Intel® IXP1200 Network Processor

Specification Update

March 2004

Notice: The IXP1200 Network Processor may contain design defects or errors known as errata. Characterized errata that may cause the IXP1200 Network Processor's behavior to deviate from published specifications are documented in this specification update.

Part Number: 278316-018

Revision History

Revision Date	Revision	Description
10/07/99	001	Beta 2 release.
1/5/00	002	Added Errata 14 through 22.
1/24/00	003	Errata revisions and corrections
5/10/00	004	Added A3 and B0 steppings. Updated Errata 2, 4, 5, 7, 8, 10, 11, 12, 15, 16, 17, 19, 21, 22. Added Errata 23 through 30. Added B0 register bit changes.
5/26/00	005	Added Errata 31. Former Specification Change 1 removed because it did not apply to A2, A3, or B0 steppings; Datasheet contained correct information. Added new Specification Changes 1 and 2.
6/7/00	006	Changed SRAM bus signal timing values in Specification Change 1.
8/14/00	007	Revised Errata 2, 3, 6, 12, 24, 25, 27, 28, and 29. Added Errata 32 through 35. Added Specification Changes 3 through 5. Deleted Documentation Change 1 and replaced it with another Documentation Change. Added Documentation Changes 2 through 4.
9/27/00	008	Incorporated Specification Changes and Documentation Changes into the v1.1 IXP1200 documents.
02/21/01	009	Updates for C0 stepping. Changed Errata 12, 25, 26, 28, 32, 34, 35 and added 36.
04/02/01	010	Added Errata 37, Specification Clarifications 1 and 2, and Documentation Changes 1 – 4.
05/09/01	011	Added Errata 38. Revised Specification Clarification 2. Added Marking Information. Corrected Errata 33 status as Fixed.
06/19/01	012	Revised Errata 3. Added Errata 39. Removed documentation changes, which have been incorporated into their respective manuals.
08/08/01	013	Added Errata 40 - 44. Added Specification Changes 1 and 2 and Specification Clarification 3.
9/27/01	014	Revised errata 37.
12/14/01	015	Updates for D0 stepping.
1/16/02	016	Additional updates for D0 stepping
11/25/03	017	Changed Errata 33 from Fixed to No Fix.
03/25/04	018	Added Errata 45.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at http://www.intel.com.

Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2004, Intel Corporation

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
IXP1200 Network Processor Datasheet	278298
IXP1200 Network Processor Family Microcode Programmer's Reference Manual	278304
IXP1200 Network Processor Family Development Tools User's Guide	278302
IXP1200 Network Processor Family Hardware Reference	278303
IXP1200 Network Processor Family Microcode Software Reference	278306
IXP1200 Network Processor Family Microcode Example Software User's Guide	278317

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 lifecycle, 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 IXP1200 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
	A2	A3	B0	C0	D0			
1	X	X	X	X	X	12	NoFix	SRAM Registers
2	X	X	X	X	X	12	NoFix	CSR Access Using PCI Memory Cycles
3	X	X				12	Fixed	Spurious SDRAM Parity Errors
4	X	X				13	Fixed	DMA Write Data Concurrency
5	X	X				13	Fixed	I_2O Counter Data Concurrency
6	X	X	X	X	X	13	NoFix	PCI_DMA Instruction
7	X	X				14	Fixed	SDRAM Memory Reference with SDRAM optimize_ mem Instruction Qualifier
8	X	X				14	Fixed	SRAM Memory Reference with SRAM optimize_ mem Instruction Qualifier
9						14	Fixed	Clock Synchronization Differences Between StrongARM* Core and IX Bus Clock
10	X	X				15	Fixed	Data Transfers on Multiword PCI Target Read and DMA Transfers
11	X	X				15	Fixed	PCI Performance
12	X	X				15	Fixed	Packet Status Length
13	X	X	X	X	X	15	NoFix	Clock Setup Time
14	X	X				16	Fixed	FDAT and FBE# Signals After Reset
15	X	X				16	Fixed	Flow Control Data
16	X	X				16	Fixed	Sixteen Quadword Transfers With SDRAM Instruction
17	X	X				16	Fixed	BIST
18	X	X				16	Fixed	Internal Arbiter
19	X	X				17	Fixed	Burst Lengths Exceeding One Page
20	X	X				17	Fixed	Cache Line Size of 0
21	X	X				17	Fixed	PCI Bus at Frequencies Less than 33 MHz
22	X	X				17	Fixed	Prepend in 32-Bit Unidirectional IX Bus Mode
23	X	X				18	Fixed	PCI CBE# Values on I/O Reads
24	X	X	X	X		18	Fixed	Deleted. Included in Errata 12.
25	X	X	X			18	Fixed	PCI Extended Capabilities Support Detection
26	X	X	X			18	Fixed	Flow-Thru SRAM Read-Modify-Write
27	X	X	X	X	X	18	NoFix	Hold Time Issues for all PCI Signals (Both Bused and Control)
28	X	X	X			19	Fixed	Non-aligned PCI DMA Transfers
29	X	X				19	Fixed	SDRAM Memory Configuration.
30	X	X	X	X	X	19	NoFix	IX Bus Contention in Shared IX Bus Mode
31	X	X	X	X	X	20	NoFix	Tval max Timing Issues When Running at 66 MHz for all PCI Signals

Errata (Continued)

No.	Steppings					Page	Status	ERRATA
	A2	A3	B0	C0	D0			
32	X	X	X			20	Fixed	Bit Test & Set and Bit Test & Clear SRAM Operations from the StrongARM* Core
33	X	X	X	X	X	23	NoFix	Read-Lock CAM Operations from the StrongARM* Core to SRAM
34	X	X	X			23	Fixed	IXP1200 PCI_INT_LAT Register Bits [11:8]
35	X	X	X			24	Fixed	PCI CSR (Control and Status Register) Access
36	X	X	X	X		28	Fixed	Inoperative PCI_OUT_INT_MASK Register
37				X	X	28	NoFix	PCI CSR Corruption
38	X	X	X	X	X	29	NoFix	SRAM[WRITE_UNLOCK,..., BURST_COUNT] Instruction
39	X	X	X	X	X	31	NoFix	Spurious PCI Parity Errors
40	X	X	X	X		31	Fixed	SDRAM Arbiter
41	X	X	X	X		32	Fixed	SA1200 Software Reset
42	X	X	X	X	X	32	NoFix	Branch and Return
43	X	X	X	X		32	Fixed	PCI Parity Error Signal
44	X	X	X	X	X	33	NoFix	Find Bit
45	X	X	X	X	X	33	NoFix	66 MHz Capable Bit

Specification Changes

No.	Steppings					Page	SPECIFICATION CHANGES
	A2	A3	B0	C0	D0		
1				X	X	34	SRAM Bus Signal Timing Parameters
2				X	X	34	SDRAM Bus Signal Timing Parameters
3				X	X	34	FCLK AC Parameter Measurements
4				X	X	34	SRAM SCLK Signal AC Parameters
5				X	X	35	SDRAM SDCLK AC Parameters

Specification Clarifications (Sheet 1 of 2)

No.	Steppings					Page	SPECIFICATION CLARIFICATIONS
	A2	A3	B0	C0	D0		
1	X	X	X	X	X	36	Command Bus Arbitration Policy

Specification Clarifications (Sheet 2 of 2)

No.	Steppings					Page	SPECIFICATION CLARIFICATIONS
	A2	A3	B0	C0	D0		
2	X	X	X	X	X	36	SRAM Unlocks and Write Unlocks
3	X	X	X	X	X	36	Maximum Number of Chain_Ref Instructions
4	X	X	X	X	X	36	DMA Receive in Big Endian Mode

Documentation Changes

No.	Document Revision	Page	DOCUMENTATION CHANGES
			None for this release of the document.

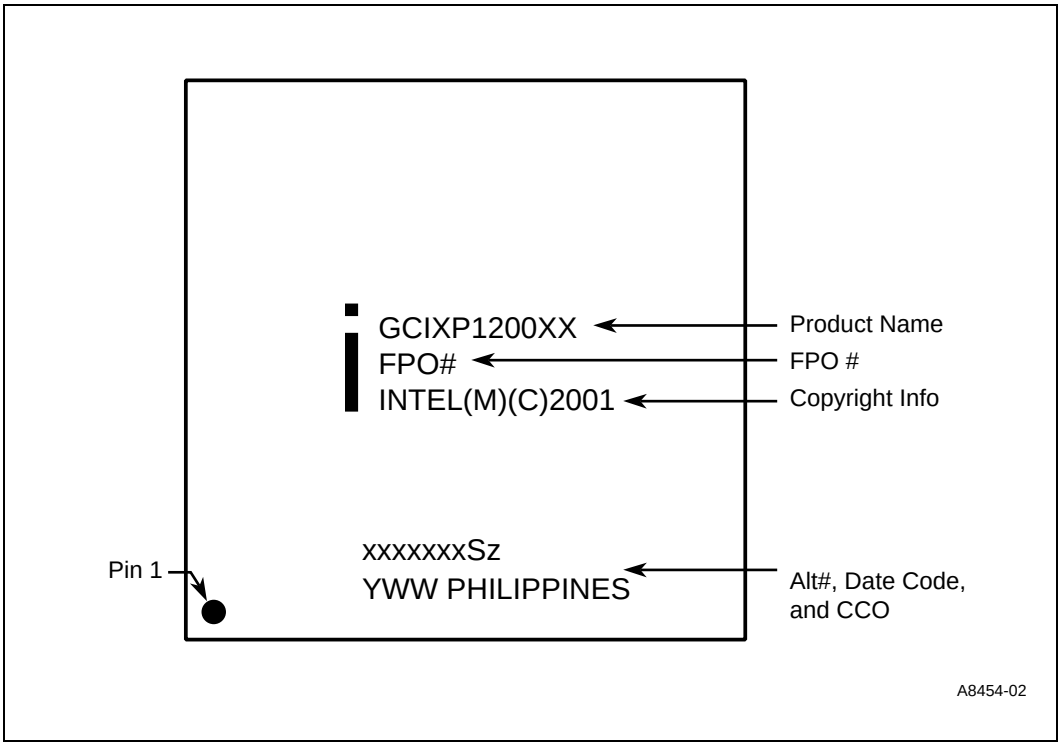
Identification Information

Markings

Product Name	Stepping	QDF Number	Marketing Part Number	Version
GCIXP1200EA	B0	NA	829314	166 MHz
GCIXP1200EB	B0	NA	829764	200 MHz
GCIXP1200FA	C0	NA	833952	166 MHz
GCIXP1200FB	C0	NA	833954	200 MHz
GCIXP1200FC	C0	NA	833956	232 MHz
GCIXP1200GC[1]	D0	QDF 310	839410	232MHz
GCIXP1200GA	D0	NA	839427	166MHz
GCIXP1200GB	D0	NA	839428	200MHz
GCIXP1200GC	D0	NA	839429	232MHz

1. Samples only.

Figure 1. Package Marking (D0 Stepping)



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. Spurious SDRAM Parity Errors

Problem: After initialization, the IXP1200 may indicate spurious SDRAM parity errors until at least 32 longwords have been transferred to the PCI bus using a target read mechanism.

Implication: Parity logic is not functional.

Workaround: No workaround.

Status: Fixed

Note: Not applicable to the B0 stepping, since SDRAM parity checking has been removed.

4. DMA Write Data Concurrency

Problem: A DMA read operation, which transfers data from the PCI bus to SDRAM indicates completion when the last data has been transferred to the DMA controller, not when it has been written to SDRAM. This causes the DMA channel to interrupt the process that initiated the DMA transfer.

Implication: If the process attempts to read the data just transferred, there is a small possibility that the read request could precede the pending PCI write request, resulting in stale data being read.

Workaround: *Workaround A*

Pad the data stream so that the last 4 longwords of data written to the SDRAM are not valid.

Workaround B

Pad the DMA descriptor table so that the last DMA transaction is a dummy read, delaying the "DMA_DONE" interrupt until after the last write occur

Workaround C

Stall the process that reads the SDRAM data for approximately 200 clock cycles to ensure that the PCI read request has been processed.

Status: Fixed

5. I_2O Counter Data Concurrency

Problem: I_2O Inbound Queue pointers are updated when data transfers from the PCI interface of the IXP1200 to the SDRAM controller interface. There may be several cycles of latency before the data is written into SDRAM by the SDRAM controller.

Implication: If the IXP1200 StrongARM* Core processor reads the I_2O Inbound Queue pointers to determine the most recently written data and then uses this value to perform a read of that address, there is a possibility that the IXP1200 StrongARM* core may issue the read request before the PCI write request has completed. This occurs because the IXP1200 StrongARM* core has higher priority to the SDRAM than does the PCI.

Workaround: All entries except the most recently posted entry may be read by the IXP1200 StrongARM* core immediately. When the most recent data is to be read, ensure that 200 IXP1200 StrongARM* core clock cycles (i.e., 100 bus clock cycles) have passed since the pointer was read.

Status: Fixed

6. 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

7. SDRAM Memory Reference with SDRAM optimize_ mem Instruction Qualifier

Problem: Due to an arbitration issue, when the SDRAM **optimize_mem** instruction qualifier is used, it is possible that an SDRAM reference may not be serviced in a timely manner. This can occur if ALL memory references are to the same bank, AND some references are qualified with **optimize_mem**, AND the non-**optimize_mem** references consistently keep the Order Queue non-empty.

Implication: The Order Queue will be continuously serviced, thus causing the same-bank Queue to be unfairly arbitrated.

Workaround: *Workaround A*

Issue references to the opposite queue regularly. For example, if a reference is issued to the Even Queue, try to schedule a successive reference to the Odd Queue. This forces the arbiter to service the Even Queue.

Workaround B

Avoid the use of the **optimize_mem** qualifier.

Status: Fixed

8. SRAM Memory Reference with SRAM optimize_ mem Instruction Qualifier

Problem: Due to an arbitration issue, when the SRAM **optimize_mem** instruction qualifier is used, if frequent references are issued to the Order Queue, then the Read Queue may be stalled for service. Also possible, but less problematic, the Priority Queue can also cause stalling of references to the Order Queues.

Implication: Results in unexpectedly long latencies in Read Queue references. It is assumed that microcode carefully manages references to the priority queue so the possibility of this occurring would be remote.

Workaround: Careful use of **optimize_mem**. The queue priority is:

1) Priority Queue
2) Read Queue, if last SRAM reference was a read
3) Order Queue
4) Read Queue

While this allows the use of **optimize_mem** to improve performance, it is possible for references issued to the Read Queue to starve references issued to the Order Queue. The microcoder should exercise careful use of the **optimize_mem** qualifier.

Status: Fixed

9. Clock Synchronization Differences Between StrongARM* Core and IX Bus Clock

Problem: Due to clock synchronization differences between the StrongARM* core and the IX Bus clock, it is possible that the IX Bus XMIT_PTR register may be incremented incorrectly during IX BUS data transfers.

Implication: Transmit operations may stop.

Workaround: None known.

Status: Fixed

10. Data Transfers on Multiword PCI Target Read and DMA Transfers

Problem: On a multiword PCI target read, or on DMA Channel Descriptor reads from SDRAM, the data read from SDRAM may be incorrect. These results have been observed under conditions of high SDRAM utilization.

Implication: A Multiword PCI Target read operation may fail with the first longword transfer being correct and the possibility of subsequent transfers returning incorrect data from SDRAM. A DMA operation by the StrongARM* core or Microengines may fail as the DMA Descriptor information may be incorrectly fetched from SDRAM causing data to be read or written at an incorrect address. It is possible that even if the Descriptor is correctly fetched on a DMA read, the read data transfer may return incorrect data.

Workaround: Limit PCI reads to only one longword per request and avoid the use of DMA operations.

Status: Fixed

11. PCI Performance

Problem: The PCI is not likely to run consistently at 66 MHz. The current estimate of the supported speed is 60 MHz. This will be confirmed once the first sample lot is characterized and if necessary, will be fixed in the next pass.

Implication: PCI devices may be limited to a maximum frequency of 60 MHz.

Workaround: Run the PCI Bus at 33 MHz or 60 MHz.

Status: Fixed

12. Packet Status Length

Problem: When the IX Bus is configured in 32-bit unidirectional mode with a packet status length of 64 bits, the status may be received in error.

Implication: Using 64-bit status in 32-bit unidirectional IX Bus mode can have unpredictable results.

Workaround: Use a packet status length of only 32 bits (set bit 26 in the RCV_REQ register to 0).

Status: Fixed

13. 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.

Status: NoFix

14. FDAT and FBE# Signals After Reset

Problem: In Shared IX Bus mode, there may be a one cycle differential between when the master and slave devices come out of reset.

Implication: For one cycle after reset, both devices may be driving the data bus high, FDAT[63:0], and the byte enable signals, FBE#[7:0].

Workaround: None

Status: Fixed

15. Flow Control Data

Problem: Flow control data is driven onto the RDYBUS[7:0] pins for three cycles. While the flow control enable signal is asserted only during the first cycle, the flow control data should be sampled on the second cycle.

Implication: If the data is not stable, incorrect data may be transferred.

Workaround: A NOP instruction should be is placed immediately prior to the flow control instruction in the RDYBUS_TEMPLATE_PROG_x registers. When a NOP instruction is executed by the ReadyBus Sequencer, the default data placed on the RDYBUS[7:0] is the flow control data. This will result in the flow control data being driven on the bus during the NOP and the flow control instruction, ensuring that the data is stable.

Status: Fixed

16. Sixteen Quadword Transfers With SDRAM Instruction

Problem: Using the SDRAM instruction with either the **r_fifo_rd** or **t_fifo_wr** command, a 16 quadword transfer is not valid.

Implication: Unpredictable results will occur.

Workaround: To do 16 quadword transfers, utilize two 8 quadword transfers.

Status: Fixed

17. BIST

Problem: In the BIST (Built-In Self Test) register, data lines [3:0] are improperly connected.

Implication: The BIST field can not be written, and, therefore, the BIST can not be performed.

Workaround: Do not attempt to perform a BIST.

Status: Fixed

18. Internal Arbiter

Problem: PCI Arbiter is arbitrating every cycle so that the grant signal is switching every clock cycle.

Implication: In cases where a master device takes more than 2 cycles to respond to a grant, if another device is always requesting the bus in a faster manner, the slower device may not be granted access to the bus.

Workaround: Disable internal arbiter and use an external arbiter.

Status: Fixed

19. Burst Lengths Exceeding One Page

Problem: When the IXP1200 is a PCI target, and a master reads data from SDRAM where the burst crosses a page boundary (require a new SDRAM row address), incorrect data may be returned if the SDRAM Unit is heavily loaded with Microengine commands.

Implication: When reading across the PCI bus with burst lengths greater than one page, the next address cannot be predicted.

Workaround: Modify software to ensure that bursts do not extend beyond page boundaries.

Status: Fixed

20. Cache Line Size of 0

Problem: Incorrect operation when PCI_CACHE_LAT_HDR_BIST register is programmed with a cache line size of zero.

Implication: When the PCI_CACHE_LAT_HDR_BIST register is programmed with a cache line size of zero, incorrect data may be returned when memory read multiple access is used.

Workaround: Do not program cache line size to 0.

Status: Fixed

21. PCI Bus at Frequencies Less than 33 MHz

Problem: Running the PCI Bus at less than 33 MHz results in incorrect or lost data transfers.

Implication: When the StrongARM* core reads the Output Buffer Queue or reads and writes to the PCI configuration registers, incorrect data may be transferred.

Workaround: Do not operate the PCI bus at frequencies less than 33 MHz

Status: Fixed

22. Prepend in 32-Bit Unidirectional IX Bus Mode

Problem: When the IX Bus is configured in 32-bit unidirectional mode and prepend is enabled, the first prepend longword is transmitted to the correct prepend element, but the second prepend longword is incorrectly transmitted to the next prepend element.

Implication: Using prepend in 32-bit unidirectional IX Bus mode has unpredictable results.

Workaround: Only use the first longword of prepend. Disregard second longword.

Status: Fixed

23. PCI CBE# Values on I/O Reads

Problem: When the StrongARM* core accesses the PCI bus through an I/O read with transactions that are not in longword lengths, the CBE# field will have incorrect values.

Implication: On data sizes less than a longword, incorrect data will be transferred.

Workaround: Limit data sizes to longwords.

Status: Fixed

24. Deleted. Included in Errata 12.

25. PCI Extended Capabilities Support Detection

Problem: Bit 20 of the PCI_CMD_STAT, which is aliased as PCI Status Register bit 4, is improperly hardwired to 0, causing improper detection of IXP1200 PCI Extended Capabilities. This is a PCI 2.2 compatibility issue for customers who may be using the IXP1200 in a PC architecture using PCI BIOS and/or PCI plug-and-play drivers.

Implication: PCI Extended Capabilities support in the IXP1200 may not be properly detected by BIOS and/or PCI plug-and-play drivers. The IXP1200 may fail Microsoft's Hardware Compatibility Tests.

Workaround: None. Even though Extended Capabilities are supported, there is no method via register reads to determine that they are supported. You may, however, use these features without confirmation.

Status: Fixed

26. Flow-Thru SRAM Read-Modify-Write

Problem: When the IXP1200 is configured with Flow-Thru SRAMs, a StrongARM* core byte write operation to SRAM will write incorrect data to the location that the operation was performed on.

Implication: Incorrect data will be in the SRAM location that the StrongARM* core operation byte write was performed on.

Workaround: When using Flow-Thru SRAMs, do not do a byte write operation form the StrongARM* core or use pipelined SRAMs.

Status: Fixed

27. 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 IXP1200 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

28. Non-aligned PCI DMA Transfers

Problem: If using non-longword aligned PCI or SDRAM addresses (address bit 1 or 0 =1) during DMA Inbound or DMA Outbound transfers, either DMA channel 1 or DMA channel 2 may experience unpredictable behavior.

Implication: Incorrect DMA operation may cause unpredictable results, including data errors.

Workaround: Do not use non-longword aligned addresses for DMA transfers.

Status: Fixed

29. SDRAM Memory Configuration.

Problem: A-stepping parts cannot correctly address 128 Mbytes of SDRAM. When SDRAM_MEMCTL0 is set for an SDRAM configuration with a Row Address Width (RAS) of 12 bits and a Column Address Width (CAS) of 10 bits (bits [7:0] = 0xCA), the MADR[13] pin, which is the high order bank select, is never asserted.

Implication: When the IXP1200 is configured for 128 Mbytes of SDRAM, accesses to the upper 64 Mbytes of SDRAM result in the lower 64 Mbytes of SDRAM being accessed.

Workaround: Do not use the 128 Mbyte SDRAM configuration with A-stepping parts.

Status: Fixed

30. 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 IXP1200 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 IXP1200s on a Shared IX Bus to inhibit initial IX Bus Owner and Ready Bus Master Mode. This ensures that no IXP1200 will be the initial IX Bus Owner and that all IXP1200s will be Ready Bus slaves. Boot software can then initialize one IXP1200 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

31. 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 IXP1200 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

32. Bit Test & Set and Bit Test & Clear SRAM Operations from the StrongARM* Core

Problem: StrongARM* core instructions that use the Bit Test & Set or the Bit Test & Clear SRAM address ranges (0x1980 0000 - 0x19FF FFFF and 0x1900 0000 - 0x197F FFFF, respectively) do not return correct data value in the SRAM_TEST_MOD register.

Implication: StrongARM* applications that share data structures with the Microengines cannot rely on the SRAM Bit Test & Set and Bit Test & Clear operations to provide atomic access to those data structures. When a StrongARM* application writes to an SRAM Bit Test & Set or Bit Test & Clear address, the SRAM controller places the original test data value in the SRAM_TEST_MOD register and modifies the contents of memory by setting or clearing the specified bits. The value in the SRAM_TEST_MOD register is read by the application to determine the pre-modified state of the specified bit(s). The bit setting and clearing works correctly, however, the SRAM controller incorrectly returns invalid data in the SRAM_TEST_MOD register. This will create a state whereby the application cannot determine if any of the specified bits were set or cleared.

Workaround: With some source code changes, code that uses the Bit Test & Set or the Bit Test & Clear operations can be modified as explained below to use SRAM Bit_set and Bit_clear operations as a communication mechanism between the Microengines and StrongARM* application. This alternate method does not require the use of the SRAM_TEST_MOD register. Note that this workaround may incur additional SRAM reads from the StrongARM* core. Two bits are used for the semaphore between the Microengines and StrongARM* core - one bit is used by the Microengines to control semaphore ownership across Microengines and a second bit is used to indicate a request from the StrongARM* application to acquire the semaphore.

In the example that follows, bit 0 is the test-and-set bit, used to arbitrate between the Microengine threads, and bit 1 indicates the StrongARM* core wants to get ownership. At a high level, the four operations are:

1. StrongARM* core takes semaphore:

   StrongARM* core sets StrongARM* core bit (0x0002). StrongARM* core polls, until word == 0x0002 (i.e., the StrongARM* core requests the semaphore, and then it polls waiting for any Microengine that has the semaphore to release it.)

2. StrongARM* core releases semaphore:

   StrongARM* core clears StrongARM* core bit (0x0002).

3. Microengine takes semaphore:

   Start:

   Microengine does test-and-set of bit 0 (0x0001)

   Switch (test-and-set result data)
   case 00: Microengine now has semaphore

case 01: Another Microengine has semaphore; goto Start.
case 10: StrongARM* core has ownership, but just set bit 0, so Microengine clears bit 0
Loop, reading word until StrongARM* core bit (bit 1) cleared
goto Start
case 11: StrongARM* core has ownership, but we didn't just set bit 0, so
Loop, reading word until StrongARM* core bit (bit 1) cleared
goto Start

4. Microengine releases semaphore:
   Microengine clears bit 0

This is illustrated in the following two blocks of code. The first is a test application written in C, and the second is the test application written for the Microengine. The basic form of each is that within a global loop, the application will:

1. Take the semaphore
2. Access locked resource
3. Release the semaphore

```
/* Filename: tands.c */
/* test-and-set code for StrongARM* core */

/*
  In word, bit 0 is used for the test and set,
           bit 1 is used for StrongARM* core ownership
*/

#include <stdio.h>

#define  WORD_ADDR 0x10000000
#define  DATA_ADDR (WORD_ADDR + 4)
#define CLEAR_ADDR (WORD_ADDR | 0x08000000)
#define   SET_ADDR (WORD_ADDR | 0x08800000)
#define TANDS_ADDR (WORD_ADDR | 0x09800000)

#define SRAM_TEST_MOD 0x38000010

#define IXP1200_REG_READ(a,val) ((val) = *(volatile UINT32 *)(a))
#define IXP1200_REG_WRITE(a,val) (*(volatile UINT32 *)(a) = (val))

#define TANDSBIT 0x0001
#define COREBIT  0x0002
int
tandsInit()
{
    IXP1200_REG_WRITE(WORD_ADDR, 0);
    IXP1200_REG_WRITE(DATA_ADDR, 0);
    return 0;
}

int
tandsLoop(int n)
{
    int i=0;
    int j;
    int xfer;

    for (; n > 0; n--) {
        while (1) {
            IXP1200_REG_WRITE(SET_ADDR, COREBIT);
            IXP1200_REG_READ(WORD_ADDR, xfer);
```

```
            if (xfer == COREBIT)
                break;
            if (xfer != (COREBIT | TANDSBIT)) {
                printf("Error: word addr = %X\n",xfer);
                return xfer;
            }
            taskDelay(1);
        } /* end while 1 */

        /* got test-and-set */

        /* get data */
       /*
       Place code here to access locked resource
       */
        /* release test-and-set */
        IXP1200_REG_WRITE(CLEAR_ADDR, COREBIT);
    }
    return i + 0x10000;
}


; Filename: tands.uc
; test-and-set code for Microengines

#define TANDSBIT 0x01
#define COREBIT  0x02

#define CASE00 0
#define CASE01 TANDSBIT
#define CASE10 COREBIT
#define CASE11 (TANDSBIT | COREBIT)


    immed[addr, 0]

start#:

    immed[$xfer, TANDSBIT]

take_sem#:
    sram[bit_wr, $xfer, addr, 0, test_and_set_bits], ctx_swap

    ; case 00, got ownership
    alu[--, $xfer, -, CASE00]
    br=0[got_it#]

    ; case 01, another Microengine has ownership, try again
    alu[--, $xfer, -, CASE01]
    br=0[take_sem#]

    ; case 10, StrongARM* core has ownership
    alu[--, $xfer, -, CASE10]
    br!=0[core_loop#]

    ; need to clear 1
    sram[bit_wr, $xfer, addr, 0, clear_bits]

core_loop#:
    ; case 11 or 10, StrongARM* core has ownership
    ; loop until 0x
    sram[read, $xfer, addr, 0, 1], ctx_swap
    alu[--, $xfer, and~, TANDSBIT]
    br!=0[core_loop#]

    ; try again
    br[take_sem#]
```

```
got_it#:


    ; got test-and-set

    ;
    ; Place code here to access locked resource
    ;

    ; release test-and-set
    immed[$xfer, TANDSBIT]
    sram[bit_wr, $xfer, addr, 0, clear_bits], ctx_swap

    br[start#]
```

Status: Fixed

33. 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

34. IXP1200 PCI_INT_LAT Register Bits [11:8]

Problem: Bits [11:8] of the IXP1200 PCI_INT_LAT register are writable from PCI. These bits correspond to PCI Interrupt Configuration Register bits [11:8]. These bits should be read-only from PCI.

Implication: The IXP1200 interrupt to PCI (INTA#) could be inadvertently disabled by BIOS or PCI plug-and-play drivers if these bits are written by PCI enumeration code to a value of 0000.

Workaround: The StrongARM* core can poll this register and rewrite correct value after BIOS or a driver writes this register.

Status: Fixed

35. PCI CSR (Control and Status Register) Access

Problem: StrongARM* core writes to any of the registers in the PCI Unit can get blocked if the write coincides in time with a PCI Master-to-CSR access. The StrongARM* core does the write operation, but the write data may not be written to the register.

Implication: The implications of this problem vary widely because of the number of PCI registers affected (a total of 82 registers). Some examples are:

1. Incorrect operation of DMA channels
2. Incorrect operation of PCI I_2O operations
3. Incorrect operation of PCI interrupts
4. Incorrect operation of Timers 1 through 4

Workaround: In general, the solution to the lost write problem is to write the register, then read it back and verify that the write was successful. This works for the 66 of the 82 PCI registers that exhibit no side effects from the write operation. The remaining 16 PCI registers have some degree of side effects from a write operation. For registers with no side effects, a read to verify the write data is unambiguous, so subsequent writes are attempted only when absolutely necessary. This is true even if there is an arbitrary time between the write and the read, which could be the case if the StrongARM* core is interrupted between these two operations.

The following table lists all CSRs that reside in the PCI unit. The table lists the type of bits in the register -- Read/Write, Read Only, Write Only, Write-1-to-Clear, Write-1-to-Set. The table also lists whether or not writes to the register have side effects. Entries that have side effects are shaded. If a register has W1C and/or W1S bits, a write has the side effect of modifying those bits. However, some registers with no W1C or W1S bits also have side effects. For each register with side effects a potential workaround, if available, is provided in the paragraphs that follow.

Register Name	Offset	Type	Side Effects
PCI Configuration Registers - Accessible by Both PCI Master and StrongARM* Core			
Vendor ID (PCI_VEN_DEV_ID)	0	RO	N
Command (PCI_CMD_STAT)	4	RW/RO	N
Status (PCI_CMD_STAT)	6	RO/W1C	Y
Rev_ID (PCI_REV_CLASS)	8	RO	N
Class Code (PCI_REV_CLASS)	9	RO	N
Cache Line (PCI_CACHE_LAT_HDR_BIST)	C	RW	N
Latency Timer (PCI_CACHE_LAT_HDR_BIST)	D	RW	N
Header Type (PCI_CACHE_LAT_HDR_BIST)	E	RO	N
BIST (PCI_CACHE_LAT_HDR_BIST)	F	RW	N
Mem Base Address (PCI_MEM_BAR)	10	RW	N
IO Base Address (PCI_IO_BAR)	14	RW	N
DRAM Base Address (PCI_DRAM_BAR)	18	RW	N
Subsys ID (PCI_SUBSYS)	2c	RW	N
Cap Pointer (PCI_CAP_PTR)	34	RW	N
Int Line (PCI_INT_LAT)	3c	RW	N
Int Pin (PCI_INT_LAT)	3d	RW	N

Register Name	Offset	Type	Side Effects
Min Gnt (PCI_INT_LAT)	3e	RW	N
Max Lat (PCI_INT_LAT)	3f	RW	N
Capability Ident (CAP_PTR_EXT)	70	RO	N
Power Mgmt Cap (PWR_MGMT)	72	RW	N
Power Mgmt CSR (PWR_MGMT)	74	RW	N
Power Mgmt Data (PWR_MGMT)	77	RW	N
PCI Memory/IO Space CSRs - Accessible by Both PCI Master and StrongARM* Core			
Interrupt Status (PCI_OUT_INT_STATUS)	30	RO	N
Interrupt Mask (PCI_OUT_INT_MASK)	34	RW	N
Mailbox0 (MAILBOX_0)	50	RW	N (See below)
Mailbox1 (MAILBOX_1)	54	RW	N
Mailbox2 (MAILBOX_2)	58	RW	N
Mailbox3 (MAILBOX_3)	5c	RW	N
Doorbell (DOORBELL)	60	W1S	Y
Doorbell Setup (DOORBELL_SETUP)	64	RW	N
StrongARM* Core Accessible Registers - Not Accessible by PCI Master			
Chan 1 Byte Count (CHAN_1_BYTE_COUNT)	80	RW	N
Chan 1 PCI Addr (CHAN_1_PCI_BAR)	84	RW	N
Chan 1 DRAM Addr (CHAN_1_DRAM_ADDR)	88	RW	N
Chan 1 Desc Pointer (CHAN_1_DESC_PTR)	8c	RW	N
Chan 1 Control (CHAN_1_CONTROL)	90	RW/W1C	Y
Chan 2 Byte Count (CHAN_2_BYTE_COUNT)	A0	RW	N
Chan 2 PCI Addr (CHAN_2_PCI_BAR)	A4	RW	N
Chan 2 DRAM Addr (CHAN_2_DRAM_ADDR)	A8	RW	N
Chan 2 Desc Pointer (CHAN_2_DESC_PTR)	Ac	RW	N
Chan 2 Control (CHAN_2_CONTROL)	B0	RW/W1C	Y
DMA Inf Mode (DMA_INF_MODE)	9c	RW	N
CSR Base Addr Mask (CSR_BASE_ADDR_MASK)	F8	RW	N
DRAM Base Addr Mask (DRAM_BASE_ADDR_MASK)	100	RW	N
I2O Inb Free List Head (I2O_INB_FLIST_HPTR)	120	RW	N
I2O Inb Post List Tail (I2O_INB_PLIST_TPTR)	124	RW	N
I2O Outb Post List Head (I2O_OUTB_PLIST_HPTR)	128	RW	N
I2O Outb Free List Tail (I2O_OUTB_FLIST_TPTR)	12c	RW	N
I2O Inb Free List Count (I2O_INB_FLIST_CNT)	130	RW	Y
I2O Outb Post List Count (I2O_OUTB_PLIST_CNT)	134	RW	Y
I2O Inb Post List Count (I2O_INB_PLIST_CNT)	138	RW	Y
SA Control (SA_CONTROL)	13c	RW/W1C	Y
PCI Address Extension (PCI_ADDR_EXT)	140	RW	N

Register Name	Offset	Type	Side Effects
Doorbell PCI Mask (DBELL_PCI_MASK)	150	RW	N
Doorbell SA Mask (DBELL_SA_MASK)	154	RW	N
IRQ_Status (IRQ_STATUS)	180	RO	N
IRQ_Raw_Status (IRQ_RAW_STATUS)	184	RO	N
IRQ_Enable (IRQ_ENABLE)	188	RO	N
IRQ_Enable_Set (IRQ_ENABLE_SET)	188	W1S	Y
IRQ_Enable_Clear (IRQ_ENABLE_CLEAR)	18c	W1C	Y
IRQ_Soft (IRQ_SOFT)	190	RW	N
FIQ_Status (FIQ_STATUS)	280	RO	N
FIQ_Raw_Status (FIQ_RAW_STATUS)	284	RO	N
FIQ_Enable (FIQ_ENABLE)	288	RO	N
FIQ_Enable_Set (FIQ_ENABLE_SET)	288	W1S	Y
FIQ_Enable_Clear (FIQ_ENABLE_CLEAR)	28c	W1C	Y
FIQ_Soft (FIQ_SOFT)	290	RW	N
Timer 1 Load (TIMER_1_LOAD)	300	RW	N
Timer 1 Value (TIMER_1_VALUE)	304	RO	N
Timer 1 Control (TIMER_1_CONTROL)	308	RW	N
Timer 1 Clear (TIMER_1_CLEAR)	30c	WO	Y
Timer 2 Load (TIMER_2_LOAD)	320	RW	N
Timer 2 Value (TIMER_2_VALUE)	324	RO	N
Timer 2 Control (TIMER_2_CONTROL)	328	RW	N
Timer 2 Clear (TIMER_2_CLEAR)	32c	WO	Y
Timer 3 Load (TIMER_3_LOAD)	340	RW	N
Timer 3 Value (TIMER_3_VALUE)	344	RO	N
Timer 3 Control (TIMER_3_CONTROL)	348	RW	N
Timer 3 Clear (TIMER_3_CLEAR)	34c	WO	Y
Timer 4 Load (TIMER_4_LOAD)	360	RW	N
Timer 4 Value (TIMER_4VALUE)	364	RO	N
Timer 4 Control (TIMER_4_CONTROL)	368	RW	N
Timer 4 Clear (TIMER_4_CLEAR)	36c	WO	Y

Workarounds depend on the register affected:

Status (PCI_CMD_STAT), StrongARM* Core Control (SA_CONTROL)
W1C bits are normally used for bits that are set by asynchronous events that must be acknowledged by the StrongARM* core, and cleared to denote that acknowledgement. The problem with the write/verify workaround is the case when the write clears the bit, and then the bit gets set again before the read happens. The compare would assume the write did not happen (because the W1C bit is set) and then write again, thereby losing the second event. Doing the write/verify operation atomically (meaning with interrupts disabled) shortens the interval of vulnerability, but does not guarantee perfect operation.

An alternative solution is to use some RW field that can be written to different values without causing some other problem. At the beginning of the write/verify operation, read the register and note the value of the chosen RW field. Create the write data with a different value for the RW field, and the W1C bits set to 1. Test that the write happened or not by comparing only on the value of the RW field, masking the W1C bit position(s).

For example, the Status Register (PCI_CMD_STAT) W1C bits are for PCI bus error conditions. Normally, errors are rare so writing multiple times has a low risk of losing an error bit. If the application does not use I/O space, then the I/O space enable bit [0] could be used.

In the StrongARM* core Control Register (SA_CONTROL), the W1C bit is for the PCI SERR asserted error. This is normally used for severe errors, so should not occur frequently.

Doorbell (DOORBELL)
The Doorbell (DOORBELL) register is W1S. It is similar in intent to the use of W1C bits. Host and the StrongARM* core use the bits to request and acknowledge service. Write/verify could lose an interrupt as described above. Doing the write/verify atomically will be safe. The Host can not clear the bit again until it sees it as set, which it won't be able to do fast enough.

DMA Channel Control (CHAN_1_CONTROL, CHAN_2_CONTROL)
Each Channel Control Register has several status bits that are W1C, but more important is the side effect of starting the channel by writing a 1 to the Enable bit. If the write/verify is done atomically, the channel will not be able to complete quickly enough such that such that it completes (which clears the Enable bit) before the read. An alternative is to use the method described under Status register, and use one of the RW fields as an indicator for the write.

IRQ_ENABLE_SET, IRQ_ENABLE_CLEAR, FIQ_ENABLE_SET, FIQ_ENABLE_CLEAR
These are W1S and W1C aliases for IRQ_Enable and FIQ_Enable. Doing the write/verify atomically will be safe because the only thing that can change them is another thread of operation in the StrongARM* core. Blocking interrupts during the write/verify will prevent that thread from running. Note that the write is to the xxx_Enable_Set or xxx_Enable_Clear, and the read is from xxx_Enable. Test only the bit(s) being changed to verify the write operation. For example, if the operation is to Set bits, use the data of the write to mask the read bits, and verify that all of them are '1'.

TIMER_1_CLEAR, TIMER_2_CLEAR, TIMER_3_CLEAR, TIMER_4_CLEAR
These registers are WO and each has the side effect of clearing the Timer interrupt. Because the interrupt is set asynchronously (by the Timer), it is possible that a Timer interrupt could be lost. This is the same problem described above under Status register. Doing the write/verify atomically will minimize the probability of that case. There are no RW fields in this register to use the second proposed workaround for Status register.

I2O Inbound Free List Count (I2O_INB_FLIST_CNT), I_2O Outbound Post List Count (I2O_OUTB_PLIST_CNT), I_2O Inbound Post List Count (I2O_INB_PLIST_CNT)
The side effect of a write to these registers is to modify the count by incrementing or decrementing (based on which register), and discarding the write data. The counts are also modified by PCI bus operations to the I_2O addresses. Therefore, it is not possible to read the value after the write to see if it is correct, since a PCI operation might have modified it between the read/write/read.

MAILBOX_0, MAILBOX_1, MAILBOX_2, MAILBOX_3
The Mailbox registers are RW and have no side effects. The assumption made is that ownership of each mailbox by either the StrongARM* core or PCI Host is either statically defined in the application, or coordinated by some other means such as Doorbells.

Status: Fixed

36. Inoperative PCI_OUT_INT_MASK Register

Problem: The IXP1200 Network Processor PCI_OUT_INT_MASK register is not functional. This register is intended to prevent the IXP1200 from asserting **pci_irq_l**. A write to the PCI_OUT_INT_MASK register does not change the value of the register. A read of the PCI_OUT_INT_MASK register returns the value of the PCI_CAP_PTR register.

Implication: The Outbound Post List Interrupt and Doorbell Interrupt cannot be disabled.

Workaround: None.

Status: Fixed

37. 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.

Note: Only applicable to the C0 stepping.

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:

Workaround: **1.**

1.1 Insert the following piece of code into the header file **ixp1200eb.h** located in the IXP1200 Developer's Workbench software installation in the directory

Boardsupport\VxWorks\IXP1200EB.

```
#ifdef PCI_WORKAROUND
#define AMBA_TIMER_WRITE(reg, data) ({\
__asm__ __volatile (""); \
(*((volatile UINT32 *)(reg)) = (data)); \
((void)*(volatile UINT32 *)(IXP1200_PCI_ADDR_EXT)); \
__asm__ __volatile (""); })
#endif
```

1.2. Define the compiler directive **PCI_WORKAROUN**D, either in your project build settings or as a #define in the header file. Without this directive, the compiler may reorder the instructions.

1.3. Rebuild the VxWorks image. Refer to the README file entitled *Building the VxWorks BSP*, for directions on how to build the image.

Workaround: **2.**

In a multiprocess system, however, the PCI CSR write/read combination may be interceded by an interrupt which can cause a context switch; the new process may initiate PCI memory writes thus possibly corrupting PCI CSR Registers. Workaround 1 above isn't affected since it executes with interrupts disabled.

However, there are two modules: ixp1200HPC.c and ixp1200IntrCtl.c, which use the macro IXP1200_PCI_REG_WRITE().Two functions in ixp1200IntrCtl.c: ixp1200IntLvlEnable(), ixp1200IntLvDisable(), for example, are called by the OS functions intEnable(), intDisable(). These can be called by user functions; if they are, the code needs to be modified to do intLock(), PCI_REG_WRITE(), PCI_REG_READ(), intUnlock() sequences to avoid problems. Similarly with code in ixp1200HPC.c.

2.1 The workaround is to precede all references to the IXP1200_PCI_REG_WRITE() macro with an intlock() function call and follow such references with an intUnlock() function call:

```
intLock();
PCI_REG_WRITE();
PCI_REG_READ();
intUnlock();
```

2.2 Optionally, a more global workaround which avoids coding each PCI_REG_READ() occurrence is to call taskSwitchHookAdd() with a pointer to a function that does a PCI_REG_READ(). This function is called on every context switch and virtually guarantees that the PCI CSR is not left in an indeterminate state. Care and good programming practice must be observed in calling complex functions when interrupts and scheduling are disabled.

Status: NoFix

38. 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

39. Spurious PCI Parity Errors

Problem: After initialization, the IXP1200 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 IXP1200. During this interval, ignore PCI parity errors.

Status: NoFix

40. 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 sequence of SDRAM 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

41. SA1200 Software Reset

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

Three mechanisms are used to reset the IXP1200:

- 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 IXP1200 via the PCI_RST# input or the IXP1200_RESET register (SW reset).

In summary:

1. RESET_IN#: No reported problems with the IXP1200 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 IXP1200 hanging during the boot sequence.

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

Workaround: None

Status: Fixed

42. Branch and Return

Problem: RTN or JUMP instructions following a Class 3 branch instruction will cause dropping of the 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* 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

43. 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

44. 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: NoFix

45. 66 MHz Capable Bit

Problem: The IXP1200 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 IXP1200 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 IXP1200'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. Command Bus Arbitration Policy

Issue: Clarification of the Command Bus Arbiter operation.

Information contained in the *IXP1200 Network Processor Family Hardware Reference Manual* incorrectly described the arbiter operation. The Command Bus Arbiter arbitrates between the six Microengines to determine which Command FIFO will be serviced next. The Command Bus Arbiter uses the following information to determine which Command FIFO is to be serviced.

- A priority scheme between the types of commands.
- A rotating priority scheme between the Microengines.
- A back-pressure signal from the functional units.

Note: Details on optimizing Command Bus operation will be detailed in an application note to be released shortly.

2. 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.

3. Maximum Number of Chain_Ref Instructions

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.

4. 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 IXP1200:

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 revision of the specification update.