Operation

When the Model 4205 is reset or power is applied, the MPC745X processor copies the RedBoot Monitor boot code from FLASH memory to SDRAM and then executes the boot code from SDRAM. Section 8.3.1 shows the SDRAM memory map for boot operation. Section 8.3.2 describes the Pentek factory–installed MPC745X firmware boot operation.



The first 1 MBytes of SDRAM are reserved for use by the default boot firmware (Section 8.3.1).

Loading user code into this area or accessing data in this area can cause unpredictable behavior.

8.3 Firmware Boot Operation (continued)

8.3.1 Use of SDRAM During Boot

The boot sequence uses fixed areas in the first 1 MByte of SDRAM for controlling and storing results of boot initialization. The boot sequence initializes this SDRAM to the data indicated in Table 8–2, below. The SDRAM memory addresses listed are relative to the MPC745X memory map (Section 3.2.1).

Table 8–2: SDRAM Boot Map

SDRAM Memory Address	SDRAM Boot Use	Contents
0x0000 0000 – 0x0000 3FFF	MPC745X Exception Handlers	–
0x0000 4000 – 0x0000 007F	MPC745X Boot Primary Scan Results	Device & Vendor ID of PCI Bus Devices [2]
0x0000 4080 – 0x0000 40FF	MPC745X Boot Secondary Scan Results	
0x0000 4100 – 0x0000 41FF	Model 4205 Board ID EEPROM Contents [1]	See Table A–1
0x0000 4200	Peak Temperature Voltage Status	See Table 8–4
0x0000 4204 – 0x0000 42FF	Peak Temperature Voltage Values	
0x0000 4300	Current Temperature Voltage Status	See Table 8–3
0x0000 4304 – 0x0000 43FF	Current Temperature Voltage Values	
0x0000 4400 – 0x0000 447F	VIM A ID EEPROM Contents [1]	See Table 8–5
0x0000 4480 – 0x0000 44FF	VIM B ID EEPROM Contents [1]	
0x0000 4500 – 0x0000 457F	VIM C ID EEPROM Contents [1]	
0x0000 4580 – 0x0000 45FF	VIM D ID EEPROM Contents [1]	
0x0000 4800 – 0x0000 481F	Command Table	See Table 8–8
0x0000 4820	Test Patterns	0x1122 3344
0x0000 4824		0xFFFF FFFF
0x0000 4828		0x5555 5555
0x0000 482C		0xAAAA AAAA
0x0000 4830		0x1234 5678
0x0000 4834		0x8765 4321
0x0000 4838	Board ID Revision	0x0000 0000 [3]
0x0000 4848	RedBoot Checksum	0xXXXX XXXX [4]
0x0000 4C40	Diagnostic Status	See Table 8–9
0x0000 4C47 – 0x0000 4FFF	Reserved	–
0x0000 5000 – 0x0000 57FF	Primary PCI Bus Configuration Space Registers	See Appendix C
0x0000 5800 – 0x0000 5FFF	Secondary PCI Bus Configuration Space Registers	
0x0000 6000 – 0x0000 67FF	Tertiary PCI Bus Configuration Space Registers	Option 150/151/152 only, see Appendix C
0x0000 6800 – 0x0000 6FFF	Quaternary PCI Bus Configuration Space Registers	
0x0000 7000 – 0x0000 707F	MPC745x Boot Tertiary Scan Results	Device & Vendor ID of PCI Bus Devices [1]
0x0000 7080 – 0x0000 70FF	MPC745x Boot Quaternary Scan Results	
0x0000 7100 – 0x0000 7FFF	Reserved	–
0x0000 8000 – 0x0009 FFFF	RedBoot Code and Data	Chapter 9
0x0010 0000 – 0x3FFF FFFF	User Area	–

1 – Contents of Board and VIM ID EEPROMs copied into SDRAM during Boot operation, Section 8.3.2
2 – Device ID and Vendor ID are first 32–bit word of PCI Configuration Space Registers, Appendix C
3 – Board revision number, starting from 0
3 – Checksum verified when boot program loads the RedBoot software

8.3 Firmware Boot Operation (continued)

8.3.1 Use of SDRAM During Boot (continued)

During normal operation, while the RedBoot Monitor is at the user prompt, the processor reads the board's voltages and temperature at regular intervals (about 1 minute). It stores the new values in the 'Current Temperature Voltage Values' area of SDRAM at the locations indicated in the following table.

Table 8–3: Current Temperature/Voltage SDRAM Boot Map

SDRAM Memory Address	Current Temperature/Voltage Value	Contents
0x0000 4300	Current Temperature Voltage Status	0x1234 5678 [1]
0x0000 4304	+1.5 Voltage Sensor Reading	Floating Point Numbers [2]
0x0000 4308	+1.6 Voltage Sensor Reading	
0x0000 430C	+1.8 Voltage Sensor Reading	
0x0000 4310	+2.5 Voltage Sensor Reading	
0x0000 4314	+3.3 Voltage Sensor 0 Reading	
0x0000 4318	+3.3 Voltage Sensor 1 Reading	
0x0000 431C	+5 Voltage Sensor 0 Reading	
0x0000 4320	+5 Voltage Sensor 1 Reading	
0x0000 4324	+12 Voltage Sensor 0 Reading	
0x0000 4328	+12 Voltage Sensor 1 Reading	
0x0000 432C	Internal Temperature Sensor 0 Reading	Integer Numbers
0x0000 4330	Internal Temperature Sensor 1 Reading	
0x0000 4334	VIM A/B FPGA Temperature Sensor Reading	
0x0000 4338	VIM C/D FPGA Temperature Sensor Reading	
0x0000 433C	PCB Temperature Sensor Reading	
0x0000 4340 - 0x0000 43FF	Reserved	0x0

1 – If the current sensor readings are not valid, the status word contents = **0x0**.
2 – The Voltage Sensor Readings are stored in IEEE Floating Point format.

The MPC745X then compares the new readings with the 'Peak Temperature Voltage Values' in SDRAM (see Table 8–4, next page). If any new readings are higher than the Max, or lower than the Min, the MPC745X stores the new readings in the appropriate 'Peak Temperature Voltage Values' locations.

The MPC745X also stores these new readings in the corresponding words of the ID EEPROM, then generates a new checksum and stores it in the last word of the EEPROM.

8.3 Firmware Boot Operation (continued)

8.3.1 Use of SDRAM During Boot (continued)

Table 8–4: Peak Voltage/Temperature SDRAM Boot Map

SDRAM Memory Address	Peak Temperature/Voltage Value	Contents
0x0000 4200	Peak Temperature Voltage Status	0x1234 5678 [1]
0x0000 4204	+1.5 Voltage Sensor Reading Min	Floating Point Numbers [2]
0x0000 4208	+1.5 Voltage Sensor Reading Max	
0x0000 420C	+1.6 Voltage Sensor Reading Min	
0x0000 4210	+1.6 Voltage Sensor Reading Max	
0x0000 4214	+1.8 Voltage Sensor Reading Min	
0x0000 4218	+1.8 Voltage Sensor Reading Max	
0x0000 421C	+2.5 Voltage Sensor Reading Min	
0x0000 4220	+2.5 Voltage Sensor Reading Max	
0x0000 4224	+3.3 Voltage Sensor 0 Reading Min	
0x0000 4228	+3.3 Voltage Sensor 0 Reading Max	
0x0000 422C	+3.3 Voltage Sensor 1 Reading Min	
0x0000 4230	+3.3 Voltage Sensor 1 Reading Max	
0x0000 4234	+5 Voltage Sensor 0 Reading Min	
0x0000 4238	+5 Voltage Sensor 0 Reading Max	
0x0000 423C	+5 Voltage Sensor 1 Reading Min	
0x0000 4240	+5 Voltage Sensor 1 Reading Max	
0x0000 4244	+12 Voltage Sensor 0 Reading Min	
0x0000 4248	+12 Voltage Sensor 0 Reading Max	
0x0000 424C	+12 Voltage Sensor 1 Reading Min	
0x0000 4250	+12 Voltage Sensor 1 Reading Max	
0x0000 4254	Internal Temperature Sensor 0 Reading Max	Integer Numbers
0x0000 4258	Internal Temperature Sensor 1 Reading Max	
0x0000 425C	VIM A/B FPGA Temperature Sensor Reading Max	
0x0000 4260	VIM C/D FPGA Temperature Sensor Reading Max	
0x0000 4264	PCB Temperature Sensor Reading Max	
0x0000 4268 - 0x0000 43FF	Reserved	0x0

1 – If the peak sensor readings are not valid, the status word contents = **0x0**.

2 – The Voltage Sensor Readings are stored in IEEE Floating Point format.

8.3 Firmware Boot Operation (continued)

8.3.1 Use of SDRAM During Boot (continued)

Table 8–5: VIM ID EEPROM Example Code (Model 6230 shown)

EEPROM Word Location	SDRAM Location *	Contents	Comments
00/01	+0x00	0x00EE CODE	Valid data flag (VIM_Status word)
02/03	+0x04	0x6230 0000	Model, Model Extension (Extension currently unused)
04/05	+0x08	0x0354 2001	PCB number—this is taken from the VIM board
06/07	+0x0C	0x0001 FFFF	Assembly Rev, Option #1 Rev A = **0001**, Rev B = **0002**, etc.; Option #1 = option number, if no option field = **FFFF**
08/09	+0x10	0xFFFF FFFF	Option #2, Option #3
10/11	+0x14	0xFFFF FFFF	Option #4, Option #5
12/13	+0x18	0xFFFF FFFF	Option #6, Option #7
14/15	+0x1C	0xFFFF FFFF	Option #8, Option #9
16/17	+0x20	0xFFFF FFFF	Module–specific Code goes in locations 16 through 61
18/19	+0x24	0xFFFF FFFF	
20/21	+0x28	0xFFFF FFFF	
22/23	+0x2C	0xFFFF FFFF	
24/25	+0x30	0xFFFF FFFF	
26/27	+0x34	0xFFFF FFFF	
28/29	+0x38	0xFFFF FFFF	
30/31	+0x3C	0xFFFF FFFF	
32/33	+0x40	0xFFFF FFFF	
34/35	+0x44	0xFFFF FFFF	
36/37	+0x48	0xFFFF FFFF	
38/39	+0x4C	0xFFFF FFFF	
40/41	+0x50	0xFFFF FFFF	
42/43	+0x54	0xFFFF FFFF	
44/45	+0x58	0xFFFF FFFF	
46/47	+0x5C	0xFFFF FFFF	
48/49	+0x60	0xFFFF FFFF	
50/51	+0x64	0xFFFF FFFF	
52/53	+0x68	0xFFFF FFFF	
54/55	+0x6C	0xFFFF FFFF	
56/57	+0x70	0xFFFF FFFF	
58/59	+0x74	0xFFFF FFFF	
60/61	+0x78	0xFFFF FFFF	
62/63	+0x7C	0x0000 XXXX	Word 62 should always be cleared (**0000**). Word 63 contains the 16–bit checksum of words 0 – 61.

* Relative to the starting Memory Address for the applicable VIM module, Table 8–2

8.3 Firmware Boot Operation (continued)

8.3.2 MPC745X Boot Sequence

The following steps describe the default boot sequence for the MPC745X processor. This boot sequence is illustrated in Figure 8–1 on page 209. The boot sequence starts with interrupts and cache memory disabled, and initializes SDRAM to the data indicated in Table 8–2, Section 8.3.1.

Each boot sequence step is identified by 'State N', which refers to a code displayed on the processor LEDs (Section 2.5.4)—green LEDs **1**, **2**, and **3** are illuminated with the Hex value of the State number, with LED **1** as the LSB, as shown in the table at the right (x = LED **0**).

Table 8–6: LED Boot Codes

Boot State	LED Code
State 1	001x
State 2	010x
State 3	011x
State 4	100x
State 5	101x
State 6	110x
State 7	111x
State 8	000x

1) State 1 – The MPC745X processor loads and executes the RedBoot Monitor boot sequence. This boot program first initializes the GT–64260 to set up the communications and data paths between the MPC745X and the SDRAM, FLASH Memory, and the Board Control/Status Registers.

   Following a Power On Reset or when continuous BIST is enabled, the first 1 MBytes of SDRAM memory are tested. If a failure occurs, the MPC745X toggles the front panel red **Fault** LED continuously. The error code and the failed address are written to the GT–64260's Outbound Message 0 and 1 Registers. If the SDRAM memory test passes, the boot program copies itself to SDRAM, starting at SDRAM address **0x0000 1000**, and continues executing in SDRAM.

   If an error occurs during the remaining states, the boot program tries to continue. At the completion of BIST, the error log is displayed on both of the serial ports, the front panel **Fault** Led is illuminated, an error code is stored into SDRAM Boot Memory at address **0x0000 4C80**, and the MPC745X blinks its red LED **0** repetitively. See Section 8.5 for description of these test failure codes. If no errors occur during BIST, green LED **1** blinks continuously.

2) State 2 – The boot program completes GT–64260 initialization, and initializes the SDRAM Boot Memory Area.

8.3 Firmware Boot Operation (continued)

8.3.2 MPC745X Boot Sequence (continued)

3) State 3 – The boot program scans the Primary PCI Bus and verifies that all devices are present. The Device and Vendor IDs of each PCI Bus device are stored in SDRAM Boot Memory, starting at address **0x0000 4000**. Each PCI device is configured to reflect the default memory map and enabled. The first 16 PCI Configuration Space registers of each PCI Device are stored in SDRAM Boot Memory, starting at address **0x0000 5000**.

4) State 4 – The boot program scans the Secondary PCI Bus and verifies that all devices are present. The Device and Vendor IDs of each PCI Bus device are stored in SDRAM Boot Memory, starting at address **0x0000 4080**. Each PCI device is configured to reflect the default memory map and enabled. The first 16 PCI Configuration Space registers of each PCI Device are stored in SDRAM Boot Memory, starting at address **0x0000 5800**.

5) State 5 – The boot program reads the Board ID Serial EEPROM and stores it's contents into SDRAM Boot Memory, starting at address **0x0000 4100**.

6) State 6 – The boot program initializes both of the ADM1024 Temperature/ Voltage monitors.

7) State 7 – The boot program scans each VIM site to determine location of each VIM Module. If a VIM module is detected, the contents of the VIM serial ID EEPROM is copied to the SDRAM Boot Memory, starting at address **0x0000 4400** (VIM A), **0x0000 4480** (VIM B), **0x0000 4500** (VIM C), or **0x0000 4580** (VIM D), and the EEPROM checksum is verified.

8) State 8 – The boot sequence is complete. The boot program writes a status word (**0x4205 0000**) to the Universe II Mailbox 0 (Universe II Registers start address, Section 5.4.1, + **0x0348**) at the completion of the boot sequence. An external VMEbus Master should wait until this value is stored before issuing commands to the board.

   If an error occurred during BIST, the error log is displayed on both serial ports, the front panel **Fault** Led is illuminated, an error code is stored into SDRAM Boot Memory at address **0x0000 4C80**, and the MPC745X blinks its red LED **0** repetitively. See Section 8.5 for description of these test failure codes. If no errors occur during BIST, green LED **1** blinks continuously.

RedBoot will execute a user boot script if enabled in FLASH memory or scan the serial ports and Ethernet port waiting for a GDB connection or user input. Refer to Chapter 9 for description of the RedBoot Monitor commands.

8.3 Firmware Boot Operation (continued)

8.3.2 MPC745X Boot Operation (continued)

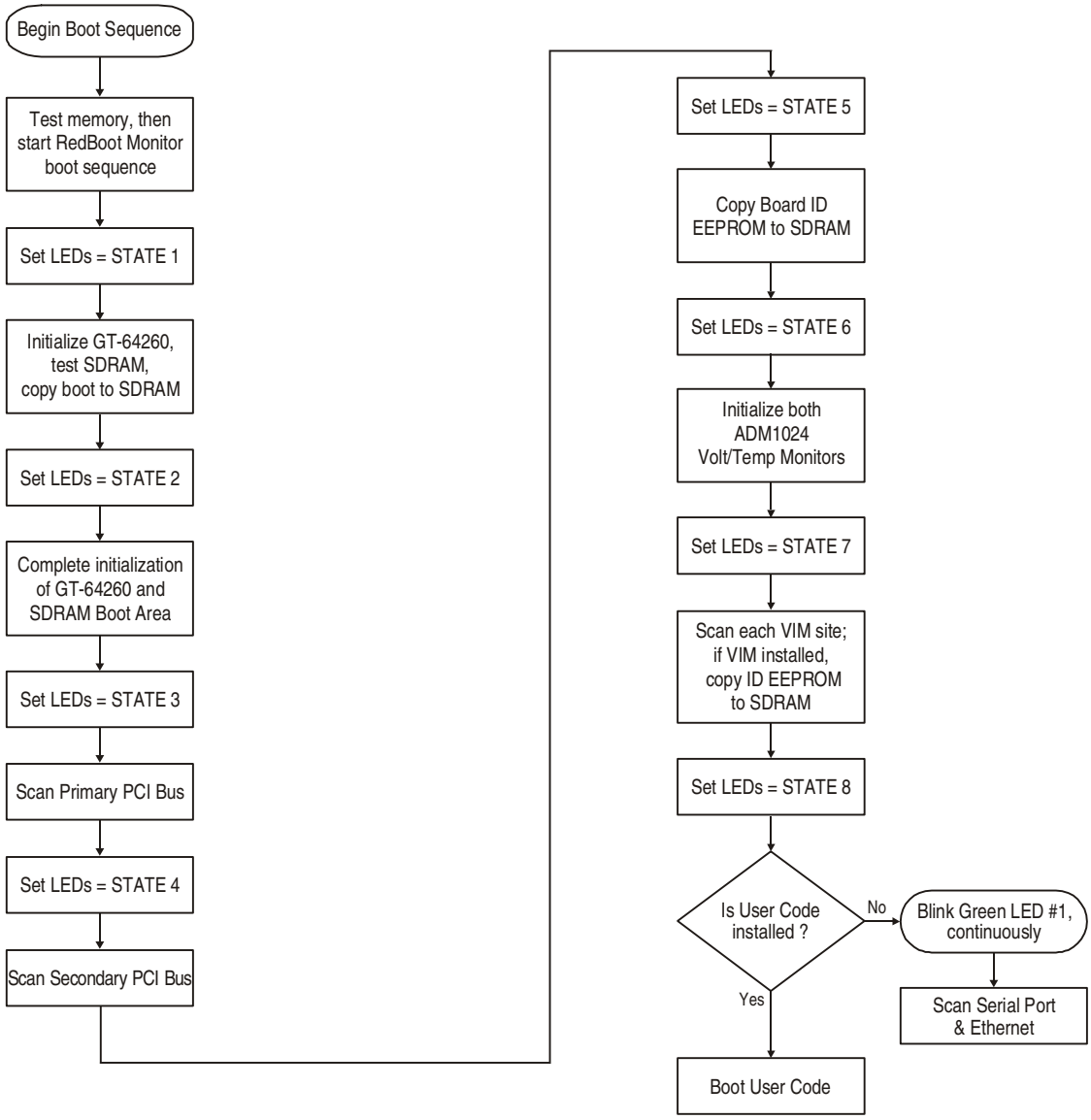


Figure 8–1: MPC745X Firmware Boot Sequence

8.4 MPC745X Firmware Functions

The RedBoot Monitor firmware supplied in FLASH provides the following MPC745X functions that can be called using the Command Table in SDRAM (Table 8–2).

Table 8–7: MPC745X Firmware Function Summary

Function	Code	Argmnt. 1	Argmnt. 2	Argmnt. 3	Description
Read Data (Look)	**0x1**	MPC745X Address	Data Read	N/A	Read data into argument 2 from the address specified by argument 1
Write Data (Set)	**0x2**	MPC745X Address	Data to Write	N/A	Write data from argument 2 into the address specified by argument 1
Move Data (Block Move)	**0x3**	MPC745X Destination Address	MPC745X Source Address	Move Length	Move the number of words specified by argument 3, from the address specified by argument 2, to the address specified by argument 1
Run Code (Go)	**0x4**	MPC745X Address	N/A	N/A	Execute the code located at the address specified by argument 1

After boot, any device that has access to the SDRAM Command Table can call the MPC745X firmware functions listed above. To call MPC745X firmware functions:

1) Write up to three arguments and a function code, as specified in Table 8–7, to the SDRAM Command Table addresses in the table below. Write the Function Code last, since this word is polled by the MPC745X firmware to detect a function call.

Table 8–8: SDRAM Command Table

SDRAM Memory Address	Command Table Parameter
0x0000 4800	Function Code
0x0000 4804	Argument 1
0x0000 4808	Argument 2
0x0000 480C	Argument 3

2) The following occur in the MPC745X in response to a write to the Function Code:

   a) The MPC745X processor reads the parameters from the Command Table, and performs the requested function.

   b) When the function is complete, the MPC745X writes **0x80** (complete) or **0xFF** (error) to the Function Code word in the Command Table.

3) Read the results from the Command Table, if any.

8.5 Firmware Test Failure Codes

A group of four processor LEDs are mounted on the front of the Model 4205 PCB, to the left of the top blank front panel (see figure at the right and Section 2.5.4). The upper two LEDs, labeled **0** and **1** on the panel, indicate the results of the boot initialization self–tests.

- If all boot initialization tests pass, the green LED **1** blinks at a rate of about five times per second.
- If any test fails, the red LED **0** blinks repetitively, remains off for about a second, and then repeats this cycle until you remove power or press the **Reset** button (the Reset cycle does not include the self–test). In addition, a test failure diagnostic code is stored in the Diagnostic Status word of SDRAM at address **0x0000 4C40** (see Table 8–2). The test failure diagnostic codes and LED **0** blink codes are listed in Table 8–9, below.

A red **Fault** LED is located on the Model 4205 front panel center plate (see figure at the right and Section 2.5.5). The MPC745X illuminates this **Fault** LED if any fault is detected during boot initialization (Section 8.3.2).

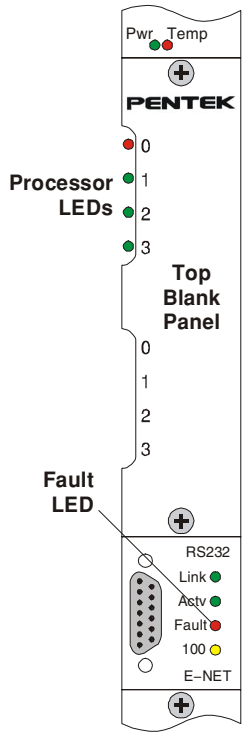


Figure 8–2: Front Panel LEDs

Table 8–9: Boot Firmware Test Failure Codes

BIST Failure	Diag. Code	LED 0 Blinks	BIST Failure	Diag. Code	LED 0 Blinks
SDRAM Increment Error	0x01	1	VIM C EEPROM Checksum Error	0x10	16
SDRAM Inverse Increment Error	0x02	2	VIM D EEPROM Checksum Error	0x11	17
SDRAM Zero Test Error	0x03	3	+1.5 Volt Limit Exceeded	0x12	18
SDRAM Test Pattern Error	0x04	4	+1.6 Volt Limit Exceeded	0x13	19
GT–64260 PCI_0 Device Error	0x05	5	+1.8 Volt Limit Exceeded	0x14	20
GT–64260 PCI_1 Device Error	0x06	6	+2.5 Volt Limit Exceeded	0x15	21
GT–64260 PCI_0 64 Error	0x07	7	+3.3 Volt Limit Exceeded	0x16	22
GT–64260 PCI_1 64 Error	0x08	8	+5 Volt Limit Exceeded	0x17	23
Universe II Device ID Error	0x09	9	+12 Volt Limit Exceeded	0x18	24
PLX 9656, VIM Control/Status Error	0x0A	10	ADM1024 1 Internal Temp Error	0x19	25
PLX 9656, VIM A/B Error	0x0B	11	ADM1024 2 Internal Temp Error	0x1A	26
PLX 9656, VIM C/D Error	0x0C	12	VIM A/B FPGA Temp Error	0x1B	27
Board ID EEPROM Checksum Error	0x0D	13	VIM A/B FPGA Temp Error	0x1C	28
VIM A EEPROM Checksum Error	0x0E	14	PCB Temperature Error	0x1D	29
VIM B EEPROM Checksum Error	0x0F	15	Option 150 Bridge 1 Device Error	0x1E	30

This page is intentionally blank

Chapter 9: The Monitor Program

9.1 Overview

The easiest and most direct path for tailoring the Model 4205's features to the requirements of your application is using the Red Hat™ Embedded Debug and Bootstrap program (RedBoot™), which is accessible from the board's RS–232 serial port. Pentek has modified the RedBoot program to run on the MPC745X processor, providing the following capabilities for the Model 4205:

- Configuring Ethernet network interface(s)
- Managing VIM FPGA configurations
- Viewing and editing PCI Configuration Space of PCI devices
- Setting the system Date and Time
- Configuring VME Slave access
- Configuring VME Master access
- Configuring VME interface parameters such as Bus Request Level and Arbitration Modes
- Viewing SDRAM and FLASH memory locations
- Loading and managing FLASH images
- Loading and executing programs over RS–232 or Ethernet interfaces

9.2 Connecting to the Serial Port

Connect the Model 4205 RS–232 serial port (see Section 2.5.6 for the serial port connector) to an RS–232 terminal or to an RS–232 serial port on any PC or workstation capable of running terminal emulation software. The RedBoot Monitor factory default configuration is for RS–232 operation. The default RS–232 parameters are:

```
Baud Rate: 9600
Parity: None
Data Bits: 8
Stop Bits: 1
```

Once connected, you can modify the default baud rate or enable Ethernet operation using the RedBoot **`fconfig`** command (see Appendix E).

9.3 Connecting to Ethernet Interface

The RedBoot Monitor supports loading and debugging programs over a single Ethernet interface. Ethernet parameters are configured using the RedBoot **`fconfig`** command (see Appendix E). By default, RedBoot uses the GT–64260 Ethernet interface. When the Model 4205 contains a second Ethernet interface (Option 007), the default network device can be modified using the RedBoot **`fconfig`** command. The default network device can be set to either "**`gt64260_eth0`**", for the GT–64260 Ethernet interface, or "**`i82545_eth1`**", for the Option 007 Ethernet interface.

9.4 RedBoot Monitor Commands

The RedBoot Monitor provides a simple command line user interface for program configuration and management, accessible from a serial (terminal) or Ethernet (telnet) interface. Refer to Appendix E for description of these commands. Appendix E is a reprint of the RedBoot™ User's Guide, Chapter 2, RedBoot Commands and Examples.

In addition to the RedBoot commands shown in Appendix E, Pentek has augmented RedBoot to provide the following user command set and their arguments. All references to SDRAM or FLASH memories in these commands are based on the MPC745X Processor Memory Map in Section 3.2.1.

NOTE: Any Model 4205 configuration parameters set by the RedBoot commands (**`appnetconfig`**, **`cconfig`**, **`fconfig`**, **`fpgaconfig`**, **`vmemasterconfig`**, and **`vmeslaveconfig`**) are saved in FLASH and restored after a board reset.

In the following command syntax descriptions, arguments enclosed within brackets **[]** are optional. The commands are listed alphabetically.

9.4.1 appnetconfig

Syntax: **`appnetconfig [-l]`**

Description: Manage application's network parameters kept in FLASH memory. These parameters configure the Ethernet interface(s) when they are used by the User application.

The standard Model 4205 baseboard contains a 10/100 Mbps Ethernet controller built into the GT–64260 Controller. Model 4205 baseboards with Option 007 contain an additional 10/100/1000 Mbps Gigabit Ethernet Controller, Intel 82545GM. Parameters for this additional interface have been added to RedBoot Version 1.4.

9.4 RedBoot Monitor Commands (continued)

9.4.1 appnetconfig (continued)

Description (continued)**:**

The configurable parameters are:

`Enable Network (std)` – This parameter enables the standard Ethernet network interface. Set to **`true`** (**`t`**) to enable or **`false`** (**`f`**) to disable. The default setting is **`false`**, network disabled.

`Use BOOTP for network configuration (std)` – When the network is enabled, the Model 4205 can obtain its Ethernet IP Address from a BOOTP server, or the IP address can be entered manually. Set to **`true`** (**`t`**) to use BOOTP, or **`false`** (**`f`**) to enter manually with the following parameters. The default is **`false`**.

`Local IP Address (std)` – This parameter specifies the IP Address for this board. The Local IP Address is specified with the following format: **`XXX.XXX.XXX.XXX`**.

`Local IP Address Mask (std)` – This parameter specifies the network mask. The Local IP Address Mask is specified with the following format: **`XXX.XXX.XXX.XXX`**. The default mask is **`255.255.255.0`**.

`Gateway Address (std)` – This parameter specifies the IP address of the machine used as a gateway to another network. The gateway address is specified in the following format: **`XXX.XXX.XXX.XXX`**. To access machines that are only on a local network, specify a gateway address of **`0.0.0.0`**. The default address is **`0.0.0.0`**.

`Server IP Address (std)` – This parameter specifies the IP address of a default server on the network. The Server IP Address is specified with the following format: **`XXX.XXX.XXX.XXX`**. The default address is **`0.0.0.0`**.

`Broadcast Address (std)` – This parameter specifies the IP address for broadcasting packets to all systems on a network. The broadcast address is specified with the following format: **`XXX.XXX.XXX.XXX`**. The default address is **`0.0.0.0`**.

9.4 RedBoot Monitor Commands (continued)

9.4.1 appnetconfig (continued)

Description (continued)**:**

`Enable Network (opt7)` – This parameter enables the Option 007 Ethernet network interface. Set to **`true`** (**`t`**) to enable or **`false`** (**`f`**) to disable. The default setting is **`false`**, network disabled.

`Use BOOTP for network configuration (opt7)` – When the network is enabled, the Model 4205 can obtain its Ethernet IP Address from a BOOTP server, or the IP address can be entered manually. Set to **`true`** (**`t`**) to use BOOTP, or **`false`** (**`f`**) to enter manually with the following parameters. The default is **`false`**.

`Local IP Address (opt7)` – This parameter specifies the IP Address for this board. The Local IP Address is specified with the following format: **`XXX.XXX.XXX.XXX`**.

`Local IP Address Mask (opt7)` – This parameter specifies the network mask. The Local IP Address Mask is specified with the following format: **`XXX.XXX.XXX.XXX`**. The default mask is **`255.255.255.0`**.

`Gateway Address (opt7)` – This parameter specifies the IP address of the machine used as a gateway to another network. The gateway address is specified in the following format: **`XXX.XXX.XXX.XXX`**. To access machines that are only on a local network, specify a gateway address of **`0.0.0.0`**. The default address is **`0.0.0.0`**.

`Server IP Address (opt7)` – This parameter specifies the IP address of a default server on the network. The Server IP Address is specified with the following format: **`XXX.XXX.XXX.XXX`**. The default address is **`0.0.0.0`**.

`Broadcast Address (opt7)` – This parameter specifies the IP address for broadcasting packets to all systems on a network. The broadcast address is specified with the following format: **`XXX.XXX.XXX.XXX`**. The default address is **`0.0.0.0`**.

9.4 RedBoot Monitor Commands (continued)

9.4.1 appnetconfig (continued)

Description (continued)**:**

`DNS domain name` – This parameter specifies the domain name used when querying a DNS Server.

`DNS Server IP Address` – This parameter specifies the IP Address of the DNS Server. The DNS Server IP address is specified with the following format: **`XXX.XXX.XXX.XXX`**. The default address is **`0.0.0.0`**.

Arguments: **`-l`** List current configuration settings. If this argument is omitted, you can modify each parameter as it is listed. To modify a value, first delete the old value and then enter the new value. To keep the displayed value, just use the **Enter** key.

9.4 RedBoot Monitor Commands (continued)

9.4.2 cconfig

Syntax: `cconfig [-l]`

Description: Manages the MPC745X's L2 and L3 Cache Configuration stored in FLASH memory. The MPC745X has three levels of cache that is divided as:

L1 Data Cache – 32 Kbytes
L1 Instruction Cache – 32 Kbytes
L2 Data and Instruction Cache – 512 Kbytes
L3 Data and Instruction Cache – 4 Mbytes

L2 and L3 Caches can be configured to cache instructions and data, data only, or instructions only. This command allows the user to configure these settings.

The configurable parameters are:

`L2 Cache - Data Only` – This parameter configures the L2 Cache to data only. Set to **`true`** (**`t`**) to configure for data only or **`false`** (**`f`**) to configure for instruction and data. The default is **`false`**.

`L2 Cache - Instruction Only` – This parameter configures the L2 Cache to instruction only. Set to **`true`** (**`t`**) to configure for instruction only or **`false`** (**`f`**) to configure for instruction and data. The default setting is **`false`**.

Note: If Data Only and Instruction Only parameters are both set to **`true`**, the L2 cache will not be enabled.

`L3 Cache - Data Only` – This parameter configures the L3 Cache to data only. Set to **`true`** (**`t`**) to configure for data only or **`false`** (**`f`**) to configure for instruction and data. The default is **`false`**.

`L3 Cache - Instruction Only` – This parameter configures the L3 Cache to instruction only. Set to **`true`** (**`t`**) to configure for instruction only or **`false`** (**`f`**) to configure for instruction and data. The default setting is **`false`**.

Note: If Data Only and Instruction Only parameters are both set to **`true`**, the L3 cache will not be enabled.

9.4 RedBoot Monitor Commands (continued)

9.4.2 cconfig (continued)

Arguments: **-l** List current cache configuration settings. If this argument is omitted, you can modify each parameter as it is listed. To modify a value, first delete the old value and then enter the new value. To keep the displayed value, just use the **Enter** key.

Example: Display cache configuration settings.

```
RedBoot> cconfig -l

L2 Cache - Data Only: false
L2 Cache - Instruction Only: false
L3 Cache - Data Only: false
L3 Cache - Instruction Only: false
```

9.4 RedBoot Monitor Commands (continued)

9.4.3 displayTempVoltage

Syntax: `displayTempVoltage [-c] [-p]`

Description: Displays the current temperature and voltage readings from the board sensors.

Arguments:

`-c`	Continuous display of current temperature and voltage readings. The board must be reset to terminate this command.
`-p`	Display the peak temperature and voltage readings.

Example: Display current temperature and voltage settings.

```
RedBoot> displayTempVoltage

Current Temperature and Voltage Settings
+1.5V Value = 1.471
+1.6V Value = 1.576
+1.8V Value = 1.765
+2.5V Value = 2.490
+3.3V Value(Adm1024 #1) = 3.236
+3.3V Value(Adm1024 #2) = 3.253
+5V Value(Adm1024 #1) = 5.006
+5V Value(Adm1024 #2) = 5.032
+12V Value(Adm1024 #1) = 11.953
+12V Value(Adm1024 #2) = 11.829
Adm1024 #1 Internal Temperature Sensor Value =  25 % C
Adm1024 #2 Internal Temperature Sensor Value =  50 % C
VIM A/B FPGA Temperature Sensor Value =  44 % C
VIM C/D FPGA Temperature Sensor Value =  35 % C
PCB Temperature Sensor Value(near heat sink) =  35 % C
```

9.4 RedBoot Monitor Commands (continued)

9.4.4 fpgaconfig

Syntax: `fpgaconfig [-l]`

Description: Manages the FPGA configuration parameters stored in FLASH memory. The Model 4205 contains two FPGAs that can be configured in different ways. This command selects the FPGA image to be loaded into each of the FPGAs following a board reset. All FPGA images must be stored in FLASH Memory.

The Model 4205 provides two default FPGA images in FLASH:

- VIM to Processor (see Section 6.5.1)
- VIM to VIM (see Section 6.5.2)

This command can also select an FPGA image created in FLASH by the user. RedBoot's **`fis create`** command (see Appendix E) can be used to create a new FPGA image in FLASH and define the name of that image.

The FPGA configuration parameters are:

`VIM A/B FPGA Program` – This parameter selects the FPGA image to be loaded into the FPGA connected to VIM A and B. This parameter can be set to **`vim2vim`** or **`vim2proc`**, or a user–defined image name. The default is **`vim2proc`**.

`VIM C/D FPGA Program` – This parameter selects the FPGA image to be loaded into the FPGA connected to VIM C and D. This parameter can be set to **`vim2vim`** or **`vim2proc`**, or a user–defined image name. The default is **`vim2proc`**.

Arguments:

`-l`	List current FPGA configuration settings. If this argument is omitted, you can modify each parameter as it is listed. To modify a value, first delete the old value and then enter the new value. To keep the displayed value, just use the **Enter** key.

Example: Display current FPGA configuration settings.

```
RedBoot> fpgaconfig -l

VIM A/B FPGA Program: vim2proc
VIM C/D FPGA Program: vim2proc
```

9.4 RedBoot Monitor Commands (continued)

9.4.5 fpgaload

Syntax: **`fpgaload -b <location> -f <fpga> -l <length>`**

Description: Loads an FPGA image stored in SDRAM to one of the FPGAs. RedBoot's **`load`** command (see Appendix E) can be used to load the FPGA image into SDRAM.

Note that this FPGA configuration is volatile, and will be replaced with the default (or user−defined) image selected by **`fpgaconfig`** (see Section 9.4.4) following a board power on or reset.

Arguments:

`-b location` Starting address of FPGA image.

`-f fpga` FPGA to be loaded (**`0`** = VIM A/B, **`1`** = VIM C/D).

`-i length` Size of image in bytes.

Example: Load the FPGA image stored in memory at **`0x400000`** to the FPGA connected to VIM A and B.

```
RedBoot> fpgaload -b 0x400000 -f 0 -l 0x7c970
```

9.4 RedBoot Monitor Commands (continued)

9.4.6 help

Syntax: `help`

Description: Displays a list of commands and their arguments (including the standard RedBoot command set listed in Appendix E).

Arguments: None

Example:

```
RedBoot> help

Manage aliases kept in FLASH memory
   alias name [value]
Manage application's network parameters kept in FLASH memory
   appnetconfig [-l]
Set/Query the system console baud rate
   baudrate [-b <rate>]
Manage L2/L3 Cache Configuration parameters kept in FLASH memory
   cconfig [-l]
Manage machine caches
   cache [ON | OFF]
Display/switch console channel
   channel [-1|<channel number>]
Compute a 32bit checksum [POSIX algorithm] for a range of memory
   cksum -b <location> -l <length>
Display Current Voltage and Temperature Values
   displayTempVoltage -c -p
Display (hex dump) a range of memory
   dump -b <location> [-l <length>] [-s] [-1|2|4]
Manage FLASH images
   fis {cmds}
Manage configuration kept in FLASH memory
   fconfig [-i] [-l] [-n] [-f] [-d] | [-d] nickname [value]
Manage FPGA configuration parameters kept in FLASH memory
   fpgaconfig [-l]
Load FPGA Image
   fpgaload -b <location> -f <fpga> -l <length>
Execute code at a location
   go [-w <timeout>] [entry]
Help about help?
   help [<topic>]
Set/change IP addresses
   ip_address [-l <local_ip_address>] [-h <server_address>]
```

9.4 RedBoot Monitor Commands (continued)

9.4.6 help (continued)

Example (continued)**:**

```
Load a file
   load [-r] [-v] [-h <host>] [-m <varies>] [-c <channel_number>]
        [-b <base_address>] <file_name>
Compare two blocks of memory
   mcmp -s <location> -d <location> -l <length> [-1|-2|-4]
Fill a block of memory with a pattern
   mfill -b <location> -l <length> -p <pattern> [-1|-2|-4]
Display (hex dump) PCI Configuration Space
   pdump -d <device> [-f <function>] [-l <length>]
Edit a register in PCI Configuration Space
   pedit -d <device> -f <function> -o <offset> -v <value>
Network connectivity test
   ping [-v] [-n <count>] [-l <length>] [-t <timeout>] [-r <rate>]
        [-i <IP_addr>] -h <IP_addr>
Reset the system
   reset
Set Current Time and Date
   time
Display RedBoot version information
   version
Manage VME Master Address parameters kept in FLASH memory
   vmemasterconfig [-l]
Manage VME Slave Address parameters kept in FLASH memory
   vmeslaveconfig [-l]
Display (hex dump) a range of memory
   x -b <location> [-l <length>] [-s] [-1|2|4]
RedBoot>
```

9.4 RedBoot Monitor Commands (continued)

9.4.7 pdump

Syntax: **pdump -b <*bus*> -d <*device*> [-f <*function*>] [-l <*length*>]**

Description: Displays the contents of a device in PCI Configuration Space, in Hex format.

Arguments:

-b *bus* PCI Bus containing the Device

-d *device* PCI Device to be accessed. The devices available are:

0 – GT–64260
2 – Universe II VME Interface
4 – VIM Control/Status PLX9656
5 – PXB++ (with Option 005 or 010)
7 – Serial Mezzanine
8 – VIM A/B PLX9656
9 – VIM C/D PLX9656
10 – PMC

When this parameter is omitted, the last specified device would be used.

-f *function* Function Number. Used to select which function of a multi–function device is being accessed. Range is **0** to **7**. When this parameter is omitted, the last specified function would be used. The default is **0**.

-l *length* The number of 32–bit words to be displayed. When this parameter is omitted, the last specified length would be used. The default is **16**.

Example: Dump the PCI Configuration Space registers for the PXB++ (with Option 005 or 010 only).

```
RedBoot> pdump -b 0 -d 5

00000000: 00010034 04800007 06040002 00010000
00000010: f0000008 18600000 00000100 04800111
00000020: 20001870 fff0f000 00000000 00000000
00000030: 00000000 00000000 18700000 00000000
```

9.4 RedBoot Monitor Commands (continued)

9.4.8 pedit

Syntax: **`pedit -d <`*`device`*`> [-f <`*`function`*`>] -o <`*`offset`*`> -v <`*`value`*`>`**

Description: Edits the contents of a device register in PCI Configuration Space.

Arguments:

`-d `*`device`* PCI Device to be accessed. The devices available are:

`0` – GT–64260
`2` – Universe II VME Interface
`4` – VIM Control/Status PLX9656
`5` – PXB++ (with Option 005 or 010)
`7` – Serial Mezzanine
`8` – VIM A/B PLX9656
`9` – VIM C/D PLX9656
`10` – PMC

`-f `*`function`* Function Number. Used to select which function of a multi–function device is being accessed. Range is **`0`** to **`7`**. When this parameter is omitted, the default value is **`0`**.

`-o `*`offset`* Byte offset from the start of the device's PCI configuration space.

`-v `*`value`* The value to be stored in the register.

Example: Modify the PMC's command/status register at offset 4.

```
RedBoot> pedit -d 10 -f 0 -o 4 -v 0x7
```

9.4 RedBoot Monitor Commands (continued)

9.4.9 time

Syntax: `time`

Description: Displays or sets the system time and date. The clock runs in 24–hour mode.

Arguments: None

Example: Display the current date and time.

```
RedBoot> time

The current date and time is: Thu Jun 26 14:36:11 2003
Enter the new time (hh:mm:ss):
Enter the new date (mm-dd-yy):
```

9.4 RedBoot Monitor Commands (continued)

9.4.10 vmemasterconfig

Syntax: `vmemasterconfig [-l]`

Description: Manages the VME Master parameters stored in Flash Memory.

The configurable parameters are as follows (the default values provide the Master memory map shown in Section 5.2):

`Enable VME Master A32/D16 Space` – This parameter enables VME Master A32/D16 Address Space. Set to **`true`** (**`t`**) to enable or **`false`** (**`f`**) to disable. The default is **`true`**, enabled.

`VME Master A32/D16 PCI Address` – When A32/D16 Space is enabled, this parameter contains the PCI address that provides access to this region. The default PCI address is **`0x90000000`**.

`VME Master A32/D32 VME Address` – When A32/D16 Space is enabled, this parameter contains the starting VME address that is generated when the PCI address defined above is accessed. The default VME address is **`0x0`**.

`VME Master A32/D16 Space Size` – When A32/D16 Space is enabled, this parameter contains the size, in bytes, of the A32/D16 window. The default size is **`0x10000000`**.

`Enable VME Master A32/D32 Space` – This parameter enables VME Master A32/D32 Address Space. Set to **`true`** (**`t`**) to enable or **`false`** (**`f`**) to disable. The default is **`true`**.

`VME Master A32/D32 PCI Address` – When A32/D32 Space is enabled, this parameter contains the PCI address that provides access to this region. The default PCI address is **`0xA0000000`**.

`VME Master A32/D32 VME Address` – When A32/D32 Space is enabled, this parameter contains the starting VME address that is generated when the PCI address defined above is accessed. The default VME address is **`0x0`**.

`VME Master A32/D32 Space Size` – When A32/D32 Space is enabled, this parameter contains the size, in bytes, of the A32/D32 window. The default size is **`0x40000000`**.

9.4 RedBoot Monitor Commands (continued)

9.4.10 vmemasterconfig (continued)

Description (continued)**:**

`Enable VME Master A32/D64 Space` – This parameter enables VME Master A32/D64 Address Space. Set to **`true`** (**`t`**) to enable or **`false`** (**`f`**) to disable. The default is **`true`**.

`VME Master A32/D64 PCI Address` – When A32/D64 Space is enabled, this parameter contains the PCI address that provides access to this region. The default PCI address is **`0x84000000`**.

`VME Master A32/D64 VME Address` – When A32/D64 Space is enabled, this parameter contains the starting VME address that is generated when the PCI address defined above is accessed. The default VME address is **`0x0`**.

`VME Master A32/D64 Space Size` – When A32/D64 Space is enabled, this parameter contains the size, in bytes, of the A32/D64 window. The default size is **`0x0C000000`**.

`Enable VME Master Image 7 User Configurable` – This parameter enables Universe II's PCI Target Image 7. This master image can be set up as needed depending on the user's application. Set to **`true`** (**`t`**) to enable or **`false`** (**`f`**) to disable this image. The default is **`false`**. Refer to the Tundra Universe II manual.

`VME Master Image 7 PCI Address` – When VME Master Image 7 is enabled, this parameter specifies the PCI address that provides access for this image.

`VME Master Image 7 VME Address` – When VME Master Image 7 is enabled, this parameter contains the starting VME address that is generated when the PCI address defined above is accessed.

`VME Master Image 7 Size` – When VME Master Image 7 is enabled, this parameter contains the size, in bytes, of this window.

`VME Master Image 7 Control Register Setting` – When VME Master Image 7 is enabled, this parameter specifies the PCI Target Image 7 control register setting. Refer to the Tundra Universe II manual.

9.4 RedBoot Monitor Commands (continued)

9.4.10 vmemasterconfig (continued)

Description (continued)**:**

`VME Master Image 7 User AM` – When VME Master Image 7 is enabled, this optional parameter specifies a custom AM code. This parameter is needed if the ones selectable in the PCI Target Image control register are not applicable. A setting of **`0x0`** disables this parameter.

`VMEBus Release Mode` – This parameter selects the VMEbus Release Mode. Set to **`0`** for release when done or **`1`** for release on request. The default is **`0`**.

`VMEBus Request level` – This parameter selects the VMEbus Request Level. Set to **`0`**, **`1`**, **`2`**, or **`3`**. The default is **`3`**.

`VMEBus Request Mode` – This parameter selects the VMEbus Request Mode. Set to **`0`** for request on demand or **`1`** for fair requester. The default is **`0`**.

Arguments:

`-l`	List current VME Master configuration settings. If this argument is omitted, you can modify each parameter as it is listed. To modify a value, first delete the old value and then enter the new value. To keep the displayed value, just use the **Enter** key.

9.4 RedBoot Monitor Commands (continued)

9.4.11 vmeslaveconfig

Syntax: `vmeslaveconfig [-l]`

Description: Manages the VME Slave parameters stored in Flash Memory.

The configurable parameters are:

`Enable VME Slave Extended Memory Map` – This parameter enables the VME Slave Extended Memory Map for access to various resources on the board from an external VMEbus Master. Set to **`true`** (**`t`**) to enable or **`false`** (**`f`**) to disable VME Slave access. The default is **`false`**, disabled.

The Model 4205 as shipped from the factory only has access to the Tundra Universe II's registers in A16 or A24 Address Modes (see Section 5.3.2). When VME Slave Extended Memory Map is enabled (above), the following parameters can be configured. The default values provide the A32 Slave memory map shown in Section 5.3.4.

`Enable A24 VME Slave Address Space` – This parameter selects either the A24 or A32 VME Slave Extended Memory Map. Set to **`true`** (**`t`**) to enable A24 access or **`false`** (**`f`**) to enable A32 access. The default is **`false`**, A32 address space.

`VME Slave Base Address` – This parameter selects the VME Slave Extended Memory Map's Base Address. When A24 access is enabled, this parameter should be set to either **`0x0`** or **`0x800000`**. When A32 access is enabled (default), this parameter must be in the range **`0x0`** to **`0xFC000000`**, in multiples of **`0x04000000`**. The default setting is **`0x0`**.

`SDRAM Boot Memory Offset/Message Unit Offset` – This parameter specifies the offset from the VME Slave Base Address where access to the GT–64260's message unit and SDRAM Boot Memory Area begins. The default setting is **`0x1000`**.

`SDRAM Boot Memory Offset/Message Unit Size` – This parameter specifies the size, in bytes, of the VME access window. To disable access to these resources, specify a size of **`0`**. The default setting is **`0x5000`**.

9.4 RedBoot Monitor Commands (continued)

9.4.11 vmeslaveconfig (continued)

Description (continued)**:**

`GT-64260 Internal Registers Offset` – This parameter specifies the offset from the VME Slave Base Address where access to the GT−64260's Internal Registers begins. The default setting is **`0x10000`**.

`GT-64260 Internal Registers Size` – This parameter specifies the size, in bytes, of the VME access window. To disable access to these resources, specify a size of **`0`**. The default setting is **`0x10000`**.

`PMC Access Offset` – When A32 VME Slave Access is enabled, this parameter specifies the offset from the VME Slave Base Address where access to the PMC Memory begins. The default setting is **`0x100000`**.

`PMC Access Size` – When A32 VME Slave Access is enabled, this parameter specifies the size, in bytes, of the VME access window. To disable access to these resources, specify a size of **`0`**. The default setting is **`0x500000`**.

`Board Registers Offset` – When A32 VME Slave Access is enabled, this parameter specifies the offset from the VME Slave Base Address where access to the Board Control/Status and Interrupt Registers begins. The default setting is **`0x600000`**.

`Board Registers Size` – When A32 VME Slave Access is enabled, this parameter specifies the size, in bytes, of the VME access window. To disable access to these resources, specify a size of **`0`**. The default setting is **`0x400000`**.

`VIM/PXB Registers Offset` – When A32 VME Slave Access is enabled, this parameter specifies the offset from the VME Slave Base Address where access to the VIM Control/Status PLX9656, VIM Interface Registers, VIM Module, and PXB++ Registers begins. The default setting is **`0xA00000`**.

`VIM/PXB Registers Size` – When A32 VME Slave Access is enabled, this parameter specifies the size, in bytes, of the VME access window. To disable access to these resources, specify a size of **`0`**. The default setting is **`0x400000`**.

9.4 RedBoot Monitor Commands (continued)

9.4.11 vmeslaveconfig (continued)

Description (continued)**:**

`VIM BIFO/FPGA Access Offset` – When A32 VME Slave Access is enabled, this parameter specifies the offset from the VME Slave Base Address where access to the VIM BIFO/FPGA begins. The default setting is **`0xE00000`**.

`VIM/BIFO/FPGA Registers Size` – When A32 VME Slave Access is enabled, this parameter specifies the size, in bytes, of the VME access window. To disable access to these resources, specify a size of **`0`**. The default setting is **`0x2200000`**.

`SDRAM Access Offset` – This parameter specifies the offset from the VME Slave Base Address where access to SDRAM Base + **`0x100000`** begins. The default setting is **`0x3000000`**.

`SDRAM Access Size` – This parameter specifies the size, in bytes, of the VME access window. To disable access to these resources, specify a size of **`0`**. The default setting is **`0x1000000`**.

`Enable VME Slave Image 7 User Configurable` – This parameter enables Universe II's VME Slave Image 7. This slave image can be set up as needed depending on the user's application. Set to **`true`** (**`t`**) to enable or **`false`** (**`f`**) to disable this image. The default is false. Refer to the Tundra Universe II manual.

`VME Slave Image 7 Base Address` – When VME Slave Image 7 is enabled, this parameter specifies the complete VME address for this image.

`VME Slave Image 7 Destination Address` – When VME Slave Image 7 is enabled, this parameter specifies the starting PCI Destination Address.

`VME Slave Image 7 Size` – When VME Slave Image 7 is enabled, this parameter specifies the size of this image window.

`VME Slave Image 7 Control Register Setting` – When VME Slave Image 7 is enabled, this parameter specifies the VME Slave Image 7 control register setting. Refer to the Tundra Universe II manual.

9.4 RedBoot Monitor Commands (continued)

9.4.11 vmeslaveconfig (continued)

Arguments: **-l** List current VME Slave configuration settings. If this argument is omitted, you can modify each parameter as it is listed. To modify a value, first delete the old value and then enter the new value. To keep the displayed value, just use the **Enter** key.

Example: Display the current VME Slave parameters.

```
RedBoot> vmeslaveconfig -l

Enable VME Slave Extended Memory Map: true
Enable A24 VME Slave Address Space(true=A24,false=A32): false
VME Slave Base Address: 0x0
SDRAM Boot Memory Offset/Message Unit Offset: 0x1000
SDRAM Boot Memory Offset/Message Unit Size: 0x5000
GT-64260 Internal Registers Offset: 0x10000
GT-64260 Internal Registers Size: 0x10000
PMC Access Offset: 0x100000
PMC Access Size: 0x500000
Board Registers Offset: 0x600000
Board Registers Size: 0x400000
VIM/PXB Registers Offset: 0xA00000
VIM/PXB Registers Size: 0x400000
VIM BIFO/FPGA Access Offset: 0xE00000
VIM BIFO/FPGA Access Size: 0x2200000
SDRAM Base Address Offset: 0x3000000
SDRAM Access Size: 0x1000000
Enable VME Slave Image 7 - User Configurable: false
```

Appendix A: Model 4205 ID EEPROM

A.1 Introduction

The Model 4205 contains an Identification (ID) EEPROM, with a checksum stored in the last word. When the board is booted, the processor reads this ID EEPROM and generates a 16–bit checksum from the contents read. If the generated checksum does not match the stored value, the processor illuminates its red LED. (Even though the processor illuminates the red LED, the processor is fully functional.)

The contents of the ID EEPROM, plus the calculated checksum, are stored in Global SDRAM (see Section 8.3.1). The first word stored is a check flag based on the checksum: valid checksum (**0x00EE CODE**), bad checksum (**0x000B ADC5**), or no VIM module installed (**0xFFFF FFFE**).

You can read the Model 4205 serial ID EEPROM on the Galileo GT–64260A I^2C Interface at binary address **1010100b** (see Section 3.5.13).

NOTE: The ID EEPROM is shipped with the board identification information preset by Pentek, and should **NOT** be modified by the user.

A.2 ID EEPROM Format and Contents

Table A–1 on the next page shows the contents of the ID EEPROM, and an brief explanation of the contents of each pair of 16–bit words. The SDRAM location listed in the table is the address where each pair of words is stored in Global SDRAM.

The Temperature and Voltage words in the ID EEPROM are used to record the operating temperatures and voltages from sensors on the Model 4205 PCB. The temperature values are stored as degrees C and the voltages are stored as mVolts. (These are stored as binary values, represented by the hex codes **0xXXXX XXXX** in Table A–1.)

During normal operation, the processor reads the board's voltages and temperatures at regular intervals (about 1 minute). It stores the new values in the 'Readings' area of the Global SDRAM (see Section 8.3.1) and compares them with the latest values from the ID EEPROM. If the new readings are higher, the processor stores these new readings into the appropriate words of this ID EEPROM. When new voltage/temperature values are stored in this ID EEPROM, a new checksum is generated and stored in the last word of the EEPROM. (The Global SDRAM copy of the EEPROM is also updated when new values and checksum are stored into the ID EEPROM.)

A.2 EEPROM Format Example (continued)

Table A–1: EEPROM Example Code

EEPROM Word Location	SDRAM Location	Contents	Comments
00/01	0x0000 4100	0x00EE 4205	Valid data flag
02/03	0x0000 4104	0x4205 0000	Model, Model Extension (Extension currently unused)
04/05	0x0000 4108	0x0354 2001	PCB number — this is taken from the board
06/07	0x0000 410C	0x0001 FFFF	Assembly Rev (Rev A = **0001**, B = **0002**, etc.), Option #1 — up to 9 installed options may be specified, when there are no options, '**F**' appears in the entire field
08/09	0x0000 4110	0xFFFF FFFF	Option #2, Option #3
10/11	0x0000 4114	0xFFFF FFFF	Option #4, Option #5
12/13	0x0000 4118	0xFFFF FFFF	Option #6, Option #7
14/15	0x0000 411C	0xFFFF FFFF	Option #8, Option #9
16/17	0x0000 4120	0x0001 FFFF	PCB Rev (Rev A = **0001**, B = **0002**, etc.), **0xFFFF**
18/19	0x0000 4124	0x3042 051A	Firmware Model Number and Rev for certain elements of the board— the first seven hex digits are the Pentek Part Number, and the last hex digit is the Rev level of the part. For example **0x3042 051A** refers to the part number 030.42051, revision A
20/21	0x0000 4128	0x3042 052A	
22/23	0x0000 412C	0x3042 053A	
24/25	0x0000 4130	0x3042 054A	
26/27	0x0000 4134	0x3042 055A	
28/29	0x0000 4138	0x3042 056A	
30/31	0x0000 413C	0x3042 057A	
32/33	0x0000 4140	0xFFFF FFFF	Reserved
34/35	0x0000 4144	0xFFFF FFFF	
36/37	0x0000 4148	0xFFFF FFFF	
38/39	0x0000 414C	0xFFFF FFFF	
40/41	0x0000 4150	0xFFFF FFFF	
42/43	0x0000 4154	0xFFFF FFFF	
44/45	0x0000 4158	0xFFFF FFFF	
46/47	0x0000 415C	0xFFFF FFFF	
48/49	0x0000 4160	0xFFFF FFFF	
50/51	0x0000 4164	0xFFFF FFFF	
52/53	0x0000 4168	0xFFFF FFFF	
54/55	0x0000 416C	0xFFFF FFFF	
56/57	0x0000 4170	0xFFFF FFFF	
58/59	0x0000 4174	0xFFFF FFFF	
60/61	0x0000 4178	0xFFFF FFFF	
62/63	0x0000 417C	0x0000 XXXX	Word 62 should always be cleared (**0000**). Word 63 contains the 16–bit checksum of words 0 – 61.

Appendix B: Tundra® Universe II™ Registers

The following table lists the Tundra Universe II Control and Status Registers, their functions, and their memory address offset. This offset is from the Universe II Registers starting address in the applicable memory map: MPC745X Processor, Section 3.5.1; PCI Bus, Section 4.5.7; or Default VMEbus A16/A24 Slave, Section 5.4.1. For description of each register, please refer to the Tundra Universe II User Manual.

Table B–1: Universe II Control and Status Registers

Address Offset	Register Function	Name
0x000	PCI Configuration Space ID	PCI_ID
0x004	PCI Configuration Space Control and Status	PCI_CSR
0x008	PCI Configuration Class	PCI_CLASS
0x00C	PCI Configuration Miscellaneous 0	PCI_MISC0
0x010	PCI Configuration Base Address 0	PCI_BS0
0x014	PCI Configuration Base Address 1	PCI_BS1
0x018 – 0x024	PCI Unimplemented	
0x028 – 0x02C	PCI Reserved	
0x030	PCI Unimplemented	
0x034 – 0x038	PCI Reserved	
0x03C	PCI Configuration Miscellaneous 1	PCI_MISC1
0x040 – 0x0FF	PCI Unimplemented	
0x100	PCI Target Image 0 Control	LSI0_CTL
0x104	PCI Target Image 0 Base Address	LSI0_BS
0x108	PCI Target Image 0 Bound Address	LSI0_BD
0x10C	PCI Target Image 0 Translation Offset	LSI0_TO
0x110	Reserved	
0x114	PCI Target Image 1 Control	LSI1_CTL
0x118	PCI Target Image 1 Base Address	LSI1_BS
0x11C	PCI Target Image 1 Bound Address	LSI1_BD
0x120	PCI Target Image 1 Translation Offset	LSI1_TO
0x124	Reserved	
0x128	PCI Target Image 2 Control	LSI2_CTL
0x12C	PCI Target Image 2 Base Address	LSI2_BS
0x130	PCI Target Image 2 Bound Address	LSI2_BD
0x134	PCI Target Image 2 Translation Offset	LSI2_TO
0x138	Reserved	
0x13C	PCI Target Image 3 Control	LSI3_CTL
0x140	PCI Target Image 3 Base Address	LSI3_BS
0x144	PCI Target Image 3 Bound Address	LSI3_BD
0x148	PCI Target Image 3 Translation Offset	LSI3_TO
0x14C – 0x16C	Reserved	

Table B–1: Universe II Control and Status Registers		
0x170	Special Cycle Control	SCYC_CTL
0x174	Special Cycle PCI Bus Address	SCYC_ADDR
0x178	Special Cycle Swap/Compare Enable	SCYC_EN
0x17C	Special Cycle Compare Data	SCYC_CMP
0x180	Special Cycle Swap Data	SCYC_SWP
0x184	PCI Miscellaneous	LMISC
0x188	Special PCI Target Image	SLSI
0x18C	PCI Command Error Log	L_CMDERR
0x190	PCI Address Error Log	LAERR
0x194 – 0x19C	Reserved	
0x1A0	PCI Target Image 4 Control	LSI4_CTL
0x1A4	PCI Target Image 4 Base Address	LSI4_BS
0x1A8	PCI Target Image 4 Bound Address	LSI4_BD
0x1AC	PCI Target Image 4 Translation Offset	LSI4_TO
0x1B0	Reserved	
0x1B4	PCI Target Image 5 Control	LSI5_CTL
0x1B8	PCI Target Image 5 Base Address	LSI5_BS
0x1BC	PCI Target Image 5 Bound Address	LSI5_BD
0x1C0	PCI Target Image 5 Translation Offset	LSI5_TO
0x1C4	Reserved	
0x1C8	PCI Target Image 6 Control	LSI6_CTL
0x1CC	PCI Target Image 6 Base Address	LSI6_BS
0x1D0	PCI Target Image 6 Bound Address	LSI6_BD
0x1D4	PCI Target Image 6 Translation Offset	LSI6_TO
0x1D8	Reserved	
0x1DC	PCI Target Image 7 Control	LSI7_CTL
0x1E0	PCI Target Image 7 Base Address	LSI7_BS
0x1E4	PCI Target Image 7 Bound Address	LSI7_BD
0x1E8	PCI Target Image 7 Translation Offset	LSI7_TO
0x1EC – 0x1FC	Reserved	
0x200	DMA Transfer Control	DCTL
0x204	DMA Transfer Byte Count	DTBC
0x208	DMA PCI Bus Address	DLA
0x20C	Reserved	
0x210	DMA VMEbus Address	DVA
0x214	Reserved	
0x218	DMA Command Packet Pointer	DCPP
0x21C	Reserved	
0x220	DMA General Control and Status	DGCS
0x224	DMA Linked List Update Enable	D_LLUE
0x228 – 0x2FC	Reserved	
0x300	PCI Interrupt Enable	LINT_EN

Table B–1: Universe II Control and Status Registers		
0x304	PCI Interrupt Status	LINT_STAT
0x308	PCI Interrupt Map 0	LINT_MAP0
0x30C	PCI Interrupt Map 1	LINT_MAP1
0x310	VMEbus Interrupt Enable	VINT_EN
0x314	VMEbus Interrupt Status	VINT_STAT
0x318	VMEbus Interrupt Map 0	VINT_MAP0
0x31C	VMEbus Interrupt Map 1	VINT_MAP1
0x320	Interrupt Status/ID Out	STATID
0x324	VIRQ1 Status/ID Out	V1_STATID
0x328	VIRQ2 Status/ID Out	V2_STATID
0x32C	VIRQ3 Status/ID Out	V3_STATID
0x330	VIRQ4 Status/ID Out	V4_STATID
0x334	VIRQ5 Status/ID Out	V5_STATID
0x338	VIRQ6 Status/ID Out	V6_STATID
0x33C	VIRQ7 Status/ID Out	V7_STATID
0x340	PCI Interrupt Map 2	LINT_MAP2
0x344	VMEbus Interrupt Map 2	VINT_MAP2
0x348	Mailbox 0	MBOX0
0x34C	Mailbox 1	MBOX1
0x350	Mailbox 2	MBOX2
0x354	Mailbox 3	MBOX3
0x358	Semaphore 0	SEMA0
0x35C	Semaphore 1	SEMA1
0x360 – 0x3FC	Reserved	
0x400	Master Control	MAST_CTL
0x404	Miscellaneous Control	MISC_CTL
0x408	Miscellaneous Status	MISC_STAT
0x40C	User AM Codes	USER_AM
0x410 – 0xEFC	Reserved	
0xF00	VMEbus Slave Image 0 Control	VSI0_CTL
0xF04	VMEbus Slave Image 0 Base Address	VSI0_BS
0xF08	VMEbus Slave Image 0 Bound Address	VSI0_BD
0xF0C	VMEbus Slave Image 0 Translation Offset	VSI0_TO
0xF10	Reserved	
0xF14	VMEbus Slave Image 1 Control	VSI1_CTL
0xF18	VMEbus Slave Image 1 Base Address	VSI1_BS
0xF1C	VMEbus Slave Image 1 Bound Address	VSI1_BD
0xF20	VMEbus Slave Image 1 Translation Offset	VSI1_TO
0xF24	Reserved	
0xF28	VMEbus Slave Image 2 Control	VSI2_CTL
0xF2C	VMEbus Slave Image 2 Base Address	VSI2_BS
0xF30	VMEbus Slave Image 2 Bound Address	VSI2_BD

Table B–1: Universe II Control and Status Registers		
0xF34	VMEbus Slave Image 2 Translation Offset	VSI2_TO
0xF38	Reserved	
0xF3C	VMEbus Slave Image 3 Control	VSI3_CTL
0xF40	VMEbus Slave Image 3 Base Address	VSI3_BS
0xF44	VMEbus Slave Image 3 Bound Address	VSI3_BD
0xF48	VMEbus Slave Image 3 Translation Offset	VSI3_TO
0xF4C – 0xF60	Reserved	
0xF64	Location Monitor Control	LM_CTL
0xF68	Location Monitor Base Address	LM_BS
0xF6C	Reserved	
0xF70	VMEbus Register Access Image Control	VRAI_CTL
0xF74	VMEbus Register Access Image Base Address	VRAI_BS
0xF78 – 0xF7C	Reserved	
0xF80	VMEbus CSR Control	VCSR_CTL
0xF84	VMEbus CSR Translation Offset	VCSR_TO
0xF88	VMEbus AM Code Error Log	V_AMERR
0xF8C	VMEbus Address Error Log	VAERR
0xF90	VMEbus Slave Image 4 Control	VSI4_CTL
0xF94	VMEbus Slave Image 4 Base Address	VSI4_BS
0xF98	VMEbus Slave Image 4 Bound Address	VSI4_BD
0xF9C	VMEbus Slave Image 4 Translation Offset	VSI4_TO
0xFA0	Reserved	
0xFA4	VMEbus Slave Image 5 Control	VSI5_CTL
0xFA8	VMEbus Slave Image 5 Base Address	VSI5_BS
0xFAC	VMEbus Slave Image 5 Bound Address	VSI5_BD
0xFB0	VMEbus Slave Image 5 Translation Offset	VSI5_TO
0xFB4	Reserved	
0xFB8	VMEbus Slave Image 6 Control	VSI6_CTL
0xFBC	VMEbus Slave Image 6 Base Address	VSI6_BS
0xFC0	VMEbus Slave Image 6 Bound Address	VSI6_BD
0xFC4	VMEbus Slave Image 6 Translation Offset	VSI6_TO
0xFC8	Reserved	
0xFCC	VMEbus Slave Image 7 Control	VSI7_CTL
0xFD0	VMEbus Slave Image 7 Base Address	VSI7_BS
0xFD4	VMEbus Slave Image 7 Bound Address	VSI7_BD
0xFD8	VMEbus Slave Image 7 Translation Offset	VSI7_TO
0xFDC – 0xFEC	Reserved	
0xFF0	VME CR/CSR Reserved	
0xFF4	VMEbus CSR Bit Clear	VCSR_CLR
0xFF8	VMEbus CSR Bit Set	VCSR_SET
0xFFC	VMEbus CSR Base Address	VCSR_BS

Appendix C: PCI Configuration Space Registers

C.1 Introduction

This Appendix shows the pre–defined header region of PCI Configuration Space, for the basic Model 4205 Primary and Secondary PCI buses, and for Option 150/151/152 Tertiary and Quaternary PCI buses.

C.2 Primary and Secondary PCI Buses

The PCI interface on the Model 4205 Primary and Secondary PCI buses uses Header Type 00h, which has the layout shown in the table below. The PCI Configuration Space consists of fields that uniquely identify the device and allow the device to be generi–cally controlled. The first 16 Bytes (four words) are defined the same for all types of PCI devices. The remaining words can have different layouts depending on the base function that the device supports.

The table lists the PCI Configuration Space registers, their functions, and their base address in Configuration Space. For detailed description of each register for the PCI bus interfaces of Model 4205, please refer to the PLX PCI 9656 Data Book.

Table C–1: PCI Configuration Space Header (Type 00h)

Address	Register Function			
0x00	Device ID		Vendor ID	
0x04	Status		Command	
0x08	Class Code			Revision ID
0x0C	BIST	Header Type	Latency Timer	Cache Line Size
0x10	Base Address Register 0 (BAR0)			
0x14	Base Address Register 1 (BAR1)			
0x18	Base Address Register 2 (BAR2)			
0x1C - 0x2B	Reserved [1]			
0x2C	Subsystem ID		Subsystem Vendor ID	
0x30	Reserved [1]			
0x34	Reserved [1]			New Capability
0x38	Reserved [1]			
0x3C	Max Latency	Min Grant	Interrupt Pin	Interrupt Line

1 – These registers have different uses depending on the Device plugged into this PCI bus. Refer to the PLX PCI 9656 Data Book and the applicable Device's Operating Manual/Data Book for further information on the use of these registers.

C.3 Option 150/151/152 PCI Buses

The Option 150/151/152 PMC Extender Board for the Model 4205 provides additional PCI Mezzanine Card (PMC) sites. It extends the baseboard PMC connector into a second VMEbus slot, where up to three PMC modules can be mounted. This additional PMC support is provided without interfering with baseboard VIM mezzanine sites, so that up three PMC modules and two VIM–2 modules can be mounted on a single baseboard simultaneously.

Option 150/151/152 adds two more PCI buses to the Model 4205. The following is a simplified block diagram of the Model 9220 PMC Extender, showing these additional Tertiary and Quaternary PCI buses.

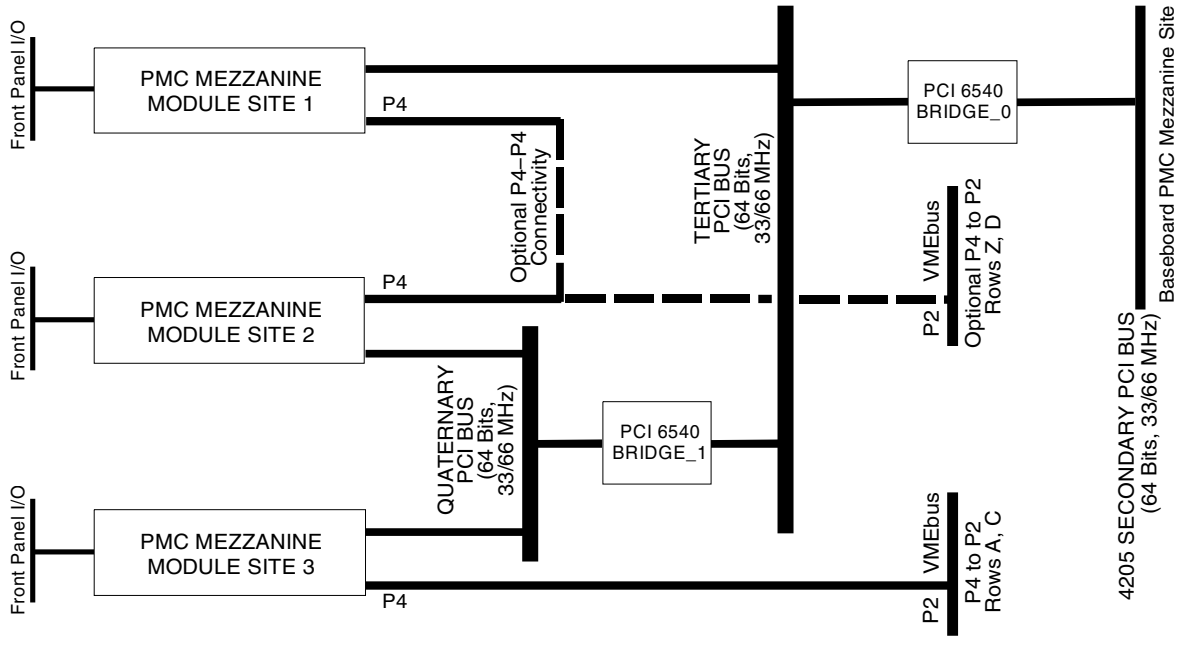


Figure C–1: Option 150/151/152 Extender Block Diagram

NOTE: The Option 150/151/152 PMC Extender Board is documented separately from the Model 4205, as the Model 9220 PMC Extender Board. Refer to the Model 9220 Operating Manual (which is supplied with your Model 4205 Option 150/151/152) for detailed description of the Extender Board setup and operation.

C.3 Option 150/151/152 PCI Buses (continued)

The Option 150/151/152 PMC Extender Board can be set for Transparent or Non–Transparent operation using jumper **JB3** on the Extender Board. Refer to the Model 9220 Operating Manual supplied with Option 150/151/152 for these jumper settings, and to the PLX PCI 6540 Data Book for description of PCI Transparent or Non–Transparent operation.

When the Extender Board is set for Non–Transparent mode of operation, the PCI Configuration Registers on the Model 4205 Tertiary and Quaternary PCI buses uses Header Type 00h, which has the layout shown in Table C–1 on page C–1.

When the Extender Board is set for Transparent mode of operation, the PCI interface on the Model 4205 Tertiary and Quaternary PCI buses uses a Transparent Mode PCI Configuration Register address map. For detailed description of the PCI Configuration Registers on the Model 4205 Tertiary and Quaternary PCI bus interfaces in Transparent Mode, please refer to the PLX PCI 6540 Data Book.

This page is intentionally blank

Appendix D: Option 005/010 RACEway Operation

D.1 Overview

Model 4205 Option 005 provides a single RACEway RACE++ interface to the Primary PCI Bus. Option 010 includes the Option 005 RACE++ interface and adds a second RACE++ interface to the Secondary PCI Bus.

- With Option 005, the Model 4205 allows data transfers up to 267 MBytes/second with other RACEway–equipped VME boards. Interlink Modules (ILKs or ILKPs) connect the RACE++ (RACEway) busses of two or more VME boards.

- With Option 010, the Model 4205 allows data transfers up to 533 MBytes/second with other RACEway–equipped VME boards. Option 010 requires RACE++ Dual–Port Interlink Modules (DLKPs) to connect to the RACE++ busses of other VME boards.

This manual assumes that the reader is familiar with the RACEway interface and its protocols. The complete RACEway Interlink standard, ANSI/VITA 5.1–1999, is available for purchase from the VMEbus International Trade Association at their web site, **http://www.vita.com**. A Raceway Interlink Functional Specification is available from Mercury Computer Systems's web site, **http://www.mc.com**.

D.2 General Operation

When either Option 005 or 010 is installed, the Model 4205 supports both outgoing (Master) and incoming (Slave) RACEway operations using the PCI bus. These options provide translation of PCI addresses to or from RACEway Route and Address Words for the PCI address space of the 4205. Any Model 4205 device that has access to the PCI bus can use this RACEway interface.

NOTE: The optional Raceway interfaces can generate Broadcast/Multicast writes, but cannot receive them.

Pentek board support software (see Section 1.12) is available for control and operation of all RACEway functions. Refer to the Model 4996 Option 205 Board Support Package manual (Pentek part #803.42050) and the Model 4999 Option 205 ReadyFlow Board Support Libraries manual (Pentek part #801.42050) for description of these capabilities.

D.3 RACEway Transfer Rates

The transfer rates between the Model 4205 MPC745X Processor and other RACEway–equipped VME boards are as follows:

Writes to RACEway:

- Option 005: (TBD)
- Option 010: (TBD)

Reads from RACEway:

- Option 005: (TBD)
- Option 010: (TBD)

Appendix E: RedBoot™ User's Guide

E.1 Introduction

The following pages are a reprint of the RedBoot™ User's Guide, Chapter 2, RedBoot Commands and Examples, extracted from the eCos Reference Manual.

eCos Reference Manual Copyright

Copyright © 1998, 1999, 2000, 2001, 2002, 2003 by Red Hat, Inc. Nick Garnett (eCosCentric) Jonathan Larmour (eCosCentric) Andrew Lunn (Ascom) Gary Thomas (MLB Associates) Bart Veer (eCosCentric).

Documentation licensing terms

This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at **http://www.opencontent.org/openpub/**).

Warranty

eCos and RedBoot are open source software, covered by a modified version of the GNU General Public Licence (**http://www.gnu.org/copyleft/gpl.html**), and you are welcome to change it and/or distribute copies of it under certain conditions. See **http://sources.redhat.com/ecos/license–overview.html** for more information about the license.

eCos and RedBoot software have NO WARRANTY.

NOTE: In addition to the RedBoot commands shown in this appendix, Pentek has augmented RedBoot to provide several user commands particular to the Model 4205. Refer to Chapter 9 for description of these commands.

This page is intentionally blank

Chapter 2. RedBoot Commands and Examples

Introduction

RedBoot provides three basic classes of commands:

- Program loading and execution
- Flash image and configuration management
- Miscellaneous commands

Given the extensible and configurable nature of eCos and RedBoot, there may be extended or enhanced sets of commands available.

The basic format for commands is:

```
RedBoot> COMMAND [-S]... [-s val]... operand
```

Commands may require additional information beyond the basic command name. In most cases this additional information is optional, with suitable default values provided if they are not present.

Format	Description	Example
-S	A boolean switch; the behavior of the command will differ, depending on the presence of the switch. In this example, the **`-f`** switch indicates that a complete initialization of the FIS data should be performed. There may be many such switches available for any given command and any or all of them may be present, in any order.	`RedBoot> fis init -f`
-s *`val`*	A qualified value; the letter "s" introduces the value, qualifying it's meaning. In the example, **`-b 0x100000`** specifies where the memory dump should begin. There may be many such switches available for any given command and any or all of them may be present, in any order.	`RedBoot> dump -b 0x100000 -l 0x20`

Format	Description	Example
operand	A simple value; some commands require a single parameter for which an additional **-X** switch would be redundant. In the example, **JFFS2** is the name of a flash image. The image name is always required, thus is no need to qualify it with a switch. Note that any un-qualified operand must always appear at the end of the command.	RedBoot> **fis delete JFFS2**

The list of available commands, and their syntax, can be obtained by typing **help** at the command line:

```
RedBoot> help
Manage aliases kept in FLASH memory
      alias name [value]
Set/Query the system console baud rate
      baudrate [-b <rate>]
Manage machine caches
      cache [ON | OFF]
Display/switch console channel
      channel [-1|<channel number>]
Display disk partitions
      disks
Display (hex dump) a range of memory
      dump -b <location> [-l <length>] [-s]
Manage flash images
      fis {cmds}
Manage configuration kept in FLASH memory
      fconfig [-i] [-l] [-n] [-f] [-d] | [-d] nickname [value]
Execute code at a location
      go [-w <timeout>] [entry]
Help about help?
      help [<topic>]
Set/change IP addresses
      ip_address [-l <local_ip_address>] [-h <server_address>]
Load a file
      load [-r] [-v] [-d] [-c <channel>] [-h <host>] [-m {TFTP | HTTP | {x|y}MODEM | disk}]
      [-b <base_address>] <file_name>
Network connectivity test
      ping [-v] [-n <count>] [-t <timeout>] [-i <IP_addr]
      -h <host>
Reset the system
      reset
Display RedBoot version information
      version
Display (hex dump) a range of memory
      x -b <location> [-l <length>] [-s]
```

Commands can be abbreviated to their shortest unique string. Thus in the list above, **d,du,dum** and dump are all valid for the **dump** command. The **fconfig** command can be abbreviated **fc**, but **f** would be ambiguous with **fis**.

There is one additional, special command. When RedBoot detects '$' or '+' (unless escaped via '\') in a command, it switches to GDB protocol mode. At this point, the eCos GDB stubs take over, allowing connections from a GDB host. The only way to get back to RedBoot from GDB mode is to restart the platform.

NOTE: Multiple commands may be entered on a single line, separated by the semi-colon ";" character.

The standard RedBoot command set is structured around the bootstrap environment. These commands are designed to be simple to use and remember, while still providing sufficient power and flexibility to be useful. No attempt has been made to render RedBoot as the end-all product. As such, things such as the debug environment are left to other modules, such as GDB stubs, which are typically included in RedBoot.

The command set may be also be extended on a platform basis.

Common Commands

alias

Name

`alias` — Manipulate command line aliases

Synopsis

alias { *name*} [*value*]

Arguments

Name	Type	Description	Default
name	Name	The name for this alias.	*none*
value	String	Replacement value for the alias.	*none*

Description

The **alias** command is used to maintain simple command line aliases. These aliases are shorthand for longer expressions. When the pattern %{name} appears in a command line, including in a script, the corresponding value will be substituted. Aliases may be nested.

If no value is provided, then the current value of the alias is displayed.

If the system supports non-volatile configuration data via the **fconfig** command (see the Section called *Persistent State Flash-based Configuration and Control* in Chapter 2), then the value will be saved and used when the system is reset.

Examples

Set an alias.

```
RedBoot> alias joe "This is Joe"
Update RedBoot non-volatile configuration - continue (y/n)? n
```

Display an alias.

```
RedBoot> alias joe
'joe' = 'This is Joe'
```

Use an alias. Note: the "=" command simply echoes the command to to console.

```
RedBoot> = %{joe}
This is Joe
```

Aliases can be nested.

```
RedBoot> alias frank "Who are you? %{joe}"
Update RedBoot non-volatile configuration - continue (y/n)? n
RedBoot> = %{frank}
Who are you? This is Joe
```

Notice how the value of %{frank} changes when %{joe} is changed since the value of %{joe} is not evaluated until %{frank} is evaluated.

```
RedBoot> alias joe "This is now Josephine"
Update RedBoot non-volatile configuration - continue (y/n)? n
RedBoot> = %{frank}
Who are you? This is now Josephine
```

baudrate

Name

`baudrate` — Set the baud rate for the system serial console

Synopsis

baudrate [-b *`rate`*]

Arguments

Name	Type	Description	Default
-b *`rate`*	Number	The baud rate to use for the serial console.	*none*

Description

The **baudrate** command sets the baud rate for the system serial console.

If no value is provided, then the current value of the console baud rate is displayed.

If the system supports non-volatile configuration data via the **fconfig** command (see the Section called *Persistent State Flash-based Configuration and Control* in Chapter 2), then the value will be saved and used when the system is reset.

Examples

Show the current baud rate.

```
RedBoot> baudrate
Baud rate = 38400
```

Change the console baud rate. In order to make this operation safer, there will be a slight pause after the first message to give you time to change to the new baud rate. If it doesn’t work, or a less than affirmative answer is given to the "continue" prompt, then the baud rate will revert to the current value. Only after the baud rate has been firmly established will *RedBoot* give you an opportunity to save the value in persistent storage.

```
RedBoot> baudrate -b 57600
Baud rate will be changed to 57600 - update your settings
Device baud rate changed at this point
Baud rate changed to 57600 - continue (y/n)? y
```

```
Update RedBoot non-volatile configuration - continue (y/n)? n
```

cache

Name

`cache` — Control hardware caches

Synopsis

cache [on | off]

Arguments

Name	Type	Description	Default
on		Turn the caches on	*none*
off		Turn the caches off	*none*

Description

The **cache** command is used to manipulate the caches on the processor.

With no options, this command specifies the state of the system caches.

When an option is given, the caches are turned off or on appropriately.

Examples

Show the current cache state.

```
RedBoot> cache
Data cache: On, Instruction cache: On
```

Disable the caches.

```
RedBoot> cache off
RedBoot> cache
Data cache: Off, Instruction cache: Off
```

Enable the caches.

```
RedBoot> cache on
```

```
RedBoot> cache
Data cache: On, Instruction cache: On
```

channel

Name

`channel` — Select the system console channel

Synopsis

channel [-1 | *`channel_number`*]

Arguments

Name	Type	Description	Default
-1		Reset the console channel	*none*
channel_number	Number	Select a channel	*none*

Description

With no arguments, the **channel** command displays the current console channel number.

When passed an argument of 0 upward, this command switches the console channel to that channel number. The mapping between channel numbers and physical channels is platform specific but will typically be something like channel 0 is the first serial port, channel 1 is the second, etc.

When passed an argument of -1, this command reverts RedBoot to responding to whatever channel receives input first, as happens when RedBoot initially starts execution.

Examples

Show the current channel.

```
RedBoot> channel
Current console channel id: 0
```

Change to an invalid channel.

```
RedBoot> channel 99
**Error: bad channel number '99'
```

Revert to the default channel setting (any console mode).

```
RedBoot> channel -1
```

cksum

Name

`cksum` — Compute POSIX checksums

Synopsis

cksum {-b *`location`*} {-l *`length`*}

Arguments

Name	Type	Description	Default
-b *`location`*	Memory address	Location in memory for stat of data.	*none*
-l *`length`*	Number	Length of data	*none*

Description

Computes the POSIX checksum on a range of memory (either RAM or FLASH). The values printed (decimal cksum, decimal length, hexadecimal cksum, hexadecimal length) can be compared with the output from the Linux program ’cksum’.

Examples

Checksum a buffer.

```
RedBoot> cksum -b 0x100000 -l 0x100
POSIX cksum = 3286483632 256 (0xc3e3c2b0 0x00000100)
```

Checksum an area of memory after loading a file. Note that the base address and length parameters are provided by the preceding load command.

```
RedBoot> load -r -b %{FREEMEMLO} redboot.bin
Raw file loaded 0x06012800-0x0602f0a8
RedBoot> cksum
Computing cksum for area 0x06012800-0x0602f0a8
POSIX cksum = 2092197813 116904 (0x7cb467b5 0x0001c8a8)
```

disks

Name

`disks` — List available disk partitions.

Synopsis

disks

Arguments

None.

Description

The **disks** command is used to list disk partitions recognized by RedBoot.

Examples

Show what disk partitions are available.

```
RedBoot> disks
hda1     Linux Swap
hda2     Linux
00100000: 00 3E 00 06 00 06 00 06  00 00 00 00 00 00 00 00  |.>..............|
00100010: 00 00 00 78 00 70 00 60  00 60 00 60 00 60 00 60  |...x.p.‘.‘.‘.‘.‘|
```

dump

Name

`dump` — Display memory.

Synopsis

dump {-b *`location`*} [-l *`length`*] [-s] [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-b *`location`*	Memory address	Location in memory for start of data.	*none*
-l *`length`*	Number	Length of data	32
-s	Boolean	Format data using Motorola S-records.	
-1		Access one byte (8 bits) at a time. Only the least significant 8 bits of the pattern will be used.	-1
-2		Access two bytes (16 bits) at a time. Only the least significant 16 bits of the pattern will be used.	-1
-4		Access one word (32 bits) at a time.	-1

Description

Display a range of memory on the system console.

The **x** is a synonym for **dump**.

Note that this command could be detrimental if used on memory mapped hardware registers.

The memory is displayed at most sixteen bytes per line, first as the raw hex value, followed by an ASCII interpretation of the data.

Examples

Display a buffer, one byte at a time.

```
RedBoot> mfill -b 0x100000 -l 0x20 -p 0xDEADFACE
RedBoot> x -b 0x100000
00100000: CE FA AD DE CE FA AD DE  CE FA AD DE CE FA AD DE  |................|
00100010: CE FA AD DE CE FA AD DE  CE FA AD DE CE FA AD DE  |................|
```

Display a buffer, one short (16 bit) word at a time. Note in this case that the ASCII interpretation is suppressed.

```
RedBoot> dump -b 0x100000 -2
00100000: FACE DEAD FACE DEAD  FACE DEAD FACE DEAD
00100010: FACE DEAD FACE DEAD  FACE DEAD FACE DEAD
```

Display a buffer, one word (32 bit) word at a time. Note in this case that the ASCII interpretation is suppressed.

```
RedBoot> dump -b 0x100000 -4
00100000: DEADFACE DEADFACE DEADFACE DEADFACE
00100010: DEADFACE DEADFACE DEADFACE DEADFACE
```

Display the same buffer, using Motorola S-record format.

```
RedBoot> dump -b 0x100000 -s
S31500100000CEFAADDECEFAADDECEFAADDECEFAADDE8E
S31500100010CEFAADDECEFAADDECEFAADDECEFAADDE7E
```

Display a buffer, with visible ASCII strings.

```
RedBoot> d -b 0xfe00b000 -l 0x80
0xFE00B000: 20 25 70 0A 00 00 00 00  41 74 74 65 6D 70 74 20 | %p.....Attempt |
0xFE00B010: 74 6F 20 6C 6F 61 64 20  53 2D 72 65 63 6F 72 64 |to load S-record|
0xFE00B020: 20 64 61 74 61 20 74 6F  20 61 64 64 72 65 73 73 | data to address|
0xFE00B030: 3A 20 25 70 20 5B 6E 6F  74 20 69 6E 20 52 41 4D |: %p [not in RAM|
0xFE00B040: 5D 0A 00 00 2A 2A 2A 20  57 61 72 6E 69 6E 67 21 |]...*** Warning!|
0xFE00B050: 20 43 68 65 63 6B 73 75  6D 20 66 61 69 6C 75 72 | Checksum failur|
0xFE00B060: 65 20 2D 20 41 64 64 72  3A 20 25 6C 78 2C 20 25 |e - Addr: %lx, %|
0xFE00B070: 30 32 6C 58 20 3C 3E 20  25 30 32 6C 58 0A 00 00 |02lX <> %02lX...|
0xFE00B080: 45 6E 74 72 79 20 70 6F  69 6E 74 3A 20 25 70 2C |Entry point: %p,|
```

help

Name

`help` — Display help on available commands

Synopsis

help [*topic*]

Arguments

Name	Type	Description	Default
topic	String	Which command to provide help for.	All commands

Description

The **help** command displays information about the available RedBoot commands. If a *topic* is given, then the display is restricted to information about that specific command.

If the command has sub-commands, e.g. **fis**, then the topic specific display will print additional information about the available sub-commands. special (ICMP) packets to a specific host. These packets should be automatically returned by that host. The command will indicate how many of these round-trips were successfully completed.

Examples

Show generic help. Note that the contents of this display will depend on the various configuration options for RedBoot when it was built.

```
RedBoot> help
Manage aliases kept in FLASH memory
   alias name [value]
Manage machine caches
   cache [ON | OFF]
Display/switch console channel
   channel [-1|<channel number>]
Compute a 32bit checksum [POSIX algorithm] for a range of memory
   cksum -b <location> -l <length>
Display (hex dump) a range of memory
   dump -b <location> [-l <length>] [-s] [-1|2|4]
Manage FLASH images
   fis {cmds}
Manage configuration kept in FLASH memory
   fconfig [-i] [-l] [-n] [-f] [-d] | [-d] nickname [value]
```

```
Execute code at a location
   go [-w <timeout>] [entry]
Help about help?
   help [<topic>]
Set/change IP addresses
   ip_address [-l <local_ip_address>] [-h <server_address>]
Load a file
   load [-r] [-v] [-d] [-h <host>] [-m {TFTP | HTTP | {x|y}MODEM -c <channel_number>}]
        [-b <base_address>] <file_name>
Compare two blocks of memory
   mcmp -s <location> -d <location> -l <length> [-1|-2|-4]
Fill a block of memory with a pattern
   mfill -b <location> -l <length> -p <pattern> [-1|-2|-4]
Network connectivity test
   ping [-v] [-n <count>] [-l <length>] [-t <timeout>] [-r <rate>]
        [-i <IP_addr>] -h <IP_addr>
Reset the system
   reset
Display RedBoot version information
   version
Display (hex dump) a range of memory
   x -b <location> [-l <length>] [-s] [-1|2|4]
```

Help about a command with sub-commands.

```
RedBoot> help fis
Manage FLASH images
   fis {cmds}
Create an image
  fis create -b <mem_base> -l <image_length> [-s <data_length>]
      [-f <flash_addr>] [-e <entry_point>] [-r <ram_addr>] [-n] <name>
Display an image from FLASH Image System [FIS]
  fis delete name
Erase FLASH contents
  fis erase -f <flash_addr> -l <length>
Display free [available] locations within FLASH Image System [FIS]
  fis free
Initialize FLASH Image System [FIS]
  fis init [-f]
Display contents of FLASH Image System [FIS]
  fis list [-c] [-d]
Load image from FLASH Image System [FIS] into RAM
  fis load [-d] [-b <memory_load_address>] [-c] name
Write raw data directly to FLASH
  fis write -f <flash_addr> -b <mem_base> -l <image_length>
```

ip_address

Name

`ip_address` — Set IP addresses

Synopsis

ip_address [-l *`local_IP_address`*] [-h *`server_IP_address`*] [-d *`DNS_server_IP_address`*]

Arguments

Name	Type	Description	Default
-l *`local_IP_address`*	Numeric IP or DNS name	The IP address RedBoot should use.	*none*
-h *`server_IP_address`*	Numeric IP or DNS name	The IP address of the default server. Use of this address is implied by other commands, such as **load**.	*none*
-d *`DNS_server_IP_address`*	Numeric IP or DNS name	The IP address of the DNS server.	*none*

Description

The **ip_address** command is used to show and/or change the basic IP addresses used by RedBoot. IP addresses may be given as numeric values, e.g. 192.168.1.67, or as symbolic names such as www.redhat.com if DNS support is enabled.

The `-l` option is used to set the IP address used by the target device.

The `-h` option is used to set the default server address, such as is used by the **load** command.

The `-d` option is used to set the default DNS server address which is used for resolving symbolic network addresses. Note that an address of 0.0.0.0 will disable DNS lookups.

Examples

Display the current network settings.

```
RedBoot> ip_address
IP: 192.168.1.31, Default server: 192.168.1.101, DNS server IP: 0.0.0.0
```

Change the DNS server address.

```
RedBoot> ip_address -d 192.168.1.101
IP: 192.168.1.31, Default server: 192.168.1.101, DNS server IP: 192.168.1.101
```

Change the default server address.

```
RedBoot> ip_address -h 192.168.1.104
IP: 192.168.1.31, Default server: 192.168.1.104, DNS server IP: 192.168.1.101
```

load

Name

`load` — Download programs or data to the RedBoot platform

Synopsis

load [-v] [-d] [-r] [-m [[xmodem | ymodem] | tftp | disk]] [-h *`server_IP_address`*] [-b *`location`*] [-c *`channel`*] [*`file_name`*]

Arguments

Name	Type	Description	Default
-v	Boolean	Display a small spinner (indicator) while the download is in progress. This is just for feedback, especially during long loads. Note that the option has no effect when using a serial download method since it would interfere with the protocol.	*quiet*
-d	Boolean	Decompress data stream (gzip data)	*non-compressed data*
-r	Boolean	Raw (or binary) data	*formatted (S-records, ELF image, etc)*
-m tftp		Transfer data via the network using TFTP protocol.	TFTP
-m http		Transfer data via the network using HTTP protocol.	TFTP
-m xmodem		Transfer data using *X-modem* protocol.	TFTP
-m ymodem		Transfer data using *Y-modem* protocol.	TFTP
-m disk		Transfer data from a local disk.	TFTP
-h *`server_IP_address`*	Numeric IP or DNS name	The IP address of the TFTP or HTTP server.	Value set by **ip_address**

Name	Type	Description	Default
-b *location*	Number	Address in memory to load the data. Formatted data streams will have an implied load address which this option may override.	*Depends on data format*
-c *channel*	Number	Specify which I/O channel to use for download. This option is only supported when using either xmodem or ymodem protocol.	*Depends on data format*
file_name	String	The name of the file on the TFTP or HTTP server or the local disk. Details of how this is specified for TFTP are host-specific. For local disk files, the name must be in *disk*: *filename* format. The disk portion must match one of the disk names listed by the **disks** command.	*None*

Description

The **load** command is used to download data into the target system. Data can be loaded via a network connection, using either the TFTP or HTTP protocols, or the console serial connection using the X/Y modem protocol. Files may also be loaded directly from local filesystems on disk. Files to be downloaded may either be executable images in ELF executable program format, Motorola S-record (SREC) format or raw data.

Examples

Download a Motorola S-record (or ELF) image, using TFTP, specifying the base memory address.

```
RedBoot> load redboot.ROM -b 0x8c400000
Address offset = 0x0c400000
Entry point: 0x80000000, address range: 0x80000000-0x8000fe80
```

Download a Motorola S-record (or ELF) image, using HTTP, specifying the host [server] address.

```
RedBoot> load /redboot.ROM -m HTTP -h 192.168.1.104
Address offset = 0x0c400000
Entry point: 0x80000000, address range: 0x80000000-0x8000fe80
```

Load an ELF file from /dev/hda1 which should be an EXT2 partition:

```
RedBoot> load -mode disk hda1:hello.elf
Entry point: 0x00020000, address range: 0x00020000-0x0002fd70
```

mcmp

Name

`mcmp` — Compare two segments of memory

Synopsis

mcmp {-s *`location1`*} {-d *`location1`*} {-l *`length`*} [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-s *`location1`*	Memory address	Location for start of data.	*none*
-d *`location2`*	Memory address	Location for start of data.	*none*
-l *`length`*	Number	Length of data	*none*
-1		Access one byte (8 bits) at a time. Only the least significant 8 bits of the pattern will be used.	-4
-2		Access two bytes (16 bits) at a time. Only the least significant 16 bits of the pattern will be used.	-4
-4		Access one word (32 bits) at a time.	-4

Description

Compares the contents of two ranges of memory (RAM, ROM, FLASH, etc).

Examples

Compare two buffers which match (result is *quiet*).

```
RedBoot> mfill -b 0x100000 -l 0x20 -p 0xDEADFACE
RedBoot> mfill -b 0x200000 -l 0x20 -p 0xDEADFACE
RedBoot> mcmp -s 0x100000 -d 0x200000 -l 0x20
```

Compare two buffers which don’t match. Only the first non-matching element is displayed.

```
RedBoot> mcmp -s 0x100000 -d 0x200000 -l 0x30 -2
Buffers don’t match - 0x00100020=0x6000, 0x00200020=0x0000
```

mfill

Name

`mfill` — Fill RAM with a specified pattern

Synopsis

mfill {-b *`location`*} {-l *`length`*} {-p *`value`*} [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-b *`location`*	Memory address	Location in memory for start of data.	*none*
-l *`length`*	Number	Length of data	*none*
-p *`pattern`*	Number	Data value to fill with	0
-1		Access one byte (8 bits) at a time. Only the least significant 8 bits of the pattern will be used.	-4
-2		Access two bytes (16 bits) at a time. Only the least significant 16 bits of the pattern will be used.	-4
-4		Access one word (32 bits) at a time.	-4

Description

Fills a range of memory with the given pattern.

Examples

Fill a buffer with zeros.

```
RedBoot> x -b 0x100000 -l 0x20
00100000: 00 3E 00 06 00 06 00 06  00 00 00 00 00 00 00 00  |.>..............|
00100010: 00 00 00 78 00 70 00 60  00 60 00 60 00 60 00 60  |...x.p.‘.‘.‘.‘.‘|
RedBoot> mfill -b 0x100000 -l 0x20
RedBoot> x -b 0x100000 -l 0x20
00100000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

```
00100010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Fill a buffer with a pattern.

```
RedBoot> mfill -b 0x100000 -l 0x20 -p 0xDEADFACE
RedBoot> x -b 0x100000 -l 0x20
00100000: CE FA AD DE CE FA AD DE  CE FA AD DE CE FA AD DE  |................|
00100010: CE FA AD DE CE FA AD DE  CE FA AD DE CE FA AD DE  |................|
```

ping

Name

`ping` — Verify network connectivity

Synopsis

ping [-v] [-i *`local_IP_address`*] [-l *`length`*] [-n *`count`*] [-t *`timeout`*] [-r *`rate`*] {-h *`server_IP_address`*}

Arguments

Name	Type	Description	Default
-v	Boolean	Be verbose, displaying information about each packet sent.	*quiet*
-n *`local_IP_address`*	Number	Controls the number of packets to be sent.	10
-i *`local_IP_address`*	Numeric IP or DNS name	The IP address RedBoot should use.	Value set by **ip_address**
-h *`server_IP_address`*	Numeric IP or DNS name	The IP address of the host to contact.	*none*
-l *`length`*	Number	The length of the ICMP data payload.	64
-r *`length`*	Number	How fast to deliver packets, i.e. time between successive sends. A value of 0 sends packets as quickly as possible.	1000ms (1 second)
-t *`length`*	Number	How long to wait for the round-trip to complete, specified in milliseconds.	1000ms (1 second)

Description

The **ping** command checks the connectivity of the local network by sending special (ICMP) packets to a specific host. These packets should be automatically returned by that host. The command will indicate how many of these round-trips were successfully completed.

Examples

Test connectivity to host 192.168.1.101.

```
RedBoot> ping -h 192.168.1.101
Network PING - from 192.168.1.31 to 192.168.1.101
PING - received 10 of 10 expected
```

Test connectivity to host 192.168.1.101, with verbose reporting.

```
RedBoot> ping -h 192.168.1.101 -v -n 4
Network PING - from 192.168.1.31 to 192.168.1.101
 seq: 1, time: 1 (ticks)
 seq: 2, time: 1 (ticks)
 seq: 3, time: 1 (ticks)
 seq: 4, time: 1 (ticks)
PING - received 10 of 10 expected
```

```
Test connectivity to a non-existent host (192.168.1.109).
RedBoot> ping -h 192.168.1.109 -v -n 4
PING: Cannot reach server ’192.168.1.109’ (192.168.1.109)
```

reset

Name

`reset` — Reset the device

Synopsis

reset

Arguments

None

Description

The **reset** command causes the target platform to be reset. Where possible (hardware support permitting), this will be equivalent to a power-on reset condition.

Examples

Reset the platform.

```
RedBoot> reset
... Resetting.+... Waiting for network card: .
Socket Communications, Inc: Low Power Ethernet CF Revision C 5V/3.3V 08/27/98
Ethernet eth0: MAC address 00:c0:1b:00:ba:28
IP: 192.168.1.29, Default server: 192.168.1.101

RedBoot(tm) bootstrap and debug environment [ROM]
Non-certified release, version UNKNOWN - built 10:41:41, May 14 2002

Platform: Compaq iPAQ Pocket PC (StrongARM 1110)
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x01fc0000, 0x00014748-0x01f71000 available
FLASH: 0x50000000 - 0x51000000, 64 blocks of 0x00040000 bytes each.
RedBoot>
```

reset

version

Name

`version` — Display RedBoot version information

Synopsis

version

Arguments

None

Description

The **version** command simply displays version information about RedBoot.

Examples

Display RedBoot's version.

```
RedBoot> version
RedBoot(tm) debug environment - built 09:12:03, Feb 12 2001
Platform: XYZ (PowerPC 860)
Copyright (C) 2000, 2001, Red Hat, Inc.
RAM: 0x00000000-0x00400000
```

Flash Image System (FIS)

If the platform has flash memory, RedBoot can use this for image storage. Executable images, as well as data, can be stored in flash in a simple file store. The **fis** command (fis is short for Flash Image System) is used to manipulate and maintain flash images.

fis init

Name

`fis init` — Initialize Flash Image System (FIS)

Synopsis

fis init [*-f*]

Arguments

Name	Type	Description	Default
-f		All blocks of flash memory (except for the boot blocks) will be erased as part of the initialization procedure.	

Description

This command is used to initialize the Flash Image System (FIS). It should normally only be executed once, when RedBoot is first installed on the hardware. If the reserved images or their sizes in the FIS change, due to a different configuration of RedBoot being used, it may be necessary to issue the command again though.

> **Note:** Subsequent executions will cause loss of previously stored information in the FIS.

Examples

Initialize the FIS directory.

```
RedBoot> fis init
About to initialize [format] flash image system - continue (y/n)? y
```

```
*** Initialize FLASH Image System
    Warning: device contents not erased, some blocks may not be usable
... Erase from 0x00070000-0x00080000: .
... Program from 0x0606f000-0x0607f000 at 0x00070000: .
```

Initialize the FIS directory and all of flash memory, except for first blocks of the flash where the boot monitor resides.

```
RedBoot> fis init -f
About to initialize [format] flash image system - continue (y/n)? y
*** Initialize FLASH Image System
... Erase from 0x00020000-0x00070000: .....
... Erase from 0x00080000-0x00080000:
... Erase from 0x00070000-0x00080000: .
... Program from 0x0606f000-0x0607f000 at 0x00070000: .
```

fis list

Name

`fis list` — List Flash Image System directory

Synopsis

fis list [*-f*]

Arguments

Name	Type	Description	Default
-c		Show image checksum instead of memory address (column `Mem addr` is replaced by `Checksum`).	
-d		Show image data length instead of amount of flash occupied by image (column `Length` is replaced by `Datalen`).	

Description

This command lists the images currently available in the FIS. Certain images used by RedBoot have fixed names and have reserved slots in the FIS (these can be seen after using the **fis init** command). Other images can be manipulated by the user.

> **Note:** The images are listed in the order they appear in the FIS directory, not by name or creation time.

Examples

List the FIS directory.

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
RedBoot           0x00000000  0x00000000  0x00020000  0x00000000
RedBoot config    0x0007F000  0x0007F000  0x00001000  0x00000000
FIS directory     0x00070000  0x00070000  0x0000F000  0x00000000
```

List the FIS directory, with image checksums substituted for memory addresses.

```
RedBoot> fis list -c
Name              FLASH addr  Checksum    Length      Entry point
RedBoot           0x00000000  0x00000000  0x00020000  0x00000000
RedBoot config    0x0007F000  0x00000000  0x00001000  0x00000000
FIS directory     0x00070000  0x00000000  0x0000F000  0x00000000
```

List the FIS directory with image data lengths substituted for flash block reservation lengths.

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Datalen     Entry point
RedBoot           0x00000000  0x00000000  0x00000000  0x00000000
RedBoot config    0x0007F000  0x0007F000  0x00000000  0x00000000
FIS directory     0x00070000  0x00070000  0x00000000  0x00000000
```

fis free

Name

`fis free` — Free flash image

Synopsis

fis free

Description

This command shows which areas of the flash memory are currently not in use. When a block contains non-erased contents it is considered in use. Since it is possible to force an image to be loaded at a particular flash location, this command can be used to check whether that location is in use by any other image.

> **Note:** There is currently no cross-checking between actual flash contents and the FIS directory, which mans that there could be a segment of flash which is not erased that does not correspond to a named image, or vice-versa.

Examples

Show free flash areas.

```
RedBoot> fis free
        0xA0040000 .. 0xA07C0000
        0xA0840000 .. 0xA0FC0000
```

fis create

Name

`fis create` — Create flash image

Synopsis

fis create {-b *data address*} {-l *length*} [-f *flash address*] [-e *entry*] [-r *relocation address*] [-s *data length*] [-n] [*name*]

Arguments

Name	Type	Description	Default
-b	Number	Address of data to be written to the flash.	Address of last loaded file. If not set in a load operation, it must be specified.
-l	Number	Length of flash area to occopy. If specified, and the named image already exists, the length must match the value in the FIS directory.	Length of area reserved in FIS directory if the image already exists, or the length of the last loaded file. If neither are set, it must be specified.
-f	Number	Address of flash area to occopy.	The address of an area reserved in the FIS directory for extant images. Otherwise the first free block which is large enough will be used.
-e	Number	Entry address for an executable image, used by the **fis load** command.	The entry address of last loaded file.
-r	Number	Address where the image should be relocated to by the **fis load** command. This is only relevant for images that will be loaded with the **fis load** command.	The load address of the last loaded file.

Name	Type	Description	Default
-s	Number	Actual length of data written to image. This is used to control the range over which the checksum is made.	It defaults to the length of the last loaded file.
-n		When set, no image data will be written to the flash. Only the FIS directory will be updated.	
name	String	Name of flash image.	

Description

This command creates an image in the FIS directory. The data for the image must exist in RAM memory before the copy. Typically, you would use the RedBoot **load** command to load file into RAM and then the **fis create** command to write it to a flash image.

Examples

Trying to create an extant image, will require the action to be verified.

```
RedBoot> fis create RedBoot -f 0xa0000000 -b 0x8c400000 -l 0x20000
An image named ‘RedBoot’ exists - continue (y/n)? n
```

Create a new test image, let the command find a suitable place.

```
RedBoot> fis create junk -b 0x8c400000 -l 0x20000
... Erase from 0xa0040000-0xa0060000: .
... Program from 0x8c400000-0x8c420000 at 0xa0040000: .
... Erase from 0xa0fe0000-0xa1000000: .
... Program from 0x8c7d0000-0x8c7f0000 at 0xa0fe0000: .
```

Update the RedBoot[RAM] image.

```
RedBoot> load redboot_RAM.img
Entry point: 0x060213c0, address range: 0x06020000-0x06036cc0
RedBoot> fis create RedBoot[RAM]
No memory address set.
An image named ’RedBoot[RAM]’ exists - continue (y/n)? y
* CAUTION * about to program ’RedBoot[RAM]’
            at 0x00020000..0x00036cbf from 0x06020000 - continue (y/n)? y
... Erase from 0x00020000-0x00040000: ..
... Program from 0x06020000-0x06036cc0 at 0x00020000: ..
... Erase from 0x00070000-0x00080000: .
... Program from 0x0606f000-0x0607f000 at 0x00070000: .
```

fis load

Name

`fis load` — Load flash image

Synopsis

fis load [-b *`load address`*] [-c] [-d] [*`name`*]

Arguments

Name	Type	Description	Default
-b	Number	Address the image should be loaded to. Executable images normally load at the location to which the file was linked. This option allows the image to be loaded to a specific memory location, possibly overriding any assumed location.	If not specified, the address associated with the image in the FIS directory will be used.
-c		Compute and print the checksum of the image data after it has been loaded into memory.	
-d		Decompress gzipped image while copying it from flash to RAM.	
`name`	String	The name of the file, as shown in the FIS directory.	

Description

This command is used to transfer an image from flash memory to RAM.

Once the image has been loaded, it may be executed using the **go** command.

Examples

Load and run RedBoot[RAM] image.

```
RedBoot> fis load RedBoot[RAM]
RedBoot> go
```

fis delete

Name

`fis delete` — Delete flash image

Synopsis

fis delete {*name*}

Arguments

Name	Type	Description	Default
name	Number	Name of image that should be deleted.	

Description

This command removes an image from the FIS. The flash memory will be erased as part of the execution of this command, as well as removal of the name from the FIS directory.

> **Note:** Certain images are reserved by RedBoot and cannot be deleted. RedBoot will issue a warning if this is attempted.

Examples

```
RedBoot> fis list
Name              flash addr   Mem addr    Length    Entry point
RedBoot          0xA0000000   0xA0000000  0x020000  0x80000000
RedBoot config   0xA0FC0000   0xA0FC0000  0x020000  0x00000000
FIS directory    0xA0FE0000   0xA0FE0000  0x020000  0x00000000
junk             0xA0040000   0x8C400000  0x020000  0x80000000
RedBoot> fis delete junk
Delete image ‘junk’ - continue (y/n)? y
... Erase from 0xa0040000-0xa0060000: .
... Erase from 0xa0fe0000-0xa1000000: .
... Program from 0x8c7d0000-0x8c7f0000 at 0xa0fe0000: .
```

fis lock

Name

`fis lock` — Lock flash area

Synopsis

fis lock {-f *`flash_address`*} {-l *`length`*}

Arguments

Name	Type	Description	Default
`flash_address`	Number	Address of area to be locked.	
`length`	Number	Length of area to be locked.	

Description

This command is used to write-protect (lock) a portion of flash memory, to prevent accidental overwriting of images. In order to make make any modifications to the flash, a matching **fis unlock** command must be issued. This command is optional and will only be provided on hardware which can support write-protection of the flash space.

> **Note:** Depending on the system, attempting to write to write-protected flash may generate errors or warnings, or be benignly quiet.

Examples

Lock an area of the flash

```
RedBoot> fis lock -f 0xa0040000 -l 0x20000
... Lock from 0xa0040000-0xa0060000: .
```

fis unlock

Name

`fis unlock` — Unlock flash area

Synopsis

fis unlock {-f *`flash_address`*} {-l *`length`*}

Arguments

Name	Type	Description	Default
`flash_address`	Number	Address of area to be unlocked.	
`length`	Number	Length of area to be unlocked.	

Description

This command is used to unlock a portion of flash memory forcibly, allowing it to be updated. It must be issued for regions which have been locked before the FIS can reuse those portions of flash.

> **Note:** Some flash devices power up in locked state and always need to be manually unlocked before they can be written to.

Examples

Unlock an area of the flash

```
RedBoot> fis unlock -f 0xa0040000 -l 0x20000
... Unlock from 0xa0040000-0xa0060000: .
```

fis erase

Name

`fis erase` — Erase flash area

Synopsis

fis erase {-f *`flash_address`*} {-l *`length`*}

Arguments

Name	Type	Description	Default
`flash_address`	Number	Address of area to be erased.	
`length`	Number	Length of area to be erased.	

Description

This command is used to erase a portion of flash memory forcibly. There is no cross-checking to ensure that the area being erased does not correspond to an existing image.

Examples

Erase an area of the flash

```
RedBoot> fis erase -f 0xa0040000 -l 0x20000
... Erase from 0xa0040000-0xa0060000: .
```

fis write

Name

`fis write` — Write flash area

Synopsis

fis write {-b *`mem_address`*} {-l *`length`*} {-f *`flash_address`*}

Arguments

Name	Type	Description	Default
`mem_address`	Number	Address of data to be written to flash.	
`length`	Number	Length of data to be writtem.	
`flash_address`	Number	Address of flash to write to.	

Description

This command is used to write data from memory to flash. There is no cross-checking to ensure that the area being written to does not correspond to an existing image.

Examples

Write an area of data to the flash

```
RedBoot> fis write -b 0x0606f000 -l 0x1000 -f 0x00020000
* CAUTION * about to program FLASH
            at 0x00020000..0x0002ffff from 0x0606f000 - continue (y/n)? y
... Erase from 0x00020000-0x00030000: .
... Program from 0x0606f000-0x0607f000 at 0x00020000: .
```

Persistent State Flash-based Configuration and Control

RedBoot provides flash management support for storage in the flash memory of multiple executable images and of non-volatile information such as IP addresses and other network information.

RedBoot on platforms that support flash based configuration information will report the following message the first time that RedBoot is booted on the target:

```
flash configuration checksum error or invalid key
```

This error can be ignored if no flash based configuration is desired, or can be silenced by running the **fconfig** command as described below. At this point you may also wish to run the **fis init** command. See other fis commands in the Section called *Flash Image System (FIS)*.

Certain control and configuration information used by RedBoot can be stored in flash.

The details of what information is maintained in flash differ, based on the platform and the configuration. However, the basic operation used to maintain this information is the same. Using the **fconfig -l** command, the information may be displayed and/or changed.

If the optional flag `-i` is specified, then the configuration database will be reset to its default state. This is also needed the first time RedBoot is installed on the target, or when updating to a newer RedBoot with different configuration keys.

If the optional flag `-l` is specified, the configuration data is simply listed. Otherwise, each configuration parameter will be displayed and you are given a chance to change it. The entire value must be typed - typing just carriage return will leave a value unchanged. Boolean values may be entered using the first letter (`t` for true, `f` for false). At any time the editing process may be stopped simply by entering a period (.) on the line. Entering the caret (^) moves the editing back to the previous item. See “RedBoot Editing Commands”, the Section called *RedBoot Editing Commands* in Chapter 1.

If any changes are made in the configuration, then the updated data will be written back to flash after getting acknowledgment from the user.

If the optional flag `-n` is specified (with or without `-l`) then “nicknames” of the entries are used. These are shorter and less descriptive than “full” names. The full name may also be displayed by adding the `-f` flag.

The reason for telling you nicknames is that a quick way to set a single entry is provided, using the format

```
RedBoot> fconfig nickname value
```

If no value is supplied, the command will list and prompt for only that entry. If a value is supplied, then the entry will be set to that value. You will be prompted whether to write the new information into flash if any change was made. For example

```
RedBoot> fconfig -l -n
boot_script: false
bootp: false
bootp_my_ip: 10.16.19.176
bootp_server_ip: 10.16.19.66
dns_ip: 10.16.19.1
gdb_port: 9000
net_debug: false
RedBoot> fconfig bootp_my_ip 10.16.19.177
bootp_my_ip: 10.16.19.176 Setting to 10.16.19.177
Update RedBoot non-volatile configuration - continue (y/n)? y
```

```
... Unlock from 0x507c0000-0x507e0000: .
... Erase from 0x507c0000-0x507e0000: .
... Program from 0x0000a8d0-0x0000acd0 at 0x507c0000: .
... Lock from 0x507c0000-0x507e0000: .
RedBoot>
```

Additionally, nicknames can be used like aliases via the format %{nickname}. This allows the values stored by **`fconfig`** to be used directly by scripts and commands.

Depending on how your terminal program is connected and its capabilities, you might find that you are unable to use line-editing to delete the 'old' value when using the default behaviour of **fconfig *`nickname`*** or just plain **fconfig**, as shown in this example:

```
RedBoot> fco bootp
bootp: false_
```

The user deletes the word "false;" and enters "true" so the display looks like this:

```
RedBoot> fco bootp
bootp: true
Update RedBoot non-volatile configuration - continue (y/n)? y
... Unlock from ...
RedBoot> _
```

To edit when you cannot backspace, use the optional flag `-d` (for "dumb terminal") to provide a simpler interface thus:

```
RedBoot> fco -d bootp
bootp: false ? _
```

and you enter the value in the obvious manner thus:

```
RedBoot> fco -d bootp
bootp: false ? true
Update RedBoot non-volatile configuration - continue (y/n)? y
... Unlock from ...
RedBoot> _
```

One item which is always present in the configuration data is the ability to execute a script at boot time. A sequence of RedBoot commands can be entered which will be executed when the system starts up. Optionally, a time-out period can be provided which allows the user to abort the startup script and proceed with normal command processing from the console.

```
RedBoot> fconfig -l
Run script at boot: false
Use BOOTP for network configuration: false
Local IP address: 192.168.1.29
Default server IP address: 192.168.1.101
DNS server IP address: 192.168.1.1
```

```
GDB connection port: 9000
Network debug at boot time: false
```

The following example sets a boot script and then shows it running.

```
RedBoot> fconfig
Run script at boot: false t
      Boot script:
Enter script, terminate with empty line
>> fi li
    Boot script timeout: 0 10
Use BOOTP for network configuration: false .
Update RedBoot non-volatile configuration - continue (y/n)? y
... Erase from 0xa0fc0000-0xa0fe0000: .
... Program from 0x8c021f60-0x8c022360 at 0xa0fc0000: .
RedBoot>
RedBoot(tm) debug environment - built 08:22:24, Aug 23 2000
Copyright (C) 2000, Red Hat, Inc.


RAM: 0x8c000000-0x8c800000
flash: 0xa0000000 - 0xa1000000, 128 blocks of 0x00020000 bytes ea.
Socket Communications, Inc: Low Power Ethernet CF Revision C \
5V/3.3V 08/27/98 IP: 192.168.1.29, Default server: 192.168.1.101 \
== Executing boot script in 10 seconds - enter ^C to abort
RedBoot> fi li
Name              flash addr   Mem addr    Length    Entry point
RedBoot          0xA0000000   0xA0000000  0x020000  0x80000000
RedBoot config   0xA0FC0000   0xA0FC0000  0x020000  0x00000000
FIS directory    0xA0FE0000   0xA0FE0000  0x020000  0x00000000
RedBoot>
```

NOTE: The bold characters above indicate where something was entered on the console. As you can see, the **fi li** command at the end came from the script, not the console. Once the script is executed, command processing reverts to the console.

NOTE: RedBoot supports the notion of a boot script timeout, i.e. a period of time that RedBoot waits before executing the boot time script. This period is primarily to allow the possibility of canceling the script. Since a timeout value of zero (0) seconds would never allow the script to be aborted or canceled, this value is not allowed. If the timeout value is zero, then RedBoot will abort the script execution immediately.

On many targets, RedBoot may be configured to run from ROM or it may be configured to run from RAM. Other configurations are also possible. All RedBoot configurations will execute the boot script, but in certain cases it may be desirable to limit the execution of certain script commands to one RedBoot configuration or the other. This can be accomplished by prepending `{<startup type>}` to the commands which should be executed only by the RedBoot configured for the specified startup type. The following boot script illustrates this concept by having the ROM based RedBoot load and run the RAM based RedBoot. The RAM based RedBoot will then list flash images.

```
RedBoot> fco
Run script at boot: false t
Boot script:
Enter script, terminate with empty line
>> {ROM}fis load RedBoot[RAM]
>> {ROM}go
>> {RAM}fis li
>>
Boot script timeout (1000ms resolution): 2
Use BOOTP for network configuration: false
 ...
Update RedBoot non-volatile configuration - continue (y/n)? y
... Unlock from 0x007c0000-0x007e0000: .
... Erase from 0x007c0000-0x007e0000: .
... Program from 0xa0015030-0xa0016030 at 0x007df000: .
... Lock from 0x007c0000-0x007e0000: .
RedBoot> reset
... Resetting.
+Ethernet eth0: MAC address 00:80:4d:46:01:05
IP: 192.168.1.153, Default server: 192.168.1.10

RedBoot(tm) bootstrap and debug environment [ROM]
Red Hat certified release, version R1.xx - built 17:37:36, Aug 14 2001

Platform: IQ80310 (XScale)
Copyright (C) 2000, 2001, Red Hat, Inc.

RAM: 0xa0000000-0xa2000000, 0xa001b088-0xa1fdf000 available
FLASH: 0x00000000 - 0x00800000, 64 blocks of 0x00020000 bytes each.
== Executing boot script in 2.000 seconds - enter ^C to abort
RedBoot> fis load RedBoot[RAM]
RedBoot> go
+Ethernet eth0: MAC address 00:80:4d:46:01:05
IP: 192.168.1.153, Default server: 192.168.1.10

RedBoot(tm) bootstrap and debug environment [RAM]
Red Hat certified release, version R1.xx - built 13:03:47, Aug 14 2001

Platform: IQ80310 (XScale)
Copyright (C) 2000, 2001, Red Hat, Inc.

RAM: 0xa0000000-0xa2000000, 0xa0057fe8-0xa1fdf000 available
FLASH: 0x00000000 - 0x00800000, 64 blocks of 0x00020000 bytes each.
== Executing boot script in 2.000 seconds - enter ^C to abort
RedBoot> fis li
Name              FLASH addr  Mem addr    Length      Entry point
RedBoot           0x00000000  0x00000000  0x00040000  0x00002000
RedBoot config    0x007DF000  0x007DF000  0x00001000  0x00000000
FIS directory     0x007E0000  0x007E0000  0x00020000  0x00000000
RedBoot>
```

Executing Programs from RedBoot

Once an image has been loaded into memory, either via the **load** command or the **fis load** command, execution may be transfered to that image.

> **NOTE:** The image is assumed to be a stand-alone entity, as RedBoot gives the entire platform over to it. Typical examples would be an eCos application or a Linux kernel.

go

Name

`go` — Execute a program

Synopsis

go [-w *`timeout`*] [*`start_address`*]

Arguments

Name	Type	Description	Default
-w *`timeout`*	Number	How long to wait before starting execution.	0
`start_address`	Number	Address in memory to begin execution.	Value set by last **load** or **fis load** command.

Description

The **go** command causes RedBoot to give control of the target platform to another program. This program must execute stand alone, e.g. an eCos application or a Linux kernel.

If the -w option is used, RedBoot will print a message and then wait for a period of time before starting the execution. This is most useful in a script, giving the user a chance to abort executing a program and move on in the script.

Examples

Execute a program - *no explicit output from RedBoot*.

```
RedBoot> go 0x40040
```

Execute a program with a timeout.

```
RedBoot> go -w 10
About to start execution at 0x00000000 - abort with ^C within 10 seconds
^C
RedBoot>
```

Note that the starting address was implied (0x00000000 in this example). The user is prompted that execution will commence in 10 seconds. At anytime within that 10 seconds the user may type Ctrl+C on the console and RedBoot will abort execution and return for the next command, either from a script or the console.

exec

Name

`exec` — Execute a Linux kernel

Synopsis

exec [-w *timeout*] [-r *ramdisk_address*] [-s *ramdisk_length*] [-b *load_address* {-l *load_length*}] [-c *kernel_command_line*] [*entry_point*]

Arguments

Name	Type	Description	Default
-w *timeout*	Number	Time to wait before starting execution.	0
-r *ramdisk_address*	Number	Address in memory of "initrd"-style ramdisk - passed to Linux kernel.	*None*
-s *ramdisk_length*	Number	Length of ramdisk image - passed to Linux kernel.	*None*
-b *load_address*	Number	Address in memory of the Linux kernel image.	Value set by **load** or **fis load**
-l *load_length*	Number	Length of Linux kernel image.	*none*
-c *kernel_command_line*	String	Command line to pass to the Linux kernel.	*None*
entry_address	Number	Starting address for Linux kernel execution	Implied by architecture

Description

The **exec** command is used to execute a non-eCos application, typically a Linux kernel. Additional information may be passed to the kernel at startup time. This command is quite special (and unique from the **go** command) in that the program being executed may expect certain environmental setups, for example that the MMU is turned off, etc.

The Linux kernel expects to have been loaded to a particular memory location which is architecture dependent(0xC0008000 in the case of the SA1110). Since this memory is used by RedBoot internally, it is not possible to load the kernel to that location directly. Thus the requirement for the "-b" option which tells the command where the kernel has been loaded. When the **exec** command runs, the image will be relocated to the appropriate location before being started. The "-r" and "-s" options are used to pass information to the kernel about where a statically

loaded ramdisk (initrd) is located.

The "-c" option can be used to pass textual "command line" information to the kernel. If the command line data contains any punctuation (spaces, etc), then it must be quoted using the double-quote character '"'. If the quote character is required, it should be written as '\"'.

Examples

Execute a Linux kernel, passing a command line, which needs relocation. The result from RedBoot is normally quiet, with the target platform being passed over to Linux immediately.

```
RedBoot> exec -b 0x100000 -l 0x80000 -c "noinitrd root=/dev/mtdblock3 console=ttySA0"
```

Execute a Linux kernel, default entry address and no relocation required, with a timeout. The *emphasized lines* are output from the loaded kernel.

```
RedBoot> exec -c "console=ttyS0,38400 ip=dhcp nfsroot=/export/elfs-sh" -w 5
Now booting linux kernel:
Base address 0x8c001000 Entry 0x8c210000
Cmdline : console=ttyS0,38400 ip=dhcp nfsroot=/export/elfs-sh
About to start execution at 0x8x210000 - abort with ^C within 5 seconds
Linux version 2.4.10-pre6 (...) (gcc version 3.1-stdsh-010931) #3 Thu Sep 27 11:04:23 BST 2001
```

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP