PowerPC Embedded Processors Application Note

IBM CPC700 Memory Controller & PCI Bridge Frequently Asked Questions (FAQ)

IBM Microelectronics
4400 Silicon Drive
Research Triangle Park, NC 27709
ppsupp@us.ib.com
http://www.chips.ibm.com
(919) 543-5701

08/27/01

Abstract - This document addresses topics and questions frequently asked about the CPC700 Memory Controller & PCI Bridge Chip. This information highlights and complements the information found in the CPC700 Users Manual and CPC700 Memory Controller & PCI Bridge Technical Datasheets.

1. What is the intended application for CPC700?

The CPC700 is meant to be a general purpose solution to problems of interfacing a high performance, superscalar 60x, 7xx processor to a PCI bus.

2. Does the CPC700 support cache snooping?

The CPC700 does not support cacheable memory on the PCI bus. The only snooping supported by the CPC700 is snooping on the 60x L1 cache on the PCI access to CPC700 local memory.

3. How much SDRAM memory can CPC700 support?

The CPC700 was designed to support two rows (banks) of 64 bit unbuffered SDRAM without the addition of buffers to the board. The PowerPC$_{TM}$ 6xx/7xx Evaluation board demonstrates that capability using one 168 pin DIMM, which may be populated either single or double sided. At the time of this testing 128MB was the largest DIMM commonly available. Sizes 16 through 128MByte, without ECC were tested and recommended accordingly. Larger DIMMs at the same time were registered SDRAM, which can be used in the non-registered mode (simulating unbuffered SDRAM) by appropriate choice of a REGE input. The wiring of the board allows the use of these parts. However, we were not able to obtain timing information for the DIMMs used in this mode from the DIMM data sheets. Therefore, we neither tested nor recommend this mode.

If they work with the chip they should work with the board hardware, since it is a glueless connection. The only board complication might be in the 6xx/7xx evaluation board code. This code reads the Serial Presence Detect ROM on the DIMM and configures CPC700 appropriately to match the DIMM.

4. How does the CPC700 performance compare to the MPC106?

Please *contact the PowerPC Embedded Applications Support hotline at: ppcsupp@us.ibm.com* for performance details.

5. What is the relationship between the CPU clock, CPC700 clock, and SDRAM clock?

The SYS_CLK into CPC700 must be ½ the frequency of the 60x bus clock and the memory clock. All must be in phase, synchronized on the rising edge. All CPC700 events (inputs & outputs) are synchronized on the rising edge of the clock.

CPC700 contains a PLL which doubles the 33MHz input to create a 66MHz phase aligned internal clock which matches the 60x bus clock. Our timing analysis budgets for 1ns of clock skew.

6. Are there models available for the CPC700?

Yes. CPC700 IBIS models may be obtained by contacting *PowerPC Embedded Applications Support hotline at*: *ppcsupp@us.ibm.com.* You can also download the models from our web page *http://www.chips.ibm.com/products/powerpc/chips/7xx.html*.

7. On the CPC700, can the PCI clock run asynchronous to the CPU bus?

Yes. The CPC700 works in both modes asynchronous & synchronous. The sync mode performance is more favorable. However, to run the PCI interface above 33MHz, asynchronous mode must be used. Please see section 6.1 in the CPC700 User's Manual for information on configuring the PCI Clock.

8. Do the CPC700 UART ports support handshaking type protocols?

No. CPC700 does not support handshaking type protocols; the associated registers are similar to, but not identical to the NS16550 UART.

9. Do unused IIC pins need to be pulled up or down?

The safest course is to tie them high via resistors. If the core is reset, and if the core is not programmed, then you could leave the I/O's floating. If the IIC is programmed or if the core is not reset, then the I/O's need to be tied high.

10. Does CPC700 support the 32-bit data mode on the 6xx/7xx processor?

No, it supports the 64-bit data bus mode only.

11. What is the minimum pulse width on system reset input to the CPC700?

At power up, the SYS_RESET_N must be active and stable until all power rails and input strapping pins are stable and all clock inputs are stable and at their correct operating frequency, so the power on reset length will be system dependent. Reset is asynchronous into the CPC700 and is then synchronized internally. The minimum reset pulse width, assuming stable clocks and power, should be 16 system clock periods.

Once the CPC700 gets the async reset pulse, the reset is latched internally and a counter runs to hold it active internally for the required amount of time (500us).

12. Can the PCI interface of CPC700 be configured via an external PCI agent, or must it be configured exclusively by the processor which is attached to the local 60x bus of the CPC700?

Yes, it can be configured from the PCI bus. The CPC700 has two sets of configuration registers for configuring the PCI interface, handling errors, and reporting status. The PCI configuration registers control PCI related functions and can be accessed from both the 60x and the PCI. See section 4.8 of the CPC700 User's Manual for more details.

13. How does the CPC700 handle lwarx/stwx support (reservation canceling snoops)? There is a pin from the PowerPC 750 for reservation bit indication (RSRV) that seems not to have an equivalent pin on the CPC700. Is this method of communicating reservation just for other processors? Does the CPC700 not need to know about the reservation status bit? Should this pin be left unconnected?

There is not an equivalent reservation pin on the CPC700; this bit is generally used with processors that have L2 cache for snooping purposes. Since bridge chips generally don't have an L2 interface, this pin is not used.

14. What is the difference between the PowerPC 603e-PID6, PowerPC 603e-PID7v and PowerPC 603e-PID7t versions of the PowerPC 6xx/7xx evaluation kit?

The daughter card containing the processor is unique in each case, while the motherboard is identical. The 603e processors vary in frequency, with the 603e-PID6 at 100MHz, the 603ev-PID7v at 166MHz, and the 603ev-PID7t at 200MHz.

15. What is the 6xx/7xx evaluation kit relationship to the CPC700?

The 6xx/7xx Evaluation Kit is a Reference Design, not a product. The purpose of the Reference Design is to enable a customer to bring a PowerPC design to market in a minimum time frame. The Reference Design includes schematics, technical reference, EPLD files, all of the required software, and the design files. The Evaluation Reference Design Kit information is available on the following website: www.chips.ibm.com/products/powerpc/tools.

16. Can the PCI bus run at 41.66 MHz synchronous mode with the processor bus running at 83 MHz?

The CPC700 chip will run at PCI bus frequency of 41.66MHz; however to do so at a system level would require that all devices run at 41.66 MHz. One concern is that there is no pure PCI spec on the IO at the 41.66 MHz frequency; 33 MHz and 66 MHz are specified in the PCI specification.

17. What is the problem with the CPC700 JTAG port? When will it be fixed?

In the CPC700 version dd1.0 and dd1.1 the boundary scan portion of the JTAG port is nonfunctional. The CPC700 does not respond correctly to BSDL commands. This problem has been corrected in the CPC700 revision dd1.2.

Please see the *CPC700 dd1.x Errata List and Advisories* for more details on this and other errata. This errata list is located on the following website: *http://www.chips.ibm.com/products/powerpc/chips* or you may contact *the PowerPC Embedded Applications Support hotline at: ppcsupp@us.ibm.com* to obtain a copy**.**

18. Using an external arbiter, with TT[4] strapped appropriately, at power up can the CPC700 and the external arbiter be allowed to come out of reset at the same time?

Yes, as long as the arbiter is ready at the first PCI access from the CPC700, which will be initialization of other adapters on the PCI bus. The evaluation board external arbiter is ready immediately out of reset.

19. The timing diagrams in the CPC700 Users Manual do not show CKE in a low state except for self-refresh. Are there other conditions besides self-refresh that may cause the CKE to pulse low during normal operation?

There are conditions where CKE may be pulsing low other than in the refresh mode; due to pipelining the CPC700 can have outstanding read request to the SDRAM and be unable to accept the data immediately. In this case, CKE will assert to stall the SDRAM until the CPC700 can receive the data.

This timing condition was not shown in the users manual. It will be included in the update to the user’s manual.

20. Where is the best place to connect high-speed high-bandwidth devices on CPC700? and why?

The natural place to put high-speed high-bandwidth devices in a PowerPC system is on the PCI bus. The PCI bus is specifically designed for these applications, and all high-speed peripherals should be placed on the PCI bus.

The ROM/Peripheral Interface on the CPC700 is not architecturally a fast bus. Since traffic on the bus cannot be controlled by DMA, all transfers from a peripheral to memory have to be done twice. Also, all PI transfers take the place of the CPU to memory transfers; no concurrence is possible.

21. With two CPC700 bridges in a design; how can the local memory be made accessible to the processor if it is attached to the other CPC700 bridge?

The PTM, BAR and PMM registers can be used to do this. Assume CPC700 #1 has 32MB of SDRAM local addresses 0x0000_0000 to 0x0200_0000. You wish to map CPC700 #1's SDRAM to PCI addresses 0x2000_0000-0x2200_0000 and CPC700 #2's SDRAM to PCI addresses 0x3000_0000-0x3200_0000. Here are the register settings to do this:

CPC700-1

These PMM settings will map local PLB (CPU) address 0x8000_0000-0x8200_0000 to PCI address 0x3000_0000-0x3200_0000
PMM0LA (0x8000_0000) - address range (0x8000_0000 to 0x8200_0000) to use on CPC700-1 to access CPC700-2's SDRAM
PMM0PCILA (0x3000_0000) - set to PCI address of CPC700-2's SDRAM
PMM0PCIHA (0x0000_0000)
PMM0MA (0xFE00_0001) - set for 32MB and enable

These PTM/BAR settings will map PCI address 0x2000_0000-0x2200_0000 to local PLB(CPU) address 0x0000_0000-0x0200_0000
PTM1LA (0x0000_0000)
PTM1MS (0xFE00_0001) - set for 32MB and enable
PTM1BAR (0x2000_0008)

CPC700-2

These PMM settings will map local PLB (CPU) address 0x8000_0000-0x8200_0000 to PCI address 0x2000_0000-0x2200_0000
PMM0LA (0x8000_0000) - address range (0x8000_0000 to 0x8200_0000) to use on CPC700-2 to access CPC700-1's SDRAM
PMM0PCILA (0x2000_0000) - set to PCI address of CPC700-1's SDRAM
PMM0PCIHA (0x0000_0000)
PMM0MA (0xFE00_0001) - set for 32MB and enable

These PTM/BAR settings will map PCI address 0x3000_0000-0x3200_0000 to local PLB(CPU) address 0x0000_0000-0x0200_0000
PTM1LA (0x0000_0000)
PTM1MS (0xFE00_0001) - set for 32MB and enable
PTM1BAR (0x3000_0008)

Please see the section "Example Address Map Setup" in the CPC700 User Manual for a detailed example of PMM and PTM/BAR settings. The example and diagram will probably clear up some of the confusion.

22. Can the CPC700 handle Spread Spectrum Clocking? What is the function of the PLL tuning bits?

Yes. The CPC700 can handle Spread Spectrum clocking, but it requires changing the PLL tuning bits, as well as adjusting the PLL synchronization.

The PLL tuning bits, TUNE (5:0), are used to control the PLL loop parameters by modifying the internal gain of the charge pump. This external control allows the PLL to be stable over a wide range of frequencies and multiplication factors. Another benefit of the tuning bits is that they enable fine tuning the PLL to compensate for either high levels of noise or input jitter.

23. We are having a problem getting the processor to see data that was DMA'd by the PCI bus master. We have tried enabling MWI support, but it does not seem to work. I'm confused by the the statement in the CPC700 Users Manual that states "MWI is not supported". Does this mean the CPC700 will not respect the MWI protocol for transactions involving itself, or does it mean the part does not support MWI?

The Memory Write and Invalidate (MWI) command is not generated by the CPC700. Also a memory read line is never generated. All PCI memory writes are performed with memory write. The CPC700 accepts MWI on incoming transactions, but never generates MWI. Also cache coherency is not supported or maintained across the PCI bridge.

24. Will CPC700 support PCI/ISA legacy devices?

Yes, but there are problems with using CPC700 with some ISA bridges. The deadlocks below describe some of the problems.

CPC700 requires the PCI arbiter to be fair, and PCI targets to retry, rather than hang, when they are busy. Some ISA bridges violate one or both of these requirements. Note that these are standard PCI requirements. Older x86 PCI bridges were designed to operate with these ISA bridge violations, however the CPC700 was not.

Case 2-1: the PCI target (ISA bridge) needed retry, rather than hang, to avoid a deadlock. Upon receiving a retry, the CPC700 clears (ARTRY#) the CPU-to-PCI cycle, and thus allows the ISA bridge to complete its cycle. A work-around for this case may be to have a device (PAL) that detects this condition and retry the cpc700 cycle.

Case 2-2: the Winbond PCI arbiter would need to be fair to avoid a deadlock. Thus, in response to the ISA/DMA access being retried, the PCI arbiter should (at some point) grant to the CPC700. The CPC700 could complete its cycle, clearing the path for ISA/DMA access; or if the CPC700 attempts to access the ISA bus, it should be retried, which also clears (ARTRY#) the path for the ISA/DMA access. We are not aware of a work-around when using the Winbond arbiter.

We should point out that the CPC700 should NOT be used with ISA bridges, unless it is known that the ISA bridge does not violate the rules described above.

25. What is the state of BA[1:0] signals for a bank configured as ROM? Can these signals be used as further decode when accessing the ROM or are they tri-stated?

The BA signals are only used for SDRAM accesses. They cannot be used as part of the decode for ROM access. The value driven on the BA signals during ROM access is whatever value the SDRAM controller happens to be driving on those pins (as the controller is the sole owner of those pins). Note the value may be old BA from a previous SDRAM access, or it may be something else. These should be treated as “undefined” or “invalid” for ROM/Peripheral accesses.

26. What are the DMA capabilities of the CPC700?

The CPC700 does not have a DMA controller; an external DMA controller would be needed to have DMA capabilities using the CPC700.

27. Design situation using SDRAM DIMMs. How is it possible that no glue logic is needed when using DIMMs such as IBM's 256MB DIMM; pn IBM13N32644JCA which has 2 physical banks that require TWO chip selects? Assuming that the DIMM is mapped into a single memory bank of the CPC700 then only one chip select (one of five) is available for this particular DIMM. How do I drive the second chip/bank select line (of the DIMM)? The specific question is, given these two DIMMs IBM13N64644HCA and IBM13N32644JCA, what mode should be used and how should you wire DIMMs chip/bank select lines (there are four - pins: 30, 45, 114 and 129 of the DIMM socket)?

The DIMMs identified have two "logical" chip selects, each of which must connect to a separate CPC700 chip select and the corresponding registers for each chip select must be programmed according to the density and addressing mode.

For a DIMM with two physical banks there are two corresponding chip selects, each bank being selected by its logical chip select. One bank is selected by a single chip select (say CPC700 CS1#) wired to S0# (pin 30) and S2# (pin 45). The second bank is selected by a second chip select (say CPC700 CS2#) wired to S1# (pin 114) and S3# (pin 129).

In the case of the 256MB DIMM (2 physical banks each being 128 MBytes), each chip select (CPC700 CS1# and CS2# in the context of this discussion) addresses 128 MBytes of memory.
The MB1SA register must be programmed with the starting address for this 1st bank of 128MB memory within the address map.
The MB1EA register must be programmed with (MB1SA + (128M -1)) to define the ending address.
The DAM_1 (DAM[2:3]) should be set to 2'b01 to select Mode 2 to support the 12x10 - 4 Bank devices used on the DIMM.

The MB2SA register must be programmed with the starting address for this 2nd bank of 128MB memory within the address map.
The MB2EA register must be programmed with (MB2SA + (128M -1)) to define the ending address.
The DAM_2 (DAM[4:5]) should be set to 2'b01 to select Mode 2 to support the 12x10 - 4 Bank devices used on the DIMM.

Likewise, if the same socket were populated with the 512MB DIMM (2 physical banks each being 256 MBytes), each chip select (CPC700 CS1# and CS2# in the context of this discussion) addresses 256 MBytes of memory.
The MB1SA register must be programmed with the starting address for this 1st bank of memory within the address map.
The MB1EA register must be programmed with (MB1SA + (256M -1)) to define the ending address.
The DAM_1 (DAM[2:3]) should be set to 2'b10 to select Mode 3 to support the 13x10 - 4 Bank devices used on the DIMM.

The MB2SA register must be programmed with the starting address for this 2nd bank of memory within the address map.
The MB2EA register must be programmed with (MB2SA + (256M -1)) to define the ending address.

The DAM_2 (DAM[4:5]) should be set to 2'b10 to select Mode 3 to support the 13x10 - 4 Bank devices used on the DIMM

The above is an example of a single DIMM socket and how the corresponding registers would be programmed depending on which of the two part numbers are installed. Obviously, there could be two DIMM sockets wired to CPC700 CS3# and CS4# (analogous to the CS1# and CS2# connections described above). In this case MB3SA, MB3EA, DAM_3, MB4SA, MB4EA, and DAM_4 would need to be programmed for specific memory density and addressing mode for the installed memory.

The above is the standard way that all memory controllers interface, gluelessly, to external memory. There is nothing unique to the CPC700, with respect to other commercially available memory controller/north bridge chips, for this interface.
The key is understanding that a dual bank DIMM of density X MBytes is comprised of two independent physical banks, addressed by individual chip selects, each of density X/2 MBytes.

The starting addresses will be specific to the customer’s application software, etc. Individual banks of memory must not occupy overlapping regions of that application specific address map.

28. With the ECC enabled, we've noticed on partial writes from the processor to SDRAM that CPC700 asserts ARTRY to next memory access until the read-modify-write is completed for the partial write, rather than just holding off TA until the RMW can complete and the next access is performed. Is this behavior configurable?

This is not configurable. In this scenario, we would only retry if the next processor access to memory is a write and RMW has not completed (basically because the post write buffer is allocated until the RMW completes) OR if the next PPC access to memory is a read to the same cache line that the RMW targeted.

29. The CPC700 is configured for internal PCI arbitration at power-up via a strapping resistor. The CPC700 will then arbitrate among six different requesting devices. Twelve pins are configured to handle the six requests and six grants. If the CPC700 is set up to be the arbiter, how does it handle PCI requests from that same device?

CPC700 has two pairs of pins that are either the REQ/GNT for CPC700 when using external ARB mode or REQ0/GNT0 from some other device when using internal ARB mode. See Signal Descriptions, REQ0_N/GNT_N and GNT0_N/REQ_N pins.

30. How does CPC700 control PCI LOCK# signal?

CPC700 does not support PCI LOCK#. There is no way to generate an indivisible read-modify-write to the PCI bus. Also there is no way for a PCI device to generate an indivisible-modify-write to CPC700 local memory; however when using PPC750 with CPC700, the PPC750 can effectively generate a read-modify-write to the CPC700 local memory using the load-reserve/store-conditional instruction.

31. Can the CPC700 reside on a PCI bus other than PCI bus 0?

With respect to configuration cycles originating from the local side (PLB), CPC700 can do one of three things:
1. Access its internal config registers
2. Execute a type 0 config cycle on the PCI bus
3. Execute a type 1 config cycle on the PCI bus

To do internal config accesses, bus number and device number in CONFIG ADDRESS must both be zero.
To do type 0 cycles, bus number must be 0 and device number must be non-zero.
To do type 1 cycles, bus number must be 1 (device number can be anything).

These requirements result in a few limitations on what types of config cycles CPC700 can do:
1. It cannot perform type 1 cycles to bus number 0.
2. It cannot perform type 0 cycles to device number 0.

These are the only limitations, regardless of whether CPC700 is an adapter or a host. As an adapter, CPC700 can reside on any bus number and accept config cycles. As an adapter it could potentially be executing type 0 or type 1 (probably only type 0) cycles with the above limitations.

As a host, CPC700 could (and would) execute type 0 and type 1 cycles with the above limitations.
As long as these limitations don't break a system's requirements.

Also note that even though internal config accesses require a value of 0 in the bus number field of CONFIG ADDRESS, that doesn't mean that CPC700 can't reside on a bus other than 0. These internal accesses are still independent of the actual number of the bus that the CPC700 is attached to.

32. The CPC700 Interrupt Controller supports 12 external interrupts. Let's say two devices share one of these physical interrupt inputs (assuming the devices have open drain outputs). If one of the two devices asserts its interrupt line, software needs to be able to know which of the two devices interrupted. What happens when the interrupt acknowledge cycle occurs with the CPC700? Is there any information returned that indicates which of the two devices sharing the interrupt line asserted its interrupt? If an interrupt vector is returned, where does that vector come from and how does it get there?

The CPC700 can generate a “PCI Interrupt Acknowledge Cycle” on the PCI bus in response to a processor read of a dedicated CPC700 address. This PCI Interrupt Acknowledge Cycle is a read implicitly addressed to the system interrupt controller and is only useful if your system has an interrupt controller on the PCI bus downstream of the CPC700.

Assuming you are using the CPC700 as the system interrupt controller, the CPC700 Universal Interrupt Controller (UIC) can be configured to provide an interrupt vector to the processor via the UIC Vector Register (UICVR). The value in the UICVR is the sum of the Interrupt Vector Base Address and an offset corresponding to the highest priority enabled and active interrupt in the CPC700. In order to use vectors supplied in this way, the service routines must be located 512 bytes apart in the memory map and the Interrupt Vector Base Address must be programmed into the UIC Vector Configuration Register; refer to the CPC700 User’s Manual for details.

In the example above, where a single physical interrupt pin is shared among two devices, no information is available to the CPC700 to indicate which of the two devices are interrupting. Interrupt service software will need to figure this out from the interrupting devices' status registers (i.e., external hardware must supply this information).

33. In a scenario where two or more devices share the same CPC700 interrupt pin, how does the CPC700 handle multiple simultaneous interrupts?

Interrupt software needs to check the status register of each device sharing an interrupt pin to insure an interrupt is not missed. Each interrupt should be programmed as level sensitive. Once an interrupt is active, it must be reset before the interrupt controller can recognize future interrupts. As is typical, interrupts must be cleared at the source and in the register.

34. How does CPC700 design handle the little endian PCI protocol?

CPC700 PCI interface hardware implements the registers in little-endian byte ordering, which means the bit 0 is the MSB and bit 31 is the LSB. Software that runs big endian should do a store (STEW) or byte-reversed store (STWBR) to accomplish little endian effect. The little endian to big endian conversion may also be accomplished through the use of the byte swapping region registers on the CPC700's processor interface.

35. Assuming the PowerPC Processor and CPC700 are in big endian mode, is it true that unless the hardware in an external PCI device will take care of byte swapping PCI data (big endian and little endian byte alignment support), the software will have to always perform a byte swap on the PCI data?

Yes, the PCI interface hardware implements the little endian order; this must be taken into consideration when software is written.

36. How can we get our CPC700 to do burst writes into SDRAM when doing 32-byte 32-byte-aligned writes from the PCI to memory?

32-byte writes are supported, but instead of waiting to get a full 32 bytes of data we do pipeline back to back writes which gather data as soon as possible and pass it to memory. This gives the same effective performance with lower write latency.

37. The CPC700 User's Manual states the buffering between PLB and PCI is a PLB slave 32-byte write post buffer.

In reality they are 32-byte buffers, but can only hold 4 bytes of data.

38. When the PLB slave 32-byte write post buffer is full, and starts to write data in this buffer to the PCI device, how are the data in another buffer transferred into next buffers.

A given buffer has to completely empty on to the PCI bus before it can accept new data into the buffer (talking PLB to PCI).

39. How many PCI devices can the CPC700 support running at 66MHz?

The CPC700 can definitely support 1 PCI device at 66MHz, and depending on the system topology it may support 2 PCI devices at 66MHz. The closer the PCI devices are to the CPC700, the greater the chances for supporting 2 PCI devices. System analysis and simulation should be done to determine if 2 PCI devices can be supported.

40. This question concerns the CPC700 when talking to the IIC bus; the data write is a FIFO of four bytes. Which clock is used to clock this data through. Would it be the 33MHZ system clock or the divided clock of 100khz?

The system clock moves the data through the FIFO (and clocks the core). The clock divide is used for operating the IIC bus.

41. This question is related to using the CPC700 support chip with the 603e and the SDRAM controller. Does CPC700 support the use of the two PowerPC instructions: eciwx and ecowx? These instructions are unique in accessing memory. If these two instructions are not supported by the CPC700 but are valid instructions of the 603e cpu, how do I prevent a compile from using these instruction?

The eciwx & ecowx instructions are recognized by the CPC700 but are not supported transactions; eciwx & ecowx will be acknowledged with AACK & TA but MCP will be asserted as well if it is enabled, and, the eciwx will return only 1's on the 60x bus. I am not of any bridge chips that use these instructions. It is a condition of the complier to prevent these instructions from being used; the CPC700 has no control over this.

42. What are the PACR, PEAR, and PESR Registers?

The PLB Abiter Control Register (PACR) is a 32-bit register which controls the arbitration priority scheme, consisting of PLB priority mode and PLB priority order. The contents of the PACR can be accessed by using the move from device control register **(mfdcr)** and move to device control register **(mtdcr)** instruction.

The PACR can be accessed with CPU_dcrAddr(6:9)=0x7. At reset, all bits in the PARC are loaded with zeroes and Fixed Priority Order ‘000’ is detected.

The Error Address Register (PEAR) is a 32-bit red-only register which contains the address of the access where the bus timeout error occurred. As in the case of the PESR master error status fields, the Pear can be locked by the master having its Mn_lockErr signal asserted for the transfer.Once locked, the Pear cannot be updated if any subsequent error occurs until all FLCK bits in the PESR are cleared.

The contents of the PEAR can be accessed by using the move from device control register **(mfdcr)** and move to device control register **(mtdcr)** instruction.

The PEAR can be accessed with CPU_dcrAddr(6:9)=0x6. At reset, all bits in the PEAR is not affected by Reset.

The PLB Error Status Register (PESR) is a 32-bit register whose bits identify timeout errors on PLB bus transfers, the master initiating the transfer, and the type of transfer. The contents of the PESR can be loaded into a general purpose register via the move from device control register **(mfdcr)** instruction.

Each master error status field can be locked by the master having its MN_lockErr signal asserted for the transfer in which the timeout is encountered. Once locked, the master error status field cannot be updated if any subsequent error occurs until the FLCK bit is cleared via the move to device control register **(mtdcr)** instruction. To clear a bit in the PESR, a “1” must be written to the bit. Writing a “0” to any bit in the PESR will not affect the status of the bit.

The PESR can be accessed with CPU_dcrAddr(6:9)=0x4 (read/clear) and CPU_dcrAddr(6:9)=0x5. At reset, all bits in the PESR are loaded with zeroes.