For:billk

Printed on:Wed, Jun 2, 1999 17:41:12

From book:VR_ERS

Document:Contents

Last saved on:Wed, Jun 2, 1999 17:39:25

Document:Overview

Last saved on:Tue, Jun 1, 1999 13:09:09

Document:Pipeline

Last saved on:Tue, Jun 1, 1999 13:10:12

Document:TLB

Last saved on:Tue, Jun 1, 1999 14:05:02

Document:Floating Point

Last saved on:Tue, Jun 1, 1999 14:06:03

Document:Instruction Cache

Last saved on:Tue, Jun 1, 1999 14:06:51

Document:Data Cache

Last saved on:Tue, Jun 1, 1999 14:07:36

Document:Memory & SLC

Last saved on:Tue, Jun 1, 1999 14:08:22

Document:I/O

Last saved on:Tue, Jun 1, 1999 14:12:43

Document:Fault Tolerance

Last saved on:Tue, Jun 1, 1999 14:13:43

Document:Diagnose

Last saved on:Tue, Jun 1, 1999 14:17:25

(...)

PA7300LC ERS

Hewlett–Packard

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT−PACKARD MAKES NO WARANTEE OF ANY KIND WITH REGARD TO THE MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Hewlett−packard shall not be liable for errors contained herein of for incidental of consequential damages in connection with furnishing, performance, or use of this material.

Hewlett−Packard assumes no responsibility for the reliability of its software on equipment that is not furnished by Hewlett−Packard.

This document contains proprietary information that is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of Hewlett−Packard Company.

Contents

1. Overview

1.1 Introduction

The PA7300LC PA–RISC processor is a continuation of the price/performance optimized processor line begun with the PA7100LC. Like the PA7100LC, it is a superscalar processor with an integer execution unit, an integer/memory execution unit and a floating point execution unit. Also like the PA7100LC, it incorporates an integrated high performance memory and I/O controller. However, the PA7300LC is the first PA–RISC processor to include large, split, on–chip, Level 1 (L1) caches. It also includes an interface for an optional off–chip Level 2 (L2) cache. The PA7300LC features extensive support for initial turn–on and debugging.

1.1.1 Superscalar Execution Units

Every cycle, the PA7300LC is capable of dispatching instructions to two of three execution units. The first execution unit can execute all integer instructions execpt for memory reference, system control, and memory management instructions. The second execution unit handles memory reference (integer and floating point), system control and memory management instructions plus non–nullifying integer ALU instructions. The third execution unit handles all non–memory floating point instructions. Not all combinations of pairs of instructions may be executed simultaneously, see the chapter on performance coding. Under certain circumstances, a pair of integer load or store word instructions may be executed together. The PA7300LC supports the multimedia halfword arithmetic instructions first implemented on the PA7100LC.

1.1.2 Integrated Caches and TLB

The PA7300LC contains 128Kbytes of integrated L1 caches. The L1 caches are split into a 64 Kbyte instruction cache and a 64 Kbyte data cache. Each cache is 2–way set associative and has a cache line size of 32 bytes. The caches have a 64 bit data width to the execution units, and a 256 bit data width to memory. Both reads and writes of the caches are accomplished in a single cycle. The instruction cache incorporates a one line prefetch buffer while the data cache incorporates a two line copyin buffer. The instruction cache can issue three simultaneous read requests while the data cache can issue two simultaneous reads. The PA7300LC processor supports both big and little endian modes. If the PSW E–bit is set, instructions will be swapped on their way into the instruction cache while data is swapped between the data cache and the execution units. Uncached memory pages are supported via the TLB U–bit. Block copy store hints and semaphore hints are supported at all privilege levels.

The PA7300LC has a 96 entry unified translation lookaside buffer or TLB. Each of the entries maps one 4K PA–RISC instruction or data page. In addition, the TLB contains 8 block TLB entries that are capable of mapping large, contiguous blocks of memory. The block TLB entries are only accessable through diagnose instructions. In addition to the unified TLB, the PA7300LC also has a 4 entry instruction lookaside buffer or ILAB. The ILAB contains four of the most recently used instruction page translations, including the translation corresponding to the program counter. Entries corresponding to the next virtually sequential page will be opportunistically moved from the unified TLB into the ILAB in anticipation of their reference. The PA7300LC also implements several ‘‘fast’’ TLB insertion instructions to speed TLB miss trap handler code.

1.1.3 Memory Controller

The integrated memory controller directly connects to a 64 or 128 bit wide DRAM array. It provides a high bandwidth connection between the caches and memory and between the I/O bus and memory. The memory controller also directly supports an optional off–chip, L2 unified cache. The L2 cache and DRAM arrays share data lines to reduce pin count. The L2 cache is write–through, direct mapped, physically indexed, and physically tagged.

1.1.4 I/O Controller

The integrated I/O controller connects to the GSC bus introduced on the PA7100LC. It handles programmatic I/O reads and writes as well as GSC mastered DMA transfers to and from the memory controller. Certain ''accelerated'' I/O writes are capable of being processed in parallel with memory transfers to and from the L1 caches.

1.1.5 Debug Unit

The PA7300LC implements an extension of the debug features found on the PA7100LC. Intended solely for functional and electrical failure analysis on prototype PA7300LC microprocessors and systems, debug makes visible selected internal processor signals on both pins multiplexed with the GSC interface and on dedicated debug pins (some of which require a special package bonding option for use). The Sample On the Fly (SOF) technique first implemented on the PA7100LC is also available, providing a single cycle ”snapshot” of the chip’s scanpaths. Despite being limited to one clock cycle, each SOF trace provides a very valuable look at the internal chip state.

1.1.6 Architecture

The PA7300LC implements the PA–RISC architecture revision 1.1 (3rd edition). The PA7300LC does not implement any performance monitor instructions, nor does it implement the debug SFU present on Tornado. Support for an integer multiplier SFU was dropped halfway through the design cycle. The PA7300LC also does not implement the Load Coherence Index (LCI) or Synchronize DMA (SYNCDMA) instructions.

1.2 Differences from the PA7100LC

The PA7300LC is a design leveraged from its predecessor, the PA7100LC. Many parts of the PA7300LC were directly borrowed from the PA7100LC, while other parts were completely redesigned. The following sections describe the differences in microarchitecture between the two chips. Where appropriate, changes visible to software are pointed out.

1.2.1 TLB Organization

The unified TLB was increased in size from 64 to 96 entries. The number of block entries remains the same at eight. Because of the 32 additional page entries, the layout of the ''TLB'' diagnose register was changed slightly. The hardware TLB miss handler remains the same. Note that CR28 is still undefined for taken ITLB miss faults (i.e. defined only for DTLB miss faults).

The ILAB was grown from a single entry to four entries. Two diagnose registers (''ILAB_VPN'' and ''ILAB_RPN'') and two diagnose instructions (''ILAB_READ'' and ''ILAB_WRITE'') were added for configuration and test of the ILAB.

1.2.2 Cache Organization

Instead of a small on chip instruction cache and a combined off chip cache, the PA7300LC has an L1 instruction cache and a L1 data cache both integrated on chip. Each cache is 64 Kilobytes and two–way set associative. An optional L2 unified cache is supported (described below). Hashing has been eliminated for both the instruction and data caches. The instruction prefetch buffer was moved from the memory controller to the L1 instruction cache. This allows prefetch hits to incur no penalty. Because the data cache is now integrated, write operations take only one cycle. Thus, the ‘‘store tail’’ penalty has been eliminated. The data cache can have two outstanding misses pending without stalling the CPU, as long as there is only one load miss. Traps are not suppressed on prefetches (loads to GR0). Unlike the PA7100LC (and like the PA7100), block copy store hints are supported for all privilege levels. Unlike the PA7100, the cache line will be zeroed, even for privilege level 0. Uncacheable pages are still supported. **Note:** PA7100LC did not require the U–bit to be set on TLB entries for I/O pages, but the PA7300LC does (this was always an architectural requirement).

Like the PA7100LC, the instruction and data caches are parity protected and will cause an HPMC if parity error signalling is enabled. Instruction cache parity errors may be recoverable while data cache parity errors are unrecoverable.

The programming model for diagnostic access to the instruction and data caches is completely different from the PA7100LC (see the Instruction Cache, Data Cache and Register Definitions chapters). Diagnose access reuses the hardware features for the BIST (built–in self–test) engines incorporated into the L1 caches. Diagnose operations are largely consistent between the instruction and data caches. Most of the registers used for BIST/diagnose access to the L1 caches are located in I/O space. This means that normal load and store instructions are used to read and write those registers (as opposed to the special move–to and move–from diagnose instructions). An additional restriction placed on diagnose access of the data cache is that the store queue must be flushed prior to placing the data cache into test mode. The store queue is most easily flushed by executing two writes to I/O space.

Because the PA7300LC implements a two–way set associative L1 data cache, the FDCE (flush data cache entry) instruction works differently than on the PA7100LC. Because there could be two valid and dirty lines at any data cache index, the FDCE instruction must be executed twice to each L1 data cache index to assure that the L1 data cache has been completely flushed. The FDC instruction behaves the same as prior PA–RISC implementations.

1.2.3 Memory

The memory controller incorporates several new features to increase performance. The main memory (DRAM) interface was widened to allow either 64 or 128 bits of error corrected data, instead of just 64 bits on the PA7100LC. Also new is an optional L2 unified cache interface, sharing the DRAM data pads. This L2 cache is not an ‘‘architected’’ cache in that the cache flush instructions (FIC and FDC) do not affect it. The memory controller can queue up more memory read and write transactions than the PA7100LC. It also allows reads (copyins) to pass writes (copyouts) in the queue, reducing cache miss latency. The programming model for the memory controller has changed from the PA7100LC. New registers have been added to configure the L2 cache, many registers have been reorganized and several registers that were formerly located in I/O space are now diagnose registers accessable only with the move–to and move–from diagnose instructions.

1.2.4 I/O

The PA7300LC supports a wider range of ratios between GSC and processor frequencies than the PA7100LC. The GSC protocol was enhanced to allow 1.5x transfer rates using the ‘‘WRITEV’’ transaction, which the PA7300LC supports. The memory controller now has a separate queue for I/O stores. Because this queue can operate independantly of the memory interface I/O write (e.g. graphics) performance can be markedly improved. In order to prevent architectural ordering violations, only writes to a subset of I/O space can be accelerated in this manner. A new diagnose register was added to configure the address range enabled for acceleration.

1.2.5 Configuration

The following is a table of the PA7100LC and PA7300LC processor features that can be configured by software and how they differ between the two (not including diagnose and debug support):

Feature	PA7100LC Location	PA7300LC Location	Difference between PA7100LC and PA7300LC
L2 Icache HPMC disable	DR0.L2IHPMC_DIS	SLTCV.chktp SLTCV.sledcen	Replaced with L2 unified cache tag parity checking enable and data error correction/ detection enable.
L2 Dcache HPMC disable	DR0.L2DHPMC_DIS	DR0.L1DHPMC_DIS	Replaced with L1 data cache HPMC disable.
L1 Icache HPMC disable	DR0.L1IHPMC_DIS	DR0.L1IHPMC_DIS	None.
Dcache ‘‘safe–mode’’ enable		DR0.DC_SAFE	Not implemented on the PA7100LC. It effectively causes a SYNC to be executed after every instruction on the PA7300LC.
Icache streaming enable	DR0.ISTRM_EN	DR0.ISTRM_EN	None.
Dual issue disable	DR0.DUAL_DIS	DR0.DUAL_DIS	None.
Default endian	DR0.ENDIAN	DR0.ENDIAN	None.
Stall–on–use enable	DR0.SOU_EN	DR0.SOU_EN	Unlike the PA7100LC, this only affects load misses on the PA7300LC.
Store hints enable	DR0.SHINT_EN	DR0.SHINT_EN	Affects all privilege levels on the PA7300LC.
Icache prefetch enable	DR0.IPREF_EN	DR0.IPREF_EN	None.
Dcache hashing enable	DR0.DHASH_EN		Not supported on the PA7300LC.
Icache hashing enable	DR0.IHASH_EN		Not supported on the PA7300LC.
L1 Icache enable	DR0.L1ICACHE_EN	DR0.L1ICACHE_EN	The L1 Icache can only be disabled for test purposes only on the PA7300LC.
Undefined minor opcode detection enable		DR0.RMIN_EN DR0.LMIN_EN	Not implemented on the PA7100LC. If enabled on the PA7300LC, undefined minor opcodes will cause an illegal instruction trap.
ILAB prefetching enable		DR7.ILPRE_ENH	Not implemented on the PA7100LC. If enabled on the PA7300LC, the translation for the next sequential virtual page will be opportunistically moved into the ILAB.
L1 Dcache test mode enable		DR11.TEST_MODEH	Not implemented on the PA7100LC. Disables the L1 Dcache and places it into test mode.
HTLB Handler Base	DR24.HTLB_BASE	DR24.HTLB_BASE	None.
Dcache size configuration	DR24.DCACHE_SIZE		Not supported on the PA7300LC. The L1 Dcache is not configurable in size.
HTLB Handler Mask	DR25.HTLB_MASK	DR25.HTLB_MASK	None.

FP Delay	DR25.FP_DELAY	DR25.FP_DELAY	None.
HTLB Control	DR25.HTLB_CNTL	DR25.HTLB_CNTL	None.

1.2.6 Diagnose Operations

The number of cpu diagnose registers has increased from 9 to 35. Because this is more registers than can be specified with a 5–bit register address, a second ‘‘page’’ of diagnose registers was added. Two new diagnose instructions were added to set the page that subsequent move–from and move–to diagnose instructions will access. As mentioned earlier, the PA7100LC diagnose instructions used to access the instruction and data caches have been replace with just two instructions that initiate diagnose/BIST operations. Note that the ‘‘RDTLB’’ diagnose instruction no longer exists. In addition, two new instructions were added to provide diagnose access to the 4–entry ILAB.

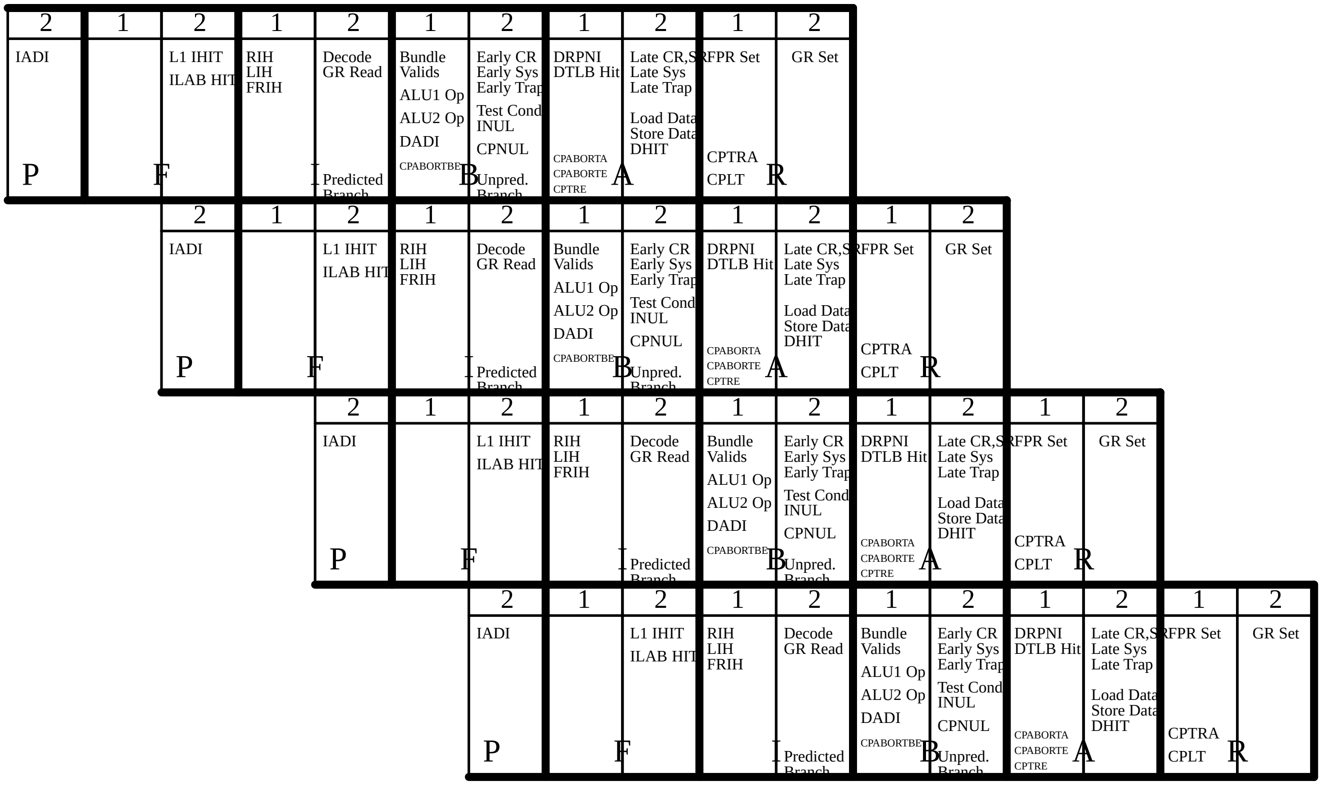
1.2.7 Miscellaneous

Unlike the PA7100LC, the PA7300LC has a feature that allows instructions with undefined minor opcodes to cause an illegal instruction trap. This may be useful in preventing undefined instructions from exposing hardware problems. The minor opcode detection can be enabled on the RIH and LIH instruction buses independently. Minor opcode checking is not supported for floating point instructions.

2. Pipeline

2.1 Introduction

The PA7300LC pipeline is very similar to the PA7100LC pipeline, the main differences being in the ILAB and data cache stores. The following diagram shows the pipeline:



2.2 Pipeline Details

This section will explain what happens in each pipe stage. Note that many of the events described below are really tied to the prior pipe stage plus one cycle and may not occur during the stage given if there is a pipeline stall. For instance, the instructions busses (RIH, LIH and FRIH) are actually driven on CK1 F+1, and load and store data are actually driven on CK2 B+1. All of the events of the R–stage are actually A+1 events. Again, this only makes a difference if there is a stall.

2.2.1 P–Stage

On CK2: a new instruction fetch address is generated from the program counter, branch adder or the PC Queue. The address bits are sent to the L1 ICache and latched on CK2 falling.

2.2.2 F–Stage

On CK1: The L1 ICache and ILAB is accessed.

On CK2: The L1 ICache completes its access, returns a doubleword, and signals hit or miss. The ILAB completes its access and signals hit or miss.

2.2.3 I–Stage

On CK1: Instructions are selected from the data the ICache delivers for the separate integer (RIH), load/store (LIH), and floating point (FRIH) instruction busses. Branch address calculation for predicted branches begins.

On CK2: The instructions are latched by all instruction decode blocks and instruction decode begins. General register operands (which are potentially bypassed from older in–flight instructions) and immediate operands are valid by the end of CK2. The nullify indications (from the previous instruction), and the CPU interlock indication (ILOCK) are latched by control logic. Branch address calculation for branches which are predicted taken is completed.

2.2.4 B–Stage

On CK1: The ALU and SMU generate their results. The data address (DADI) is calculated by the ALU. This address is latched by the L1 DCache and the UTLB on falling CK1. Branch address calculation for branches which are predicted untaked begins. Early trap qualifiers are latched here. ‘‘Early’’ traps are the traps numbered 2–7, 10, 11, 23–25 and sometimes 1 (HPMC). The remaining traps are classified as ‘‘Late’’ traps. The CPU takes advantage of the fact that Early traps are known before the other traps (generally to prevent architectural state other than general registers and memory from being modified).

On CK2: The L1 DCache and UTLB begin a data access. The L1 DCache tag array is read for both loads and stores, while the L1 DCache data and dirty array is read for loads and written (from the front of the store queue) for stores. The Early traps are ORed into one signal. The branch and nullify conditions are known here. The branch condition is used to select between the sequential and branch addresses on this phase. Branch address calculation for branches which are predicted untaken is completed. Control register 11 is set here for MTCTL instructions. This is done to avoid an interlock for MTCTL X,11 followed by an instruction which uses CR11. The system mask is set here for SSM,RSM and MTSM instructions. This is done to avoid an interlock after these instructions. The PSW Carry/Borrow bits and the PSW V bit are also set here to avoid an interlock. MFCTL data from all CRs except 17, 19, and 20 is valid on this phase early enough to be bypassed (without an interlock) to the next instruction. The recovery counter is decremented for non–nullifed instructions if the PSW R bit is set.

2.2.5 A–Stage

On CK1: The data real page number is driven from the UTLB to the L1 DCache hit compare logic. The entire physical address is sent to the MIOC in anticipation of a L1 DCache miss. Qualifiers from all traps except DTLB Protection traps and Assist exception traps are valid here.

On CK2: On loads, data from the L1 DCache is driven to the integer or floating point units; on stores, data is driven from the integer or floating point units to the store queue in the L1 DCache datapath (both via FCDIH). L1 DCache hit is valid. All traps except the Assist exception traps are ORed into one signal. MFCTL data for CR 17, 19 and 20 is valid here. Data for MFSP, LDSID, LPA, PROBE and MFCPU_x is also valid here.

2.2.6 R–Stage (A+1)

On CK1: The Assist exception trap signal is valid. The MIOC is signalled to begin a transaction if there was a L1 DCache miss.

On CK2: The general registers are set here after all traps, misses, and nullifies have been resolved.

2.3 Pipeline Stalls

Certain conditions cause the pipeline to ‘‘hang’’ or stall. While the pipeline is frozen, the condition which caused the exception is either serviced or cleared (e.g. a trap will cause a miss to be ignored). May stall conditions can occur simultaneously and there are usually serviced sequentially in a fixed order. However, in many cases there is some overlap in the servicing of the different stall conditions. The order that these are serviced is based on the ‘‘priority’’ of the signal indicating the stall condition. At the time that a stall condition can be serviced, the hightest priority stall condition is the next condition to be serviced. In this section, the different types of stall conditions are explained in order of their priority.

The pipeline stalls between CK1 and CK2. The pipeline advances by states which start with CK2 and end with CK1. Unfortunately, the pipe stages labels (PFIBAR) are associated with CK1–CK2 states instead of CK2–CK1 states. Although there is no good reason for this, the pipe stage labels have been around too long to change them.

The CPU control logic guarantees that addresses and operands are valid on the hang state immediately preceding a pipeline step. The I and D addresses and operands from the last pipeline step is repeated during the freeze states by default unless there is a need to change them. This allows the pipeline to be stalled easily at any time

2.3.1 Reset

The MIOC looks at the RESETL pad to determine when to drive PRESETDL to reset the main control state machines. The MIOC will also signal reset when receiving a broadcast reset transaction on the GSC bus. All state machine are reset. All PSW bits except the M bit is reset. The M bit is set. The CPU begins fetching instructions from address 0xf0000004. This address is in PDC space.

2.3.2 Coprocessor Interlock

See the Floating Point chapter in this document.

2.3.3 DTLB Miss (Hardware Handler)

When a DTLB miss occurs, the CPU will attempt to handle the TLB miss in hardware. The CPU will read the PDIR entry (or a software ‘‘cache’’ of entries) and insert the entry into the UTLB if it hits. More details on the hardware TLB handling mechanism are given in the TLB chapter. The DTLB Miss stall is a higher priority than an Assist Exception trap. This stall will occur after the A–step (CK1/R) of stores and non–interlocked loads. It will occur after the interlock–‘‘step’’ of interlocked loads.

2.3.4 Interruptions (Traps)

An interruption occurs when any of the interruption conditionis true. THe cause of interruptions 2–4 and 6–28 are architected. An interruption stalls the pipeline after CK1/R.

Interruption 1 (HPMC) occurs fo all cache parity errors and double bit memory errors in addition to HPMC conditions on the GSC bus. Also, a transfer–of–control (TOC) will cause the CPU to vector to the HPMC address. See the ‘‘Fault

Tolerance'' chapter of this ERS for a discussion of when HPMCs are recoverable. HPMC and TOC conditions vector to address 0xf0000000 in PDC space.

Interruption 5 (LPMC) occurs for corrected single bit memory (DRAM and SLC) errors and certain GSC errors. LPMC vectors directly to IVA + 5*32.

TOCs are completely decoupled from the PSW M bit, they are not qualified by the M bit and do not set the M bit when they occur. However, TOCs are masked by a HVERSION (implementation specific) bit. This bit is **set** when 1) a TOC is taken, 2) a Broadcast reset occurs, or 3) when a special diagnose instruction (TOC_DIS) is executed to set this bit. This bit is **reset** when a special diagnose instruction (TOC_EN) is executed to reset this bit. PDC must reset this bit after it clears the MIOC TOC bit but before it enters the OS_TOC handler. After broadcast reset, PDC should leave this bit set (TOC disabled) until the hard or soft boot has completed (at the software IPL interface). It is an architectural requirement that TOCs should not interrupt Hard or Soft boot. PDC may want to set this bit if it discovers a situation where TOCs shold be masked. One such situation may be in an HPMC handler when it discovers and invalid checksum for the IVA.

Only the Group 3 traps are inhibited if the instruction is nullified.

The Group 4 traps are implemented as ''Taken–Before'' traps rather than ''Taken–After'' traps. That is, transfer traps occur after the privilege change and the Taken Branch trap occurs on the delay slot instruction. However, the trapped instruction does not execute (backout is required) and these traps are the hishest priority interruptions (even higher than HPMC). If HPMC was higher priority than the Group 4 traps, software could not recover from an HPMC because it would not know whether a Group 4 interruption was lost (i.e. signalled but never taken) due to taking the HPMC.

The FIC instruction uses the UTLB for translation and can cause a Non–Access DTLB miss trap.

The PA7300LC mantains a set of backup GRs or ''shadow'' registers for the purpose of reducing the DTLB and ITLB miss penalty. At the time of a trap (any trap) the values of GRs 1,8,9,16,17,24 and 25 are copied into their corresponding backup registers. A special instruction, called RFIR, can be used to recover these values aftwer trap servicein has been completed. This feature allows a trap handler to use those GRs without the overhead of saving and restoring them. This feature has been architected into the PA–RISC 1.1 architecture.

All interruptions except assist exception trap require three pipeline stall cycles. Assist exception traps on FP loads, FP stores, or FTEST instructions require five pipeline stall cycles. Assist exception trap is special only because it is valid at the PCU at a later time than the other traps. In the hand state the trap conditions are recalculated.

Effects of an interruption:

- Wait for pending instruction and data cache misses to finish.
- Push PSW to IPSW.
- Clear PSW. Set M bit if HPMC, reset otherwise.
- Backup ''early'' SAR and recovery counter (since these are set early). The PSW bits where are set in CK2/B never get set into the ''real'' PSW. These bits are also cleared here.
- Prevent the execution of the trapped instruction. This entails backing out of a store, inhibiting register sets, etc.
- Flush the pipeline. Namely, the next two instructions which have entered the pipeline (in the I–stage and B–stage) must not execute.
- Start fetching instructions from (IVA + 32*(trap_class)) if not HPMC, else fetch from 0xf0000000.
- Set GR ''shadow registers'' from GRs 1, 8, 9, 16, 17, 24 and 25.

- Ignore the following stall conditions: CPU interlock, inserts, TLB purges, diagnose, FIC instructions, flushes, PDC, load and clear instructions, DCache miss, ICache miss, branch and RFI.

2.3.5 Data Cache Hang

The CPU will stall when a data cache miss is signalled for loads, stores, and load and clears that miss the L1 DCache (including those in I/O space, except for some I/O stores). It will also unconditionally stall for FDC, FDCE and PDC instructions for at least one cycle. See the Coding Hints chapter for more details on data cache misses. The pipeline will stall after CK1/R.

2.3.6 Incorrect Branch Prediction / RFI

The pipeline requires one extra cycle for the fetching of an incorrectly predicted branch target. An additional cycle is required for external branches, BVs, and RFIs.

The following branches cannot bre predicted because the address or privilege calculation is dependant upon a general register: GATE, BLR, BV, BE, BLE.

The following branches are always predicted untaken: Forware PC–relative branches. These include COMBT, COMBF, COMIBT, COMIBF, MOVIB, MOVB, BB, BVB, ADDBT, ADDBF, ADDIBT, and ADDIBF.

Backward PC–relative branches are usually predicted taken. Take are predicted untaken only in the following cases: 1) they are in the delay slot of another branch or ICache or memory reference instruction (opcode = 000x0x), or 2) the branch was the target of an RFI.

Note that the nullify indication has no effect on determining whether a branch is predicted or not. E.g. a nullified backward PC–relative branch is still predicted as taken.

The RFI instruction causes this stall condition to be true twice. Once to fetch the front of the PCO/PCS queues and once to fetch the rear of the PCO/PCS queues. The penalty is two cycles fro each RFI target.

Incorrect branch prediction stalls the pipeline after CK1/A. RFI stalls the pipeline after CK1/A and CK1/R.

Effects of incorrect branch prediction/RFI:

- Continue I–fetching (for one cycle). The address of this instruction is the target address for a taken, unpredicted branch, or PCOffset + 8 for an untaken, predicted branch.
- Pop the PSW on the first RFI hang state (if the RFI does not trap). The instruction following the RFI (which has entered the pipeline) is not executed and never traps.
- Set SR[0] if BLE. This avoids an interlock for BLE.
- Update the ILAB for each external branch and RFI target.

2.3.7 Store Interlocks

A store interlock occurs when the data portion of the L1 DCache is being written at the same time the instruction in the B–stage of the pipeline wishes to read it. This is caused by subword store instructions and by load and clear instructions, both of which advance (write the front entry of) the store queue and also need to read the data cache (as part of a read–modify–write process). In these cases, the store queue write has higher priority and proceeds. The read is delayed by one cycle. The pipeline will stall for one cycle after CK1/A.

Subword store instructions are not subject to this one cycle stall penalty if they are not followed by an instruction bundle containing a memory reference. In this case, no stall occurs and the cache read is delayed until CK2/A of the subword store.

2.3.8 TLB Insert/Purge, Diagnose

The following instructions stall the pipeline after CK1/A: IITLBA, IITLBP, IDTLBA, IDTLBP, PITLB, PITLBE, PDTLB, PDTLBE, and all diagnose instructions except MTDIAG/MFDIAG. These operations are not performed if the instruction traps or is nullified. This stall condition is also entered for the implementation specific instructions IITLBPF and IDTLBPF (Fast TLB Insert Protection). See the PA7300LC Specific Instructions chapter. These instructions need to stall the pipeline because they are multi–state operations.

2.3.9 CPU Interlock (load–use, other)

The CPU has two types of interlocks. The first is the Load–Use interlock. This occurs when a GR operand of an instruction directly after a ‘‘load’’ is the same general register the ‘‘load’’ is setting. ‘‘Load’’ in the previous sentence actually includes the following instructions: All loads, all load and clears, MFCTL (CR17, CR19 or CR20), MFSP, LDSID, LPA, PROBE and MFCPU_T. In addition, the Load–Use interlock is actually signalled any time either of the five bit fields 6:10 or 11:15 in the instruction bundle after the ‘‘load’’ matches the ‘‘load’’ target number, even if these fields indicate immediate data.

The second type of CPU interlock is due to MTCLT (except to CR11), MTSP, and MTCPU instructions. These instructions always cause an interlock regardless of the next instruction. These interlocks simplify the control.

The CPU Interlock occurs a maximum of once per instructions and stalls the pipeline after CK1/B. Effects of CPU Interlock:

- Latch ‘‘Load’’ data, set GRs for previous instruction, advance target number and result data pipelines, advance offset and space pipelines. These operations are similar to what occurs on a normal pipe advance, and is sometimes referred to as an ‘‘interlock step’’.
- Set SRs, CRs, and diagnose registers if MTSP, MTCTL or MTCPU respectively.
- Handle TLB and data cache misses if the load or load and clear misses the TLB and/or data cache.

2.3.10 Stall–on–Use Stall

This stall is similar to a Load–Use interlock stall, in that an instruction is attempting to read a register before it has been written by a prior load or load and clear. In this case, however, the load or load and clear encountered a data cache miss. A performance optimization (called appropriately enough, ‘‘stall–on–use’’) allows the load instruction to be retired before the data cache miss completes by recording which register was the target of the load. If an instruction attempts to read that register before the critical data arrives from the MIOC and written to that register, then the pipeline must stall until the critical data does arrive. The pipeline will stall after CK1/B.

2.3.11 ILAB Miss

If the virtual page number address of the instruction being fetched is not in the four–entry ILAB (ITLB lookaside buffer), a pipeline stall will occur while the ILAB is updated from the UTLB. The stall usually takes two cycles, but under certain condtions the stall will only take one cycle. The conditions are when the ILAB miss is due to a branch to a new page, and the branch is not bundled with a memory reference instruction. The pipeline will stall after CK1/I.

2.3.12 ITLB Miss (Hardware Handler)

When a needed instruction virtual address does not exist in etiher the ILAB or the UTLB, the CPU will attempt to handle the ITLB Miss in hardware. The CPU will read the PDIR entry (or a software ‘‘cache’’ of entries) and inser the entry into the UTLB if it hits. If the hardware handler hits cache and the PDIR then the ITLB miss incurs only a 11 cycle penalty. More details on the hardware TLB handling mechanism are given in the TLB chapter. The pipeline will stall after CK1/I.

2.3.13 Instruction Cache Miss

When an instruction fetch does not hit the L1 ICache, the CPU will stall until the critical data arrives from the MIOC. The CPU will ‘‘stream’’, or execute on the fly, the instructions as they arrive. See the Coding Hints chapter for more information. The pipeline will stall after CK1/I.

3. TLB

3.1 General Overview

The PA7300LC CPU is equipped with a unified instruction/data tlb. The TLB is organized as 96 fully associative page entries. Each page entry maps 4k bytes of virtual space. In addition to the 96 page entries, the tlb contains 8 Block entries. Each block entry is capable of mapping a contiguous virtual address space ranging in size from 128 pages (smallest) to 16K pages (largest). Some size and alignment restrictions apply to block entries.

In addition to the Unified TLB, the PA7300LC contains a four–entry Instruction Lookaside buffer (LAB). The LAB generally contains translations for instruction pages that were recently accessed or about to be accessed. If a hit is encountered in the LAB the UTLB is free to perform a Data translation without incurring a penalty.

3.1.1 TLB organization

The TLB produces real addresses from virtual addresses whenever a memory or IO transaction or instruction fetch occurs in virtual mode. TLB translations are accessed through a 36 bit virtual page number (VPN) computed as follows:

VPN[0:35] = cat(SID[0:15],Page_offset[0:19]);

The translation process produces either a 20 bit real page number (RPN) or any of the TLB traps specified by the architecture.

To facilitate trap generation, the four architected Protection ID (PID) registers (CRs 8,9,12 and 13) are visible to the TLBs.

The TLB contains a 16 bit Diagnose Control register which assists a variety of test and initialization functions.

3.1.1.1 Page entries

Each page entry stores and compares a 36 bit VPN which identifies a single 4k page. A translation is stored for each entry which contains the 20 bit RPN, the architected PID (15 bits), Access rights (7 bits) and E–flag (valid bit). Translations for data accesses additionally contain the architected T, D, B, and U (uncached page) flags.

Each page entry can be individually locked out (ie. its hit comparator disabled) or locked in (ie. excluded from consideration for replacement).

Insertion of a page entry is accomplished through the use of the architected IITLBA,IDTLBA instructions. The TLB contains hardware which automatically selects an appropriate entry to be the target for these instructions. The protection fields (PID, AR and flags) for the target entry are cleared (zeroed) by these instructions. Insertion of protection is accomplished through the use of the architected IITLBP or IDTLBP instructions. Purging of translations is accomplished through the use of the architected PITLB or PDTLB instructions. Execution of a PITLBE or PDTLBE instruction serves to invalidate ALL page entries in a single instruction. The PA7300LC also has non–architected, faster–executing insert address/protection instructions. See the last section of this chapter.

3.1.1.2 Block entries

Translations via block entries differ from those via the normal page entries in two respects:

- Each of these entries maps a minimum of 128 pages, thus the low seven bits of the VPN are excluded from hit comparison (i.e., not stored). The maximum space mapped by these entries is 16K pages, thus seven additional bits of the VPN are optionally excluded from hit comparison.
- When a block entry hit is encountered, the 20 bit RPN is assembled by bypassing low order bits of the VPN into the RPN corresponding to the bit positions that were excluded from the VPN compare.

Thus for the smallest space mapped by a block entry (128pages = 512kbytes):

```
if virtual_address[0:28] = VPN[0:28]          {hit on block entry}
then RPN[0:19] = cat(TRANS[0:12],VPN[29:35])  {assemble effective RPN}
```

For the largest space mapped by a block entry (16k pages = 64mbytes):

```
if virtual_address[0:21] = VPN[0:21]          {hit on block entry}
then RPN[0:19] = cat(TRANS[0:5],VPN[22:35])   {assemble effective RPN}
```

Other legal space sizes include 256 pages, 512 pages, 1k pages, 2k pages, 4k pages, 8k pages and 16k pages. Insertion of addresses and protection and the specification of block sizes is accomplished through pdc calls. The mechanisms provided by the CPU for use by this code are discussed under the subtitle Diagnose Functionality. Normally, block entries are unaffected by the architected Insert and Purge instructions.

3.1.1.3 TLB Page Replacement

The target entries for IITLBA and IDTLBA instructions are selected by a hardwired algorithm resident in the TLB. When an IITLBA or IDTLBA is performed, one of the 96 page entries, numbered from 0 to 95, is selected as the target for the insertion. The highest priority is given to any entry whose VPN field matches the VPN being inserted, in order to avoid multiple mappings of the same page. If no entry with a matching VPN is found, the lowest numbered entry which is invalid (i.e., its protection field's E–bit is 0) is selected. If both of these attempts fail, then the TLB must select a currently valid entry to replace. A ”not–recently–used” selection is made to determine the target in this case (ie. the lowest number entry that has the ‘‘used bit’’ clear is replaced, or if all entries are ‘‘used’’, then entry 0 is replaced and all the ‘‘used’’ bits are cleared).

It is possible to partially shield an entry from being the target of a replacement. Each entry contains, in addition to the architected contents, a one–bit field known as the *lock–in* bit. If this bit in an entry is set, the hardware algorithm will only select that entry for replacement if its VPN matches the VPN being inserted. The entry is then said to be *locked in*. It is also possible, for initialization and test purposes, to PARTIALLY circumvent the hardware algorithm and specify directly which entry is to be replaced during the next insertion, using a six bit pointer to indicate the target entry. (A VPN match in a page entry will ALWAYS cause an insert or purge to operate on that entry in addition to any entry pointed to by the diagnose register). This function is known as *diagnostic insertion*. When this method of selecting the targets for IITLBA and IDTLBA is in use, even locked–in entries may be selected for replacement. Both locking entries in and explicitly designating the replacement target are done through the diagnose functionality present in the TLB.

3.2 System Start–Up

At chip power on, all TLB entries are locked out. Code is responsible for clearing the lock out bit before an entry can be used. This must be done by inserting a unique address, using the diagnostic insertion mechanism, into each location whose lock out bit is to be cleared. Once the entries have had their lock–out bits cleared and their VPN fields set with unique addresses, the subsequent uniqueness of addresses in each entry is guaranteed by the hardwired replacement algorithm. Code must also initialize (clear) the lock in bits to enable replacement. All 96 E–bits should also be cleared (via PxTLBE).

It is possible for initialization code to test the TLB through use of the Probe instructions. Entries can be tested noninteractively by enabling them (clearing the lock–out bit) one entry at a time. The diagnose replacement pointer may be handy for this purpose. Individual failed entries can be mapped out of the TLB by setting both the lock out and the lock in bits. Once initialization is complete, the control diagnose register should be loaded with a value such that all the diagnose features are disabled.

3.3 Test/Debug

3.3.1 Entry Lock–in

Each page entry contains a lock–in bit. This bit is loaded from the Inhibit Replacement control bit in the control diagnose register (DR8[17]) whenever an address insertion to that entry occurs. When this bit in an entry is set, the entry will only be selected for replacement if its VPN matches the VPN being inserted or if the entry is the target of a Diagnostic Insertion.

3.3.2 Entry Lock–out

Each page and block entry contains a lock–out bit. This bit is loaded from the Force VPN Mismatch control bit in the control diagnose register (DR8[16]) whenever an insert address to that entry occurs. When set, the effect of the lock–out bit is to force a VPN mismatch on that entry. At system power–on, all 96 page entries and all 8 block entries are locked out. Initialization code is required to clear all of the lock out bits and load the VPN in each entry with a unique address.

It is important to make sure that for all locked out entries the lock–in bit is also set. Otherwise, a locked out entry might be selected as the target of an insertion. As long as both these bits are set in an entry, it will never be a target for replacement and it will never be used during a translation. It is always possible to clear the lock bits for an entry using the Diagnostic Insertion mechanism.

3.3.3 Diagnostic Insertion

Diagnostic Insertion provides a mechanism to explicitly designate, with a seven–bit pointer indicating a number between 0 and 95, which page entry will receive the next insertion. To use this mechanism, the Replacement Pointer field (DR8[18:24]) in the control register should be set to the integer index of the entry to be inserted to. In addition, the Page Entry LRU–Insert Disable bit (DR8[25]) should be set to ’1’, and the Block Entry Force Insert Enable bit (DR8[31]) should be set to ’0’. The next address insertion will occur to the indexed page entry (AND also to any page entry that has a SID.VPN match to the address being inserted). To disable diagnostic insertion to a page entry, a number greater than 95 must be placed into DR8[18:24], with 127 (0x7f) being the preferred value.

3.3.4 Insertion of Block TLB Entries

Block TLB insertion is accomplished by pdc using a combination of diagnose write functionality and architected Insert instructions. The block entries do not respond to the architected Insert Address and Protection instructions, nor do they respond to Purge, Purge Entry or Broadcast Purge, unless the Diagnose Control register has been previously loaded with a certain set of values.

The block entry insertion process is explained by the following example:

To insert to block entry N (N=0..7):

1. Ensure NO page entry has a SID.VPN match (even for INVALID entries).
2. Set the Page Replacement Pointer (DR8[18:24]) to 0x7f.
3. Set the Page Entry LRU–Insert Disable bit to 1 (DR8[25]).
4. Set the Block Entry Force–Insert Enable bit to 1 (DR8[31]).
5. Set the Block Entry Select field in the Diagnose register (DR8[28:30]) so that it points to the Block entry slot to be inserted (N).
6. Assemble a virtual address (= cat(space,offset[0:19])) for any page within the block to be mapped, then
7. modify that Virtual address by overwriting the seven least significant bits of the offset (offset[13:19]) with the block size specifier (see table this section, below).
8. Assemble the Physical page No. (in GR[r][7:26]), then
9. modify that Physical page No. by overwriting the seven least significant bits of the offset (offset[13:19]) with the block size specifier (see table this section, below).
10. Execute an Insert Address instruction.
11. Execute an Insert Protection instruction as with a page entry.
12. Reset the Diagnose register to its original (disabled) value.

mask[6:12]	**vpn bits compared**	**rpn bits returned**	**block TLB size (in 4KB pages)**
0000000	0–5	0–5	16,384 pages
1000000	0–6	0–6	8,192 pages
1100000	0–7	0–7	4,096 pages
1110000	0–8	0–8	2,048 pages
1111000	0–9	0–9	1,024 pages
1111100	0–10	0–10	512 pages
1111110	0–11	0–11	256 pages
1111111	0–12	0–12	128 pages

All other values of mask[6:12] are illegal.

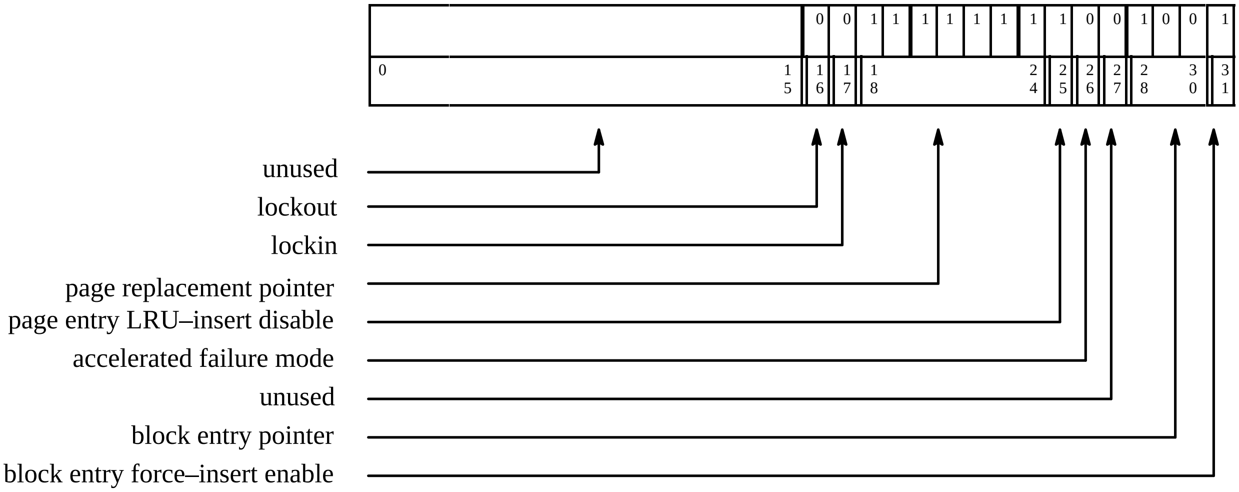
Example:
Block TLB entry:

```
space[16:31] = ”0001111100000110”
vpn[0:12]    = ”0111011001011”
rpn[0:12]    = ”0001001000110”
mask[6:12]   = ”1100000”
```

Translations:

space[16:31]	offset[0:31]	real addr[0:31] (or TLB miss)
0x1f06	0x76584321	0x12584321
0x1f06	0x76ffffff	0x12ffffff
0x1f06	0x77777777	TLB miss (bit 7 of vpn)
0x1f16	0x76000000	TLB miss (bit 27 of space)

The diagnose register for an insert to block entry #4 should be:



```
0x00003fc9  Lockout = 0
            Lockin = 0
            Page Replacement Pointer = 127
            Page Entry LRU–Insert Disable = 1
            Accelerated Failure Mode = 0
            Block Entry Pointer = 4
            Block Entry Force Insert Enable = 1
```

A PxTLBE instruction WON'T invalidate the block TLB entries; just the page entries.

A block entry may be purged in a similar manner: When DR8[18:24] = 127 and DR8[25] = DR8[31] = 1, the block entry selected by DR8[28:30] becomes the target for subsequent Purge instructions. Note that insertion of protection in block entries does not function in the same manner as in page entries, which select the IDTLBP or IITLBP target through the VPN referenced in the instruction. Block entries must have both protection and address inserted using their block entry select bit (DR [28:30]). Block entries do not respond to Purge entry instructions. Note also that page entries (even invalid ones) that have a matching SID.VPN will always respond to insert or purge instructions, regardless of the setting of the diagnose register.

Probe and LPA instructions are essentially TLB translation accesses, and thus function the same for block entries as for page entries.

3.3.5 Instruction Lookaside Buffer

In addition to the 96 page entries and 8 block entries of the UTLB, there is a four–entry Instruction lookaside buffer (ILAB). The ILAB is architecturally invisible but affects performance due to penalties incurred to update the buffer. These penalties are enumerated in the Performance section of this ERS. The following description of the ILAB is included as a qualitative description of the implementation:

The LAB contains:

13. VPN 0:19
14. IAD 30:31 (current privilege)
15. RPN 0:19
16. gateway privilege 30:31
17. tlb miss trap indication (1 bit)
18. tlb prot trap indication (1 bit)

In real mode operation the LAB is bypassed. No entries are written during real mode execution.

In normal virtual fetching a matching entry's RPN is used to check the instruction cache for misses, the gateway privelege is used for a GATE instruction, and the trap bit indicates whether the address should cause an ITLB protection trap (if instruction at that address is not nullified). If a miss is signalled, the main UTLB will be checked for the entry. If it is not there, the hardware TLB miss handler will be invoked if enabled, and if the hardware handler is unsuccessful an ITLB miss trap will be taken (if the instruction at that address is not nullified).

The following cases describe when the LAB is updated:

`Case #1: BE, BLE, BV`

During the Bstate of these instructions the LAB is updated. At that point the IHANG that would normally be taken for these instructions is incurred.

`Case #2: Branches (predicted right or predicted wrong)`

These incur no penalty unless they are taken and the target is on a different page than current page (determined by checking the LAB's VPN). If taken to a different page and the PSW–C bit is set, there is a one additional state penalty to update the LAB and check traps.

`Case #3: RFI`

An RFI hangs for two states, takes a step, then hangs for two more. The LAB updates and traps are checked for both RFI targets. Each one causes an additional hang state (total of three for each RFI hang).

`Case #4: P–bit changes`

Whenever a system mask instruction executes the LAB has to be invalidated and restored from the UTLB.

`Case #5: Control register changes`

Whenever a MTCTL occurs to one of the PID registers the LAB must be invalidated and restored from the UTLB.

`Case #6: TLB instructions`

All TLB instructions will cause a LAB invalidate. The ‘‘Entry’’ forms will invalidate the entire 4 entry LAB; all the other TLB instructions will only invalidate a matching entry.

3.3.5.1 TLB penalties

The processor must hang the pipeline to handle TLB inserts and purges. The various penalties are summarized below in number of states:

```
IxTLBA,IxTLBP,PxTLB with PSW–C=0   2
                    with PSW–C=1   3
PxTLBE                        2
```

3.3.6 Hardware TLB Miss Handler

The hardware TLB miss handler on the PA7300LC is designed to reduce the TLB miss penalty while being low–cost to implement (in complexity and area). The hw TLB handler is invoked on I–side and D–side translations that miss the on–chip TLB. The handler computes the address of a PDIR entry based on the missing space and vpn. It then accesses the PDIR entry. The PDIR entry is checked for three things:

1. Valid tag,
2. Matching tag, and
3. Reference bit = 1.

If the checks pass, the RPN and protection of the PDIR entry are inserted into the on–chip TLB and the original access is re–translated.

If any of the checks fail, the handler will not insert the entry and the instruction will trap to software. For DTLB misses only, a pointer to a PDIR entry is passed to the software TLB handler so that it doesn't have to recompute the PDIR address. The pointer is passed in CR28. CR28 will either have:

1. the address of the current PDIR entry, or
2. the address of the next PDIR entry.

Whether the current or next PDIR entry is passed will depend on the configuration of the diagnose register and which of the checks failed.

The hw TLB handler looks into a table of PDIR entries that will be referred to as the hardware–visible table. For an inverted page table, the hw–visible table contains the first level entries. For a forward mapped page table, the hw–visible table contains a "cache" of entries that are distinct from the actual page tables (this mode is targeted for OSF). Note that these tables must be stored in Little–Endian mode if the default Endian bit in diagnose register #0 is set.

There are bits in diagnose register #25 that disable the hw TLB handlers. If they are not enabled, the PA7300LC will take TLB miss traps without first activating the hardware handler.

3.3.6.1 PDIR Address Generation

The starting address of the hw–visible table is stored in DR24. The table must start on a 4KB page boundary (i.e. bits 0–19 of DR24 are significant). To generate the address of the PDIR entry, the missing space and vpn are hashed together. Upper bits are masked off depending on the size of the hw–visible table to give an offset into the table. This offset is then merged with DR24 to get a 32–bit real address that is used to access the cache. Note that the PDIR is accessed in real–mode and the default Endian bit in diagnose register #0 is used to control big/little Endian mode.

The base address is *merged* with the offset (not added to the offset). As a result software must align the hw–visible table to a multiple of its size.

```
Pseudocode for address generation:

     spc  : Missing space[16:31]
```

```
    off  : Missing offset[0:31]
    dr24 : diagnose register #24 –– contains base address of table
    dr25 : diagnose register #25 –– contains mask bits (based on table size)

  extru   off,19,20,off_tmp1           ; right shift vpn
  zdep    off_tmp1,27,20,off_tmp2      ; position vpn at 8:27
  zdep    spc,22,16,spc_tmp1           ; position space at 7:22
  xor     spc_tmp1,off_tmp2,hash_addr  ; perform hash
  mfdiag  dr25,mask                    ; get mask value from diag reg
  depi    –1,31,12,mask                ; don’t mask bits 20:31
  and     mask,hash_addr,hash_addr     ; mask out bits of hashed addr based
                                       ;   on table size
  mfdiag  dr24,base                    ; get base addr from diag reg
  depi    0,31,12,base                 ; only bits 0:19 are significant
  or      base,hash_addr,hash_addr     ; merge in the base addr of the table

  Hash_addr is a 32–bit real addr that is used to access the cache
  and to check for a dmiss.
```

3.3.6.2 Structure of Hardware–Visible Table

This is what is assumed about the organization of the hw–visible table in order to implement the hw TLB handler. Each 8–word line of the table holds two entries and is organized as shown below. Bits reserved are indicated by S for software fields, 0 for hardware fields.

word0 tag1

0	1 15	16 31
V	offset[0:14]	space[16:31]

word1 prot1

0 4	5 11	12	13 15	16 30	31
RSTDB	ACR(7)	U	000	access_id(15)	S

word2 rpn1

0 6	7 26	27 31
SSS0000	rpn[0:19]	SS000

word3 next1

0 31
real address of next pdir entry

word4 tag2

0	1 15	16 31
V	offset[0:14]	space[16:31]

word5 prot2

0 4	5 11	12	13 15	16 30	31
RSTDB	ACR(7)	U	000	access_id(15)	S

word6 rpn2

0 6	7 26	27 31
SSS0000	rpn[0:19]	SS000

word7 next2

0 31
real address of next pdir entry

Offset bits 15–19 are implicit in the address of the PDIR entry and are not stored in the tag.

The hashing algorithm will provide an address that points to either word0 or word4 of a line. If the line is not resident in cache the processor will bring it in from main memory (it is required to be in main memory). The tag (either word0 or word4) is read, the valid bit of the tag is checked to ensure that it is valid and the tag is compared to the missing vpn[0:14] and space[16:31]. Also the reference bit (R–bit) of the protection word (either word1 or word5) is checked to ensure that it is a ”1”. If all of the checks pass then the protection and rpn are inserted into the on–chip TLB.

After the insert to the on–chip TLB is done, the access that had the TLB miss is re–translated and any TLB traps resulting from the re–translation will occur (for example DATA_MEM_PROT_TRAP).

If any of the checks fail then the CPU will trap to software with a DTLB_MISS_FAULT. CR28 is used to pass information to the trap handler about the address of the PDIR. The table below shows how CR28 is updated:

((R=1) and (V=1))	(Tag Match)	DR25[29]	DR25[30]	CR28 value
False	DC	DC	DC	current pdir
True	False	0	DC	current pdir
True	False	1	0	next pdir
True	False	1	1	word3 of line

current pdir : real addr of current pdir entry (points to word0 or word4)
next pdir : real addr of next pdir entry (this is word3 if we hashed to word0 or word7 if we hashed to word4).
word3 of line : this is word3 regardless of whether we use first or second entry in the line.

New intro for this list?????

- Penalties

Type	**Penalty**
DTLB hit and insert	11 states
DTLB trap to software	11 states

ITLB hit and insert	11 states
ITLB trap to software	11 states

- Buddy Pages

  The PA7300LC will not implement buddy page handling. The decision was made based upon the performance numbers and the estimated complexity that would be required for implementation.

- Relied–upon Translations

  If the hardware TLB handler inserts any entries to the on–chip TLB it must force a re–translation of B–stage data access (if any) to ensure we don't violate the rules for relied–upon translations outlined in the PA–RISC manual.

- Implementation Specific Inserts

  There are two new implementation specific instructions added to reduce the TLB miss penalty for the software miss handler. These instructions use undefined minor opcode bits. The instructions reduce the TLB miss penalty in two different ways:

  (1) They insert directly from control registers. For DTLB misses ISR/IOR are used. For ITLB misses the front elements of IIASQ/IIAOQ are used.

  (2) They execute in fewer cycles than the regular TLB inserts.

  The architected insert instructions are also implemented in order to be architecturally compliant and also to provide easy–to–use inserts for code which does not need to be handcrafted for performance.

```
Summary of instructions:
    idtlbaf –– fast insert dtlb address     (0 penalty states)
    idtlbpf –– fast insert dtlb protection  (1 penalty state)
    iitlbaf –– fast insert itlb address     (0 penalty states)
    iitlbpf –– fast insert itlb protection  (3 penalty states)
```

See the Appendix for instruction formats and software restrictions.

4. Floating Point

4.1 Overview

The PA7300LC contains 64 bit floating–point ALU, multiply, and divide/square root circuits and a 32x64 bit floating–point register file.

The floating point unit implements the PA–RISC 1.1 architecture, Third Edition. In addition, it implements the following product–specific or architecturally optional features:

- Multiply and truncate
- Hardware underflow mode (Denormalized As Zero bit)

The latencies and issue rates of floating–point operations are in the table below. The first number is the latency in cycles and the second number is cycles per instruction issue.

	Single	Double
Add/Subtract	2/1	2/1
Multiply	2/1	3/2
Divide	8/8	15/15
Square Root	8/8	15/15

Peak performance at 160 MHz is 320 megaflops single precision and 160 megaflops double precision.

The PA7300LC floating–point model number is x’0F. The revision for the first release is x’01.

4.2 Instruction Decoding Rules

4.2.1 Reserved–Op Exceptions

The reserved–op exception occurs on:

- A x’0C or x’0E instruction with a reserved or undefined sub–op:
  - □ class 0 subops 1,6,7
  - □ class 2 subops 2–7
  - □ class 3 subops 4–7
- A x’0E instruction, with a FMT code of b’11.
- A x’0E instruction other than XMPYU with bit 23 set (integer op).
- A XMPYU with bit 20 set (double precision).

Reserved–op exceptions are always reported through the unimplemented exception/trap rather than an immediate assists exception trap. In other words, the trap handler will see the T bit set and the offending instruction marked unimplemented in exception register 2.

4.2.2 Emulated Instructions

The PA7300LC relies on software to emulate the following instructions:

- Any quad precision flop
- Any FRND instruction

These will raise the unimplemented exception/trap. The trap handler will see the T bit set and the offending instruction marked unimplemented in exception register 2.

A subroutine call may be much faster than the OS emulation routines.

4.2.3 Product–Specific Instructions

- FMPYCFXT is encoded as FMPYADD with zeroes in the ra field.

For more information see the ”Product–Specific Features” section.

4.2.4 FTEST look–alikes

The following are undefined:

- N bit set on a x’0E instruction.
- N bit set on a x’0C instruction other than FTEST (class 2, subop 1).
- N bit clear on an FTEST (x’0C, class 2, subop 1).
- Class 2, subop 1 (would–be FTEST) on a x’0E instruction.

4.2.5 Miscellaneous Undefined Instructions

The following are undefined:

- Any flop which uses register 1, 2, or 3 as a source operand.
- Any flop which uses register 0, 1, 2, or 3 as a result operand.
- FMPYCFXT,SGL when fp register 16L is nonzero.
- A x’0C instruction with a format code of 10.
- FCNVFF,SGL,SGL – this is treated as a FCPY,SGL.
- FCNVFF,DBL,DBL – this is treated as a FABS,DBL
- COPR,0,0 if the most recent FP instruction was not an FSTD 0.
- COPR,0,0 if the next FP instruction is not an FSTD 0.
- FTEST with clip–test completer, if the most recent FP instruction was a COPR,0,0.

4.3 Unimplemented Exception/Trap

The only kind of exception generated by the PA7300LC floating point is the unimplemented exception. It is always signaled with a delayed floating–point exception trap. The unimplemented trap is raised instead of the overflow, underflow, division by zero, invalid and inexact traps.

The I, V, O, U and Z flags in the floating–point status register are never set due to an unimplemented trap.

In the lists below, operands are classified as either norm, denorm, zero, inf, or NaN. Some of the cases are marked with (D) or (!D). This means the exception only occurs when the D bit is set or cleared, respectively.

The following conditions cause the unimplemented trap:

- Various:
  - Reserved–op or Emulated conditions (see above)
- FABS:
  - only the Reserved–op and emulated conditions (see above)
- FCPY:
  - only the Reserved–op and emulated conditions (see above)
- XMPYU:
  - only the Reserved–op and emulated conditions (see above)
- FADD:
  - input nan
  - +inf + –inf with invalid enabled
  - overflow with overflow enabled
  - inexact with inexact enabled
  - input denorm (!D) unless the other operand is inf
  - tiny result (!D)
  - tiny result with inexact enabled (D)
- FSUB:
  - input nan
  - +inf – +inf with invalid enabled
  - –inf – –inf with invalid enabled
  - overflow with overflow enabled
  - inexact with inexact enabled
  - input denorm (!D) unless the other operand is inf
  - tiny result (!D)
  - tiny result with inexact enabled (D)
- FCNVff:

  - input nan
  - overflow with overflow enabled
  - inexact with inexact enabled
  - input denorm (!D)
  - tiny result (!D)
  - tiny result with inexact enabled (D)
- FCNVxf:
  - inexact with inexact enabled
- FCNVfx:
  - input nan
  - input inf
  - input denorm (!D)
  - overflow **
  - inexact with inexact enabled
- FCNVfxt:
  - input nan
  - input inf
  - input denorm (!D)
  - overflow **
  - inexact with inexact enabled
- FCMP:
  - input signalling nan
  - input quiet nan with bit 31 of instruction set
- FMPY:
  - input nan
  - inf * zero and invalid enabled
  - inf * denorm and invalid enabled (D)
  - overflow with overflow enabled
  - inexact with inexact enabled
  - tiny result (!D)
  - tiny result with inexact enabled (D)
  - input denorm (!D) unless other operand is zero or inf
- FDIV:
  - input nan

- □ inexact with inexact enabled
- □ overflow with overflow enabled
- □ denorm / denorm (!D)
- □ denorm / norm (!D)
- □ norm / denorm (!D)
- □ inf / inf with invalid enabled
- □ zero / zero with invalid enabled
- □ zero / denorm with invalid enabled (D)
- □ denorm / zero with invalid enabled (D)
- □ denorm / denorm with invalid enabled (D)
- □ denorm / zero with divz enabled (!D)
- □ norm / zero with divz enabled
- □ norm / denorm with divz enabled (D)
- □ tiny result (!D)
- □ tiny result with inexact enabled (D)

- ■ FSQRT:
  - □ input nan
  - □ input negative norm and invalid enabled
  - □ input negative infinity and invalid enabled
  - □ input negative denorm and invalid enabled (!D)
  - □ input positive denorm (!D)
  - □ inexact with inexact enabled
- ■ FMPYADD:
  - □ union of multiply and add conditions
- ■ FMPYSUB:
  - □ union of multiply and subtract conditions
- ■ FMPYCFXT:
  - □ union of multiply and truncate conditions

** fcnvfx,dbl,sgl and fcnvfxt,dbl,sgl will raise the unimplemented exception when their operand is equal to the largest representable negative integer, even though this is not strictly necessary.

When either operation of a multi–op instruction traps, neither operation writes its result and no exception flags are set in the floating–point status register.

4.3.1 Overflow Exception

The following conditions raise the overflow exception provided that the unimplemented trap does not occur. The O flag in the floating–point status register is set.

- FADD, FSUB, FCNVff, FMPY, FDIV, FMPYADD, FMPYSUB:
  - □ overflow with overflow disabled
- FMPYCFXT:
  - □ overflow in multiply with overflow disabled

4.3.2 Division by Zero Exception

The following conditions raise the division by zero exception provided that the unimplemented trap does not occur. The Z flag in the floating–point status register is set.

- FDIV:
  - □ denorm / zero with divz disabled (!D)
  - □ norm / zero with divz disabled
  - □ norm / denorm with divz disabled (D)

4.3.3 Invalid Exception

The following conditions raise the invalid exception provided that the unimplemented trap does not occur. The V flag in the floating–point status register is set. The result register is set to a quiet NaN.

- FADD:
  - □ +inf + –inf with invalid disabled
- FSUB:
  - □ +inf – +inf with invalid disabled
  - □ –inf – –inf with invalid disabled
- FMPY:
  - □ inf * zero with invalid disabled
  - □ inf * denorm with invalid disabled (D)
- FSQRT:
  - □ input negative infinity with invalid trap disabled
  - □ input negative norm with invalid trap disabled
  - □ input negative denorm with invalid trap disabled (!D)
- FDIV:
  - □ inf / inf with invalid disabled

- □ zero / zero with invalid disabled
- □ zero / denorm with invalid disabled (D)
- □ denorm / denorm with invalid disabled (D)

4.3.4 Inexact Exception

The following conditions raise the inexact exception provided that the unimplemented trap does not occur. The I flag in the floating–point status register is set.

- ■ FADD, FSUB, FCNVFF, FCNVFX, FCNVFXT, FCNVXF, FMPY, FDIV, FSQRT, FMPYADD, FMPYSUB, FMPYCFXT:
  - □ inexact and inexact disabled
- ■ FADD, FSUB, FCNVff, FMPY, FDIV, FSQRT, FMPYADD, FMPYSUB, FMPYCFXT:
  - □ tiny result and inexact disabled (D)

4.3.5 Underflow Exception

The following conditions raise the underflow exception provided that the unimplemented trap does not occur. The U flag in the floating–point status register is set.

- ■ all operations: never

4.3.6 Exception Registers

The excepting flop will be in exception register 2. PA–RISC 1.1 exception codes are used. Exception registers 1, 3, 4, 5, 6, and 7 may be loaded or stored, but hardware will never place an excepting flop in them.

Exception register 2 is guaranteed to retain its contents as long as the T bit remains set and continuing thereafter until a flop is executed, or until it is explicitly cleared by software with a load.

4.4 Product–Specific Features

The PA7300LC has floating–point features which are product–specific (not part of PA–RISC 1.1). Programmers who choose to take advantage of these features are warned that some older machines do not implement them and even future machines may not implement them. Behavior on other machines will be undefined, meaning that even a trap for emulation is not guaranteed. Except for Hardware Underflow Mode, these features cannot be disabled.

4.4.1 Hardware Underflow Mode

The PA7300LC implements the optional quick hardware underflow mode for floating–point operations. This mode is enabled by setting bit 26 of the floating–point status register, called the ”D” bit (for Denormalized As Zero).

In hardware underflow mode, operations which would normally signal the underflow exception will just return a zero result with no exception. Input denorms are treated as signed zeroes. The underflow flag is never set. The inexact flag and inexact exception are detected just as in IEEE mode except that denormalized operands are treated assigned zeroes, and when a result is flushed to zero the inexact flag is set (if inexact traps are enabled there will be an exception).

Note that when this mode is enabled, computations are not compliant with the IEEE floating–point standard.

This mode does not affect the speed of floating point operations. It just saves the overhead of a trap handler when there are denormalized operands or when an operation underflows.

Note that FABS and FCPY are not affected by this mode. They do not flush denorms to zero.

Hardware underflow mode is an optional feature of the PA–RISC 1.1 architecture (Third Edition), and may not be implemented on other PA–RISC processors.

4.4.2 Multiply and Truncate FMPYCFXT

Format: `FMPYCFXT,fmt rm1,rm2,tm ta`

06	rm1	rm2	ta	0	f	tm
6	5	5	5	5	1	5

Purpose: To perform a floating point multiply and change the format of a floating point value to a fixed–point value.

Description: This instruction is like FMPYADD except that instead of doing the add it performs an FCNVFXT,fmt,sgl on register ta and puts the resulting signed integer back into ta.

The convert always produces a 32 bit result. If the format is double then the result is placed in the MSW (bits 0:31) of ta and the LSW of ta becomes undefined.

The single precision version of this instruction (FMPYCFXT,sgl) is undefined if register 16L is nonzero. So before using FMPYCFXT,sgl you must load a zero into register 16L.

Conditions: None

Operation: FPR[tm] ← FPR[rm1] * FPR[rm2]
FPR[ta] ← convert_float_to_fixed(FPR[ta],fmt,sgl,ROUND_TOWARD_ZERO);

Exceptions: Assist emulation trap
Assist exception trap

4.5 Performance Tuning

4.5.1 Notation

Minimum Distance is the number of cycles between two instructions necessary to avoid a pipeline interlock. Consecutive instructions are 1 cycle apart. For example, without super scalar execution this:

```
A
other instruction
B
```

would cause a 1 cycle penalty if there were a minimum distance of 3 between A and B. If there were a minimum distance of 3 between A and B and a minimum distance of 6 between A and C, then

```
A
B
other instruction
C
```

would cause a 3 cycle penalty: 2 cycles on B and 1 cycle on C. This is because penalties associated with two or more constraints can be served in parallel.

4.5.2 Latencies

A Flop is one of the instructions shown in the table below. Loads, stores and FTESTs are not flops. The table shows the Execution Time (latency cycles) for each flop and which functional unit is used to execute it.

Flop	Execution Time	Functional Unit
FCPY,SGL/DBL	2	ALU
FABS,SGL/DBL	2	ALU
FADD,SGL/DBL	2	ALU
FSUB,SGL/DBL	2	ALU
FCMP,SGL/DBL	2	ALU
FCNV*,SGL/DBL	2	ALU
FMPYADD,SGL	2	ALU and MPY
FMPYSUB,SGL	2	ALU and MPY
FMPYCFXT,SGL	2	ALU and MPY
FMPY,SGL	2	MPY
FMPYADD,DBL	3	ALU and MPY
FMPYSUB,DBL	3	ALU and MPY
FMPYCFXT,DBL	3	ALU and MPY
FMPY,DBL	3	MPY
XMPYU	3	MPY (integer multiply)
FDIV,SGL	8	DIV
FDIV,DBL	15	DIV
FSQRT,SGL	8	DIV
FSQRT,DBL	15	DIV

In the tables that follow, N1 is the execution time of the first instruction.

4.5.3 Data Cache Contention Constraints

None. The PA7300LC cache design removes the ”store–tail” penalty, so there is no longer a 1 cycle interlock for loads or stores that immediately follow store instructions. Thus, the minimum distance is now 1 cycle.

1st Instruction	ST*,FSTW*,FSTD*
2nd Instruction	LD*,ST*,FLD*,FST*
Minimum Distance	1 cycle (i.e. no penalty)

For the second instruction, LD* does not include LDIL or LDO since these ‘‘loads” do not actually access the data cache.

4.5.4 Functional Unit Contention Constraints

1st Instruction	FDIV or FSQRT
2nd Instruction	*
Minimum Distance	N1 cycles

1st Instruction	XMPYU or FMPY,dbl
2nd Instruction	*
Minimum Distance	2 cycles

This rule means that there is always a 1–cycle penalty for any double precision FMPY, a 7–cycle penalty for a single precision FDIV/FSQRT, and a 14–cycle penalty for a double precision FDIV/FSQRT.

4.5.5 Data Dependency Constraints (Performance Cases)

1st Instruction	FLD*
2nd Instruction	Flop
Minimum Distance	2 cycles

The target of the load equals one of the sources of the flop. (Increase the distance by 1 cycle if the load is singleword singleword and the flop’s source is double precision.)

1st Instruction	Flop
2nd Instruction	Flop
Minimum Distance	N1 cycles

The target of the first flop equals one of the sources of the second flop. (Increase the distance by 1 cycle if the format is different, i.e., sgl–dbl, dbl–sgl, int–fp, fp–int.)

1st Instruction	FCMP
2nd Instruction	FTEST
Minimum Distance	2 cycles

4.5.6 Dependency Constraints (Non–performance Cases)

1st Instruction	FLD*
2nd Instruction	FST*,FLD*
Minimum Distance	3 cycles

The target of the first load equals the target of the second load or source of the store.

1st Instruction	FLD*
2nd Instruction	Flop
Minimum Distance	3 cycles

The target of load equals a target of the flop.

1st Instruction	Flop
2nd Instruction	FLD*/FST* 0–3
Minimum Distance	N1+1 cycles

1st Instruction	FLD*/FST* 0–3
2nd Instruction	Flop
Minimum Distance	4 cycles

1st Instruction	FLDD*/FSTD* 0
2nd Instruction	FLD*/FST*/FTEST
Minimum Distance	4 cycles

These are loads and stores of the status and exception registers.

4.5.7 Other Rules

If the first instruction is a load that suffers a d–cache miss then add the latency of the miss to the minimum distance.

The minimum distance may be shortened if one or both of the two instructions is nullified. This varies from case to case and can depend on other factors such as unrelated interlocks.

4.5.8 Example

This example is coded so there are no interlocks.

```
FLD*     mem,r4
other1
FADD,sgl r4,r4,r5   # 2 cycles away from load
FADD,sgl r7,r8,r9   # independent of 1st add
FMPY,sgl r5,r4,r6   # 2 cycles away from 1st add
FST*     r6,mem     # will not bundle with FMPY
other2
FLD*     mem,r10    # can execute the cycle after FST
```

With dual issue this could execute in 7 cycles because the FST could bundle with other2, and the FLD can execute on the next cycle since the PA7300LC implements single cycle stores. Note that the first FADD will be bundled with other1, but you don't gain any cycles because the FADD needs to be 2 cycles away from the FLD. Also, unlike the PA7100LC, the FMPY and FST will not bundle.

5. Instruction Cache

5.1 General Overview

The PA7300LC incorporates a 64 kilobyte 2 way set associative instruction cache. Fetches from this cache are done in doublewords, with a steering and bundling block sending either one or two instructions to the execution units. This cache also includes a one line (32 byte) prefetch buffer. This prefetch buffer is arranged to look to the fetch unit like a third way of associativity in the cache. That means, if the address being fetched is in the prefetch buffer, there is no pipeline penalty. The PA7300LC will also execute instructions as they are returned from the memory controller during a copyin, as the prefetch buffer is being filled. Another performance feature of the instruction cache is its ability to accept a copyin in one state. The prefetch buffer becomes the copyin buffer when the fetch stream hits it while missing the array. The prefetch buffer is also used for write data during diagnose writes and built–in self test (BIST).

Instruction cache line prefetching is enabled through CPU diagnose register zero. When enabled, the instruction cache controller will attempt to fill the prefetch buffer with the next line whenever the fetch address misses the cache or hits a valid prefetched line. The first line of a page will never be prefetched, nor will IO addresses be prefetched. A valid prefetched line will only be written into the cache when the fetch address misses the array but hits the prefetch buffer.

On the PA7300LC, the FICE instruction will write an invalid tag to both groups at the index computed by the instruction. Whatever is in the prefetch/copyin buffer will be written into the data and tag arrays, except the upper four bits of the tags will be forced to ones to indicate an invalid line. Thus a sequence of addresses from zero through 1023 will invalidate the entire on chip instruction cache.

5.2 System Start–Up

The instruction cache must be disabled by clearing bit 29 (L1ICACHE_EN) in diagnose register zero (CPU_CFG) until all the lines have been made invalid with FICE instructions. No code may be executed which is located in memory space at any time the Icache is disabled. Any time BIST or diagnose is being performed, software should set bit 11 (L1IHPMC_DIS) in CPU_CFG to mask HPMCs. Prior to enabling the Icache, software should clear bit 10 (L1IHPMC) in CPU_CFG.

The main action to perform for the instruction cache at startup is to invalidate all the lines. This is done by executing 1024 FICE instructions, with addresses from 0 to 1023 left shifted 5 bits. A built–in self–test is not necessary, but may be performed at startup. BIST is described in detail in the Test/Debug section below. Note that even if BIST is performed, the instruction cache must still be invalidated with an FICE sequence. (Alternatively, a last pass of BIST may be performed with a PFBTAG with all four most significant bits set.) Any pending parity errors must be cleared by writing a one to bit 10 of diagnose register zero, and by performing a write of any value to any of the four ICPFBDATA registers.

After the instruction cache has been invalidated, and parity errors cleared, it can safely be enabled by setting L1ICACHE_EN. Cache line prefetching may also be enabled at this time (bit 26, IPREF_EN). There is no reason it shouldn’t be enabled for normal system operation; this enable bit is intended only for pre–production testing.

5.3 Test/Debug

The instruction cache reuses the prefetch buffer for both diagnose writes and built–in self–test (BIST). In addition, a signature analyzer captures all array data during normal operation, or diagnose read data, or array data modified by a polynomial during BIST. When capturing normal fetch data, update of the register will be inhibited when a parity error is detected, so that the HPMC handler may recover the data which produced the error.

BIST is intended for manufacturing test, and therefore should be unnecessary for normal system startup.

5.3.1 Diagnose and Built–In Self–Test

The on–chip first level instruction cache can be tested through a built–in test mechanism. This mechanism is shared with the diagnose functionality implemented on previous processor chips. There are four sections of functionality: the address, the write data, the read data, and the configuration.

- IC_BIST_ADDR_CFG (DR10) – This register holds the cache index and the BIST/diagnose configuration bits. During a full pass BIST, the address portion of this register will automatically increment or decrement. Increments follow the polynomial $P(x) = x^{12} + x^7 + x^4 + x^3 + 1$. Decrements use the inverse of P(x), $P^*(x) = x^{12} + x^9 + x^8 + x^5 + 1$. The configuration bits control BIST LFSR increment/ decrement, whether the BIST continues until the LFSR reaches all zeroes, whether the access is BIST or diagnose, and which group is written for a diagnose write.

- ICPFBDATA, ICPFBSP (IO mapped) – Three register addresses comprise a 'doubleword' in the prefetch buffer, which is used as the write path for diagnose writes to the instruction cache array.

- ICPFBTAG (IO mapped) – This register contains the cache tag and force parity bit for the Prefetch Buffer. Note that there is no special bit to indicate a line is invalid; bits zero through 3 must be set to all ones to invalidate a line with a diagnose write.

- ICBAR (IO mapped) – This is actually a set of registers (from the viewpoint of the programmer) which contain the most recently read bits from the instruction cache array, if the last operation was a diagnose read. If the last operation was a BIST cycle, the registers will contain the array data XOR'ed with the previous value of these registers, according to the polynomials $P(x) = x^{21} + x^2 + 1$ for tags and $P(x) = x^{74} + x^{16} + x^{15} + x + 1$ for data. If the last operation was a fetch with a parity error, these registers will freeze with the erroneous data until the IHPMC bit is cleared in DR0. A write to any of these zeroes out the entire set of registers, independent of the source data value.

- IC_TIMING (IO mapped) – This register is not used for production systems. It is included only for phase 2 prototype testing.

5.3.2 Using the Instruction Cache BIST

To use either BIST or diagnose operations in the Icache, first it must be disabled by clearing the L1ICACHE_EN bit (bit 29) in diagnose register 0. Currently, this bit does not power up cleared, so PDC must clear it before running any Icache testing code. This also means that all code which enables or disables the Icache and any code executed while it is disabled must be located in IO space. There is currently no facility to execute memory space code while the Icache is disabled.

The pseudocode shown below implements a functional test of the on–chip first level instruction cache:

5.4 Instruction Cache Parity Errors

Parity is computed, stored, and checked on each 36 bits of instruction (32 plus 4 steering bits), and on the 20 bit tags. Parity errors may be signalled for the tag arrays or the data arrays in the instruction cache. Parity errors in the data arrays will only be signalled if one of the groups hits (in that case, the parity error may be in either group). Parity errors will always be signalled for the tags. When a parity error is detected, the BIST analyzer registers will freeze with the data fetched which caused the parity error. Note that this is only the case for normal fetches and flushes. When BIST is active, parity errors will not cause the analyzer registers to freeze. Note that because the fetch machine can issue addresses which are ahead of the execution stream, an HPMC trap may be taken for an address which did NOT have a parity error. For this reason, the trapping address and the following line address should be flushed when trying to recover in the trap handler.

5.4.1 When Icache Parity Errors are Signalled

Parity errors will be signalled for all addresses which the fetch machine generates, provided it is not to IO space. Although an invalid cache line is indicated by an IO–space–like tag, parity errors will be signalled for the tags even if there is a miss (especially a miss due to an invalid/IO–like tag). All flushes (FIC and FICE) will allow a parity error to be signalled on the target address. Again, **NOTE:** the prefetch buffer is written to the array during a flush, so **BE SURE TO CLEAR THE FORCE_PARITY_ERROR BITS** in the prefetch buffer before executing any flush instructions. Note also that flushes will not write all words of a line, and so may not clear a data parity error.

Since the instruction cache has two sets (or groups), a fetch (or flush) access reads a half line from each set. The half line consists of either the even doublewords or the odd doublewords, as indicated by address bit 28.

A side effect of an instruction cache parity error is that the ICBAR register will freeze with the offending data. Once software is finished reading these registers, they must be cleared and unfrozen by performing a store to any of the ICBAR_XX registers. Otherwise, the next parity error will not cause the data to be saved in ICBAR.

A parity error in the upper 4 bits of either tag are a special case. Generally, they are unrecoverable. If software detects a parity error in a tag, it will only be recoverable if it is a soft error. An FICE must be performed to invalidate both tags, then a diagnose read must be performed (after clearing the ICBAR) to determine if the FICE was successful. If not, there is no possible recovery from the parity error. If the FICE was successful, the parity error was a soft error.

5.4.2 Using Icache Diagnose

Configuration bits in diagnose register 10 (IC_BIST_ADR_CFG) allow the IC_DIAG instruction to perform a diagnostic read or write of the instruction cache. Diagnostic writes are functionally equivialent to a cache line copyin; in fact, they use the copyin buffer. Diagnostic reads are functionally equivalent to fetches; a diagnose read returns a half line plus tag from each of the two cache sets.

Before performing a diagnose read or write of the instruction cache, software must be executing from I/O space, the level one cache must be disabled and Icache prefetching must be disabled via diagnose register zero. To perform a diagnose write, all 12 of the prefetch buffer data/steering registers and the tag register must be written. Diagnose register 10 must be set to include the cache index and the target group (set). The DIAG_RDH bit must be cleared and the BIST_ENH bit must be cleared. Then an IC_DIAG instruction is executed to put the prefetch buffer data into the cache at the DR10 indicated index. The cache may then be enabled and the line may be branched to as if it were copied in by normal fetching.

A diagnose read does not read a single cache line. It reads two half lines, similar to the way data is fetched from the instruction cache. In order to read a single logical cache line, two diagnose reads must be performed. Additionally, the data returned by reads of the ICBAR must be ‘unscrambled’, as the bits are in an electrically optimal order.

6. Data Cache

6.1 General Overview

The PA7300LC incorporates a 64 kilobyte two–way set associative Level 1 on–chip data cache. It is virtually indexed and physically tagged. The cache line size is 32 bytes. The data cache is composed of four separate arrays: a tag array, a dirty array, a left data array, and a right data array. The tag array contains 1024 entries. Each tag entry is composed of 42 bits: two 20 bit tags, one for each way of associativity (group 0 or group 1), plus one tag parity bit for each group. The dirty array contains 1024 entries, each entry containing 4 bits: one dirty bit per group plus one dirty parity bit per group. Each data array contains 2048 entries, each entry containing 132 bits: 16 data bytes plus one data parity bit per 4 bytes. The data cache controller powers down the entire data cache when it is not being referenced by an instruction or any pending cache miss operations. Furthermore, each data array is divided into two halves, one containing the upper 16 bytes of each cache line and the other containing the lower 16 bytes of each cache line. Each of these halves can be powered down independently of the other. For load and store instructions, the halves of the data arrays that are not being referenced by the instruction are powered down.

The two data arrays are organized such that on any given cycle, either two doublewords corresponding to the two ways of associativity (or groups) are accessed from the upper or lower halves of the arrays or a single entire cache line for just one group is accessed. These two different modes of operation are called doubleword mode and checkerboard mode, respectively. The selection between the two modes is accomplished strictly through the addressing of the two data arrays. Since each data array contains 2048 entries, 11 address bits are needed per array. The first 10 of these address bits are common between the two arrays, but the least significant, or 11th, bit of address can vary between the two arrays. For doubleword mode, the 11th bit of each data array is set to the same value, corresponding to bit 28 of the data address specified by the instruction causing the reference. For checkerboard mode, the 11th bit of the left array is set to the group being accessed (either 0 or 1) and the 11th bit of the right array is set to the complement of this value. Because of this organization, no additional sense amps are needed to read an entire cache line over the number of sense amps that are required for normal two–way doubleword reads.

Word and doubleword load and store instructions can access the cache in a single cycle, thus avoiding ‘‘store tail” penalties. Byte and halfword loads also require only one cycle. Byte and halfword stores require two cycles to access the cache, although the penalty for the second access cycle will be hidden if the following instruction bundle does not reference the data cache.

The PA7300LC data cache also includes a two entry, FIFO, store queue. Store data is placed into the rear of the store queue whenever a store instruction executes (specifically, one cycle after the B–step of a store). The store queue advances on each store instruction (specifically, on the B–step of a store), with the head of the queue being written to the data cache array.

A full cache line read occurs on a data cache miss when the line selected for replacement is dirty and needs to be written to memory. It also occurs when a data cache flush instruction hits a dirty line in the cache. This full cache line transfer from cache to memory is termed a copyout. All 256 bits of data on a copyout are sent in parallel to the memory controller. A full cache line write occurs for the cache fill (or copyin) of a data cache miss. Because data is sent from the memory controller to the data cache in 64 bit quantities on cache misses, this data is accumulated in one of two copyin buffers until the entire cache line has been received, at which point it will be written into the data cache array as soon as a free data cache cycle is available. The two copyin buffers are arranged as a FIFO queue, refered to as the copyin queue. Two queue entries are needed because the data cache controller can issue up to two simultaneous reads to the memory controller. Copyin writes (to the array) will only occur from the front of the copyin queue. On cache reads, the front copyin queue entry behaves logically as if it had already written the cache array, provided that the data being accessed has been received from the memory controller.

Unlike prior processors, the PA7300LC implements Block Copy store hints uniformly across privilege levels. Regardless of the current privilege level, a hinted, line–aligned, word or doubleword store will zero–fill the addressed cache line without reading memory. The CPU diagnose bit CPU_CFG.SHINT_EN needs to be set for this to occur, however.

The data cache will check for correct parity any time the arrays are powered up to perform a read. If enabled, data cache parity errors will cause an HPMC. CPU diagnose bit CPU_CFG.L1DHPMC_DIS disables taking an HPMC upon detection of a data cache parity error. Although HPMCs can be suppressed, there is no way to disable the detection of data cache parity errors. Because a parity error on a data cache line may indicate data corruption, an HPMC caused by a data cache parity error is not recoverable. Data cache HPMCs should be disabled when the data cache is placed into test mode (see below).

Because there can be two dirty lines at any given cache index and because a flush instruction will generate at most one copyout, two FDCE instructions must be issued to each cache index to ensure that the data cache is completely invalidated.

6.2 System Start–Up

The data cache must be invalidated before any memory references occur (this does not include I/O loads and stores). The method used to invalidate can not allow any copyouts to occur. In addition, the data cache must be written with correct parity values before data cache parity errors are enabled, even if no memory references are to be issued. It is recommended that these operations be performed by initializing the data cache with invalid cache lines that have correct parity. The easiest way to do this is to execute a BIST operation (see below) that writes each entry in all of the arrays with the proper values. Recommeded values are 0xF0000 for the tags and zeros for all other bits (including parity).

6.3 Test/Debug

Incorporated into the data cache is built–in self–test (BIST) hardware. While the BIST engine is mainly used for manufacturing test of the data cache arrays, diagnose access to the data cache also uses the BIST hardware. Diagnose operations on the data cache allow the reading or writing of an arbitrary tag, dirty, or data array entry.

Prior to executing a BIST operation or performing any diagnose operations, the data cache must be placed into test mode. When the data cache is in test mode, it is effectively disabled. Because the store queue may contain entries to be written to the data cache, the store queue must be flushed before placing the data cache into test mode. The recommended method for flushing the store queue is to execute two store instructions to a side–effect free location in I/O space. The requirement to flush the store queue can be relaxed if no stores to cachable memory space have been executed since the processor was last reset. In addition, there must not be any data cache misses pending when test mode is enabled. The best way to ensure this is to execute a SYNC instruction prior to placing the data cache into test mode.

6.3.1 Diagnose and Built–In Self–Test Registers

There are three sets of registers associated with the data cache diagnose and BIST hardware: address/configuration, write data, and read/LFSR data. Except for address and configuration these sets encompass multiple physical registers.

6.3.1.1 Address/Configuration Register

This is the DC_BIST_ADDR_CFG diagnose register. It is composed of several fields, all of which are read/write:

- **TEST_MODEH** – When set to a one, this bit places the data cache into test mode. The store queue must be flushed and a SYNC must be executed before this bit is set to a one.

- **BIST_ENH** – This bit determines whether a DC_DIAG diagnose instruction will perform a diagnose operation or a BIST operation. When set to a one, BIST operations are selected.
- **BIST_SSH** – This bit determines whether a BIST operation will cycle through every array entry or whether only a single array entry will be tested (single–step). A BIST single–step will leave the address LFSR incremented (or decremented) to the next address. When set to a one, the BIST engine will perform a single–step when started.
- **DIAG_RDH** – This bit determines whether a DC_DIAG diagnose instruction will perform a diagnose write or a diagnose read. This bit is ignored if BIST_ENH is set to a one. When DIAG_RDH is set to a one, diagnose reads are selected.
- **BIST_FWDH** – This bit determines whether the address LFSR will increment or decrement. When set to a one, the addresses will increment. Note that incrementing the address does not result in the next sequential address being generated.
- **CHKRBRDH** – This bit determines whether diagnose and BIST operations will operate on half a line from two different groups or on a full line for a single group. The portion of the line being operated on (if set to a 0) or the group of the line being operated on (if set to a 1) is determined by LFSR_ADDRH[27].
- **LFSR_ADDRH[17:28]** – This is a 12–bit field that determines what address will be accessed by diagnose and BIST operations. When BIST_ENH is set to a zero (diagnose mode), the first 10 bits (LFSR_ADDRH[17:26]) are used to select a tag and dirty array entry, the first 11 bits (LFSR_ADDRH[17:27]) are used to select a left and right data array entry, and the 12th bit (LFSR_ADDRH[28]) is used to select which tag and dirty group is written on diagnose writes. If CHKRBRDH is set to a zero, then LFSR_ADDRH[27] selects between reading and writing doublewords 0 and 2 of both groups (if LFSR_ADDRH[27] == 0) or doublewords 1 and 3 of both groups (if LFSR_ADDRH[27] == 1). If CHKRBRDH is set to a one, then LFSR_ADDRH[27] selects between reading and writing all doublewords of group 0 (if LFSR_ADDRH[27] == 0) or all doublewords of group 1 (if LFSR_ADDRH[27] == 1). When BIST_ENH is set to a one (BIST mode), the 12–bits in this field behave as a 12–bit linear feedback shift register (LFSR). The CHKRBRDH bit determines whether the 11th address bits on the two data arrays will be set to the same value (if CHKRBRDH is 0) or complementary values (if CHKRBRDH is 1). The polynomial for ‘‘incrementing’’ the address (when BIST_FWDH is set to 1) is $P(x) = x^{12} + x^7 + x^4 + x^3 + 1$ and the polynomial for ‘‘decrementing’’ (when BIST_FWDH is set to 0) is the inverse, $P^*(x) = x^{12} + x^9 + x^8 + x^5 + 1$.

6.3.1.2 Write Registers

These registers contain the data to be written into the tag, dirty and data arrays on a diagnose write operation or during a BIST operation. These registers are write–only.

- **DTAG_IN** – This register contains the tag, tag parity, dirty and dirty parity bits to be written. On a diagnose write, the group to be written is selected by LFSR_ADDRH[28]. On BIST operations, both groups are written with the same value. The parity bits are even.
- **DC_DW0_IN** – This pair of registers contains the data to be written to either doubleword 0 of group 0 (if LFSR_ADDRH[27] == 0) or doubleword 1 of group 1 (if LFSR_ADDRH[27] == 1).
- **DC_DW1_IN** – This pair of registers contains the data to be written to either doubleword 0 of group 1 (if LFSR_ADDRH[27] xor CHKRBRDH == 0) or doubleword 1 of group 0 (if LFSR_ADDRH[27] xor CHKRBRDH == 1).

- **DC_DW2_IN** – This pair of registers contains the data to be written to either doubleword 2 of group 0 (if LFSR_ADDRH[27] == 0) or doubleword 3 of group 1 (if LFSR_ADDRH[27] == 1).
- **DC_DW3_IN** – This pair of registers contains the data to be written to either doubleword 2 of group 1 (if LFSR_ADDRH[27] xor CHKRBRDH == 0) or doubleword 3 of group 0 (if LFSR_ADDRH[27] xor CHKRBRDH == 1).
- **DC_DW0P_IN** – This register contains the two bits of even parity to be written associated with the data in DC_DW0_IN. There is one bit of parity per 32 bits of data.
- **DC_DW1P_IN** – This register contains the two bits of even parity to be written associated with the data in DC_DW1_IN. There is one bit of parity per 32 bits of data.
- **DC_DW2P_IN** – This register contains the two bits of even parity to be written associated with the data in DC_DW2_IN. There is one bit of parity per 32 bits of data.
- **DC_DW3P_IN** – This register contains the two bits of even parity to be written associated with the data in DC_DW3_IN. There is one bit of parity per 32 bits of data.

6.3.1.3 Read/LFSR Registers

These registers contain the data read from the tag, dirty and data arrays on a diagnose read operation. They contain the signature data of the tag, dirty and data arrays after a BIST operation has completed. These register are read/clear. Writing any value to any of these registers results in the entire set being cleared to zero. During a BIST operation, each of these registers functions as an LFSR signature analyzer. The characteristic polynomial for the tag registers is $P(x) = x^{23} + x^5 + 1$. The characteristic polynomial for the data registers is $P(x) = x^{66} + x^{10} + x^9 + x + 1$. The tag polynomial incoporates tag, tag parity, dirty and dirty parity (23 bits). The data polynomial incorporates data and data parity (66 bits).

- **DTAG0_OUT** – This register contains the diagnose read or signature data from the group 0 tag, tag parity, dirty and dirty parity bits.
- **DTAG1_OUT**– This register contains the diagnose read or signature data from the group 1 tag, tag parity, dirty and dirty parity bits.
- **DC_DW0_OUT** – These registers contain the diagnose read data from either doubleword 0 of group 0 or doubleword 1 of group 1 (selected by LFSR_ADDRH[27]). It contains signature data from the upper half of the left data array.
- **DC_DW1_OUT** – These registers contain the diagnose read data from either doubleword 1 of group 0 or doubleword 0 of group 1 (selected by the exclusive–or of LFSR_ADDRH[27] with CHKRBRDH). It contains signature data from the upper half of the right data array.
- **DC_DW2_OUT** – These registers contain the diagnose read data from either doubleword 2 of group 0 or doubleword 3 of group 1 (selected by LFSR_ADDRH[27]). It contains signature data from the lower half of the left data array.
- **DC_DW3_OUT** – These registers contain the diagnose read data from either doubleword 3 of group 0 or doubleword 2 of group 1 (selected by the exclusive–or of LFSR_ADDRH[27] with CHKRBRDH). It contains signature data from the lower half of the right data array.
- **DC_DW0P_OUT** – This register contains the two diagnose read or signature parity bits associated with the data in DC_DW0_OUT. This register is part of the same LFSR as DC_DW0_OUT.

- **DC_DW1P_OUT** – This register contains the two diagnose read or signature parity bits associated with the data in DC_DW1_OUT. This register is part of the same LFSR as DC_DW1_OUT.
- **DC_DW2P_OUT** – This register contains the two diagnose read or signature parity bits associated with the data in DC_DW2_OUT. This register is part of the same LFSR as DC_DW2_OUT.
- **DC_DW3P_OUT** – This register contains the two diagnose read or signature parity bits associated with the data in DC_DW4_OUT. This register is part of the same LFSR as DC_DW3_OUT.

6.3.2 Diagnose Read Operations

A Level 1 data cache diagnose read operation will be initiated when a DC_DIAG instruction is executed with TEST_MODEH set to one, BIST_ENH set to zero and DIAG_RDH set to one. LFSR_ADDRH[17:27] will be used to address the tag, dirty and data arrays. The data read from the array entries addressed will be loaded into the read registers (DTAG0_OUT, DTAG1_OUT, DC_DW0_OUT, DC_DW1_OUT, DC_DW2_OUT, DC_DW3_OUT, DC_DW0P_OUT, DC_DW1P_OUT, DC_DW2P_OUT, DC_DW3P_OUT). The value set into CHKRBRDH determines whether the right data array entry is read in doubleword or checkerboard mode.

6.3.3 Diagnose Write Operations

A Level 1 data cache diagnose write operation will be initiated when a DC_DIAG instruction is executed with TEST_MODEH set one, BIST_ENH set to zero and DIAG_RDH set to zero. LFSR_ADDRH[17:27] will be used to address the tag, dirty and data arrays. The data in the write registers (DTAG_IN, DC_DW0_IN, DC_DW1_IN, DC_DW2_IN, DC_DW3_IN, DC_DW0P_IN, DC_DW1P_IN, DC_DW2P_IN, DC_DW3P_IN) will be written into the array entries addressed. The value set into CHKRBRDH determines which whether the right data array entry is written in doubleword or checkerboard mode. LFSR_ADDRH[28] determines which tag and dirty group is written.

6.3.4 BIST Operations

A Level 1 data cache BIST operation will be initiated when a DC_DIAG instruction is executed with TEST_MODEH and BIST_ENH both set to one. A BIST operation will take multiple cycles to execute (3 cycles if BIST_SSH is set to 1 and over 12000 cycles if BIST_SSH is set to 0). The BIST operation, once initiated by the DC_DIAG instruction will run in the background, that is, will run while allowing the cpu pipeline to execute instructions in parallel. A SYNC instruction will force the cpu to halt until any pending BIST operations are completed. Because of the this ability to run BIST operations in the background, both instruction cache and data cache BIST operations may be executed in parallel.

6.3.4.1 BIST Algorithm

While a discussion of the theory of LFSR signature analysis and self–test is beyond the scope of this document, the LFSR hardware causes addresses to be automatically generated in a pseudo–random order such that addresses for all of the entries in all of the arrays will be created. If BIST_SSH is set to a zero, a single BIST operation will cause all 2048 unique addresses to be generated. For each address, an entry is selected from each array. The data from those entries is read into the signature analysis registers. Then the entries are written with the data in the write registers conditionally based on the least or two least significant bits of the address LFSR. Then the data from the entries is read into the signature analysis registers again. Finally the address is incremented or decremented. If BIST_SSH is set to a one, the operation is similar except that only one read–write–read cycle is performed and the LFSR address will only be incremented or decremented once. Because an n–bit LFSR will only generate 2^n–1 unique addresses (address 0 is skipped), the LFSR address generator is 12–bits wide, even though 2^{11} is 2048. The least significant LFSR bit (LFSR_ADDRH[28]) is not used to address the arrays but is used as a write mask bit. Since the 12–bit LFSR will generate 4095 addresses (of which there are only 2048 unique 11–bit addresses), the write mask bit ensures that every unique address is written to only once by allowing array writes only when the 12th bit is a one. With this scheme, most entries will be read from more than once. In addition, since there are only half as many tag and dirty array entries as there are data array entries, tag and dirty array writes only occur when the 11th bit (LFSR_ADDRH[27]) is a one. This write masking behavior does not occur for diagnose reads and writes. The proper starting value for a forward pass is DC_BIST_ADDR_CFG = 0x00194c80. The proper starting value for a reverse pass is DC_BIST_ADDR_CFG = 0x00180010.

7. Memory & SLC

7.1 Queues

7.1.1 Memory Controller and SLC Transaction Input Queue Block

The input queue block is designed to enable parallelism, reduce latency for CPU reads, and buffer transactions from the CPU to reduce hang states. Primarily the input queue block improves performance of CPU transactions, however it also receives read and write requests from the DMA control block.

The DMA control block has a cache structure that buffers DMA accesses and improves DMA read latency via a combination of the cacheing, and prefetching from memory.

PA-RISC is mostly strongly ordered. Strongly ordered means that any agent performing accesses to memory must not be able to detect any out of order processing utilized by the MIOC to improve performance. PA-RISC allows two exception to the strong ordering rule.

Posting of castouts to memory may be delayed. Software is responsible for forcing castouts to memory with a sync operation when necessary.

A new feature known as accelerated I/O (aio) allow I/O operations to become out of order with respect to memory operations. Again, software is responsible for issuing sync operations when necessary.

The input queue block exploits both of these exceptions to improve CPU memory read latency.

Functionally, the queue section consists of six different queues, four for transaction addresses, and two for transaction data, as illustrated in Figure 1.

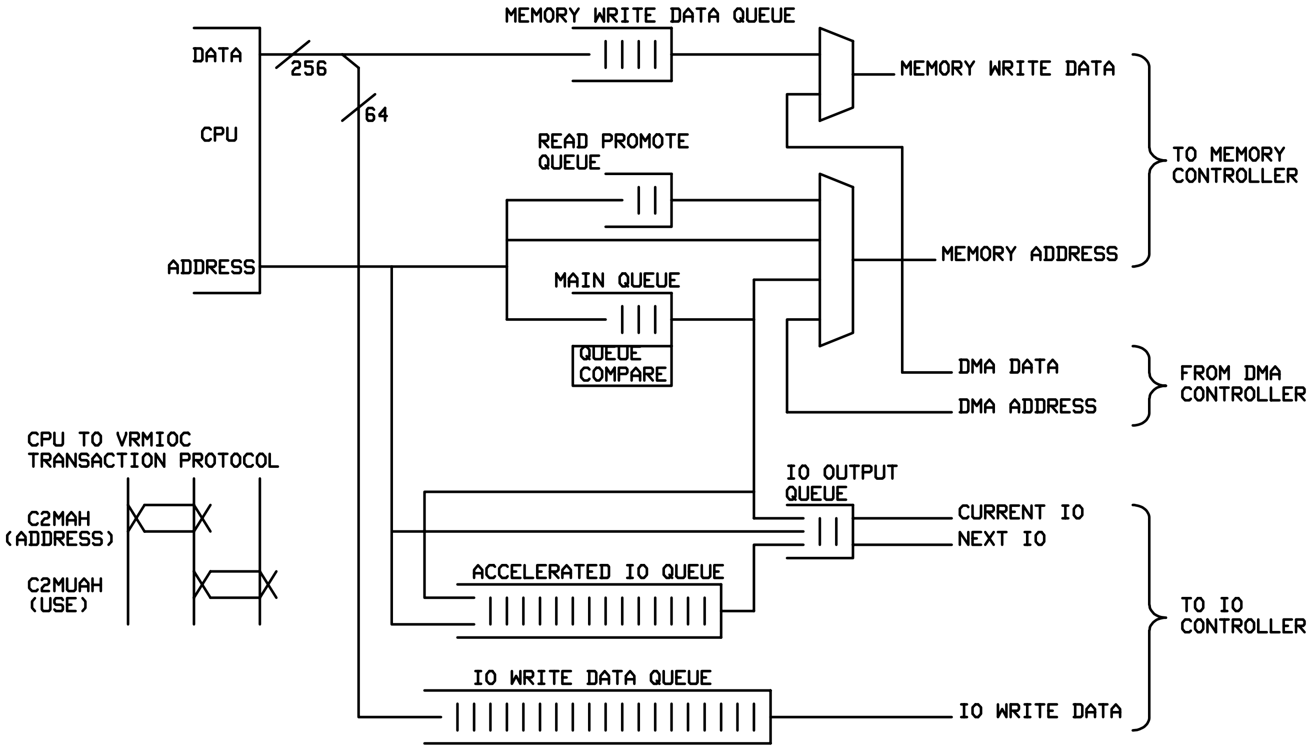


Figure 1. MIOC Queues

7.1.2 Three-Entry Main Address Queue

This is the default queue for incoming transactions, and may hold any type of transaction. All entries in this queue have an associated comparator to detect address conflicts. The CPU to VRMIOC protocol has no provision for deferring a transaction once it has been issued. Hence, as long as the VRMIOC is signaling not queue full to the CPU, there must be at least one free entry in the main queue, even if there are free entries in the other queues.

7.1.3 Two-Entry Read Promote Queue

The read promote queue may contain CPU memory read transactions. These may be copyins, or uncached loads. Upon receipt of a new transaction address, the address is compared with any valid entries in the main queue. If there are no conflicts, and all valid transactions in the main queue are castouts, and there is room in the read promote queue, the new transaction will be placed in the read promote queue. Once a transaction becomes resident in the read promote queue it becomes strictly prioritized over anything in the main queue.

7.1.4 Two-Entry I/O Output Queue

The I/O output queue may contain any type of I/O transaction. The top two entries are exposed to the I/O Controller to facilitate sequentiality compare for GSC writev operations. Transactions may enter the I/O output queue directly from the CPU, from the main queue, or from the accelerated I/O queue.

If the main queue is empty, unaccelerated I/O transactions will be placed in the I/O output queue. As more transactions are received, they will back up into the main queue. If there are any transactions in the main queue, unaccelerated I/O transactions must be placed behind them in the main queue. This rule maintains strong ordering. When I/O transactions find themselves at the head of the main queue, they may move to the I/O output queue at the maximum rate of one per state.

7.1.5 Fourteen-Entry Accelerated I/O Queue

This queue may receive transactions marked as accelerated by the CPU. Only I/O write transactions are allowed to accelerated. This condition is necessary to guarantee the strict FIFO data return ordering of the CPU.

Aio operations are allowed to execute out of order with respect to memory transactions, but must remain in order with respect to any other I/O. That is, the I/O stream must remain in order with respect to itself, regardless of the mix of aio transactions, and normal I/O transactions.

Upon receipt of an aio transaction, the queue controller looks to see if there are any other I/O transactions in the main queue. If not, it then looks to see if there is room in the I/O output queue. If so, the transaction is placed there. If the I/O output queue is full, it then looks for room in the aio queue, and stores it there if possible. Finally, if the aio queue is full, the transaction will be stored in the main queue.

When the I/O controller accepts a new transaction, room is made in the I/O output queue. The I/O output queue can accept a new entry from the aio queue, the main queue, or from possibly directly from the CPU. To maintain I/O self relative order, the I/O output queue prioritizes to the aio queue, then the main queue, then directly from the CPU.

If the movement of an entry from the aio queue to the I/O output queue creates room in the aio queue, a new entry in the aio queue may come from the main queue, or possibly from the CPU, but is prioritized to the main queue to maintain ordering.

Under heavy aio write burst conditions as might be expected from graphics workloads, an aio write may be held off by queue full conditions, then as entries are retired by the I/O controller, entered into the main queue, work its way through the main and into the aio queue, through the aio queue into the I/O output queue, and finally accepted by the I/O controller.

7.1.6 Memory Write Data Queue

The memory write data queue may contain the data for up to four transactions. The data size may be up to thirty two bytes (linesize), but may also be eight bytes (doubleword), or sub doubleword. The less than linesize quantities result from uncached stores.

The input bus into the memory write data queue is 256 bits wide, and an entire cache line is inserted in one state. Uncached store operations consume an entire linesize entry, although only part of it is ultimately written to memory.

7.1.7 I/O Write Data Queue

There is a nineteen entry I/O write data queue. Each entry is a doubleword. Each transaction consumes a full entry, even if the transaction size is less than doubleword.

Both data queues are strictly FIFO. Even for aio transactions where the address may follow a circuitous path through the main queue, the aio queue, and the I/O output queue. This simplifying strategy is correct because memory and I/O transactions are always FIFO with respect to themselves.

The data queues have no full or empty controls/indicators, i.e. flow control. Each is large enough to store the maximum number of transactions that may be stored in the corresponding address queues.

The CPU to MIOC protocol supports only a strict FIFO protocol with respect to read return data. With respect to each other, memory and I/O reads are processed strictly in order, always.

7.1.8 Inserting Queue Entries

The processor pipeline fundamentally knows the address of a transaction one state before it knows miss, abort, etc. status. On every state the CPU core sends an address, and an indication that it might validate the transaction in the next state. Two processes are performed by the VRMIOC during this state. If the memory and second level cache controllers are idle, and the ”might validate” hint are true, the address is bypassed directly to the SLC and MC address busses. Meanwhile, queue control logic examines the transaction to determine into which queue the transaction should be inserted. If the transaction is validated on the next state, one of two things happen. If VRMIOC was idle, bypassing and the transaction is to a memory address, then the transaction immediately enters the memory and SLC controllers and is never queued. Otherwise the transaction is entered into the appropriate queue as was determined in the previous state.

7.1.9 Retiring Queue Entries

The memory and SLC controllers are the consumers of main and read promote queues. The I/O controller is the sole consumer of the I/O output queue. When a transaction reaches the head of its respective queue, it will be accepted by the corresponding controller. The memory and SLC controllers always accept a new transaction simultaneously. When a transaction is consumed, the respective controller signals the queue controller, and the corresponding queue is advanced. The queue controller receives one advance signal from the I/O, and a unified advance from the SLC and memory controllers. The queue controller determines which queue to advance. Logic in the queue controller also gates the acceptance of transactions from the main/read promote queue, and the I/O output queue to maintain ordering between transactions bound for memory or I/O. This logic takes aio into account and allows simultaneous consumption of transactions from main/read promote queue, and the I/O output queue.

DMA requests have highest priority for the memory and SLC controllers. DMA requests are almost always serviced next. Some exceptions are made to support bypassing.

7.2 Memory Controller

Contained within the 7300LC is a main memory controller. The memory controller is complete and requires only buffers and DRAM to build a system.

The 7300LC's built in memory controller is very flexible and programmable to support a wide range of system options:

- Logical support for 4M, 16M, 64M, and 256M DRAM densities
- FPM or EDO DRAM
- 64(72) or 128(144) data bus width options
- Optional SEDC error control
- Up to 16 physical slots
- Highly programmable for optimal memory performance across a wide range of core frequencies
- Full support of 3.3 and 5.0 volt legacy DRAM.

7.2.1 DRAM Interface Signals

[D]DRD[0:127] Data bus connecting the 7300LC with the DRAMs. If system is not configured for SLC, 3.3V DRAMs may be connected directly to DRD, otherwise a FET switch must be used in between the CPU/SLC SRAM and the DRAMs. Creates the DDRD bus. When dwmode == 0, DRD is configured to be 64 bits, in which case DDRD[0:63] are used.

DRA[0:13]
DRA13LO Multiplexed DRAM address bus.

[D]MECC[0:15] Two ECC bytes corresponding to DRD[0:63], and DRD[64:127]. MECC[0:7] is used when dwmode == 0. External timing of DRD and MECC are identical.

ROW[0:3] DRAM RAS strobes. Configurable polarity.

COL[0:3] DRAM CAS strobes. Configurable polarity.

MWRITE[0:1] DRAM WRITE. Configurable polarity.

MOE[0:1] DRAM Output Enable. Configurable polarity. Logically identical, two copies improve AC performance.

7.2.2 Address Decoding

Up to sixteen banks of DRAM are supported. Each bank can have its own row/column mux function. Each bank can range in size from 8M to 512M bytes. Enough flexibility is provided such that any combination of banks of any size can be configured into a contiguous real address space.

Associated with each bank is a comparator and two programming registers, MIOC_MEM_COMP[0:15], and MIOC_MEM_MASK[0:15]. The MIOC_MEM_COMP registers contain bits 0:8 of the base address of the bank. The MIOC_MEM_MASK registers contain three fields. A three bit field selecting one of five row/column multiplex functions. A six bit field that specifies masking bits 3:8 of the base address programmed in the corresponding MIOC_MEM_COMP register. This is used to program the size of the bank. And finally, a single bit that enables/disables the comparator/bank.

Four ROW (RAS), and four COL (CAS) signals are generated and driven to the memory banks. Logically, ROW and COL are arranged in a 4x4 array. Each memory bank is connected to a unique combination of the ROW and COL lines. During normal operation only one ROW and one COL signal will be asserted. Only the memory bank connected to the unique combination of ROW and COL will perform the intended operation. Some unselected banks in systems with more that four memory banks will receive RAS only cycles, and some CAS only cycles.

7.2.3 Row/Column Muxing Function

The row/column muxing function is programmable on a per bank basis. For each function, the row bits are always the same. This allows row address bypassing before the lookup function is completed.

The table below shows how the real address bits are assigned to the row and column addresses. The function is dependent on dwmode, and the contents of MIOC_MEM_MASK[x][0:2] for the corresponding memory bank.

In the following table, row and col bits show the number of bits allowed in the row and column address. For example, in mux option 000, dwmode == 0 mode, 10x10 and 10x9 DRAMs are supported

.			**DRA[0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13]**	**DRA13LO**
			real address bits														
row add			5	6	8	9	10	11	12	13	14	15	16	17	18	19	19
	bits																
	row	**col**															
singlewide, dwmode == 0																	
col add[000]	10	[9:10]					9	20	21	22	23	24	25	26	27	28	NA
col add[001]	11	[9:11]				7	8	20	21	22	23	24	25	26	27	28	NA
col add[010]	12	[9:12]			5	6	7	20	21	22	23	24	25	26	27	28	NA
col add[011]	13	[10:13]		3	4	5	7	20	21	22	23	24	25	26	27	28	NA
col add[100]	14	[10:12]			3	4	7	20	21	22	23	24	25	26	27	28	NA
doublewide, dwmode == 1																	
col add[000]	10	[8:10]					8	9	20	21	22	23	24	25	26	27	27
col add[001]	11	[8:11]				6	7	8	20	21	22	23	24	25	26	27	27
col add[010]	12	[8:12]			4	5	6	7	20	21	22	23	24	25	26	27	27
col add[011]	13	[9:12]			3	4	5	7	20	21	22	23	24	25	26	27	27
col add[100]	14	[9:11]				3	4	7	20	21	22	23	24	25	26	27	27

7.2.4 About DRA13LO

CPU line read requests are satisfied critical doubleword first, followed by three more doublewords, wrapping on the aligned 32 byte cache line boundary if necessary.

When dwmode == 0, real address bits 27 and 28 are generated by adding the initial or base address with the doubleword count as each doubleword is read from the DRAM.

When dwmode == 1, two doublewords are read from DRAM simultaneously. DRA[13] and DRA13LO are used to address the two doubleword sides of the array separately such that data is returned to the CPU in the correct order.

DRA[13] is connected to the DRAMs associated with the most significant doubleword, i.e. DRD[0:63]. DRA13LO drives the DRAMs associated with the least significant doubleword, i.e. DRD[64:127].

DRA[13] and DRA13LO are calculated as follows:

```
        mem_add[27:28] = base_add[27:28] + count[0:1]
        DRA[13] = dwmode ? mem_add[27] exor mem_add[28] : mem_add[27]
        DRA13LO = mem_add[27]
```

count is incremented by two after each quadword access.

When dwmode == 1, the sequences will be:

	{DRA[13], DRA13LO}	
base_add[27:28]	**count == 0**	**count = 2**
0b00	0b00	0b11
0b01	0b10	0b01
0b10	0b11	0b00
0b11	0b01	0b10

7.2.5 Input Transaction Types

Memory controller transactions are received from the CPU, and the DMA controller. Since error control is performed across 64 bits (doubleword), transactions are in terms of doublewords. All writes of less than a complete doubleword force a read-modify-write (RMW) cycle. Note that these are not ”DRAM read-modify-write cycles”, but a DRAM read followed by a page mode early write cycle. The 7300LC memory controller never performs DRAM read-modify-write cycles.

For the CPU, read transaction sizes are cache lines, sub and one word reads, and double word reads. Write transactions sizes are cache lines (castouts), sub and one word writes and double word writes. A special load_and_clear operation reads a word from memory, returns the word to the CPU, then atomically zeros the word and writes it back to memory.

Since the minimum read size is eight bytes, the memory controller always returns at least a doubleword to the CPU, even if the CPU only requested a single or sub word read. The CPU selects the requested word, and discards the remainder.

The DMA controller is allowed to read or write 1,2,3, or 4 doublewords. To support sub doubleword writes, the DMA controller includes eight byte enable signals with each transaction. The byte enables are defined for the last doubleword of the write transaction. For example, if a three doubleword transaction were sent with byte enables 0b10101010, the entire first two doublewords would be written, and then a RMW cycle would be performed to read the current contents of memory, perform error correction, merge in every other byte as defined by the byte enables, calculate the new error correction code, then write the data to memory.

The memory controller examines each transaction received and decides how to map them into DRAM cycles. Page mode cycles are used whenever possible. The state of the dwmode bit is used to figure out how many DRAM cycles to perform. For example, if dwmode == 0, then four page mode read cycles will be performed for a cache line read. If dwmode == 1, then only two page mode cycles are necessary.

7.2.6 DRAM Cycle Types

All transactions are satisfied with three basic DRAM cycles. Refresh is performed with CAS-before-RAS cycles. DRAM reads use the basic DRAM read cycle, and writes use ”early write cycles”. An ”early write cycle” is one for which the DRAM write line MWRITE, is asserted before CAS.

Page mode is used whenever possible. Reads and writes of greater than doubleword for dwmode == 0, and four words for dwmode == 1, are always satisfied with a series of page mode cycles. Once a transaction is begun, it will continue uninterrupted until completed. The maximum number of CAS cycles per transaction is four when dwmode == 0, and two when dwmode == 1.

If the memory controller receives a stream of transactions to the same page (4K system page), it will string them together within a single page mode sequence. This transaction ”streaming” is independent of the requester (CPU or DMA), and may also switch back and forth between read and write any number of times.

Once the current transaction completes, several things can happen. If refresh has been requested, then the DRAM will be de-RASed precharged, and a refresh cycle will be performed. If the next transaction is for a different page, then de-RAS, precharge, re-RAS is performed. If the input queue is empty, then the precharge policy is invoked.

7.2.7 Cycle Transitions

As noted above, several transactions may be satisfied in page mode within a single assertion of RAS. These may be any combinations of read or write cycles. For read to read, and write to write combinations which are received very close together, there will be no pause between the transactions. The memory controller is capable of generating very long streams of full speed page mode reads and page mode writes. If there there is a switch from DRAM read to DRAM write, several extra states are introduced to allow time for the bus to transition from DRAM driver to CPU driver. From writes to reads, no extra time is necessary because the CPU drivers turn off very quickly.

7.2.8 Precharge Policy

Once the memory controller has completed a transaction, and becomes idle, it must decide whether to keep the DRAM page open (RAS asserted), or close the page (deassert RAS). This policy is programmable via the lzpwait field. Depending on lzpwait, the controller will close the page as soon as possible (eager precharge), keep the page open until refresh or a new transaction force a de-RAS command (lazy precharge), or something in between.

lzpwait is a five bit number. If lzpwait == 0, then eager precharge is selected. lzpwait == 31 selects lazy precharge. For all other values, the controller will wait lzpwait states after the last assertion of CAS before closing the page.

7.2.9 Refresh

CAS before RAS refresh is used. A refresh cycle may be recognized by the assertion of all four COL lines, then all four ROW lines. The system design must take the resulting refresh current spike into account.

The refresh frequency is generated by a fifteen bit counter operating at the CPU clock rate. Refresh requests are generated when this counter overflows. Refresh frequency is controlled by setting a programmable seed value into the counter on overflow. Refresh frequency is the ones complement of the value set into the seed register, plus two. A side affect of writing to the seed register is causing the new value to load into the refresh counter without waiting for a counter overflow. Refresh cannot be disabled, but can be prevented by periodic writes to the seed register.

7.2.10 Memory Write Data

All memory (SLC and DRAM) writes are performed under the control of the memory controller. The memory controller is responsible for extracting data from the proper data queue (CPU or DMA) moving it across the chip and through the ecc generation block and into the pads.

The SLC is a write through design. All DMA write transactions, and with one exception, CPU writes including load_and_clear cause the line if resident in the SLC, to be marked invalid. The exception is if the write is a CPU castout, and the configuration bit up4cout == 1. In this case, the SLC controller will write the line into the SLC coincident with the memory controller writing the line into DRAM. The memory controller sends a signal to the SLC controller when write data is valid on the data bus. The SLC controller performs the appropriate write sequencing.

Castout data from the L1 data cache is set into the data queue via a 256-bit dedicated connection. Due to the “Checkerboard” arrangement (sometimes referred to as “Violet trick”) of the L1 cache this data will be either doublewords {0, 1, 2, 3} or {1, 0, 3, 2}. CPU to memory controller address bit c2mah1n[29] indicates the order of the data.

The memory controller sorts this out when it moves the data from the queue to the pads. The data queue output is 64 bits and strictly FIFO. In doublewide mode the data is properly arranged by setting it into the correct doubleword when demuxed into the quadword external memory bus interface. For singlewide mode, bit 27 is manipulated such that doublewords are written to memory in the correct sequence.

7.2.11 Basic Memory Controller Transaction Sequencing

Operation of the memory controller is built around processing transactions through the address datapath, illustrated in Figure 2.

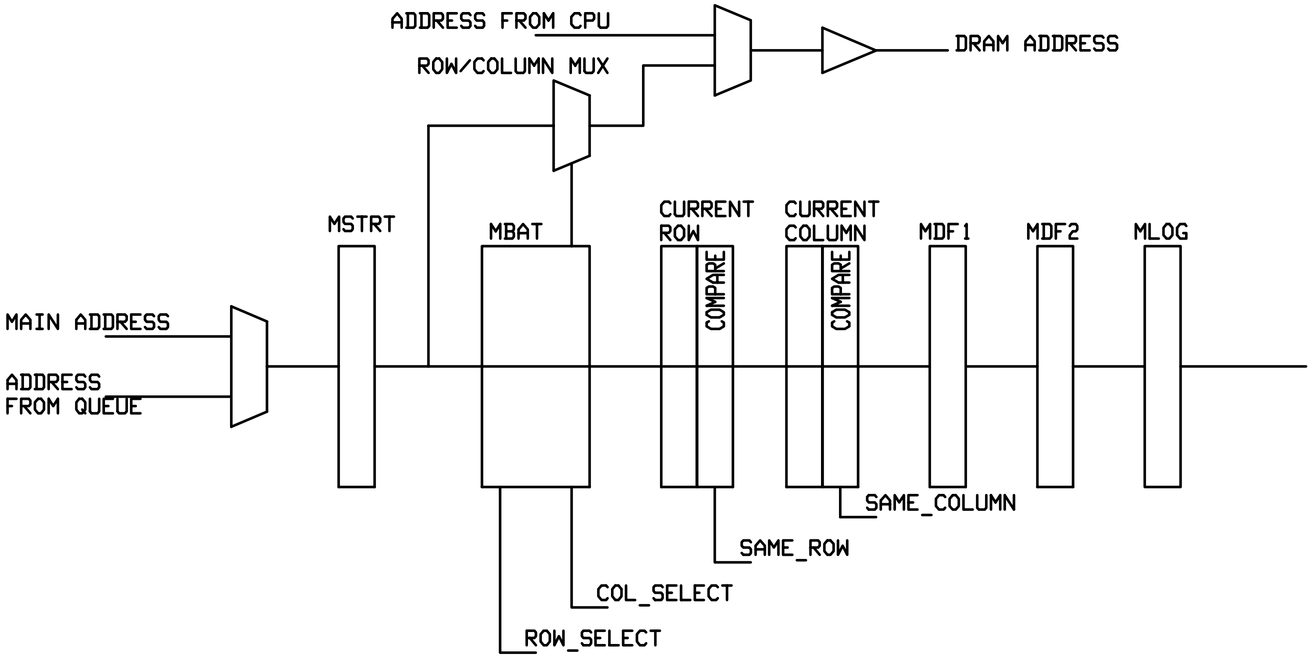


Figure 2. Memory Transaction Address Path

The memory controller receives new transactions from the queue block, or from the main address path. Most addresses are received from the queue block, but are taken from main address under certain conditions where the memory controller falls behind because the SLC has been able to process transactions without intervention from the memory controller (SLC cache hits for CPU copyins).

The transaction finds itself in the MSTRT stage. From here it is driven to the DRA pads as well as the Memory Bank Attribute Table (MBAT) block. The MBAT contains a CAM like structure which determines which ROW and COL lines to drive, and hence which memory bank will be accessed for this transaction. The MBAT also determines which row/column mux function, as well as the actual hardware to perform the row/column muxing.

When confronted with a new transaction, the memory controller first endeavors to strobe the correct row address into the DRAMs. From the memory controller’s logical point of view, there is only one active DRAM row address at any given time. All address bits above the page offset are considered to be part of the row address, even though this may span many banks of DRAM. The MBAT takes care of mapping the correct memory bank for a given address.

If there is currently no active row address, then a new row address cycle is started. If there is currently an active address, then the same page comparator results are used to determine if a precharge followed by a re-ras cycle is necessary. If the new transaction is to the current page, then the column part of the cycle may proceed immediately.

For each stage of the address datapath pipeline, there is a corresponding stage of control information. Physically, the control pipeline is implemented in the standard cell control block.

Before the assert COL command can be given, several factors must come into alignment. The Column Precharge (CP) timer must have completed. If the transaction is a write, then the data must have been placed into the pads and setup to the DRAM for the proper number of number of cycles.

A transaction remains in the MSTRT stage until the last COL command for it is given. Depending on dwmode, the transaction type and size, from one to five column cycles are launched. As each COL command is given a copy of the transaction is set into the MDF1 latch. This includes the control information necessary to direct the data at the end of the column cycle.

One state before the end COL command is given, MDF1 is set into MDF2. Again, MDF2 includes both address and control information necessary to complete the process.

At about the same time the data latch command is given, MDF2 is moved into MLOG. MLOG is the final stage of the pipeline before a possible set into the memory error address register. This register is set as the result of a detected data error for either a DRAM or SLC read. The address portions of MDF1, MDF2, and MLOG serve no purpose other than pipelining the address along for possible logging in the event of an error.

At the same time that the last COL command is given, a new transaction is loaded into the MSTRT register. Comparisons are made, and the controller is ready to start the precharge, or the next COL command at the earliest allowable time.

Moving write data from the data queues to the pads is the critical process for initiating write cycles. This is especially important for this chip due to the size of the chip, internal bus width limitations, and because the data pads are on the opposite side of the chip from the memory controller. In order to reduce the amount of wait, a pseudo independent ”move write data” process becomes active whenever there is a valid write transaction in the MSTRT stage. Since write data is strictly FIFO, and write transactions are always processed in order, it is OK to move data into the pads well before the end of the current transaction. In fact, if the column cycle is configured to be especially long, write data may move into pads well before the end of the column cycle for the previous transaction. This does not cause problems for streams of write cycles because the pads contain separately controllable master slave latches. The masters are loaded as soon as possible, but the data is not moved into the slaves until the previous cycle completes.

7.2.12 Interactions with the SLC

In order to modularize the design, interactions between the SLC and the memory controller were minimized. However, since the SLC and the memory controller share a common data bus some interaction is unavoidable.

It is expected that most traffic will be SLC read hits. It makes sense to optimize SLC hits. When the system is idle, the data bus is configured for SLC reads.

The SLC and memory controller generally receive new transactions simultaneously. Upon receiving a new transaction, an SLC access is initiated. Meanwhile, the memory controller is working on the transaction. If the desired page is already active, the memory controller simply waits until the hit/miss status is determined. If there is a page miss, the memory controller will precharge if necessary, and then RAS up the correct page. It is probable for the memory controller to generate a series of RAS only cycles on a stream of SLC read hits.

For the most part, the SLC controller sequences through data bus configuration changeovers. As mentioned above, it defaults to SLC read mode, but on SLC miss will configure for DRAM reads, then hand control over to the memory controller. For writes, the SLC immediately configures for the DRAMs, and then checks the tags. Once the DRAM controller has completed its operations, it hands control back to the SLC controller which configures the bus back to SLC read mode.

For SLC copyins, data is written directly from the DRAM into the SLC SRAMs at it passes by on the data bus. The SLC controller sets up the SRAMs and then waits for the DRAM controller to indicate data is valid. The same is true for castouts, except the data is sourced from the 7300LC.

7.2.13 EDO DRAM Support

The memory controller supports Fast Page Mode (FPM) as well as Extended Data Out (EDO) DRAMs. There is no EDO mode bit. The basic cycle is the same for both types. Several features were implemented to accommodate EDO.

The data sampling point is programmable with the CAC parameter in the MTCV register. For FPM DRAMs this parameter is set to zero, which means the data is sampled corresponding to the rising edge of cas. The value of this register can be increased so that data is sampled up to seven states after rising cas. For EDO, CAC is normally, but not necessarily programmed so that data is sampled at the next possible falling edge of cas.

If ras is to be deasserted at the end of the current column read cycle, and CAC is greater than one, then deassertion of ras is held off until data is sampled.

FPM DRAMs always turn off their output buffers whenever cas is deasserted. EDO DRAMs will only turn off their output buffers when explicit deassertion of ras or output enable, or assertion of write. At the end of any read transaction, after data has been sampled, the memory controller always deasserts output enable, unless the next transaction is valid, and it is a read, or the read portion of a RMW cycle.

In the early days of EDO, some vendors required longer tAR time for writes than for reads. This is accommodated with independently programmable RRAH, and WRAH.

7.2.14 Miscellaneous

All memory locations must be initialized with correct ECC before error checking is enabled.

DRAM manufacturers require eight refresh cycles following powerup before reliable operation is guaranteed.

7.2.15 Memory Timing Control

There are about 50 DRAM timing parameters. These parameters are interdependent. For example, tPC ~= tCP + tCAS. If either tCP or tCAS are changed, then tPC will follow.

The MTCV register has nine parameters that control the core DRAM cycle characteristics. Generally these nine parameters map onto familiar key DRAM cycle elements, but also affect related DRAM parameters as noted above.

MCBD[0:1]	Memory controller bus divisor. Controls timing for most set up and bus driver change delays.
RAS[0:3]	Min RAS assert time. Normally only significant for refresh cycles.
RP[0:3]	RAS precharge.
RRAH[0:2]	Read cycle row address hold time.
WRAH[0:2]	Write cycle row address hold time.
CAS[0:2]	Min CAS low time.
CP[0:2]	Min CAS precharge time.
CAC[0:2]	Min access time from CAS.
RAL[0:2]	Min ROW address lead time.
REFRESH[0:14]	Ones complement of number of CPU states between refreshes.

There is no external indication of when read data is latched. It may be calculated to be the rising edge of cas plus CAC[0:2] states.

All timing parameters are in CPU states.

Hitachi parlance with the exception of txAR, tOCH, tCHO, and tOEP which are Samsung.

tCHO and tOEP are only guaranteed for page mode read to write transitions.

tRRH, tWCR, tCWL, tRWL, tDHR, tROH, tWP are from HP Spec (part #1818–5528).

			MIN		MAX	
SPEC	MIN	MAX	@120 MHz		@250 MHz	
			States	nS	States	nS
tRC	12 + RAS + RP	2 * REF_INTVRL	12	100	42	168
tRP	4 + RP	–	4	33	19	76
tRAS	8 + RAS	2 * REF_INTVRL	8	67	23	92
tASR	2 + MCBD	–	2	17	4	16
tRRAH	1 + RRAH	–	1	8	16	64
tWRAH	1 + WRAH	–	1	8	16	64
tRAC	4 + RRAH + CP + CAS + CAC	–	4	33	32	128
tRRAD	1 + RRAH	–	1	8	16	64
tWRAD	1 + WRAH	–	1	8	16	64
tRSH	2 + CAS + RAL	–	2	17	16	64
tRAL	3 + CP + CAS + RAL	–	3	25	24	96
tRRCD	2 + RRAH + CP	–	2	17	16	64
tWRCD	2 + WRAH + CP	–	2	17	16	64
tCPRH	3 + CP + CAS + RAL	–	3	25	24	96
tRAR	4 + RRAH + CP + CAS	–	4	33	25	100
tWAR	4 + WRAH + CP + CAS	–	4	33	25	100
tPC	3 + CP + CAS	–	3	25	17	68
tCP	1 + CP	–	1	8	8	32
tCAS	2 + CAS	–	2	17	9	36
tASC	1 + CP	–	1	8	8	32
tCAH	2 + CAS	–	2	17	9	36
tCAC	2 + CAS + CAC	–	2	17	16	64
tAA	3 + CP + CAS + CAC	–	3	25	24	96
tCPA	3 + CP + CAS + CAC	–	3	25	24	96
tRCSH	4 + RRAH + CP + CAS	–	4	33	25	100
tWCSH	4 + WRAH + CP + CAS	–	4	33	25	100
tCRP	4 + RP	–	4	33	19	76
tCAL	3 + CP + CAS	–	3	25	17	68
tCDD	2 * (2 + MCBD)	–	4	33	8	32
tOED	2 * (2 + MCBD)	–	4	33	8	32
tOEA	3 + CP + CAS + CAC	–	3	25	24	96
tOCH	3 + CP + CAS	–	3	25	17	68
tCHO	2 + MCBD	–	2	17	4	16
tOEP	2 * (2 + MCBD)	–	4	33	8	32
tDZC	1 + CP	–	1	8	8	32
tDZO	0	–	0	0	0	0
tRCS	1 + CP	–	1	8	8	32
tRCH	2 + MCBD	–	2	17	4	16
tWCS	1 + CP	–	1	8	8	32
tWCH	2 + CAS	–	2	17	9	36
tDS	1 + CP	–	1	8	8	32
tDH	2 + CAS	–	2	8	9	36
tCSR	2 * (2 + MCBD)	–	4	33	8	32
tCHR	8 + RAS	–	8	67	23	92
tWRP	4 + RP + RAL	–	4	33	26	104
tWRH	13 + RP + RAS + WRAH	–	13	108	40	160
tRPC	3 + MCBD	–	2	17	4	16
tRASP	8 + RAS	2 * REF_INTVRL				
tRRH	5 + RP + WRAH	–	5	42	27	108
tWCR	4 + WRAH + CP + CAS	–	4	33	25	100
tWP	3 + CP + CAS	–	3	25	17	68
tCWL	3 + CP + CAS	–	3	25	17	68
tRWL	3 + CP + CAS + RAL	–	3	25	24	96
tDHR	4 + WRAH + CP + CAS	–	4	33	25	100
tROH	3 + CP + CAS	–	3	25	17	68

Restrictions on MTCV:

CAC <= CP + CAS + 1
MCBD <= 2

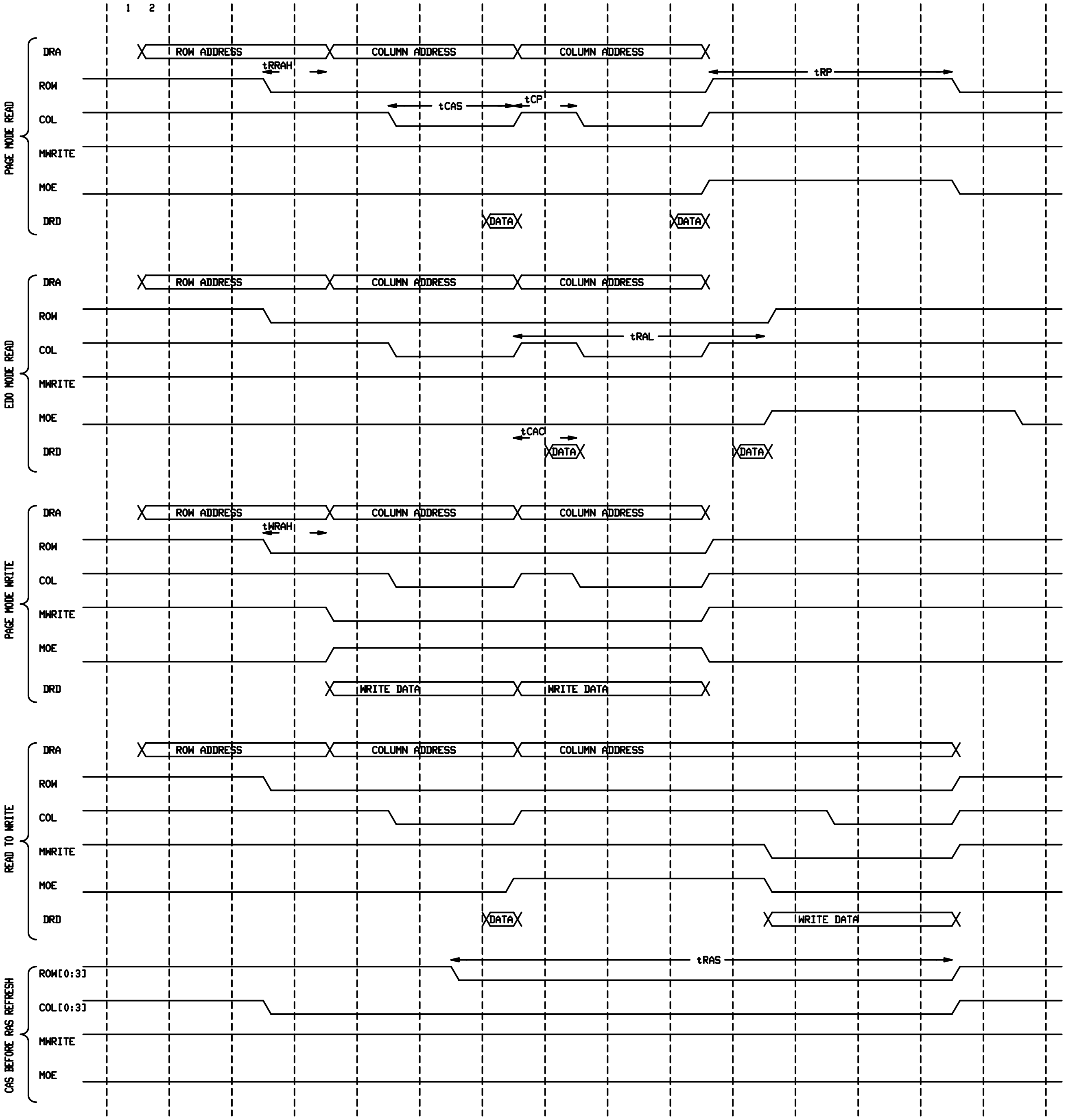


Figure 3. DRAM Timing Diagrams

7.3 Second Level Cache Controller

The 7300LC optionally supports a complete Second Level Cache (SLC) subsystem. The SLC is direct mapped, physically indexed and may range is size from 256K to 64M bytes.

Both arrays, tag and data are constructed with industry available SRAMs, external to the 7300LC processor chip. All logic, comparators, etc. are contained within the 7300LC, and other than the SRAM arrays, no other support componentry is required.

Optimal cost is achieved by sharing the data bus with the DRAM subsystem. The SLC data array SRAMs connect directly to the 7300LC data bus. A FET switch is used to isolate the SLC data array from the DRAM subsystem. The switch is opened when SLC accesses are in progress, and closed for DRAM accesses.

All other SLC tag and address busses connect directly to the 7300LC and are not shared with any other function.

The 7300LC SLC controller is very flexible and is programmable to support a wide range of system configurations:

- 256K to 64M bytes capacity
- 64 or 128 bit data bus (but must be the same as the DRAM subsystem)
- SEDC error control (but only if also configured for the DRAM subsystem)
- Async, Flow-through, or Register to Register mode SRAM independent for tag and data arrays
- Basic timing programmable for 2, 3, or 4 CPU cycles.

7.3.1 SLC Interface

DRD[0:127] Data bus connecting the 7300LC with the SLC data SRAMs. When dwmode == 0, DRD is configured to be 64 bits, in which case DRD[0:63] are used.

MECC[0:15] Two ECC bytes corresponding to DRD[0:63], and DRD[64:127]. MECC[0:7] is used when dwmode == 0. External timing wise DRD and MECC are identical.

SLA[6:27]
SLA27LO
SLATV_13 SLC address bus. Mostly used for both tag and data arrays.

SLDOE[0:1] Second Level cache Data array Output Enables. Logically identical, two copies to improve AC performance.

SLDW[0:1] Second Level cache Data array Write. Logically identical two copies to improve AC performance. Not used with async SRAMs.

SLDWCK[0:1] Second Level cache Data array Write/Clock. Logically identical two copies to improve AC performance. Used as the write line for async SRAMs, and the clock for flow_through or register_register parts.

SLT[0:14] Second Level cache Tag bus. SLT[14] is the parity bit.

SLTOE Second Level cache Tag array Output Enable.

SLTW Second Level cache Tag array Write. Not used with async SRAMs.

SLTWCK Second Level cache Tag array Write/Clock. Used as the write line for async SRAMs, and the clock for flow_through or register_register parts.

DRDCNTL[0:1] FET switch control. Logically identical, two copies improve AC performance. Polarity is configurable.

7.3.2 DRD Buses and Control

The SLC SRAMs connect directly to the 7300LC. Since the 7300LC is not 5V tolerant, the SRAMs must use 3.3V I/O, or use a translator.

For systems configured to support SLC, a FET switch must be placed in between the CPU/SLC SRAM and the DRAM bus. The isolated DRAM bus is known as DDRD. This FET switch serves to isolate SLC parts from the heavily loaded DRAM bus. They also serve as a level translator for compatibility with legacy 5V DRAMs.

Via the FET switch, the DRD and DDRD busses are either connected or isolated. When performing SLC reads, the switch is open so that the SLC SRAMs avoid driving the heavily loaded DRAM bus and can operate at high speed. The SLC controller determines when DRAM is necessary to complete a transaction, deasserts SLDOE[0:1], switches DRDCNTL, and hands control over to the memory controller. When the memory controller has completed its work, it hands control back to the SLC controller, which opens the FET switch, and asserts SLDOE[0:1]. The drdw field in the MIOC_CONTROL register controls how many CPU states are allowed for driver hand off. The memory controller is responsible for managing the DRAM and CPU drivers such that drive fights are avoided when it has control.

7.3.3 SLC Addresses and Tags

The real address is split into two pieces. Any given real address bit is either part of the tag, or part of the SLC address (sometimes called cache index). In order to support multiple second level cache sizes, this split is programmable.

SLC address bits that are programmed not to be part of the address are forced to one. Similarly, tag bits programmed to not be part of the tag, will be forced to one on all tag writes, and tag compares.

Forcing the unused address and tag bits to one is not absolutely necessary, but provides some conveniences for the system designer. For example, if a system were built with a fixed tag array, and optional data array sizes, the full range of SLCs allowed by the tags could be achieved by simply inserting the desired data array DIMM, and programming appropriately.

It is not necessary to connect RAM to masked tag bits. However, a simplification of the design is to always compare the entire tag for match. Since unused tag bits are forced to one for compares, unused tag bits must either connect to RAM, or be tied high with a pullup resistor.

Real address bits 0 through 5 are always part of the tag. Bits 14 through 26 are always part of the tag ”index”. Bits 27 and 28 are used to address doublewords within a cache line, and are only driven to the SLC data array. Real address bits 6 through 13 can programmed to be part of the tag, or part of the index.

Bits 6 through 12 of tagmask register selects whether corresponding bits of the real address are part of the address or tag. If a bit in the tagmask register is one, then the corresponding bit of the real address is part of the cache address, else it is part of the tag.

7.3.4 About Real Address Bit 13, SLA[13], and SLATV_13

The SLC may be configured for either 64 or 128 bit data bus width. This selection is controlled by the dwmode (double wide mode) bit in the MIOC_CONTROL register. dwmode == 0 selects 64 bit data bus width, and dwmode == 1 selects 128 bit data bus width.

Addressing of the tag and data arrays is performed slightly differently for the two bus widths. SLC addressing is set up such that a system designed for doublewide mode can be run in singlewide mode with only configuration bit changes. No rewiring of the motherboard is necessary.

The dwmode bit controls whether bit 13 of the real address is part of the tag, or part of the SLC address. If dwmode == 0, bit 13 is selected to be part of the tag, and real address bit 28 is sent on SLA[13]. If dwmode == 1, then real address bit 13 is part of the address and is sent on SLA[13].

SLA[13] is only connected to the data array. SLATV_13 is a special SLC address bit connected only to the tag array. When dwmode == 0, real address bit 13 is part of the tag, and SLATV_13 is connected to 3.3V. When dwmode == 1, real address bit 13 is part of the SLC address and is driven onto SLATV_13.

	dwmode == 0	**dwmode == 1**
SLA[13]	real_add[28]	real_add[13]
SLATV_13	3.3V	real_add[13]

7.3.5 About SLA27LO

CPU line read requests are satisfied critical doubleword first, followed by three more doublewords, wrapping on the aligned 32 byte cache line boundary if necessary.

When dwmode == 0, real address bits 27 and 28 are generated by adding the initial or base address with the doubleword count as each doubleword is read from the data array.

When dwmode == 1, two doublewords are read from the data array simultaneously. SLA[27] and SLA27LO are used to address the two doubleword sides of the array separately such that data is returned to the CPU in the correct order.

SLA[27] is connected to the SRAMs associated with the most significant doubleword, i.e. DRD[0:63]. SLA27LO drives the SRAMs associated with the least significant doubleword, i.e. DRD[64:127].

SLA[27] and SLA27LO are calculated as follows:

```
mem_add[27:28] = base_add[27:28] + count[0:1]
SLA[27] = dwmode ? mem_add[27] exor mem_add[28] : mem_add[27]
SLA27LO = mem_add[27]
```

count[0:1] is incremented by two after each quadword access.

When dwmode == 1, the sequences will be:

	{SLA[13], SLA13LO}	
base_add[27:28]	**count == 0**	**count = 2**
0b00	0b00	0b11
0b01	0b10	0b01
0b10	0b11	0b00
0b11	0b01	0b10

7.3.6 Data Errors

The SLC may optionally use the EDC hardware, but only if it is also configured and enabled for main memory. Single bit errors are corrected, and all double bit, and three and four bit aligned burst errors are detected.

When an error is detected during an slc read, if no other higher rank errors have been detected, the data and address are logged for later diagnosis.

If a single or multiple bit error is detected during a copyin from main memory to SLC, the SLC controller marks the line invalid in the cache.

7.3.7 SLC Policies

The SLC is direct mapped, real indexed, and writethrough. It is not ”architectural”, and is never allowed to be different than memory.

Copyins are always written into the SLC as the data passes by on its way from the DRAMs to the CPU. If the configuration bit up4cout == 1, castouts will be written into the SLC as the data passes from the CPU to the DRAMs. All other CPU and DMA transactions bypass the second level cache and operate directly out of main memory. If the operation changes memory, the SLC is checked. On hit, the line is marked invalid.

7.3.8 SLC Operation

The SLC controller processes several types of transactions from the CPU and DMA. These transactions are reduced to four major actions described below.

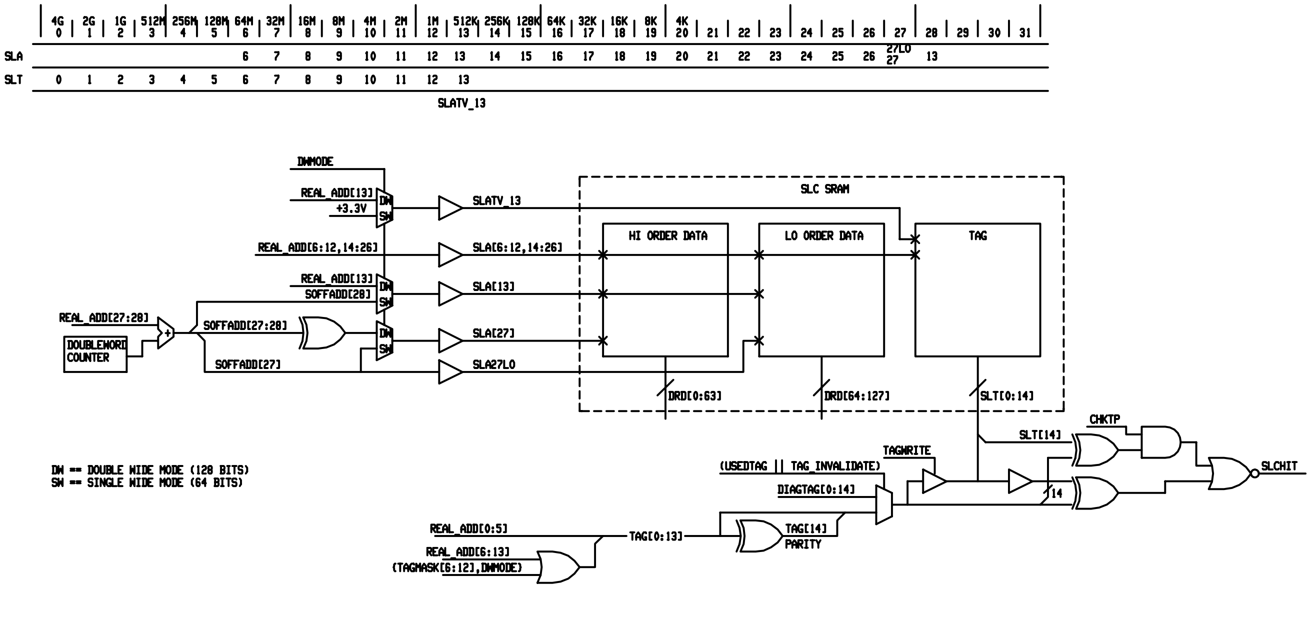
transaction	SLC actions
CPU cached miss	check SLC, copyin in on hit, else SLC miss to DRAM
CPU uncached load	release DRD bus to memory controller DMA read
CPU castout, up4cout == 1	release DRD bus to memory controller, set data into SRAMs during memory writes, update tag
CPU uncached store CPU ld_clr CPU castout, up4cout == 0 DMA write	release DRD bus to memory controller, check SLC, if hit, mark line invalid

When a new transaction is received from the queues, its address is driven onto the SLA bus. Since most of the address bus is shared between the data and tag arrays, data and tag are accessed simultaneously.

When the results of the tag and data become available, if there is a cache hit, data is forwarded onto the CPU. Otherwise miss processing is started, the data bus is changed from SLC mode to DRAM mode, and control is passed to the memory controller.

7.3.9 SLC Tag Operation

The SLC tag is quasi independent from the data array. Since the address bus is mostly shared, tag controller logic depends on the data array logic to present the correct address.



	4G 0	2G 1	1G 2	512M 3	256M 4	128M 5	64M 6	32M 7	16M 8	8M 9	4M 10	2M 11	1M 12	512K 13	256K 14	128K 15	64K 16	32K 17	16K 18	8K 19	4K 20	21	22	23	24	25	26	27	28	29	30	31
SLA							6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27LO 27	13			
SLT	0	1	2	3	4	5	6	7	8	9	10	11	12	13																		
														SLATV_13																		

Figure 4. Address Tag Compare Path

The tag is up to 14 bits wide, plus an optional parity bit. SLT[0:13] corresponds to real address bits [0:13], the parity bit is SLT[14]. The parity bit is calculated such that is SLT[0:14] always has an odd number of ones.

Internal to the 7300 LC, a bidirectional tag bus connects the tag I/O circuitry with the SLC datapath. All manipulations of the tag itself are performed inside the datapath.

For optimal performance, tag compare is performed in the I/O ring and simultaneously driven to the CPU cache controller and the SLC controller. The CPU uses the tag compare result to determine if it can use data bypassed from the internal DRD bus.

For tag checks, a tag is read from the external tag array. At the same time as the tag read results are available on the pad, a compare tag is sent to the I/O ring from the datapath. The compare tag always includes a calculated parity bit. Thus, as far as the tag compare is concerned, parity checking is simply part of the tag compare. At this level a tag parity error simply appears as an SLC miss.

For all tag accesses, the tag is driven back to the SLC datapath where further processing of the tag is performed to determine if there was a parity error. If an error is detected, and there are no previously logged tag parity errors, the errant tag will be logged along with full real address of the tag read.

Except for parity, the tag comparator is insensitive to tag width. Tag compares always operate across the entire tag width. Datapath logic forces ones for all insignificant tag bits for tag writes, and tags sent to the I/O ring for compare. Masked tag bits must either connect to tag SRAM, or be tied high with pullup resistors. The tag parity enable bit, chktp in SLTCV, is sent to the I/O ring and when chktp == 0 will mask the parity bit (SLT[14]) out of tag compares.

The tag is updated whenever a new line is written into the SLC. New lines are written in the SLC on copyin from DRAM, or on castouts if up4cout == 1. In either case the memory controller determines the actual state when data is written to the SLC data array. This is not always consistent with when the tag can be written. The tag update machine is independent of the data write process, and may finish after the data write. If this happens, the SLC controller waits until the tag update process completes before accepting the next transaction.

For uncached stores, ld_clr, DMA write, or castout with up4cout == 0 transactions the SLC transitions the DRD bus to DRAM mode and hands control over to the DRAM controller. At the same time the SLC is checked for hit. If there is a hit, the tag is updated to mark the line invalid.

SLC lines are marked invalid by writing 0xf into SLT[0:3]. SLT[0:3] == 0xf corresponds to an I/O address in PA-RISC. Since I/O transactions are never sent to the SLC, lines tags with bits [0:3] == 0xf will never hit.

The pattern written to the tag is sourced from the DIAGTAG register. DIAGTAG is nominally used for SLC self test, but under normal operating conditions should be programmed with the invalidate pattern. A correct invalidate pattern is DIAGTAG[0:3] == 0xf, DIAGTAG[4:5] == 2bx, DIAGTAG[6:13] = (8bx | {TAGMASK[6:12], dwmode}) and DIAGTAG[14] calculated such that DIAGTAG[0:14] has an odd number of ones. It is recommended that DIAGTAG[0:14] be set to all ones. This works for configurations.

The DIAGTAG register is useful for self test and initialization of the SLC. If the usedtag bit is set to one, the contents of the DIAGTAG register will be used for all tag compares, and all tag writes. DIAGTAG is used for the entire tag, including parity. With DIAGTAG, any pattern may be written into the tags. Also, any pattern may be sourced for a tag compare.

If usedtag == 1, the error logging hardware functions differently. Instead of just logging the most recent tag with bad parity, all parity reads are logged. The results of the tag compare are included with the logged tag. Software using this feature must carefully controller misses from both first level caches to achieve expected results.

7.3.10 DRD Bus Changeover & Control Timing

The DRD bus may be driven from three different sources, the 7300LC, the SLC SRAM, or the DRAM. States are inserted between the deassertion of one output enable to the assertion the next driver.

The drdw[0:1] field of the MIOC_CONTROL register controls most of the driver changeover times. Driver changeover times involving DRAM read to write transitions are controlled by the Memory Timing Control Vector (MTCV).

The idrdcntl field of the MIOC_CONTROL register programs the polarity of the DRDCNTL[0:1] bits.

Symbol	Definition
SOZ	SLC output high impedance
SED	SLC output enabled
DOZ	DRAM output high impedance
DEN	DRAM output enabled
COZ	CPU output high impedance
CEN	CPU output enabled
FOZ	FET switch open
FEN	FET switch closed

Name	Minimum number of states guaranteed by controller	Description
tSOZ2CEN	2 + DRDW	SLC hi-z to CPU output enable
tSOZ2DEN	2 + DRDW	SLC hi-z to DRAM output enable
tCOZ2SEN	2 + DRDW	CPU hi-z to SLC output enable
tCOZ2DEN	1 + CP	CPU hi-z to DRAM output enable
tDOZ2CEN	2 * (MCBD + 2)	DRAM hi-z to CPU output enable
tFOZ2SEN	2 + DRDW	FETSW open to SLC output enable
tFEN2CEN	2 + DRDW	FETSW closed to CPU output enable
tFEN2DEN	2 + DRDW	FETSW closed to DRAM output enable

Conditions the controller does not intentionally create:

CEN && !FEN
DEN && !FEN
SEN && FEN
(CEN && FEN) || (CEN && DEN) || (SEN && DEN)

7.3.11 Second Level Cache Timing Control Vector (SLTCV)

Timing and configuration of the SLC is mostly controlled via several fields in the SLTCV register.

Field	Description
slp	SLC low power mode. Inhibits some address transitions
slbd[0:1]	SLC bus divisor
slstrttag[0:2]	Determines when the first tag is sampled
slstrtdata[0:2]	Determines when the first data is sampled
sltcnfg[0:1]	Tag clock configuration
sldcnfg[0:1]	Data clock configuration
avwl	Address valid, write low delay

Since tag and data sides are quasi independent, they should be checked/designed independently.

slbd sets the SLC ”basic_cycle”. basic_cycle is the number of CPU states allowed for each cache read or write cycle. Allowed values for basic_cycle are 2, 3, or 4 CPU states. slbd may be 0, 1, 2, or 3. Basic cycle time is slbd + 2, except for when slbd == 3, in which case it is slbd + 1.

slbd[0:1]	basic_cycle
0	2
1	3
2	4
3	4

sl[t,|d]cnfg[0:1] sets the clock mode. Tag and data clock modes are independent. Clock mode is also independent of slbd. In async mode SL[T,D]WCK is used as the SRAM write line, and SL[T,D]W is unused.

sl[t,d]cnfg[0:1]	mode
0	async
1	undefined
3	ft (flow_through)
4	rr (register_register)

The waveforms below illustrate the relationship between SL[T,D]WCK and SLA for synch SRAM modes (sl[t,d]cnfg[0:1] == {3 or 4}). For async mode (sl[t,d]cnfg[0:1] == 0), SL[T,D]WCK is normally high, and pulses low for one basic cycle for each write.

slbd == 0:

```
Phase clocks    | 2   1 | 2   1 | 2   1 | 2   1 | 2   1 | 2
               _ _______________ _______________ __
SLA            _X_______________X_______________X__
                         _______         _______
SL[T,D]WCK    __________/       \_______/       \__
(ft or rr)
```

slbd == 1:

```
Phase clocks    | 2   1 | 2   1 | 2   1 | 2   1 | 2   1 | 2   1 |
               _ _______________________ _______________________ __
SLA            _X_______________________X_______________________X__
                                 ___________             ___________
SL[T,D]WCK    __________________/           \___________/           \__
(ft or rr)
```

slbd == 2:

```
Phase clks  | 2   1 | 2   1 | 2   1 | 2   1 | 2   1 | 2   1 | 2  1 | 2  1 |
           _ _______________________________ _____________________________ _
SLA        _X_______________________________X_____________________________X_
                             _______________                 _____________
SL[T,D]WCK__________________/               \_______________/             \_
(ft or rr)
```

slbd == 3:

```
Phase clks  | 2   1 | 2   1 | 2   1 | 2   1 | 2   1 | 2   1 | 2  1 | 2  1 |
           _ _______________________________ _____________________________ _
SLA        _X_______________________________X_____________________________X_
                                     _______________                 _______
SL[T,D]WCK__________________________/               \_______________/
(ft or rr)
```

If slp == 1, then tAVAV, or minimum time between address bus transitions, should not be faster than the basic_cycle.

The tag array connects to SLA[6:12], SLA[14:26], and SLATV_13, only. The data array connects to SLA[6:27], and SLA27LO. This is important for configurations with sync data SRAMs, and async tags.

Shown below is an SLC access with register_register data array and an async tag array. Since the tag SRAMs do not connect to address bits 13, 27, and 27LO, the tag address does not change every two cycles as the controller steps through the line. Four cycles are therefore allowed for the tag access, and slower parts may be used.

```
Phase clocks    | 2   1 | 2   1 | 2   1 | 2   1 | 2   1 | 2  1 |
               _ _______________ _______________ ______________ ___
SLA            _X_______________X_______________X______________X___
including
bits 13, 27, 27LO

                         _______         _______         ______
SLDWCK        __________/       \_______/       \_______/
                                                        ^  DRD sampled

Phase clocks    | 2   1 | 2   1 | 2   1 | 2   1 | 2   1 | 2  1 |
               _ _______________________________ __________________
SLA            _X_______________________________X__________________
not including
bits 13, 27, 27LO
                                                ^  SLT sampled
```

The input structure of both the tag and data receivers is a master/slave flip flop. The falling edge of the master latch determines the window around which data must be valid. The master latch clock is free running. The slave latch clock CK2N, and only fires when the controller is actually using the data. For tags, the master is clocked by free running CK1N. For data, the master clock is RCK.

RCK is a special version of the system clock generated from RSYNCH and RSYNCL. RSYNCH and RSYNCL are similar to the main clock inputs, SYNCH, and SYNCL, except for special timing necessary to optimize DRAM and SLC performance. RCK is nominally CK1N, however it may be delayed via PC board trace up to one phase.

With the exception of SL[T,D]WCK, all SLC control and address pins change on rising CK2N. SL[T,D]WCK always change on rising CK2N, except for when slbd == 1, and sync SRAMs are used. In this case, SL[T,D]WCK will transition low to high on rising CK2N, and transition high to low on rising CK1N.

slstrttag and slstrtdata define when the tag or data buses are sampled respectively. There are independent timers associated with the tag and data sides. Conceptually, these timers are loaded on the CK2N rising associated with the SLA transition from the first to the second address of a transaction. The CK2N slave latch clock in the receivers described above fires on the CK2N corresponding to when the counter first reaches zero. If slstrttag or slstrtdata equals zero, then the timer first reaches zero on the same state as the first address transition and the data latch fires.

Example showing slstrttag == 2 and slstrtdata == 3:

```
Phase clocks    | 2   1 | 2   1 | 2   1 | 2   1 | 2   1 | 2  1 |
               _ _______________ _______________ _______
SLA            _X__first add____X__second add___X_______
               _     ___     ___     ___     ___     ___     ___
Master clock    \___/   \___/   \___/   \___/   \___/   \___/   \

Tag timer       X   0   X   0   X   2   X   1   X   0   X   0   X
                                                 ___
Slave clock data _______________________________/   \_____________
                                                ^  tag latched

Data timer      X   0   X   0   X   3   X   2   X   1   X   0   X
                                                         ___
Slave clock data _______________________________________/   \______
                                                        ^  data latched
```

slstrtdata programs when the first latch clock happens. Subsequent data clocks always occur on basic_cycle intervals, as defined by slbd.

The start tag and start data timers function independent of the values programmed into slbd and sl[t,d]cnfg, but calculating the correct values for slstrttag and slstrtdata depend directly on what is programmed into slbd and sl[t,d]cnfg.

Nominally, the first latch clock should fire on the phase following an address transition for async, on the phase following the second rising SL[T,D]WCK for ft, and following the third rising edge of SL[T,D]WCK for rr mode.

In all cases the conditions slstrttag <= slstrtdata and slstrtdata <= (2 * basic_cycle) must be satisfied.

Nominal programming parameters for slstrttag and slstrtdata:

	slbd ==	**0**	**1**	**2**	**3**
slstrttag	async tag	0	0	0	0
slstrtdata	sync data	0	0	0	0
slstrttag	ft || async tag	1	2	2	3
slstrtdata	ft data	1	2	2	3
slstrttag	async tag	2	3	4	4
slstrtdata	rr data	3	5	6	7
slstrttag	ft || rr tag	3	5	6	7
slstrtdata	rr data	3	5	6	7

Depending on the minimum and maximum delays on the address and control lines external to the chip, it is possible to obtain correct operation with other values. Also, it is allowed for slsstrt[tag,data] to be equal to 2 * basic_cycle, in which case the start data counter may not ever get to zero during pipelining in dwmode. In this case, the data is sampled where data timer would_have_been zero if the pipelining had not occurred (confused yet?).

Tag side async SRAMs:

tAVAV	2 + slstrttag
tAVQV	2 + slstrttagt

AVWL	basic_cycle
tWHAX	if slbd == 2, then 2, else 1
tWLWH	basic_cycle
tAVWH	tAVWL + tWLWH
tDVWH	basic_cycle
tGLQV	2 + slstrttag
tWHDX	if slbd == 2, then 2, else 1

Data side async SRAMs:

tAVAV	2 + slstrtdata
tAVQV	2 + slstrtdatat
AVWL	avwl + 1
tWHAX	if slbd == 2, then 2, else 1
tWLWH	(4 + drdw + cas + cac) – (basic_cycle + avwl + 1) or (3 + cas + cp) – (3 if slbd == 2, else 2)
tAVWH	tAVWL + tWLWH
tGLQV	basic_cycle
tWHDX	if slbd == 2, then 2, else 1
tDVWH	For copyins the data valid window is defined by the memory controller. Setup and hold requirements must be guaranteed through proper programming of the DRAM controller, component selection, and board design. Normally, the rising edge of SLDWCK should occur at the same time as the data is latched in the CPU. For copyouts: if slbd == 2 && cas => 1 then tDVWH = 1 + cas + cp else tDVWH = 2 + cas + cp

Since data is simultaneously written into the SLC during a copyin from the DRAM, async write timing is dependent on DRAM controller configuration. It is possible to program WLWH < 1, in which case there will be no write pulse.

Data side sync SRAMs:

slbd ==	0	1	2	3
tKHKH	2	3	4	4
tKHKL	1	1.5	2	2
tKLKH	1	1.5	2	2
tAVKH	1	2	2	3
tWVKH	1	2	2	3
tKHAX	1	1	2	1
tKHWX	1	1	2	1

From rising SLDWCK to data latched by the 7300LC:

```
if slbd == 2
  then tKHQV = ((slstrtdata + 1) modulo basic_cycle) + 1
  else tKHQV = (slstrtdata modulo basic_cycle) + 1
```

```
case slbd
  0: tGLQV = tKHQV + 1
  1,2: tGLQV = tKHQV + 2
  3: tGLQV = tKHQV + 2
```

For copyins the data valid window is defined by the memory controller and the DRAM emulator. Setup and hold requirements must be guaranteed through proper programming of the DRAM controller, component selection, and system design. Normally, the rising edge of SLDWCK should occur at the same time as the data is latched in the CPU.

For copyouts,

```
        if slbd == 2 && cas => 1
            then
                tDVKH = 1 + cas + cp
                tKHDX = 2
            else
                tDVKH = 2 + cas + cp
                tKHDX = 1
```

If slbd == {2,3} and (cp + cas + 3 < 4), the DRAM interface will write faster than the SLC can cycle. This will result in a tKLKH violation, and is not considered a valid configuration.

Tag side sync SRAMs:

slbd ==	**0**	**1**	**2**	**3**
tKHKH	2	3	4	4
tKHKL	1	1.5	2	2
tKLKH	1	1.5	2	2
tAVKH	1	2	2	3
tWVKH	1	2	2	3
tKHAX	1	1	2	1
tKHWX	1	1	2	1
tDVKH	2	3	4	4
tKHDX	1	1	2	1

Referenced from rising edge of SLTWCK to rising edge of slave latch clock:

```
if slbd == 2
  then tKHQV = ((slstrttag + 1) modulo basic_cycle) + 1
  else tKHQV = (slstrttag modulo basic_cycle) + 1
```

```
case SLBD
  0: tGLQV = tKHQV + (1 if FT, else 3)
  1: tGLQV = tKHQV + (2 if FT, else 5)
  2: tGLQV = tKHQV + (2 if FT, else 6)
  3: tGLQV = tKHQV + (3 if FT, else 7)
```

SLT drive changeover:

tTSOZ2CEN	basic_cycle	SLTOE high until CPU drive
tTCOZ2TEN	0	CPU high z to SLTOE low

7.4 Errors and Error Logging

7.4.1 Memory Data Errors

The MIOC implements an single bit correct, double bit detect error scheme. This feature may be turned off or on independently for the SLC and main store DRAM. However, if enabled for the SLC, it must also be enabled for DRAM.

The error correction code can detect any double bit error, and all three and four bit burst errors within an aligned nibble. If x4 DRAMs are used and connected to aligned nibbles on the data bus, any complete DRAM failure is detectable.

An eight bit error code is calculated across a 64 bit data word. In double wide mode, two calculations are made for each half of the 128(144) bit word. Upon error, the 64 bit double word is stored in M_ERR0, and M_ERR1. The 8 check bits are stored in M_ERR_BYTE. The double word address is stored in MDERRADD.

M_ERR0, M_ERR1, M_ERR_BYTE, and MDERRADD log errors for both the SLC and mainstore DRAMs. merrsrc (MIOC_STATUS[23]) indicates which source caused the logged error.

Single bit errors set sedc (MIOC_STATUS[24]), and double bit errors set dedc (MIOC_STATUS[25]).

7.4.2 Memory Address Errors

The PA-RISC 1.1 architecture divides its 32 bit address space into a 256M byte I/O space, and a 4G – 256M byte memory space. Most systems will have less than maximum memory installed. Accesses to the unconfigured memory space area normally shouldn’t occur, but may happen in case of an OS, or possibly hardware failure.

If the MIOC simply ignored these transactions, the system would hang because the CPU hardware would not receive a response for what it thinks is a perfectly valid request.

The MIOC gracefully handles this situation by stepping through the transaction as if the address were valid. The transaction will not hit in the SLC because only valid memory address are ever marked valid. The memory controller takes over after the miss and steps through the transaction as if memory were present. Data error logging is suppressed, the address logged in MDERRADD, and pmae (MIOC_STATUS[29]) is set.

Setting the pmae bit leads to HPMC, and prevents the processor from executing too far past the error.

The memory controller will step through the access and assert control signals normally, except for ROW and COL, which will not cycle. For this error case it is possible to generate cycles with MOE and MWRITE cycling without corresponding ROW and COL.

DMA requests into unconfigured memory space are detected and logged by the DMA controller and should never make it to the memory controller.

7.4.3 Memory Error Ranking

Since only a single resource is available to store memory errors, the least recent, most severe error is stored. Single bit data errors are considered low rank and will overwrite no other errors. Double bit data errors and processor memory address errors are considered high rank, and overwrite single bit memory errors, but not themselves or each other.

7.4.4 Cache Tag Parity Errors

When enabled, the 7300LC checks parity on SLC tag reads. When bad parity is detected, the SLC misses. The address that caused the bad tag read is logged in SLTEADD. The tag is stored in SLTSTAT, and sltperr (MIOC_STATUS[22]) is set. Once sltperr is set, no more errors will be recorded. If sltperr is set, HPMC is signaled to the processor.

8. I/O

8.1 General Overview

Like the PA7100LC, the PA7300LC features a built-in I/O controller for a dedicated I/O bus. This I/O bus, called GSC, was introduced by the PA7100LC processor, and contains a 32-bit multiplexed address/data bus with a maximum cycle frequency of 40 MHz. Over this bus, the CPU can communicate with external I/O devices, external I/O devices can perform DMA to system memory, and both the CPU and external I/O devices can access PA7300LC diagnose registers. Unlike the PA7100LC, the PA7300LC also implements the GSC-1.5X extension of the WriteV cycle type, which gives the CPU higher throughput for I/O stores to graphics devices; and it includes DMA buffers, both to speed up I/O cycles, and also to minimize their impact on CPU accesses to DRAM.

8.1.1 GSC Overview

The PA7300LC implements the following GSC signals:

IODATA[31:0]	**in/out**	32-bit multiplexed address and data.
PARITY	**in/out**	Odd parity on (i.e., the exclusive-or of) the 32 IODATA lines.
TYPE[0:3]	**in/out**	Indicates the transaction type and byte enables.
ADDVL	**in/out**	Indicates an address is valid (start of cycle).
READYL	**in/out**	Indicates the slave is ready (completion of cycle).
ERRORL	**in/out**	Indicates a timeout or parity error.
LSL	**input**	Indicates a master is locking cycles together or a slave is splitting a cycle.
CREQUESTL	**output**	Indicates the PA7300LC wishes to own the GSC bus.
CGRANTL	**input**	Indicates the PA7300LC owns the GSC bus.
RESETL	**output**	Resets and synchronizes all other GSC devices.

A GSC write transaction consists of an address phase on the bus (indicated by ADDVL), immediately followed by 1 to 8 words of data; it is complete after READYL has been asserted or the last word of data has been sent, whichever comes later. A GSC read transaction begins with an address phase; when the slave can provide the requested data, it asserts READYL together with the first data word, followed immediately by the remaining words of data (if any); the cycle is complete after the last word of data is sent.

A slave is also allowed to “split” a transaction. By doing so, the slave temporarily sets aside the original cycle, masters one or more cycles of its own (for example, to resolve a potential deadlock situation), and then completes the original cycle. Note, however, that regular GSC is *not* a “split transaction bus” — a read from a slow device ties up the whole bus until the read data can be returned.

GSC is parity protected; the device receiving an address or data word is responsible for checking parity and indicating an error. A timeout counter (within the PA7300LC) terminates cycles whose addresses are invalid or contain a parity error, and in general helps to ensure that the GSC bus does not get “stuck”.

Please refer to the official GSC specification for all of the details of GSC signalling, cycle types, and protocols. Note, however, that the latest versions of the spec include information for all of the various flavors of GSC, including the GSC+ and GSC-2X enhancements that are *not* implemented by the PA7300LC. Only the generic GSC and GSC-1.5X descriptions in the spec apply to the PA7300LC.

8.1.2 Differences From the PA7100LC

A top priority in designing the PA7300LC was maintaining compatibility with the PA7100LC's interface and existing GSC devices. However, product designers who use the PA7300LC should be aware of the following differences and enhancements:

- The PA7300LC can generate WriteV cycles on the GSC bus, in addition to Write1 and Write2 cycles. The GSC15X_CONFIG register at I/O address 0xF###'F7A0 identifies the 8 MB regions of I/O space that can understand the WriteV transaction; when the CPU stores into one of these regions, up to 8 sequential words will be merged into a single, more efficient, WriteV GSC cycle. This feature is especially useful for graphics devices. PDC should detect WriteV-capable devices at boot-up, and set the corresponding bits in the GSC15X_CONFIG register. See the "CPU Accesses to I/O" section below, and the "Register Descriptions" chapter, for more information.
- The PA7300LC supports the architected FLEXID register, at I/O address 0xFFFC'0020. This register allows PDC to set the Bus_ID for the PA7300LC's CPU and MIOC HPAs. By default, the CPU and MIOC HPAs reside at the same locations as they do in the PA7100LC; but this register allows them to be re-mapped in case of an address conflict. Note that, if PDC does write to the FLEXID register, *all* architected devices on the GSC bus will be affected. See the "Register Descriptions" for more information.
- The PA7300LC does not support GSC's `XQL` signal. This is partly because the PA7300LC's DMA buffering and prefetching automatically provide much of the benefit of `XQL` on *all* DMA accesses, and partly to avoid confusion — `XQL` has at least three different meanings on various existing systems. Omitting `XQL` creates no functional problems; it simply makes the PA7300LC deny all `XQL` requests, and it is never mandatory to acknowledge an `XQL` request. Note, however, that the PA7300LC doesn't provide a bus holder on `XQL`, so the system board must include a pull-up resistor or some other means of keeping `XQL` from floating.
- The PA7300LC drives the GSC `RESETL` signal for the entire system. The system board must ensure that other devices which drive `RESETL` in a PA7100LC system (LASI, for example) are instead programmed to receive it.
- To allow higher CPU clock frequencies, the PA7300LC supports CPU cycle-to-GSC cycle ratios of 3, 4, 5, and 6; the PA7100LC supports only ratios 2 and 3. The GSC bus timing varies slightly for each ratio, but the PA7300LC's timing varies only within the range that the PA7100LC will generate. Two pins on the PA7300LC (GSCR[0:1]) select one of the four ratios at power-up; these ratio select signals must also be wired to the external clock generator chip (JarvicJr).
- In addition to accepting a write to the COMMAND register (at I/O address 0xF###'E030) to signal a transfer-of-control (TOC), the PA7300LC includes a direct transfer-of-control input pin (`TOCL`). When driven low, this pin signals a TOC to the CPU. Note that the `TOCL` pin is not "debounced," and it is level-sensitive; so PDC must be able to handle the repeated TOCs that will be generated while the system's TOC button remains pressed.
- The PA7300LC decouples the logging and reporting of I/O errors from memory errors. This may provide more accurate information in cases of multiple errors, but also requires that PDC examine more error logging registers to get that information. See in particular the MIOC_STATUS, DMAERR, and DIOERR "Register Definitions."
- The PA7300LC adds and rearranges other bits in the general configuration registers. The MIOC_CONTROL register's bits are rearranged, and the I/O-related `dma_noecc`, `dma_nocache`, and `fast_memory` bits are new. The GSC timeout is now controlled via the separate GSC_TIMEOUT register. The `ponstat` bit in the MIOC_STATUS register is moved,

and the `gscdiv` field is new. See the “Register Definitions” for more details; see the Memory Controller section of this ERS for information about the added memory- and SLC-related bits in these registers.

- The PA7300LC features a sophisticated DMA data buffering and prefetching system. The MIOC automatically handles all data coherency issues, so DMA hardware and software will see no functional differences. However, hardware will see timing differences when compared to the PA7100LC. In particular, DMA single- and partial-word writes and most DMA reads will complete in fewer GSC states, while a DMA write immediately followed by a DMA read may take longer to complete.

8.2 System Start-Up

Upon coming out of reset, the CPU will immediately begin fetching instructions from PDC (boot ROM) via GSC. Thus, the I/O system comes up ready to run CPU cycles to the GSC bus, without requiring any configuration. However, the following registers must be programmed to configure the I/O system before DMA is enabled, and before the CPU communicates with general I/O devices:

- FLEXID (address 0xFFFC’0020), if the system contains any GSC devices that implement this architected register, but don’t assign it a default value at reset. In such systems, set FLEXID to 0xF3FE’0001 to leave the PA7300LC’s registers in their “usual” places, compatible with the PA7100LC (this is important if, for example, a GSC device uses a hard-wired address to write into the CPU’s EIR).

- MIOC_CONTROL (address 0xF###’F080). This register contains bits that control both the I/O system and the memory system. Refer to the memory controller section for information on setting the memory-related bits; the I/O-related bits should be set as follows:
  * set `dma_noecc` if the PA7300LC should *not* flag bad data during DMA reads;
  normally, if a double-bit memory error is detected during an outbound DMA
  transaction, the bad data word(s) are marked with bad parity on the GSC bus;
  * set `lpmc_en` to detect and generate an LPMC on DMA accesses to memory
  space that fall outside of the system’s installed physical memory;
  * set `fast_memory` if `dwmode` (also in MIOC_CONTROL) is set;
  * set `lopowhilat` if the PA7300LC’s power consumption is critical, and you
  wish to slightly reduce it, even at the expense of increasing CPU to I/O latency;
  * (set `dma_nocache`, `pgape`, and `pgdpe` only for diagnostic purposes).

- GSC_TIMEOUT (address 0xF###’F0F0). The proper value to set in this register depends on what I/O devices are in the system. The timeout count must never be shorter than the longest expected latency for I/O cycles. However, particularly before doing an I/O walk, the timeout count also should not be set to an excessively large value, or else waiting for bus timeouts will waste a lot of time. Before doing the I/O walk, set the GSC_TIMEOUT slightly greater than the latency for the slowest expected CPU to GSC access; then, before enabling normal bus operation, set the GSC_TIMEOUT to be greater than the latency for *any* expected I/O transaction — this may be a much larger number, especially if there are bridges to other busses (such as EISA or VME) in the system.

- GSC15X_CONFIG (address 0xF###’F7A0), if the PDC detects a (graphics) device that supports GSC 1.5 extension WriteV cycles. Identify which 8MB I/O address ranges are accepted by slaves that understand these WriteV transactions, then set the corresponding bits in GSC15X_CONFIG.

- HIDMAMEM (address 0xF###’F0F4). After scanning the installed memory boards, set this register to indicate the top of installed physical memory.

8.3 Test/Debug

To aid the functional test of the I/O system, most of the I/O-related diagnose registers are both readable and writeable by software. The MIOC_CONTROL, GSC_TIMEOUT, HIDMAMEM, GSC15X_CONFIG, DMAERR, and DIOERR registers will all read back the last value written into them. But note that, except for the two error-reporting registers, these registers also affect the operation of the MIOC, and care must be used when setting them to nonstandard values.

The MIOC_STATUS register can be tested by writing 1's into individual bits. When the MIOC detects a write to this register, it will toggle the state of any bit set to 1. If that bit was previously set, it will be cleared (this is also how the MIOC_STATUS register will be used during normal system operation). But if that bit was previously clear, it will be set, and all of the usual consequences of setting the bit will ensue (for example, signalling an LPMC or an HPMC to the processor).

8.4 Clocking

The GSC bus operates at a submultiple of the CPU's clock frequency, and must be properly synchronized to the CPU clock. An external clock generator chip, named JarvicJr, generates the system clock for the PA7300LC, the GSC bus clock for I/O devices, and a signal to the PA7300LC that indicates the GSC clock phase. A simple block diagram of the clock connectivity in a PA7300LC system follows in Figure 5.

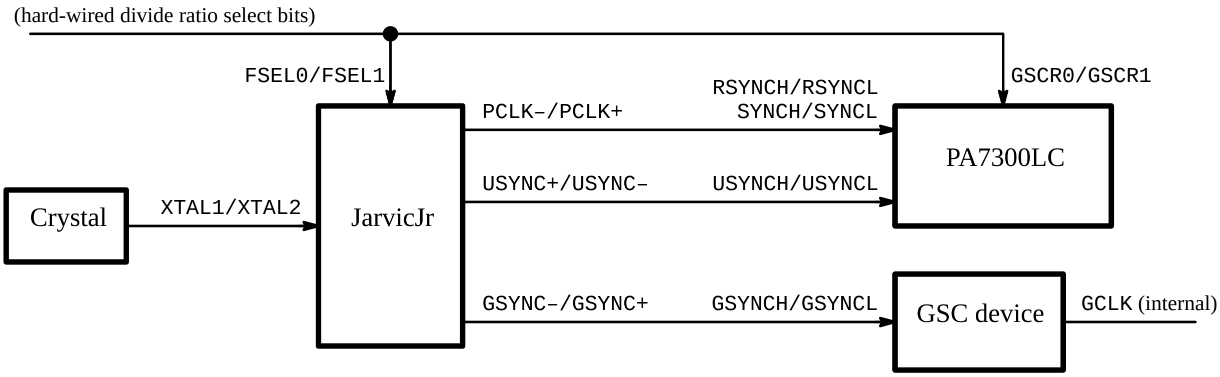


Figure 5. System Clock Connectivity

(note that JarvicJr's PCLK and GSYNC differential pairs are inverted as compared to the system's requirements).

The two divide ratio select bits, named GSCR0 & GSCR1 on the PA7300LC, determine the ratio of the CPU's clock frequency to GSC's operating frequency. The four ratios that can be selected are 3:1, 4:1, 5:1, and 6:1. These permit the CPU to be clocked from 75 MHz to 240 MHz, while GSC operates in its specified range of 25 MHz to 40 MHz.

Note that GSC's "operating frequency" is measured by GCLK, a clock that each GSC device generates internally. The GSYNCH/GSYNCL clock pair physically sent across the GSC bus is twice the frequency of GCLK. Thus, in a typical 4:1 configuration (GSCR[0:1] = %01), SYNCH will be 160 MHz, GSYNCH will be 80 MHz, and GCLK will be 40 MHz. Also note that the main GSC clock pair, GSYNCH/GSYNCL, does *not* connect to the PA7300LC.

The relationships between the various clocks and GSC signals at the PA7300LC and the clocks at a GSC device, for each divide ratio, are shown in Figure 6.. The beginning of each GSC state is marked by a dotted line:

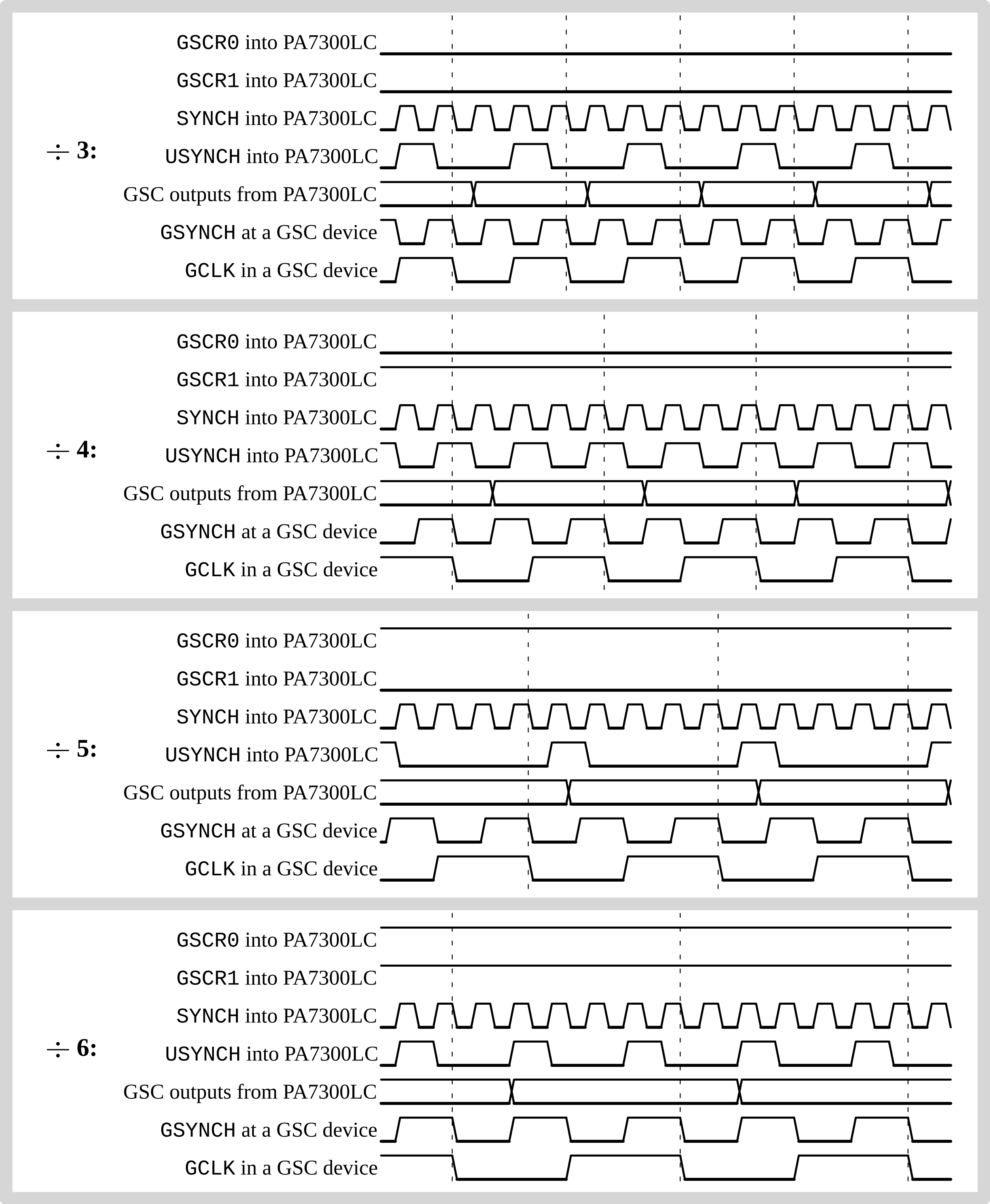


Figure 6. Clock Relationships in Various GSC Divide Ratios

8.5 Reset

There are two kinds of resets that can affect the GSC bus: hardware reset, and broadcast reset. Hardware reset typically occurs once upon power-up; broadcast resets can be issued by the CPU (or any GSC device) to reset a running system. And although the end result of both types of reset is the same, the behavior of each type of reset is quite different.

8.5.1 Hardware Reset

Hardware reset, indicated by an active (low) level on PON, is typically signalled by the power supply subsystem at power-up (although in some systems, a watchdog timer timeout or some other catastrophic event may also assert PON). The PA7300LC receives PON, and generates RESETL for all other GSC devices. A typical hardware reset sequence is shown in Figure 7.

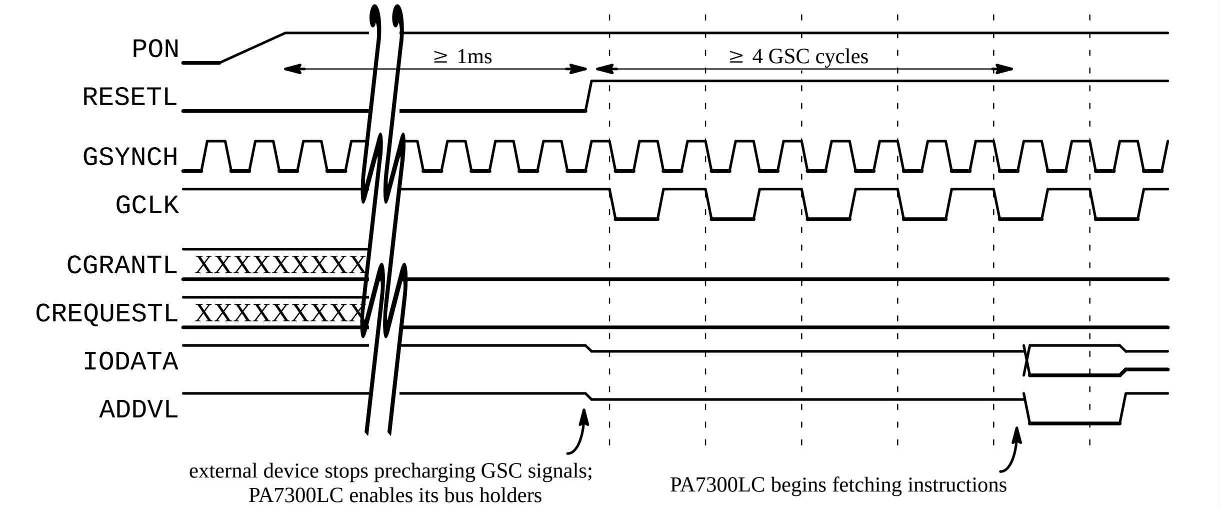

PON
RESETL
GSYNCH
GCLK
CGRANTL
CREQUESTL
IODATA
ADDVL
XXXXXXXXX
XXXXXXXXX
≥ 1ms
≥ 4 GSC cycles
external device stops precharging GSC signals; PA7300LC enables its bus holders
PA7300LC begins fetching instructions

Figure 7. Hardware Reset Sequence

The PA7300LC holds RESETL active (low) while PON is active, and then for at least 1ms more after PON is deasserted. GSC's RESETL signal actually serves two purposes: first, it performs a hard initialization of all devices on the GSC bus; second, it establishes the phase of GCLK. To guarantee the success of the first purpose, PON at power-up must be kept asserted for at least 100ms after the power supply voltages become stable. To achieve the second purpose, the PA7300LC precisely synchronizes the rising (inactive) edge of RESETL with the system clocks, such that the first falling (active) GSYNCH edge after RESETL rises will correspond to a falling (active) GCLK edge. Note that the PA7300LC knows where the GSYNCH edges occur, even though it does not receive GSYNCH, by using the GSCR bits and USYNCH.

The PA7300LC places weak bus holders on the GSC signals ADDVL, READYL, ERRORL, LSL, IODATA[31:0], PARITY, and TYPE[0:3]. However, during reset (while RESETL is low), the bus holders are turned off. It is the responsibility of the system board, or of an I/O device on GSC (such as Lasi or Dino), to establish valid logic levels on the GSC signals during reset. In particular, the GSC control signals (ADDVL, READYL, ERRORL, and LSL) must be high (inactive) coming out of reset, or the PA7300LC will not be able to start fetching instructions. When RESETL rises (deasserts), the device precharging the GSC bus signals can stop driving, as the PA7300LC will enable its GSC bus holders to preserve the quiescent values established during reset. But note that the PA7300LC does *not* connect to XQL, or to the GSC+ control signals PENDL, PACKL, RETRYL, and DRRL; it is the responsibility of the system board to place bus holders or pull-ups on these signals, if necessary.

The PA7300LC always strongly drives `RESETL` and `CREQUESTL`, even during reset; it assumes that an external arbiter will always drive `CGRANTL`, even during reset. The PA7300LC places no bus holders on any of these signals. `CREQUESTL` will follow `CGRANTL` by one state during reset (just as when the GSC arbiter issues the PA7300LC a default bus grant).

Shortly (but at least 4 GSC states) after `RESETL` rises, the PA7300LC will attempt to fetch its first instruction via GSC at address 0xF000’0004. In most systems, the PA7300LC should have a default bus grant at that time; however, if it does not, it will arbitrate for the bus as usual. If an external device has a bus grant coming out of reset, it must wait for at least 8 GSC states before starting a cycle to guarantee that the CPU is fully out of reset and ready to handle GSC traffic.

Software can determine the cause of the most recent CPU reset by reading the `ponstat` bit in the MIOC_STATUS register: `ponstat=0` indicates it was a hardware reset.

8.5.2 Broadcast Reset

Broadcast reset is initiated when the CPU (or any other GSC device) writes 0x05 into the least significant byte of either the LOCAL_COMMAND or the GLOBAL_COMMAND register (addresses 0xFFFC’0030 or 0xFFFE’0030). The PA7300LC will ready the GSC cycle and then reset itself.

During a broadcast reset, GSC’s `RESETL` is *not* asserted, `GCLK` does *not* get resynchronized, and the PA7300LC does *not* turn off its GSC bus holders. In other words, the PA7300LC offers no external indication that a reset is taking place. Internally, it will take up to 16 GSC cycles to completely reset its state (during which the GSC bus *must* remain idle to prevent a system hang). Then, no sooner than 16 GSC cycles after the end of the broadcast reset command, it will attempt its first instruction fetch at address 0xF000’0004 — the same as if it were coming out of a hardware reset. As during a hardware reset, the `CREQUESTL` line will follow `CGRANTL`; the CPU will not necessarily hold on to the GSC bus after it generates a broadcast reset. External I/O devices must not issue any GSC transactions within the 16 GSC cycles after a broadcast reset, as the CPU may not be ready to handle them and the GSC bus may hang.

All GSC devices can (and are encouraged to) see and handle broadcast resets. However, since many existing devices do not, it is unfortunately easy to leave GSC in a confused state if a broadcast reset is issued at the wrong time or in the wrong way. To avoid problems, it is strongly encouraged that broadcast resets be issued *only* from the PA7300LC, and then *only* after arbitration for other GSC devices is turned off. Also note that a data parity error during the broadcast reset will be reported via the `ERRORL` signal, but it will *not* inhibit the reset.

Software can determine the cause of the most recent CPU reset by reading the `ponstat` bit in the MIOC_STATUS register: `ponstat=1` indicates it was a broadcast reset.

8.6 GSC Bus Arbitration

The PA7300LC does not act as the system's GSC bus arbiter. Instead, it provides `CREQUESTL` and `CGRANTL` signals to an external arbiter, such as Lasi or Clark, and arbitrates for the bus just as any other GSC device does. The central arbiter may treat the CPU special in one way, however: issuing it a default bus grant when no device wants the bus. The PA7300LC will acknowledge a default bus grant by asserting its `CREQUESTL`; while it has the default grant, it can begin an I/O load or store immediately, without suffering the bus arbitration latency.

During reset, most systems will want to give the default bus grant to the PA7300LC, since the first GSC activity after power-up should be the CPU fetching instructions from the boot ROM. However, this is not a requirement. While reset is active, the PA7300LC's bus grant will follow its bus request (the bus grant is treated as a regular default grant); after reset deasserts, the PA7300LC will request the bus if it does not already own it, in order to fetch its first instruction.

The PA7300LC is a "good" GSC bus citizen. In other words, it requests the bus only when it needs to master a GSC cycle (or when it is given a default bus grant), and then promptly releases the bus (after any currently-running cycle completes) when its grant is taken away. If the PA7300LC needs to master another cycle when it loses its bus grant, it will immediately re-request the bus. This behavior ensures that DMA won't be starved, even when the CPU executes a long string of I/O loads or stores (as it may do in graphics-intensive applications).

8.6.1 Arbitration Timing

Since bus arbitration latencies directly affect GSC throughput at every bus ownership change, the PA7300LC always updates its `CREQUESTL` signal as quickly as GSC permits.

`CREQUESTL` is asserted (driven low):

- as soon as the CPU issues a load or store to I/O space; or
- one GSC state after `CGRANTL` gives the PA7300LC a default bus grant; or
- one GSC state after `CREQUESTL` was negated, if the PA7300LC has another I/O cycle to run.

`CREQUESTL` is negated (driven high):

- one GSC state after `CGRANTL` removes the PA7300LC's default bus grant; or
- one GSC state after the last data word if the PA7300LC loses its grant during a read; or
- one GSC state after the ready or last data word if the PA7300LC loses its grant during a write.

Except in the case of a default grant, the PA7300LC will always begin a GSC cycle (by asserting ADDVL) one GSC state after receiving CGRANTL. The PA7300LC's arbitration-related timing is illustrated in Figure 8.

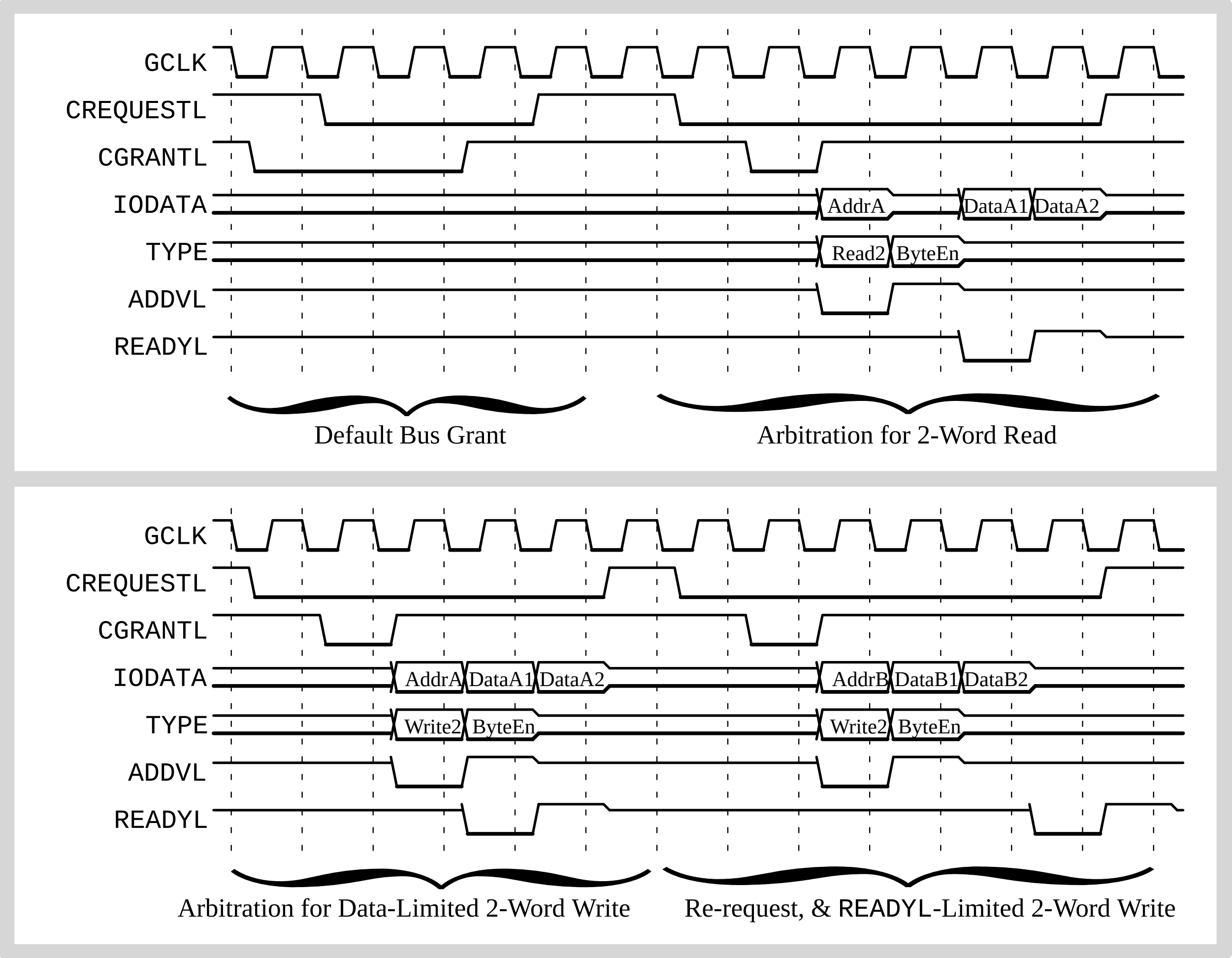


Figure 8. Arbitration-Related GSC Timing

8.7 CPU-Mastered GSC Cycles

The CPU issues reads and writes on the GSC bus to read status from or send commands to I/O devices in the system. These I/O devices may reside directly on the GSC bus, or may live on a standard bus such as EISA or PCI and be connected to GSC via a bus bridge. The PA7300LC also issues GSC bus cycles to read or write its own I/O-mapped diagnose registers (in which case the PA7300LC acts as both master and slave to the GSC cycle) — the specific timing and requirements of these cycles are described in a later section.

8.7.1 Basic Cycle Types

Physical addresses 0xF000’0000 through 0xFFFF’FFFF are defined as “I/O space” in the PA architecture; any CPU access to an address in that range will be translated into an I/O cycle on the GSC bus. When the CPU executes code from I/O space (as it will to execute PDC upon coming out of reset), it always fetches two instructions at a time, via 2-word reads on the GSC bus. When the CPU loads or stores data in I/O space, the type of load or store instruction determines the size of the GSC cycle. Instructions such as LDW and STW initiate 1-word GSC cycles. Instructions such as LDH, LDB, STH, and STB initiate partial-word GSC cycles, selectively enabling the GSC byte lane(s) corresponding to the specified address and transfer size. Doubleword floating-point loads and stores such as FLDDX and FSTDX initiate 2-word GSC cycles. Load and clear instructions (e.g. LDCWS) to I/O space are architecturally undefined; however, the PA7300LC will treat them as simple loads.

Normally, there is a one-to-one correspondence between CPU load and store instructions to I/O space, and transactions issued on the GSC bus, as described above; and all I/O transactions are strongly ordered. However, some devices (e.g., graphics) may wish to remove these restrictions for higher performance. The PA7300LC can be configured to generate WriteV GSC cycles and/or to accelerate I/O stores to such devices; these features are described below.

8.7.2 WriteV Cycles

The WriteV cycle is a special type of GSC transfer specified in the GSC-1.5X specification. A WriteV cycle can *only* be issued to a GSC slave that understands this cycle type; a “regular” GSC slave will not acknowledge a WriteV type of transfer, resulting in an HPMC. Therefore, the PA7300LC’s GSC15X_CONFIG register, at address 0xF###’F7A0, must be programmed to enable WriteV cycles only in those regions of I/O space (if any) that accept and understand the WriteV cycle type.

The WriteV cycle type allows up to 8 words of data to be written in a single GSC transfer (i.e. after sending only one address), greatly improving GSC’s data throughput. For example, the fastest slave could receive eight 1-word writes in 16 GSC states, or four 2-word writes in 12 GSC states; but it could receive the same 8 words in a single WriteV transaction in only 9 GSC states. Slower slaves see an even greater improvement: for example, a slave which takes 6 GSC states before issuing a READYL would receive eight 1-word writes in 64 GSC states, or four 2-word writes in 32 GSC states; but it would still receive an 8-word WriteV in only 9 GSC states.

The WriteV cycle differs from GSC’s regular 8-word write in that it need not be aligned, can transfer any number of data words from 1 to 8, and does not have to indicate the number of words in the transfer until the last data word is sent. These factors make it feasible for the PA7300LC’s hardware to automatically coalesce a sequential group of ordinary 1- and/or 2- word stores.

The PA7300LC will initiate a WriteV cycle when:

- the CPU issues two sequential single- and/or double-word stores; and
- the stores are in a WriteV-able block of I/O space, and do not cross a 4kB boundary; and
- the second store is queued up before the first store is begun on the GSC bus.

Note in particular that partial-word stores are never coalesced into a WriteV; and that timing is involved — minor changes to a code sequence, or to bus activity on GSC, may suddenly change whether stores get coalesced.

The PA7300LC will terminate a WriteV after the current data word when:

- the current word is the 8th in the WriteV, and the WriteV began at an even word address; or
- the current word is the 7th in the WriteV, and the WriteV began at an odd word address; or
- the current word is the last word in a 4kB page; or
- there is not another sequential single- or double-word store from the CPU at the head of the queue.

Note that an 8-word WriteV cannot be generated if the starting address of the WriteV is odd, and that timing is again involved in sustaining the WriteV for its maximum duration. But otherwise, WriteV generation is quite flexible; for example, the sequence “STW, STW, FSTDX, FSTDX, STW, STW” (with sequential addresses) *can* coalesce into a single 8-word WriteV.

WriteV cycle generation and accelerated I/O (described next) serve different functions, are enabled differently and independently, and in fact need not be enabled in the same I/O spaces. Typically, however (in particular, for graphics devices), an I/O space which can generate WriteV cycles should also be accelerated — otherwise, the CPU may not be able to issue stores in quick enough succession to effectively generate WriteV transactions. But a single WriteV *can* contain both accelerated and non-accelerated stores (as might happen if a STBYS instruction ends a sequence of I/O stores); the WriteV-generation logic is completely independent from the accelerated I/O generation.

8.7.3 Cycle Timing

When the PA7300LC masters a sequence of I/O operations, it will pack them together as tightly as GSC permits. A transaction will begin on the cycle following the last data word, the ready phase, or the bus turnaround phase (whichever comes latest) of the previous transaction. Typical back-to-back cycle timing is illustrated in Figure 9.

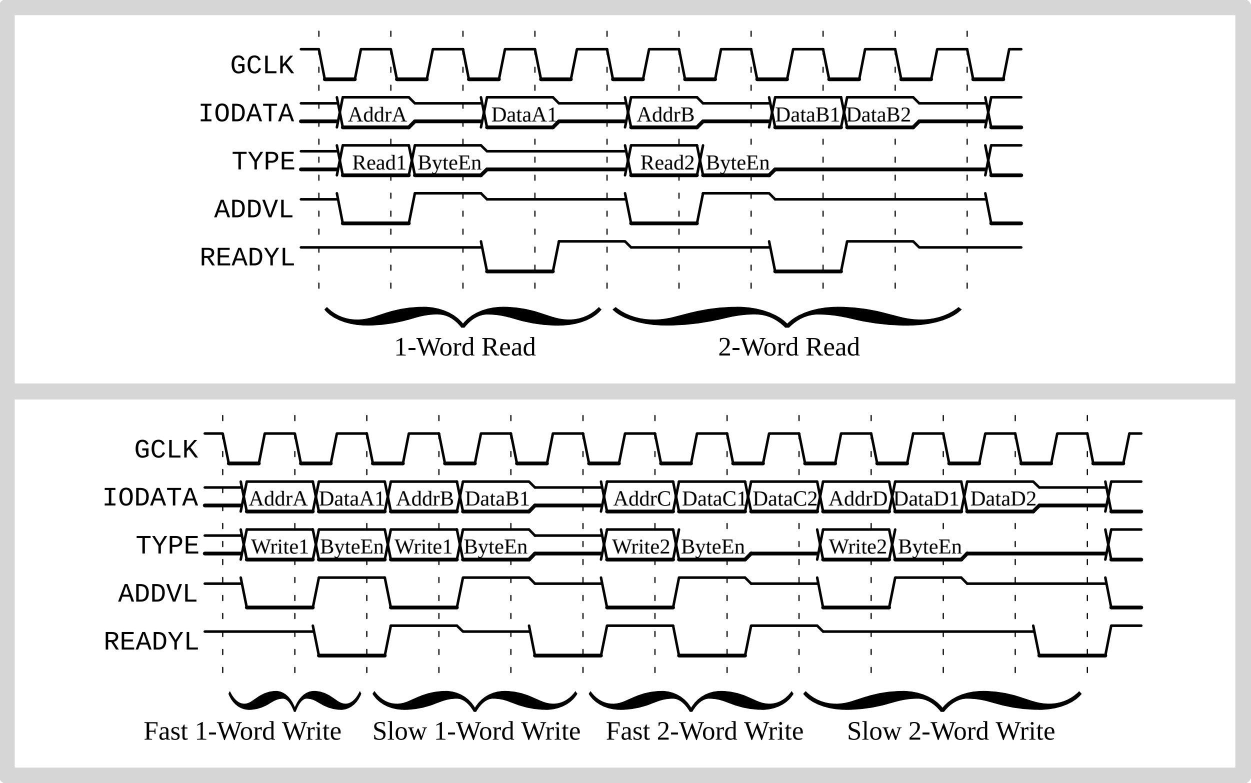


Figure 9. Back-to-Back Cycle GSC Timing

8.7.4 Accelerated I/O

The order of all I/O transactions from the CPU is *always* maintained. Normally, the ordering between I/O and memory accesses is also maintained; and the completion of a SYNC instruction guarantees that no CPU access to I/O is still in progress. Accelerated I/O is a feature which relaxes these last two restrictions. It requires no special hardware on the receiving I/O device; in fact, an accelerated store on GSC is completely indistinguishable from a non-accelerated one. But the software driver for an accelerated I/O device must be written with these differences in mind.

The ACCEL_IO register (diagnose register 13 in page 0) can enable accelerated I/O in various blocks of I/O address space. Any CPU store (except a STBYS) to an address within an enabled block automatically becomes an accelerated store (CPU loads cannot be accelerated; they must always wait for data to be returned from the I/O device). As far as the CPU and memory subsystems are concerned, an accelerated store disappears from consideration as soon as it is queued up. Thus, subsequent CPU instructions and/or memory accesses can execute without waiting for the pending accelerated store(s) to complete (or even to be issued) on GSC. Even a CPU SYNC instruction can retire while there are pending accelerated stores. Also, if any accelerated store generates an HPMC (due to a GSC parity error, for example), that HPMC may be reported to the CPU *long* after the offending instruction actually retired.

However, recall that all I/O transactions are executed in order, even if some of them are accelerated. So, the CPU can clean out all of its pending I/O either by executing an I/O load, or by executing a *non-accelerated* I/O store (including a STBYS inside an acceleratable I/O space) followed by a SYNC. Either sequence will flush out any pending accelerated stores, and will be retired only after the PA7300LC's I/O subsystem is idle.

Often (in particular, for graphics devices), an I/O space which is accelerated will also be configured to generate WriteV cycles (see the previous subsection). However, this is not a requirement; some drivers may be able to improve their performance by accelerating their I/O stores, even if their hardware does not understand the WriteV cycle type.

8.7.5 Split Cycles

The slave to any CPU-mastered GSC read or write can split the CPU's cycle to resolve a potential deadlock situation. The slave must adhere to the GSC specification (in particular, it cannot issue a broadcast reset within a split); but otherwise, the PA7300LC gives a slave complete flexibility in using (or abusing) splits. Specifically, a slave can assert LSL as many times as it wants within a single transaction, hold LSL active for as long as it wants each time, and run as many transactions as it wants (to system memory, or even to MIOC registers) while holding LSL active.

Since the same GSC signal is used to indicate both split and locked cycles, the PA7300LC must assume that all split cycles are also locked; see the description of locked cycles in the next section for the implications of this. Every time LSL is released, any buffered DMA write data will be flushed to memory, but there is no guarantee that the flush will actually take place before the originally-split CPU cycle completes.

8.7.6 Errors

Two kinds of errors may occur when the CPU masters a GSC cycle: an address error, or a data parity error.

An address error occurs if no slave device responds, and a cycle times out. This can occur either if the address references an undefined I/O location, or if there is an address parity error on the GSC bus. In either case, the `pioae` bit is set in the MIOC_STATUS register, and an HPMC is signalled to the CPU. If the offending cycle is a GSC write, the store data will be lost; if it is a GSC read, the load data returned to the CPU will be undefined.

A data parity error occurs if the recipient of a data word detects bad parity. If the offending cycle is a GSC write, the I/O device will check the parity and flag a data parity error (and can choose whether to use the erroneous data or ignore the cycle); if it is a GSC read, the PA7300LC will check the parity and flag an error (and return the erroneous data to the CPU). In either case, the `pmdp` bit is set in the MIOC_STATUS register, and an HPMC is signalled to the CPU.

Two bits in the MIOC_CONTROL register permit easy testing of the GSC parity generating and checking logic. The `pgape` bit forces the PA7300LC to generate bad address parity on its next GSC cycle; the `pgdpe` bit forces the PA7300LC to generate or check for bad data parity on its next GSC cycle. Software must execute a SYNC instruction after setting one of these bits, and must execute from memory space. The effects of a forced parity error are identical to any other parity error, including the generation of an HPMC.

8.7.7 GSC Debug Data

During idle times on the GSC bus, when the CPU has a default bus grant on GSC but is not running a cycle, the PA7300LC can be configured to continuously dump its internal processor state onto the GSC bus. Other GSC devices are not affected by this debug data, since all of the GSC control lines remain inactive; but a logic analyzer can use this data (along with the information available from the PA7300LC's dedicated debug pads) to trace internal CPU operation.

When debug mode is enabled (via CPU diagnose register 26) and the CPU has the default GSC bus grant, 36 bits of internal state are dumped onto GSC's `IODATA` and `TYPE` lines on *every CPU state*. Note that this is 3 to 6 times more often than the GSC signals usually transition, so special hardware, or shortening the GSC bus traces, or both, may be necessary to extract valid debug data. While debug data is being driven, GSC's `PARITY` signal may flotch around, and `CREQUESTL` will be continuously asserted; but all other GSC signals will be held in their inactive states.

The specific timing of when the PA7300LC will start or stop driving debug data depends on the GSC divide ratio. Roughly, debug data will commence 1-1/2 GSC states after:

- the PA7300LC acknowledges a default bus grant with CREQUESTL, or
- the slave to a PA7300LC write deasserts READYL, or
- the slave to a PA7300LC read completes driving the last word of data on IODATA.

Debug data will cease either:

- right before the PA7300LC masters a read or a write, or
- one GSC state after the PA7300LC drives CREQUESTL inactive to relinquish its default grant.

The PA7300LC does not attempt to drive debug data while a cycle is in progress, because it would not have enough advance warning of returning data during a read, or a split during a write, to avoid a drive fight on IODATA. Typical debug data timing in GSC's divide-by-3 mode is shown in Figure 10. ("★" indicates debug data)

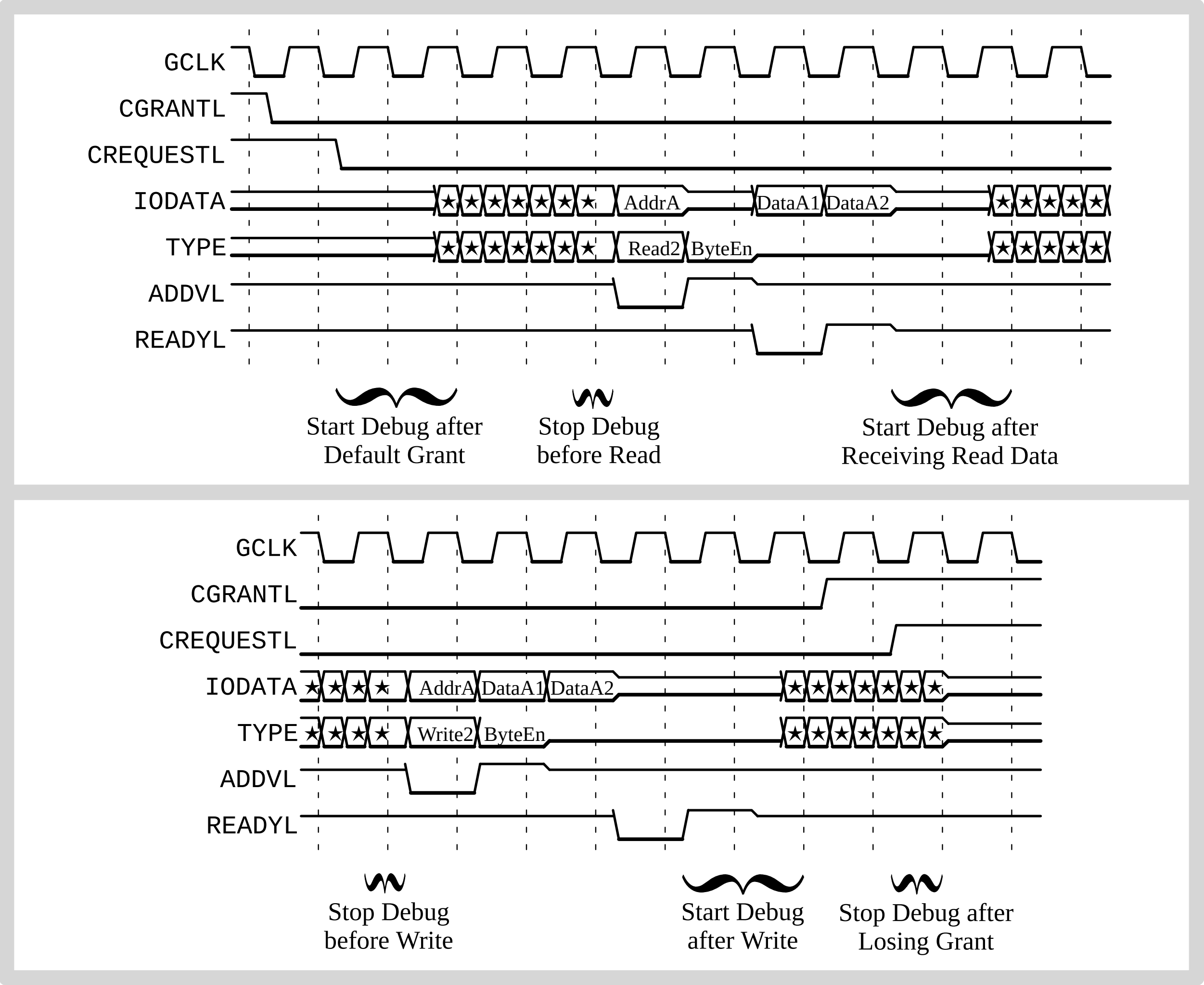


Figure 10. Timing of Debug Data on GSC (Divide by 3 Mode)

8.8 I/O-Mastered GSC Cycles to Memory

An external I/O device can issue reads and writes on the GSC bus to perform inbound or outbound DMA to system memory. An I/O device can also issue commands (such as “reset” or “interrupt”) to the CPU by writing to the PA7300LC’s I/O-mapped diagnose registers; these will be described in the next section.

8.8.1 Basic Cycle Types

An external I/O device issues a GSC read or write to PA-architected memory space (from 0x0000’0000 through 0xEFFF’FFFF) to perform DMA to or from system memory. The PA7300LC’s I/O controller forwards the request to its memory controller, and performs the necessary data transfer (assuming physical memory is installed at the requested address; otherwise, the GSC cycle will time out). All regular GSC cycles — partial- and full-word, and 2-, 4-, and 8-word transactions — are accepted; but all other types of cycles, including GSC-1.5X’s WriteV cycle type, are not.

To ease the burden on system memory, and to reduce I/O latency, the PA7300LC does not necessarily send each individual DMA operation directly to memory. Instead, it maintains several line buffers to coalesce groups of small inbound DMA transfers (I/O writes into memory space) into larger, more efficient units, and to hold data that upcoming outbound DMA transfers (I/O reads from memory space) are likely to ask for. Hardware interlocks in the PA7300LC prevent the buffers from holding stale data, and ensure that the buffering is transparent to software and external I/O hardware.

8.8.2 Cycle Timing

The exact timing of a DMA cycle is difficult to predict, as it depends on how busy the memory system is with previously-scheduled DMA, CPU, or refresh accesses to the DRAM. Outbound DMA must wait until buffered inbound data (if any) has been flushed to memory, and the requested outbound data has been returned; inbound DMA may have to wait for previously-buffered data to be flushed to memory, to make room in the DMA buffers. However, the PA7300LC’s fastest (best-case) timing for any DMA cycle is to wait 2 GSC states after ADDVL before asserting READYL (this is 1 GSC state slower than GSC’s top speed for reads, and 1 or 2 GSC states slower for writes). This fastest timing for the various DMA transactions is shown on the following page, in Figure 11.

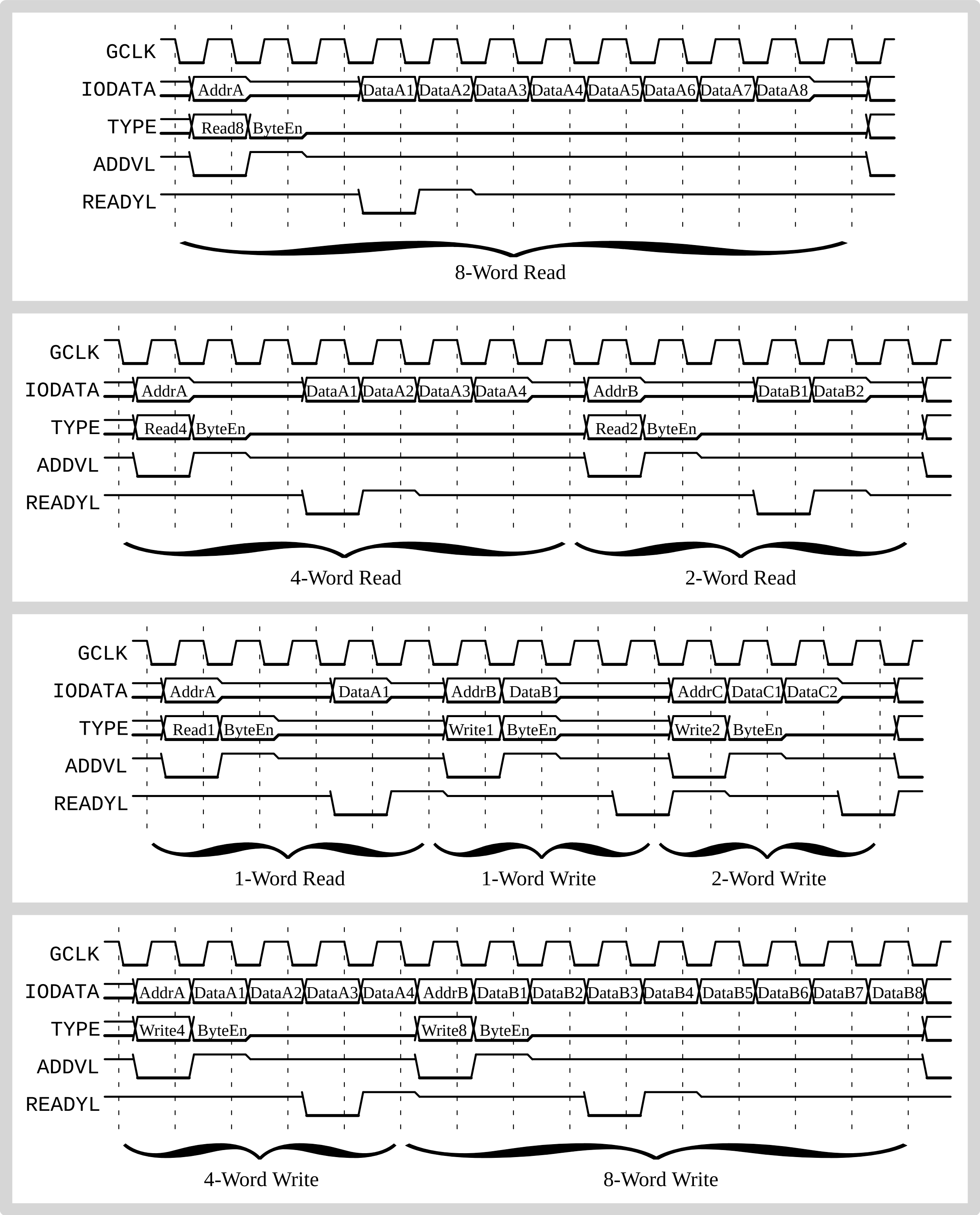


Figure 11. DMA Timing

8.8.3 Outbound DMA Buffering and Prefetching

Any “new” outbound DMA cycle will fetch an entire line of data from memory, critical doubleword first (a cycle for which data is already in a buffer will of course simply return the buffered data). This is done because DMA transfers tend to be sequential, and the probability that other data in the same line will eventually be accessed is pretty high. And, compared to the time required to send a new address to DRAM and get the first doubleword of data back, the time required to get the other three doublewords in the line is pretty low (especially in a 128-bit wide DRAM system).

In addition, an outbound DMA cycle will prefetch the *next* sequential line from memory if that next line has not already been prefetched, and:

- the cycle is a 2-, 4-, or 8-word read, or
- a DMA buffer already contains the data requested by the cycle.

In other words, the only outbound DMA cycle that will *not* result in having prefetched data available for the next cycle is a single- or partial-word read to an address which has neither read nor prefetch data in the DMA buffers. Prefetched data wraps around on each 4kB boundary; for example, a read2 from address 0x0000’3FE0 will prefetch the line at address 0x0000’3000. This simplifies the hardware, and ensures that there are no problems with prefetching past the end of physical memory, while having an insignificant effect on the prefetching’s effectiveness.

Read and prefetched data will remain in the DMA buffers until:

- another outbound DMA cycle occurs, which will replace them with new data, or
- an inbound DMA cycle occurs, which will invalidate both buffers, or
- the CPU writes into the same line as one of the buffers, which will invalidate that buffer.

No other checks are necessary to prevent stale data, since the memory and I/O controllers together know every event that can change system memory.

The `dma_nocache` bit in the MIOC_CONTROL register can disable this buffering and prefetching for diagnostic purposes; it should never be set in normal operation, as it will seriously degrade system performance. When this bit is set, the I/O controller will never prefetch lines from memory, and will never leave read data marked valid. Thus, every single outbound DMA cycle on GSC will issue its own (4 doubleword) read to system memory, and return the data from that new read.

8.8.4 Inbound DMA Buffering

Any inbound DMA cycle will be buffered, at least temporarily. The I/O controller attempts to gather neighboring DMA transactions into a single buffer, and then send a single large (up to 8-word) transaction to the memory controller, to make more efficient use of memory accesses. On the other hand, this gathering process must ensure that dirty data does not linger in the DMA buffers long enough to cause coherency problems. To meet all of these goals, the I/O controller follows a rather complicated buffering algorithm:

A currently-filling inbound DMA buffer will be immediately flushed if:

- an outbound DMA cycle begins, or
- there is an access to an I/O-mapped MIOC diagnose register, or
- an inbound DMA cycle occurs to an address other than the “next expected doubleword”, or
- an inbound DMA cycle begins with the state of GSC’s `LSL` signal changed
  (i.e., the beginning or end of a locked sequence of transfers), or
- the CPU receives the GSC bus grant.

An inbound DMA cycle will be merged into the currently-filling inbound DMA buffer, and *then* flush the buffer if:

- an inbound DMA cycle occurs to the “next expected doubleword” address, but
  overwrites one or more bytes in that doubleword that a previous DMA cycle wrote, or
- an inbound DMA cycle completely fills the last doubleword in its DMA buffer.

The “next expected doubleword” is the lowest doubleword within a DMA buffer line, at or after the first doubleword of valid data, in which one or more bytes have not yet been written. For example, writing a byte to address 0x0000’344F or a word to address 0x0000’3448 (or both, in either order) will set the next expected doubleword to 0x0000’3448; writing a doubleword to address 0x0000’5670 will set the next expected doubleword to 0x0000’5678. This is a very flexible scheme, which not only merges sequential 1-byte transfers (as an HPIB device may generate) into a single buffer, but also merges descending transfers (as (E)ISA DMA devices are capable of doing) or even random byte writes within a doubleword before sending them to memory; and it remains very efficient for the larger GSC transaction sizes too.

From the correctness standpoint, all of these rules ensure that synchronizing events (such as interrupts) will flush any dirty data before being seen by the CPU, that the memory controller is guaranteed to see a locked sequence as an indivisible unit, that an I/O device which writes repeatedly to the same memory location and expects the CPU to notice will work, and that data from an I/O device is always flushed shortly after that device releases the bus. Except for increasing DMA throughput, this buffering is invisible to hardware and software.

The dma_nocache bit in the MIOC_CONTROL register can disable this buffering for diagnostic purposes; it should never be set in normal operation, as it will seriously degrade system performance. When this bit is set, the I/O controller will flush the DMA buffers after every transaction. Thus, every inbound DMA cycle on GSC will be individually and immediately written to system memory.

8.8.5 Locked Cycles

The PA7300LC permits an I/O device to guarantee exclusive access to system memory, either for a simple read-modify-write semaphore operation, or for an extended sequence of memory transactions.

An I/O device begins a locked sequence by asserting LSL at the same time as ADDVL on a access to system memory. The I/O controller will clean out its DMA buffers, if necessary, then request exclusive access to memory. When the PA7300LC asserts READYL to complete the cycle, the I/O device knows it exclusively owns system memory. Subsequent transactions in which LSL accompanies ADDVL will all be locked together. When the PA7300LC receives either an ADDVL with LSL deasserted, or a GSC bus grant, it will clean out its DMA buffers and then release the exclusive access lock, once again letting the CPU have access to the memory.

Note that the GSC specification requires that LSL be held active continuously through the locked transaction sequence. However, the PA7300LC mimics the PA7100LC by examining LSL only when ADDVL is active. Thus, an I/O device designed for the PA7100LC which releases LSL between two locked ADDVLs will continue to work properly in a PA7300LC system. But newly-designed devices should adhere to the GSC spec for maximum flexibility.

8.8.6 Errors

Three kinds of errors may occur when an I/O device masters a DMA cycle on GSC: an address parity error, a DMA memory limit error, or a data parity error.

If an address parity error occurs, no device (including the PA7300LC) will respond to the cycle, and it will time out. In this case, the PA7300LC ignores the cycle completely, because it has no way of knowing if the cycle was really intended for system memory, or if an erroneous address bit simply made it appear that way. It is up to an I/O device to detect and handle address parity errors in a way that is appropriate for the device. The PA7300LC will also not respond if a cycle has an invalid encoding in its type bits (i.e. TYPE[0] = 1), so such a cycle will also time out.

A DMA memory limit error occurs if an I/O device masters a valid cycle to PA-architected memory space (0x0000’0000 through 0xEFFF’FFFF), but no physical memory is installed at that address (as indicated by the HIDMAMEM configuration register at address 0xF###’F0F4). In this case, the PA7300LC will not respond, and the cycle will time out on the GSC bus; but the PA7300LC will log the error as a dml in the MIOC_STATUS register, and signal an LPMC to the CPU if the lpmc_en bit in the MIOC_CONTROL register is set. The I/O device is still responsible for handling the error appropriately.

A data parity error occurs if the recipient of a data word detects bad parity on it. If the offending cycle is an outbound DMA, the I/O device will check the parity and flag a data parity error (and, in some cases, will want to abort the transaction rather than possibly propagating bogus data); if it is an inbound DMA, the PA7300LC will check the parity and flag an error (but will still write the possibly erroneous data into memory). In either case, the `dmdp` bit is set in the MIOC_STATUS register, and an HPMC is signalled to the CPU.

During outbound DMA, if the PA7300LC detects a double-bit (uncorrectable) error in the data from DRAM, it will *purposely* generate bad data parity on GSC, to flag the bad data for the I/O device. If this behavior is undesirable, and no I/O devices in the system need to know about questionable data, the `dma_noecc` bit in the MIOC_CONTROL register can be set to disable this feature. The memory system will still alert the CPU to the double-bit error, even though the GSC cycle will complete as though nothing were wrong.

8.9 I/O-Mapped Diagnose Register Accesses

Any device on GSC can access any of the CPU's I/O-mapped diagnose registers at any time. Generally, though, the CPU will access its diagnose registers only once at power-up to perform self-test and configuration (while executing PDC), and perhaps later read error-logging registers to find the source of a failure; and external I/O devices will write only to a COMMAND or EIR register, to reset or interrupt the CPU. Regardless of its source or purpose, any access to a diagnose register must be a full 1-word GSC read or write. Partial-word or multiple-word cycles to diagnose registers are *not* supported — they will be accepted and handled with correct GSC protocol, but functionally will be treated as full-word cycles (for example, a partial-word write will write garbage data from the unused byte lanes into the addressed register, or a multiple-word read will simply return the same data from the addressed register multiple times).

8.9.1 Cycle Timing

If the DMA buffers are idle, the PA7300LC will wait 2 GSC cycles after `ADDVL` before asserting `READYL` (this is the same timing as most DMA accesses). However, if there is dirty data in a DMA buffer, it will first be flushed; the PA7300LC will not assert `READYL` to signal that a register access is complete until all flushes (and prefetch reads) to system memory are done, which may slow down register accesses that are near DMA transactions by a few cycles. The typical (fastest) register access timing is shown below in Figure 12.

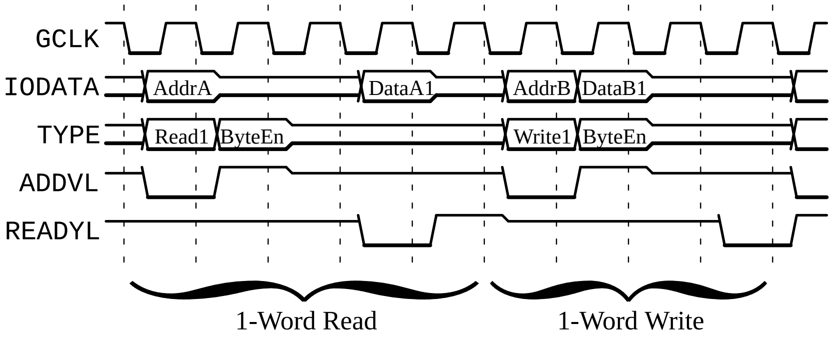


Figure 12. Register Access Timing

8.9.2 Errors

The usual address parity errors and data parity errors may occur on GSC during register accesses; but there are no other error conditions. A data parity error will set the `dmdp` bit in the MIOC_CONTROL register and signal an HPMC to the CPU, just like a data parity error encountered during DMA to system memory. If the CPU mastered the register access, both types of errors are also handled as described in the “CPU-Mastered GSC Cycles” section of this chapter. A data parity error will *not* inhibit or in any way affect the effect of a register access.

8.10 Interrupts and Transfers of Control

Just as it can signal an HPMC or an LPMC to the CPU to flag an error condition, the I/O controller can signal two other events to the CPU at any time — events which are not error conditions, but which require special event processing.

The first is an external interrupt request, or EIR. An external I/O device can issue an EIR by writing to any of the EIR or LOCAL_EIR or GLOBAL_EIR registers, as described in the “Register Definitions”. As with all register writes, any dirty data in the DMA buffers will be flushed before the register write occurs, so the CPU is guaranteed to see the EIR in the correct order with respect to other DMA traffic on GSC.

The second is a transfer of control, or TOC. An external I/O device can generate a TOC either by writing to the CONTROL register at 0xF###’E030, or by driving the PA7300LC’s dedicated `TOCL` pin low (active) for at least one GSC state. Note that the `TOCL` pin is level-sensitive, so if it is still asserted after the CPU services a TOC and re-enables TOC traps, another TOC trap will be taken.

8.11 Performance

The following table lists the PA7300's performance for the various types of GSC cycles.

Throughput numbers are for best-case, steady-state conditions:

- the GSC master has a continuous bus grant;
- the external I/O device drives its control signals active as quickly as possible;
- the memory system is not busy with refreshes or CPU requests;
- for CPU to I/O transactions, the CPU issues an uninterrupted stream of I/O loads or stores;
- for CPU WriteV stores, the CPU writes to an accelerated I/O space; and
- for I/O reads from memory, sequential reads have caused data to be prefetched from DRAM.

An I/O DMA read from memory that was not predicted and prefetched (mainly, the first read in a new DMA transfer) will incur some additional latency. The exact amount of additional delay is difficult to determine, as it depends on refresh, CPU activity, memory timing, etc.; but in most cases it will be about 4 to 6 GSC cycles. Additional DMA latencies due to refresh or CPU-to-memory traffic will seldom exceed 6 GSC cycles.

GSC Cycle Type	Data Transfer Rate	Example 1	Example 2	Example 3	Units
CPU clock frequency		132	160	200	MHz
CPU clock to GSC cycle divide ratio		÷ 4	÷ 4	÷ 5	
GSC cycle frequency		33	40	40	MHz
CPU to I/O transactions:					
CPU 1-word load	4 bytes / 4 GSC cycles	33.0	40.0	40.0	MB/s[*]
CPU 2-word load	8 bytes / 5 GSC cycles	52.8	64.0	64.0	MB/s
CPU 1-word store	4 bytes / 2 GSC cycles	66.0	80.0	80.0	MB/s
CPU 2-word store	8 bytes / 3 GSC cycles	88.0	106.7	106.7	MB/s
CPU WriteV store	32 bytes / 9 GSC cycles	117.3	142.2	142.2	MB/s
I/O to register transactions:					
1-word register read	4 bytes / 5 GSC cycles	26.4	32.0	32.0	MB/s
1-word register write	4 bytes / 3 GSC cycles	44.0	53.3	53.3	MB/s
I/O to memory (DMA) transactions:					
1-word memory read	4 bytes / 5 GSC cycles	26.4	32.0	32.0	MB/s
2-word memory read	8 bytes / 6 GSC cycles	44.0	53.3	53.3	MB/s
4-word memory read	16 bytes / 8 GSC cycles	66.0	80.0	80.0	MB/s
8-word memory read	32 bytes / 12 GSC cycles	88.0	106.7	106.7	MB/s
1-word memory write	4 bytes / 4 GSC cycles	33.0	40.0	40.0	MB/s
2-word memory write	8 bytes / 4 GSC cycles	66.0	80.0	80.0	MB/s
4-word memory write	16 bytes / 5 GSC cycles	105.6	128.0	128.0	MB/s
8-word memory write	32 bytes / 9 GSC cycles	117.3	142.2	142.2	MB/s

[*] 1 MB/s = 1,000,000 bytes per second

9. Fault Tolerance

9.1 Introduction

The PA7300LC has error detection on both internal caches. Single bit errors in these caches are detected an cause an HPMC. Thus, it is up to software to determine the recoverability of these errors and to attempt any recovery. The TLB (and other on–chip circuitry) does not have parity circuitry and does not detect errors. The memory system (including both the second level cache and DRAMs) supports single–bit correction, double–bit detection Hamming codes. Therefore, single bit memory errors are automatically corrected by the hardware and may signal an LPMC for logging purposes. Double bit memory errors are detected and reported through an HPMC.

All TOCs (transfers–of–control) and instruction cache HPMCs should be recoverable, depending upton the IPSW Q–bit being set. All data cache, second level cache and memory HPMCs are unrecoverable. Group 4 interruptions are never lost due to HPMC. See the rest of this chapter for more details on recoverability.

Note that after any error indication, software must clear the appropriate diagnose register error flags on the CPU (via the MTCPU diagnose instruction) or the appropriate error flags in the MIOC status register (via I/O write) before performing an RFI from the HPMC or LPMC trap handler. For details of what information is saved in the diagnose registers, see the diagnose register chapter.

9.2 Level 1 Instruction Cache

9.2.1 Hardware

The Level 1 instruction cache is protected by a simple parity scheme. There is one parity bit for each 32 bit instruction word and 4 bit steering vector. There is also one parity bit for each 20 bit tag. The error signal coming from the parity trees arrives too late to stop the instruction(s) with the parity error from entering the pipeline. The CPU will, however, stop the erroneous instruction from affecting architected statte and will take an HPMC on the instruction that caused the error, or perhaps a prior instruction. Note that an HPMC taken while the PSW Q bit is zero is unrecoverable.

9.2.2 Software

Since it is not possible to run out of memory space with the Level 1 instruction cache disabled, only soft errors of the instruction cache should be considered to be recoverable. It may be possible to run with a hard failure in the instruction cache, but the performance loss would be extremely large. Software responsible for recovering from a Level 1 instruction cache HPMC must test the instruction cache and initialize all data and tag fields before returning back to memory space.

9.3 Level 1 Data Cache

9.3.1 Hardware

The Level 1 data cache is protected by a simple parity scheme. There is one parity bit for each 32 bit word of data. There is one parity bit for each 20 bit tag. Finally, the dirty bit has its own separate parity bit. The error signal coming from the parity trees arrives too late to stop the erroneous data from corrupting architected state. It also comes too late to guarantee that the CPU will take the HPMC interruption on the instruction that caused the error. The HPMC can occur on the offending instruction or on any of the next several instruction bundles in the pipeline. In addition, since parity checking is enable whenever the data cache arrays are powered up, and since it is possible for the arrays to be powered up for instruction that do not reference the data cache, it is possible for a non–data reference instruction to take an HPMC due to a data cache parity error.

9.3.2 Software

Since software has no way to regenerate dirty data that has an error, and since the IIASQF/IIAOQF are not guaranteed to alow retry of instructions will errors, there is no way for software to recover from an HPMC caused by a data cache error. Recovery is also precluded by the fact that a data cache HPMC is not guaranteed to abort all architected side–effects of the intruction taking the HPMC.

9.4 Second Level Cache and Memory

Single bit Level 2 cache and memory errors are fully corrected by the memory controller and will be reported to software via an LPMC trap if MIOC_CONTROL.slen is set. Double bit errors are detected but not corrected and are reported via an HMPC trap. Software cannot recover from a double bit memory error. The MIOC_STATUS register contains information regarding the error, and the error indicator must be cleared before re–enabling further LPMC and HPMC traps. See the memory controller chapter for details. HPMC traps due to Level 2 cache and memory errors are taken asynchronously with regard to, and usually well after, the instuction that caused them, similar to Level 1 data cache HPMC traps.

9.5 I/O

I/O errors on the GSC bus resulting in an HPMC or LPMC trap will be detected by the MIOC and will cause the CPU to vector to the appropriate trap handler. Note that the MIOC_CONTROL.lpmc_en bit affects the signalling of LPMC traps. Software can poll the MIOC_STATUS register to determine whether a GSC error has occurred. See the I/O controller chapter for details.

9.6 Software Requirements

This section lists some requirements on PDC software for HPMC and LPMC handling. The next section which describes PIM issues is related to this topic.

9.6.1 Requirements for HPMC

- When an HPMC is signalled during a memory transaction (due to a cache miss), the cache line which was copied into the instruction or data cache may have bad data. Since the cache controllers will validate this line, the line line must be purged (or flushed) before an RFI from the HPMC handler.

- The diagnose bit indicating the source of the error must be cleared before an RFI from the HPMC handler. The bits indicating the source of the error are in the CPU_CFG register (diagnose register 0) or in the MIOC_STATUS register.

9.6.2 Requirements for LPMC

- The diagnose bit indicating the source of the error must be cleared in the MIOC_STATUS register before an RFI from the LPMC handler, to avoid another LPMC for the same condition. The CPU has no LPMC indicators in the CPU_CFG register.

9.7 PIM Issues

This section will indicate how each of the PIM (Processor Internal Memory) bits are determined from processor state.

9.7.1 CPU State Word

This includes the following bits:

Bit	Meaning	Value
iqv	IIA Queue Valid	Always set.
iqf	IIA Queue Failure	Never set. The CPU cannot guarantee to trap on the instruction that caused the HPMC trap.
ipv	IPRs Valid	Never set. The CPU cannot guarantee to trap on the instruction that caused the HPMC trap.
grv	General Registers Valid	Always set.
crv	Control Registers Valid	Always set.
srv	Space Registers Valid	Always set.
trv	Temporary Registers Valid	Always set.
tl	Trap Lost	Never set.
hd	Hardware Damage	Set to 0 (CHECK_CRITICAL) for Level data cache, Memory, and GSC errors. Set to 1,2 or 3 (CHECK_TRANSPARENT or CHECK_ISOLATED) for Level instruction cache errors or Transfer–of–Control (TOC) traps
sis	Storage Integrity Synchronized	Always set.

9.7.2 Cache Check Word

By polling the CPU_CFG register, software can determine whether a cache HPMC was caused by the Level 1 instruction cache or the Level 1 data cache.

9.7.3 TLB Check Word

The TLB is on–chip and does not have parity detection circuitry. Therefore, this word should never be set as a result of an HPMC or LPMC trap.

9.7.4 Bus Check Word

See the Memory & SLC and I/O chapters for details concerning the Bus Check Word, Slave Address, and Master Address.

9.7.5 Other Check Words

The Assist Check word, and Assist ID word bits are never set because there is never an HPMC or LPMC trap due to an assist.

10. Diagnose

10.1 Introduction

The PA7300LC has a rich set of implementation–dependent registers to facilitate low–level testing and system startup. These registers are accessed through a combination of diagnose instructions and I/O loads and stores. This split was necessary because of routing constraints on the chip.

In addition to diagnose registers, the PA7300LC implements diagnose instructions which perform some operations which would be difficult or impossible with only architected instructions.

The Instruction Cache, Data Cache, Unified TLB, Instruction Translation Cache (ILAB), and architected interval timer all have some diagnose capability. Additionally, the MIOC and a special–purpose Debug block have diagnose programming interfaces.

PA7300LC diagnose is different than previous PCX/PCXS/PCXT diagnose implementations. One of the largest changes is the removal of the DOUBLE–DIAGNOSE rule. There are also changes to the diagnose registers, the diagnose instructions, and the software restrictions.

This chapter, together with the fault tolerance chapter, should provide enough information for an architectural review, as well as system initialization, selftest, and operating system needs. If you need additional information, contact the chapter author.

Tables 1 through 5 in the following chapter summarize the registers described in this chapter. Use it as a quick reference. The chapter after that contains descriptions for all the PA7300LC implementation specific diagnose instructions.

Because of the large number of diagnose registers available on the PA7300LC, they have been split into two ‘‘pages”. One page contains the basic CPU diagnose registers, and the other contains the registers used to control debugging. Two new diagnose instructions are used to set the page before a diagnose register read or write can be performed.

10.2 Software Constraints

This section lists software restrictions which were not mentioned in earlier sections or which are repeated in case you missed it earlier. This is not a complete list of restrictions however. PLEASE READ THIS.

- Diagnose instructions do NOT need to come in PAIRS. Earlier (PCX/PCXS/PCXT) processors had this requirement, BUT THE PA7300LC DOES NOT! Each single diagnose instruction will be executed independently, and they CAN be nullified.
- Diagnose instructions must not be immediately followed by RFI or RFIR.
- Diagnose instructions must not be immediately preceded by a floating point instruction that will take trap 14 (assists exception trap).
- The instruction immediately following a DR_PAGE0 or DR_PAGE1 instruction cannot be a MTCPU, MFCPU_C or MFCPU_T. An exception to this is allowed if moving to or from diagnose register #0.
- Two SYNC instructions should precede a MTCPU0 if the SOU_EN bit is being changed from 1 to 0 and there is any chance an outstanding DCache miss is still being serviced.

- When executing IC_DIAG to read the Level1 instruction cache, the Level 1 cache must first be disabled by clearing the L1ICACHE_EN bit in CPU diagnose register #0. It must also be guaranteed that the instruction immediately preceding the IC_DIAG instruction is not a branch that branches to a new page.
- When executing a DC_DIAG instruction to read, write or test the Level 1 data cache, the data cache must first be placed into test mode (DC_BIST_ADDR_CFG.TEST_MODEH set to 1). Prior to placing the data cache into test mode, two SYNC instructions must be executed to force all pending data cache miss operations to completing, and the store queue must be flushed by executing two stores to I/O space. It is also advisable to disable data cache HPMCs.
- When executing either an IC_DIAG or a DC_DIAG instruction that initiates a BIST operation, a SYNC instruction will cause the processor to stall until the BIST operation completes. However, a MTCPU instruction that writes the IC_BIST_ADDR_CFG register (for instruction BIST) or the DC_BIST_ADDR_CFG register (for data BIST) must not be placed immediately after the SYNC instruction. In addition, the SYNC instruction must be located at least two instructions after the DC_DIAG.
- Before executing an IC_DIAG or DC_DIAG instruction, a SYNC instruction should be used to allow all pending I/O writes to complete. This SYNC must be placed at least two instructions before the IC_DIAG or DC_DIAG.
- The instruction cache *must* be taken out of BIST mode (IC_BIST_ADDR_CFG.BIST_ENH cleared) before executing from memory space.
- After forcing parity errors in the instruction cache, the force parity error bits in the ICPFBSPx and ICPFBTAG registers must be cleared before executing code from memory space that could result in a cache miss. Failure to do so will cause parity errors to be seeded on lines brought in by instruction cache misses.
- As mentioned in the Fault Tolerance section, the CPU does not guarantee that it will take an HPMC trap on the same instruction that caused it to be signaled. For example, a load instruction to a non–existent memory location will not trap on this instruction but will trap on a subsequent instruction. PDC self–test software which wishes to cause an ‘‘expected” HPMC, therefore needs to know how long to wait for HPMC traps to occur. The answer depends on many things: 1) I–fetch vs. D–Access, 2) Stall on Use Miss, and 3) whether the MIOC_HPMCH line is signalled during or after the transaction causing the error. Here is one way to check for an expected HPMC: Load instruction to non–existent memory location with the SOU_EN (stall–on–use enable) bit clear, followed by two ‘‘sync’’ instructions will guarantee that the HPMC will be taken on or before the next instruction (following the second sync).

11. Register Definitions

11.1 Alphabetical List of Registers

Name	Group	Address/Reg #	Definition
ACCEL_IO	Page 0	13	Page 11–14
COMMAND	MIOC	0xf###e030	Page 11–32
COMMAND_GLOBAL	MIOC	0xfffe0030	Page 11–61
COMMAND_LOCAL	MIOC	0xfffc0030	Page 11–59
CPU_CFG	Page 0/1	0	Page 11–6
DATOER0	Page 1	10	Debug*
DATOR0	Page 1	9	Debug*
DATSIER0	Page 1	12	Debug*
DATSIR0	Page 1	11	Debug*
DC_BIST_ADDR_CFG	Page 0	11	Page 11–13
DC_DW0_IN	D–cache	0xf###e100–4	Page 11–17
DC_DW0P_IN	D–cache	0xf###e108	Page 11–17
DC_DW1_IN	D–cache	0xf###e110–4	Page 11–17
DC_DW1P_IN	D–cache	0xf###e118	Page 11–17
DC_DW2_IN	D–cache	0xf###e120–4	Page 11–18
DC_DW2P_IN	D–cache	0xf###e128	Page 11–18
DC_DW3_IN	D–cache	0xf###e130–4	Page 11–18
DC_DW3P_IN	D–cache	0xf###e138	Page 11–18
DC_DW0_OUT	D–cache	0xf###e140–4	Page 11–19
DC_DW0P_OUT	D–cache	0xf###e148	Page 11–19
DC_DW1_OUT	D–cache	0xf###e150–4	Page 11–19
DC_DW1P_OUT	D–cache	0xf###e158	Page 11–19
DC_DW2_OUT	D–cache	0xf###e160–4	Page 11–20
DC_DW2P_OUT	D–cache	0xf###e168	Page 11–20
DC_DW3_OUT	D–cache	0xf###e170–4	Page 11–20
DC_DW3P_OUT	D–cache	0xf###e178	Page 11–20
DC_TIMING	D–cache	0xf###e18c	Page 11–22
DEBUG	Page 0	26	Debug*
DELTMR	Page 1	25	Debug*
DIAGTAG	MIOC	0xf###f0a8	Page 11–40
DIOERR	MIOC	0xf###f0ec	Page 11–49
DMAERR	MIOC	0xf###f0e8	Page 11–48
DTAG_IN	D–cache	0xf###e188	Page 11–22
DTAG0_OUT	D–cache	0xf###e180	Page 11–21
DTAG1_OUT	D–cache	0xf###e184	Page 11–21
EIR	MIOC	0xf###e000	Page 11–31
EIR_GLOBAL	MIOC	0xfffe0000	Page 11–60
EIR_LOCAL	MIOC	0xfffc0000	Page 11–57
FLEXID	MIOC	0xfffc0020	Page 11–58
GSC_TIMEOUT	MIOC	0xf###f0f0	Page 11–50
GSC15X_CONFIG	MIOC	0xf###f7a0	Page 11–56
HIDMAMEM	MIOC	0xf###f0f4	Page 11–51
HTLB_ADDR	Page 0	24	Page 11–14
HTLB_CFG	Page 0	25	Page 11–15
IATOER0	Page 1	3	Debug*
IATOR0	Page 1	2	Debug*
IATRPNER0	Page 1	7	Debug*
IATRPNR0	Page 1	6	Debug*
IATSIER0	Page 1	5	Debug*
IATSIR0	Page 1	4	Debug*
IC_BIST_ADDR_CFG	Page 0	10	Page 11–12
IC_TIMING	I–cache	0xf###e280	Page 11–28
ICBAR_BL	I–cache	0xf###e220–8	Page 11–23
ICBAR_BR	I–cache	0xf###e22c–34	Page 11–24
ICBAR_T0	I–cache	0xf###e238	Page 11–24
ICBAR_T1	I–cache	0xf###e23c	Page 11–24
ICBAR_TL	I–cache	0xf###e200–8	Page 11–23
ICBAR_TR	I–cache	0xf###e20c–14	Page 11–23
ICPFBDATAA	I–cache	0xf###e240–4	Page 11–25
ICPFBDATAB	I–cache	0xf###e250–4	Page 11–26
ICPFBDATAC	I–cache	0xf###e260–4	Page 11–26
ICPFBDATAD	I–cache	0xf###e270–4	Page 11–27
ICPFBSPA	I–cache	0xf###e248	Page 11–25
ICPFBSPB	I–cache	0xf###e258	Page 11–26
ICPFBSPC	I–cache	0xf###e268	Page 11–27
ICPFBSPD	I–cache	0xf###e278	Page 11–28
ICPFBTAG	I–cache	0xf###e284	Page 11–29
ILAB_RPN	Page 0	9	Page 11–12
ILAB_VPN	Page 0	7	Page 11–10
ITERCNT	Page 1	24	Debug*
ITIMER	Page 0	1	Page 11–8
LITER0	Page 1	14	Debug*
LITR0	Page 1	13	Debug*
M_ERR_BYTE	Page 0	6	Page 11–10
M_ERR0	Page 0	3	Page 11–9
M_ERR1	Page 0	4	Page 11–9
M_RD_CHK	Page 0	5	Page 11–9
M_WRT_CHK	Page 0	2	Page 11–8
MDERRADD	MIOC	0xf###f0e0	Page 11–47
MEMCOMP[0:15]	MIOC	0xf###f100–3c	Page 11–52
MEMMASK[0:15]	MIOC	0xf###f140–7c	Page 11–53
MEMTEST	MIOC	0xf###f180	Page 11–54
MINTERM	Page 1	8	Debug*
MIOC_CONTROL	MIOC	0xf###f080	Page 11–33
MIOC_STATUS	MIOC	0xf###f084	Page 11–35
MTCV	MIOC	0xf###f0c0	Page 11–43
OUTCHK	MIOC	0xf###f1c0	Page 11–55
REF	MIOC	0xf###f0cc	Page 11–45
RITER0	Page 1	16	Debug*
RITR0	Page 1	15	Debug*
SCRATCH	Page 0	12	Page 11–14
SLTCV	MIOC	0xf###f0a0	Page 11–37
SLTEADD	MIOC	0xf###f0b0	Page 11–42
SLTESTAT	MIOC	0xf###f0ac	Page 11–41
TAGMASK	MIOC	0xf###f0a4	Page 11–39
TLB	Page 0	8	Page 11–11
TRIGCTL	Page 1	26	Debug*

*see the PA7300LC Debug Document.

11.2 Register Summaries

Table 1. Summary of PA7300LC Page Zero Diagnose Registers

Name	R/W	Register Number	Function	Defined on
CPU_CFG	r/w[1]	0	Enables/disables a variety of CPU features; also has some status bits.	Page 11–6
ITIMER	w	1	Sets a specific value into CR 16 count (not the compare value).	Page 11–8
M_WRT_CHK	r/w	2	Set the DRAM write check bits	Page 11–8
M_ERR0	r/w	3	Raw read data for MSW	Page 11–9
M_ERR1	r/w	4	Raw read data for LSW	Page 11–9
M_RD_CHK	r/w	5	Raw read data for 8 check bits	Page 11–9
M_ERR_BYTE	r/w	6	Byte pointer to error bit	Page 11–10
ILAB_VPN	r/w	7	Data for ILAB reads and writes.	Page 11–10
TLB	r/w[2]	8	Data/locking bits for UTLB	Page 11–11
ILAB_RPN	r/w	9	Data for ILAB reads and writes.	Page 11–12
IC_BIST_ADDR_CFG	r/w	10	Address LFSR and diagnose/bist control for the Instr Cache	Page 11–12
DC_BIST_ADDR_CFG	r/w	11	Address LFSR and diagnose/bist control for the Data Cache	Page 11–13
SCRATCH	r/w	12	Scratch register for HPMC handler to test HBT	Page 11–14
ACCEL_IO	w	13	Accelerated I/O physical address range configuration	Page 11–14
HTLB_ADDR	w	24	Address for TLB miss hardware handler	Page 11–14
HTLB_CFG	r/w[3]	25	TLB miss hardware table size and CR update configuration	Page 11–15

1 – not all bits are writeable; some clear when corresponding bit in GR is set
2 – not all bits are readable
3 – some read–only, some write–only, some r/w

Table 2. Summary of PA7300LC Page One Diagnose Registers (Debug)

Name	R/W	Register Number	Function	Defined on
CPU_CFG	r/w[1]	0	Same as on page 0; this register responds independent of DRPAGE1H	Page 11–6

1 – not all bits are writeable; some clear when corresponding bit in GR is set

Table 3. Summary of PA7300LC IO Addressed Data Cache Diagnose Registers

Name	R/W	Address[1]	Function	Defined on
DC_DW0_IN	w	0xf###e100,e104	Diagnose write data for doubleword 0	Page 11–17
DC_DW0P_IN	w	0xf###e108	2 bit diagnose write data for DW0 parity bits	Page 11–17
DC_DW1_IN	w	0xf###e110,e114	Diagnose write data for doubleword 1	Page 11–17
DC_DW1P_IN	w	0xf###e118	2 bit diagnose write data for DW1 parity bits	Page 11–17
DC_DW2_IN	w	0xf###e120,e124	Diagnose write data for doubleword 2	Page 11–18
DC_DW2P_IN	w	0xf###e128	2 bit diagnose write data for DW2 parity bits	Page 11–18
DC_DW3_IN	w	0xf###e130,e134	Diagnose write data for doubleword 3	Page 11–18
DC_DW3P_IN	w	0xf###e138	2 bit diagnose write data for DW3 parity bits	Page 11–18
DC_DW0_OUT	r/c[2]	0xf###e140,e144	Signature analyzer/diagnose read data for doubleword 0	Page 11–19
DC_DW0P_OUT	r/c	0xf###e148	2 bit signature analyzer/diagnose read data for DW0 parity bits	Page 11–19
DC_DW1_OUT	r/c	0xf###e150,e154	Signature analyzer/diagnose read data for doubleword 1	Page 11–19
DC_DW1P_OUT	r/c	0xf###e158	2 bit signature analyzer/diagnose read data for DW1 parity bits	Page 11–19
DC_DW2_OUT	r/c	0xf###e160,e164	Signature analyzer/diagnose read data for doubleword 2	Page 11–20
DC_DW2P_OUT	r/c	0xf###e168	2 bit signature analyzer/diagnose read data for DW2 parity bits	Page 11–20
DC_DW3_OUT	r/c	0xf###e170,e174	Signature analyzer/diagnose read data for doubleword 3	Page 11–20
DC_DW3P_OUT	r/c	0xf###e178	2 bit signature analyzer/diagnose read data for DW3 parity bits	Page 11–20
DTAG0_OUT	r/c	0xf###e180	Catches tag and dirty bit array bits for BIST and diagnose reads	Page 11–21
DTAG1_OUT	r/c	0xf###e184	Same as above, but for group 1 instead of group 0	Page 11–21
DTAG_IN	w	0xf###e188	Allows writing data dache tag, dirty and parity bits	Page 11–22
DC_TIMING	w	0xf###e18c	Data cache clock edge timing control	Page 11–22

1 – ‘###’ is determined by the FLEXID register and defaults to ‘ffb’

2 – all data output registers are cleared by writing to any of them, and both tag/dirty signature analyzer registers clear when writing to either of them.

Table 4. Summary of PA7300LC IO Addressed Instruction Cache Diagnose Registers

Name	R/W	Address[1]	Function	Defined on
ICBAR_TL	r/c[2]	0xf###e200–e208	BIST signature analyzer register, top left	Page 11–23
ICBAR_TR	r/c	0xf###e20c–e214	BIST signature analyzer register, top right	Page 11–23
ICBAR_BL	r/c	0xf###e220–e228	BIST signature analyzer register, bottom left	Page 11–23
ICBAR_BR	r/c	0xf###e22c–e234	BIST signature analyzer register, bottom right	Page 11–24
ICBAR_T0	r/c	0xf###e238	BIST signature analyzer register, group 0 tag and tag parity	Page 11–24
ICBAR_T1	r/c	0xf###e23c	BIST signature analyzer register, group 1 tag and tag parity	Page 11–24
ICPFBDATAA	w	0xf###e240,e244	Data for PFB write, either Group 0 DW 0 or Group 1 DW 1	Page 11–25
ICPFBSPA	w	0xf###e248	Steering for ICPFBDATAA and force parity error for ICPFBDATAA and ICPFBDATAC	Page 11–25
ICPFBDATAB	w	0xf###e250,e254	Data for PFB write, either Group 0 DW 1 or Group 1 DW 0	Page 11–26
ICPFBSPB	w	0xf###e258	Steering for ICPFBDATAB and force parity error for ICPFBDATAB and ICPFBDATAD	Page 11–26
ICPFBDATAC	w	0xf###e260,e264	Data for PFB write, either Group 0 DW 2 or Group 1 DW 3	Page 11–26
ICPFBSPC	w	0xf###e268	Steering for ICPFBDATAC and force parity error for ICPFBDATAA and ICPFBDATAC	Page 11–27
ICPFBDATAD	w	0xf###e270,e274	Data for PFB write, either Group 0 DW 3 or Group 1 DW 2	Page 11–27
ICPFBSPD	w	0xf###e278	Steering for ICPFBDATAD and force parity error for ICPFBDATAB and ICPFBDATAD	Page 11–28
IC_TIMING	w	0xf###e280	Instr Cache timing configuration	Page 11–28
ICPFBTAG	w	0xf###e284	Tag and force parity error bits for Prefetch Buf	Page 11–29

1 – '###' is determined by the FLEXID register and defaults to 'ffb'

2 – all data output registers are cleared by writing to any of them, and both tag signature analyzer registers clear when writing to either of them.

Table 5. Summary of PA7300LC IO Addressed Memory and IO Controller Diagnose Registers

Name	R/W	Address[1]	Function	Defined on
EIR	w	0xf###e000	Architected external interrupt register (processor HPA)	Page 11–31
COMMAND	w	0xf###e030	Architected I/O command register (processor HPA). Causes a TOC.	Page 11–32
MIOC_CONTROL	r/w	0xf###f080	General MIOC configuration and control	Page 11–33
MIOC_STATUS	r/t[2]	0xf###f084	General MIOC status	Page 11–35
SLTCV	r/w	0xf###f0a0	Second Level Cache enable and configuration	Page 11–37
TAGMASK	r/w	0xf###f0a4	Selects compare bits depending on size of SLC	Page 11–39
DIAGTAG	r/w	0xf###f0a8	Tag used for invalidates of SLC lines	Page 11–40
SLTESTAT	r/w	0xf###f0ac	Logged SLC tag and hit indications	Page 11–41
SLTEADD	r/w	0xf###f0b0	Logged SLC real address	Page 11–42
MTCV	r/w	0xf###f0c0	Memory configuration	Page 11–43
REF	r/w	0xf###f0cc	DRAM refresh and output signal control	Page 11–45
MDERRADD	r/w	0xf###f0e0	Doubleword address of least recent, most severe memory error	Page 11–47
DMAERR	r/w	0xf###f0e8	Address of least recent, most severe DMA error	Page 11–48
DIOERR	r/w	0xf###f0ec	Address of least recent, most severe DIO error	Page 11–49
GSC_TIMEOUT	r/w	0xf###f0f0	Delay to signalling a GSC timeout error	Page 11–50
HIDMAMEM	r/w	0xf###f0f4	Size of configured memory, for DMA transactions	Page 11–51
MEMCOMP[0:15]	w	0xf###f100–f13c	Programmed addresses for the sixteen memory address comparators	Page 11–52
MEMMASK[0:15]	w	0xf###f140–f17c	Programmed masks for the sixteen memory address comparators	Page 11–53
MEMTEST	w	0xf###f180	Address used to test memory address comparators	Page 11–54
OUTCHK	r	0xf###f1c0	Memory address comparator test outputs	Page 11–55
GSC15X_CONFIG	r/w	0xf###f7a0	Configures ranges in I/O space that accept WRITEV transactions	Page 11–56
EIR_LOCAL	w	0xfffc0000	Architected external interrupt register (local broadcast)	Page 11–57
FLEXID	w	0xfffc0020	Architected flex register. Sets processor and MIOC HPA addresses	Page 11–58
COMMAND_LOCAL	w	0xfffc0030	Architected I/O command register (local broadcast). Causes a reset.	Page 11–59
EIR_GLOBAL	w	0xfffe0000	Architected external interrupt register (global broadcast)	Page 11–60
COMMAND_GLOBAL	w	0xfffe0030	Architected I/O command register (global broadcast). Causes a reset.	Page 11–61

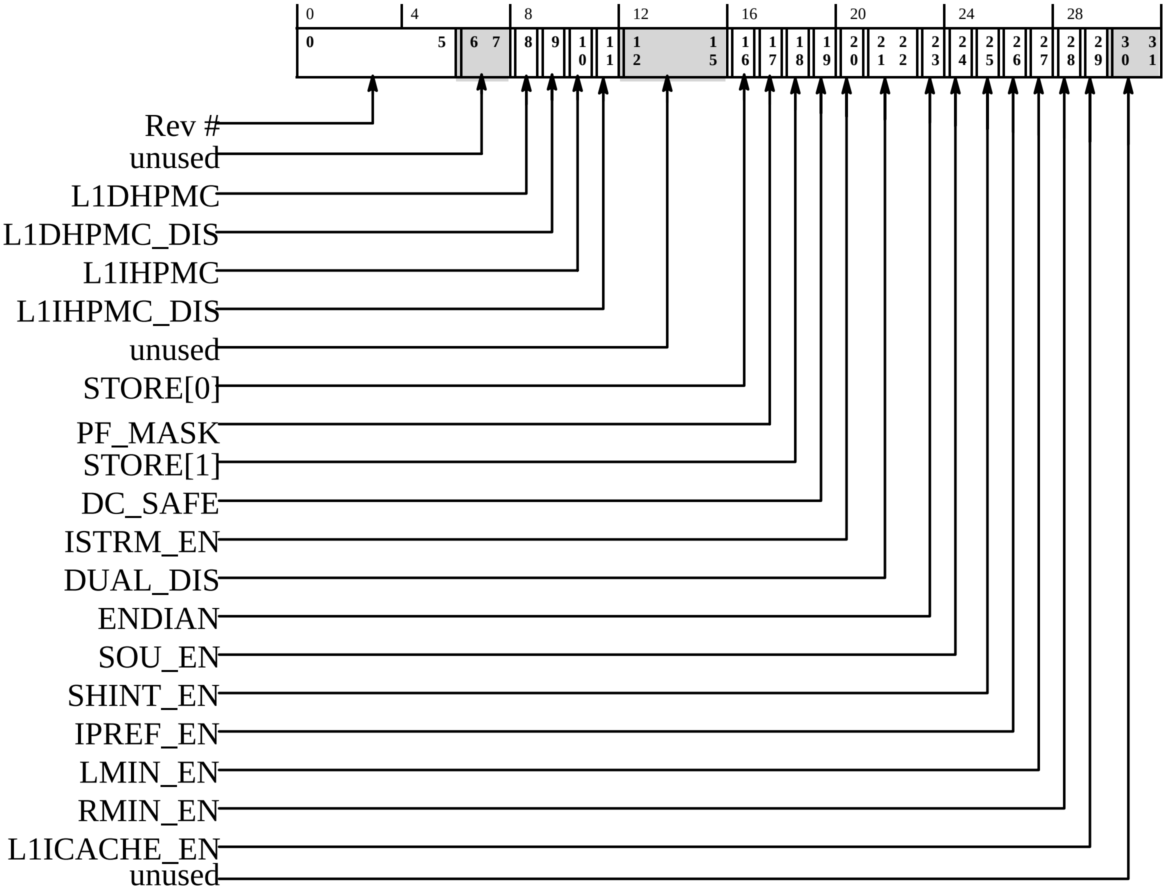
[1] – ‘###’ is determined by the FLEXID register and defaults to ‘ffb’

[2] – Some bits toggle when a ‘1’ is written to their position

11.3 CPU Diagnose Registers

11.3.1 CPU_CFG — CPU Diagnose Register 0, Page 0

The format of the CPU Diagnose Register 0 is shown below:



Field	Name	R/W	Description
0:5	Rev #	r	CPU Revision Number. This holds the revision number of the CPU chip. For the first release of the CPU the revision number is 0.
6:7	unused		Reads return 0000.
8	L1DHPMC	r/c	Level1 (on–chip) D–Cache Error Flag. This bit is set whenever a Level1 (on–chip) D–Cache Parity Error is detected. It is qualified by the L1DHPMC_DIS bit before causing a trap. This bit remains set until software clears it in the HPMC handler. It is cleared only when a ’1’ is moved to this bit position. Moving a ’0’ to this bit position does not change this bit.

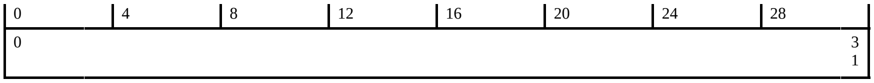
9	L1DHPMC_DIS	r/w	Level1 (on–chip) D–Cache HPMC Disable (Mask). This bits disable taking an HPMC due to a Level1 (on–chip) D–Cache Parity Error. This is provided as a debug feature for early CPU releases. This is more accurately called a mask bit because HPMCs are still collected and are held pending.
10	L1IHPMC	r/c	Level1 (on–chip) I–Cache Error Flag. This bit is set whenever a Level1 (on–chip) ICache Parity Error is detected. It is qualified by the L1IHPMC_DIS bit before causing a trap. This bit remains set until software clears it in the HPMC handler. It is cleared only when a ’1’ is moved to this bit position. Moving a ’0’ to this bit position does not change this bit.
11	L1IHPMC_DIS	r/w	Level1 (on–chip) I–Cache HPMC Disable (Mask). This bits disable taking an HPMC due to a Level1 (on–chip) ICache Parity Error. This is provided as a debug feature for early CPU releases. This is more accurately called a mask bit because HPMCs are still collected and are held pending.
12:15	unused		
16	STORE[0]	r/w	Scratch Space. These bits provide for temporary storage. One self–test convention for one of these is to indicate when an HPMC is expected by self–test code rather than ’real’.
17	PF_MASK	r/w	Power–fail trap mask. When this bit is set to 1, power–fail traps are disabled.
18	STORE[1]	r/w	Scratch Space. These bits provide for temporary storage. One self–test convention for one of these is to indicate when an HPMC is expected by self–test code rather than ’real’.
19	DC_SAFE	r/w	Serialize all data cache hangs. Disables overlapping of data cache operations when set to 1.
20	ISTRM_EN	r/w	Enable ICache streaming. Enable instruction cache streaming for instruction fetches from memory. Disable only to debug prototype systems.
21:22	DUAL_DIS	r/w	Disable Dual–Issue (superscalar ececution). Disable Dual–Issue (superscalar execution). The following options are supported: 00=all bundles enabled, 01=all bundles except ldw/ldw,stw/stw are enabled, 10=only flop–non_flop bundles are enabled, 11=no bundles are enabled (single issue mode). Disable only to debug prototype hardware.
23	ENDIAN	r/w	Use Little_Endian mode when taking a trap. Set this bit to enable Little–Endian mode for traps and hardware TLB accesses (ie. on traps, this bit is set into the PSW–E bit, and this bit is used in place of the PSW–E bit for HTLB accesses).
24	SOU_EN	r/w	Stall–on–Use enable for Data Cache Misses. Enable the “Stall–on–Use” optimization for D–Misses (load misses only). Disable only to debug prototype hardware.
25	SHINT_EN	r/w	No–Fill on Miss Store Hints enable. Enable Store Hints for store instructions. Disable only to debug prototype hardware.
26	IPREF_EN	r/w	L2 to L1 Instruction cache prefetch enable. Enable L2 to L1 instruction cache prefetching. Disable only to debug prototype hardware.
27	LMIN_EN	r/w	Enable taking illegal instruction traps for an illegal (undefined) minor opcode on the LIH instruction bus.

28	RMIN_EN	r/w	Enable taking illegal instruction traps for an illegal (undefined) minor opcode on the RIH instruction bus.
29	L1ICACHE_EN	r/w	Level1 (on–chip) I–Cache Enable. L1ICACHE_EN: Enable the Level1 (on–chip) instruction cache. Disable only to debug prototype hardware or to operate in degraded mode after sensing a permanent cache error. Disabling the L1 ICache does not automatically disable HPMC's from L1 ICache. Software should set L1IHPMC_DIS when clearing L1ICACHE_EN. See the Software Constraints for another important restriction.
30:31	unused		

(r) means Read–Only
(r/w) means Read–Write
(r/c) means Read–Clear (Clear by Moving '1' to it)

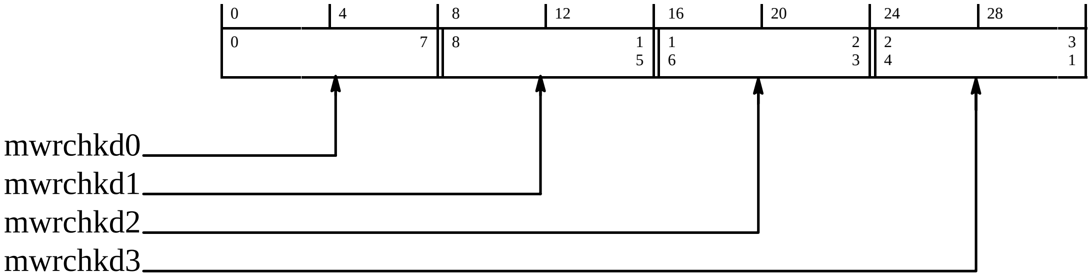
POWERUP values:
DR0[0:5] powers up with the revision number
DR0[6:15] powers up with undefined values
DR0[16:31] powers up to zero, except [22]=1

11.3.2 ITIMER — Diagnose Register 1, Page 0



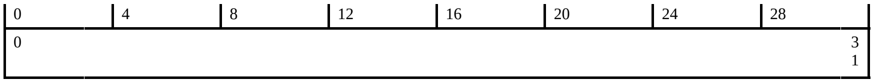
This register allows setting of the architected Interval Timer value. This counter is normally only readable, through the move from control register instruction MFCTL 16,r. When using the move to control register instruction, only the compare value is set, not the actual timer register value. By using a MTCPU_C r,1 instruction (described in section ??) the actual timer count may be set. This is intended for use by pre–production testing only.

11.3.3 M_WRT_CHK — Diagnose Register 2, Page 0



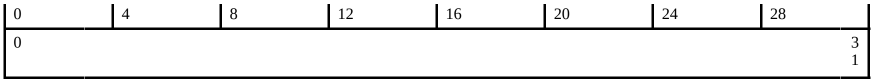
Field	Name	R/W	Description
0:7	mwrchkd0	r/w	Doubleword 0 check bits for next memory write (if memory checkbit generation is disabled) (addr[27:28] == ‘00’)
8:15	mwrchkd1	r/w	Doubleword 1 check bits for next memory write (addr[27:28] == ‘01’)
16:23	mwrchkd2	r/w	Doubleword 2 check bits for next memory write (addr[27:28] == ‘10’)
24:31	mwrchkd3	r/w	Doubleword 3 check bits for next memory write (addr[27:28] == ‘11’)

11.3.4 M_ERR0 — Diagnose Register 3, Page 0



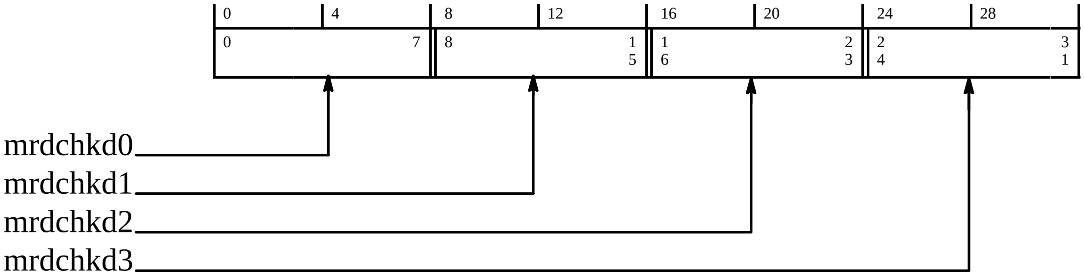
The most significant data word of the last logged memory error.

11.3.5 M_ERR1 — Diagnose Register 4, Page 0



The least significant data word of the last logged memory error.

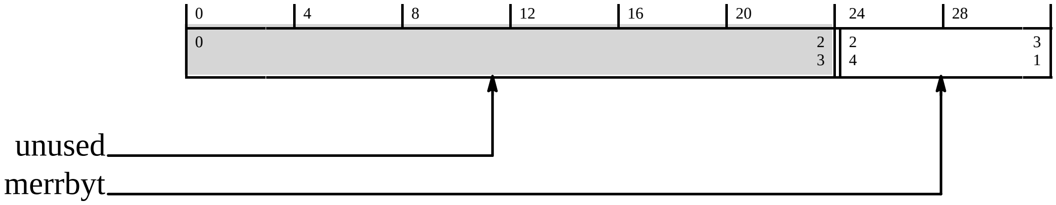
11.3.6 M_RD_CHK — Diagnose Register 5, Page 0



Field	Name	R/W	Description
0:7	mrdchkd0	r/w	Doubleword 0 check bits from most recent memory read (addr[27:28] == ‘00’)
8:15	mrdchkd1	r/w	Doubleword 1 check bits from most recent memory read (addr[27:28] == ‘01’)

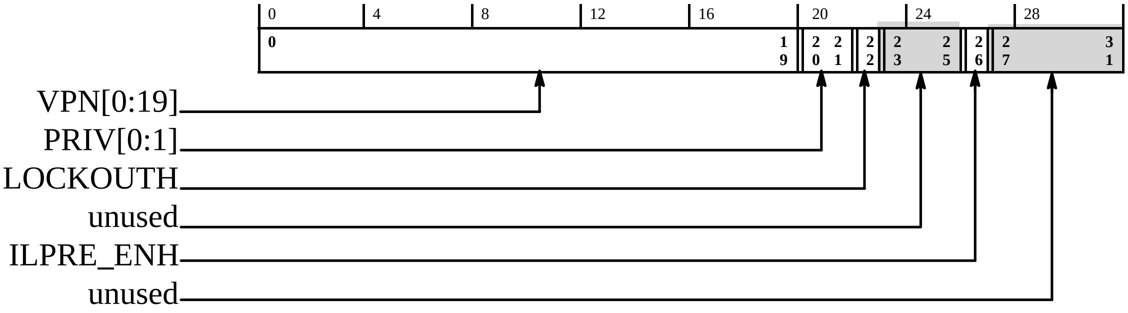
16:23	mrdchkd2	r/w	Doubleword 2 check bits from most recent memory read (addr[27:28] == ‘10’)
24:31	mrdchkd3	r/w	Doubleword 3 check bits from most recent memory read (addr[27:28] == ‘11’)

11.3.7 M_ERR_BYTE — Diagnose Register 6, Page 0



Field	Name	R/W	Description
0:23	unused		
24:31	merrbyt	r/w	Check byte of the last logged memory error

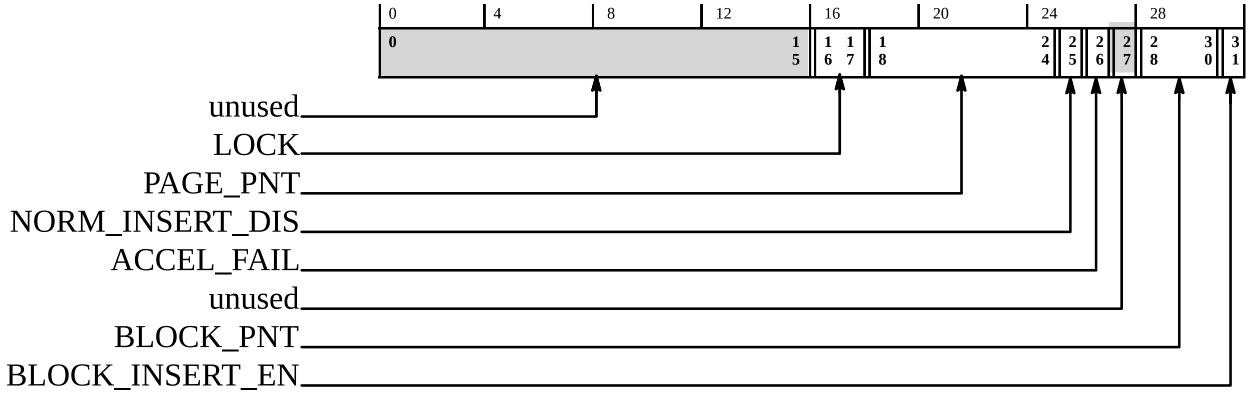
11.3.8 ILAB_VPN — Diagnose Register 7, Page 0



Field	Name	R/W	Description
0:19	VPN	r/w	virtual page used in associative compare with fetch address
20:21	PRIV	r/w	privilege bits used in associative compare
22	LOCKOUTH	r/w	intended only for pre–production test; prevent some entries from matching.
23:25	unused		
26	ILPRE_ENH	r/w	enable prefetching of next sequential translation from UTLB to avoid ILAB miss
27:31	unused		

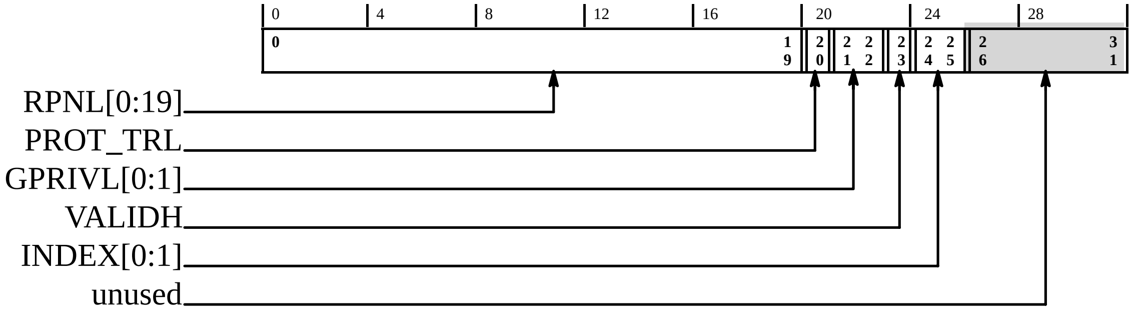
11.3.9 TLB — Diagnose Register 8, Page 0

A Diagnose Control register is defined for the TLB. Thus is a 16–bit register which is loaded via Move_to_Diagnose instructions (see Diagnose Chapter for encoding). This diagnose register has partial–read capability as shown below.



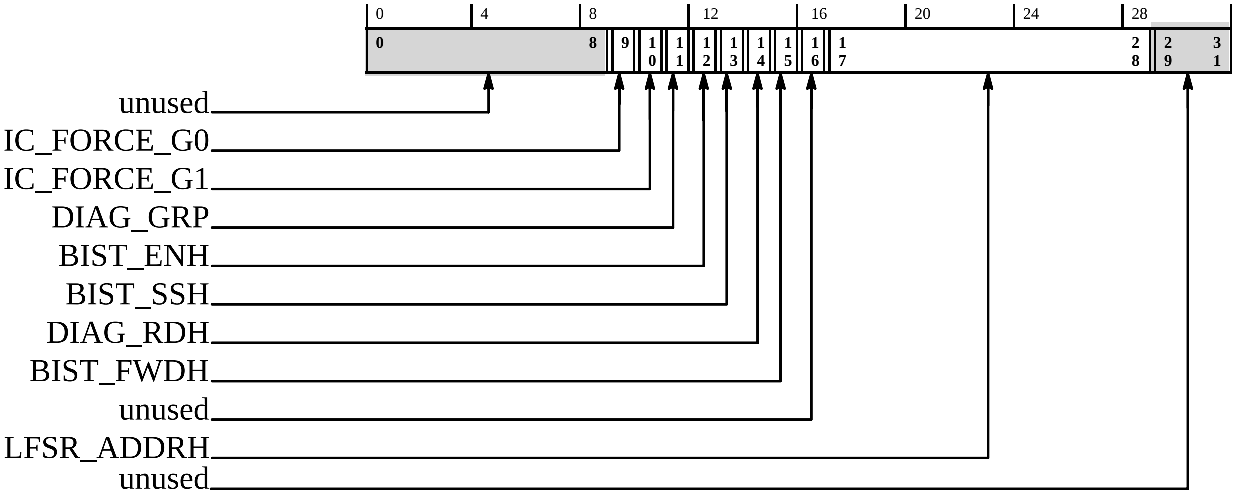
Field	Name	R/W	Description
0:15	unused		
16:17	LOCK	r/w	00 – subsequently inserted addresses will be neither locked in nor locked out 01 – lock–in: subsequently inserted addresses are shielded from replacement unless they become the targets of a diagnostic insertion or they match the VPN of some future translation to be inserted 11 – lock–out: subsequently inserted addresses will be forced to mismatch on every translation attempt 10 – undefined operation will result feature disabled value: 0
18:24	PAGE_PNT	w	7–bit unsigned integer (bit 18 is msb). If within the range [0:95], subsequent insert instructions (IxTLBx) will insert to the page–entry specified (AND to any page–entry that has a SID–VPN match). feature disabled value: 0x7f
25	NORM_INSERT_DIS	r/w	While set, the normal LRU replacement algorithm that selects page entries for replacement on IxTLBx instructions will be disabled. Set this bit when you want to target a specific page–entry or block–entry for insertion. Set to 0 for normal operation. feature disabled value: 0
26	ACCEL_FAIL	w	Provided as a test feature for wafer screen. CODE MUST GUARANTEE THIS BIT IS CLEARED (ZERO) AT ALL TIMES. feature disabled value: 0
27	unused		
28:30	BLOCK_PNT	w	3–bit unsigned integer (bit 28 is msb). It indexes the Block Entry targeted for inserts when DR8[31]=1. feature disabled value: don’t care
31	BLOCK_INSERT_EN	r/w	When set, subsequent insert instructions (IxTLBx) will insert to the block–entry pointed to by bits 28:30 (AND to any page–entry that has a SID–VPN match). Set to zero for normal operation. feature disabled value: 0

11.3.10 ILAB_RPN — Diagnose Register 9, Page 0



Field	Name	R/W	Description
0:19	RPNL[0:19]	r/w	Real page to be used for icache compare when corresponding entry's VPN and PRIV match the fetch address in virtual mode. Negative true.
20	PROT_TRL	r/w	Protection trap – cause an exception when the corresponding VPN & RPN are accessed. Negative true.
21:22	GPRIVL[0:1]	r/w	Gateway privilege from UTLB access rights field. Negative true.
23	VALIDH	r/w	Entry is valid (i.e., no ITLB miss trap will be signalled)
24:25	INDEX[0:1]	r/w	Pointer directing diagnose operations using these two registers
26:31	unused		

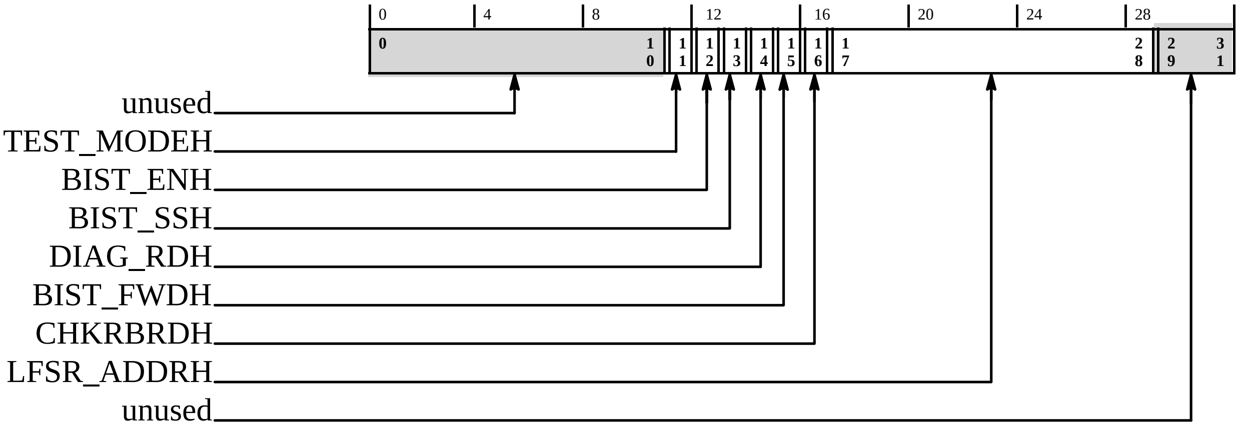
11.3.11 IC_BIST_ADDR_CFG — Diagnose Register 10, Page 0



Field	Name	R/W	Description
0:8	unused		
9	IC_FORCE_G0	r/w	forces group 0 replacement

10	IC_FORCE_G1	r/w	forces group 1 replacement
11	DIAG_GRP	r/w	selects group 0 or 1 for diagnose writes
12	BIST_ENH	r/w	BIST enable. When true, writes are half group 0, half group 1. Data writes are enabled when LFSR_ADDRH[28] == 1. Tag writes are enabled when LFSR_ADDRH[27:28] == 01.
13	BIST_SSH	r/w	causes BIST engine to stop after one address
14	DIAG_RDH	r/w	selects between read & write for diagnose
15	BIST_FWDH	r/w	controls direction of BIST address LFSR
16	unused		
17:28	LFSR_ADDRH	r/w	address reg shared by BIST and diagnose
29:31	unused		

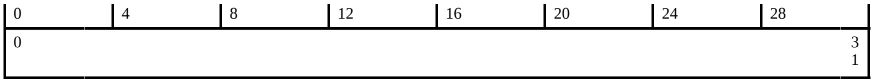
11.3.12 DC_BIST_ADDR_CFG — Diagnose Register 11, Page 0



Field	Name	R/W	Description
0:10	unused		
11	TEST_MODEH	w	must be set to 1 before doing diagnose or BIST*
12	BIST_ENH	w	controls 3 state bist vs. 1 state diag, & alt tag write disable
13	BIST_SSH	w	causes BIST engine to stop after one address
14	DIAG_RDH	w	selects between read & write for diagnose
15	BIST_FWDH	w	controls direction of BIST address LFSR
16	CHKRBRDH	w	data array will be accessed in checkerboad mode such that the data for an entire cache line may be read or written. The data group is determined by LFSR_ADDRH[27]. For diagnose reads and writes, the tag group is determined by LFSR_ADDRH[28].
17:28	LFSR_ADDRH	r/w	address reg shared by BIST and diagnose
29:31	unused		

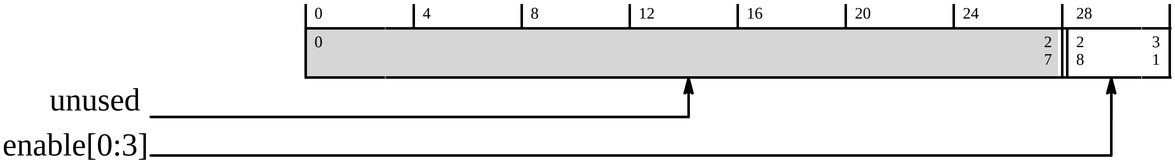
* store queue must be flushed first with 2 stores to I/O and two SYNC instructions must be executed before setting this bit

11.3.13 SCRATCH — Diagnose Register 12, Page 0



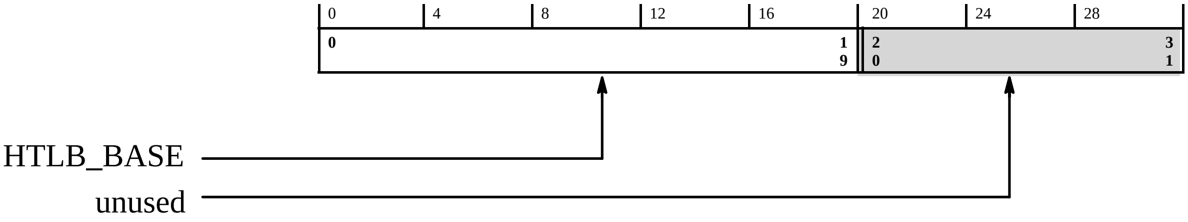
This register is provided as a convenience to temporarily store a value without referencing memory or I/O.

11.3.14 ACCEL_IO — Diagnose Register 13, Page 0



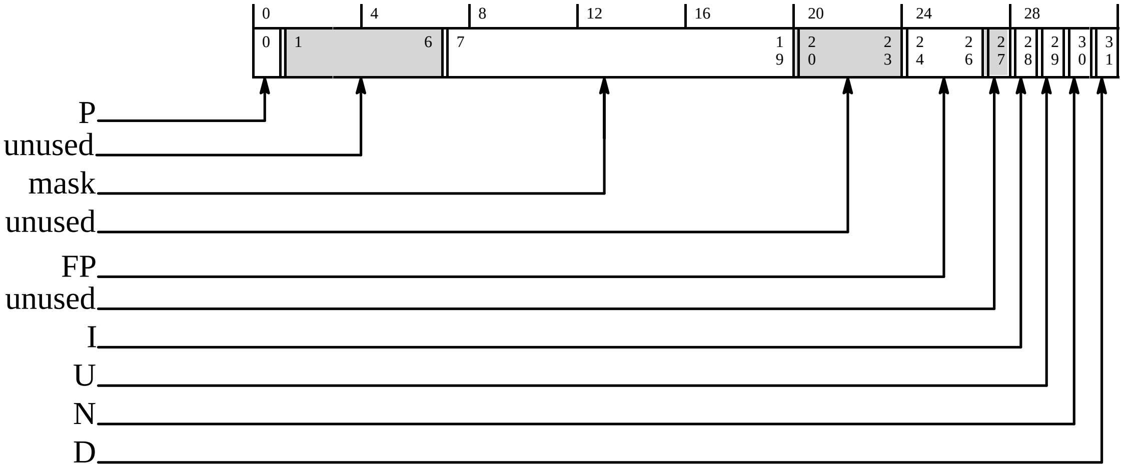
Field	Name	R/W	Description
0:27	unused		
28	enable[0]	w	Enable accelerated I/O writes to addresses within the range [0xf4000000, 0xf5ffffff].
29	enable[1]	w	Enable accelerated I/O writes to addresses within the range [0xf6000000, 0xf7ffffff].
30	enable[2]	w	Enable accelerated I/O writes to addresses within the range [0xf8000000, 0xf9ffffff].
31	enable[3]	w	Enable accelerated I/O writes to addresses within the range [0xfa000000, 0xfbffffff].

11.3.15 HTLB_ADDR — Diagnose Register 24, Page 0



Field	Name	R/W	Description
0:19	HTLB_BASE	w	HTLB Handler Base. Address in Main Memory where the Hardware visible “PDIR” table resides. See the TLB chapter.
20:31	unused		

11.3.16 HTLB_CFG — Diagnose Register 25, Page 0



Field	Name	R/W	Description
0	P	r	continuously latches the power–fail signal (read–only)
1:6	unused		
7:19	mask	w	Effectively sets the size of the hw–visible table by determining which bits come from the hashed address and which bits come from diag reg #24 (base address for table). mask [7:19] enables an ‘‘or” of base_addr[7:19] and hash_addr[7:19] Example: A ‘‘0” in mask[13] will set real_addr[13] = base_addr[13] A ‘‘1” in mask[13] will set real_addr[13] = hash_addr[13]
20:23	unused		
24:26	FP	r/w	Sets the FP delay as explained in the diagnose chapter.
27	unused		
28	I	r/w	Set to ‘‘1” to disable the ITLB hw handler, ‘‘0” to enable. It is not set by power–on or reset but rather by PDC code.
29	U	r/w	Set to ‘‘1” to enable the updating of CR28 with the next–pointer if the tag was valid and didn’t match the missing space/offset. If set to ‘‘0”, CR28 will always contain the address of the entry in the hw–visible table (current pdir).
30	N	r/w	Set to ‘‘1” to force the next pointer to always come from word3. If set to ‘‘0”, the next pointer will come from word3 if we use the entry in the even quadword of from word7 if we use the entry in the odd quadword.
31	D	r/w	Set to ‘‘1” to disable the DTLB hw handler, ‘‘0” to enable. It is not set by power–on or reset but rather by PDC code.

The minimum size of the hw–visible table is 4Kbytes (256 PDIR entries).
The maximum size of the hw–visible table is 32Mbytes.
The mask field is set based on the table size as shown below:

table size	entries	mask[0:19]
4 Kbytes	256	00000000000000000000
8 Kbytes	512	00000000000000000001
<etc>		
32 Mbytes	2M	00000001111111111111

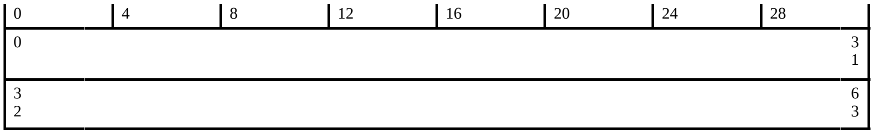
For every bit set in the mask field the corresponding bit in the base address must be a ”0”.

11.3.17 DEBUG — Diagnose Register 26, Page 0

Diagnose Register #26, for hardware debug, is described in the PA7300LC Debug Document, available from Hosein Naaseh.

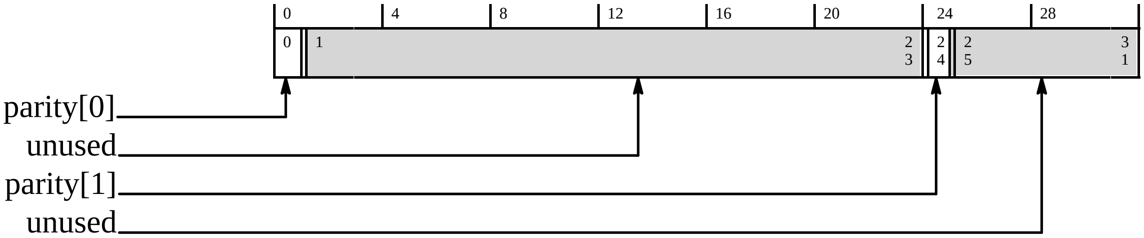
11.4 I/O Mapped Data Cache Diagnose Registers

11.4.1 DC_DW0_IN — I/O Address 0xf###e100, 0xf###e104



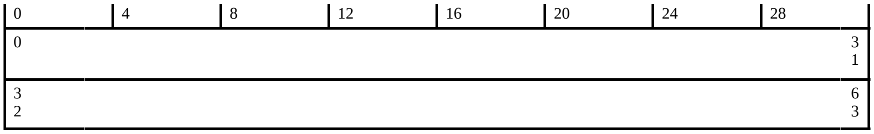
This contains a doubleword of diagnose or BIST data to be written to either doubleword 0 of group 0 or doubleword 1 of group 1.

11.4.2 DC_DW0P_IN — I/O Address 0xf###e108



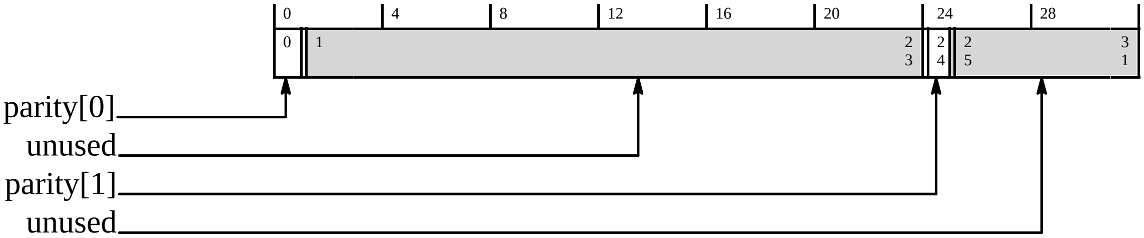
This contains the two (even) parity bits associated with DC_DW0_IN that will be written at the same time as DC_DW0_IN. parity[0] contains the parity bit for DC_DW0_IN[0:31] and parity[1] contains the parity bit for DC_DW0_IN[32:63].

11.4.3 DC_DW1_IN — I/O Address 0xf###e110, 0xf###e114



This contains a doubleword of diagnose or BIST data to be written to either doubleword 1 of group 0 or doubleword 0 of group 1.

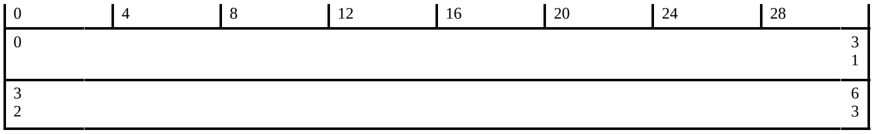
11.4.4 DC_DW1P_IN — I/O Address 0xf###e118



This contains the two (even) parity bits associated with DC_DW1_IN that will be written at the same time as

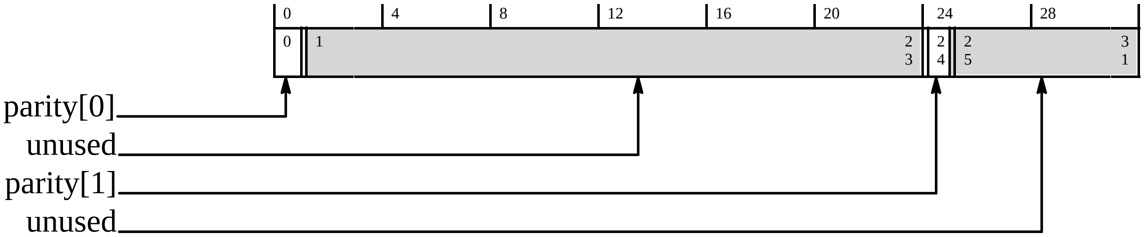
DC_DW1_IN. parity[0] contains the parity bit for DC_DW1_IN[0:31] and parity[1] contains the parity bit for DC_DW1_IN[32:63].

11.4.5 DC_DW2_IN — I/O Address 0xf###e120, 0xf###e124



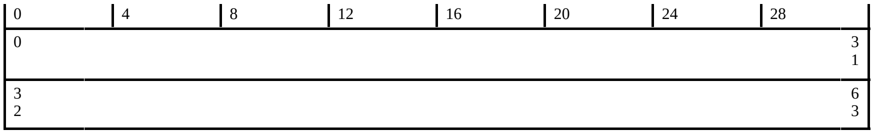
This contains a doubleword of diagnose or BIST data to be written to either doubleword 2 of group 0 or doubleword 3 of group 1.

11.4.6 DC_DW2P_IN — I/O Address 0xf###e128



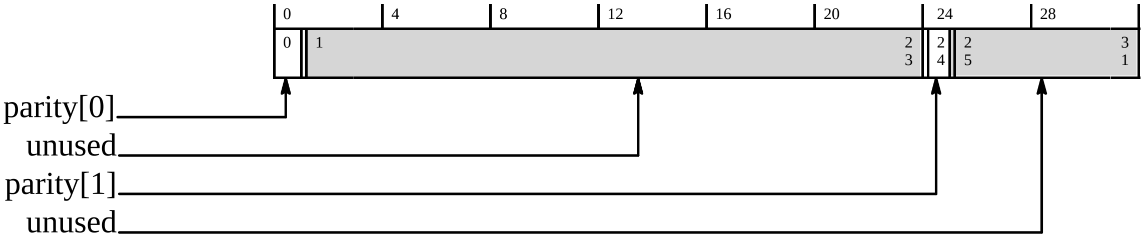
This contains the two (even) parity bits associated with DC_DW2_IN that will be written at the same time as DC_DW2_IN. parity[0] contains the parity bit for DC_DW2_IN[0:31] and parity[1] contains the parity bit for DC_DW2_IN[32:63].

11.4.7 DC_DW3_IN — I/O Address 0xf###e130, 0xf###e134



This contains a doubleword of diagnose or BIST data to be written to either doubleword 3 of group 0 or doubleword 2 of group 1.

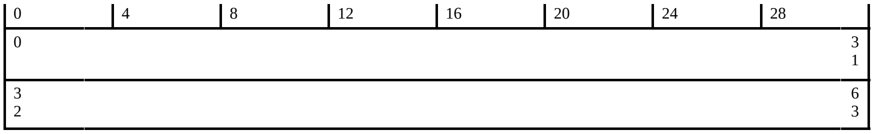
11.4.8 DC_DW3P_IN — I/O Address 0xf###e138



This contains the two (even) parity bits associated with DC_DW3_IN that will be written at the same time as

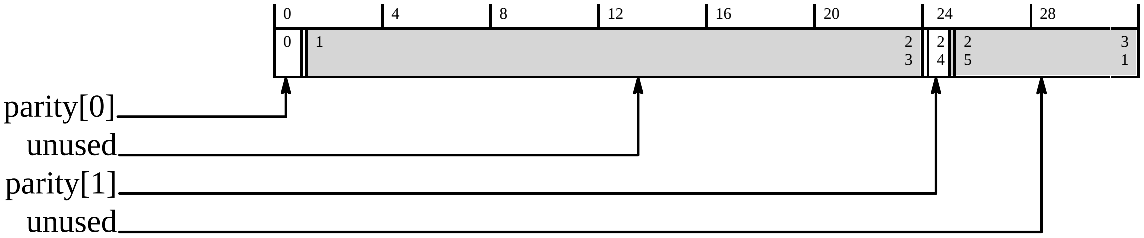
DC_DW3_IN. parity[0] contains the parity bit for DC_DW3_IN[0:31] and parity[1] contains the parity bit for DC_DW3_IN[32:63].

11.4.9 DC_DW0_OUT — I/O Address 0xf###e140, 0xf###e144



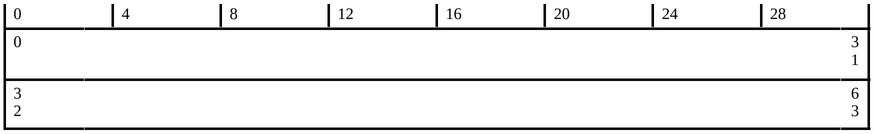
This contains a doubleword of diagnose or BIST signature data read from either doubleword 0 of group 0 or doubleword 1 of group 1.

11.4.10 DC_DW0P_OUT — I/O Address 0xf###e148



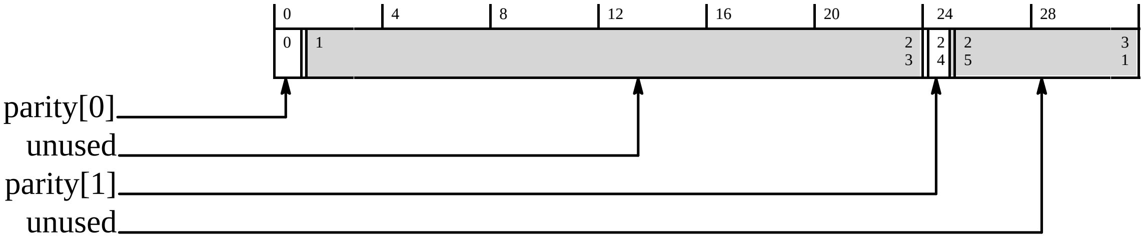
This contains the two (even) parity bits associated with DC_DW0_OUT that were read at the same time as DC_DW0_OUT. parity[0] contains the parity bit for DC_DW0_OUT[0:31] and parity[1] contains the parity bit for DC_DW0_OUT[32:63].

11.4.11 DC_DW1_OUT — I/O Address 0xf###e150, 0xf###e154



This contains a doubleword of diagnose or BIST signature data read from either doubleword 1 of group 0 or doubleword 0 of group 1.

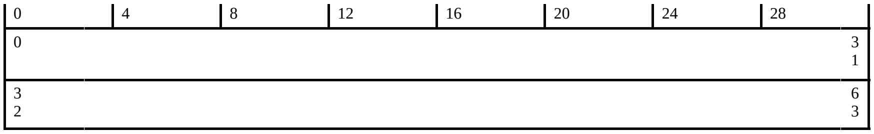
11.4.12 DC_DW1P_OUT — I/O Address 0xf###e158



This contains the two (even) parity bits associated with DC_DW1_OUT that were read at the same time as

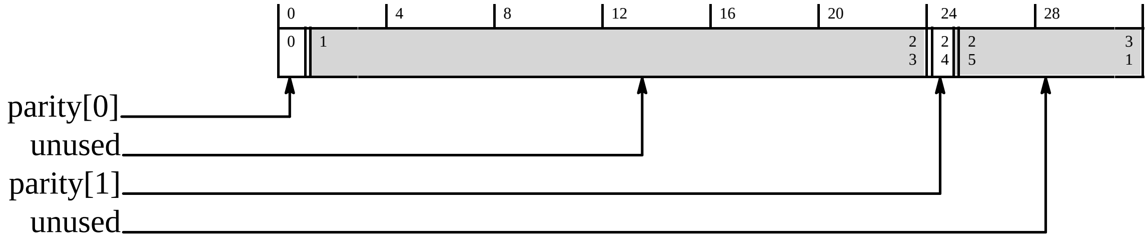
DC_DW1_OUT. parity[0] contains the parity bit for DC_DW1_OUT[0:31] and parity[1] contains the parity bit for DC_DW1_OUT[32:63].

11.4.13 DC_DW2_OUT — I/O Address 0xf###e160, 0xf###e164



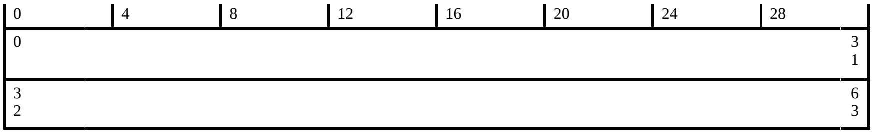
This contains a doubleword of diagnose or BIST signature data read from either doubleword 2 of group 0 or doubleword 3 of group 1.

11.4.14 DC_DW2P_OUT — I/O Address 0xf###e168



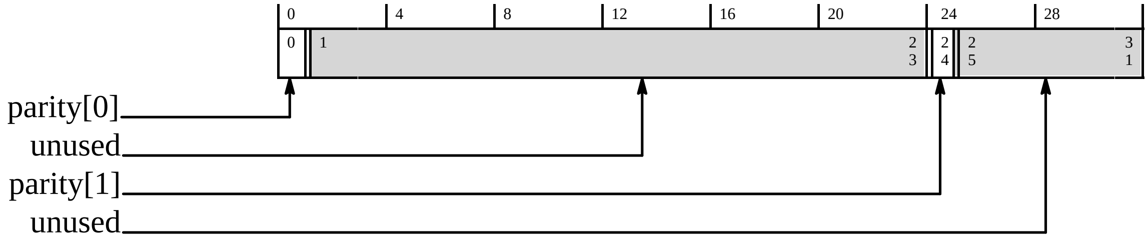
This contains the two (even) parity bits associated with DC_DW2_OUT that were read at the same time as DC_DW2_OUT. parity[0] contains the parity bit for DC_DW2_OUT[0:31] and parity[1] contains the parity bit for DC_DW2_OUT[32:63].

11.4.15 DC_DW3_OUT — I/O Address 0xf###e170, 0xf###e174



This contains a doubleword of diagnose or BIST signature data read from either doubleword 3 of group 0 or doubleword 2 of group 1.

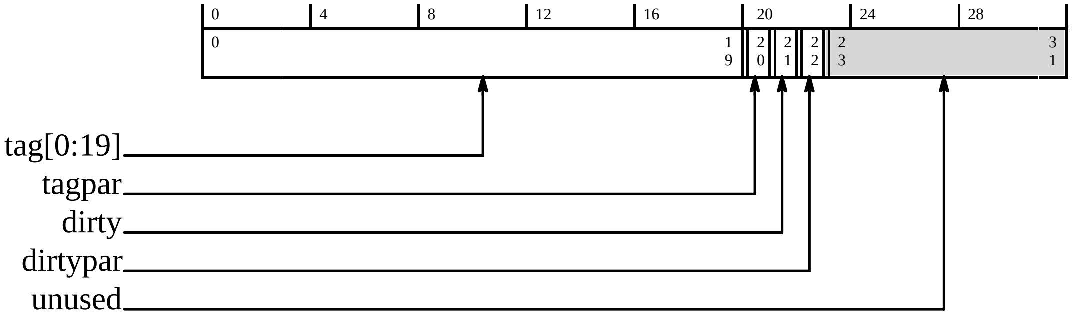
11.4.16 DC_DW3P_OUT — I/O Address 0xf###e178



This contains the two (even) parity bits associated with DC_DW3_OUT that were read at the same time as

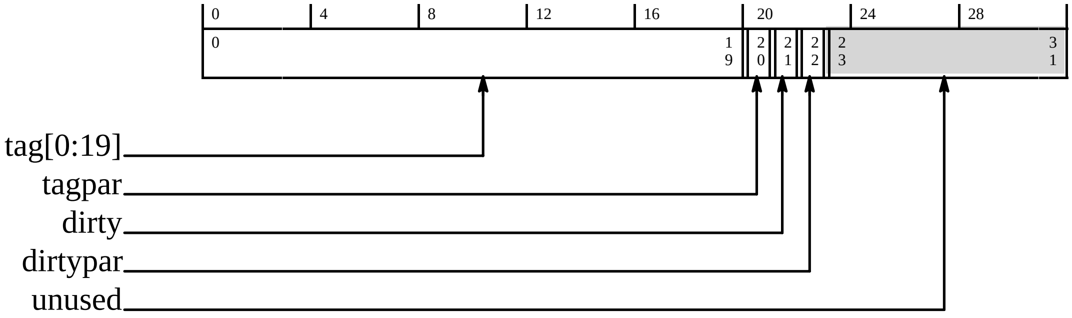
DC_DW3_OUT. parity[0] contains the parity bit for DC_DW3_OUT[0:31] and parity[1] contains the parity bit for DC_DW3_OUT[32:63].

11.4.17 DTAG0_OUT — I/O Address 0xf###e180



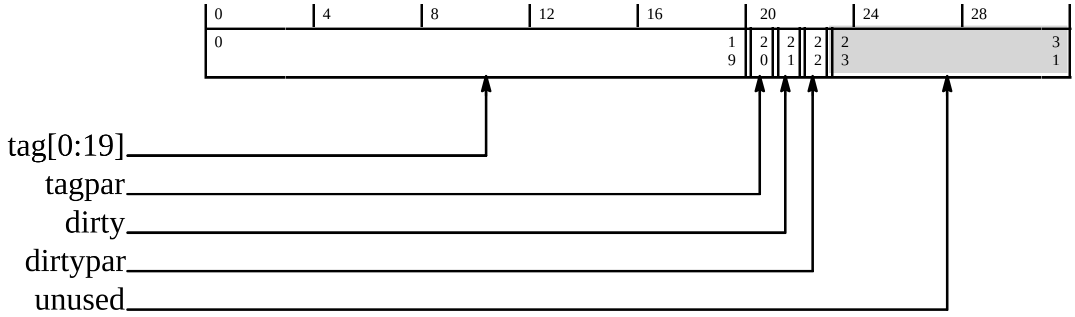
Field	Name	R/W	Description
0:19	tag	r/c	diagnose read data or BIST signature data for the group 0 tag
20	tagpar	r/c	parity bit (even) corresponding to tag[0:19]
21	dirty	r/c	diagnose read data or BIST signature data for the group 0 dirty bit
22	dirtypar	r/c	parity bit (even) corresponding to dirty bit
23:31	unused		

11.4.18 DTAG1_OUT — I/O Address 0xf###e184



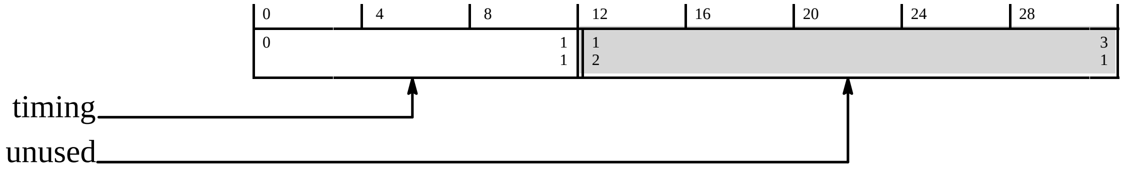
Field	Name	R/W	Description
0:19	tag	r/c	diagnose read data or BIST signature data for the group 1 tag
20	tagpar	r/c	parity bit (even) corresponding to tag[0:19]
21	dirty	r/c	diagnose read data or BIST signature data for the group 1 dirty bit
22	dirtypar	r/c	parity bit (even) corresponding to dirty bit
23:31	unused		

11.4.19 DTAG_IN — I/O Address 0xf###e188



Field	Name	R/W	Description
0:19	tag	w	Diagnose or BIST write data for tag (both groups)
20	tagpar	w	Parity bit corresponding to tag[0:19]. Even parity.
21	dirty	w	Diagnose or BIST write data for dirty bit (both groups)
22	dirtypar	w	Parity bit corresponding to dirty bit. Even parity.
23:31	unused		

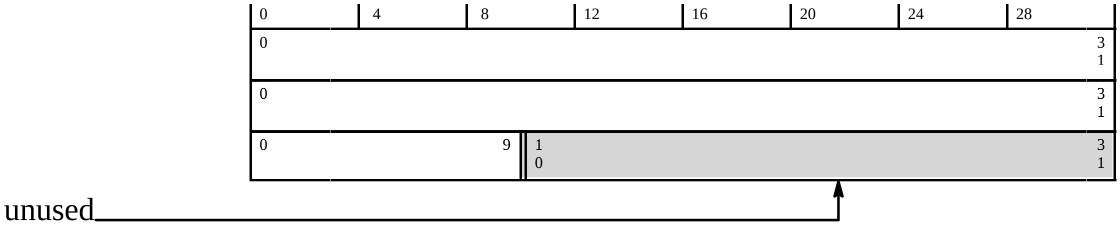
11.4.20 DC_TIMING — I/O Address 0xf###e18c



Field	Name	R/W	Description
0:11	timing	w	Frequency specific data cache timing configuration value. Resets to all zeros. This register will probably not be written to, but if it is, the value written will be empiracally determined from Phase II electrical characterization.
12:31	unused		

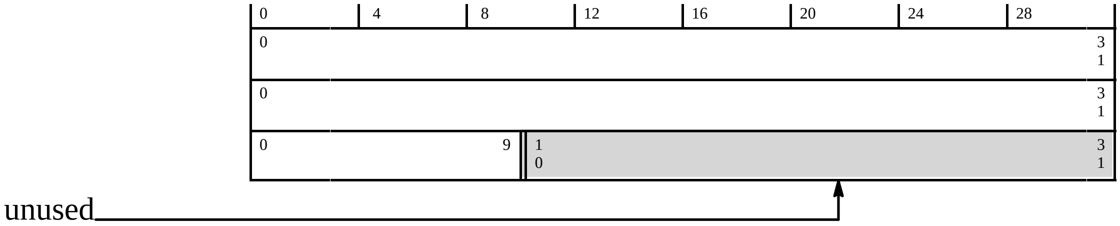
11.5 I/O Mapped Instruction Cache Diagnose Registers

11.5.1 ICBAR_TL — I/O Address 0xf###e200–0xf###e208



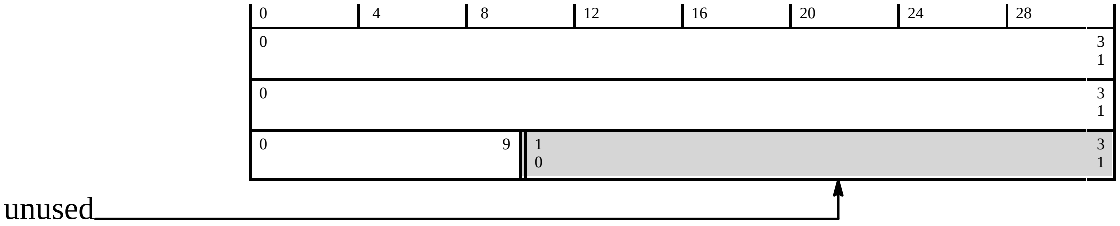
These three registers contain 74 bits of diagnose or BIST signature data associated with the top left quarter of the instruction cache data array.

11.5.2 ICBAR_TR — I/O Address 0xf###e20c–0xf###e214



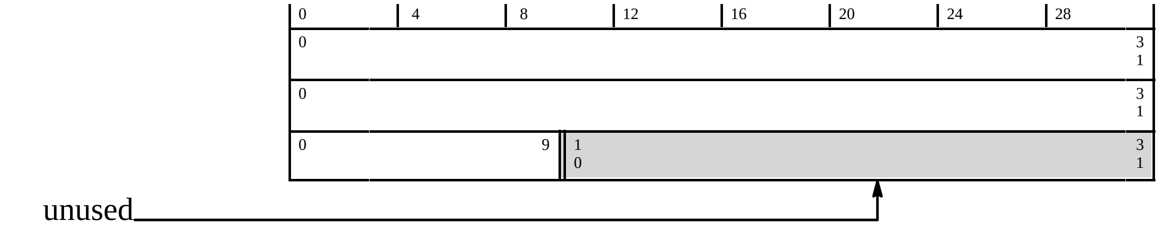
These three registers contain 74 bits of diagnose or BIST signature data associated with the top right quarter of the instruction cache data array.

11.5.3 ICBAR_BL — I/O Address 0xf###e220–0xf###e228



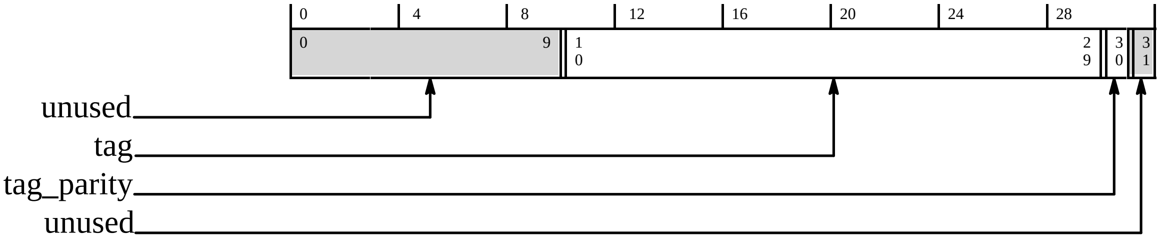
These three registers contain 74 bits of diagnose or BIST signature data associated with the bottom left quarter of the instruction cache data array.

11.5.4 ICBAR_BR — I/O Address 0xf###e22c–0xf###e234



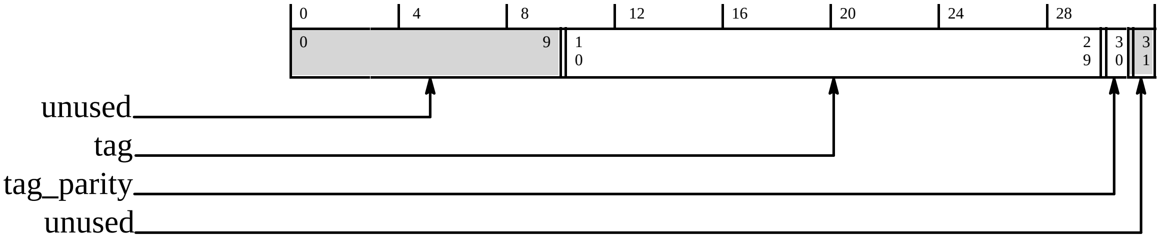
These three registers contain 74 bits of diagnose or BIST signature data associated with the bottom right quarter of the instruction cache data array.

11.5.5 ICBAR_T0 — I/O Address 0xf###e238



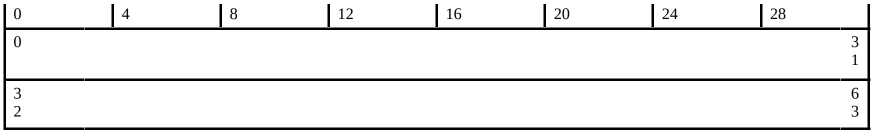
Field	Name	R/W	Description
0:9	unused	r/c	
10:29	tag	r/c	Contains diagnose or BIST signature data read from the group 0 tag
30	tag_parity	r/c	Contains the parity bit (even) associated with group 0 tag[0:19]
31	unused		

11.5.6 ICBAR_T1 — I/O Address 0xf###e23c



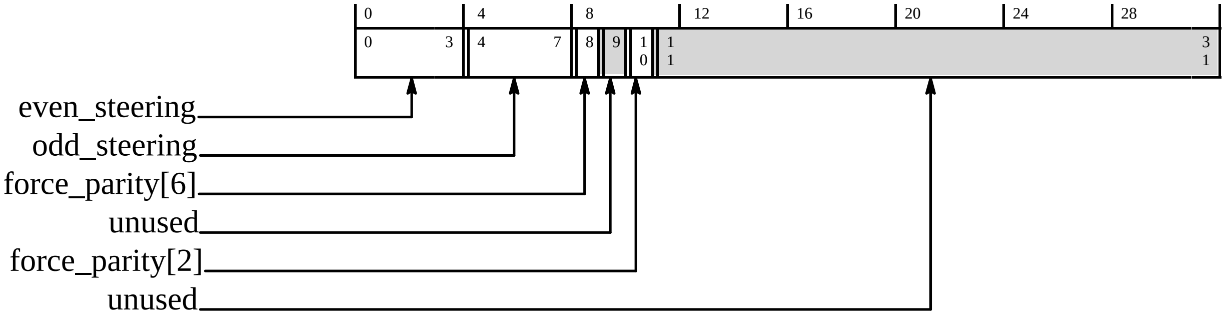
Field	Name	R/W	Description
0:9	unused	r/c	
10:29	tag	r/c	Contains diagnose or BIST signature data read from the group 1 tag
30	tag_parity	r/c	Contains the parity bit (even) associated with group 1 tag[0:19]
31	unused		

11.5.7 ICPFBDATAA — I/O Address 0xf###e240,0xf###e244



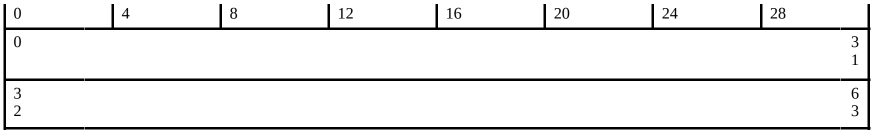
This contains a doubleword of diagnose or BIST write data for either doubleword 0 of group 0 or doubleword 1 of group 1.

11.5.8 ICPFBSPA — I/O Address 0xf###e248



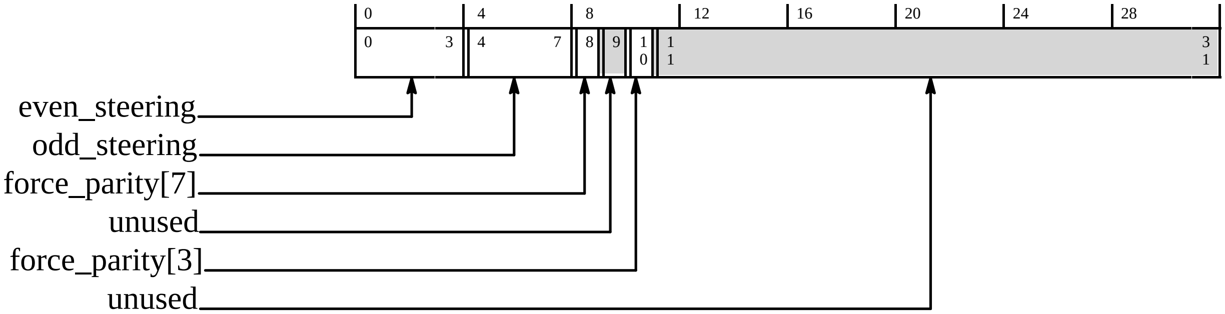
Field	Name	R/W	Description
0:3	even_steering	w	Contains the steering bits associated with ICPFBDATAA[0:31]
4:7	odd_steering	w	Contains the steering bits associated with ICPFBDATAA[32:63]
8	force_parity[6]	w	When set true, will force a parity error in the bottom parity bit associated with ICPFBDATAA and ICPFBDATAC
9	unused		
10	force_parity[2]	w	When set true, will force a parity error in the middle–top parity bit associated with ICPFBDATAA and ICPFBDATAC
11:31	unused		

11.5.9 ICPFBDATAB — I/O Address 0xf###e250,0xf###e254



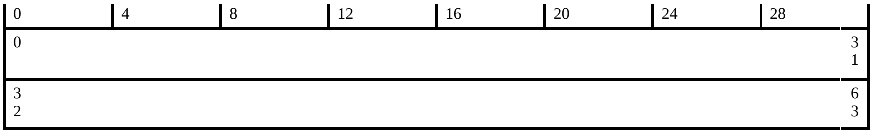
This contains a doubleword of diagnose or BIST write data for either doubleword 1 of group 0 or doubleword 0 of group 1.

11.5.10 ICPFBSPB — I/O Address 0xf###e258



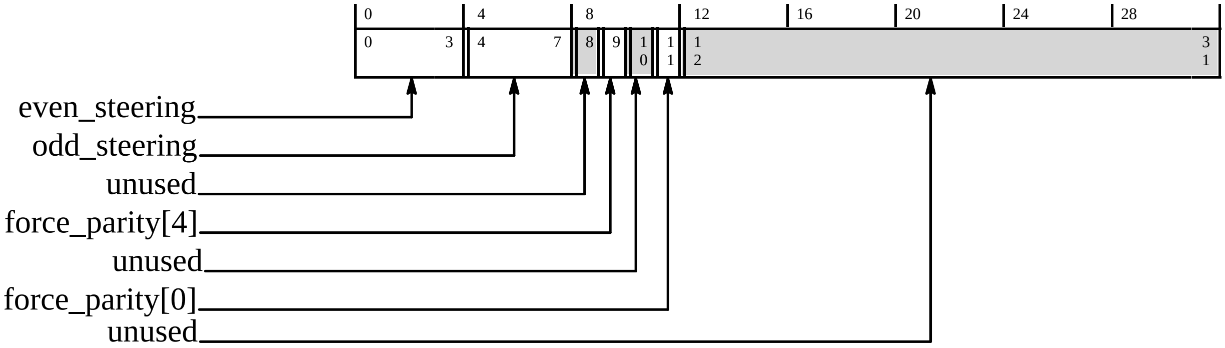
Field	Name	R/W	Description
0:3	even_steering	w	Contains the steering bits associated with ICPFBDATAB[0:31]
4:7	odd_steering	w	Contains the steering bits associated with ICPFBDATAB[32:63]
8	force_parity[7]	w	When set true, will force a parity error in the bottom parity bit associated with ICPFBDATAB and ICPFBDATAD
9	unused		
10	force_parity[3]	w	When set true, will force a parity error in the middle–top parity bit associated with ICPFBDATAB and ICPFBDATAD
11:31	unused		

11.5.11 ICPFBDATAC — I/O Address 0xf###e260,0xf###e264



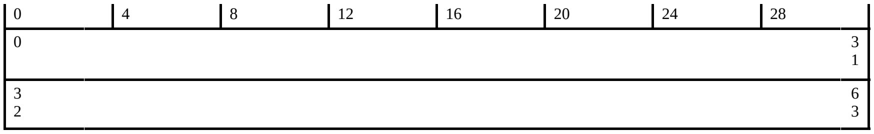
This contains a doubleword of diagnose or BIST write data for either doubleword 2 of group 0 or doubleword 3 of group 1.

11.5.12 ICPFBSPC — I/O Address 0xf###e268



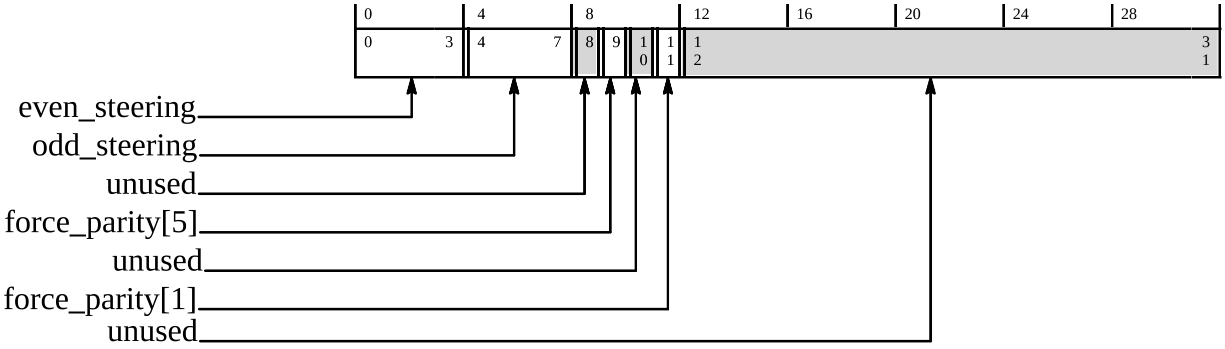
Field	Name	R/W	Description
0:3	even_steering	w	Contains the steering bits associated with ICPFBDATAC[0:31]
4:7	odd_steering	w	Contains the steering bits associated with ICPFBDATAC[32:63]
8	unused		
9	force_parity[4]	w	When set true, will force a parity error in the middle–bottom parity bit associated with ICPFBDATAA and ICPFBDATAC
10	unused		
11	force_parity[0]	w	When set true, will force a parity error in the top parity bit associated with ICPFBDATAA and ICPFBDATAC
12:31	unused		

11.5.13 ICPFBDATAD — I/O Address 0xf###e270,0xf###e274



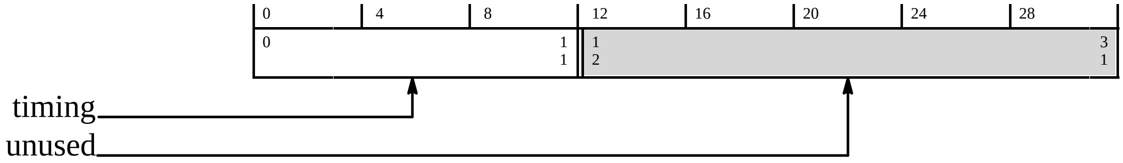
This contains a doubleword of diagnose or BIST write data for either doubleword 3 of group 0 or doubleword 2 of group 1.

11.5.14 ICPFBSPD — I/O Address 0xf###e278



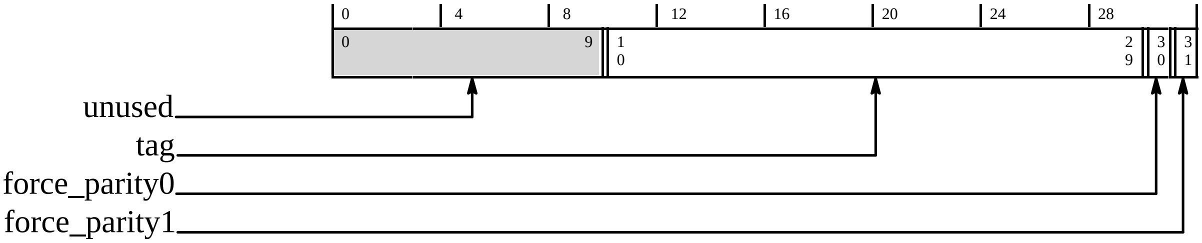
Field	Name	R/W	Description
0:3	even_steering	w	Contains the steering bits associated with ICPFBDATAD[0:31]
4:7	odd_steering	w	Contains the steering bits associated with ICPFBDATAD[32:63]
8	unused		
9	force_parity[5]	w	When set true, will force a parity error in the middle–bottom parity bit associated with ICPFBDATAB and ICPFBDATAD
10	unused		
11	force_parity[1]	w	When set true, will force a parity error in the top parity bit associated with ICPFBDATAB and ICPFBDATAD
12:31	unused		

11.5.15 IC_TIMING — I/O Address 0xf###e280



Field	Name	R/W	Description
0:11	timing	w	Frequency specific instruction cache timing configuration value. Resets to all zeros. This register will probably not be written to, but if it is, the value written will be empiracally determined from Phase II electrical characterization.
12:31	unused		

11.5.16 ICPFBTAG — I/O Address 0xf###e284



Field	Name	R/W	Description
0:9	unused	w	
10:29	tag	w	Specifies negative true diagnose or BIST write data for the tags
30	force_parity0	w	When set true will force a parity error in the group 0 tag parity bit
31	force_parity1	w	When set true will force a parity error in the group 1 tag parity bit

11.6 I/O Mapped MIOC Diagnose Registers

The PA7300LC MIOC responds as a GSC target for two pages of registers, plus the broadcast spaces:

- The Processor HPA contains the architected IO_EIR and IO_COMMAND registers, plus diagnose registers for the first-level data and instruction caches — see sections 11.4 and 11.5 for descriptions of those diagnose registers.
- The Memory Controller HPA contains various registers to configure, report status on, and handle errors in the MIOC.
- The Local and Global Broadcast spaces contain the architected IO_EIR, IO_COMMAND, and FLEXID registers.

Any of these registers can be accessed by either the CPU (via an I/O load or store), or by an external GSC master. There is no internal bypassing of CPU I/O cycles; if the CPU accesses a register, the cycle will appear on the GSC bus, with the MIOC acting as both master and slave.

The broadcast spaces are architecturally defined, and have fixed addresses. Local broadcast space is located at 0xfffc'0000–0xfffd'ffff, and global broadcast space is at 0xfffe'0000–0xffff'ffff. The processor and memory controller HPAs are always located in sequential pages, but they can be moved (together) by programming the FLEXID register at 0xfffc0020. The characters "###" in a register address represent the programmable part of the address; at power-up, these 12 address bits default to 0xffb. See the FLEXID register description for more information.

To adhere to the PA architecture definition, the MIOC will READY and run a technically proper GSC cycle for all transactions to the processor or memory controller HPAs or broadcast spaces, as long as the GSC address has valid parity and GSC TYPE[0]=0. But, for defined results, all PA7300LC registers *must* be accessed as word-sized entities. Since byte enables are ignored, partial-word writes will corrupt the register being written; any data after the first word of a 2-, 4-, or 8-word GSC cycle will be ignored; and reads from undefined registers will return undefined data. But otherwise, invalid register accesses will have no side-effects.

In the following descriptions, the "**CPU bits**" column identifies bit fields using the CPU's bit numbering system; while "**GSC bits**" refers to bit numbers as they will appear on the GSC I/O bus. The "**SW Access**" column shows the access that software has for each field: "W" for write-only; "R" for read-only; "RW" for read and write; or "RT" for read/toggle (such a bit toggles its state if a 1 is written to it). Reads from bits that do not have read access will return undefined data, but will have no side-effects. Writes to bits that do not have write access will be ignored and will have no side-effects. The "**HW Access**" column shows whether hardware conditions within the PA7300LC MIOC can change the state of the field: "W" indicates hardware can write that field; "I" indicates hardware will initialize the field once at power-up; "St" indicates hardware can set (but not clear) a bit; or "Cl" indicates hardware can clear (but not set) a bit. The "**Default**" column shows what value each field will contain after a chip reset; "?" indicates the field is not automatically initialized, and might come out of reset in any state.

Bits that have no specific purpose in life are labeled as either reserved, unused, or undefined:

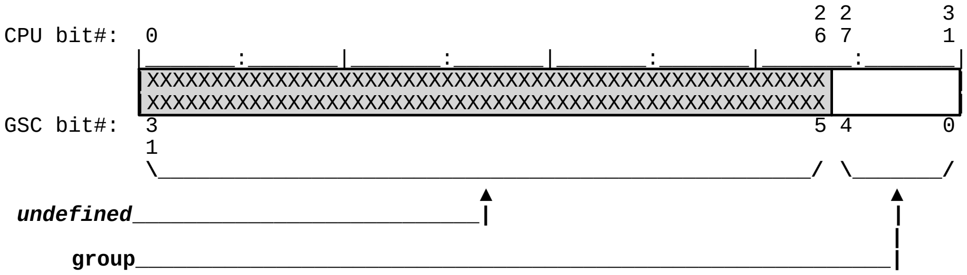
- An "**unused**" bit always reads as 0.
- An "**undefined**" bit may read as either 0 or 1.
- A "**reserved**" bit will read the last value written to it.

When writing to registers with unused or undefined fields, those fields should be set to 0 (to improve code clarity and consistency). When writing to registers with reserved bits, those bits should be initialized to 0, then should be left unchanged if selected fields of the register are later modified; this will simplify things if a reserved bit must be pressed into service in the future. Presently, however, a reserved bit does not affect the PA7300LC's operation.

Numerical conventions:

- "**0x**" preceding a number indicates hexadecimal;
- "**%**" preceding a number indicates binary;
- unannotated numbers are in decimal.

11.6.1 EIR — I/O Address 0xf###e000



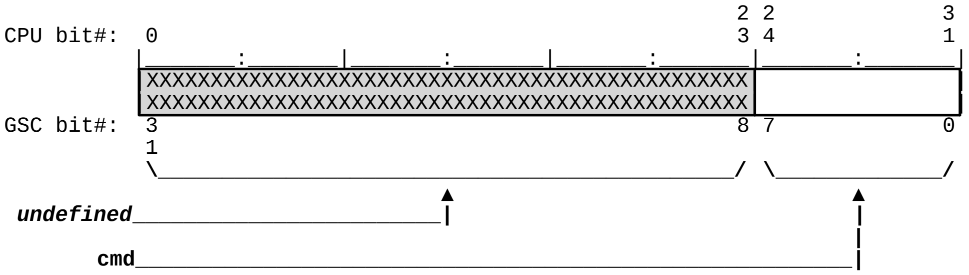
CPU bits	GSC bits	Name	Access SW	HW	Default	Description
0:26	31:5	*undefined*			?	
27:31	4:0	group	W		none	Indicates which bit to set in the CPU’s EIRR.

Writing to this architected register sends an external interrupt request to the CPU. Specifically, the write sets one bit in the CPU’s EIRR (External Interrupt Request Register) to 1. The 5-bit `group` field determines which of the 32 bits to set; for example, if `group=0x03`, bit #3 of the EIRR is set. If the indicated EIRR bit is already set, the write does not change the EIRR. Unused bits should be set to zero.

Reading from this register returns undefined data.

This register is functionally identical to both the EIR_LOCAL at 0xfffc0000 and the EIR_GLOBAL at 0xfffe0000.

11.6.2 COMMAND — I/O Address 0xf###e030



CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:23	31:8	*undefined*			?	
24:31	7:0	cmd	W		none	Selects an architected command (0x05=TOC).

Writing to this architected register, with `cmd=0x05`, sends a TOC (Transfer of Control) signal to the CPU. Writing any other value into the `cmd` field has no effect on the CPU, since the PA7300LC defines no other command codes. Unused bits should be set to zero.

Reading from this register returns undefined data.

See also the COMMAND_LOCAL register at 0xfffc0030, and COMMAND_GLOBAL at 0xfffe0030. Those registers are similar to this one, but they will completely reset the PA7300LC rather than generate a TOC.

11.6.3 MIOC_CONTROL — I/O Address 0xf###f080

```
                                   1 1 1 1 1 1 1 1 1 2 2                   3
  CPU bit#:  0 1 2 3 4 5           1 2 3 4 5 6 7 8 9 0 1                   1
            |_______:_______|_______:_______|_______:_______|_______:_______|
            | |   | | |0000000000000| | | | | | | | |-|000000000000000000000|
            | |   | | |0000000000000| | | | | | | | |-|000000000000000000000|
  GSC bit#:  3 3 2 2 2 2           2 1 1 1 1 1 1 1 1 1 1                   0
             1 0 9 8 7 6           0 9 8 7 6 5 4 3 2 1 0
             ▲ \_/ ▲ ▲ \___________/ ▲ ▲ ▲ ▲ ▲ ▲ ▲ ▲ ▲ \___________________/
      dwmode_|  ▲  | |        ▲      | | | | | | | | |           ▲
                |  | |        |      | | | | | | | | |           |
        drdw____|  | |        |      | | | | | | | | |           |
                   | |        |      | | | | | | | | |           |
    idrdcntl_______| |        |      | | | | | | | | |           |
                     |        |      | | | | | | | | |           |
        rpen_________|        |      | | | | | | | | |           |
unused________________________|      | | | | | | | | |           |
                                     | | | | | | | | |           |
   dma_noecc_________________________| | | | | | | | |           |
                                       | | | | | | | |           |
     lpmc_en___________________________| | | | | | | |           |
                                         | | | | | | |           |
 dma_nocache_____________________________| | | | | | |           |
                                           | | | | | |           |
 fast_memory_______________________________| | | | | |           |
                                             | | | | |           |
  lopowhilat_________________________________| | | | |           |
                                               | | | |           |
       pgape___________________________________| | | |           |
                                                 | | |           |
       pgdpe_____________________________________| | |           |
                                                   | |           |
        slen_______________________________________| |           |
reserved_____________________________________________|           |
unused___________________________________________________________|
```

CPU bits	GSC bits	Name	Access SW	 HW	Default	Description
0	31	dwmode	RW		0	Selects doubleword mode.
1:2	30:29	drdw	RW		0	Sets the delays for FET switch changeovers.
3	28	idrdcntl	RW		0	Controls polarity of DRDCNTL.
4	27	rpen	RW		0	Enables read promotion in the main MIOC queue.
5:11	26:20	*unused*			0	
12	19	dma_noecc	RW		0	Inhibits signalling double-bit errors during DMA.
13	18	lpmc_en	RW		0	Enables detection of DMA memory limit errors.
14	17	dma_nocache	RW		0	Inhibits all DMA data buffering.
15	16	fast_memory	RW		0	Indicates memory returns data fast enough for GSC.
16	15	lopowhilat	RW		0	Inhibits speculative issuing of DIO addresses.
17	14	pgape	RW	Cl	0	Generates even parity on next DIO address.
18	13	pgdpe	RW	Cl	0	Generates/checks even parity on next DIO data.
19	12	slen	RW		0	Enables LPMC for detected single-bit mem errors.
20	11	*reserved*	RW		0	
21:31	10:0	*unused*			0	

This register controls many of the overall and general capabilities of the MIOC. Other registers in the MIOC are dedicated to the detailed control and configuration of the memory system, second level cache, and GSC bus. The defined bits in this register are all readable, to simplify software that needs to determine the system configuration or to change just one field.

dwmode selects double-word (128-bit) mode when equal to one. It applies to both the second level cache and the memory controller, and affects the second level cache address and tag bits as follows:

dwmode	SLA[13]	SLATV_13	SLT[13]
0	rpn[28]	1	rpn[13]
1	rpn[13]	rpn[13]	1

drdw sets the delays for FET switch changeovers.

idrdcntl controls the polarity of DRDCNTL. If `idrdcntl=0` then DRDCNTL will equal 1 when the controller desires the FET switch to be closed.

rpen enables read promotion in the main MIOC queue. Only uncached memory reads and copyins will be promoted, and then only in front of copyouts.

dma_noecc inhibits the GSC controller from driving bad parity onto the GSC bus when data read from system memory by a DMA device has an uncorrectable (double-bit) error. Normally, the PA7300LC will flag a potentially bad word of data, so that the DMA device can know of the problem before it uses the faulty data. However, if `dma_noecc`=1, the PA7300LC will always drive correct parity during DMA reads, even for questionable data.

lpmc_en enables DMA memory limit errors (DMLs) to be detected and logged, and to generate LPMCs. If `lpmc_en=0`, the PA7300LC itself makes no record of a DMA access to an invalid memory location, and instead relies on the mastering device to log the error. Regardless of the status of this bit, the PA7300LC will not respond to a DMA access to an invalid address, and so that GSC transaction will time out.

dma_nocache should be set to 0 in all systems. If set to 1, `dma_nocache` inhibits the PA7300LC's normal inbound and outbound DMA data buffering, delivering all GSC cycles directly to the memory system without the merging and coalescing and prefetching that normally take place for efficiency.

fast_memory should be set to 1 for all currently-supported system configurations. This indicates that the read data throughput from the memory system is at least as fast as GSC's data bandwidth.

lopowhilat slightly reduces chip power, by not speculatively driving I/O addresses from the CPU onto an internal GSC address bus. The tradeoff is that, if `lopowhilat=1`, then starting a DIO transaction from the CPU may be delayed by one GSC state.

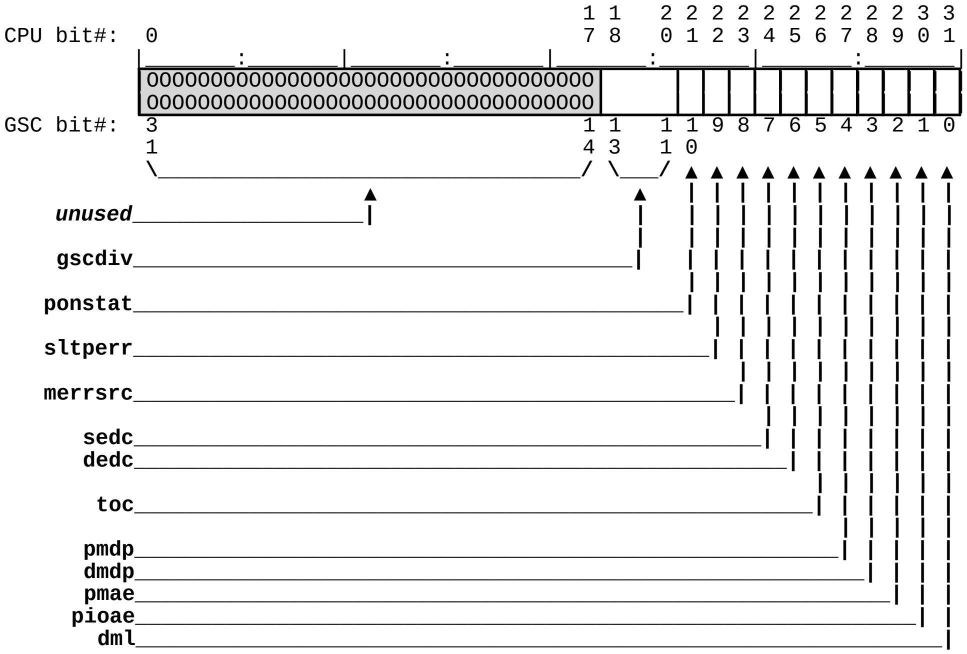
pgape and **pgdpe** are intended to test the GSC parity hardware. Setting `pgape` to 1 forces the PA7300LC to generate even (bad) parity during the address phase of the next processor-mastered I/O cycle. Setting `pgdpe` to 1 causes the PA7300LC to generate even (bad) parity during the first data phase of the next processor I/O write, or to check for even parity during the first data phase of the next processor I/O read. These bits are automatically cleared after they affect one address or one word of GSC data.

A processor sync inctruction must be executed between writing to `pgape` or `pgdpe` and the instruction for which they are intended to affect. The sync instruction must execute from memory space to avoid application of `pgape` or `pgdpe` to the instruction fetch.

The `pgape` and `pgdpe` bits are marked read/writable; but note that if the processor reads the MIOC_CONTROL register when either of these bits is set, that read will itself be affected, and these two bits should be considered undefined.

slen enables the MIOC to send an LPMC to the processor when it detects a single-bit memory error (SEDC). Note that single-bit errors will be logged in the MIOC status registers, regardless of the status of this bit.

11.6.4 MIOC_STATUS — I/O Address 0xf###f084


CPU bit#: 0 17 18 20 21 22 23 24 25 26 27 28 29 30 31
GSC bit#: 31 14 13 11 10 9 8 7 6 5 4 3 2 1 0
unused
gscdiv
ponstat
sltperr
merrsrc
sedc
dedc
toc
pmdp
dmdp
pmae
pioae
dml

CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:17	31:14	*unused*			0	
18:20	13:11	gscdiv	R	I	below	Indicates the ratio between CPU and GSC clocks.
21	10	ponstat	R	I	below	Indicates what caused the latest chip reset.
22	9	sltperr	RT	St	0	Indicates second level cache tag parity error.
23	8	merrsrc	RT	W	?	Distinguishes memory from 2nd level cache errors.
24	7	sedc	RT	St	0	Indicates single bit memory error.
25	6	dedc	RT	St	0	Indicates double bit memory error.
26	5	toc	RT	St	0	Indicates TOC has been generated.
27	4	pmdp	RT	St	0	Indicates processor mastered data parity error.
28	3	dmdp	RT	St	0	Indicates DMA mastered data parity error.
29	2	pmae	RT	St	0	Indicates processor memory address error.
30	1	pioae	RT	St	0	Indicates processor I/O address error.
31	0	dml	RT	St	0	Indicates DMA memory limit error.

This register contains general status information, and logs various error conditions within the MIOC. The gscdiv and ponstat fields are established during chip reset and then will not change; each remaining bit either can be set when the PA7300LC hardware detects an error condition, or can be toggled (changing a 0 into a 1, *or* changing a 1 into a 0) when software writes a 1 into its bit position.

`gscdiv` indicates how fast the GSC bus is clocked, relative to the PA7300LC's main clock. Two dedicated input pads on the PA7300LC, GSCR[0:1], establish the divide ratio during power-up reset; note that the system's Jarvic clock generator must be programmed to the same divide ratio. The following table shows the relationship between `gscdiv`, GSCR[0:1]'s status at power-up, the frequency of GSC's main SYNCH clock, and GSC's actual signalling rate (i.e. the frequency of a GSC device's internal GCLK, which is one-half of SYNCH's frequency):

	GSCR[0:1]	GSC SYNCH	imaginary GCLK
gscdiv	pads	frequency	frequency
%001	%00	CPU clock / 1.5	CPU clock / 3
%010	%01	CPU clock / 2	CPU clock / 4
%011	%10	CPU clock / 2.5	CPU clock / 5
%100	%11	CPU clock / 3	CPU clock / 6

All other `gscdiv` values are currently undefined (and will never be returned when reading this register).

`ponstat` indicates whether the most recent chip reset was caused by the system's power supply (if it equals 0), or by a software-initiated broadcast reset command (if it equals 1).

`sltperr` indicates that the second level cache detected a tag parity error.

`merrsrc` indicates whether the currently-logged memory error was due to a DRAM memory read (if it equals 0), or due to a second level cache read (if it equals 1). If no memory error is currently logged, this bit is undefined.

`sedc` indicates that the memory controller detected (and corrected) a single bit memory error.

`dedc` indicates that the memory controller detected a multiple bit memory error.

`toc` indicates that the MIOC has signalled a transfer-of-control to the processor. This bit can be set either when software initiates a TOC by writing to the COMMAND register in the processor's HPA page; or when hardware initiates a TOC by pulling the PA7300LC's TOCL pin active (low).

`pmdp` indicates that the GSC controller detected a processor-mastered data parity error. This bit is set when bad data parity is detected on the GSC bus during a transaction mastered by the PA7300LC.

`dmdp` indicates that the GSC controller detected a DMA-mastered data parity error. This bit is set when bad data parity is detected on the GSC bus during a transaction in which the PA7300LC is the slave.

`pmae` indicates that the memory controller detected a processor memory address error. This bit is set when the processor issues a transaction into memory space, but where no memory is configured.

`pioae` indicates that the GSC controller detected a processor I/O address error. This bit is set when the processor issues a transaction into I/O space that times out. This could result from addressing an undefined I/O location, or from an address parity error on the GSC bus.

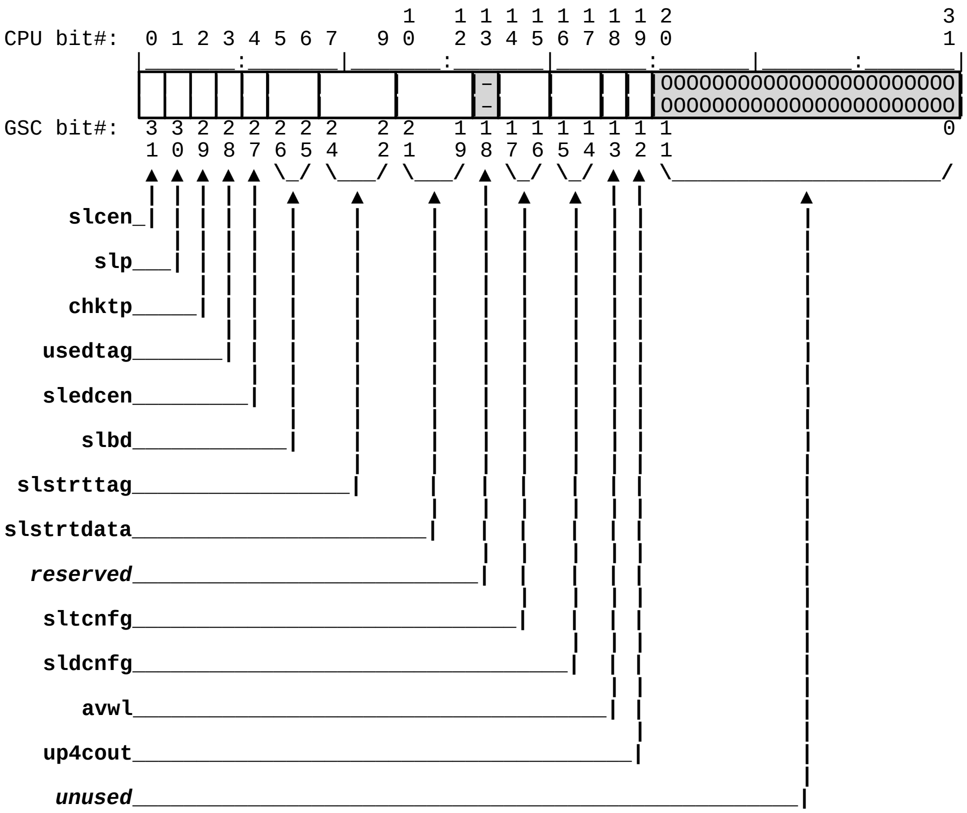
`dml` indicates that the GSC controller detected a DMA memory limit error. This bit is set when a GSC device issues a transaction into unconfigured memory space, *and* the `lpmc_en` bit in the MIOC_CONTROL register is set.

The MIOC generates `toc`, `hpmc`, and `lpmc` signals and sends them to the CPU. They are statically decoded from the above status bits as follows:

```
 toc = toc
hpmc = sltperr or dedc or pmdp or dmdp or pmae or pioae
lpmc = dml or (slen and sedc)
```

Multiple logged errors may exist simultaneously. When conflicts occur, the least recent, most severe error will be logged. For example, `sedc`, `dedc`, and `pmae` use the same error address register. `sedc` is considered lower severity than `dedc` or `pmae`; so a `dedc` or `pmae` will overwrite a previously-logged `sedc`, but a newly-detected `sedc` will not overwrite `sedc`, `dedc`, or `pmae`. Nothing overwrites previously-detected `dedc`s or `pmae`s. Also, `dmdp` and `dml` use the same error address register; but `dml` is lower-severity than `dmdp`, so a newly-detected `dml` will not overwrite a previous `dmdp`.

11.6.5 SLTCV — I/O Address 0xf###f0a0



CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0	31	slcen	RW		0	Enables second level cache.
1	30	slp	RW		0	Enables second level cache low power mode.
2	29	chktp	RW		0	Enables second level cache tag parity checking.
3	28	usedtag	RW		0	Enables use of diagnose tag from DIAGTAG.
4	27	sledcen	RW		0	Enables second level cache data error det/corr.
5:6	26:25	slbd	RW		0	Selects second level cache bus divisor.
7:9	24:22	slstrttag	RW		0	Sets sample cycle for second level cache tag.
10:12	21:19	slstrtdata	RW		0	Sets sample cycle for second level cache data.
13	18	*reserved*	RW		0	
14:15	17:16	sltcnfg	RW		0	Selects type of DRAM for second level cache tag.
16:17	15:14	sldcnfg	RW		0	Selects type of DRAM for second level cache data.
18	13	avwl	RW		0	Adds extra cycle to Address Valid Write Low delay.
19	12	up4cout	RW		0	Updates second level cache on CPU copyouts.
20:31	11:0	*unused*			0	

`slcen` enables the second level cache, and turns on the second level cache control outputs.

`slp` enables low power mode for the second level cache. This means that the address bus will transition no more often than the number of states specified by `slbd`.

`chktp` enables parity checking for second level cache tag reads.

`usedtag` causes the second level cache controller to source the value in the DIAGTAG register (at 0xf###f0a8) for tag compares and tag writes.

`sledcen` enables error detection/correction for data side second level cache reads.

`slbd` selects the second level cache bus divisor or `basic_cycle`, according to the following table:

slbd	CPU states/basic_cycle
%00	2
%01	3
%10	4
%11	4

`slbd` values %10 and %11 behave similarly, differing only in when the second level cache clock is asserted. The clock will be asserted 2 states after the address is asserted if `slbd`=%10; or 3 states after the address if `slbd`=%11.

`slstrttag` sets the number of states after the start of a transaction when the tag will be sampled. Nominally, the tag will be sampled on the CK2 following a (`basic_cycle` + `slstrttag`) delay from the start of a transaction. Note that this will be several states after the initial address cycle.

`slstrtdata` sets the number of states after the start of a transaction when the data will be initially sampled. Once data sampling starts, it will continue every `basic_cycle` states until the entire line is read. Nominally, the first data will be sampled on the CK2 following a (`basic_cycle` + `slstrtdata`) delay from the start of a transaction. Note that this will be several states after the initial address cycle.

`sltcnfg` selects the type of SRAM used for second level cache tags, as follows:

sltcnfg	SLC tag SRAM type
%00	async
%01	*undefined*
%10	flow_thru
%11	reg_to_reg

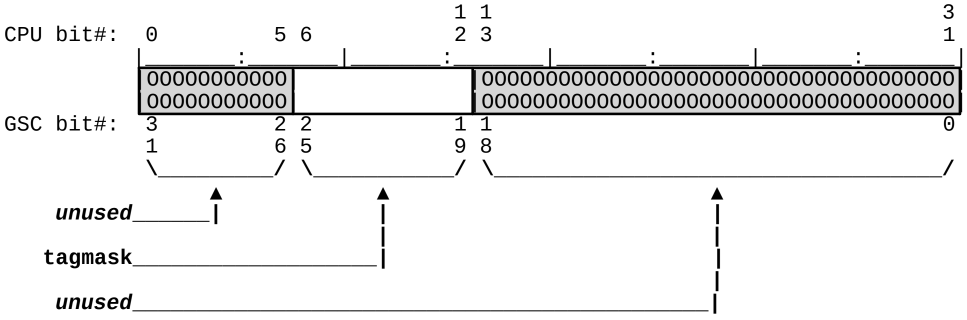
`sldcnfg` selects the type of SRAM used for second level cache data, as follows:

sldcnfg	SLC data SRAM type
%00	async
%01	*undefined*
%10	flow_thru
%11	reg_to_reg

`avwl` adds an extra state to the Address Valid Write Low delay for second level cache data writes.

`up4cout` controls the second level cache's behavior on CPU castouts. If `up4cout`=0, the cache will be interrogated, and the line will be marked invalid on a hit; if `up4cout`=1, the second level cache will update on CPU castouts. Cast out data is always posted to memory.

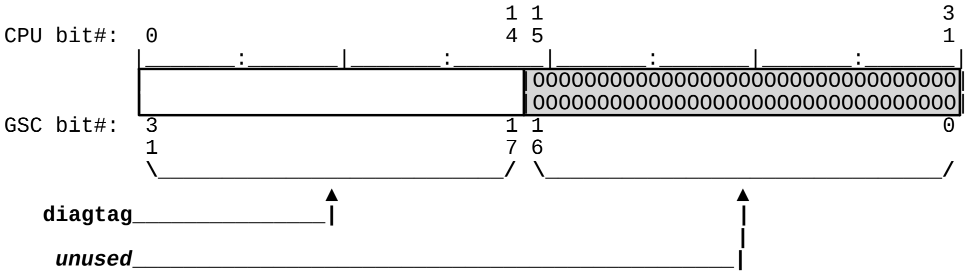
11.6.6 TAGMASK — I/O Address 0xf###f0a4



CPU bits	GSC bits	Name	Access SW	 HW	Default	Description
0:5	31:26	*unused*			0	
6:12	25:19	tagmask	RW		?	Contains the tag mask for the second level cache.
13:31	18:0	*unused*			0	

This register contains the tagmask for the second level cache. Each bit in the defined `tagmask` field defines whether the corresponding real address bit is part of the second level cache tag, or the second level cache index. For any real address bit whose `tagmask` bit=%1, the corresponding bit on the tag bus will be driven to 1 for tag updates and compares, unless `usedtag`=1 in which case the diagnose tag will be used. For any real address bit whose `tagmask` bit=%0, the tag for compares and updates will be passed through, and the corresponding bit on the SLA bus will be forced high. `usedtag` does not affect the SLA bus. `dwmode` also affects how second level cache tags and addresses are generated. See the definitions of `dwmode` in the MIOC_CONTROL register, and `usedtag` in the SLTCV register for further details.

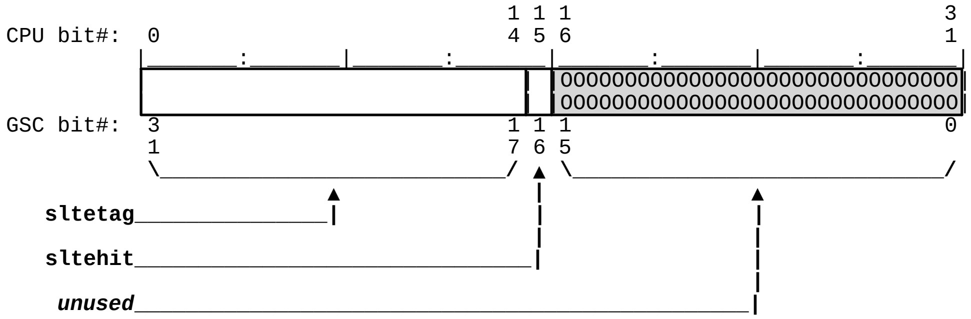
11.6.7 DIAGTAG — I/O Address 0xf###f0a8



CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:14	31:17	diagtag	RW		?	Second level cache tag for invalidates/`usedtag`.
15:31	16:0	*unused*			0	

This register contains the second level cache tag that is used for invalidates, *and* when `usedtag` in the SLTCV register is set. Since this register contains the invalidate pattern, its upper 4 bits (i.e. DIAGTAG[0:3], using CPU bit numbering) must equal 0xf when normal second level cache operation is desired.

11.6.8 SLTESTAT — I/O Address 0xf###f0ac



CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:14	31:17	sltetag	RW	W	?	Contains tag from last logged tag read.
15	16	sltehit	RW	W	?	Distinguishes hit/miss for last logged tag read.
16:31	15:0	*unused*			0	

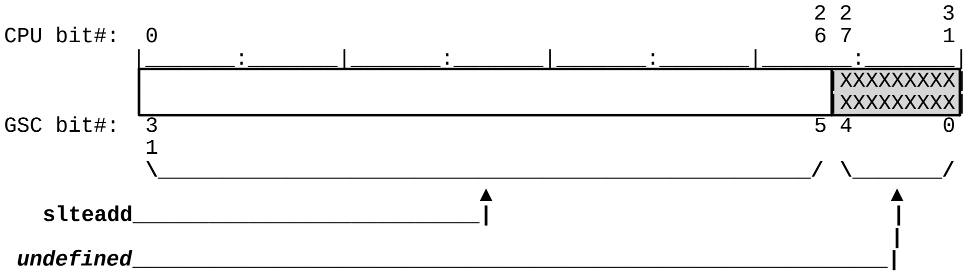
This register contains the tag, and the hit or miss indication, for the most recently logged tag read. If chktp=0 (in the SLTCV register), then *all* tag accesses will be logged in this register. If chktp=1 and sltperr=0 (in the MIOC_STATUS register), then tag accesses with parity errors will be logged — in other words, if chktp=1, this register will log the least recent tag parity error since sltperr was cleared.

sltetag contains the tag value from the most recent tag read (if chktp=0), or from the least recent tag read which had a parity error (if chktp=1).

sltehit contains the hit or miss indication from the most recent tag read (if chktp=0), or from the least recent tag read which had a parity error (if chktp=1). sltehit=1 indicates a hit; sltehit=0 indicates a miss.

See also the SLTEADD register, which logs the corresponding real address.

11.6.9 SLTEADD — I/O Address 0xf###f0b0



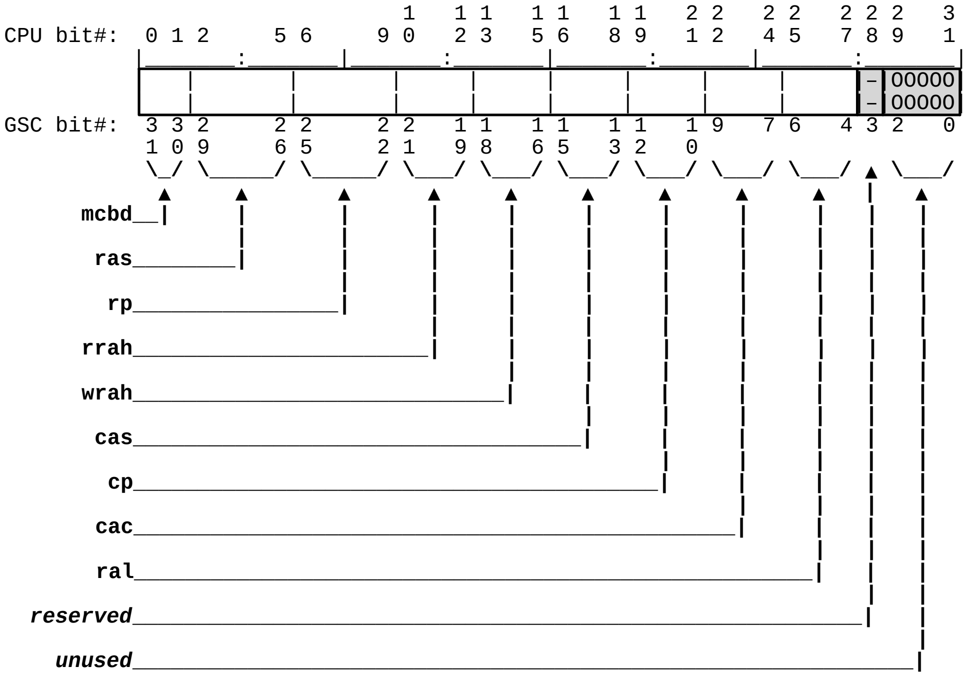
CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:26	31:5	slteadd	RW	W	?	Contains real address from last logged tag read.
27:31	4:0	*undefined*			?	

This register contains the real address for the most recently logged tag read. If `chktp`=0 (in the SLTCV register), then *all* tag accesses will be logged in this register. If `chktp`=1 and `sltperr`=0 (in the MIOC_STATUS register), then tag accesses with parity errors will be logged — in other words, if `chktp`=1, this register will log the least recent tag parity error since `sltperr` was cleared.

`slteadd` contains the real address from the most recent tag read (if `chktp`=0), or from the least recent tag read which had a parity error (if `chktp`=1).

See also the SLTESTAT register, which logs the corresponding tag and hit/miss indication.

11.6.10 MTCV — I/O Address 0xf###f0c0



CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:1	31:30	mcbd	RW		0	Sets memory controller bus divisor.
2:5	29:26	ras	RW		0	Sets minimum assertion time of ROW[0:3].
6:9	25:22	rp	RW		0	Sets minimum deassertion time of ROW[0:3].
10:12	21:19	rrah	RW		0	Sets address hold time for read cycles.
13:15	18:16	wrah	RW		0	Sets address hold time for write cycles.
16:18	15:13	cas	RW		0	Sets minimum assertion time of COL[0:3].
19:21	12:10	cp	RW		0	Sets minimum deassertion time of COL[0:3].
22:24	9:7	cac	RW		0	Sets data sample time after assertion of COL[0:3].
25:27	6:4	ral	RW		0	Sets minimum delay from assertion of COL[0:3] to deassertion of ROW[0:3].
28	3	*reserved*	RW		0	
29:31	2:0	*unused*			0	

mcbd is the memory controller’s “bus divisor”. This field generally controls the number of states allowed for setup and hold times, but has other impacts as well. It is normally a function of the system frequency.

ras controls the minimum assertion time of ROW[0:3]. This is normally only a factor during refresh cycles.

rp controls the minimum deassertion time of ROW[0:3].

`rrah` sets the row address hold time for read cycles.

`wrah` sets the row address hold time for write cycles.

`cas` controls the minimum assertion time of COL[0:3].

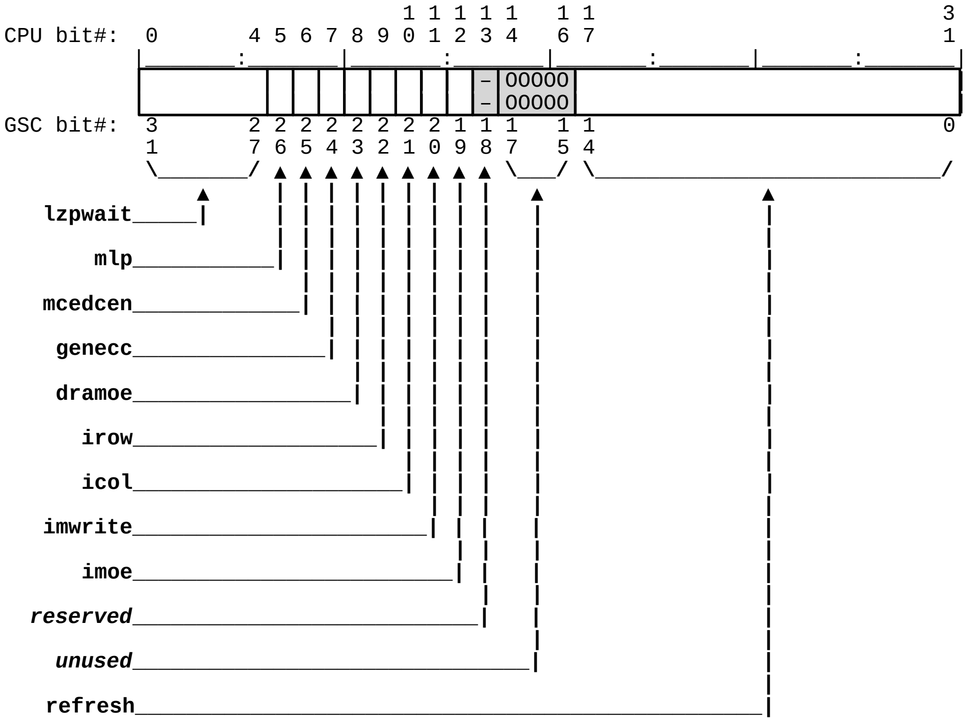
`cp` controls the minimum deassertion time of COL[0:3].

`cac` defines how many states after the assertion of COL[0:3] data will be sampled.

`ral` defines the minimum delay from the assertion of COL[0:3] to the deassertion of ROW[0:3].

Warning: DRAM timing is more complicated than indicated above. The above descriptions are only the first-order affects.

11.6.11 REF — I/O Address 0xf###f0cc


CPU bit#: 0 4 5 6 7 8 9 10 11 12 13 14 16 17 31
GSC bit#: 31 27 26 25 24 23 22 21 20 19 18 17 15 14 0
- 00000
- 00000
lzpwait
mlp
mcedcen
genecc
dramoe
irow
icol
imwrite
imoe
reserved
unused
refresh

CPU bits	GSC bits	Name	Access SW	HW	Default	Description
0:4	31:27	lzpwait	RW		0	Sets # of cycles to hold an idle active page open.
5	26	mlp	RW		0	Enables memory controller low power mode.
6	25	mcedcen	RW		0	Enables memory error detection/correction.
7	24	genecc	RW		0	Enables generation of check bits.
8	23	dramoe	RW		0	Output enable for DRAM signals.
9	22	irow	RW		0	Sets sense of ROW[0:3] to be negative true.
10	21	icol	RW		0	Sets sense of COL[0:3] to be negative true.
11	20	imwrite	RW		0	Sets sense of MWRITE[0:1] to be negative true.
12	19	imoe	RW		0	Sets sense of MOE[0:1] to be negative true.
13	18	*reserved*	RW		0	
14:16	17:15	*unused*			0	
17:31	14:0	refresh	RW		0	Sets DRAM refresh frequency.

lzpwait sets the number of states the memory controller will hang with an active page when idle. If lzpwait=0x1f, the memory controller will hang in the current page indefinitely, until refresh, or until another access causes a page switch.

mlp causes the memory controller to avoid changing the DRAM address bus unless necessary.

mcedcen enables error detection and correction for accesses to memory.

genecc controls whether the memory controller will generate proper error control bits (if genecc=1), or instead use the bytes from the MEM_WRITE_CHK_DATA register (if genecc=0).

dramoe is the output enable control for DRA[0:13], DRA13LO, ROW[0:3], COL[0:3], MWRITE[0:1], and MOE[0:1]. The output enable control for DRDCNTL equals (dramoe or slcen). See also the SLTCV register at 0xf###f0a0.

irow controls whether the DRAM ROW[0:3] lines are negative true (if irow=1) or positive true (if irow=0).

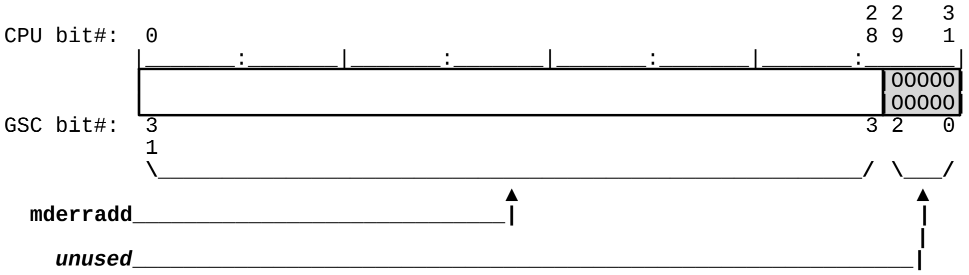
icol controls whether the DRAM COL[0:3] lines are negative true (if icol=1) or positive true (if icol=0).

imwrite controls whether the DRAM MWRITE[0:1] lines are negative true (if imwrite=1) or positive true (if imwrite=0).

imoe controls whether the DRAM MOE[0:1] lines are negative true (if imoe=1) or positive true (if imoe=0).

refresh sets the refresh requency for the DRAM. The ones' complement of this value, plus 2, equals the refresh period (in CPU clock cycles). In other words, to get a refresh frequency of R with a system clock frequency of S, set refresh=ones_complement((S/R) – 2). This field powers up to the slowest possible refresh rate; but refresh can never be completely disabled. The actual refresh interval counter is loaded from the refresh field automatically when the REF register is written, and then whenever a refresh pulse is generated.

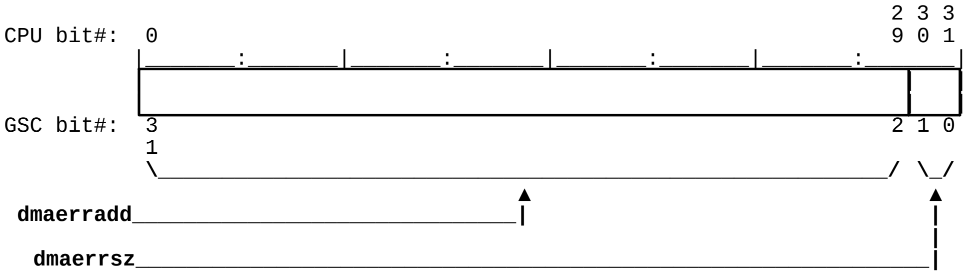
11.6.12 MDERRADD — I/O Address 0xf###f0e0



CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:28	31:3	mderradd	RW	W	?	Address of least recent, most severe memory error.
29:31	2:0	*unused*			0	

This register contains the doubleword address of the least recent, most severe read error from either memory or SLC.

11.6.13 DMAERR — I/O Address 0xf###f0e8



CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:29	31:2	dmaerradd	RW	W	?	Address of least recent, most severe DMA error.
30:31	1:0	dmaerrsz	RW	W	?	Size of least recent, most severe DMA error.

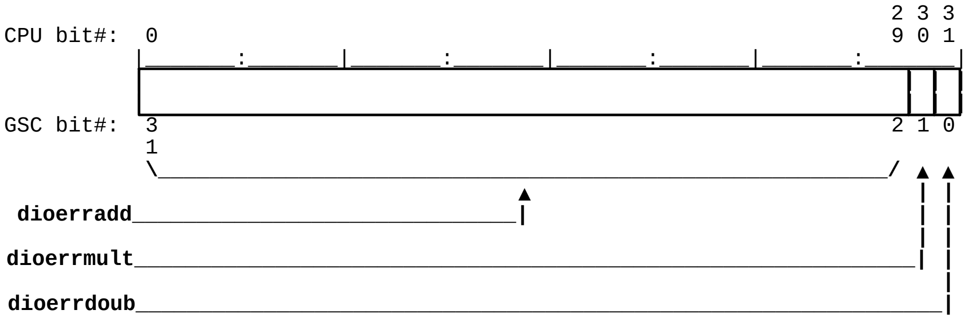
This register holds the address and size of the least recent, most severe DMA error (from a GSC transaction in which the PA7300LC was the slave). Such an error could be either a DMA data parity error, or a DMA memory limit error.

dmaerradd holds the word address of the offending DMA transfer.

dmaerrsz represents the length of the offending DMA transfer. This field is set from the TYPE[2:3] lines on the GSC bus, so it has the same encodings as TYPE[2:3]:

dmaerrsz	GSC transfer size
%00	one–word or sub–word
%01	two–word
%10	four–word
%11	eight–word

11.6.14 DIOERR — I/O Address 0xf###f0ec



CPU bits	GSC bits	Name	Access SW	 HW	Default	Description
0:29	31:2	dioerradd	RW	W	?	Address of least recent, most severe DIO error.
30	1	dioerrmult	RW	W	?	Indicates if DIO error occurred within a WRITEV.
31	0	dioerrdoub	RW	W	?	Indicates if DIO error was in a doubleword cycle.

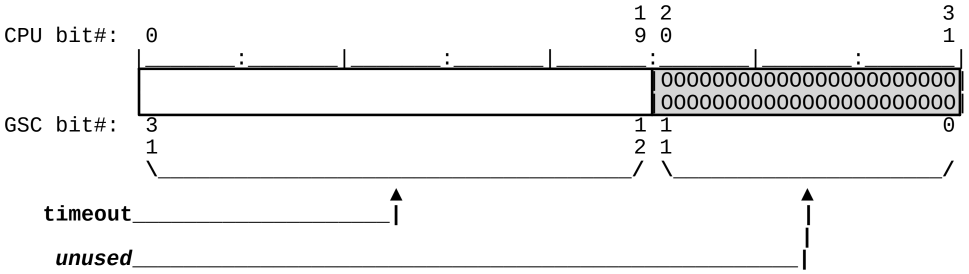
This register holds the address and size of the least recent, most severe DIO error (from a GSC transaction in which the PA7300LC was the master). Such an error could be either a timeout/address parity error, or a data parity error.

dioerradd holds the word address of the offending DIO transfer.

dioerrmult indicates if the offending DIO transfer was a WRITEV cycle, in which several writes from the CPU were possibly coalesced into one GSC transfer.

dioerrdoub indicates if the offending DIO transfer was issued as a doubleword load or store from the CPU.

11.6.15 GSC_TIMEOUT — I/O Address 0xf###f0f0



CPU bits	GSC bits	Name	Access SW	 HW	Default	Description
0:19	31:12	timeout	RW		0	Ones' complement of GSC timeout delay.
20:31	11:0	*unused*			0	

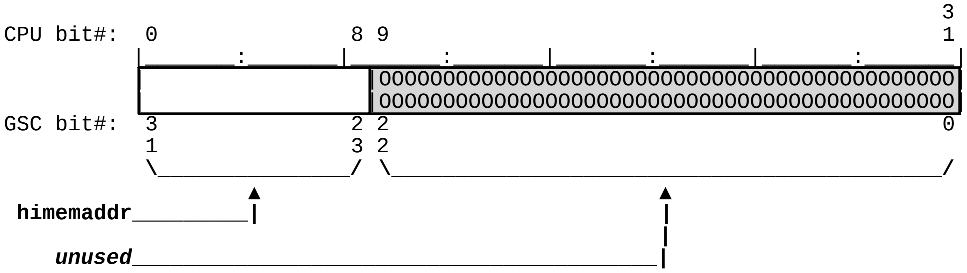
This register contains the ones' complement of the number of GSC cycles that must elapse before a GSC timeout error is signalled. Three timeout intervals are enforced: from ADDVL to READYL, from ADDVL to LSL assertion (starting a split), and from LSL negation (ending a split) to READYL. If any of these intervals exceedes the programmed timeout value, the PA7300LC will assert ERRORL two cycles later, to terminate the cycle with a timeout error.

The default value of `timeout`, zero, produces the longest timeout delay: (2^20–1) GSC cycles, or about 26ms on a 40MHz GSC bus. Some systems with bus bridges may need such long timeouts; but most systems should set the timeout delay to be much shorter, especially if the OS is to have any chance of recovering from I/O errors. Also, PDC must program a short timeout delay before doing anything (such as searching for I/O devices) that is *expected* to generate GSC addressing errors. GSC devices that might have longer latencies than that can seriously impact system performance; but in some cases, such as bus bridges, the possibility of longer latencies may be unavoidable. In such cases, the GSC device can reset the timeout counter by asserting LSL; or the timeout counter can be permanently disabled, by writing a value from 0xfff0 through 0xffff to the `timeout` field (i.e., attempting to set a timeout value less than 16 GSC cycles). Always use caution when attempting to override the timeout feature, as any problem on GSC might then hang the entire system.

In rare situations, such as if another GSC device needs to provide the GSC timeout function, it may be desirable to completely disable the PA7300LC's GSC timeout counter. It can be permanently disabled by writing a value from 0xffff0 through 0xfffff to the `timeout` field (i.e., attempting to set a timeout value less than 16 GSC cycles). Alternatively, a slave device can reset the timer during a particular cycle by asserting LSL. But defeating the timeout counter should be done only with extreme caution, as any problem on GSC might then hang the entire system.

Additional warnings: ensure that the timeout is always set longer than the latency to ROM, or else the boot process will hang. Also, GSC is not well-behaved (and may experience drive fights) if a device readies a transaction at nearly the same time as a timeout occurs; to avoid this problem, ensure that the timeout is set longer than the maximum expected latency for the slowest enabled device on the bus.

11.6.16 HIDMAMEM — I/O Address 0xf###f0f4



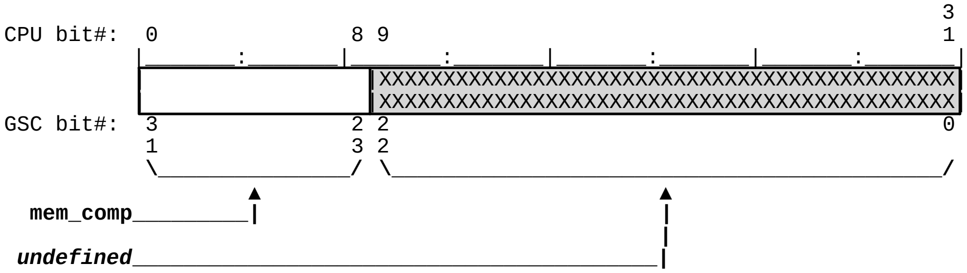
CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:8	31:23	himemaddr	RW		0	Holds 1 + address of top of installed memory.
9:31	22:0	*unused*			0	

This register tells the GSC subsystem how much DRAM memory is installed in the system. The GSC controller expects DRAM to exist at addresses 0x0000’0000 through HIDMAMEM–1, and will pass transactions within that range to the memory controller. It believes no DRAM exists at addresses HIDMAMEM through 0xefff’ffff, and will report a DMA memory address limit error for any GSC transactions within that range.

For example, if a system holds 16MB of DRAM, at addresses 0x0000’0000 through 0x00ff’ffff, HIDMAMEM should be set to 0x0100’0000.

Note that this register defaults to zero at power-up, which inhibits *all* GSC accesses to system memory.

11.6.17 MEMCOMP[0:15] — I/O Addresses 0xf###f100 thru 0xf###f13c



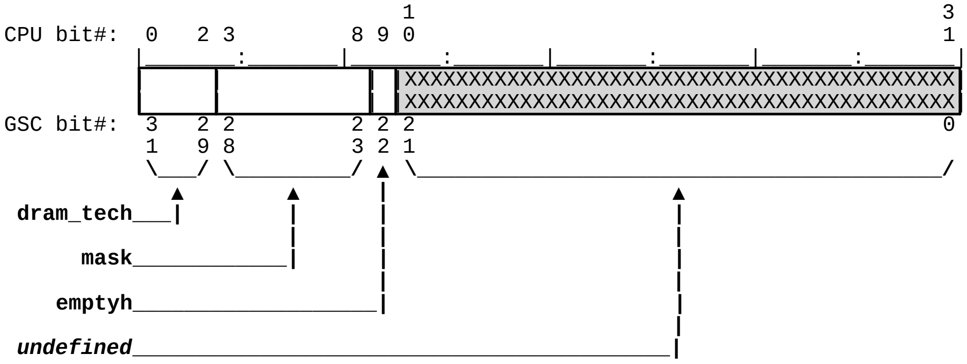
CPU bits	GSC bits	Name	Access SW	HW	Default	Description
0:8	31:23	mem_comp	W		?	Programs one of 16 memory address comparators.
9:31	22:0	*undefined*			?	

Each of these 16 registers programs one of the 16 memory block's memory address comparator. Memory block *x* will be selected when (memory_address[0:8] = MEMCOMP[*x*][0:8]), in combination with the masking specified in the MEMMASK[*x*] register. Refer to the description of MEMMASK[0:15], at addresses 0xf###f140–0xf###f17c.

To avoid conflict with the I/O address space, `mem_comp[0:3]` must *not* equal 0xf.

Writes to these registers should be followed by a sync instruction.

11.6.18 MEMMASK[0:15] — I/O Addresses 0xf###f140 thru 0xf###f17c



CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:2	31:29	dram_tech	W		?	Selects a memory block’s row/column mux scheme.
3:8	28:23	mask	W		?	Selects a memory block’s address compare mask.
9	22	emptyh	W		1	Disables a memory block’s address comparator.
10:31	21:0	*undefined*			?	

Each of these 16 registers is used in conjunction with its corresponding MEMCOMP register to program one of 16 memory address comparators. Refer to the description of MEMCOMP[0:15], at addresses 0xf###f100–0xf###f13c.

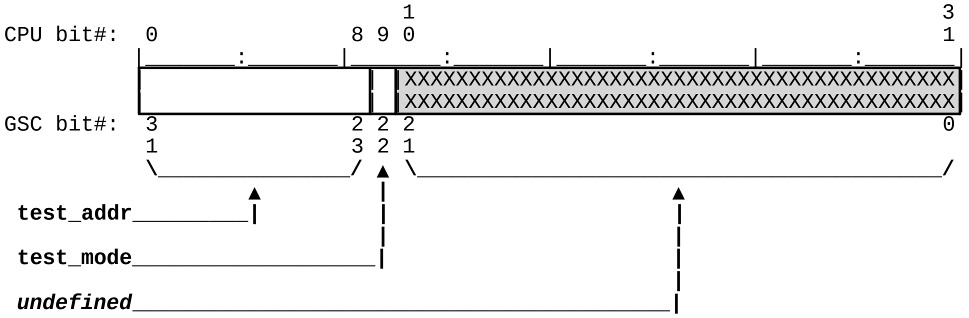
dram_tech selects the row and column address multiplexing scheme for the memory block. The proper value for this field depends on DRAM density and organization. Refer to the memory interface section for a detailed description.

mask selects how many bits of the address should be compared. The proper value for this field depends on the size of this comparator’s memory block. The six bits of this field correspond to bits [3:8] of the memory_address; a 0 in this field compares the corresponding bits of the memory_address and the MEMCOMP register; a 1 masks (inhibits) the comparison for that address bit. Bits [0:2] of the address cannot be masked and are always compared.

emptyh indicates whether this address comparator should be enabled. If `emptyh=0`, the corresponding comparator is enabled, and represents an installed block of DRAM; if `emptyh=1`, the comparator is disabled.

Writes to these registers should be followed by a sync instruction.

11.6.19 MEMTEST — I/O Address 0xf###f180



CPU bits	GSC bits	Name	Access SW	HW	Default	Description
0:8	31:23	test_addr	W		?	Sets address to force into address comparators.
9	22	test_mode	W		1	Disables memory address comparator test mode.
10:31	21:0	*undefined*			?	

This register is used in conjunction with the OUTCHK register, to test the memory address comparator array before it is programmed. MEMTEST is written to set an address to be tested; then OUTCHK is read to verify the correct decoding of that address. See also the description of OUTCHK, at address 0xf###f1c0.

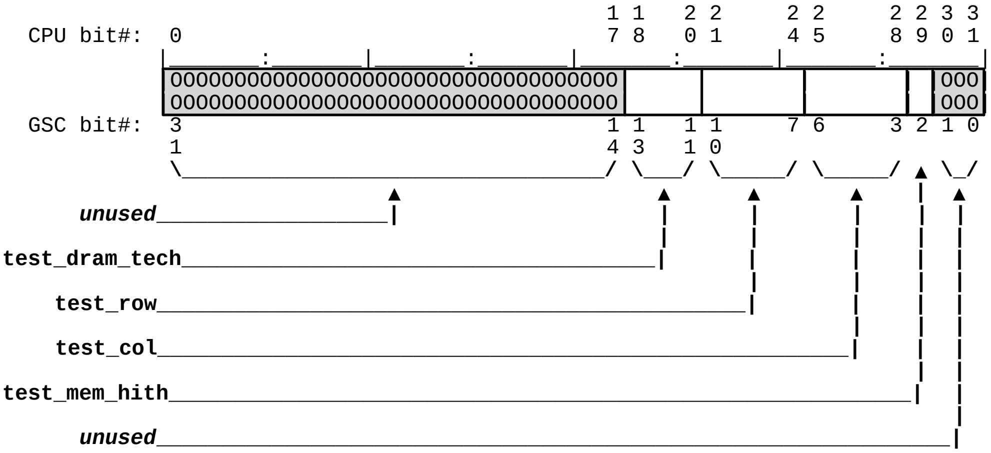
test_addr contains a 9-bit address that will be driven into the comparator array during test mode.

test_mode selects between normal operation and memory address comparator test mode. If test_mode=0, test mode is enabled, and test_addr will be driven to the address comparators, and the results of this operation can be read from the OUTCHK register. If test_mode=1, normal operation is enabled.

The comparator test should be performed with dramoe=0 (see the REF register at address 0xf###f0cc) to avoid unintentional assertion of the ROW[0:3] and COL[0:3] lines.

Writes to this register should be followed by a sync instruction.

11.6.20 OUTCHK — I/O Address 0xf###f1c0


CPU bit#: 0 17 18 20 21 24 25 28 29 30 31
GSC bit#: 31 14 13 11 10 7 6 3 2 1 0
unused
test_dram_tech
test_row
test_col
test_mem_hith
unused

CPU bits	GSC bits	Name	Access		Default	Description
			SW	HW		
0:17	31:14	*unused*			0	
18:20	13:11	test_dram_tech	R		?	Tests dram_tech from comparators.
21:24	10:7	test_row	R		?	Tests ROW[0:3] from comparators.
25:28	6:3	test_col	R		?	Tests COL[0:3] from comparators.
29	2	test_mem_hith	R		?	Tests internal hit signal from comparators.
30:31	1:0	*unused*			0	

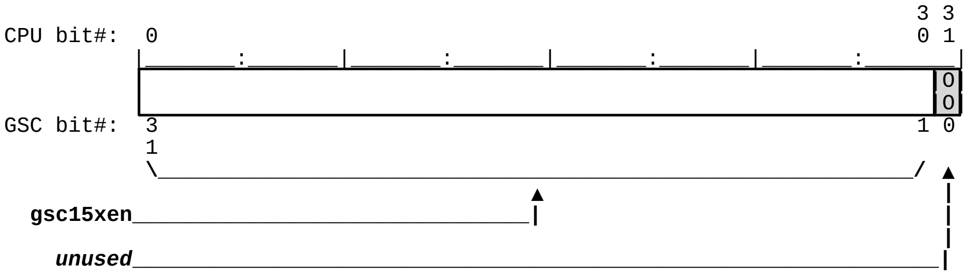
This register is used in conjunction with the MEMTEST register, to test the memory address comparator array before it is programmed. MEMTEST is written to set an address to be tested; then OUTCHK is read to verify the correct decoding of that address. See also the description of MEMTEST, at address 0xf###f180.

test_dram_tech contains the value programmed into dram_tech (in a MEMMASK register), from the comparator corresponding to the test_addr (in the MEMTEST register).

test_row, and **test_col** contain the values of ROW[0:3] and COL[0:3], driven from the comparator corresponding to the test_addr (in the MEMTEST register).

test_mem_hith is an internal signal that, when 1, indicates a hit on any comparator.

11.6.21 GSC15X_CONFIG — I/O Address 0xf###f7a0

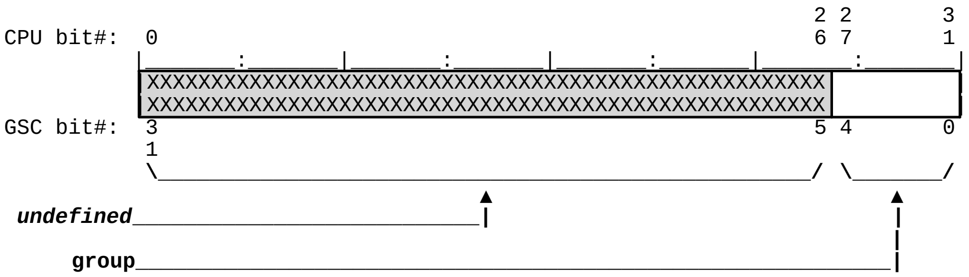


CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:30	31:1	gsc15xen	RW		0	Selects writev-capable blocks in I/O space.
31	0	*unused*			0	

This register selects which 8MB blocks of I/O space can accept “writev” type transactions from the processor (i.e., are decoded by a GSC 1.5X compatible slave device). CPU bit 0 corresponds to the lowest 8MB block in I/O space: 0xf000’0000 through 0xf07f’ffff; the remaining bits correspond to sequentially higher 8MB blocks. CPU bit 31, which *would* correspond to the highest 8MB block in I/O space (0xff80’0000 through 0xffff’ffff) is hard-wired to 0, since the GSC spec forbids any writev transactions to broadcast address space (which falls within that block). Software must ensure that *all* devices which reside within an 8MB block understand the writev transaction type, before setting the gsc15xen bit for that address block.

At power-up, all CPU single- and double-word stores to I/O space will be translated into GSC write1 and write2 transactions, respectively. But after any bits in this register are set to 1, CPU stores (single- *or* double-word) within those corresponding address blocks will be translated into GSC writev transactions, coalescing up to 8 sequential words into a single writev transaction.

11.6.22 EIR_LOCAL — I/O Address 0xfffc0000



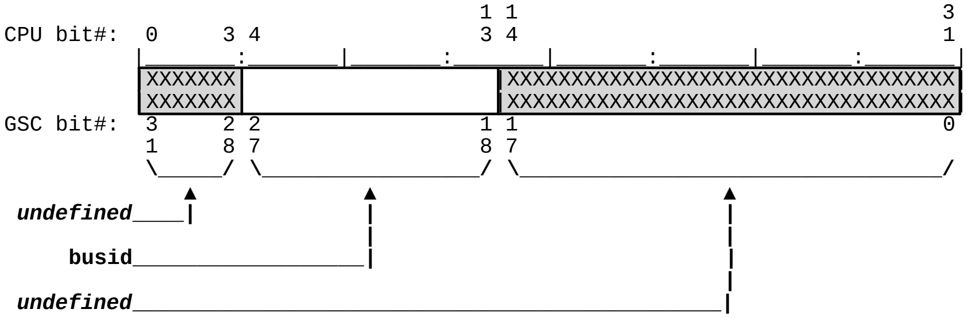
CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:26	31:5	*undefined*			?	
27:31	4:0	group	W		none	Indicates which bit to set in the CPU’s EIRR.

Writing to this architected register sends an external interrupt request to the CPU. Specifically, the write sets one bit in the CPU’s EIRR (External Interrupt Request Register) to 1. The 5-bit `group` field determines which of the 32 bits to set; for example, if `group=0x03`, bit #3 of the EIRR is set. If the indicated EIRR bit is already set, the write does not change the EIRR. Unused bits should be set to zero.

Since this register is in broadcast space, reading from it is illegal.

This register is functionally identical to both the EIR at 0xf###e000 and the EIR_GLOBAL at 0xfffe0000.

11.6.23 FLEXID — I/O Address 0xfffc0020



CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:3	31:28	*undefined*			?	
4:13	27:18	busid	W		0x3fe	Sets the GSC Bus_ID of the CPU and MIOC HPAs.
14:31	17:0	*undefined*			?	

Writing to this architected register sets the address range at which the PA7300LC will locate its CPU and MIOC HPA pages within I/O space. The PA7300LC will decode an access to its module register space if:

GSC address bits 31–28 = 0xf (which defines I/O space);
GSC address bits 27–18 = **busid** (PA7300LC's configured Bus_ID);
GSC address bits 17–14 = 0xf (PA7300LC's fixed Slot_ID of %1111); and
GSC address bit 13 = 1 (PA7300LC's fixed Submodule_IDs of %10 [for CPU] and %11 [for the MIOC]).

Throughout this document, the configurable part of register addresses is flagged with "###". The actual 12-bit hexadecimal value of "###" equals the 10-bit busid, concatenated with %11. For example, this table shows where the MIOC_CONTROL register will be decoded, for various values of busid:

```
busid     "###"    MIOC_CONTROL address
-----     -----    --------------------
0x000     0x003         0xf003f080
0x001     0x007         0xf007f080
0x002     0x00b         0xf00bf080
0x003     0x00f         0xf00ff080
0x004     0x013         0xf013f080
                 . . .
0x3fe     0xffb         0xfffbf080      (the default)
```

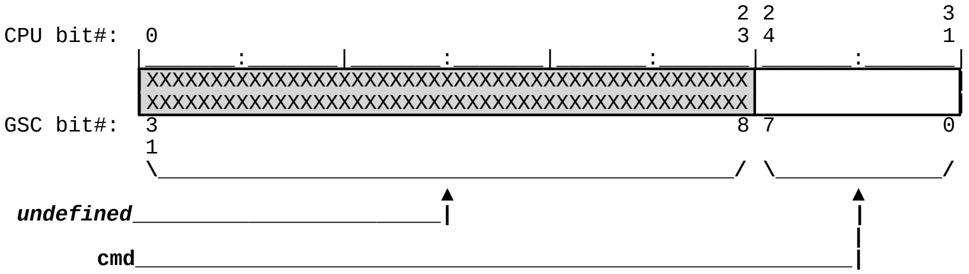
By default, the PA7300LC's CPU and MIOC HPAs are located in the same pages as the PA7100LC's. If FLEXID is written to move them, software must ensure that their new locations do not conflict with any other I/O devices. Note that busid must never be set to 0x3ff, as that would conflict with the broadcast registers.

Since the CPU must *always* be allowed to master GSC transactions, the PA7300LC does not implement a "bus master enable" bit as part of this register, as is mentioned in the GSC spec.

Note that FLEXID does not affect itself or any of the other broadcast registers — all broadcast register addresses are fixed.

Since this register is located in broadcast space, reading from it is illegal.

11.6.24 COMMAND_LOCAL — I/O Address 0xfffc0030



CPU bits	GSC bits	Name	Access		Default	Description
			SW	HW		
0:23	31:8	*undefined*			?	
24:31	7:0	cmd	W		none	Selects an architected command (0x05=reset).

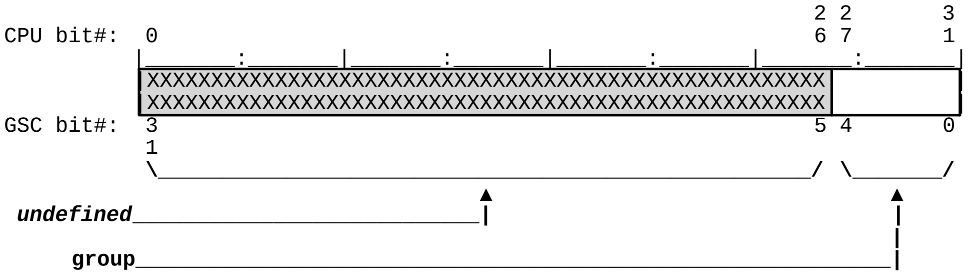
Writing to this architected register, with `cmd=0x05`, initiates a hard reset of the CPU and MIOC. Writing any other value into the `cmd` field has no effect, since the PA7300LC defines no other command codes. Unused bits should be set to zero.

Since this register is located in broadcast space, reading from it is illegal.

This register is functionally identical to the COMMAND_GLOBAL register at 0xfffe0030.

See also the COMMAND register at 0xf###e030. That register is similar to this one, but will generate a TOC rather than completely reset the PA7300LC.

11.6.25 EIR_GLOBAL — I/O Address 0xfffe0000



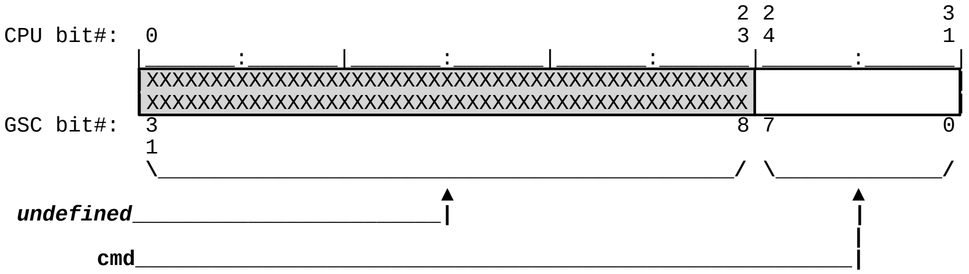
CPU bits	GSC bits	Name	Access SW	 HW	Default	Description
0:26	31:5	*undefined*			?	
27:31	4:0	group	W		none	Indicates which bit to set in the CPU’s EIRR.

Writing to this architected register sends an external interrupt request to the CPU. Specifically, the write sets one bit in the CPU’s EIRR (External Interrupt Request Register) to 1. The 5-bit `group` field determines which of the 32 bits to set; for example, if `group=0x03`, bit #3 of the EIRR is set. If the indicated EIRR bit is already set, the write does not change the EIRR. Unused bits should be set to zero.

Since this register is in broadcast space, reading from it is illegal.

This register is functionally identical to both the EIR at 0xf###e000 and the EIR_LOCAL at 0xfffc0000.

11.6.26 COMMAND_GLOBAL — I/O Address 0xfffe0030



CPU bits	GSC bits	Name	Access SW	Access HW	Default	Description
0:23	31:8	*undefined*			?	
24:31	7:0	cmd	W		none	Selects an architected command (0x05=reset).

Writing to this architected register, with `cmd=0x05`, initiates a hard reset of the CPU and MIOC. Writing any other value into the `cmd` field has no effect, since the PA7300LC defines no other command codes. Unused bits should be set to zero.

Since this register is located in broadcast space, reading from it is illegal.

This register is functionally identical to the COMMAND_LOCAL register at 0xfffc0030.

See also the COMMAND register at 0xf###e030. That register is similar to this one, but will generate a TOC rather than completely reset the PA7300LC.

12. Diagnose Instructions

12.1 Diagnose Instruction Encodings

Instruction	Opcode Extesion						
	hex	binary					
	19:26	19	20:22	23	24	25	26
MTCPU	12	X	001	X	X	1	X
MFCPU_T	A0	1	010	X	X	0	X
MFCPU_C	30	0	011	X	X	0	X
IC_DIAG	40	0	100	X	X	0	X
DC_DIAG	42	0	100	X	X	1	X
ILAB_READ	C0	1	100	X	X	0	X
ILAB_WRITE	C2	1	100	X	X	1	X
TOC_EN	50	0	101	X	X	0	X
TOC_DIS	52	0	101	X	X	1	X
SHDW_GR	D0	1	101	X	X	0	X
GR_SHDW	D2	1	101	X	X	1	X
HBTRFI	60	0	110	0	X	X	X
DR_PAGE0	70	0	111	X	X	0	X
DR_PAGE1	72	0	111	X	X	1	X

X indicates reserved bit (assume X==0).

Engineers in ESL have set up a macro file for diagnose instructions for the PA7300LC. Users are STRONGLY encouraged to use this macro file when coding diagnose instructions for the PA7300LC. Contact us for an electronic copy of this file. We also have macros for cache hint, and the implementation specific instructions.

12.2 Diagnose Instruction Descriptions

The PA7300LC diagnose instructions and their encodings are described below.

Move to Diagnose Register MTCPU

Format: `MTCPU x,t`

05	t	x	0	12	0
6	5	5	3	8	5

Purpose: To move data into a CPU diagnose register.

Description: The contents of General Register x is moved to CPU Diagnose Register t. There are no Coprocessor diagnose registers. Not all diagnose register bits can be affected by a Move To Diagnose Instruction because some diagnose register bits are read–only. These cases are identified in the section describing the Diagnose Registers.

The HPMC bits in CPU Diagnose Register 0 behave differently than other diagnose bits. A Move To Diagnose instruction with a ‘1’ value of GR[x] for these bit positions causes the corresponding bit to be reset while a ‘0’ value leaves the bit unchanged. This allows MTCPU to DR0 without changing the value of these error bits.

Because there are two ‘‘pages’’ of CPU diagnose registers, the proper page must be selected before executing this instruction. This is accomplished via the DR_PAGE0 and DR_PAGE1 instructions.

Operation: DR[t,dr_page] ← GR[x];

Exceptions: Privileged operation trap

Move From CPU Diagnose Register via TH MFCPU_T

Format: `MFCPU_T r,t_th`

05	r	0	0	A0	t_th
6	5	5	3	8	5

Purpose: To move data from a CPU diagnose register to a general register.

Description: The contents of CPU diagnose register r is moved to general register t_th. There are two ‘‘flavors’’ of MFCPU in order to make the design easier. All CPU diagnose registers except 0 and 8 use MFCPU_T. Not all diagnose register bits in each register can be affected by a Move From Diagnose Instruction because some diagnose register bits are write–only. These cases are identified in the section describing the Diagnose Registers. There are no Coprocessor diagnose registers.

Because there are two ‘‘pages’’ of CPU diagnose registers, the proper page must be selected before executing this instruction. This is accomplished via the DR_PAGE0 and DR_PAGE1 instructions.

Operation: GR[t_th] ← DR[r,dr_page];

Exceptions: Privileged operation trap

Move From CPU Diagnose Register via CH MFCPU_C

Format: `MFCPU_C r,t_ch`

05	r	t_ch	0	30	0
6	5	5	3	8	5

Purpose: To move data from a CPU diagnose register to a general register.

Description: The contents of CPU diagnose register r is moved to general register t_ch. There are two ‘‘flavors’’ of MFCPU in order to make the design easier. Diagnose registers 0,8 are the only ones that uses MFCPU_C. All other CPU diagnose registers use MFCPU_T. Not all diagnose register bits in each register can be affected by a Move From Diagnose Instruction because some diagnose register bits are write–only. These cases are identified in the section describing the Diagnose Registers. There are no Coprocessor diagnose registers.

Because there are two ‘‘pages’’ of CPU diagnose registers, the proper page must be selected before executing this instruction. This is accomplished via the DR_PAGE0 and DR_PAGE1 instructions.

Operation: GR[t_ch] ← DR[r,dr_page];

Exceptions: Privileged operation trap

Instruction Cache Diagnose Operation IC_DIAG

Format: `IC_DIAG`

05	0	0	0	40	0
6	5	5	3	8	5

Purpose: To initiate an instruction cache diagnose or BIST operation.

Description: The diagnose/BIST engine located in the instruction cache is started. A subsequent ‘‘sync’’ instruction will cause the CPU to stall until the diagnose/BIST operation completes. See the Instruction Cache chapter for more details about the diagnose/BIST engine.

Operation: begin_IC_BIST;

Exceptions: Privileged operation trap

Data Cache Diagnose Operation DC_DIAG

Format: `DC_DIAG`

05	0	0	0	42	0
6	5	5	3	8	5

Purpose: To initiate a data cache diagnose or BIST operation.

Description: The diagnose/BIST engine located in the data cache is started. A subsequent ‘‘sync’’ instruction will cause the CPU to stall until the diagnose/BIST operation completes. See the Data Cache chapter for more details about the diagnose/BIST engine.

Operation: begin_DC_BIST;

Exceptions: Privileged operation trap

ILAB Diagnose Read ILAB_READ

Format: `ILAB_READ`

05	0	0	0	C0	0
6	5	5	3	8	5

Purpose: To perform an ILAB diagnose read operation.

Description: A single ILAB entry is read into the ILAB_VPN and ILAB_RPN cpu diagnose registers. See the TLB chapter for more details.

Operation: ilab_diagnose_read;

Exceptions: Privileged operation trap

ILAB Diagnose Write ILAB_WRITE

Format: `ILAB_WRITE`

05	0	0	0	C2	0
6	5	5	3	8	5

Purpose: To perform an ILAB diagnose write operation.

Description: A single ILAB entry is written from the ILAB_VPN and ILAB_RPN cpu diagnose registers. See the TLB chapter for more details.

Operation: ilab_diagnose_write;

Exceptions: Privileged operation trap

Transfer of Control Enable TOC_EN

Format: `TOC_EN`

05	0	0	0	50	0
6	5	5	3	8	5

Purpose: To enable the taking of a transfer of control (TOC) trap.

Description: An internal flag that allows a TOC is set. It does not affect collecting the TOC condition, it merely controls the mask on the TOC trap in the same manner that the PSW–M bit masks an HPMC trap.

Operation: TOC_enable ← 1;

Exceptions: Privileged operation trap

Transfer of Control Disable TOC_DIS

Format: `TOC_DIS`

05	0	0	0	52	0
6	5	5	3	8	5

Purpose: To disable the taking of a transfer of control (TOC) trap.

Description: An internal flag that allows a TOC is cleared. It does not affect collecting the TOC condition, it merely controls the mask on the TOC trap in the same manner that the PSW–M bit masks an HPMC trap.

Operation: TOC_enable ← 0;

Exceptions: Privileged operation trap

Move From Shadow Registers SHDW_GR

Format: SHDW_GR

05	0	0	0	D0	0
6	5	5	3	8	5

Purpose: To move the shadow registers into their corresponding general registers.

Description: The contents of the shadow registers are set into general register 1, 8, 9, 16, 17, 24, and 25. This instruction must be preceded by 2 sync instructions to guarantee no stall–on–use bypass cases colliding with the GR move to/from shadow.

Operation:

```
GR[1]  ← SHR[0];
GR[8]  ← SHR[1];
GR[9]  ← SHR[2];
GR[16] ← SHR[3];
GR[17] ← SHR[4];
GR[24] ← SHR[5];
GR[25] ← SHR[6];
```

Exceptions: Privileged operation trap

Restrictions: A “sync” instruction should precede these instructions to separate them from any preceding loads, stores, or flushes.

Move To Shadow Registers GR_SHDW

Format: `GR_SHDW`

05	0	0	0	D2	0
6	5	5	3	8	5

Purpose: To set the shadow registers from their corresponding general registers.

Description: The contents of the general registers 1, 8, 9, 16, 17, 24, and 25 are set into the shadow registers. This instruction must be preceded by 2 sync instructions to guarantee no stall–on–use bypass cases colliding with the GR move to/from shadow.

Operation:

```
SHR[0] ← GR[1];
SHR[0] ← GR[8];
SHR[0] ← GR[9];
SHR[0] ← GR[16];
SHR[0] ← GR[17];
SHR[0] ← GR[24];
SHR[0] ← GR[25];
```

Exceptions: Privileged operation trap

Restrictions: A “sync” instruction should precede these instructions to separate them from any preceding loads, stores, or flushes.

Implementation Dependant RFI HBTRFI

Format: HBTRFI

05	0	0	0	60	0
6	5	5	3	8	5

Purpose: To perform an implementation dependant RFI operation.

Description: This instruction behaves exactly like a normal RFI instruction with some additional implementation dependant side–effects.

Operation:

```
if (priv != 0)
        priviledged_operation_trap;
else {
        PSW          ← IPSW;
        IAOQ_Back ← IIAOQ_Back;
        IAOQ_Front ← IIAOQ_Front;
        IASQ_Back  ← IIASQ_Back;
        IASQ_Front ← IIASQ_Front;
        /* plus implementation dependant side–effects */
}
```

Exceptions: Privileged operation trap

Select Diagnose Page 0 DR_PAGE0

Format: `DR_PAGE0`

05	0	0	0	70	0
6	5	5	3	8	5

Purpose: To select CPU diagnose register page 0.

Description: An internal bit that indicates which diagnose page subsequent MTCPU, MFCPU_T, and MFCPU_C instructions will operate on is set to page 0.

Operation: dr_page ← 0;

Exceptions: Privileged operation trap

Select Diagnose Page 1 DR_PAGE1

Format: `DR_PAGE1`

05	0	0	0	72	0
6	5	5	3	8	5

Purpose: To select CPU diagnose register page 1.

Description: An internal bit that indicates which diagnose page subsequent MTCPU, MFCPU_T, and MFCPU_C instructions will operate on is set to page 1.

Operation: dr_page ← 1;

Exceptions: Privileged operation trap

13. PA7300LC Specific Instructions

This section describes the set of non–diagnose instructions that the PA7300LC implements that are not part of the PA–RISC 1.1 Architecture, Third edition. These instructions are purely specific to the PA7300LC and are not guaranteed to be forward or backward compatible with other PA–RISC processors.

Halfword Parallel Add HADD

Format:

```
HADD        r1,r2,t
HADD,ss     r1,r2,t
HADD,us     r1,r2,t
```

02	r2	r1	0	minor	t
6	5	5	4	7	5

minor	instruction
18	HADD,us
1A	HADD,ss
1E	HADD
58	[reserved]

Purpose: To perform two parallel halfword additions.

Description: This instruction specifies two parallel halfword adds. The two left halfwords of r1 and r2 added, and the two right halfwords of r1 and r2 are added. No exception is generated on overflow. The determination of how overflow is handled is given by the completer:

none: two unsigned operands in the range [0000,FFFF] are added with modular arithmetic (no saturation).

,ss: two signed operands in the range [8000,7FFF] are added, with signed saturation, i.e., the result is clamped between [8000,7FFF].

,us: an unsigned operand in the range [0000,FFFF] and a signed operand in the range [8000,7FFF] are added, with unsigned saturation, i.e., the result is clamped between [0000,FFFF].

Conditions: No conditions are generated. Software must put zeros in the ‘‘cf’’ field. A non–zero encoding of the ‘‘cf’’ field is undefined.

Operation:

```
switch (completer) {
    case “,ss”:    resL ← sign_ext (r1L, 16) + sign_ext (r2L, 16);
                   if (resL > 0x00007FFF) then tL ← 0x7FFF;
                   else if (resL < 0xFFFF8000) then tL ← 0x8000;
                   else tL ← resL mod $2^{16}$;

                   resR ← sign_ext (r1R, 16) + sign_ext (r2R, 16);
                   if (resR > 0x00007FFF) then tR ← 0x7FFF;
                   else if (resR < 0xFFFF8000) then tR ← 0x8000;
                   else tR ← resR mod $2^{16}$;

    case “,us”:    resL ← zero_ext (r1L,16) + sign_ext (r2L, 16);
                   if (resL > 0x0000FFFF) then tL ← 0xFFFF;
                   else if (resL < 0x00000000) then tL ← 0x0000;
                   else tL ← resL mod $2^{16}$;

                   resR ← zero_ext (r1R,16) + sign_ext (r2R, 16);
                   if (resR > 0x0000FFFF) then tR ← 0xFFFF;
                   else if (resR < 0x00000000) then tR ← 0x0000;
                   else tR ← resR mod $2^{16}$;

    default:       tL ← (r1L + r2L) mod $2^{16}$;
                   tR ← (r1R + r2R) mod $2^{16}$;
}
```

Exceptions: None

Halfword Parallel Subtract HSUB

Format:

```
HSUB        r1,r2,t
HSUB,ss     r1,r2,t
HSUB,us     r1,r2,t
```

02	r2	r1	0	minor	t
6	5	5	4	7	5

minor	instruction
08	HADD,us
0A	HADD,ss
0E	HADD
48	[reserved]

Purpose: To perform two parallel halfword subtractions.

Description: This instruction specifies two parallel halfword subtracts. The left halfword of r2 is subtracted from the left halfword of r1. The right halfword of r2 is subtracted from the right halfword of r1. No exception is generated on overflow. The determination of how overflow is handled is given by the completer:

none: two unsigned operands in the range [0000,FFFF] are subtracted, with modular arithmetic (no saturation).

,ss: two signed operands in the range [8000,7FFF] are subtracted, with signed saturation, i.e., the result is clamped between [8000,7FFF].

,us: an signed operand in the range [8000,7FFF] is subtracted from an unsigned operand in the range [0000,FFFF], with unsigned saturation, i.e., the result is clamped between [0000,FFFF].

Conditions: No conditions are generated. Software must put zeros in the ‘‘cf’’ field. A non–zero encoding of the ‘‘cf’’ field is undefined.

Operation:

```
switch (completer) {
    case “,ss”:    resL ← sign_ext (r1L, 16) – sign_ext (r2L, 16);
                   if (resL > 0x00007FFF) then tL ← 0x7FFF;
                   else if (resL < 0xFFFF8000) then tL ← 0x8000;
                   else tL ← resL mod $2^{16}$;

                   resR ← sign_ext (r1R, 16) – sign_ext (r2R, 16);
                   if (resR > 0x00007FFF) then tR ← 0x7FFF;
                   else if (resR < 0xFFFF8000) then tR ← 0x8000;
                   else tR ← resR mod $2^{16}$;

    case “,us”:    resL ← zero_ext (r1L,16) – sign_ext (r2L, 16);
                   if (resL > 0x0000FFFF) then tL ← 0xFFFF;
                   else if (resL < 0x00000000) then tL ← 0x0000;
                   else tL ← resL mod $2^{16}$;

                   resR ← zero_ext (r1R,16) – sign_ext (r2R, 16);
                   if (resR > 0x0000FFFF) then tR ← 0xFFFF;
                   else if (resR < 0x00000000) then tR ← 0x0000;
                   else tR ← resR mod $2^{16}$;

    default:       tL ← (r1L – r2L) mod $2^{16}$;
                   tR ← (r1R – r2R) mod $2^{16}$;
}
```

Exceptions: None

Halfword Parallel Average HAVE

Format: `HAVE r1,r2,t`

02	r2	r1	0	16	t
6	5	5	4	7	5

Purpose: To perform two parallel halfword averages.

Description: This instruction specifies two parallel halfword averages. The two left halfwords of r1 and r2 are averaged, and the two right halfwords of r1 and r2 are averages. The average is obtained by added the two hlafwords, and shifting the result right by one bit, to perform a divide by 2, with the halfword carry bit from the addition shifted back into the leftmost porition of each result.

This operation is defined for unsigned operands, with modular arithmetic. No overflow is defined for unsigned operands, with modular arithmetic. No overflow is generated. Unbiased rounding is performed on the result to reduce the accumulation of rounding errors with cascaded operations.

Conditions: No conditions are generated. Software must put zeros in the ‘‘cf’’ field. A non–zero encoding of the ‘‘cf’’ field is undefined.

Operation:

```
resL ← r1L + r2L;
resR ← r1R + r2R;
tL ← round (cL, resL >> 1);
tR ← round (cR, resR >> 1);
```

where ‘‘round’’ means unbiased rounding defined as follows:

Let yz be the 2 least significant bits of the sum before the 1–bit right shift, and let y’ be the least significant bit of the result after the right shift and rounding:

yz	round to y’	with rounding error
00	0	0
01	1	+0.5
10	1	0
11	1	–0.5

Exceptions: None

Halfword Shift Right and Add HSRxADD

Format:

```
HSR1ADD     r1,r2,t
HSR2ADD     r1,r2,t
HSR3ADD     r1,r2,t
```

02	r2	r1	0	minor	t
6	5	5	4	7	5

minor	instruction
2A	HSR1ADD
2C	HSR2ADD
2E	HSR3ADD

Purpose: To perform two parallel halfword shift right and additions.

Description: This instruction specifies two parallel halfword shift and adds. It is used as a primitive operation in performing halfword multiplication by fractional constants. Eash HSRxADD performs the equivalent of two parallel halfword multiply and add operations, where the multiplication is by 1/2, 1/4 or 1/8.

The r1L and r1R inputs are first shifted right by 1, 2, or 3 bits for HSR1ADD, HSR2ADD, or HSR3ADD respectively and added to r2L and r2R. The signs of r1L and r1R are preserved during the shift. Signed saturation is performed bas on overflows calculated by the adder in the same way as for the HADD instructions.

Conditions: No conditions are generated. Software must put zeros in the ‘‘cf’’ field. A non–zero encoding of the ‘‘cf’’ field is undefined.

Operation:

```
resL ← sign_ext (r1L >> x, 16–x) + sign_ext (r2L, 16);
if (resL > 0x00007FFFF) then tL ← 0x7FFF;
else if (resL < 0xFFFF8000) then tL ← 0x8000;
else tL ← resL mod $2^{16;}$

resR ← sign_ext (r1R >> x, 16–x) + sign_ext (r2R, 16);
if (resR > 0x00007FFFF) then tR ← 0x7FFF;
else if (resR < 0xFFFF8000) then tR ← 0x8000;
else tR ← resR mod $2^{16;}$
```

where ‘‘x’’ is 1, 2, or 3 for HSR1ADD, HSR2ADD, or HSR3ADD respectively.

Exceptions: None

Halfword Shift Left and Add HSLxADD

Format:

```
HSL1ADD    r1,r2,t
HSL2ADD    r1,r2,t
HSL3ADD    r1,r2,t
```

02	r2	r1	0	minor	t
6	5	5	4	7	5

minor	instruction
3A	HSL1ADD
3C	HSL2ADD
3E	HSL3ADD

Purpose: To perform two parallel halfword shift left and additions.

Description: This instruction specifies two parallel halfword shift and adds. It is used as a primitive operation in performing halfword multiplication by integer constants. Eash HSLxADD performs the equivalent of two parallel halfword multiply and add operations, where the multiplication is by 2, 4 or 8.

The r1L and r1R inputs are first shifted left by 1, 2, or 3 bits for HSL1ADD, HSL2ADD, or HSL3ADD respectively and added to r2L and r2R. The shifter calculates signed positive and signed negative overflows to be used in determining saturation.

Signed saturation is performed bas on overflows calculated by the added in the same way as for the HADD instruction or by overflow detected by the preshifter. The preshifter overflow occurs when the any bit shifted out differs from the leftmost bit following the shift.

Conditions: No conditions are generated. Software must put zeros in the ‘‘cf’’ field. A non–zero encoding of the ‘‘cf’’ field is undefined.

Operation:

```
op1L ← sign_ext (r1L << x, 16+x);
resL ← op1L + sign_ext (r2L, 16);
if (op1L > 0x00007FFFF) then tL ← 0x7FFF;
else if (op1L < 0xFFFF8000) then tL ← 0x8000;
else if (resL > 0x00007FFFF) then tL ← 0x7FFF;
else if (resL < 0xFFFF8000) then tL ← 0x8000;
else tL ← resL mod $2^{16;}$

op1R ← sign_ext (r1R << x, 16+x);
resR ← op1R + sign_ext (r2R, 16);
if (op1R > 0x00007FFFF) then tR ← 0x7FFF;
else if (op1R < 0xFFFF8000) then tR ← 0x8000;
else if (resR > 0x00007FFFF) then tR ← 0x7FFF;
else if (resR < 0xFFFF8000) then tR ← 0x8000;
else tR ← resR mod $2^{16;}$
```

where ‘‘x’’ is 1, 2, or 3 for HSL1ADD, HSL2ADD, or HSL3ADD respectively.

Exceptions: None

Notes: An overflow in the preshifter takes precedence over an overflow in the adder.

Fast TLB Insert IxTLBxF

Format:

```
IITLBPF     r
IITLBAF     r
IDTLBPF     r
IDTLBAF     r
```

01	0	r	0	minor	0	0
6	5	5	2	8	1	5

minor	instruction
10	IITLBPF
11	IITLBAF
50	IDTLBPF
51	IDTLBAF

Purpose: To add an entry to the TLB using the Interruption Instruction Address Queues (for IITLBxF) or the Interruption Parameter Registers (for IDTLBxF) as the source for the virtual address.

Description: These instructions insert to the virtual address in the IIASQ/IIAOQ registers or the ISR/IOR registers for IITLBxF or IDTLBxF respectively. The protection or real address is specified by the ‘‘r’’ field for IxTLBP or IxTLBA respectively. These instructions execute with less penalty cycles than the normal, architected, TLB insertion instructions.

Operation: They behave like the corresponding IITLBP, IITLBA, IDTLBP, and IDTLBA instructions except that the virtual address inserted is specified implicitly through control registers rather than explicitly by the instruction.

Exceptions: Privileged operation trap

Restrictions: These instructions may be executed only at the most priviledged level.

For one instruction bundle slot following any fast insert, software must ensure that no loads, stores, or branches occur. With dual–issue, this might affect the next two instructions.

The instruction immediately following any fast insert must not be a memory management instruction (major = 01).

Every IITLBPF or IDTLBPF should be preceded by an IITLBAF or IDTLBAF, respectively. They do not have to be back–to–back, but there should not be any traps between the two and no other TLB insert should appear between them.

Multi–endian Load Word Indexed LDWxX

Format:

```
LDWCX,cmplt x(s,b),t
LDWBX,cmptl x(s,b),t
LDWLX,cmptl x(s,b),t
```

03	b	x	s	u	0	z	3	m	t
6	5	5	2	1	1	2	4	1	5

z	instruction
00	LDWCX
10	LDWBX
11	LDWLX

Purpose: To load a word into a general register with a byte ordering differing from that specified by the PSW E–bit.

Description: These instructions behave exactly like a LDWX instruction, except that the byte ordering of the data loaded is specified by the *z* field instead of, or in combinition with, the PSW E–bit. The LDWCX intruction uses a byte ordering specifed by the complement of the value of the PSW E–bit. The LDWBX instruction uses big–endian byte ordering, independant of the value of the PSW E–bit. The LDWLX instruction uses little–endian byte ordering, independant of the value of the PSW E–bit.

Operation: (See LDWX)

Exceptions: (See LDWX)

Multi–endian Load Word Short **LDWxS**

Format:

```
LDWCS,cmplt d(s,b),t
LDWBS,cmptl d(s,b),t
LDWLS,cmptl d(s,b),t
```

03	b	im5	s	a	1	z	3	m	t
6	5	5	2	1	1	2	4	1	5

z	**instruction**
00	LDWCS
10	LDWBS
11	LDWLS

Purpose: To load a word into a general register with a byte ordering differing from that specified by the PSW E–bit.

Description: These instructions behave exactly like a LDWS instruction, except that the byte ordering of the data loaded is specified by the *z* field instead of, or in combinition with, the PSW E–bit. The LDWCS intruction uses a byte ordering specifed by the complement of the value of the PSW E–bit. The LDWBS instruction uses big–endian byte ordering, independant of the value of the PSW E–bit. The LDWLS instruction uses little–endian byte ordering, independant of the value of the PSW E–bit.

Operation: (See LDWS)

Exceptions: (See LDWS)

Multi–endian Store Word Short STWxS

Format:

```
STWCS,cmplt,cc    r,d(s,b)
STWBS,cmplt       r,d(s,b)
STWLS,cmptl       r,d(s,b)
```

03	b	r	s	a	1	z	B	m	im5
6	5	5	2	1	1	2	4	1	5

z	instruction
00	STWCS
10	STWBS
11	STWLS

Purpose: To store a word from a general register with a byte ordering differing from that specified by the PSW E–bit.

Description: These instructions behave exactly like a STWS instruction, except that the byte ordering of the data loaded is specified by the *z* field instead of, or in combinition with, the PSW E–bit. The STWCS intruction uses a byte ordering specifed by the complement of the value of the PSW E–bit. The STWBS instruction uses big–endian byte ordering, independant of the value of the PSW E–bit. The STWLS instruction uses little–endian byte ordering, independant of the value of the PSW E–bit. Note that the ‘‘,BC’’ cache control hint may be used with the STWCS instruction.

Operation: (See STWS)

Exceptions: (See STWS)

14. Coding Hints

System peformance is based on the code pathlength (number of instructions) times the instruction CPI (cycles per instruction). Software should attempt to reduce instruction CPI in addition to the code pathlength. This section summarizes ways in which code can be optimized for performance on the PA7300LC.

14.1 Superscalar Execution

The PA7300LC is capable of executing two instructions at a time. The instructions proceed together through the execution pipeline and are said to be **bundled**. Superscalar execution is functionally transparent to software, that is, the effects of an instruction are the same whether it was executed alone or as part of a superscalar bundle.

There are four kinds of restrictions placed upon bundling: functional unit contention, data dependency restrictions, control flow restrictions, and special instruction type restrictions. These are all described in detail below.

The bundling rules are applied entirely at run time by hardware. Compilers and handcoders seeking performance will want to order their instructions so that bundling is maximized, but they are not required to do so. The only side–effect of suboptimal instruction ordering is lower performance.

14.1.1 Instruction Classes

For the purpose of these bundling rules the instruction set is divided into classes. Below is a list of the classes and which opcodes fall into them. The opcodes are given as hex values for bits [0:5]. Conditions in parentheses refer to values of particular bits of the instruction.

Class	**Encoding**	**Description**
flop	0C([26] == 0), OE, O6, 26	floating point operations
ldw	12	LDW instruction
stw	1A	STW instruction
ldst	09/OB([27:29] != 0) 03, 10, 11, 13–19, 1B–1F	FP loads and stores other loads and stores
flex	08, 0A, 0D, 02/24/25/2C/2D([16:19] == 0)	integer ALU
mm	34/35([16:18] == 0), 27, 2E, 2F, 36,37, 3C–3F	shifts, extracts, deposits
nul	02/24/25/2C/2D([16:19] != 0), 34/35([16:18] != 0)	might nullify successor
bv	38, 3A([16] == 1)	BE, BV instructions
br	20–23, 28–2B, 30–33, 3A([16] == 0), 3B	other branches
fsys	OC([26] == 1), 09/OB([27:29] == 0)	FTEST instruction FP status/exception instructions
sys	00, 01, 04, 05, 07, 0F, 39	system control instructions

14.1.2 Functional Unit Contention

The following table indicates which combinations of instruction classes are allowed to bundle together. The rows represent the first, or older, instruction while the columns represent the second, or younger, instruction.

	flop	**ldw**	**stw**	**ldst**	**flex**	**mm**	**nul**	**bv**	**br**	**fsys**	**sys**
flop		✓	✓	✓	✓	✓	✓	✓	✓		
ldw	✓	○			✓	✓	✓		✓		
stw	✓		○		✓	✓	✓		✓		
ldst	✓				✓	✓	✓		✓		
flex	✓	✓	✓	✓	✓	✓	✓		✓	✓	
mm	✓	✓	✓	✓	✓					✓	
nul	✓										
bv											
br											
fsys											
sys											

The ldw/ldw and stw/stw bundles (indicated with the symbol “○”) are a special case called double word load/store. The effective addresses of the data references of the two instructions must point to different words of an aligned doubleword. The two instructions must use the same space and base register, the base register must contain an even word address (its bit 29 must be 0), and the displacements coded in the two instructions must be word aligned.

14.1.3 Data Dependencies

An instruction which modifies a register will not be bundled with a subsequent instruction which uses that register as an operand.

A floating point load to one word of a doubleword floating–point register will not be bundled with a flop which uses the other word.

A flop will not be bundled with a floating–point load if the flop and load have the same target register, or even if their targets are different words of the same doubleword register.

An instruction which might set the carry/borrow bits will not be bundled with an instruction which uses the carry/borrow bits. For this purpose, the instructions which might set carry/borrow are opcodes 24, 25, 2C, and 2D, and opcode 02 with bit 21 equal to 1 and either bit 23 equal to 0 or bits 24 and 25 equal to 0. The instructions which use carry borrow are SUBB, SUBBO, ADDC, ADDCO, DS,DCOR, and IDCOR.

A ldw/ldw or stw/stw pair that appears immediately after a bundle containing an instruction that will modify the base register of the pair will not be bundled. Similarly, a ldw/ldw or stw/stw pair will not bundle if the pair’s base register is the target of an unresoved load miss.

14.1.4 Control Flow

An instruction which is in the delay slot of a branch is never bundled.

An instruction which is executed as the target of a taken branch andwhich is at an odd word address is never bundled.

As shown in the bundle table above, an instruction which *might* nullify its successor will not be bundled with that successor, unless that successor is in the **flop** class. Specifically the following types of instructions might nullify their successor: opcode {02,24,25,2C,2D,34,35} with a nonzero test condition (bits 16:19 for {02,24,25,2C,2D}, or bits 16:18 for opcode {34,35}).

A ldw/ldw or stw/stw pair is not bundled if the previous instruction or instruction pair might nullify. For this purpose the instructionswhich might nullify are those mentioned above plus the ftest instruction. The nullifying instruction would have to be right before the first ldw or stw, or seperated from it only by a flop.

An instruction which is executed as the first target of an RFI or RFIR is never bundled. In addition, the second target is never bundled if it is at an odd word address.

14.1.5 Special Instruction Types

Instructions in the **sys** class are never bundled.

14.2 Store Instructions

Unlike some prior PA–RISC processors, there is no penalty for a word or doubleword store followed by an instruction that accesses the data cache. In other words, there is no ‘‘store tail’’ penalty for these stores. A one cycle stall penalty, however, still exists for subword stores that are followed by an instruction bundle containing a data cache reference (load, store, FDC, PDC, or semaphore). In addition, semaphore instructions (LDCWS, LDCWX) are always subject to a one cycle stall penalty.

14.3 Integer Load Instructions

Try not to have the instruction bundle immediately following a load to a general register contain an instruction which uses the register as an operand. This causes a one cycle interlock stall and will also disable dual–issuing that instruction pair, possibly resulting in another penalty.

14.4 I–Cache Misses

Instruction cache misses will stall the pipeline for a minimum of 5 cycles. Miss addresses are issued for the critical doubleword, and that critical doubleword is returned first. The pipeline will stall for an instruction cache miss the state after the address is issued provided that state involves a pipeline step. This address may actually be one doubleword ahead of pipeline execution, since dual issue may require a new doubleword to bundle with. That means the pipeline may stall, and a line may be copied in, which isn’t actually hit by the execution stream.

14.5 D–Cache Misses

There are fewer penalty cycles associated with data cache misses on the PA7300LC as compared with the PA7100LC. However, there are still some penalty cycles that are a function of the instruction stream. Data cache misses occur on loads, stores and semaphores that reference data not currently in the data cache. They also occur on data cache flushes and purges that reference data that *is* currently in the data cache. All data cache misses, with the exception of some stores to I/O space, are subject to a minimum one cycle stall. If the bundle following the instruction causing the data cache miss contains an instruction that references the data cache, then the minimum stall becomes two cycles. There are several factors that lead to a greater than minimum number of stall cycles. If the bundle following the instruction with the miss contains a subword store or a semaphore, then an additional stall cycle will occur. If the instruction causing the miss was a load, there will be a one cycle stall when the critical data arrives from the memory controller. If the memory controller’s queues are full (five transactions) then there will be stall cycles until the copyin and/or copyout transactions causes by the miss are accepted. If there are already two cache misses pending and the instruction causing the miss was a load, semaphore or cacheable store, then there will be stall cycles until one of the prior misses completes[1]. If there is one load miss pending and the current miss is a load miss, then there will be stall cycles until the critical data for the first miss arrives from the memory controller. Subword store and semphore misses will always stall until the critical data arraives from the memory controller.

Data cache flushes and purges are subject to an unconditional one cycle stall. Additional stall cycles may occur as described above.

There are two classes of stalls related to cache misses. These stalls are caused by instructions subsequent to the one causing the cache miss. The first type is caused by an instruction that references the target register of a load that misses. When a load miss occurs, the load instruction itself only stalls until the memory read transaction is issued to the memory controller. It does not wait until the memory controller returns the data to be loaded into the target register (refered to as the critical data). If a subsequent instruction attempts to read the target register before the critical data arrives, the processor will stall until the memory control returns the critical data and the target register is updated. The second type of stall is causes by a data cache instruction that reads the same physical cache line that is the target of a prior cache miss that has not yet completed. If an instruction attempts to read data that has not yet arrived from the memory controller, or attempts to read data being brought in by the second of two pending misses, the processor will stall until the data arrives from the memory control *and* there is only one miss pending. The second requirement is due to the fact that data can only be bypassed from the front of the copyin queue. Note that, in addition to loads, subword stores and semaphores cause reads of the data cache to be performed. Word and doubleword stores to cache lines that are the target of pending cache misses cause no penalty cycles.

Because the processor does not necessarily stall even with two cache misses pending, software prefetching can lead to better performance increases than on the PA7100LC. This is because a software prefetch, initiated by a load to general register 0, is not treated like a load miss by the data cache and therefore can be completed in the ‘‘background’’ by the hardware and is less likely to cause processor stalls.

As alluded to above, stores to I/O space are handled specially. If the memory controller queue is not full, or close to full, then the processor will not stall, even for a single cycle, for word or doubleword stores to I/O space. Subword stores to I/O space are only subject to the potential store tail penalty described earlier. If the memory controller is full or close to full, then I/O stores behave like any other cache miss. Note that this optimization only occurs on stores to I/O space, not on all uncached stores.

The block copy (‘‘,BC’’) cache control hint can be used to reduce cache miss penalties. Unlike some prior processors, the block copy hint is implemented on both privileged and non–privileged stores and its performance is independent of privilege level. The cache control hint is ignored for subword stores and for stores that do not write the first word on a cache line. Subject to the above contraints, when the block copy hint is specifed for a store instruction and the line referenced by the store is not present in the cache, a line will be created in the cache without a memory read being performed. The line will always be zeroed, regardless of privilege level. If a dirty line resides at the location the new line is being created, the victim line will be copied out of the cache, which will cause memory traffic to write the line to memory. Because the same copyin buffer and mechanism is used as a normal cache miss, block copy hinted store misses are subject to the same processor stalls associated with cache miss except for those stalls related to waiting for the memory controller to deliver data. Even so, block copy hinted store misses will complete much faster that non–hinted misses.

The data cache is two–way set associative, so it does not suffer from many of the thrashing problems that can affect a direct mapped cache. Because of this (and because it became a timing speed path), the cache index into the data cache is not hashed as on prior processors. The data cache index is always directly derived from the virtual offset (if data translation is enabled) or the real offset (if data translation is disabled).

[1] Completion of a miss is defined to occur when the critical data is returned for semaphores and uncached (including I/O) loads or when the requested line is written into the data cache for cacheable loads and stores.

14.6 TLB Misses

The PA7300LC implements block TLB (also called BATC – Block Address Translation Cache) entries for the unified TLB. Eight block entries can each be programmed to map 128 – 16K pages (512Kbyte – 64Mbyte segments). These are intended to be used to map large continuous virtual spaces for both the OS and for graphics applications.

The OS should use the implementation specific intructions for fast inserts from the interruption parameter registers. In addition, there are many issues realted to the effective use of the hardware TLB handler. Contact Joe Martinka for a copy of a study of the TLB performance of PCX–T.

The unified TLB is a 96 entry fully associative TLB. The replacement algorithm for the TLB is decribed in the TLB chapter. To reduce the penalty for page crosses (TLB lookaside buffer update) the BV, BE, or BLE instructions should be used whenever possible when branching to a new page.

The PA7300LC also allows TLB entries to be locked in. Locked in entries will not be replaced until the entry is unlocked. This may be useful when real–time performance of a certain application is critical.

The PA7300LC implements the shadow registers and corresponding RFIR instruction as defined in revision 1.1 (3rd edition) of the PA–RISC architecture.

14.7 Memory Management Instructions

Most memory management instructions cause a substantial performance penalty. Data cache flushes and purges have a minimum one cycle stall penalty. Also, multiple sequential data cache flushes that hit dirty in the cache can quickly overload the memory buffers in the memory controller, stalling the processor for even more cycles while the slower DRAMs are being updated. Instruction cache flushes have a six cycle stall penalty. Anything that can be done to reduce the amount of cache flushing is likely to help performance significantly.

14.8 Graphics Features

The on–chip memory and I/O controller supports efficient movement of data across GSC with normal load and store instructions. The PA7300LC supports floating point word and doubleword load and store operations to I/O space. Under normal circumstances a floating point store to I/O will not incur any penalty cycles. The PA7300LC supports ‘‘accelerated’’ writes to I/O space. With the appropriate bit set in Diagnose Register 13, ACCEL_IO, writes to a subset of I/O addresses will procede in parallel with memory controller activity, avoiding bottlenecks associated with maintaining proper ordering. The PA7300LC also supports the ‘‘WRITEV’’ extention to the GSC protocol. These features should increase the efficiency of large data movements to the frame buffer.

14.9 Memory Moves

The speed of memory to memory moves on large blocks of data can be improved substantially by using doubleword loads and stores through the floating point registers. The use of software prefetching, via loads to general register 0, should also increase memory move performance.

14.10 LDW/LDW and STW/STW Bundles

Whenever possible, software should try to utilize bundling of integer loads and stores to memory. For this to occur, pairs of LDW or STW instructions must be used (none of the other load or store word variants), the same base register must be used for each instruction in the pair, and the effective addresses generated by the pair must be to different words within the same aligned doubleword. If these restrictions can be met, especially for long sequences of loads or stores, the integer memory bandwidth effectively doubles.

While the cpu will bundle loads and stores to uncached locations (including I/O space), this should be avoided as it does not result in improved bandwidth and may, in fact, result in worse performance than if the instructions had not been bundled together. Floating point doubleword loads and stores should be used when high bandwidth is desired with uncached locations.

15. Errata

This section lists all known functional defects in the PA7300LC. These were left unfixed after careful determination that they would cause little or no impact to our customers and end–users.

15.1 FTEST in Delay Slot of a Branch

Some FTEST intructions in the delay slot of a branch can cause incorrect instructions to be executed. Each and every one of the following conditions must be met in order for incorrect behavior to occur:

- A branch to the last instruction on a page different from the page the branch delay slot is in.
- Either a FCMP bundled with the branch, or a load to floating point register 0 bundled with the branch or in the prior instruction bundle.
- An FTEST in the delay slot of the branch.
- The FTEST must cause the branch target to be nullified.
- The FCMP or load to FP register 0 must flip the sense of the condition the FTEST is testing.
- The branch target page must either not be in the TLB, and the TLB hardware handler (if enabled) must fail to find the proper translation in the page table, or the branch target page is in the TLB or page table but with incorrect protection. In other words, the branch target would have taken an ITLB trap (6 or 7) if it had not been nullified by the FTEST.
- The page following the branch target must be in the TLB with the proper protection.
- The first line on the page following the branch target is not in the L1 Instruction cache.

If all these conditions are met, an instruction will be executed from the copyin buffer instead of servicing the cache miss for the instruction after the branch target. This will result in effectively random code being executed in place of the first few instruction on the page after the branch target.