Intel® IXP1240 Network Processor

Specification Update

March 2004

Notice: The IXP1240 may contain design defects or errors known as errata. Characterized errata that may cause the IXP1240's behavior to deviate from published specifications are documented in this specification update.

Part Number: 278505-005

Revision History

Revision Date	Revision	Description
08/15/01	001	Initial internal release.
12/14/01	002	Update for the B0 stepping.
03/11/01	003	
11/25/03	004	Added Errata 20.
03/25/04	005	Added Errata 21.

Contents

Preface

This document is an update to the specifications contained in the Related Documents table below. This document is a compilation of device and documentation errata, specification clarifications and changes. It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools.

We have endeavored to include all documented errata in the consolidation process, however, we make no representations or warranties concerning the completeness of this Specification Update.

Information types defined in Nomenclature are consolidated into the specification update and are no longer published in other documents.

This document may also contain information that was not previously published.

Related Documents

Title	Part Number
IXP1240 Network Processor Datasheet	278405
IXP1200 Network Processor Family Hardware Reference Manual	278303

Nomenclature

Errata are design defects or errors. These may cause the published (component, board, system) behavior to deviate from published specifications. Hardware and software designed to be used with any component, board, and system must consider all errata documented.

Specification Changes are modifications to the current published specifications. These changes will be incorporated in any new release of the specification.

Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in any new release of the specification.

Documentation Changes include typos, errors, or omissions from the current published specifications. These will be incorporated in any new release of the specification.

Note: Errata remain in the specification update throughout the product's life cycle, or until a particular stepping is no longer commercially available. Under these circumstances, errata removed from the specification update are archived and available upon request. Specification changes, specification clarifications and documentation changes are removed from the specification update when the appropriate changes are made to the appropriate product specification or user documentation (datasheets, manuals, etc.).

Summary Table of Changes

The following table indicates the errata, specification changes, specification clarifications, or documentation changes which apply to the IXP1250 Network Processor product. Intel may fix some of the errata in a future stepping of the component, and account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

Codes Used in Summary Table

Stepping

X:	Errata exists in the stepping indicated. Specification Change or Clarification that applies to this stepping.
(No mark) or (Blank box):	This erratum is fixed in listed stepping or specification change does not apply to listed stepping.

Page

(Page):	Page location of item in this document.

Status

Doc:	Document change or update will be implemented.
Fix:	This erratum is intended to be fixed in a future step of the component.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
Eval:	Plans to fix this erratum are under evaluation.

Errata

No.	Steppings				Page	Status	ERRATA
	A0	B0	–	–			
1	X	X			11	NoFix	SRAM Registers
2	X	X			11	NoFix	CSR Access Using PCI Memory Cycles
3	X	X			11	NoFix	PCI_DMA Instruction
4	X	X			12	NoFix	Clock Setup Time
5	X	X			12	NoFix	Hold Time Issues for all PCI Signals (Both Bused and Control)
6	X	X			12	NoFix	IX Bus Contention in Shared IX Bus Mode
7	X	X			12	NoFix	Tval max Timing Issues When Running at 66 MHz for all PCI Signals
8	X	X			13	NoFix	PCI CSR Corruption
9	X	X			14	NoFix	SRAM[WRITE_UNLOCK,..., BURST_COUNT] Instruction
10	X				16	Fixed	PCI_OUT_INT_MASK Register Bits Not Readable
11	X	X			16	NoFix	Spurious PCI Parity Errors
12	X				16	Fixed	SDRAM Arbiter
13	X				16	Fixed	Problem with CRC
14	X				16	Fixed	SDRAM_CRC
15	X				17	Fixed	SA1200 Software Reset
16	X	X			17	NoFix	Branch and Return
17	X				17	Fixed	PCI Parity Error Signal
18	X				18	Fixed	Find Bit
19	X	X			18	NoFix	SDRAM_CRC Residue Register Corrupted Data
20	X	X			19	NoFix	Read-Lock CAM Operations from the StrongARM* Core to SRAM
21	X	X			19	NoFix	66 MHz Capable Bit

Specification Changes

No.	Steppings				Page	SPECIFICATION CHANGES
	A0	B0	–	–		
1	X	X			20	SRAM Bus Signal Timing Parameters
2	X	X			20	SDRAM Bus Signal Timing Parameters
3	X	X			20	FCLK AC Parameter Measurements
4	X	X			20	SRAM SCLK Signal AC Parameters
5	X	X			21	SDRAM SDCLK AC Parameters

Specification Clarifications

No.	Steppings				Page	SPECIFICATION CLARIFICATIONS
	A0	B0	–	–		
1	X	X			22	SRAM Unlocks and Write Unlocks
2	X	X			22	Maximum Number of Chain_Ref Instructions
3	X	X			22	DMA Receive in Big Endian Mode

Documentation Changes

No.	Page	DOCUMENTATION CHANGES
		None for this release

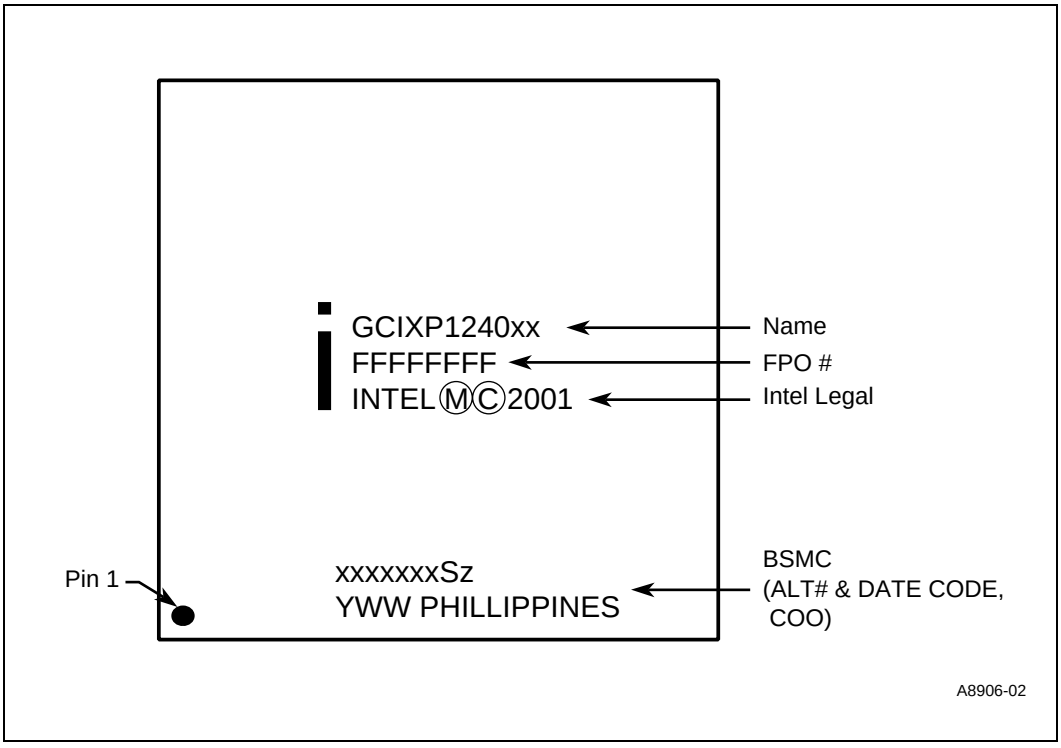
Identification Information

Markings

Product Name	Stepping	QDF Number	Marketing Part Number	Version
GCIXP1240AC[1]	A0	Q254	837510	232 MHz
GCIXP1240AC[1]	B0	Q253	837153	232 MHz
GCIXP1240AA	B0	NA	837150	166 MHz
GCIXP1240AB	B0	NA	837151	200 MHz
GCIXP1240AC	B0	NA	837152	232 MHz

1. Samples only.

Figure 1. Package Marking



Errata

1. SRAM Registers

Problem: Reads of the SRAM_BOOT_CONFIG and SRAM_SLOWPORT_CONFIG registers return the two inner bytes out of order.
SRAM_BOOT_CONFIG:
Written as: BRWA BCEA BRWD BCED
Read as: BRWA BRWD BCEA BCED

SRAM_SLOWPORT_CONFIG:
Written as: SRWA SCEA SRWD SCED
Read as: SRWA SRWD SCEA SCED

Implication: Bytes are read out of order.

Workaround: Software that reads these registers needs to put the bytes in the correct order.

Status: NoFix

2. CSR Access Using PCI Memory Cycles

Problem: The 128 byte window size is not supported. All other window sizes are valid. The CSR_BASE_ADDR_MASK[18] may only be set to 1, preventing the 128 byte window size from being selected. All other sizes are supported.

Implication: A 128 byte window size is not valid.

Workaround: Use a different PCI window size.

Status: NoFix

3. PCI_DMA Instruction

Problem: The Microengine PCI_DMA instruction SDRAM address operand is misaligned.

Implication: Incorrect addressing occurs if the address operand is not shifted.

Workaround: The SDRAM address operand of the PCI_DMA instruction requires a 1-bit right shift for proper quadword addressing. For example, the address of a Descriptor Pointer located at SDRAM address 0x1000 should be right-shifted 1 bit with the resulting operand value being 0x0800, as follows:

```
; fix address of SDRAM Descriptor Pointer
immed[tmp1, 0x1000]
alu[DESC_ADDR,--,B,tmp1,>>1]
; issue DMA request
pci_dma[DESC_ADDR, 0, any_queue]
```

Status: NoFix

4. Clock Setup Time

Problem: Because the SRAM and SDRAM setup times are directly related to the loading of SCLK and SDCLK, excessive setup times (T_{su}) may be seen under heavy loading conditions. The maximum value of T_{su} for both memory interfaces is 7.5 ns.

Implication: Inability to meet the data setup time specification for memory devices.

Workaround: Buffer SCLK and SDCLK with a zero skew clock buffer such as a Cypress CY2309.

NoFix

5. Hold Time Issues for all PCI Signals (Both Bused and Control)

Problem: The *PCI Local Bus Specification, Revision 2.2*, specifies a minimum Hold Time of 0 ns in Section 7.6.4.2. The IXP1240 requires a minimum hold time of 1.0 ns (t_h - Input Signal Hold Time from Clk).

Implication: System designers must constrain their design to tighter than worst-case PCI timing. One recommendation is to limit the trace length of the PCI bus resulting in a reduction of Tprop.

Workaround: None.

Status: NoFix

6. IX Bus Contention in Shared IX Bus Mode

Problem: In shared IX Bus mode, using the TK_IN pin to configure the initial IX Bus Owner and Ready Bus Master Mode could result in improper initialization. As a result, more than one IXP1240 may be the initial IX Bus Owner and Ready Bus Master.

Implication: This erratum causes contention on the IX Bus and the Ready Bus. It is also possible that the devices could initialize to the opposite state (not initial IX Bus Owner, Ready Bus Slave), in which case no device controls the Ready Bus as master.

Workaround: Use software to configure the initial IX Bus Owner and Ready Bus Master Mode instead of using the TK_IN strapping option. Pulldown the TK_IN inputs to all IXP1240s on a Shared IX Bus to inhibit initial IX Bus Owner and Ready Bus Master Mode. This ensures that no IXP1240 will be the initial IX Bus Owner and that all IXP1240s will be Ready Bus slaves. Boot software can then initialize one IXP1240 to initial IX Bus Owner and Ready Bus Master Mode by writing RDYBUS_TEMPLATE_CTL[8]=1. It is recommended to perform this operation as quickly as possible after reset to minimize the length of time the IX Bus and Ready Bus float.

Status: NoFix

7. Tval max Timing Issues When Running at 66 MHz for all PCI Signals

Problem: The *PCI Local Bus Specification, Revision 2.2* specifies a maximum Signal Valid Delay (Tval) time of 6.0ns in Section 7.6.4.2. The IXP1240 guarantees a worst-case Tval maximum of 6.5 ns.

Implication: The Tval maximum value of 6.5 ns requires a reduction in maximum flight time (Tprop) when running at 66 MHz.

Workaround: System designers must constrain their design to tighter than worst-case PCI timing. One recommendation is to limit the trace length of the PCI bus resulting in a reduction of Tprop.

Status: NoFix

8. PCI CSR Corruption

Problem: The PCI CSRs will be corrupted by any write access to the **PCI memory space, PCI IO space,** or **PCI config space** from the StrongARM* core to the PCI, if the previous transaction was a CSR write to registers in the PCI unit.

The affected address ranges are:

- PCI memory space (6000 0000 - 7FFF FFFF)
- PCI I/O space (5400 0000 - 5400 FFFF)
- PCI config space 0 and 1 (5200 0000 - 53BF FFFF)

The problem is dependent on the **sequence** of StrongARM* core transactions described above, and is not dependent on the **time** between these transactions.

Implication: Erratic behavior of PCI operations. The address of the register (PCI CSR) that gets corrupted during the PCI memory access equals the lower address bits of the PCI memory transaction.

Workaround: Always follow a **write** operation from the StrongARM* core to any CSR within the PCI block by a **read** to a register within the PCI.

Note: The read operation must immediately follow the write to the CSR.

The following is an example of a CSR read to the PCI_ADDR_EXTENSION 4200 0140h. Apart from the device driver writing to PCI, VxWorks also writes to PCI timer registers. To get around this:

1. Insert the following piece of code into the header file **IXP1240eb.h** located in the IXP1240 Developer's Workbench software installation in the directory **Boardsupport\VxWorks\IXP1240EB**.

```
#ifdef PCI_WORKAROUND
#define AMBA_TIMER_WRITE(reg, data) ({\
__asm__ __volatile (""); \
(*((volatile UINT32 *)(reg)) = (data)); \
((void)*(volatile UINT32 *)(IXP1240_PCI_ADDR_EXT)); \
__asm__ __volatile (""); })
#endif
```

2. Define the compiler directive **PCI_WORKAROUND**, either in your project build settings or as a #define in the header file. Without this directive, the compiler may reorder the instructions.
3. Rebuild the VxWorks image. Refer to the README file entitled *Building the VxWorks BSP*, for directions on how to build the image.

Status: NoFix

9. SRAM[WRITE_UNLOCK,..., BURST_COUNT] Instruction

Problem: The SRAM[WRITE_UNLOCK,..., ref_cnt] optional_token(s) instruction does not work correctly when ref_cnt > 1. Note that the command works correctly when the ref_cnt is equal to 1.

Implication: The SRAM[WRITE_UNLOCK,…,ref_cnt] command may not be completed by the SRAM unit when the ref_cnt is greater than 1. Instead a different SRAM command may get executed twice. This behavior is observed sporadically, when certain sequences of commands get queued to the SRAM unit. Because the commands arrive at the SRAM unit from different Microengine threads, it is impossible to determine if a software using this mode of command is prone to failure, or, when it will fail. The exact symptoms observed by the user will depend on the system software design and implementation.

For example, if the thread waits for the completion of the write_unlock command that gets dropped (either using the ctx_swap optional token, or, the sig_done optional token and ctx_arb[SRAM] command), then that thread will hang indefinitely. Further, the write to the memory location will not complete leading to data corruption problems. And, because a different command gets executed twice, two SRAM signals may be generated to a different thread, leading to improper program flow and data corruption.

It is recommended that the software programs not use the SRAM[WRITE_UNLOCK,…,ref_cnt] command with a ref_cnt > 1. If more than one long word needs to be written to memory, the software should use the workarounds described below.

Workaround: Two workarounds have been developed and are described below:

1. Break the SRAM[WRITE_UNLOCK,..., ref_cnt] instruction into a SRAM[write, …, ref_cnt] and SRAM[WRITE_UNLOCK,..., 1] pair.

Workaround 1 requires two Microengine Instruction Control Store locations, but results in one extra SRAM bus write cycle. It is possible to eliminate the extra bus cycle by suitably modifying the transfer register, address, and, ref_cnt fields, but may result extra Microengine instructions needed to compute the address. A simple case is illustrated in the examples below for this.

2. Break the SRAM[WRITE_UNLOCK,....], instruction into a SRAM[write, …, ref_cnt] and SRAM[UNLOCK,..., 1] pair.

Workaround 2 does not have the extra bus access but may require a third ctx_arb[SRAM] instruction if the program needs to wait for completion of the command. Examples shown below will illustrate this point.

Note: Great care must be taken to ensure that different optional tokens are carried over to the workaround to ensure correct program flow. The examples below are given to illustrate some key considerations.

Example A – No optional tokens.

Original code

```
SRAM[WRITE_UNLOCK, $x1, sAddr, 0, 3]
```

Workaround 1

```
SRAM[WRITE, $x2, sAddr, 1, 2]
SRAM[WRITE_UNLOCK, $x1, sAddr, 0, 1]
```

Workaround 2

```
SRAM[WRITE, $x1, sAddr, 0, 3]
SRAM[UNLOCK, --, sAddr, 0, 1]
```

Example B – CTX_SWAP optional token.

Original code
```
SRAM[WRITE_UNLOCK, $x1, sAddr, 0, 3], ctx_swap
```

Workaround 1
```
SRAM[WRITE, $x2, sAddr,1, 2]
SRAM[WRITE_UNLOCK, $x1, sAddr, 0, 1],ctx_swap
```

Workaround 2
```
SRAM[WRITE, $x1, sAddr, 0, 3], sig_done
SRAM[UNLOCK, --, sAddr, 0, 1]
CTX_ARB[SRAM]
```

Example C - When the priority queue is used, both requests must use the same queue.

Original code
```
SRAM[WRITE_UNLOCK, $x1, sAddr, 0, 3], priority, ctx_swap
```

Workaround 1
```
SRAM[WRITE, $x2, sAddr, 1, 2], priority
SRAM[WRITE_UNLOCK, $x1, sAddr, 0, 1], priority, ctx_swap
```

Workaround 2
```
SRAM[WRITE, $x1, sAddr, 0, 3], priority, sig_done
SRAM[UNLOCK, --, sAddr, 0, 1], priority
CTX_ARB[SRAM]
```

Example D – correctly handling the defer optional token.

Original code
```
alu[$x1, --, b, r1]
alu[$x2, --, b, r2]
SRAM[WRITE_UNLOCK, $x1, sAddr, 0, 3], ctx_swap, defer[1]
alu[$x3, --, b, r3]
```

Workaround 1
```
alu[$x1, --, b, r1]
alu[$x2, --, b, r2]
alu[$x3, --, b, r3]
SRAM[WRITE, $x2, sAddr, 1, 2]
SRAM[WRITE_UNLOCK, $x1, sAddr, 0, 1], ctx_swap
```

Workaround 2
```
alu[$x1, --, b, r1]
alu[$x2, --, b, r2]
alu[$x3, --, b, r3]
SRAM[WRITE, $x1, sAddr, 0, 3], sig_done
SRAM[UNLOCK, --, sAddr, 0, 1]
Ctx_arb[SRAM]
```

Status: NoFix

10. PCI_OUT_INT_MASK Register Bits Not Readable

Problem: PCI Out Interrupt Mask register at 34h. The register is write only and cannot be read back.

Implication: The mask is operational, but the only way to test it is by generating I_20 and doorbell interrupts to PCI, writing 1 to this register, and then checking if subsequent interrupts to PCI are masked.

Workaround: None.

Status: Fixed

11. Spurious PCI Parity Errors

Problem: After initialization, the IXP1240 may indicate spurious PCI parity errors until at least 32 longwords have been transferred to the PCI bus using a target read mechanism.

Implication: PCI parity errors may occur in the first 32 longwords during a target read.

Workaround: The PCI bus initialization logic should include a 32 longword (or more) target read operation to each IXP1240. During this interval, ignore PCI parity errors.

Status: NoFix

12. SDRAM Arbiter

Problem: Commands are dropped in the SDRAM controller when using sdram[], optimize_mem

Chip would “hang” due to a lockout condition in the SDRAM arbiter A specific sequence of DRAM commands to different queues would eventually cause the arbiter to NOT grant any command that isn’t intended for the high priority queue.

Implication: Using the optimize_mem token on SDRAM references may freeze microengines

Workaround: Don’t use opt_mem queue with SDRAM references

Status: Fixed

13. Problem with CRC

Problem: Problem with CRC when data not is not 64-bit aligned.

CRC calculation over unaligned data from DRAM to TFIFO was not calculated correctly. Data was also not making it to the TFIFO correctly. The problems occurred with and without CRC masking, and on all burst sizes.

Implication: Requires additional instructions to align data, thus preventing ATM OC12 speed.

Workaround: None

Status: Fixed

14. SDRAM_CRC

Problem: SDRAM_CRC instruction hangs the IXP1240.

The IXP1240 appears to hang while doing sdram[read], read_crc. The problem here is that during an SDRAM chain, a non-DRAM instruction would interleave itself on the command bus right before the last SDRAM reference in the chain. This caused the chain reference to stop and the last sdram reference (which had a ctx_swap token) would not go into the ordered queue. So the chain is sitting there waiting for the last reference, which will never arrive, and the ctx_swap would never return because its instruction will never get out of the queue.

Implication: Can’t use CRC.

Workaround: None

Status: Fixed

15. SA1200 Software Reset

Problem: PCI-SA1200 Reset register does not function as specified.

Three mechanisms are used to reset the IXP1240:

- Two hardware inputs (PCI_RST# and RESET_IN#)
- One software reset (SW) via the IXP1200_RESET register.

Problems have been observed in attempting to reset the IXP1240 via the PCI_RST# input or the IXP1200_RESET register (SW reset).

In summary:

1. RESET_IN#: No reported problems with the IXP1240 hardware reset.
2. PCI_RST#: Does not work per the specification. It does not reset the device correctly and results in a hang condition during the boot sequence.
3. IXP1200_RESET register: Unpredictable results. Specifically customers initiating a soft reset by writing a value of 0xFFFFFFFF to this register results in the IXP1240 hanging during the boot sequence.

Implication: Hangs the system, typically after running for a period of time.

Workaround: None

Status: Fixed

16. Branch and Return

Problem: Certain sequence of instructions will cause dropping of an instruction following the return instruction.

A RTN or JUMP may not follow a branch whose branch decision is made at the P3 pipeline stage. These branches include all Class 3 branch instructions and branches where the decision has been postponed to the P3 stage. Please see Section 4.5.1 (Class 3 Instructions) and Section 4.5.4 (Postponed Branch Decision) of the *IXP1200 Network Processor Family Hardware Reference Manual* and Sections 3.20 and 3.29 of the *IXP1200 Network Processor Family Programmers Reference Manual* for more information.

Implication: A program execution failure will occur.

Workaround: A RTN or JUMP may not follow a P3 stage branch; program accordingly.

Status: NoFix

17. PCI Parity Error Signal

Problem: Parity Error Signal not asserting.

The Parity bit in the PCI interface is not set correctly. The Parity error indication bit in the PCI_STATUS register is correct.

Implication: Bad parity.

Workaround: Use the register parity error indication in the PCI_STATUS register.

Status: Fixed

18. Find Bit

Problem: Find Bit works on the software model but not in the actual hardware. The operation returns zero when a non-zero result is expected.

```
; Demonstration of find_bset_with_mask erratum
;
; The data register is loaded with all 1's
; Then we do a find_bset_with_mask with a mask of 0x10,
; which should find bit 4 as the first bit set.
; On hardware, the result comes back as zero indicating no bit set.

immed[data, -1]
find_bset_with_mask[0x10, data], clr_results
nop
nop
nop
nop
nop
nop
load_bset_result1[result] ; should result in 0x104
nop
nop
nop
lab#: br [lab#]
```

Implication: Wrong Find Bit.

Workaround: Do not use the FIND_BSET_WITH_MASK instruction with an immediate mask operand.

Status: Fixed

19. SDRAM_CRC Residue Register Corrupted Data

Problem: An *sdram_crc[write, ..], initiate* command, under certain conditions, may result in corrupted data in the Residue register.

Implication: During an alignment operation of >=4 bytes, an additional cycle is required for the operation to complete when compared to an alignment of < 4 bytes. If an *sdram_crc[write, ..], initiate* command is issued immediately after an alignment operation of >= 4 bytes, when the alignment is not part of a CRC chain, the additional cycle blocks the CRC Residue register write resulting in a corrupted residue value.

Workaround: Two software workarounds exist for software designs doing both *sdram[tfifo_wr,]byte-align>=4* and CRC calculations sequences. The workarounds insert one cycle between the last *sdram[tfifo_wr,]byte-align>=4* and the *sdram_crc[write],initiate* that immediately follows.

Workaround 1: This workaround is recommended due to the ease of implementation. An extra cycle between all read and write operations is effected by programming the Read-Write Turnaround time(tRWT) value in the SDRAM CSR register to at least 0x2. The tRWT value of 0x2 places two unused cycles on the SDRAM pins between every read and write, as opposed to the required one.

The simulator will detect if this problem could occur in the simulating program and will print a warning if Workaround 1 is not used.

Workaround 2: The workaround is to always insert a regular SDRAM read without any byte alignment immediately before the CRC initiation. An example of this workaround is to chain an *sdram[read]* to every *sdram[tfifo_wr],byte-align>=4* and to place these requests in any queue

except the ordered queue. This queue allocation is not a requirement, only a recommendation for possible performance implications. No additional cycles are inserted in this workaround if the instruction sequences are valid program design goals.

Status: NoFix

20. Read-Lock CAM Operations from the StrongARM* Core to SRAM

Problem: StrongARM* core instructions that use the SRAM CAM address range (0x1200 0000 - 0x127F FFFF) to perform a read-locked access can not rely on the lock attempt succeeding.

Implication: StrongARM* applications that share data structures with the Microengines cannot rely on the SRAM CAM to provide atomic access to those data structures. When a StrongARM* application issues a read_lock operation, the operation may be placed in the read_lock fail queue by the SRAM controller. The application determines whether or not the operation was placed in the read_lock fail queue by checking the value of the RLS bit of the SRAM_CSR register. To determine when a failed read_lock request is eventually moved from the read_lock rail queue to the CAM, the application polls the SRAM_CSR register until the RLRS bit is set to 1. The SRAM controller is incorrectly failing to set the RLRS bit when read_lock operation is moved from the read_lock fail queue to the SRAM CAM. Therefore, a StrongARM* application is not notified when a read_lock request is ultimately granted. This will cause locks to be placed in the CAM without application awareness.

Workaround: None. If a mutual exclusion mechanism is required, the following approaches may be used in place of the SRAM CAM:

1. Use the SRAM Bit Test & Set and Bit Test & Clear atomic operations (refer to Errata 32).
2. Create a Microengine service thread that will access the SRAM CAM on behalf of the StrongARM* application. For information on building a service thread that is callable from the StrongARM* core refer to the description of the SHRIMP API and Dispatch Library in the *IXP1200 Network Processor Family Microcode Software Reference Manual*.

Status: NoFix

21. 66 MHz Capable Bit

Problem: The IXP1240 is 66 MHz capable, but the 66 MHz Capable Bit (bit 21) in the PCI_CMD_STAT register is incorrectly fixed to zero indicating that it is not capable of 66 MHz operation.

Implication: When this bit is read, the IXP1240 incorrectly indicates that it is not capable of operating at 66 MHz as defined in the *PCI Local Bus Specification, Revision 2.2*.

Workaround: Do not use this bit for determining the maximum operating frequency of the IXP1240's PCI bus.

Status: NoFix.

Specification Changes

1. SRAM Bus Signal Timing Parameters

The maximum clock to data output valid delay (T_{val}) value for 232 MHz operation was originally specified as 4.0 ns. The new T_{val} value is 3.35 ns.

The maximum clock to control outputs valid delay (T_{ctl}) value for 232 MHz operation was originally specified as 4.0 ns. The new T_{ctl} value is 3.05 ns.

The minimum data input setup time before SCLK for pipelined SRAMs (T_{sup}) value for 232 MHz operation was originally specified as 3.75 ns. The new T_{sup} value is 3.10 ns.

2. SDRAM Bus Signal Timing Parameters

The maximum clock to data output valid delay (T_{val}) value for 232 MHz operation was originally specified as 3.4 ns. The new T_{val} value is 3.3 ns.

The maximum SDCLK to control output valid delay (T_{ctl}) value for 232 MHz operation was originally specified as 3.4 ns. The new T_{ctl} value is 2.90 ns.

T_{sup}, the minimum data input setup time before SDCLK value for 200 MHz operation was originally specified as 3.75 ns. The new T_{sup} value is 3.70 ns.

T_{sup}, the minimum data input setup time before SDCLK value for 232 MHz operation was originally specified as 3.75 ns. The new T_{sup} value is 3.70 ns.

3. FCLK AC Parameter Measurements

The parameter values for T_{high}, T_{low}, and the T_r and T_f units have changed as follows:

The minimum Clock high time (T_{high}) was originally specified as 4.5 ns. The new T_{high}value is 3.8 ns.

The minimum Clock low time (T_{low}) was originally specified as 4.5 ns. The new T_{high}value is 3.8 ns.

Both T_{high} and T_{low} have been further clarified by the statement “T_{high} and T_{low} are based on a 50% duty cycle and can vary worst case 45-55%”.

The units used for Clock rise (T_r) and Clock fall (T_f) time was originally specified as V/ns. The new unit is ns.

4. SRAM SCLK Signal AC Parameters

The minimum Cycle Time (T_{cyc}) for 232 MHz operation was originally specified as 8.6 ns. The new T_{cyc} value is 8.62 ns.

The minimum Cycle High Time (T_{high}) for 232 MHz operation was originally specified as 4.6 ns. The new T_{high} value is 4.02 ns.

The minimum Cycle Low Time (T_{low}) for 232 MHz operation was originally specified as 4.6 ns. The new T_{low} value 4.02 ns.

5. SDRAM SDCLK AC Parameters

The minimum Cycle Time (T_{cyc}) for 232 MHz operation was originally specified as 8.6 ns. The new T_{cyc} value is 8.62 ns.

The minimum Cycle High Time (T_{high}) for 232 MHz operation was originally specified as 4.6 ns. The new T_{high} value is 4.02 ns.

The minimum Cycle Low Time (T_{low}) for 232 MHz operation was originally specified as 4.6 ns. The new T_{low} value 4.02 ns.

Specification Clarifications

1. SRAM Unlocks and Write Unlocks

Issue: Documentation had indicated that performing an SRAM write_unlock on a memory location that was not locked would only result in an SRAM write, and that an SRAM unlock on a memory address that was not locked would result in no action. In actuality, unlocking an SRAM address that is not locked will result in corruption of internal CAM pointer leading to unpredictable results.

The internal CAM pointer is implemented as a 4-bit counter which indicates the number of outstanding locks that are present in the CAM. The unlock/write_unlock of an address that is not present in the CAM will incorrectly decrement this counter. This could result in corruption of CAM contents and failure of read_locks which should have been successful. Depending on the frequency of incorrect unlocks write_unlocks, and the overall program flow, this could result in data corruption, or eventual hang of one or more threads.

2. Maximum Number of Chain_Ref Instructions

Issue: Documentation has not adequately described that for SDRAM and SDRAM_CRC instructions, the maximum number of instructions that may be chained together using the chain_ref optional token is five. Chaining more than five instructions runs the risk of overflowing the SDRAM queues, resulting in dropped references.

3. DMA Receive in Big Endian Mode

Issue: The documentation does not clearly describe the PCI receive operation when Big Endian Data In is set. The fact that byte swapping occurs before the data is aligned was not clearly articulated. The clarification in Figure 2 will eliminate all ambiguities.

Figure 2. Results for DMA Receive in Big Endian Mode - Unaligned Transfer

Given the following Endian configuration on the IXP1240:								
Big Endian Data In Enable In (SA_CONTROL:15)					1			
StrongARM Big/Little Endian (CONTROL_CP15:7)					0			
And the following data on a Little Endian host:								
Byte Address	0	1	2	3	4	5	6	7
Memory Contents	00	01	02	03	04	05	06	07
A PCI DMA from the host with a start address of 0x00 (Byte Alignment = 0), results in the following on the IXP1200:								
SDRAM (Quadwords)	04050607 00010203							
ME $$xfer0, ME $$xfer1	00010203,04050607							
StrongARM (Little Endian)	03	02	01	00	07	06	05	04
A PCI DMA from the host with a start address of 0x01 (Byte Alignment = 1), results in the following on the IXP1200:								
SDRAM (Quadwords)	XX040506 07000102							
ME $$xfer0, ME $$xfer1	07000102, XX040506							
StrongARM (Little Endian)	02	01	00	07	06	05	04	XX
A PCI DMA from the host with a start address 0x02 (Byte Alignment = 2), results in the following on the IXP1200:								
SDRAM (Quadwords)	XXXX0405 06070001							
ME $$xfer0, ME $$xfer1	06070001, XXXX0405							
StrongARM (Little Endian)	01	00	07	06	05	04	XX	XX
A PCI DMA from the host with a start address of 0x03 (Byte Alignment = 3), results in the following on the IXP1200:								
SDRAM (Quadwords)	XXXXXX04 05060700							
ME $$xfer0, ME $$xfer1	05060700, XXXXXX04							
StrongARM (Little Endian)	00	07	06	05	04	XX	XX	XX

Documentation Changes

None for this version of the specification update.