PLX-9054 PCI Performance Tests

D. W. Hawkins
dwh@ovro.caltech.edu
Document Revision: 1.3

Sept 27th, 2004

Contents

1 Introduction

The PLX PCI-9054 I/O Accelerator from PLX Technologies, Inc. (`www.plxtech.com`) is a high-performance 32-bit, 33MHz, PCI master/target interface. This document contains performance tests obtained from the COBRA Correlator boards. The objective of these tests was to determine the register settings for the COBRA Correlator System boards that would result in optimal usage of the PCI bus. Throughout this document the PLX PCI-9054 is referred to as the PLX-9054.

The PLX-9054 contains a PCI bus interface and a local bus interface. The PLX-9054 implements the PCI bus protocol and converts PCI transactions into a simpler local bus protocol. The local bus can be configured in several modes. The COBRA boards operate the PLX-9054 in J-mode, i.e., the local bus address and data are multiplexed. The local bus interfaces to the COBRA system controller FPGA. The system controller FPGA acts as a local bus target and local bus master. Local bus master mode allows the on-board DSP to access the PLX-9054 control registers from the local-bus side of the PLX-9054.

The COBRA Correlator system boards process data using onboard FPGAs and a DSP. Processed data is transferred to the host on 100ms or 500ms timescales. Monitor data is transferred on 500ms timescales. Debug access can also occur at any time. The host CPU runs Linux (eg. RedHat 9.0) without any real-time patches. A single Linux host can control up to 17-boards in a single compact PCI chassis, and up to 33-boards when two chassis are linked by PCI-to-PCI bridge boards. This document contains performance tests from single board and multiple board configurations.

Multiple board testing exposed a problem with the Linux kernel. If the CPU is used to access data from the board (PLX-9054 acts as a PCI Target), then the Linux kernel's concept of time is corrupted (probably due to missed or late timer tick interrupts). The CPU initiated reads or writes to the boards complete correctly, i.e., no PCI bus protocol violation is detected by the kernel (which is not to say that a PCI bus analyzer would not detect a problem). Since the COBRA System applications use the host time to check data timestamps from the board, when the hosts time is corrupted, all data from the boards gets erroneously flagged as being in error. Host time corruption can be detected by running an NTP daemon and using `/usr/sbin/ntpq -p` to view the host time relative to the NTP server.

2 PLX-9054 Control Register and Memory Spaces

The PLX-9054 contains several register spaces. From the PCI-side of the device, there is the standard set of PCI configuration space registers, and an additional set of device configuration registers. The device configuration registers appear as 256-bytes of non-prefetchable PCI memory in base address register (BAR) 0, and as 256-bytes of PCI I/O ports in BAR1. The PCI configuration space registers and the device configuration registers are also accessible from the local bus. Several of the PCI configuration space and device configuration registers are loaded at boot time from a serial configuration EEPROM.

Users of the PLX-9054 can define two custom PCI-to-local address spaces and an expansion ROM address space (which is not used on COBRA). The COBRA boards are typically configured to disable the PLX-9054 I/O registers in BAR1 and overlay that region with Local Address Space 0 configured as an 8MB region of prefetchable memory. An alternative configuration considered was to leave BAR1 enabled and configure BAR2 as 8MB of prefetchable memory, and BAR3 as 8MB of non-prefetchable memory, with both 8MB regions decoding to the same backend logic. The argument for the second configuration was that the 8MB prefetchable region could be used for DMA, while the non-prefetchable region could be used for register (memory mapped I/O) accesses. Accesses to the non-prefetchable region will not generate burst transactions to the backend logic, so would result in lower performance if used for block data moves. However, for control register accesses the two PCI memory regions should perform similarly. Appendix A contains both EEPROM configuration files.

3 PLX-9054 Control Registers

The register names in this section follow the designations given in the device data sheet. The byte-addresses of the registers are defined in the COBRA device driver source in the file `plx_registers.h`. This file also contains bit definitions for the most commonly used control register bits.

Sections 4 and 5 of the PLX-9054 data sheet (Version 2.1, January 2000) contain a description of the C- and J-mode operation of the device, and Section 11 contains the register descriptions. The *PCI Configuration Registers* are described in Section 11.2.1, Table 11-2, on p11-2 of the data sheet, with Section 11.3 providing detailed descriptions of the registers. The PCI configuration space registers are also accessible from the local address space starting at address 0h relative to the local-bus base address of PLX-9054 registers. The PLX-9054 specific configuration registers accessible via PCI BAR0 and BAR1 are described in Section 11.2.2 *Local Configuration Registers*, Section 11.2.3 *Runtime Registers*, Section 11.2.4 *DMA Registers*, and Section 11.2.5 *Messaging Queue Registers*. Detailed descriptions of the register bits are given in Sections 11.4 through 11.6. The PLX-9054 configuration registers are also accessible from the local address space starting at a byte offset address of 80h relative to the local-bus base address of PLX-9054 registers.

3.1 PCI Configuration Space Registers

Table 1 shows the PLX-9054 PCI configuration space registers along with their values when a board is booted with a blank EEPROM. The configuration space registers can be read under Linux using `/sbin/lspci`. The output of this command (see Appendix B) shows that the board base address registers are configured as; BAR0 256-bytes of 32-bit non-prefetchable memory, BAR1 256-bytes of I/O space, BAR2 1M-bytes non-prefetchable 32-bit memory, and BAR3 1M-bytes non-prefetchable 32-bit memory. A hex dump (`/sbin/lspci` with the `-xxx` option) of the PCI configuration space gives the reset values shown in Table 1. Several of the registers are configured by the BIOS when the system boots, eg. BAR register contents and the interrupt line routing register. The reset values of the extended PCI configuration space match those given in the data sheet.

The COBRA boards are typically used with an EEPROM configuration that disables the PLX-9054 BAR1 I/O region and overlays it with an 8MB region of prefetchable 32-bit memory. The COBRA boards can also be configured with BAR1 left enabled, and BAR2 and BAR3 with 8MB prefetchable and non-prefetchable regions. Appendix A contains the two EEPROM configuration files, while Appendix B contains register dumps of the PCI configuration space registers for a blank EEPROM and the two EEPROM configuration files.

Table 1: PLX-9054 PCI Configuration Space Registers.

Config. Byte Address	Register	Reset value (blank EEPROM)	Description
			PCI Configuration Space:
00h	PCIIDR	905410B5h	Device ID / Vendor ID
04h	PCISR	02900113h	Status
	PCICR		Command
08h	PCICCR	0680000Ah	Class Code
	PCIREV		Revision ID
0Ch	PCIBISTR	00004008h	Built-in self test
	PCIHTR		Header type
	PCILTR		Bus latency timer
	PCICLSR		Cache line size
10h	PCIBAR0	F3D00000h	BAR0 (PLX-9054 Memory-mapped registers)
14h	PCIBAR1	00002001h	BAR1 (PLX-9054 I/O-mapped registers)
18h	PCIBAR2	F3C00000h	BAR2 (Local Address Space 0: 1MB default)
1Ch	PCIBAR3	F3B00000h	BAR3 (Local Address Space 1: 1MB default)
20h	PCIBAR4	00000000h	BAR4 (Local Address Space 2: unused)
24h	PCIBAR5	00000000h	BAR5 (Local Address Space 3: unused)
28h	PCICIS	00000000h	Cardbus CIS pointer
2Ch	PCISID	905410B5h	Subsystem ID
	PCISVID		Subsystem Vendor ID
30h	PCIERBAR	00000000h	Expansion ROM BAR
34h	CAP_PTR	00000040h	Reserved / Next capabilities pointer
38h		00000000h	Reserved
3Ch	PCIMLR	00000107h	PCI maximum latency
	PCIMGR		PCI minimum grant
	PCIIPR		PCI interrupt pin
	PCIILR		PCI interrupt line routing
			Extended Configuration Space:
40h	PMC	00014801h	Power management capabilities
	PMNEXT		Power management next capability pointer
	PMCAPID		Power management capability ID
44h	PMDATA	00000000h	Power management data
	PMCSR_BSE		Power management bridge support extensions
	PMCSR		Power management control/status
48h	HS_CNTL	00804C06h	Hot swap control
	HS_NEXT		Hot swap next capability pointer
	HS_CSID		Hot swap capability ID
4Ch	PVPDAD	00000003h	Vital product data (VPD) address
	PVPD_NEXT		VPD capability pointer
	PVPD_CSID		VPD capability ID
50h	PVPDATA	00000000h	VPD data

3.2 PCI BAR0/1 Registers

The PLX-9054 is configured via the standard PCI Configuration space registers, and a set of device specific registers located in BAR0 and BAR1. The BAR1/0 registers are configured at boot time by the boot EEPROM. The BAR0 registers are typically used by the device driver. eg. when dealing with interrupts. Access to these registers is rarely required from user-space. Access is required from user space when a blank EEPROM needs to be programmed (the EEPROM write protection register needs to be cleared), and when a user wants to generate a local bus reset on a board. The COBRA device driver provides user-space access to the PLX-9054 BAR0 registers.

Table 2 shows the BAR0 registers as viewed from the PCI bus from a board booted with a blank EEPROM. Appendix C shows the results of reads from a PLX-9054 booted from a blank EEPROM, an EEPROM that creates an 8M region, and an EEPROM that creates two 8MB regions.

Table 2: PLX-9054 BAR0 Control Registers.

PCI Byte Offset	Register	Reset value	Description
			Local Configuration Registers:
00h	LAS0RR	FFF00000h	Range for PCI-to-Local Address Space 0
04h	LAS0BA	00000000h	Local Base Address for PCI-to-Local Address 0
08h	MARBR	00200000h	Mode / DMA Arbitration
0Ch	PROT_AREA	00300500h	Serial EEPROM write-protection address boundary
	LMISC		Local bus miscellaneous control
	BIGEND		Big/little endian descriptor
10h	EROMRR	FFFF0000h	Range for PCI-to-Local Expansion ROM
14h	EROMBA	00000000h	Local Base Address for PCI-to-Local Exp ROM
18h	LBDR0	40430043h	Local Bus Region Descriptors (Space 0 and ROM)
1Ch	DMRR	00000000h	Range for PCI Master-to-PCI
20h	DMLBAM	00000000h	Local Base Address for PCI Master-to-PCI Memory
24h	DMLBAI	00000000h	Local Base Address for PCI Master-to-PCI I/O
28h	DMPBAM	00000000h	PCI Base Address for PCI Master-to-PCI
2Ch	DMCFGA	00000000h	PCI Config. Address for PCI Master-to-PCI Config
			Messaging Queue Registers:
30h	OPQIS	00000000h	Outbound Post Queue Interrupt Status
34h	OPQIM	00000008h	Outbound Post Queue Interrupt Mask
38h-3Ch		00000000h	(unused)
			Runtime Registers:
40h	MBOX0	00000000h	Mailbox Register 0
44h	MBOX1	00000000h	Mailbox Register 1
48h	MBOX2	00000000h	Mailbox Register 2
4Ch	MBOX3	00000000h	Mailbox Register 3
50h	MBOX4	00000000h	Mailbox Register 4
54h	MBOX5	00000000h	Mailbox Register 5
58h	MBOX6	00000000h	Mailbox Register 6
5Ch	MBOX7	00000000h	Mailbox Register 7
60h	P2LDBELL	00000000h	PCI-to-Local Doorbell Register
64h	L2PDBELL	00000000h	Local-to-PCI Doorbell Register
68h	INTCSR	0F010180h	Interrupt Control/Status
6Ch	CNTRL	080F767Eh	EEPROM Control / PCI Cmd / User I/O / Init
70h	PCIHIDR	905410B5h	Device ID / Vendor ID
74h	PCIHREV	0000000Ah	Reserved / Revision ID
78h-7Ch		00000000h	(unused)

Table 2: PLX-9054 BAR0 Control Registers.

			DMA Registers:
80h	DMAMODE0	00000043h	DMA Channel 0 Mode
84h	DMAPADR0	00000000h	DMA Channel 0 PCI Address
88h	DMALADR0	00000000h	DMA Channel 0 Local Address
8Ch	DMASIZ0	00000000h	DMA Channel 0 Transfer Byte Count
90h	DMADPR0	00000000h	DMA Channel 0 Descriptor Pointer
94h	DMAMODE1	00000043h	DMA Channel 1 Mode
98h	DMAPADR1	00000000h	DMA Channel 1 PCI Address
9Ch	DMALADR1	00000000h	DMA Channel 1 Local Address
A0h	DMASIZ1	00000000h	DMA Channel 1 Transfer Byte Count
A4h	DMADPR1	00000000h	DMA Channel 1 Descriptor Pointer
A8h	DMACSR1	00001010h	DMA1 Cmd/Status
	DMACSR0		DMA0 Cmd/Status
ACh	DMAARB	00200000h	Mode/DMA Arbitration
B0h	DMATHR	00000000h	DMA Threshold
B4h	DMADAC0	00000000h	DMA0 PCI DAC
B8h	DMADAC1	00000000h	DMA1 PCI DAC
BCh		00000000h	(unused)
			Messaging Queue Registers:
C0h	MQCR	00000002h	Messaging Unit Configuration
C4h	QBAR	00000000h	Queue Base Address
C8h	IFHPR	00000000h	Inbound Free Head Pointer
CCh	IFTPR	00000000h	Inbound Free Tail Pointer
D0h	IPHPR	00000000h	Inbound Post Head Pointer
D4h	IPTPR	00000000h	Inbound Post Tail Pointer
D8h	OFHPR	00000000h	Outbound Free Head Pointer
DCh	OFTPR	00000000h	Outbound Free Tail Pointer
E0h	OPHPR	00000000h	Outbound Post Head Pointer
E4h	OPTPR	00000000h	Outbound Post Tail Pointer
E8h	QSR	00000050h	Queue Status/Control
ECh		00000000h	(unused)
			Local Configuration Registers: (Space 1)
F0h	LAS1RR	FFF00000h	Range for PCI-to-Local Address Space 1
F4h	LAS1BA	00000000h	Local Base Address for PCI-to-Local Address 1
F8h	LBDR1	00000043h	Local Bus Region Descriptor (Space 1)
FCh	DMDAC	00000000h	PCI Base Dual Address Cycle for PCI Master

3.3 PCI Local Bus Registers

The PCI configuration space registers and the PCI BAR0/1 registers are also accessible from the local bus. The PCI Configuration Space Registers occur first at addresses 0h to 3Ch, followed by a block of unused addresses from 40h to 7Ch, the BAR0/1 registers start at a byte offset of 80h and continue until 17Ch, and then the PCI Configuration Space Extension Registers occur at addresses 180h to 190h. The header `plx_registers.h` defines the register addresses. The layout of each of the three regions is identical to the layout from the PCI bus side, however, they each have a different starting address. Table 3 shows the local bus register map.

Access to the PLX-9054 registers from the local bus enables the on-board DSP to access the PLX-9054 registers. This access is very useful for debugging boards and learning the operation of the PLX-9054 registers. Access to these registers by the DSP enables the DSP to perform Direct Master transactions between itself and another board. Since the DSP accesses the PLX-9054 using 32-bit word aligned addresses, Table 3 also shows register addresses in terms of 32-bits words. Figure 1 shows a dump of the PLX-9054 local-bus registers using the on-board DSP.

The register values in Table 3 were loaded to reflect those from the DSP dump in Figure 1. The PCI configuration register settings are different relative to the previous table values due to a different hardware configuration. The other PLX-9054 register settings are as expected for booting with a blank EEPROM.

Table 3: PLX-9054 Local-Bus Control Registers.

Offset address		Reset value	Description
Byte	Dword		
			PCI Configuration Space:
000h	00h	905410B5h	Device ID / Vendor ID
004h	01h	02900017h	Status / Command
008h	02h	0680000Ah	Class Code / Revision ID
00Ch	03h	00004008h	BIST / Header Type / Lat. Timer / Cache Line Size
010h	04h	D9EFF800h	BAR0 (PLX-9054 Memory-mapped registers)
014h	05h	0000D801h	BAR1 (PLX-9054 I/O-mapped registers)
018h	06h	D9C00000h	BAR2 (Local Address Space 0: 1MB default)
01Ch	07h	D9D00000h	BAR3 (Local Address Space 1: 1MB default)
020h	08h	00000000h	BAR4 (Local Address Space 2: unused)
024h	09h	00000000h	BAR5 (Local Address Space 3: unused)
028h	0Ah	00000000h	Cardbus CIS pointer
02Ch	0Bh	905410B5h	Subsystem ID / Subsystem Vendor ID
030h	0Ch	00000000h	Expansion ROM BAR
034h	0Dh	00000040h	Reserved / Next capabilities pointer
038h	0Eh	00000000h	Reserved
03Ch	0Fh	00000109h	Max Lat / Min Gnt / Intr Pin / Intr Line
040h-07Ch	10h-1Ch	00000000h	(unused)
			Local Configuration Registers:
080h	20h	FFF00000h	Range for PCI-to-Local Address Space 0
084h	21h	00000000h	Local Base Address for PCI-to-Local Address 0
088h	22h	00200000h	Mode / DMA Arbitration
08Ch	23h	00300500h	Reserved / Serial EEPROM Prot. / Misc / BIGEND
090h	24h	FFFF0000h	Range for PCI-to-Local Expansion ROM
094h	25h	00000000h	Local Base Address for PCI-to-Local Exp ROM
098h	26h	40430043h	Local Bus Region Descriptors (Space 0 and ROM)
09Ch	27h	00000000h	Range for PCI Master-to-PCI
0A0h	28h	00000000h	Local Base Address for PCI Master-to-PCI Memory
0A4h	29h	00000000h	Local Base Address for PCI Master-to-PCI I/O
0A8h	2Ah	00000000h	PCI Base Address for PCI Master-to-PCI
0ACh	2Bh	00000000h	PCI Config. Address for PCI Master-to-PCI Config
			Messaging Queue Registers:
0B0h	2Ch	00000000h	Outbound Post Queue Interrupt Status
0B4h	2Dh	00000008h	Outbound Post Queue Interrupt Mask
0B8h-0BCh	2Eh-2Fh	00000000h	(unused)

Table 3: PLX-9054 Local-Bus Control Registers.

			Runtime Registers:
0C0h	30h	00000000h	Mailbox Register 0
0C4h	31h	00000000h	Mailbox Register 1
0C8h	32h	00000000h	Mailbox Register 2
0CCh	33h	00000000h	Mailbox Register 3
0D0h	34h	00000000h	Mailbox Register 4
0D4h	35h	00000000h	Mailbox Register 5
0D8h	36h	00000000h	Mailbox Register 6
0DCh	37h	00000000h	Mailbox Register 7
0E0h	38h	00000000h	PCI-to-Local Doorbell Register
0E4h	39h	00000000h	Local-to-PCI Doorbell Register
0E8h	3Ah	0F010180h	Interrupt Control/Status
0ECh	3Bh	080F767Eh	EEPROM Control / PCI Cmd / User I/O / Init
0F0h	3Ch	905410B5h	Device ID / Vendor ID
0F4h	3Dh	0000000Ah	Reserved / Revision ID
0F8h-0FCh	3Eh-3Fh	00000000h	(unused)
			DMA Registers:
100h	40h	00000043h	DMA Channel 0 Mode
104h	41h	00000000h	DMA Channel 0 PCI Address
108h	42h	00000000h	DMA Channel 0 Local Address
10Ch	43h	00000000h	DMA Channel 0 Transfer Byte Count
110h	44h	00000000h	DMA Channel 0 Descriptor Pointer
114h	45h	00000043h	DMA Channel 1 Mode
118h	46h	00000000h	DMA Channel 1 PCI Address
11Ch	47h	00000000h	DMA Channel 1 Local Address
120h	48h	00000000h	DMA Channel 1 Transfer Byte Count
124h	49h	00000000h	DMA Channel 1 Descriptor Pointer
128h	4Ah	00001010h	Reserved / DMA1 Cmd/Status / DMA0 Cmd/Status
12Ch	4Bh	00200000h	Mode/DMA Arbitration
130h	4Ch	00000000h	DMA Threshold
134h	4Dh	00000000h	DMA0 PCI DAC
138h	4Eh	00000000h	DMA1 PCI DAC
13Ch	4Fh	00000000h	(unused)
			Messaging Queue Registers:
140h	50h	00000002h	Messaging Unit Configuration
144h	51h	00000000h	Queue Base Address
148h	52h	00000000h	Inbound Free Head Pointer
14Ch	53h	00000000h	Inbound Free Tail Pointer
150h	54h	00000000h	Inbound Post Head Pointer
154h	55h	00000000h	Inbound Post Tail Pointer
158h	56h	00000000h	Outbound Free Head Pointer
15Ch	57h	00000000h	Outbound Free Tail Pointer
160h	58h	00000000h	Outbound Post Head Pointer
164h	59h	00000000h	Outbound Post Tail Pointer
168h	5Ah	00000050h	Queue Status/Control
16Ch	5Bh	00000000h	(unused)

Table 3: PLX-9054 Local-Bus Control Registers.

			Local Configuration Registers: (Space 1)
170h	5Ch	FFF00000h	Range for PCI-to-Local Address Space 1
174h	5Dh	00000000h	Local Base Address for PCI-to-Local Address 1
178h	5Eh	00000043h	Local Bus Region Descriptor (Space 1)
17Ch	5Fh	00000000h	PCI Base Dual Address Cycle for PCI Master
			PCI Configuration Space Extension Registers:
180h	60h	00014801h	Pwr Mgmt / Next Cap. Pointer / Cap. ID
184h	61h	00000000h	Power management registers
188h	62h	00804C06h	Hot Swap Control/Status / Next Cap. Pointer / Cap. ID
18Ch	63h	00000003h	Flag / VPD Address / Next Cap. Pointer / Cap. ID
190h	64h	00000000h	VPD Data
194h-1FCh	65h-7Fh	00000000h	(unused)

```
C31> D 4E1800 80

Address:         Contents:
004E1800:        905410B5        02900017        0680000A        00004008
004E1804:        D9EFF800        0000D801        D9C00000        D9D00000
004E1808:        00000000        00000000        00000000        905410B5
004E180C:        00000000        00000040        00000000        00000109
004E1810:        00000000        00000000        00000000        00000000
004E1814:        00000000        00000000        00000000        00000000
004E1818:        00000000        00000000        00000000        00000000
004E181C:        00000000        00000000        00000000        00000000
004E1820:        FFF00000        00000000        00200000        00300500
004E1824:        FFFF0000        00000000        40430043        00000000
004E1828:        00000000        00000000        00000000        00000000
004E182C:        00000000        00000008        00000000        00000000
004E1830:        00000000        00000000        00000000        00000000
004E1834:        00000000        00000000        00000000        00000000
004E1838:        00000000        00000000        0F010180        080F767E
004E183C:        905410B5        0000000A        00000000        00000000
004E1840:        00000043        00000000        00000000        00000000
004E1844:        00000000        00000043        00000000        00000000
004E1848:        00000000        00000000        00001010        00200000
004E184C:        00000000        00000000        00000000        00000000
004E1850:        00000002        00000000        00000000        00000000
004E1854:        00000000        00000000        00000000        00000000
004E1858:        00000000        00000000        00000050        00000000
004E185C:        FFF00000        00000000        00000043        00000000
004E1860:        00014801        00000000        00804C06        00000003
004E1864:        00000000        00000000        00000000        00000000
004E1868:        00000000        00000000        00000000        00000000
004E186C:        00000000        00000000        00000000        00000000
004E1870:        00000000        00000000        00000000        00000000
004E1874:        00000000        00000000        00000000        00000000
004E1878:        00000000        00000000        00000000        00000000
004E187C:        00000000        00000000        00000000        00000000
C31>
```

Figure 1: DSP RS-232 monitor interface readout of the PLX-9054 local-bus control registers (base address 4E1800h). Viewing more of the DSP 1K decode area shows the PLX-9054 registers repeating every 80h (since there are 128 registers). Note that these are the register values as read from a board booted in the cPCI chassis. The PCI BIOS modifies the PCI configuration space, so the first 16 registers will appear different on other boards (or on a board booted on the benchtop).

4 Serial EEPROM

The PLX-9054 is initialized at power-up from a serial boot EEPROM. Section 4.4, p4-7, of the PLX-9054 data sheet describes the operation of Serial EEPROM. The serial EEPROM allows the loading of a user-defined PCI device identification (we leave it as PLX's identification), and loading of the PLX-9054 configuration registers. The EEPROM can be programmed using registers in the PCI configuration space or by using registers in the device registers space to generate I2C control signals to the EEPROM directly. In either case a driver is required to provide access to the PLX-9054 registers, since the `LMISC` register enables writing to the EEPROM. The COBRA device driver provides access to these registers if a blank EEPROM is found on a board. The driver implements `ioctl()` calls to program the EEPROM using the vital product data (VPD) registers. Since the EEPROM can disable access to the VPD registers, it is important that the EEPROM be configured for 'Long serial EEPROM load' and that the EEPROM configuration leave VPD enabled. Table 4 contains the EEPROM memory map (p4-9 and p4-10 of the data sheet contain tables with equivalent information).

Appendix A contains the EEPROM programming files required to give a single 8M prefetchable region and a dual 8MB prefetchable/non-prefetchable memory regions.

Table 4: PLX-9054 Long and Extra Long Serial EEPROM Load Registers.

EEPROM Byte Address	Register Bits Affected	Example Load value	Description
			PCI Configuration Space:
00h	PCIIDR	905410B5h	Device ID/Vendor ID
04h	PCICCR	0680000Ah	Class code
	PCIREV		Revision ID
08h	PCIMLR	00000100h	PCI maximum latency
	PCIMGR		PCI minimum grant
	PCIIPR		PCI interrupt pin
	PCIILR		PCI interrupt line routing
			Runtime Registers:
0Ch	MBOX0	00000000h	Mailbox Register 0
10h	MBOX1	00000000h	Mailbox Register 1
			Local Configuration Registers:
14h	LAS0RR	FF800008h	Range for PCI-to-Local Address Space 0
18h	LAS0BA	00000001h	Local Base Address for PCI-to-Local Address 0
1Ch	MARBR	10200000h	Mode / DMA Arbitration
20h	PROT_AREA	00300600h	Serial EEPROM write-protection address boundary
	LMISC		Local miscellaneous control
	BIGEND		Big/little endian descriptor
24h	EROMRR	00000000h	Range for PCI-to-Local Expansion ROM
28h	EROMBA	00000000h	Local Base Address for PCI-to-Local Exp ROM
2Ch	LBRD0	4B4300C3h	Local Bus Region Descriptors (Space 0 and ROM)
30h	DMRR	FFFF0000h	Range for PCI Master-to-PCI
34h	DMLBAM	01390000h	Local Base Address for PCI Master-to-PCI Memory
38h	DMLBAI	013A0000h	Local Base Address for PCI Master-to-PCI I/O
3Ch	DMPBAM	00000000h	PCI Base Address for PCI Master-to-PCI
40h	DMCFGA	00000000h	PCI Config. Address for PCI Master-to-PCI Config
			PCI Configuration Space:
44h	PCISID	905410B5h	Subsystem ID
	PCISVID		Subsystem vendor ID
			Local Configuration Registers: (Space 1)
48h	LAS1RR	00000000h	Range for PCI-to-Local Address Space 1
4Ch	LAS1BA	00000000h	Local Base Address for PCI-to-Local Address 1
50h	LBRD1	00000000h	Local Bus Region Descriptor (Space 1)
			PCI Configuration Space Extension Registers:
54h	HS_NEXT	00004C06h	Next Capabilities Pointer
	HS_CNTL		Hot Swap Control/Status

5 Register Settings

The PLX-9054 contains a number of registers that affect the performance of the transfers on the PCI bus. The follow list comments on the settings of those registers. Tests in the next section compare the performance obtained with different bit settings.

1. LAS0RR: Local address space 0 range register.
   - This register sets up the PCI BAR register settings for the custom logic on the PCI board (PCI BAR2 or PCI BAR 1 depending on the settings in the LMISC register).
   - Best performance should be obtained for *prefetchable memory*.
   - The COBRA boards are configured as 8MB of prefetchable memory.
   - LAS0RR = FF800008h.
2. MARBR: Mode/DMA Arbitration.
   - This register sets up PCI read/write mode.
   - MARBR[21] = Local bus PCI Target release bus mode. This bit should normally be left at 1. Tests showed that if this bit is set to 0, then the PLX-9054 doesn't deassert HOLD, so retains ownership of the PLX local bus. This means that the DSP can not access the PLX registers, and hangs during reads to this area (since it arbitrates, but never receives the bus). The DSP only ever accesses the PLX bus during debugging, but its easier to just leave this bit at 1.
   - MARBR[24] = PCI r2.1 features enable (was Target delayed read mode bit). Default is 0, test 1. PCI r2.1 features are; 2^{15} clock timeout on retries (approximately 1ms at 33MHz), 16- and 8- clock latency rules, and enables the options available on bits 25 and 26 (which affect what to do with a write while a read is pending).
   - MARBR[28] = Read ahead mode. Default is 0, test 1.
   - MARBR = 00200000h (default).
   - Test; 01200000h (PCI r2.1 enabled), 10200000h (read ahead mode), 11200000h (both).
3. LBRD0: Local address space 0 bus region descriptor.
   - This register sets up the operation of the PLX-9054 local bus.
   - LBRD0[7] BTERM# input enable and LBRD0[24] burst enable should both be set to 1 to enable continuous bursting.
   - LBRD0[8] prefetch disable should be set to 0 to enable prefetching.
   - LBRD0[27] PCI Target write mode determines whether the PLX-9054 releases (0) or keeps (1) the PCI bus during writes when the write FIFO fills. Setting this bit to 1 improves write performance.
   - LBRD0 = 42430043h (default with long EEPROM load) (single r/w access).
   - Test; 434300C3h (continuous burst mode), 4B4300C3h (burst mode with PCI Target write mode bit set).
4. INTCSR: Interrupt control and status register.
   - This register sets up PCI and local interrupts.
   - When using DMA, the DMA controller interrupt needs to be routed to PCI. The DMAMODE0 register is used to enable the DMA done interrupt and route it to PCI.

- INTCSR[8] = 1 enables PCI interrupts.
- INTCSR[11] = 1 enables the local interrupt as an input (needed for our on-board DSP to send interrupts to the host CPU).
- INTCSR[16] = 0 disables the local interrupt as an output (we don't use any of the PLX interrupts at the DSP yet).
- INTCSR[18] = 1 Enable the DMA0 interrupt (this needs to be enabled as well as the DMAMODE0 register).
- INTCSR[15] (LINT active) and INTCSR[21] (DMA0 active) provide status on which interrupt occurred - the device driver uses this to determine what to do.
- INTCSR = 0F040900h.

5. CNTRL: DMA and PCI Master PCI read/write commands.

   - The DMA controller defaults to performing reads using the PCI command for PCI Memory Read Line (Eh) and writes using PCI Memory Write (7h). Using other command codes may result in more efficient transfers (however, the transfers may have to be restricted to multiples of cacheline sizes).

6. DMAMODE0: DMA0 Mode register

   - This register sets up the local bus operation during DMA accesses.
   - DMAMODE0[7] BTERM# input enabled and DMAMODE0[8] burst enable need to be set for continuous burst mode.
   - DMAMODE[10] done interrupt enable, and DMAMODE0[17] interrupt routing select local(0)/PCI(1) need to be set to enable interrupts to PCI.
   - DMAMODE0 = 00000043h (default).
   - Test; 000205C3h (continuous burst with PCI interrupt enabled).

Here are some comments from PLX Technical Support regarding the register settings:

- For DMA, the DMAMODEx registers rather than LBRD0 govern Local bus properties such as bursting. DMA is independent of Local Address Spaces.
- PLX recommends that you set the PCI 2.1 Features Enable bit (MARBR[24] = 1) to enable delayed reads and other protocol enhancements. Please refer to PCI 9054 Design Notes revision 1.6 #3.
- For fastest PCI Direct Slave transfer we recommend enabling continuous prefech (LBRD0[10, 8] = 00), Read Ahead mode (MARBR[28] = 1), and the Pefetchable bit (LAS0RR[3] = 1 in EEPROM). If the PCI 9054 can be permitted to hold onto Local bus ownership (i.e., other local masters don't require the bus), the PCI Target Hold Bus mode can be enabled (MARBR[21] = 0).
- For continous bursting the DMA command code should be Memory Write rather than Memory Write and Invalidate since burst length for the latter is restricted to cache line size.
- For host CPU writes to PCI our Linux driver (to be released in SDK-PRO v3.5) performance is approximately 45 MB/s.

6 Performance Tests

The performance tests in this section were obtained from boards plugged into a 19-slot cPCI backplane. The 19-slot backplane uses one slot for the CPU and has 18 peripheral slots arranged as three separate bridged PCI buses (with 6 slots per segment). The CPU is located in the right-most slot (slot 19) and a system timing board is typically located in the left-most slot (slot 1), leaving 17-slots (slots 2 through 18) available for the COBRA boards. Test results were obtained from multiple boards to see the influence of the PCI-to-PCI bridges located in the backplane. CPU initiated reads should suffer degradation across PCI-to-PCI bridges as the latency from read initiation to read completion increases.

The measured performance rates, in MB/s, are as viewed by a Linux user-space process. The process calls `gettimeofday()` either side of the transactions and uses the measured time, along with the size of the transfer, to generate a transfer rate. This is the sustained transfer rate possible over the PCI bus. The actual PCI bus transfer rate is higher than the rates measured from user-space.

6.1 MARBR register tests

This section tests bits in the Mode/DMA Arbitration (MARBR) register; specifically MARBR[24] PCI r2.1 enable and MARBR[28] PCI read ahead mode. For each test, the serial EEPROM was reprogrammed and the system was rebooted. The settings in the serial EEPROM configuration were relative to that of the single 8MB region configuration shown in Appendix A, with the setting of the MARBR register altered as shown at the top of each table.

Tables 5 through 8 show `read()` performance for various buffer sizes in the different PCI segments in the cPCI crate. The read tests were performed to the SDRAM on the board. From the performance tests, it is clear that MARBR = 0x10200000 gives the 'best' performance. However, even that performance is fairly dismal for a bus capable of 132MB/s speeds. This test was performed using the COBRA debug driver (`cobra_debug.c`) since its implementation of `read()` allows reading from the board. The driver implements the read function by performing a `memcopy_fromio()` into a kernel space buffer, and then a `copy_to_user()` to copy the data into user-space. This is a typical implementation for a PCI target device.

Tables 9 through 12 repeat the read performance tests, however this time `mmap()` and `memcpy()` were used to perform the data transfers. The performance tests are comparable to the `read()` results, with the shorter byte transfers being slightly faster for the `memcpy()` tests since there are no driver calls. The largest performance hit experienced in both sets of tests is that due to the different PCI bus segments. Take for example the Table 8 and 12 results for the 128KB transfers, there is almost a factor of ten in performance degradation for slots 1 to 6 (furthest from the CPU) relative to slots 13 to 18 (closest to the CPU).

Tables 13 through 16 show `write()` performance measurements, and Tables 17 through 20 show `mmap()` and `memcpy()` write performance tests for the same MARBR register changes made during the read tests. The write tests all perform similarly with approximately 10MB/s transfer rates. The first PCI segment (closest to the CPU), slots 13 to 18, has slightly lower performance than the other segments, probably due to the difference in write posting (acceptance) speed between the PLX-9054's that accept writes on the first segment versus the PCI-to-PCI bridge on the first segment that then passes the writes onto the other segments.

Table 5: PCI `read()` performance; Single 8M configuration with MARBR = 0x00200000.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	0.80	1.00	1.33
8	0.89	1.14	1.60
16	1.14	1.33	1.78
32	1.33	1.45	1.88
64	1.36	1.49	2.06
128	1.38	1.52	2.13
256	1.40	1.52	2.13
512	1.40	1.54	2.16
1024	1.40	1.54	2.16
2048	1.40	1.54	2.17
4096	1.40	1.55	2.16
8192	1.40	1.55	2.17
16384	1.40	1.54	2.17
32768	1.40	1.53	2.17
65536	1.40	1.53	2.16
131072	1.40	1.52	2.16

Table 6: PCI `read()` performance; Single 8M configuration with MARBR = 0x01200000, i.e., MARBR[24] = 1 PCI r2.1 features enable.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	0.80	1.00	1.00
8	1.00	1.14	1.33
16	1.14	1.33	1.60
32	1.28	1.39	1.78
64	1.36	1.52	1.88
128	1.38	1.52	1.91
256	1.38	1.56	1.95
512	1.40	1.57	1.97
1024	1.40	1.57	1.97
2048	1.40	1.57	1.87
4096	1.05	1.58	1.98
8192	0.84	1.58	1.97
16384	0.40	1.32	1.98
32768	0.37	1.44	1.98
65536	0.35	1.53	1.98
131072	0.32	1.46	1.97

Table 7: PCI `read()` performance; Single 8M configuration with MARBR = 0x10200000, i.e., MARBR[28] = 1 Read ahead mode.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	0.80	1.00	1.00
8	1.00	1.14	1.60
16	1.23	1.33	2.29
32	1.23	1.45	2.67
64	1.33	1.49	2.78
128	1.35	1.52	3.37
256	1.34	1.52	3.46
512	1.35	1.54	3.51
1024	1.35	1.54	3.56
2048	1.35	1.54	3.57
4096	1.36	1.55	3.52
8192	1.36	1.55	3.51
16384	1.33	1.54	3.53
32768	1.36	1.53	3.51
65536	1.36	1.53	3.56
131072	1.36	1.52	3.55

Table 8: PCI `read()` performance; Single 8M configuration with MARBR = 0x11200000, i.e., MARBR[24] = 1 PCI r2.1 features enable and MARBR[28] read ahead mode.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	0.80	1.00	1.00
8	1.00	1.14	1.60
16	1.14	1.33	2.29
32	1.28	1.45	2.91
64	1.31	1.49	3.20
128	1.36	1.52	3.28
256	1.35	1.55	3.46
512	0.38	1.56	3.53
1024	0.38	1.56	3.54
2048	0.38	1.57	3.57
4096	0.41	1.57	3.58
8192	0.27	1.14	3.56
16384	0.45	1.32	3.50
32768	0.35	1.22	3.57
65536	0.38	1.24	3.57
131072	0.40	1.28	3.56

Table 9: PCI read performance using `mmap()/memcpy`; Single 8M configuration with MARBR = 0x00200000.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	1.00	1.33	2.00
8	1.33	1.60	1.60
16	1.33	1.45	2.00
32	1.39	1.52	2.13
64	1.39	1.52	2.13
128	1.39	1.56	2.17
256	1.39	1.55	2.17
512	1.40	1.54	2.17
1024	1.40	1.55	2.17
2048	1.40	1.55	2.18
4096	1.40	1.55	2.16
8192	1.40	1.55	2.18
16384	1.40	1.55	2.17
32768	1.40	1.54	2.18
65536	1.40	1.55	2.18
131072	1.40	1.55	2.17

Table 10: PCI read performance using `mmap()/memcpy`; Single 8M configuration with MARBR = 0x01200000, i.e., MARBR[24] = 1 PCI r2.1 features enable.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	1.00	1.33	2.00
8	1.33	1.33	2.00
16	1.45	1.45	1.78
32	1.33	1.52	2.00
64	1.42	1.56	2.00
128	1.41	1.56	1.97
256	1.39	1.57	1.83
512	1.40	1.57	1.98
1024	1.40	1.58	1.98
2048	1.40	1.58	1.98
4096	1.40	1.14	1.98
8192	1.04	1.14	1.98
16384	1.20	1.23	1.98
32768	1.29	1.18	1.98
65536	1.15	1.44	1.98
131072	1.21	1.27	1.98

Table 11: PCI read performance using `mmap()/memcpy`; Single 8M configuration with MARBR = 0x10200000, i.e., MARBR[28] = 1 Read ahead mode.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	1.00	1.33	1.33
8	1.33	1.33	2.67
16	1.33	1.45	2.67
32	1.33	1.52	3.20
64	1.33	1.56	3.20
128	1.35	1.54	3.56
256	1.35	1.56	3.51
512	1.36	1.56	3.56
1024	1.36	1.55	3.57
2048	1.36	1.55	3.58
4096	1.36	1.55	3.58
8192	1.36	1.55	3.59
16384	1.36	1.55	3.59
32768	1.35	1.55	3.58
65536	1.36	1.55	3.58
131072	1.36	1.55	3.59

Table 12: PCI read performance using `mmap()/memcpy`; Single 8M configuration with MARBR = 0x11200000, i.e., MARBR[24] = 1 PCI r2.1 features enable and MARBR[28] read ahead mode.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	1.00	1.33	1.33
8	1.33	1.33	2.67
16	1.23	1.45	2.67
32	1.33	1.52	3.20
64	1.36	1.56	3.20
128	1.35	1.56	3.46
256	1.37	1.56	3.51
512	1.37	1.57	3.53
1024	0.38	1.57	3.57
2048	0.24	1.57	3.58
4096	0.41	1.14	3.59
8192	0.29	1.32	3.59
16384	0.31	1.57	3.59
32768	0.27	1.18	3.58
65536	0.28	1.27	3.59
131072	0.34	1.29	3.59

Table 13: PCI `write()` performance; Single 8M configuration with MARBR = 0x00200000.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	2.00	2.00	2.00
8	4.00	4.00	4.00
16	4.00	5.33	5.33
32	6.40	8.00	6.40
64	8.00	8.00	8.00
128	9.14	9.14	9.85
256	10.24	9.85	9.85
512	10.45	10.45	9.85
1024	10.34	10.34	9.57
2048	10.29	10.24	9.53
4096	10.24	10.24	9.20
8192	10.19	10.18	9.49
16384	10.05	10.11	9.47
32768	10.08	10.09	9.41
65536	10.06	10.05	9.46
131072	10.03	10.04	9.43

Table 14: PCI `write()` performance; Single 8M configuration with MARBR = 0x01200000, i.e., MARBR[24] = 1 PCI r2.1 features enable.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	2.00	2.00	2.00
8	2.67	4.00	2.67
16	5.33	5.33	5.33
32	6.40	6.40	6.40
64	8.00	8.00	8.00
128	9.14	9.85	9.14
256	9.85	9.85	9.85
512	10.45	10.45	9.85
1024	10.34	10.24	9.66
2048	10.24	10.24	9.57
4096	10.24	10.24	9.50
8192	10.19	10.18	9.48
16384	10.12	10.11	9.47
32768	10.09	10.08	9.46
65536	10.07	10.07	9.45
131072	10.04	10.04	9.40

Table 15: PCI `write()` performance; Single 8M configuration with MARBR = 0x10200000, i.e., MARBR[28] = 1 Read ahead mode.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	1.33	2.00	1.33
8	2.67	2.67	2.67
16	5.33	5.33	5.33
32	8.00	8.00	8.00
64	8.00	8.00	8.00
128	9.85	9.85	9.14
256	10.24	10.24	9.85
512	10.45	10.45	9.85
1024	10.34	10.24	9.57
2048	10.24	10.24	9.53
4096	10.24	10.21	9.50
8192	10.19	10.18	9.48
16384	10.12	10.11	9.47
32768	10.09	10.08	9.46
65536	10.07	10.05	9.45
131072	10.03	10.02	9.31

Table 16: PCI `write()` performance; Single 8M configuration with MARBR = 0x11200000, i.e., MARBR[24] = 1 PCI r2.1 features enable and MARBR[28] read ahead mode.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	1.33	2.00	2.00
8	4.00	4.00	2.67
16	5.33	5.33	5.33
32	6.40	6.40	6.40
64	8.00	9.14	8.00
128	9.85	9.85	9.14
256	9.85	10.24	9.85
512	10.45	10.45	9.85
1024	10.24	10.34	9.66
2048	10.24	10.24	9.14
4096	10.24	10.24	9.50
8192	10.19	10.18	9.34
16384	10.12	10.11	9.41
32768	10.09	10.08	9.43
65536	10.06	10.06	9.45
131072	10.03	10.04	9.43

Table 17: PCI write performance using `mmap()/memcpy`; Single 8M configuration with MARBR = 0x00200000.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	4.00	4.00	4.00
8	8.00	8.00	8.00
16	8.00	16.00	8.00
32	10.67	10.67	10.67
64	10.67	10.67	10.67
128	10.67	10.67	10.67
256	11.13	10.67	10.67
512	10.67	10.89	9.85
1024	10.45	10.45	9.66
2048	10.34	10.34	9.57
4096	10.29	10.29	9.53
8192	10.21	10.20	9.51
16384	10.10	10.11	9.43
32768	10.10	10.13	9.50
65536	10.11	10.10	9.48
131072	10.10	10.10	9.48

Table 18: PCI write performance using `mmap()/memcpy`; Single 8M configuration with MARBR = 0x01200000, i.e., MARBR[24] = 1 PCI r2.1 features enable.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	4.00	4.00	4.00
8	4.00	8.00	8.00
16	8.00	8.00	8.00
32	10.67	8.00	10.67
64	10.67	10.67	10.67
128	10.67	10.67	10.67
256	11.13	10.67	10.67
512	10.67	10.67	10.04
1024	10.45	10.56	9.66
2048	10.40	10.34	9.57
4096	10.29	10.29	9.53
8192	10.21	10.20	9.50
16384	10.16	10.16	9.45
32768	10.12	10.12	9.50
65536	10.11	10.11	9.36
131072	10.10	10.10	9.48

Table 19: PCI write performance using `mmap()/memcpy`; Single 8M configuration with MARBR = 0x10200000, i.e., MARBR[28] = 1 Read ahead mode.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	4.00	4.00	4.00
8	8.00	4.00	8.00
16	8.00	16.00	8.00
32	10.67	10.67	8.00
64	10.67	10.67	10.67
128	10.67	10.67	10.67
256	11.13	10.67	10.67
512	10.67	10.89	10.04
1024	10.45	10.45	9.66
2048	10.34	10.34	9.57
4096	10.29	10.29	9.53
8192	10.20	10.04	9.50
16384	10.15	10.15	9.50
32768	10.12	10.13	9.19
65536	10.11	10.11	9.48
131072	10.10	10.10	9.48

Table 20: PCI write performance using `mmap()/memcpy`; Single 8M configuration with MARBR = 0x11200000, i.e., MARBR[24] = 1 PCI r2.1 features enable and MARBR[28] read ahead mode.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	4.00	4.00	4.00
8	8.00	8.00	8.00
16	8.00	8.00	8.00
32	10.67	8.00	8.00
64	10.67	10.67	10.67
128	10.67	10.67	10.67
256	10.67	10.67	10.67
512	10.89	10.89	9.85
1024	10.45	10.45	9.66
2048	10.40	10.34	9.57
4096	10.27	10.27	9.53
8192	10.24	10.09	9.50
16384	10.16	10.12	9.42
32768	10.13	10.12	9.50
65536	10.11	10.09	9.48
131072	10.01	10.02	9.48

6.2 LBDR0 register tests

The default value for the Local Bus Descriptor Register for Address Space 0 (LBDR0) is 0x40430043 (see the register dump in Appendix C). The tests in the last section were performed with a LBDR0 setting of 0x4B4300C3 (see the EEPROM configuration file in Appendix A). The LBRD0 bit changes between those two settings are:

- LBDR0[24] Burst enable.
- LBDR0[25] Extra long load from serial EEPROM (so that the VPD registers stay enabled).
- LBDR0[27] PCI target write mode.
- LBRD0[7] `BTERM#` input enable.

The COBRA boards implement burst-mode transaction from the SDRAM, so LBRD0[7] should always be set. Similarly the extra long serial EEPROM load is always required so bit LBDR0[25] should also be set. This leaves burst enable and PCI target write mode as the two bits to investigate. Clearing burst mode should degrade both reads and writes, while clearing the PCI target write mode should decrease write performance. For the tests in this section, the single 8MB region EEPROM configuration was used, the LBRD0 setting was changed, and the system rebooted.

The tables in the last section for MARBD = 0x10200000 show the read and write performance with bursting enabled and the PCI write mode set. Tables 21 through 24 show read and write performance measurements when bursting is disabled and the PCI write mode bit is clear (i.e., LBRD0 = 0x424300C3). The read performance is degraded further on the segments farthest from the CPU slot, whereas the write performance is barely affected. The lack of change in write performance is due to the fact that the CPU initiated writes are not generating burst transactions to the PCI bus (a PCI bus analyzer could be used to confirm this statement).

The burst enable bit and the PCI write mode bit do have definite effects on PCI performance, its just that CPU initiated transactions can not show the effect. The DMA tests in Section 6.4 show that PCI performance is optimized with both of these bits set.

Table 21: PCI `read()` performance; Single 8M configuration with LBDR0 = 0x424300C3.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	0.44	0.80	1.00
8	0.53	0.80	1.60
16	0.62	0.80	2.29
32	0.67	0.89	2.67
64	0.66	0.88	3.05
128	0.62	0.87	3.28
256	0.67	0.88	3.46
512	0.67	0.88	3.53
1024	0.67	0.87	3.40
2048	0.68	0.87	3.57
4096	0.68	0.87	3.58
8192	0.68	0.87	3.58
16384	0.68	0.87	3.58
32768	0.68	0.87	3.57
65536	0.68	0.87	3.57
131072	0.68	0.87	3.56

Table 22: PCI read performance using `mmap()/memcpy`; Single 8M configuration with LBRD0 = 0x424300C3.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	0.57	1.00	2.00
8	0.57	0.89	2.00
16	0.64	0.84	2.67
32	0.70	0.91	3.20
64	0.66	0.89	3.37
128	0.68	0.88	3.46
256	0.65	0.88	3.51
512	0.67	0.88	3.56
1024	0.68	0.88	3.57
2048	0.68	0.88	3.58
4096	0.68	0.87	3.59
8192	0.68	0.87	3.59
16384	0.67	0.87	3.50
32768	0.68	0.87	3.59
65536	0.68	0.87	3.58
131072	0.67	0.87	3.57

Table 23: PCI `write()` performance; Single 8M configuration with LBDR0 = 0x424300C3.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	2.00	2.00	2.00
8	4.00	2.67	2.67
16	5.33	4.00	5.33
32	6.40	6.40	8.00
64	8.00	8.00	9.14
128	9.85	9.85	9.85
256	9.85	9.85	6.74
512	10.45	10.45	9.66
1024	10.34	10.34	9.66
2048	10.24	10.24	9.57
4096	10.21	10.24	9.50
8192	10.19	10.18	9.48
16384	10.12	10.10	9.47
32768	10.09	10.08	9.46
65536	10.06	10.06	9.46
131072	10.03	10.04	9.43

Table 24: PCI write performance using `mmap()/memcpy`; Single 8M configuration with LBDR0 = 0x424300C3.

Bytes	Rate (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18
4	4.00	4.00	4.00
8	8.00	8.00	8.00
16	8.00	8.00	16.00
32	10.67	8.00	10.67
64	10.67	10.67	10.67
128	10.67	10.67	10.67
256	10.67	11.13	10.67
512	10.67	10.67	10.04
1024	10.45	10.45	9.66
2048	10.34	10.34	9.57
4096	10.06	10.29	9.53
8192	10.20	10.21	9.51
16384	10.15	10.15	9.50
32768	10.12	10.12	9.50
65536	10.11	10.11	9.48
131072	10.11	10.10	9.48

6.3 PLX knowledge base articles

The results in the last couple of sections show the dismal performance achievable with CPU initiated read and write transactions. This poor performance is not due to the PLX-9054 bridge, its a 'feature' of the Intel Pentium. For example, the following comments were obtained from the PLX knowledge base:

> "**x86 architecture does not perform burst read**"
>
> If you have a PC adapter card application, Direct Slave burst read cycles cannot be performed since most PCs do not support burst read. However, bursting can be done using a PCI bus master with DMA capability (for example PCI 9054RDK). To tell if a PC is capable of doing burst read, we can look at the `FRAME#` signal. When it is driven by the master, in a PC, `FRAME#` is asserted for no longer than one clock. This indicates it is doing single cycle. When we have a PCI 9054RDK board plugged in and the PCI 9054 is the Master during a DMA cycle, you will see `FRAME#` stays longer than one clock. So we know it is doing burst read. Most PC systems do single cycles on read during a Direct Slave access. It is a limitation on a PC system. Please use DMA transfer, and you will see `FRAME#` stays longer than one clock, indicating bursting capability.
>
> "**Cached read mode**"
>
> First the North bridge starts a read cycle and requests some data from the PLX device. At this time the internal Target read FIFO of the PLX device is empty and it has to fetch the data from the local bus. In the days before PCI revision 2.1, the only way that a target could signal to an initiator that it did not have the data available was by throttling the `TRDY#` (target ready) signal. This had the effect of holding the entire PCI bus until the local data was available. If a slow device was attached to the PCI bus this could result in very poor PCI performance. To overcome this, from PCI 2.1 onwards, a target may issue a "Retry" to the initiator. This tells the initiator that the target is not ready yet and that it should come back and retry the same access later. It is also known as Delayed Read Mode. So, the PLX device issues a "Retry" and fetches the data from the local bus into its FIFO. Some time later, the North bridge tries to read again. This time we have the data ready and the PLX device can send the data. For every read access there will be a read, retry and read. This is very slow. A typical system will achieve a bandwidth of around 4-5Mbytes/s. Given the maximum bandwidth is supposed to be 132Mbytes/s (for 33MHz and 32 bit) this is a terrible use of the bandwidth. Don't blame PLX! Our devices can burst read and write much faster than this. There are some reasons (beyond the scope of this article) why Intel does not allow the North bridge to burst read, so it's probably not a good idea to shout at them either. What can we do to resolve this problem? In PLX devices we have the ability to pre-fetch and cache data in the FIFOs. When the first read cycle comes in, instead of just fetching the word which was requested, we can fetch several words. If the North bridge requests a read from a sequential address, we can immediately respond - removing the need for the retry cycle. This method can approximately double the bandwidth to about 8-10 Mbytes/s (we have a few customers who have achieved 15Mbytes/s this way). However, this is still a long way from the 132Mbytes/s. If you want to get really high data rates then you will have to use an initiator and push the data onto the PCI bus using writes. PLX Initiator devices incorporate advanced DMA engines. With these we have been able to achieve sustained data rates of up to 122Mbytes/s. In summary, even though your application may only require the functionality of a target, if your initiator can only single cycle read you may be forced to use an initiator device.

“Limited PCI bursting with x86 architecture”

Reads originated from an x86 (Intel) architecture CPU destined to a PCI slave device (such as PLX 9050) will never be bursted. This is a x86 IA32 limitation. By their nature, reads from a PCI slave device are to uncached memory. (The PCI device is mapped into uncached memory). As a result, reads are blocking (another read can’t emerge from the CPU until the earlier read is completed). So this leads to the situation that the largest ”burst” of data is confined to the largest ”single” read that the x86 CPU can perform to uncached memory. Currently, this is a 64 bit read, done using the instruction MOVQ r64, mem. So, the largest read burst that you can get by using an x86 CPU to read a PCI slave is a 64 bit, or ”two” data phase burst. (Pretty darn short burst, since it is not even a full cache line). Tricks can be played that could allow a faster read, but they are involved and error prone. For instance, the PCI slave device’s memory could be mapped as cacheable (likely writethrough mode would be best). Then when the PCI slave device was read by the CPU execution unit, the cache unit would pull in a whole cache line at a time. This could be done for consecutive addresses. The result would be ”bursting” of x86 cacheline size (64 bytes per line), or 16 PCI data transfer clocks. (Not that fantastic either, but better). Unfortunately, you then have to flush the CPU cache before transferring again from the same address. (That is usually detrimental to system performance, so no one does it). In summary, the x86 will block on a read to uncached memory. This limits the ”burst” size to the size of the largest single ”read” transaction to uncached memory, which is 64-bits, 4xbytes, 2 PCI data clocks, all byte enables. If you play games with the cache, and mark the memory as write through, then you can get 64 bytes reads, but the cost is flushing the entire CPU cache whenever address reads will repeat. This is so expensive that it defeats the gains realized in the larger bursts, so it usually is not done. (Plus getting an OS interface that allows WBINVD cache flush instruction to be executed is a hassle). As to burst writes, chipset (Intel 440BX and VIA MVP3, for example) architectures won’t burst more than 4 LWords at a time out of main memory to a PCI target device. This is because uncached writes coming out of the CPU don’t gather in posted write buffers in the chipset in greater numbers than 4 LWords at a time to allow greater bursts. You won’t get greater bursts, no matter how hard you try. You can get limited bursting if you work at it. The recommended combination is: 1. Map the device memory as USWC (write combined) if possible. NT has functions that allow for this, and some other OS do as well. This facility resets the x86 architecture MTRR registers to give a USWC format to some memory. Some video drivers call this interface; try looking at a video driver sample in the NT DDK. 2. Use a chipset with lots of posted write buffers, and one which can combine sequential writes to linearly sequential addresses into a single burst. A chipset that can combine sequential partial writes into a single PCI burst is the best. 3. Use the MOVQ mem, m64 instruction in a tight loop rather than the REP movsd instruction. The former at least outputs 64-bit partial writes, the latter only does 32-bit partial writes. Write performance with bursts of 4 LWords originating from an x86 architecture CPU is approximately 50MB/s. High performance devices such as the 9054 are set up as bus masters so they can master their own traffic with longer bursts. The 9054 can achieve transfer performance of 122MB/s.

Table 25: PCI segment-to-segment DMA performance (1MB transfers).

Master (DMA) slot	Target slot	DMA 'read' (MB/s)	DMA 'write' (MB/s)
6	7	39	88
7	6	32	69
12	13	39	113
13	12	39	69
6	13	27	52
13	6	27	56

6.4 DMA transfers between boards

The recommended method for transferring large blocks of data is via DMA. Since the Intel Pentium does not have a local DMA controller, the PLX-9054 DMA controller needs to be used for both read and write DMA operations. To determine the maximum possible bus bandwidth achievable from a COBRA board, it was necessary to add an `ioctl()` call to the COBRA debug device driver to configure board-to-board DMA. Basically the CPU sets up the registers on the Master board and then waits for the DMA interrupt to time how long the DMA transaction took. The ultimate limit to the DMA transfer tests is the performance of the backend logic. The COBRA board SDRAM controller implements a burst-mode interface to the PLX-9054 so can achieve over 100MB/s performance. It can not reach the maximum performance of 132MB/s since the SDRAM refresh cycles and page boundaries cause breaks in the burst transactions.

A board-to-board test was setup to transfer 1MB of data from one board to another. All boards were configured with the single 8MB EEPROM configuration given in Appendix A. The results of board-to-board DMA transfer tests are summarized as follows;

- Transfers between boards on the same PCI bus segment occurred at 119.5MB/s in either direction (DMA 'read' from PCI into local memory, or DMA 'write' from local memory onto the PCI bus). (This is the rate viewed from the Linux user-space process, so the transfer rate on the PCI bus is actually higher).
- Table 25 shows the transfer rates between boards on different segments. The different PCI bridges result in transfer rate asymmetries. The performance measurements across similarly located bridges is not even matched!

Given these board-to-board observations, one could suspect that the performance of the DMA transfers from the host to the COBRA boards will ultimately be limited by the configuration of the PCI-to-PCI bridges. Configuration of the PCI-to-PCI bridges in the system to optimize performance is a possibility. However this has not been investigated.

Table 26 investigates the performance of board-to-board transfers with various combinations of bit settings on both the Master (DMA) and Target PLX-9054s. The performance tests across PCI-to-PCI bridges showed quite a lot of variation. The optimal performance parameters were obtained with the settings used in the single 8MB region EEPROM configuration shown in Appendix A.

Table 26: PCI board-to-board (same segment) DMA performance (1MB transfers).

DMA Test Case	Master DMAMODE0	Master/Target MARBR	Master/Target LBRD0	Transfer rate (MB/s)
Same PCI bus				
Non-burst				
Read	000204C3h	00200000	424300C3	≥13
Write	000204C3h	00200000	424300C3	7.7
Burst				
Read	000205C3h	00200000	434300C3	119.3
Write	000205C3h	00200000	434300C3	10.1
Write (keep PCI bus)	000205C3h	00200000	4B4300C3	119.7
Write (keep and PCI r2.1)	000205C3h	01200000	4B4300C3	119.7
Read (read ahead)	000205C3h	10200000	4B4300C3	119.7
Read (PCI r2.1)	000205C3h	01200000	4B4300C3	119.7
Read (PCI r2.1 and read ahead)	000205C3h	11200000	4B4300C3	119.7
Bridged PCI bus				
Non-burst				
Read	000204C3h	00200000	424300C3	≤7.4
Write	000204C3h	00200000	424300C3	7.7
Burst				
Read	000205C3h	00200000	434300C3	≥18
Write	000205C3h	00200000	434300C3	≥10
Write (keep PCI bus)	000205C3h	00200000	4B4300C3	≥69
Write (keep and PCI r2.1)	000205C3h	01200000	4B4300C3	≥69
Read (read ahead)	000205C3h	10200000	4B4300C3	≥32
Read (PCI r2.1)	000205C3h	01200000	4B4300C3	≥18
Read (PCI r2.1 and read ahead)	000205C3h	11200000	4B4300C3	≥32

Table 27: PCI DMA `read()` (i.e., board-to-host) performance.

Bytes	Block DMA (MB/s)			Scatter-Gather DMA (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18	Slots 1-6	Slots 7-12	Slots 13-18
1K	36.6	39.4	42.7	4.7	4.5	4.0
2K	53.9	58.5	64.0	8.6	8.8	9.1
4K	65.0	71.9	77.3	15.3	10.6	16.1
8K	73.8	81.1	84.5	26.3	22.4	28.0
16K	77.3	80.3	84.5	39.3	34.2	43.1
32K	79.0	84.2	85.1	55.1	61.4	62.5
64K	80.6	81.6	85.9	66.8	77.8	78.7
128K	80.2	77.9	85.3	75.5	88.6	89.5

Table 28: PCI DMA `write()` (i.e., host-to-board) performance.

Bytes	Block DMA (MB/s)			Scatter-Gather DMA (MB/s)		
	Slots 1-6	Slots 7-12	Slots 13-18	Slots 1-6	Slots 7-12	Slots 13-18
1K	12.0	13.3	15.8	2.5	2.7	2.8
2K	13.6	16.0	18.3	5.2	5.9	6.2
4K	14.2	17.3	19.1	6.2	7.2	7.7
8K	14.6	17.7	19.8	8.8	10.3	11.2
16K	14.8	18.0	20.2	11.0	13.1	14.5
32K	14.8	18.1	20.3	12.8	15.4	17.0
64K	15.0	18.3	20.5	13.9	16.7	18.6
128K	15.0	18.2	20.4	14.4	17.4	19.4

6.5 DMA transfers to and from the host

Tables 27 and 28 show the read and write performance of DMA. For the COBRA system, DMA will be used to transfer processed data from the boards to the host, so the performance measurements of interest to that system are the DMA read measurements. The COBRA debug driver implements three read/write functions; CPU initiated, block-DMA, and scatter-gather DMA. The block-DMA implementation allocates a single kernel buffer of up to 128KB and DMAs between the host and board using that buffer. The contents of the user-space buffer are either copied into the kernel buffer for a write, or copied out of the kernel buffer for a read. The scatter-gather DMA implementation maps the user-space buffer into the kernel using Linux's kiobuf mechanism. The user-space buffer appears in the kernel as a scatter-gather list of 4K pages. The PLX-9054 DMA controller DMAs into or from these pages using a DMA descriptor list located in the kernel (locating the list in local memory did not affect performance). The difference between the block DMA performance measurements and the scatter-gather DMA performance measurements can be attributed to the lower overhead experienced by the block DMA operation. The poor DMA write performance could be due to non-optimal PCI-to-PCI bridge settings.

Since the COBRA boards will not be transferring more than 128K per DMA operation, the COBRA driver uses block DMA for data transfers.

7 User-space Application and Driver Tests

7.1 Host transfers and time corruption

The original version of the COBRA driver used CPU initiated reads and writes to perform data transfers, as testing of that driver showed the performance to be adequate. When the driver was used in the 19-slot cPCI chassis (which is segmented into three 6-slot peripheral segments by two PCI-to-PCI bridges), the driver still appeared to work fine, however Linux time was being corrupted. Operating boards in the slots nearest the CPU seemed to work fine, but moving them onto the next PCI segments caused time to break. Observing the status of time via NTP (`/usr/sbin/ntpq -p`) showed NTP jumping by a few milliseconds, to as much as several seconds! This issue was the reason behind the tests performed in this document.

The COBRA debug driver (`cobra_debug.c`) was developed to investigate CPU initiated transactions versus block mode and scatter-gather mode DMA transfers. That driver implements the PLX device driver, and a trimmed down version of the COBRA control driver, with the `read()` and `write()` functions implementing board read/write access (the COBRA control driver read/write interface implements a host-to-DSP message queue). Testing with the debug driver showed that optimal data transfer performance for our specific use of the PCI bus was achieved using block mode DMA to transfer processed data, and monitor data. Time corruption was still observed after DMA was implemented in the COBRA data device, however, once transfers were started and NTP was restarted time corruption no longer occurred. The problem was isolated as being due to the driver clearing SDRAM on device open. This operation caused the CPU to take up to 800ms to complete the `open()` system call. Removing the clearing of SDRAM reduced the open time to on the order of a millisecond, and stopped time corruption from occurring at device `open()`. Unfortunately, tests in the next section show that time glitches still occur.

The main result of the PCI tests is that Linux is unable to perform PCI operations that will cause it to become ‘too busy’, where ‘too busy’ either represents a transaction of longer than an OS tick of 10ms, or a transaction that is shorter than 10ms but causes the OS to miss, or service too late, a critical OS function. With interrupts enabled, one would expect that an OS tick would generate an interrupt, and that interrupt would be serviced regardless of what the driver was doing, however, this expectation appears to be false. User-space programs should be able to perform I/O intensive operations to the PCI bus, since they will be interrupted by the OS (although NTP kicks of 30ms have been observed for some operations). So, the conclusion is that you should not perform I/O intensive operations in the device driver! This is fairly obvious requirement, but who knew that PCI bus performance of the Intel Pentium for CPU initiated transactions was so bad!?

7.2 Testing notes

After making modifications to the COBRA device driver and debugging those modifications. The following tests were performed:

- A dual SZA chassis test containing 14 correlator boards in the first chassis and 13 correlator boards in the second. The number of boards in the two crates reflected the capabilities of the timing generator boards—the timing generator boards were designed for use in the COBRA crates and so the boards only supply the PLL reference clock to 13 slots in the new crates (the board in the first crate was modified to drive an extra slot). Various short-timescale and long-timescale tests were performed, and 30ms time glitches were observed infrequently.
- COBRA crate single band test. An COBRA band consists of 3 digitizer boards and 3 correlator boards. The digitizer RFs were fed from the lab noise source. NTP had to be restarted after the boards were setup, and tests were ran for an hour. A single timeout error was recorded during the test. Time glitches that are then pulled back in by the NTP server usually generate tens of timestamp errors, so the single error recorded during this test was likely due to OS scheduling.
- SZA crate dual-band test. An SZA band consists of 4 digitizer boards and 3 correlator boards. For the dual-band test, two of the SZA bands were plugged into an SZA crate (14 boards total), and the COBRA 6-antenna lab noise source hardware was plugged into 3 of the four SZA digitizers, with the outputs from those 3 digitizers cabled to the 3 correlator boards in the COBRA 15-baseline arrangement. The boards not plugged into the RF source were programmed to output test pattern data. During some of the crate setup and data capture tests, a couple of 30ms time glitches were observed. A data capture test was then run for two hours, and no timestamp errors were recorded (so either no 30ms glitches occurred, or they occurred and did not trigger a DSP-to-host timestamp tolerance error).

Tests of longer duration will be performed as part of system-level tests using multiple crates of boards. Any issues exposed there will be noted in future revisions of this document.

Adding DMA to the COBRA device driver resolves the issue of time being continuously corrupted when operating in the new SZA crates, however, the testing above showed that time glitches of around 30ms are still possible. Since the NTP service eventually corrects these errors, this residual issue is probably acceptable. The data transfer tests were performed using a transfer period of 100ms to the host (as was used in the COBRA correlator system). CARMA intends to use 500ms transfers to the host, so 30ms glitches relative to a 500ms transfer period will not cause data transport errors to occur. These tests did not include 500ms monitor data transport and transport of the digital delays and other information down to the DSP. When the software that implements these transfers is complete, tests will be performed to confirm that these modifications do not induce NTP glitches.

During one day of testing in the lab with `cobracpu10`, a continuous drift of time relative to NTP was observed. Time would drift until it was about 20ms out, and then the NTP service would either hold it there, or bring it back in. The CPU was rebooted with no COBRA boards in the crate, and the same problem was observed! The source of the problem was never determined. It could have been in the CPU, or the network traffic between the CPU and NTP server could have been the issue. The system should not depend on NTP to do any better than 20ms to 30ms.

The conclusions to be drawn from this testing are; that the major problem of continuous time corruption has been mitigated, and that NTP accuracy is limited to 50ms. Both conclusions are inline with CARMAs requirements.

A EEPROM Configurations

A.1 Single 8MB memory region

The following configuration sets up a single 8MB prefetchable 32-bit memory region. The PLX-9054 BAR1 I/O region is disabled, and overlayed with Local Address Space 0, the 8MB region. The EEPROM configuration file is:

```
# PLX-9054 Serial EEPROM Settings
# DWH 2/02.
#
# Byte   |  Dword
# offset |  value
# address|
#
  00 905410b5
  04 0680000a
  08 00000100
  0c 00000000
  10 00000000
  14 ff800008
  18 00000001
  1c 10200000
  20 00300600
  24 00000000
  28 00000000
  2c 4b4300c3
  30 ffff0000
  34 01390000
  38 013a0000
  3c 00000000
  40 00000000
  44 905410b5
  48 00000000
  4c 00000000
  50 00000000
  54 00004c06
```

A.2 Dual 8MB memory region

The following configuration leaves the PLX-9054 BAR1 I/O region enabled, and sets up BAR2/Local address space 0 as 8MB of prefetchable memory, and BAR3/Local address space 1 as 8MB of non-prefetchable memory. The EEPROM configuration file is:

```
# PLX-9054 Serial EEPROM Settings
# DWH 12/03. Leave PCI9054 I/O region enabled
#
# Byte   |  Dword
# offset |  value
# address|
#
  00 905410b5
  04 0680000a
  08 00000100
  0c 00000000
  10 00000000
  14 ff800008
  18 00000001
  1c 10200000
  20 00300500
  24 00000000
  28 00000000
  2c 4b4300c3
  30 ffff0000
  34 01390000
  38 013a0000
  3c 00000000
  40 00000000
  44 905410b5
  48 ff800000
  4c 00000001
  50 00000073
  54 00004c06
```

B PCI Configuration Space Registers Dumps

This appendix contains the PCI configuration space register values read back using `/sbin/lspci` from the PLX-9054 for; a blank EEPROM, single 8MB region, and dual 8MB region. The three different register settings were read from three boards booted in the same crate.

Blank EEPROM read-back:

```
03:0b.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
Subsystem: PLX Technology, Inc. PCI <-> IOBus Bridge
Control: I/O+ Mem+ BusMaster- SpecCycle- MemWINV+ VGASnoop-
         ParErr- Stepping- SERR+ FastB2B-
Status: Cap+ 66Mhz- UDF- FastB2B+ ParErr- DEVSEL=medium
        >TAbort- <TAbort- <MAbort- >SERR- <PERR-
Interrupt: pin A routed to IRQ 7
Region 0: Memory at f3d00000 (32-bit, non-prefetchable) [size=256]
Region 1: I/O ports at 2000 [size=256]
Region 2: Memory at f3c00000 (32-bit, non-prefetchable) [size=1M]
Region 3: Memory at f3b00000 (32-bit, non-prefetchable) [size=1M]
Expansion ROM at <unassigned> [disabled] [size=64K]
Capabilities: [40] Power Management version 1
Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
Status: D0 PME-Enable- DSel=0 DScale=0 PME-
Capabilities: [48] #06 [0080]
Capabilities: [4c] Vital Product Data

03:0b.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
00: b5 10 54 90 13 01 90 02 0a 00 80 06 08 40 00 00
10: 00 00 d0 f3 01 20 00 00 00 00 c0 f3 00 00 b0 f3
20: 00 00 00 00 00 00 00 00 00 00 00 00 b5 10 54 90
30: 00 00 00 00 40 00 00 00 00 00 00 00 07 01 00 00
40: 01 48 01 00 00 00 00 00 06 4c 80 00 03 00 00 00
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Single 8MB region read-back:

```
03:0d.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
Subsystem: PLX Technology, Inc. PCI <-> IOBus Bridge
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV+ VGASnoop-
         ParErr- Stepping- SERR+ FastB2B-
Status: Cap+ 66Mhz- UDF- FastB2B+ ParErr- DEVSEL=medium
        >TAbort- <TAbort- <MAbort- >SERR- <PERR-
Latency: 64, cache line size 08
Interrupt: pin A routed to IRQ 9
Region 0: Memory at f3d00800 (32-bit, non-prefetchable) [size=256]
Region 1: Memory at fa000000 (32-bit, prefetchable) [size=8M]
Capabilities: [40] Power Management version 1
Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
Status: D0 PME-Enable- DSel=0 DScale=0 PME-
Capabilities: [48] #06 [0080]
Capabilities: [4c] Vital Product Data

03:0d.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
00: b5 10 54 90 17 01 90 02 0a 00 80 06 08 40 00 00
10: 00 08 d0 f3 08 00 00 fa 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 b5 10 54 90
30: 00 00 00 00 40 00 00 00 00 00 00 00 09 01 00 00
40: 01 48 01 00 00 00 00 00 06 4c 80 00 03 00 00 00
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Dual 8MB region read-back:

```
03:0c.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
Subsystem: PLX Technology, Inc. PCI <-> IOBus Bridge
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV+ VGASnoop-
         ParErr- Stepping- SERR+ FastB2B-
Status: Cap+ 66Mhz- UDF- FastB2B+ ParErr- DEVSEL=medium
        >TAbort- <TAbort- <MAbort- >SERR- <PERR-
Latency: 64, cache line size 08
Interrupt: pin A routed to IRQ 10
Region 0: Memory at f3d00400 (32-bit, non-prefetchable) [size=256]
Region 1: I/O ports at 2400 [size=256]
Region 2: Memory at f9800000 (32-bit, prefetchable) [size=8M]
Region 3: Memory at f4000000 (32-bit, non-prefetchable) [size=8M]
Capabilities: [40] Power Management version 1
Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
Status: D0 PME-Enable- DSel=0 DScale=0 PME-
Capabilities: [48] #06 [0080]
Capabilities: [4c] Vital Product Data


03:0c.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
00: b5 10 54 90 17 01 90 02 0a 00 80 06 08 40 00 00
10: 00 04 d0 f3 01 24 00 00 08 00 80 f9 00 00 00 f4
20: 00 00 00 00 00 00 00 00 00 00 00 00 b5 10 54 90
30: 00 00 00 00 40 00 00 00 00 00 00 00 0a 01 00 00
40: 01 48 01 00 00 00 00 00 06 4c 80 00 03 00 00 00
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

C Configuration Registers Dumps

This appendix contains the register values read back from the PLX-9054 BAR0 for; a blank EEPROM, single 8MB region, and dual 8MB region. These registers were read using the `plx_debug` utility found in the COBRA driver source. The three different register settings were read from three boards booted in the same crate.

Blank EEPROM read-back:

```
00: FFF00000 00000000 00200000 00300500
10: FFFF0000 00000000 40430043 00000000
20: 00000000 00000000 00000000 00000000
30: 00000000 00000008 00000000 00000000
40: 00000000 00000000 00000000 00000000
50: 00000000 00000000 00000000 00000000
60: 00000000 00000000 0F010180 080F767E
70: 905410B5 0000000A 00000000 00000000
80: 00000043 00000000 00000000 00000000
90: 00000000 00000043 00000000 00000000
A0: 00000000 00000000 00001010 00200000
B0: 00000000 00000000 00000000 00000000
C0: 00000002 00000000 00000000 00000000
D0: 00000000 00000000 00000000 00000000
E0: 00000000 00000000 00000050 00000000
F0: FFF00000 00000000 00000043 00000000
```

Single 8MB region read-back:

```
00: FF800008 00000001 10200000 00300600
10: 00000000 00000000 4B4300C3 FFFF0000
20: 01390000 013A0000 00000000 00000000
30: 00000000 00000008 00000000 00000000
40: 00000000 00000000 00000000 00000000
50: 00000000 00000000 00000000 00000000
60: 00000000 00000000 0F010180 180F767E
70: 905410B5 0000000A 00000000 00000000
80: 00000043 00000000 00000000 00000000
90: 00000000 00000043 00000000 00000000
A0: 00000000 00000000 00001010 10200000
B0: 00000000 00000000 00000000 00000000
C0: 00000002 00000000 00000000 00000000
D0: 00000000 00000000 00000000 00000000
E0: 00000000 00000000 00000050 00000000
F0: 00000000 00000000 00000000 00000000
```

Dual 8MB region read-back:

```
00: FF800008 00000001 10200000 00300500
10: 00000000 00000000 4B4300C3 FFFF0000
20: 01390000 013A0000 00000000 00000000
30: 00000000 00000008 00000000 00000000
40: 00000000 00000000 00000000 00000000
50: 00000000 00000000 00000000 00000000
60: 00000000 00000000 0F010180 180F767E
70: 905410B5 0000000A 00000000 00000000
80: 00000043 00000000 00000000 00000000
90: 00000000 00000043 00000000 00000000
A0: 00000000 00000000 00001010 10200000
B0: 00000000 00000000 00000000 00000000
C0: 00000002 00000000 00000000 00000000
D0: 00000000 00000000 00000000 00000000
E0: 00000000 00000000 00000050 00000000
F0: FF800000 00000001 00000073 00000000
```

D PCI-to-PCI Bridge Dumps

The following PCI-to-PCI bridge configurations were read during the dual chassis test.

```
# /sbin/lspci

00:00.0 Host bridge: ServerWorks CNB20LE Host Bridge (rev 06)
00:00.1 Host bridge: ServerWorks CNB20LE Host Bridge (rev 06)
00:01.0 Ethernet controller: Intel Corp. 82559ER (rev 09)
00:02.0 Ethernet controller: Intel Corp. 82559ER (rev 09)
00:04.0 PCI bridge: Force Computers: Unknown device 0001 (rev 01)
00:0f.0 ISA bridge: ServerWorks OSB4 South Bridge (rev 50)
00:0f.1 IDE interface: ServerWorks OSB4 IDE Controller
00:0f.2 USB Controller: ServerWorks OSB4/CSB5 OHCI USB Controller (rev 04)
01:04.0 PCI bridge: Intel Corp. 21154 PCI-to-PCI Bridge
01:0a.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
01:0b.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
01:0c.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
01:0d.0 PCI bridge: Intel Corp. 21152 PCI-to-PCI Bridge
02:04.0 PCI bridge: Intel Corp. 21154 PCI-to-PCI Bridge
02:0a.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
02:0b.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
02:0c.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
02:0d.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
02:0e.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
02:0f.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
03:0b.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
03:0c.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
03:0d.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
03:0e.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
03:0f.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
04:04.0 PCI bridge: Digital Equipment Corporation DECchip 21150 (rev 06)
05:04.0 PCI bridge: Intel Corp. 21154 PCI-to-PCI Bridge
05:0a.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
05:0b.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
06:04.0 PCI bridge: Intel Corp. 21154 PCI-to-PCI Bridge
06:0a.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
06:0b.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
06:0c.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
06:0d.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
06:0e.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
06:0f.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
07:0b.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
07:0c.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
07:0d.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
07:0e.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
07:0f.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)

# /sbin/lspci -s 00:04.0 -vv

00:04.0 PCI bridge: Force Computers: Unknown device 0001 (rev 01) (prog-if 00 [Normal
decode])
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR
+ FastB2B-
Status: Cap+ 66Mhz- UDF- FastB2B- ParErr- DEVSEL=medium >TAbort- <TAbort- <MAbort-
>SERR- <PERR-
```

```
Latency: 64, cache line size 08
Region 0: Memory at ed022000 (32-bit, non-prefetchable) [size=4K]
Region 1: I/O ports at 1000 [size=128]
Bus: primary=00, secondary=01, subordinate=07, sec-latency=64
Memory behind bridge: ed100000-ed6fffff
Prefetchable memory behind bridge: ee000000-fb7fffff
BridgeCtl: Parity- SERR- NoISA+ VGA- MAbort- >Reset- FastB2B-
Capabilities: [fc] #06 [0080]
Capabilities: [f0] Message Signalled Interrupts: 64bit- Queue=0/0 Enable-
Address: 00000000  Data: 0000

# /sbin/lspci -s 01:04.0 -vv

01:04.0 PCI bridge: Intel Corp. 21154 PCI-to-PCI Bridge (prog-if 00 [Normal decode])
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR
+ FastB2B-
Status: Cap+ 66Mhz+ UDF- FastB2B+ ParErr- DEVSEL=medium >TAbort- <TAbort- <MAbort-
>SERR- <PERR-
Latency: 64, cache line size 08
Bus: primary=01, secondary=02, subordinate=03, sec-latency=68
Memory behind bridge: ed200000-ed3fffff
Prefetchable memory behind bridge: 00000000ef800000-00000000f4f00000
BridgeCtl: Parity- SERR- NoISA+ VGA- MAbort- >Reset- FastB2B-
Capabilities: [dc] Power Management version 1
Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
Status: D0 PME-Enable- DSel=0 DScale=0 PME-
Bridge: PM- B3+

# /sbin/lspci -s 01:0d.0 -vv

01:0d.0 PCI bridge: Intel Corp. 21152 PCI-to-PCI Bridge (prog-if 00 [Normal decode])
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR
+ FastB2B-
Status: Cap+ 66Mhz- UDF- FastB2B+ ParErr- DEVSEL=medium >TAbort- <TAbort- <MAbort-
>SERR- <PERR-
Latency: 64, cache line size 08
Bus: primary=01, secondary=04, subordinate=07, sec-latency=68
Memory behind bridge: ed400000-ed6fffff
Prefetchable memory behind bridge: 00000000f5000000-00000000fb700000
BridgeCtl: Parity- SERR- NoISA+ VGA- MAbort- >Reset- FastB2B-
Capabilities: [dc] Power Management version 2
Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
Status: D0 PME-Enable- DSel=0 DScale=0 PME-
Bridge: PM- B3+

# /sbin/lspci -s 02:04.0 -vv

02:04.0 PCI bridge: Intel Corp. 21154 PCI-to-PCI Bridge (prog-if 00 [Normal decode])
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR
+ FastB2B-
Status: Cap+ 66Mhz+ UDF- FastB2B+ ParErr- DEVSEL=medium >TAbort- <TAbort- <MAbort-
>SERR- <PERR-
Latency: 64, cache line size 08
Bus: primary=02, secondary=03, subordinate=03, sec-latency=68
Memory behind bridge: ed300000-ed3fffff
```

```
Prefetchable memory behind bridge: 00000000f2800000-00000000f4f00000
BridgeCtl: Parity- SERR- NoISA+ VGA- MAbort- >Reset- FastB2B-
Capabilities: [dc] Power Management version 1
Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
Status: D0 PME-Enable- DSel=0 DScale=0 PME-
Bridge: PM- B3+

# /sbin/lspci -s 04:04.0 -vv

04:04.0 PCI bridge: Digital Equipment Corporation DECchip 21150 (rev 06) (prog-if 00 [
Normal decode])
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR
+ FastB2B-
Status: Cap+ 66Mhz- UDF- FastB2B+ ParErr- DEVSEL=medium >TAbort- <TAbort- <MAbort-
>SERR- <PERR-
Latency: 64, cache line size 08
Bus: primary=04, secondary=05, subordinate=07, sec-latency=68
Memory behind bridge: ed400000-ed6fffff
Prefetchable memory behind bridge: 00000000f5000000-00000000fb700000
BridgeCtl: Parity- SERR- NoISA+ VGA- MAbort- >Reset- FastB2B-
Capabilities: [dc] Power Management version 1
Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
Status: D0 PME-Enable- DSel=0 DScale=0 PME-
Bridge: PM- B3+

# /sbin/lspci -s 05:04.0 -vv

05:04.0 PCI bridge: Intel Corp. 21154 PCI-to-PCI Bridge (prog-if 00 [Normal decode])
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR
+ FastB2B-
Status: Cap+ 66Mhz+ UDF- FastB2B+ ParErr- DEVSEL=medium >TAbort- <TAbort- <MAbort-
>SERR- <PERR-
Latency: 64, cache line size 08
Bus: primary=05, secondary=06, subordinate=07, sec-latency=68
Memory behind bridge: ed500000-ed6fffff
Prefetchable memory behind bridge: 00000000f6000000-00000000fb700000
BridgeCtl: Parity- SERR- NoISA+ VGA- MAbort- >Reset- FastB2B-
Capabilities: [dc] Power Management version 1
Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
Status: D0 PME-Enable- DSel=0 DScale=0 PME-
Bridge: PM- B3+

# /sbin/lspci -s 06:04.0 -vv

06:04.0 PCI bridge: Intel Corp. 21154 PCI-to-PCI Bridge (prog-if 00 [Normal decode])
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR
+ FastB2B-
Status: Cap+ 66Mhz+ UDF- FastB2B+ ParErr- DEVSEL=medium >TAbort- <TAbort- <MAbort-
>SERR- <PERR-
Latency: 64, cache line size 08
Bus: primary=06, secondary=07, subordinate=07, sec-latency=68
Memory behind bridge: ed600000-ed6fffff
Prefetchable memory behind bridge: 00000000f9000000-00000000fb700000
BridgeCtl: Parity- SERR- NoISA+ VGA- MAbort- >Reset- FastB2B-
Capabilities: [dc] Power Management version 1
```

```
Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
Status: D0 PME-Enable- DSel=0 DScale=0 PME-
Bridge: PM- B3+
```