Intel® 80321 I/O Processor

Specification Update

May 12, 2003

Notice: The Intel® 80321 I/O processor (80321) may contain design defects or errors known as errata that may cause the product to deviate from published specifications. Current characterized errata are documented in this specification update.

Document Number: 273519-010

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. 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 products are not intended for use in medical, life saving, life sustaining applications.

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

Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The Intel® 80321 I/O processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

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.

Contents

This page intentionally left blank.

Revision History

Date	Version	Description
May 13, 2003	010	• Added Specification Clarifications 7.
May 2003	009	• Revised Non-Core Errata 3. • Added Non-Core Errata 17. • Added Specification Clarifications 6. • Added Document Change 12.
November 2002	008	• Removed C-0 Step column from Core Errata, Non-Core Errata, Specification Changes, Specification Clarifications, Die Details, Device ID Registers. Not planned to be implemented. • Updated Status indicators in Core Errata and Non-Core Errata. • Added Specification Change 3. • Added Specification Clarifications 4 and 5.
September 2002	007	• Removed Specification Clarification 3. It does not apply to the Intel® 80321 I/O processor.
August 2002	006	• Added B-1 Stepping Column. • Added Core Errata 19 through 21. • Reworded Non-Core Errata 4. • Added two Diagrams at end of Non-Core Errata 9. • Added Non-Core Errata 15 and 16. • Reworded Specification Change 1. • Added Specification Clarification 3. • Revised Die Details and Device ID Registers.
June, 2002	005	• Added Non-Core Errata 13 and 14. • Added Specification Clarifications 1 and 2. • Document Changes 1 and 4 through 11 are now incorporated in the latest Design Guide revision.
April 11, 2002	004	• Added Core Errata 18 and Specification Changes 1 and 2. • Added notes to Core Erratum 13 and 14 and edited workaround for Non-Core Errata 10. • Deleted previous Core Errata 12. • Added *Intel® 80321 I/O Processor Advance Information Datasheet* to related/affected documents • Changed steppings for the fixes of Non-Core Errata 2, 3, 4, 7, 8, 9, 10, 11, and 12. • Added Die Details and Device ID Registers for B-0 stepping.
March 26, 2002	003	• Added Core Errata 12 and 14 (rearranged other numbering to fit new order). • Added Non-Core Errata 8 through 12. • Revised Non-Core Errata 2 through 7. • Rearranged Document Change order to reflect sequential page flow. • Added Document Change 2. • Renumbered old Document Change 10 to 3.
March 14, 2002	002	• Added Non-Core Errata 7. • Added Document Changes 1 through 3.
February 2002	001	Initial Release.

Preface

This document is an update to the specifications contained in the Affected Documents/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.

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.

Affected Documents/Related Documents

Title	Order
Intel® 80321 I/O Processor Developer's Manual	273517
Intel® 80321 I/O Processor Advance Information Datasheet	273518
Intel® 80321 I/O Processor Design Guide	273520

Nomenclature

Errata are design defects or errors. These may cause the Product Name's behavior to deviate from published specifications. Hardware and software designed to be used with any given stepping must assume that all errata documented for that stepping are present on all devices.

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 Product Name 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.

PlanFix: This erratum may be fixed in a future stepping of the product.

Fixed: This erratum has been previously fixed.

NoFix: There are no plans to fix this erratum.

Row

│ Change bar to left of table row indicates this erratum is either new or modified from the previous version of the document.

Core Errata

No.	Steppings			Page	Status	Errata
	A-0	B-0	B-1			
1	X	X	X	14	NoFix	Boundary Scan Is Not Fully Compliant to the IEEE 1149.1 Specification
2	X	X	X	14	NoFix	Drain Is Not Flushed Correctly when Stalled in the Pipeline
3	X	X	X	15	NoFix	Undefined Data Processing-'like' Instructions are Interpreted as an MSR Instruction
4	X	X	X	15	NoFix	Debug Unit Synchronization with the TXRXCTRL Register
5	X	X	X	15	NoFix	Extra Circuitry Is Not JTAG Boundary Scan Compliant
6	X	X	X	16	NoFix	Incorrect Decode of Unindexed Mode, Using Addressing Mode 5, Can Corrupt Protected Registers
7	X	X	X	16	NoFix	Load Immediately Following a DMM Flush Entry is Also Flushed
8	X	X	X	16	NoFix	Trace Buffer Does Not Operate Below 1.3 V
9	X	X	X	16	NoFix	Data Cache Unit Can Stall for a Single Cycle
10	X	X	X	17	NoFix	Aborted Store that Hits the Data Cache May Mark Writeback Data As Dirty
11	X			18	Fixed	CP15 Data Cache Unlock Command Can Cause Unlock in User Mode or when Flushed from the Pipe in Supervisor Mode
12	X			19	Fixed	Store to Cacheable Memory, Interrupted by an Exception, May Inadvertently Write to Memory
13	X			21	Fixed	Data Cache Dirty Bits May be Corrupted when a Line Invalidate is Followed Immediately by a Store
14	X			22	Fixed	Data cache dirty bits may be Corrupted when a Bus Error on a Cache Line Fill is Followed Immediately by a Store
15	X	X	X	23	NoFix	Performance Monitor Unit Event 0x1 Can Be Incremented Erroneously by Unrelated Events
16	X	X	X	23	NoFix	In Special Debug State, Back-to-Back Memory Operations Where the First Instruction Aborts May Cause a Hang
17	X			24	Fixed	Instruction Memory Management Unit Address Translation is Turned Off for the First Fetch After Exiting Special Debug State
18	X			25	Fixed	Data cache dirty bits may be corrupted when a store to cacheable memory occurs during a tag replacement for a different cache line
19	X	X	X	28	NoFix	Accesses to the CP15 ID register with opcode2 > 0b001 returns unpredictable values
20	X	X	X	28	NoFix	Disabling and re-enabling the MMU can hang the core or cause it to execute the wrong code
21	X	X	X	29	NoFix	Updating the JTAG parallel register requires an extra TCK rising edge

Non-Core Errata

No.	Steppings			Page	Status	Errata
	A-0	B-0	B-1			
1	X	X	X	30	NoFix	The SSP TXD Does Not Retain the Value of the Last Bit Transferred
2	X	X	X	30	NoFix	The ATU Returns Invalid Data for the DWORD that Target Aborted from the MCU when Using 32-Bit Memory, ECC Enabled and in PCI Mode
3	X	X	X	30	NoFix	PBI Issue When Using 16-bit PBI Transactions in PCI Mode
4	X	X	X	31	NoFix	All-zero Result Buffer” for the AAU is not Implemented
5	X	X	X	31	NoFix	MCU Pointers are Incorrect following a Restoration from a Power Fail
6	X	X	X	31	NoFix	PMU Does Not Account for when the Arbiter Deasserts GNT# One Cycle before FRAME#
7	X			32	Fixed	SCKE[1:0] Contention During a Power Failure
8	X			32	Fixed	Core Write of ECC Error Not Setting Bit #23/#24 Correctly in ECAR
9	X			33	Fixed	Improper Power Fail Sequence During a Power Failure
10	X			35	Fixed	PLL Unable to Lock at Reset
11	X	X	X	36	NoFix	Lost Data During Bursts of Large Number of Partials with 32-bit ECC Memory
12	X			36	Fixed	P_RST# to PCI-X Initialization Pattern Hold Time (Tprh)
13	X	X	X	37	NoFix	The MTTR1 (Core Multi-Transaction Timer) is not operating due to improper behavior of the core internal bus request signal (REQ#)
14	X	X	X	37	NoFix	The MCU supports a page size of 2 Kbytes for 64-bit mode
15	X	X		38	Fixed	A logic error in the Memory Controller Unit (MCU) incorrectly reports an ECC Error on memory writes. This error does not corrupt memory contents or data. There are two different conditions that exacerbate the issue.
16	X	X		42	Fixed	Intel® 80321 I/O Processor/PCI-X Bridge Unexpected Split Completion Error
17	X	X		43	NoFix	Vih Minimum Input High Voltage (Vih) level for the PCI pins

Specification Changes

No.	Steppings			Page	Specification Changes
	A-0	B-0	B-1		
1	X	X	X	44	DDR V_{CC} and DDR V_{REF} minimum specifications need to be changed on the A-0 and B-0/B-1 steppings
2	X	X	X	44	DDR SDRAM signal timing change, T_{VA3}
3	X	X	X	45	P_BMI (AE23) added to B-0/B-1 Steppings

Specification Clarifications

No.	Steppings			Page	Status	Specification Clarifications
	A-0	B-0	B-1			
1	X	X	X	47	NoFix	The Intel® 80321 I/O processor is compliant with the PCI Local Bus Specification, Revision 2.2 but it is not compliant with PCI Local Bus Specification, Revision 2.3
2	X	X	X	47	NoFix	Modifications to the Hot-Debug procedure are necessary for the Intel® 80321 I/O processor when flat memory mapping is not used (Virtual Address = Physical Address)
3	X	X	X	47	Doc	Removed. Does not apply to the Intel® 80321 I/O processor.
4	X	X	X	48	Doc	BAR0 Configuration When Using the Messaging Unit (MU)
5	X	X	X	48	Doc	Reading Unpopulated SDRAM Memory Banks
6	X	X	X	48	Doc	32-bit Writes-to-Unaligned 64-bit Addresses, are Promoted to 64-bit Aligned Writes
7	X	X	X	49	Doc	In-order Delivery not guaranteed for data blocks described by a single DMA descriptor

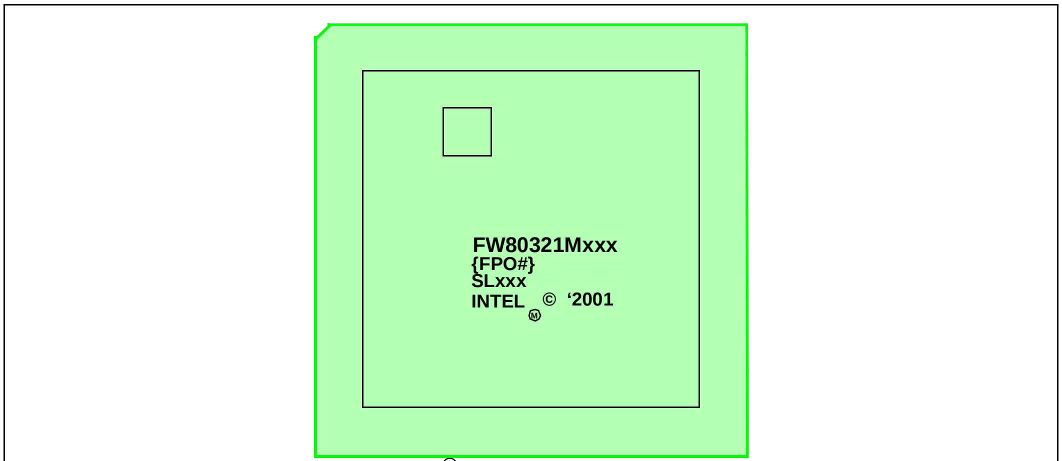
Documentation Changes

No.	Document Revision	Page	Status	Documentation Changes
1	273520-002	49	In Guide	Table 4 Page 18 second row has incorrect data
2	273518-001	49	Doc	Table 9 (Sheet 3 of 5), page 31 and Table 10 (Sheet 5 of 5), page 38 have incorrect data
3	273518-001	49	Doc	Table 10 (Sheet 2 of 5), page 35 has incorrect data
4	273520-001	49	In Guide	Section 6.2.2 on page 37 has incorrect data
5	273520-002	50	In Guide	Figure 14 on page 40 has missing text
6	273520-002	50	In Guide	Table 18, page 61 has missing data and incorrect data
7	273520-002	51	In Guide	Section 7.6.1 page 75 has incorrect data
8	273520-002	51	In Guide	Section 7.6.1 page 76 has incorrect data
9	273520-002	51	In Guide	Section 7.6.1 page 77 has incorrect data
10	273520-002	51	In Guide	Section 7.6.3 page 78 has missing data and incorrect data
11	273520-002	52	In Guide	Section 14.1 page 113 has missing data
12	273518-001	52	Doc	Channel Control Register; Channel Enable, page 248

Identification Information

Markings

Topside Markings



Intel® 80321 Processor

Die Details

Stepping	Part Number	QDF (Q)/ Specification Number (SL)	Voltage (V)	Intel® 80321 I/O Processor Speed (MHz)	Notes
A-0	FW80321M400	Q237	3.3	400	Samples - limited testing
A-0	FW80321M600	Q238	3.3	600	Samples - limited testing
A-0	FW80321M400	Q359	3.3	400	Samples
A-0	FW80321M600	Q284	3.3	600	Samples
A-0	FW80321M400	Q285	3.3	400	General Samples
A-0	FW80321M600	Q286	3.3	600	General Samples
A-0	FW80321M400	SL5PC	3.3	400	Production Material
A-0	FW80321M600	SL5PD	3.3	600	Production Material
B-0	FW80321M400	Q377	3.3	400	Samples - limited testing
B-0	FW80321M600	Q379	3.3	600	Samples - limited testing
B-0	FW80321M400	Q378	3.3	400	General Samples
B-0	FW80321M600	Q380	3.3	600	General Samples
B-0	FW80321M400	Q385	3.3	400	General Samples
B-0	FW80321M600	Q386	3.3	600	General Samples
B-0	FW80321M400	SL69L	3.3	400	Production Material
B-0	FW80321M600	SL69M	3.3	600	Production Material
B-1	FW80321M400	Q464	3.3	400	Samples - limited testing
B-1	FW80321M600	Q465	3.3	600	Samples - limited testing
B-1	FW80321M400	Q466	3.3	400	General Samples
B-1	FW80321M600	Q467	3.3	600	General Samples
B-1	FW80321M400	SL6R2	3.3	400	Production Material
B-1	FW80321M600	SL6R3	3.3	600	Production Material

Device ID Registers

Device and Stepping	Processor Device ID (CP15, Register0 - opcode_2=0)	ATU Device ID (ATUDID)	ATU Revision ID (ATURID)	JTAG Device ID
A-0 (400 MHz)	0x69052420	0x0318	0x00	0x09266013
A-0 (600 MHz)	0x69052430	0x0319	0x00	0x09267013
B-0 (400 MHz)	0x69052C21	0x0318	0x01	0x19266013
B-0 (600 MHz)	0x69052C31	0x0319	0x01	0x19267013
B-1 (400 MHz)	0x69052C22	0x0318	0x02	0x29266013
B-1 (600 MHz)	0x69052C32	0x0319	0x02	0x29267013

Core Errata

1. Boundary Scan Is Not Fully Compliant to the IEEE 1149.1 Specification

Problem: The IEEE Standard 1149.1 specifies the boundary scan logic to support two main goals:

1. To allow the interconnections between the various components to be tested, test data can be shifted into all the boundary-scan register cells associated with component output pins and loaded in parallel through the component interconnections, into those cells associated with inputs pins; and
2. To allow the components on the board to be tested, the boundary-scan register can be used as a means of isolating on-chip system logic from stimuli received from surrounding components, while an internal self-test is performed. Alternatively, when the boundary-scan register is suitably designed, it can permit a limited slow-speed static test of the on-chip system logic, since it allows delivery of test data to the component and examination of the test results. (IEEE std. 1149.1-1990, page 1-5)

The Intel® Xscale™ core does not support the second goal, because it does not support the optional INTEST or RUBIST instructions. The Intel® Xscale™ core is not required to provide these instructions, however, since it doesn't, this makes the following statement practically invalid.

The IEEE std. 1149.1 description of the SAMPLE/PRELOAD instruction states that, “When the SAMPLE/PRELOAD instruction is selected, the state of all signals flowing through system pins (input or output) shall be loaded into the boundary scan register on the rising edge of the TCK in the Capture-DR controller state.” (Page 7-8).

The boundary scan cells of the Intel® Xscale™ core bi-directional pads, do not capture the data driven from the on-chip system logic to the pins, when these pads are acting as outputs. This would only be useful when trying to capture the data driven from the on-chip logic, during normal operation of the assembled board. However, the Intel® Xscale™ core does not allow single stepping of its clocks. Thus, even when the Intel® Xscale™ core did provide the compliant boundary scan cell, it would be extremely difficult (or impossible) to synch the boundary scan logic with the state of the on-chip logic. Therefore, this feature of the boundary scan cells is not useful. This has NO effect on the ability to determine the integrity of the interconnections on boards, which is what the Intel® Xscale™ core boundary scan logic was designed to support.

Workaround: No workaround.

Status: NoFix.

2. Drain Is Not Flushed Correctly when Stalled in the Pipeline

Problem: In a load followed by a drain scenario, the load table walks and then gets a precise data abort. The core fetches the address for the abort handler, but in the same cycle does not flush the drain.

Implication: Not a functional problem, but may effect performance.

Workaround: No workaround.

Status: NoFix.

3. Undefined Data Processing-‘like’ Instructions are Interpreted as an MSR Instruction

Problem: The instruction decode allows undefined opcodes, which look similar to the MSR (Move to Status register from an ARM register) instruction, to be interpreted as an MSR instruction. The mis-decoded MSR instruction also adds a SUBNV PC,0x4 to the instruction flow.

Workaround: Do not use undefined opcodes of this form:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	0	0	0	1	0	-	1	0	-	-	-	-	-	-	-	-	-	-	-	-	0	0	1	0	-	-	-	-
-	-	-	-	0	0	0	1	0	-	1	0	-	-	-	-	-	-	-	-	-	-	-	-	0	1	0	0	-	-	-	-
-	-	-	-	0	0	0	1	0	-	1	0	-	-	-	-	-	-	-	-	-	-	-	-	0	1	1	0	-	-	-	-

Status: NoFix.

4. Debug Unit Synchronization with the TXRXCTRL Register

Problem: The RX bit in the TXRXCTRL (TX/RX Control) register comes from the JTAG clock domain to the core clock domain, and several cycles are needed for the register in the core clock domain to update. During this time, a debugger, which is running a fast JTAG clock relative to the core clock, may read the bit before it updates in the register, thus reading the old value.

Workaround: The JTAG clock should be slower than the core clock.

Status: NoFix.

5. Extra Circuitry Is Not JTAG Boundary Scan Compliant

Problem: The IEEE 1149.1 (JTAG) specification states that, “when the HIGHZ instruction is selected, all system logic outputs.... shall immediately be placed in an inactive drive state”. The JTAG unit on the core creates an internal ‘float’ signal, which is driven to the I/O pads. This signal is derived from the HIGHZ instruction; however, the HIGHZ instruction gets flopped by a rising edge of TCK first, before it is able to ‘float’ the pads. This is in violation of the JTAG specification, specifically the term “immediately”. It is possible for TCK to stop after the HIGHZ instruction is loaded and thus the pads may never ‘float’.

Workaround: Do not stop the JTAG clock (TCK).

Status: NoFix.

6. Incorrect Decode of Unindexed Mode, Using Addressing Mode 5, Can Corrupt Protected Registers

Problem: The instruction decoder incorrectly decodes the valid combination of P=0, U=1 and W=0, when using unindexed mode in addressing mode 5 (load and store coprocessor). In this case, the LDC or STC should produce consecutive address loads or stores, with no base update until the coprocessor signals that it has received enough data. Instead, the instruction gets separated into an LDR/STR and a CP access.

The LDR/STR gets decoded as a post-index address, updating the base register. Due to the decoding as post-index, the ‘option’ bits, normally reserved for the coprocessor in unindexed mode, become the 8-bit offset value used in the base register update calculation.

The implication is, that protected registers can be corrupted. This errata can cause the corruption of FIQ registers, R13-R14, in user and system modes when the LDC instruction is executed using unindexed addressing mode. It can also cause the corruption of FIQ registers, R8-R12, in any mode, when the LDC instruction is executed using unindexed addressing. The R13 register in debug mode may also be corrupted during an LDC in any mode. In the case of STC, only Rn is corrupted.

Unexpected memory accesses can also occur. In the case of an LDC, any memory location may be accessed, since the FIQ registers may be improperly used as the base register. In the case of an STC, the memory word located at Rn+4 is corrupted. This is the memory location immediately following the locations which should be modified by STC unindexed.

Workaround: Do not use unindexed addressing in addressing mode 5 – Load and Store Coprocessor.

Status: NoFix.

7. Load Immediately Following a DMM Flush Entry is Also Flushed

Problem: A load that immediately follows a data memory management (DMM) flush entry command, that also hits the data TLB, is also flushed. Therefore, the instruction immediately following the flush, is also flushed from the data TLB.

Workaround: All flush entry commands to the data TLB must be followed by two NOPs. The first ensures the erratum is not encountered, and the second ensures the speed path is not hit.

Status: NoFix.

8. Trace Buffer Does Not Operate Below 1.3 V

Problem: The trace buffer within the debug unit is not guaranteed to operate, due to voltage sensitivity, when the core voltage supply is below 1.3 V.

Workaround: Make sure the voltage is above 1.3 V during debug.

Status: NoFix.

9. Data Cache Unit Can Stall for a Single Cycle

Problem: When the data cache unit retries an operation that is in the pending buffer, a single cycle stall occurs.

Workaround: No workaround. This is a performance issue only.

Status: NoFix.

10. Aborted Store that Hits the Data Cache May Mark Writeback Data As Dirty

Problem: When there is an aborted store that hits clean data in the data cache (data in an aligned four word range, that has not been modified from the core, since it was last loaded in from memory or cleaned), the data in the array is not modified (the store is blocked), but the dirty bit is set.

When the line is then aged out of the data cache or explicitly cleaned, the data in that four word range is evicted to external memory, even though it has never been changed. In normal operation, this is nothing more than an extra store on the bus, that writes the same data to memory as is already there.

Here is the boundary condition where this might be visible:

1. a cache line is loaded into the cache at address A
2. another master externally modifies address A
3. a core store instruction attempts to modify A, hits the cache, aborts because of MMU permissions, and is backed out of the cache. That line should not be marked dirty, but because of this errata is marked as dirty.
4. the cache line at A then ages out or is explicitly cleaned. The original data from location A is evicted to external memory, overwriting the data written by the external master.

This only happens when software is allowing an external master to modify memory, that is, writeback or write-allocate in the page tables, and depending on the fact that the data is not 'dirty' in the cache, to preclude the cached version from overwriting the external memory version. When there are any semaphores or any other handshaking to prevent collisions on shared memory, this should not be a problem.

Workaround: For this shared memory region, mark it as write-through memory in the page table. This prevents the data from ever being written out as dirty.

Status: NoFix.

11. CP15 Data Cache Unlock Command Can Cause Unlock in User Mode or when Flushed from the Pipe in Supervisor Mode

Problem: Correct behavior for the “unlock data cache” command (MCR p15, 0, Rd, c9, c2, 1), issued in user mode, is to generate an invalid instruction exception and not affect the state of the cache. Instead, the exception is generated, but the cache is unlocked anyway. In this case, the illegal instruction event is generated. When the OS does not attempt to recover from illegal instructions, this erratum is no issue.

A secondary effect of this erratum, iMs that even in a privileged mode, when an instruction should execute, the data cache unlock hardware activity can occur while the actual MCR instruction is still in the execution pipeline. When an interrupt or some other event causes the instruction to be flushed from the pipe, the hardware activity may still occur. It is likely that the instruction executes shortly after returning from the handler, so unlocking the cache early probably is not an issue, unless the event handler code makes an assumption about data cache locking.

Workaround: When user mode code is well controlled, the OS can detect that user code did the illegal operation and report the error for software debugging purposes. Then re-code the user application and try again.

When code deliberately trying to crash the machine is a concern, here are possible ways to recover when the cache is unlocked in user mode:

1. When data cache locking is used with the knowledge of the OS, to keep local copies of external data for performance, it is possible, for the OS to fix things. The OS detects that the invalid instruction fault was caused by a coprocessor 15 operation, cleans out the cache, and relocks the data that was supposed to be locked. The offending user application should be terminated.
2. When data cache locking is used as SRAM as part of the data cache (allocated with the data cache line operation, rather than loading in existing memory), it is difficult for the machine to recover. When any of the SRAM locations are evicted, external memory at a random location can be corrupted. One solution is to immediately turn off the data cache at the beginning of the invalid instruction event handler to avoid evictions in the cache. The SRAM contents can then be copied to scratch memory, the cache cleaned and invalidated, and the SRAM reallocated and locked. At that point, the contents can be copied back in and operation can continue.

When early unlocking of the data cache in privileged modes is a concern, the following software workarounds exist:

1. Disable interrupts before unlocking the data cache, and ensure that no abort conditions exist around the unlock code. Imprecise external data aborts and parity errors are still potential issues, when software attempts to recover from these situations.
2. Do not make assumptions about data cache lock state in the event handlers. When the event handler code does not require access to locked data cache regions, the early unlock is transparent, as the unlock MCR is executed once the handler is finished.

Status: Fixed.

12. Store to Cacheable Memory, Interrupted by an Exception, May Inadvertently Write to Memory

Problem: In some cases, when a store to a cacheable region of memory occurs simultaneously with an exception, the store is not properly cancelled and may update memory. This may occur even when the store did not have permission to write to that memory region, and no data abort is recorded at the time memory is incorrectly modified. It is important that developers review this erratum and assess the impact to their specific application to ensure that no potential exists for data corruption.

This combination of events required for the erratum to occur:

1. A memory operation occurs that requires a cache line fill, which in-turn causes a cache line eviction. For example, a load or prefetch.
2. A few cycles later, an exception event occurs. The following exception events cause the erratum:
   a. interrupt
   b. prefetch abort
   c. instruction breakpoint
   d. invalid opcode fault
   e. imprecise data abort

   A precise data abort may not cause the erratum.
3. The instruction which is being flushed by the exception event is a store to write-back memory, and the store address coincides with the cache line that is being evicted to make room for the cache line fill.
4. Returning data on the bus causes a data cache stall at a specific cycle.

Result:

The store writes to memory when it should not. When the store is to write-back memory, the cache line is evicted and external memory is incorrectly updated as when that store had occurred. The exception return points to the store instruction.

When the store is to write-through memory, the store instruction is properly flushed and external memory is not updated.

Memory access permissions are not checked before the store updates the cache. This means the store may change memory without permission and without taking an abort.

When the store did a pre- or post-index update on its address, the register file correctly flushes the update. This means that when the store is executed on return from the exception, correct data is stored into memory at the correct address.

Example Manifestations:

The following examples explain how the erratum may manifest itself. The examples are intended to help developers assess the impact upon their applications.

Note: This is not an exhaustive list, there may be other examples depending upon how specific the architecture of the application code is.

1. Copy on Write -

   In a multi-tasking OS, a process has read access to a shared memory region, but not write access. Upon a write, the data abort handler creates a physical copy of that region and allow write access to that new region.

   In the failing case, the shared read-only region can be updated incorrectly and without a data abort. Eventually, when the original process is returned to, a data abort happens, but the problem is that the memory has already been corrupted. Linux supports Copy on Write and when activated, can experience the erratum. Other OSs may also be affected.

2. Flags to memory shared with interrupt handler –

   When there is a flag used to communicate between a process and the interrupt handler, and the store to set the flag is being set by the store that is interrupted, the interrupt handler may see the flag set, do the required work, and clear the flag. Upon return to the

   interrupted process, the store instruction would be executed and the flag set. The interrupted process would incorrectly assume that the interrupt handler had not yet seen that flag and serviced it. This flag error can also occur between two task switched processes.

3. Debug -

   Instruction breakpoints and prefetch aborts on a store can cause the erratum. When a user is single stepping through code, stops with an instruction breakpoint on a store, and then checks external memory for the pre-store value of the address the store is about to write to, the debugger may see the post store value.

Workaround: The global workaround for the erratum would be to disable write-back cache within the application. This ensures the erratum does not manifest itself. However, it is understood that this can be cumbersome in performance-sensitive applications. As an alternative, developers may simply prevent the conditions from occurring simultaneously which would cause the erratum to be manifested. By preventing these conditions, one effectively has implemented a workaround.

1. Disable write-back cache. This can apply to Linux-based applications due to the fact that Linux utilizes copy-on-write.
2. Enable write-back cache and insure the code does not allow the simultaneous occurrence of the conditions required to manifest the erratum.

Here are some workaround options for the specific examples listed above:

1. Copy on Write -

   This section is concerned with permissions as applied to first Linux* and then to an embedded real-time controller. In Linux, Copy on Write, is a memory allocation technique used to provide a performance boost. The erratum can be avoided by setting the caching policy for a copy-on-write read-protected page to write-through. There is no memory access penalties because the memory is read-only. When a write occurs and a write-able copy is made, that memory cache policy should be set to write-back. Without this change, errors may occur.

   When concerned with a real-time controller, the purpose for using permissions usually is to detect a misbehaving pointer or for quickly detecting runaway code. This erratum only applies to the misbehaving pointer problem, which when missed on one misuse is most likely found on the next, resulting in the software problem being found and fixed.

2. Flags to memory shared with interrupt handler -

   The impact of this problem most likely only involves a handful of sensitive variables for an embedded controller. That is because most variables are used within a task. Only some variables, which need to be passed from one task to another task, is affected. A classic example variable would be a semaphore flag. These variables can be protected in one of three ways:

   a. Locate the variable in locked cache or non-evicting mini-cache. This is the ideal solution for time critical variables such as those passed between an interrupt handler and a background processing task. A non-evicting mini-cache region can be created by setting up a mini-cache memory region and then limiting all access to that region to a 2K aligned address range. Because accesses are not outside the cache boundary, an eviction never occurs and the memory behaves as though it were on-board RAM.

   b. Surround the write of these variables with interrupt disable/enable commands. This is often already done because manipulation of these variables exists in ‘critical code’ regions where an interrupt between any of the instructions in the critical code region may cause an error condition. Modifying a semaphore is a typical critical code example.

   c. Use the atomic instructions, SWP or SWPB. Therefore, limiting the manipulation of sensitive variables to these instructions avoids the erratum.

3. Debug / Breakpoints -

   Most debug handlers flush the cache when a break occurs, which reduces the potential for a cache eviction and reduces the potential for a system to be affected by the erratum. So there are few times this erratum should be seen in a debug session, and misinterpreting an apparent bug may be avoided by knowing where this erratum might occur.

Status: Fixed.

13. Data Cache Dirty Bits May be Corrupted when a Line Invalidate is Followed Immediately by a Store

Problem: The dirty bits in the data cache can be corrupted by an ‘Invalidate Data Cache Line’ command to address “A” immediately followed by any store to address “B” where address “B” and address “A” are to the same cache set (bits 9:5 of the two addresses are the same).

This can lead to clean or invalid data being marked dirty in the cache, or dirty data in the cache being marked clean. Possible outcomes:

1. Invalid data being marked dirty can lead to unpredictable four word stores to an unpredictable address in memory.
2. Valid, but clean write-back data, being marked dirty can lead to unnecessary evictions to external memory.
3. Valid dirty data being marked clean can lead to data corruption. External memory is not updated upon cache line replacement.

Workaround: Do not perform a store operation immediately following an ‘Invalidate Data Cache Line’ command. ‘Invalidate Data Cache Line’ is a supervisor mode instruction. Placing a NOP between these two operations avoids the erratum.

Note: For more details, see the workaround information for “Data cache dirty bits may be corrupted when a store to cacheable memory occurs during a tag replacement for a different cache line” on page 25.

Status: Fixed.

14. Data cache dirty bits may be Corrupted when a Bus Error on a Cache Line Fill is Followed Immediately by a Store

Problem: This erratum is very similar to erratum #14. The problem is the same, but the root cause is different. When certain types of bus errors occur, and a store to the cache occurs near the time when the error is reported, the dirty bits in the array in the same set as the store may be corrupted. The bus errors that can cause this erratum are a multi-bit ECC error on a returning cache line fill or assertion of the external ABORT pin on a returning cache line fill.

This can lead to clean or invalid data being marked dirty in the cache, or dirty data in the cache being marked clean. Possible outcomes:

1. Invalid data being marked dirty can lead to unpredictable four word stores to an unpredictable address in memory.
2. Valid, but clean write-back data, being marked dirty can lead to unnecessary evictions to external memory.
3. Valid dirty write-back data being marked clean can lead to data corruption. External memory will not be updated upon cache line replacement.

Bus errors reported on any store, or on loads that are smaller than 32 bytes do not cause this bug.

Workaround: Set up the page tables such that unimplemented regions of memory do not have valid page table entries. A precise data abort occurs on accesses to those regions and no bus error is seen.

Mark any region of memory that has valid memory locations and unimplemented memory locations within one page as non-cacheable, to avoid cache line fills. This precludes the erratum. An example of this might be a memory mapped register region.

Mark any region of memory that may generate bus aborts that cannot be prevented by appropriate page table entries as non-cacheable.

When there is some reason that code must do a cache line fill that has a chance of creating a bus abort, then avoid store instructions in supervisor mode while potentially aborting cache line fills are pending. The drain write buffer CP15 command issued right after the cache line fill can be used to stall the core during these transactions.

For any triggering abort that cannot be avoided by page table permissions or cacheability (such as multi-bit ECC error on a cache line fill), there is no known workaround. Bus aborts should, in general, be avoided during normal operation.

Note: For more details, see the workaround information for “Data cache dirty bits may be corrupted when a store to cacheable memory occurs during a tag replacement for a different cache line” on page 25.

Status: Fixed.

15. Performance Monitor Unit Event 0x1 Can Be Incremented Erroneously by Unrelated Events

Problem: Event 0x1 in the performance monitor unit (PMU) can be used to count cycles in which the instruction cache cannot deliver an instruction. The only cycles counted should be those due to an instruction cache miss or an instruction TLB miss. The following unrelated events in the core, also causes the corresponding count to increment when event number 0x1 is being monitored:

1. Any architectural event (e.g. IRQ, data abort)
2. MSR instructions which alter the CPSR control bits
3. Some branch instructions, including indirect branches and those mispredicted by the BTB
4. CP15 mcr instructions to registers 7, 8, 9, or 10 which involve the instruction cache or the instruction TLB.

Each of the items above may cause the performance monitoring count to increment several times. The resulting performance monitoring count may be higher than expected when the above items occur, but never lower.

Workaround: There is no way to obtain the correct number of cycles stalled due to instruction cache misses and instruction TLB misses. Extra counts due to branch instructions mispredicted by the BTB, may be one component of the unwanted count that can be filtered out. The number of mispredicted branches can also be monitored using performance monitoring event 0x6 during the same time period as event 0x1. The mispredicted branch number can then be subtracted from the instruction cache stall number generated by the performance monitor to get a value closer to the correct one. Note that this only addresses counts contributed by branches that the BTB is able to predict. All the items listed above still affect the count. Depending on the nature of the code being monitored, this workaround may have limited value.

Status: NoFix.

16. In Special Debug State, Back-to-Back Memory Operations Where the First Instruction Aborts May Cause a Hang

Problem: When back-to-back memory operations occur in the Special Debug State (SDS, used by ICE and Debug vendors) and the first memory operation gets a precise data abort, the first memory operation is correctly cancelled and no abort occurs. However, depending on the timing, the second memory operation may not work correctly. The data cache may internally cancel the second operation, but the register file may have scoreboarded registers for that second memory operation.

The effect is that the core may hang (due to a permanently scoreboarded register) or that a store operation may be incorrectly cancelled.

Workaround: In Special Debug State, any memory operation that may cause a precise data abort should be followed by a write-buffer drain operation. This precludes further memory operations from being in the pipe when the abort occurs. Load Multiple/Store Multiple that may cause precise data aborts should not be used.

Status: NoFix.

17. Instruction Memory Management Unit Address Translation is Turned Off for the First Fetch After Exiting Special Debug State

Problem: When the processor enters SDS (Special Debug State), the Instruction Memory Management Unit is disabled. This means that the instruction TLB is no longer accessed and physical-to-virtual address translation no longer occurs. The first code executed after exiting SDS mode should cause an instruction TLB access and execute from the physical memory specified in the appropriate page table entry.

In certain timing cases, this may not occur. The physical address of the code that executes next is the same as the virtual address, without regard to the page tables. This can cause the wrong code to execute.

Workaround: Depending on the nature of the special debug routines, one of the following code sequences may be used to avoid this issue. These routines occur just before the CPSR restore instruction used to exit SDS.

1. SDS code is cacheable (external or in mini-icache):

```
    .align 5
    mov r13, addr
@ Invalidate I-cache line, causes stall until all inst fetches complete
    mcr p15, 0, r13, c7, c5, 1
CPSR restore
```

2. SDS code is non-cacheable (external or mini-icache):

```
    .align 5
    b   next_cache_line
previous_cache_line:
    CPSR restore

    .align 5
next_cache_line:
    b   previous_cache_line
```

3. SDS code is either non-cacheable or cacheable (but cannot be located in mini-icache):

This workaround assumes that code is in a region mapped 1 to 1. This ensures the data access does not actually get remapped to another region of memory which may be faster than the region containing the debug handler. Note that the clean data cache line command used below is not necessary when the location “addr” is not used to hold some state for the use of the debug handler.

```
    mov rx, addr@ use some addr that is within the dbg hdlr
@ clean and invalidate D-cache line, make sure that addr is not in D-cache
    mcr p15, 0, rx, c7, c6, 1
    mcr p15, 0, rx, c7, c10, 1
    b   next_cache_line

    .align 5   @ make sure we start fetching next cache line
               @ before loading from it.
next_cache_line:
    ldr rx, [rx]@ load from addr
    mov rx, rx @ data dependency, waits for load to complete
    CPSR restore
```

Status: Fixed.

18. Data cache dirty bits may be corrupted when a store to cacheable memory occurs during a tag replacement for a different cache line

Problem: The dirty bits in the data cache may become corrupted when a store to cacheable memory hits an outstanding cache line fill and is presented to the cache during a tag replacement for a different cache line. There is no reasonable way to avoid these events lining up in normal code. This erratum is caused by a circuit race condition, and may be sensitive to voltage, temperature, or noise.

This can lead to clean or invalid data being marked dirty in the cache, or dirty data in the cache being marked clean. Possible outcomes include:

- Invalid data being marked dirty can lead to unpredictable four-word stores to an unpredictable address in memory.
- Valid, but clean write-back data being marked dirty can lead to unnecessary evictions to external memory.
- Valid dirty data being marked clean can lead to data corruption. External memory will not be updated upon cache line replacement.

Example 1. Scenario of Data-Cache Dirty Bit Becoming Corrupted

1. A store, of 0xaaaa to address 0x1000, hits write-back memory in the cache and marks it dirty.
2. Clean and invalidate commands are issued to address 0x1000, correctly pushing the data out of the cache to memory.

   At this point a clean invalid copy of the 0xaaaa data exists in the cache.
3. A store, of 0xbbbb to address 0x1000, is cleaned and invalidated and memory is updated correctly.
4. The erratum occurs and the original, 0xaaaa copy of the data is incorrectly marked “dirty.” It is then evicted, overwriting the 0xbbbb data in memory.

In an application, this may appear to be general memory corruption. It also may seem that the second store did not occur correctly, when in fact the second store worked and old data overwrote it.

Workaround: In cache lines in the array that are marked “write-through,” the dirty bits are ignored. The workaround for the bug is to make sure all cache lines that may be replaced in the data cache are labeled “write-through” at all times (whether the lines are valid or not). Three things are necessary for this to be the case:

- All cacheable memory regions must be configured as write-through
- In the reset handler, when the data cache is enabled, the write-through bit of all cache lines must be set. This is accomplished with a code routine (supplied in Figure 1) that disables interrupts, enables the data cache and MMU, and fills the cache with write-through data.

  Even when these lines are invalidated, the write-through bit will stay set.
- No “allocate line in the data cache,” CP15 mcr operations can occur unless locking memory into the data cache to create SRAM. (See “Data-cache locking” note, below.) The line-allocate command will place a line in the cache with an unpredictable write-through state. This is normally not a problem, but violates the workaround for this errata.

  This operation is used to clean dirty data out of the data cache. Because the cache is operating in write-through mode, cleaning the cache is not necessary and can be invalidated.

Data-cache locking: The line-allocate command can be used to allocate lines locked into the data cache as SRAM. This does not violate this errata’s workaround as those lines are locked in and will not be replaced. However, at any point where the cache is unlocked, those locked lines are now available for replacement.

At that point, the cache must be loaded up with write-through data using a routine similar to the routine used at reset. Simply replace the Enable-DCU command, in that routine, with an Unlock-DCU command and you will have the needed code sequence.

Figure 1. Code for Enabling the Cache and Marking the Whole Cache Write-Through (Sheet 1 of 2)

```
@#
@# This code enables the data cache and sets all the write-through bits in the
@# data cache. The MMU must already be on.
@#
@# parameters:
@#
@# r0   contains a 32 byte aligned pointer to a 32 kbyte write-through
@#  region of memory. This memory will NOT be modified, but it
@#  must allow read access.
@#
@# registers:
@#
@# r0   used, not changed
@# r1   modified
@# r2   modified
@# r3   modified
@# r4   modified
@#
@# cpsr (interrupts and CCs) modified
@# DCU enabled
@# data cache invalidated without cleaning

@# disable interrupts
    mrs     r4,cpsr
    orr     r1,r4,#0xc0
    msr     cpsr,r1

@# enable DCU
    mcr     p15, 0, r1, c7, c10, 4@# drain the dcu

    mrc     p15, 0, r1, c1, c0, 0
    orr     r1, r1, #0x4   @# enable dcu
    mcr     p15, 0, r1, c1, c0, 0
@# cpwait
    mrc     p15,0,r1,c2,c0,0
    mov     r1,r1
    sub     pc,pc,#4
```

Figure 1. Code for Enabling the Cache and Marking the Whole Cache Write-Through (Sheet 2 of 2)

```
@# global invalidate data-cache
    mcr    p15, 0, r1, c7, c6, 0  @# invalidate dcu


@# fill main cache with write-through lines
    mov    r1,#1024
    mov    r3,r0
loop1:
    ldr    r2,[r3],#32
    subs   r1,r1,#1
    bne    loop1

@# global invalidate dcu
@#  this will allow us to use the same 32k region in the
@#  mini-cache section
    mcr    p15, 0, r1, c7, c6, 0  @# invalidate dcu


@# fill mini-cache with write-through lines (2kbytes, 64 lines)

    @# enable test feature to force all fills to the mini-cache
    mov    r1,#0x8
    mcr    p15, 0, r1, c15, c15, 3

    mov    r1,#64
    mov    r3,r0
loop2:
    ldr    r2,[r3],#32
    subs   r1,r1,#1
    bne    loop2

@# disable test feature to force all fills to the mini-cache
    mov    r1,#0x0
    mcr    p15, 0, r1, c15, c15, 3

@# global invalidate data-cache
    mcr    p15, 0, r1, c7, c6, 0  @# invalidate dcu


  @## restore cpsr
    msr    cpsr,r4
```

Status: Fixed.

19. Accesses to the CP15 ID register with opcode2 > 0b001 returns unpredictable values

Problem: The *ARM Architecture Reference Manual* (ARM DDI 0100E) states the following in chapter B-2, section 2.3:

> “If an <opcode2> value corresponding to an unimplemented or reserved ID register is encountered, the System Control processor returns the value of the main ID register.
>
> ID registers other than the main ID register are defined so that when implemented, their value cannot be equal to that of the main ID register. Software can therefore determine whether they exist by reading both the main ID register and the desired register and comparing their values. If the two values are not equal, the desired register exists.”

The Intel® Xscale™ core does not implement any CP15 ID code registers other than the Main ID register (opcode2 = 0b000) and the Cache Type register (opcode2 = 0b001). When any of the unimplemented registers are accessed by software (e.g., mrc p15, 0, r3, c15, c15, 2), the value of the Main ID register should be returned. Instead, an unpredictable value is returned.

Workaround: No workaround.

Status: NoFix.

20. Disabling and re-enabling the MMU can hang the core or cause it to execute the wrong code

Problem: When the MMU is disabled, via the CP15 control register (CP15, CR1, opcode_2 = 0, bit 0), after being enabled, certain timing cases can cause the processor to hang. In addition to this, re-enabling the MMU after disabling it can cause the processor to fetch and execute code from the wrong physical address. To avoid these issues, the code sequence below needs to be used whenever disabling the MMU or re-enabling it afterwards.

Workaround: The following code sequence can be used to disable and/or re-enable the MMU safely. The alignment of the mcr instruction that disables or re-enables the MMU needs to be controlled carefully, so that it resides in the first word of an instruction cache line.

```
@ The following code sequence takes r0 as a parameter. The value of r0 is written
@ to the CP15 control register to either enable or disable the MMU.

mcr     p15, 0, r0, c10, c4, 1@ unlock I-TLB
mcr     p15, 0, r0, c8, c5, 0@ invalidate I-TLB

    mrc p15, 0, r0, c2, c0, 0@ CPWAIT
    mov r0, r0
    sub pc, pc, #4

b       1f             @ branch to aligned code

.align 5
1:
mcr     p15, 0, r0, c1, c0, 0@ enable/disable MMU, caches

    mrc p15, 0, r0, c2, c0, 0@ CPWAIT
    mov r0, r0
    sub pc, pc, #4
```

Status: NoFix.

21. Updating the JTAG parallel register requires an extra TCK rising edge

Problem: IEEE 1149.1 states that the effects of updating all parallel JTAG registers should be seen on the falling edge of TCK in the Update-DR state. The Intel® Xscale™ core parallel JTAG registers incorrectly require an extra TCK rising edge to make the update visible. Therefore, operations like hold-reset, JTAG break, and vector traps require either an extra TCK cycle by going to Run-Test-Idle or by cycling through the state machine again in order to trigger the expected hardware behavior.

Workaround: When the JTAG interface is polled continuously, this erratum has no effect. When not, an extra TCK cycle can be caused by going to Run-Test-Idle after writing a parallel JTAG register.

Status: NoFix.

Non-Core Errata

1. The SSP TXD Does Not Retain the Value of the Last Bit Transferred

Problem: Both SSP and SPI protocols require that the TXD line retain the value of the last bit transferred. Per the SSP and SPI protocol: “At the end of transfer, TXD retains the value of the last bit sent (bit 0) through the next idle period. When the SSP port is disabled or reset, TXD is forced to zero.” However, TXD only retains the value for approximately one clock cycle and then goes to zero. It is verified that the SSP remains enabled so it is not due to the SSP being disabled.

Workaround: No workaround.

Status: NoFix.

2. The ATU Returns Invalid Data for the DWORD that Target Aborted from the MCU when Using 32-Bit Memory, ECC Enabled and in PCI Mode

The external PCI bus requests a read through the ATU to the MCU, starting at the high DWORD. Remember the MCU is in 32-bit mode. The ATU requests multiple DWORDs since it pre-fetches, but starts at the high DWORD address. The MCU issues two DWORDs. First the high, followed by the low and then a Target Abort, so the DWORD count is two. When the ATU returns the data to the external PCI agent (in PCI Mode ONLY), the logic ONLY disconnects on 64-byte QWORD boundaries. Recall the ATU DWORD count is at two. When the external PCI device returns to get data, the ATU returns the first DWORD and SHOULD disconnect, because it does not have enough data to get to the next QWORD boundary. It does not do this. Instead, it returns invalid data in the high DWORD of the second QWORD (data from a previous fetch) and the transaction is corrupted.

This issue occurs when all of the following conditions exist in the MCU:

1. 32-bit memory
2. ECC is enabled
3. The PCI bus is in PCI mode

Workaround: Use 64-bit Memory, PCI-X Mode or ECC disabled.

Status: NoFix.

3. PBI Issue When Using 16-bit PBI Transactions in PCI Mode

Problem: Under certain conditions, in bound burst and non-burst reads andwrites from the PCI bus to the PBI would appear as two writes on the PBI. However, the byte enables are not asserted for the second write.

This happens when:

1. 80321 is in PCI mode.
2. Another PCI master is attempting to access the PBI behind the 80321.
3. 16-bit mode on PBI.

Workaround: The BE# signals can be used in combination with the PCE#. The BE# prevents the second CE# from being recognized by the Flash. See the Intel® 80321 Evaluation Board FAB D schematic for a circuit design to correct this issue.

Status: NoFix.

4. All-zero Result Buffer” for the AAU is not Implemented

Problem: The “All-zero result buffer” is a new feature that was going to be added to the 80321 I/O processor. This feature is currently not implemented and therefore, cannot be used.

Workaround: No workaround. The “All-zero result buffer” is not planned to be implemented.

Status: NoFix.

5. MCU Pointers are Incorrect following a Restoration from a Power Fail

Problem: This issue occurs when:

1. There is a power failure (not during power management or normal shutdown).
2. When power is restored, the internal MCU pointers to the SDRAM may not be correct.
3. When a read from SDRAM (prior to doing a write to SDRAM) is the first MCU operation done after the power is restored, the MCU pointers may be incorrect and can be reading the wrong data.
4. However, when a write to SDRAM is the first MCU operation done after the power is restored, then the pointers are correct and everything works properly.

Workaround: Following restoration of power after a power failure, ensure that the first MCU operation done is a write to SDRAM.

Status: NoFix.

6. PMU Does Not Account for when the Arbiter Deasserts GNT# One Cycle before FRAME#

Problem: One of the countable PMU events is bus acquisition latency for the ATU. There is a condition where the acquire counter is not stopped even though the ATU starts a transaction. When the arbiter deasserts GNT# in PCI-X mode, the requestor can still start a transaction for one cycle (due to allowed pipelining). In this situation, the PMU does not properly detect the FRAME# as the ATU and continues running the counter.

Workaround: No workaround.

Status: NoFix.

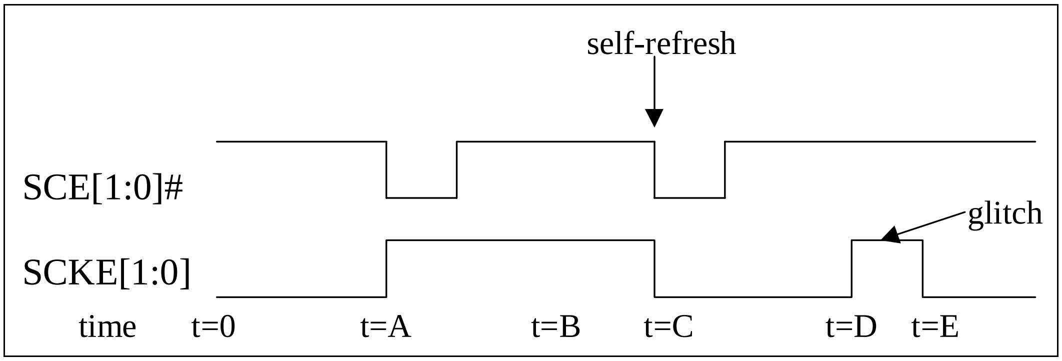
7. SCKE[1:0] Contention During a Power Failure

Problem: During power failure, the Memory Controller Unit (MCU) issues a sequence of commands to the DDR memory that puts the memory into self-refresh mode. The data is preserved in the DDR memory as long as the memory subsystem is powered with a battery source and SCKE[1:0] is held low.

While the memory is in self-refresh mode, the SCKE[1:0] signal remains low. The problem is that SCKE[1:0] is incorrectly driven high for *n* M_CLKs, soon after it was driven low. This causes the DDR memory to exit self-refresh mode, therefore data integrity cannot be guaranteed.

SCKE[0] and SCKE[1] operate independently of each other with SCKE[1] trailing SCKE[0] by one M_CLK. Here is the sequence of events as described specifically for SCKE[0]:

1. At t=0, SCKE[0] starts out low.
2. At t=A, the first initialization command goes to the MCU by software control and SCKE[0] is driven to '1'.
3. From t=A to t=B is normal operation.
4. From t=B to t=C a power fail event occurs. The MCU completes its routine at t=C, at which time SCKE[0] is driven to '0' and the DDR memory is put into self-refresh mode.
5. However, soon after t=C (about 2 - 4 M_CLKs later), the SCKE[0] “glitches” (SCKE[0] is driven to '1'). The length of time from t=D to t=E varies depending on the ability of the 80321 to issue the internal reset. Simulation has shown the width of the “glitch” to be 0 to 2 M_CLKs. However, this may vary with internal bus activity and the ability of the 80321 to service the reset when the power fail sequence is done. This “glitch” on the SCKE[0] signal causes the DDR memory to exit self-refresh mode.
6. At t=E, the falling edge of the SCKE[1:0] “glitch” is synchronous with the assertion of M_RST#.



Workaround: See the workaround for Item 9.

Status: Fixed.

8. Core Write of ECC Error Not Setting Bit #23/#24 Correctly in ECAR

Problem: In the MCU ECAR MMRs (0xFFFF E540 and 0xFFFF E544), bit 10 of the column address is incorrectly mapped to bit 23/24 (32-bit/64-bit mode, respectively) of the IB address. Recall that column address bit 10 is RESERVED for the AUTO-PRECHARGE command during a read/write. Column address bit 9 should have been routed to this location, not column address bit 10. The effect is bit 23 in 32-bit mode, or bit 24 in 64-bit mode, is incorrect (“stuck” at 0) – the actual value of the bit cannot be known.

Workaround: A stuck bit means that the customer does not know when the bit is a 1 or 0. The workaround requires a scrub of both locations; once assuming the stuck bit is at a 1 and a second time assuming the bit is at a 0.

Status: Fixed.

9. Improper Power Fail Sequence During a Power Failure

Problem: During power fail, the Memory Controller Unit (MCU) issues a sequence of commands to the DDR memory that issues an auto-refresh command followed by a self-refresh command that puts the memory into self-refresh mode. The data is preserved in the DDR memory as long as the memory subsystem is powered with a battery source and SCKE[1:0] is held low.

The minimum clock count between the auto-refresh command and the self-refresh command is eight clock cycles, to allow enough time for the DDR state machine to complete the auto-refresh command in the power fail sequence. The problem is that the number of clock cycles between these two commands is only four clock cycles. Therefore, the self-refresh command is not recognized by the DDR state machine, because the DDR state machine is still executing the auto-refresh command. Since the self-refresh command is missed by the DDR state machine, the DDR never enters self-refresh mode. Therefore, DDR memory data integrity cannot be guaranteed.

Workaround: Convert the auto-refresh command, during the power fail sequence, to a self-refresh command and to hold the SCKE[1:0] signal low until the POR# pin is deasserted. The circuit for this workaround is shown below. This circuit also provides the workaround for Item 7. PLD equations for this circuit are available upon request.

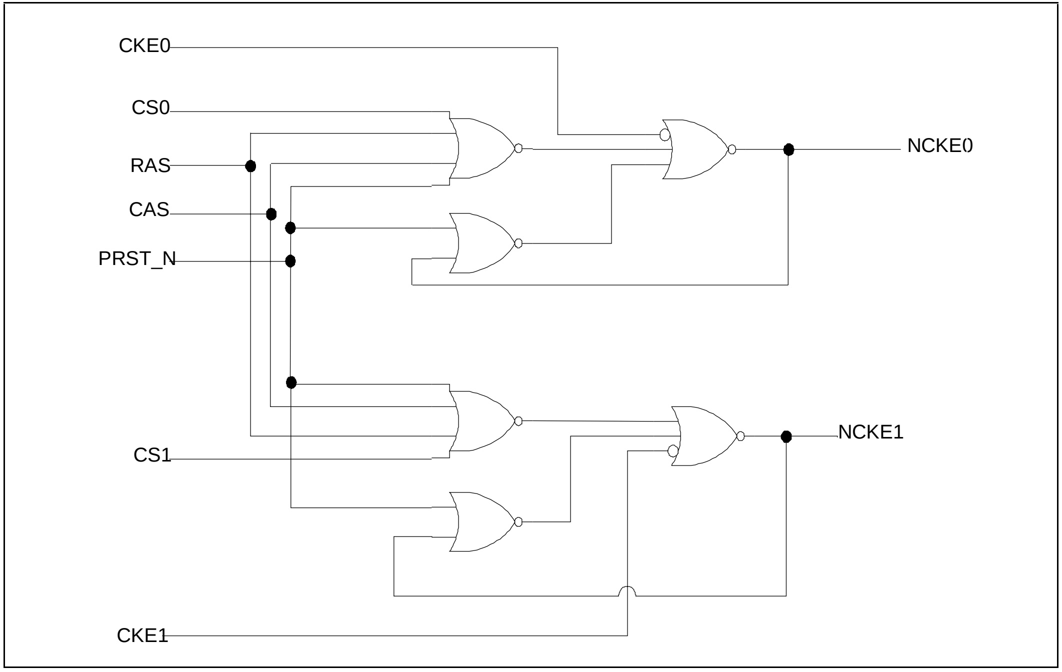


Figure 2. Power Fail State Machine

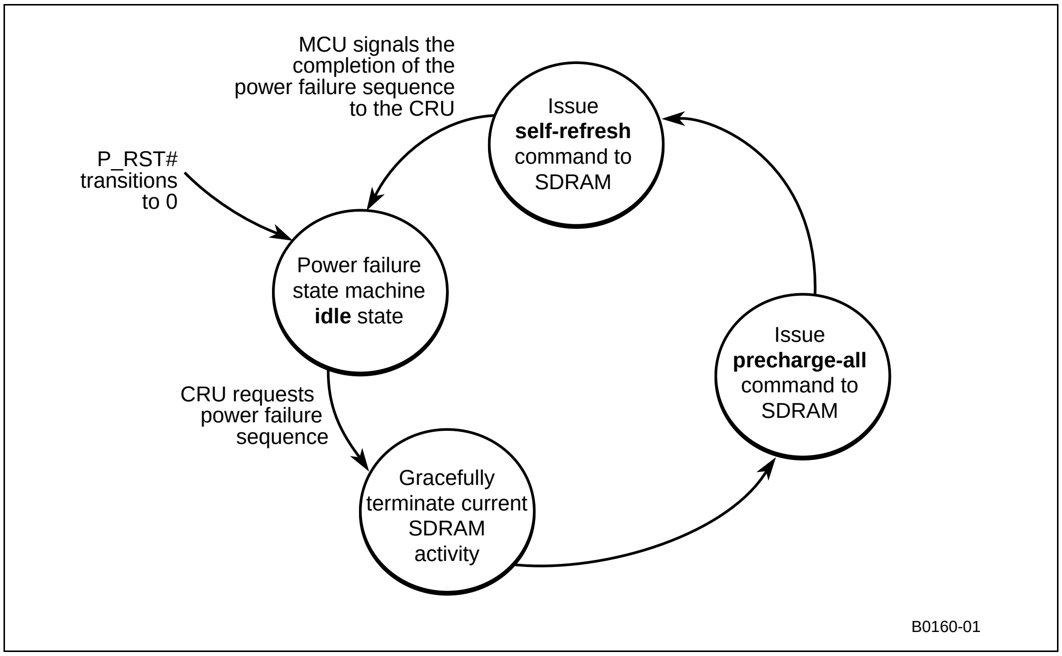
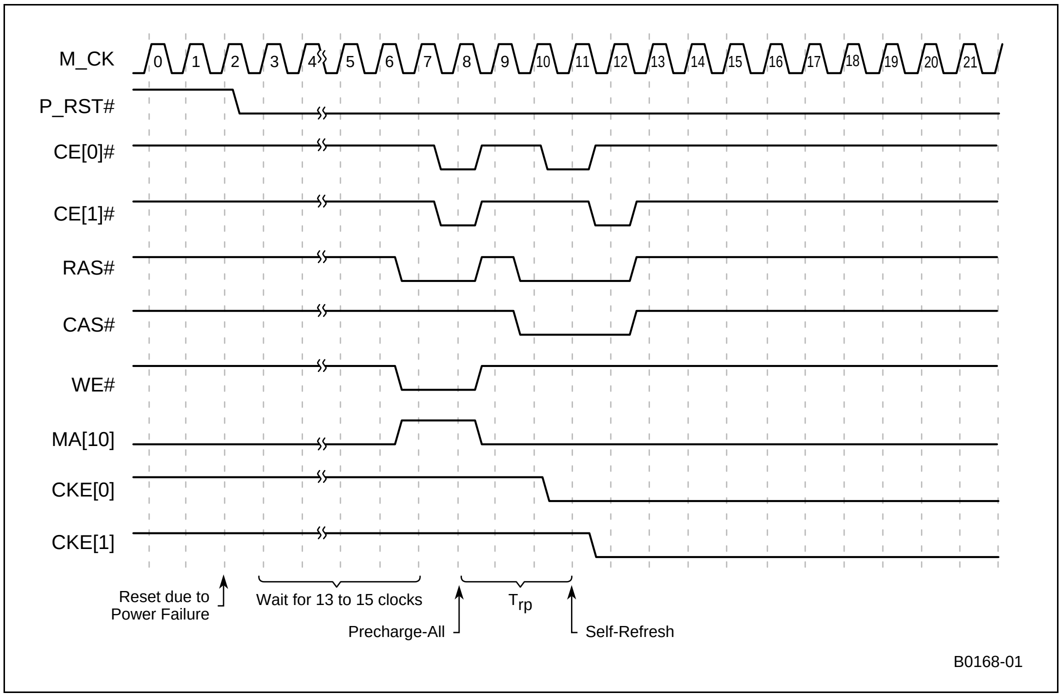

MCU signals the completion of the power failure sequence to the CRU
Issue **self-refresh** command to SDRAM
P_RST# transitions to 0
Power failure state machine **idle** state
Issue **precharge-all** command to SDRAM
CRU requests power failure sequence
Gracefully terminate current SDRAM activity
B0160-01

Figure 3. Power Failure Sequence



Status: Fixed.

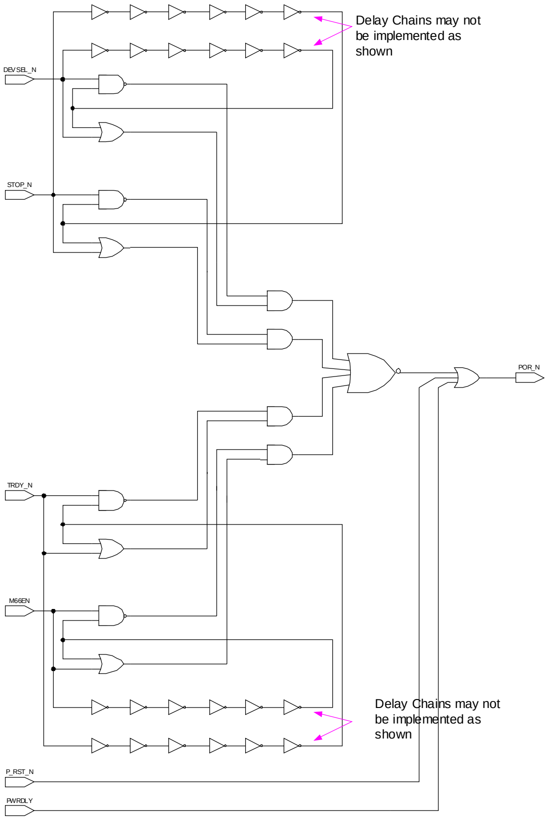
10. PLL Unable to Lock at Reset

Problem: The combination of FRAME#, IRDY#, DEVSEL#, STOP# and TRDY# signals, provide the PCI-X initialization pattern at the rising edge of P_RST#, per Table 6-2 in the *PCI-X Addendum to the PCI Local Bus Specification*, Rev. 1.0a. The PCI-X initialization pattern needs to have been in a known state for a minimum of 10 clock cycles, before the rising edge of P_RST#, to ensure the stability of the PCI-X initialization pattern.

The PCI-X initialization pattern frequency must always match the P_CLK frequency, before the rising edge of P_RST#, to ensure proper PLL lock. The problem is that the PCI-X initialization pattern frequency does not always match the frequency of P_CLK prior to the rising edge of P_RST#. Since there is a time that the PCI-X initialization pattern frequency does not match the P_CLK frequency before the rising edge of P_RST#, the PLL is put into an unknown state until it receives a pulse from the POR# pin, before the rising edge of P_RST#.

Workaround: Pulse the POR# signal once the PCI-X initialization pattern frequency matches the P_CLK, before the rising edge of P_RST#. An example circuit for this workaround is shown in the following diagram. PLD equations for this circuit are available upon request. The function of this circuit is to pulse the POR# pin when there is any change in the PCI-X initialization pattern, prior to the deassertion of P_RST#. A pulse width of approximately 200 ns is required. This allows the PLL to reset and latch the correct PCI-X initialization pattern to ensure proper PLL operation upon coming out of Reset.

When entering PCI-X mode, the M66EN signal needs to be held high **from power-up until the rising edge of P_RST#**.


Delay Chains may not be implemented as shown
DEVSEL_N
STOP_N
POR_N
TRDY_N
M66EN
Delay Chains may not be implemented as shown
P_RST_N
PWRDLY

Status: Fixed.

11. Lost Data During Bursts of Large Number of Partials with 32-bit ECC Memory

Problem: When the MCU operates in 32-bit mode only and it is hit by enough partials to cause the input posted write buffer to fill (in 32-bit mode it holds 512 bytes), the MCU has conditions where it does NOT disconnect on the IB (internal bus) before overrunning.

When the buffer overruns, the MCU momentarily thinks it is empty, allowing the refresh to occur, but also causing all data to be lost for the rest of the burst. The ATU continues to throw data at the MCU, but this data is lost.

This is strictly a 32-bit memory ECC on mode issue, as this is the only way to fill the entire buffer since all buffers on the IB are 1 K in size (except the MCU when operating in 32-bit DDR mode). The DMA, AAU, and core cannot cause the situation in 32-bit mode because they only issue up to two partials in their burst before disconnecting. In these situations, the MCU will drain enough data to prevent buffer overrun.

Workaround: Use 64-bit memory or ECC disabled.

Status: NoFix.

12. P_RST# to PCI-X Initialization Pattern Hold Time (Tprh)

Problem: The combination of FRAME#, IRDY#, DEVSEL#, STOP# and TRDY# signals, provide the PCI-X initialization pattern at the rising edge of P_RST#, per Table 6-2 in the *PCI-X Addendum to the PCI Local Bus Specification*, Rev. 1.0a.

There is a hold time requirement for the PCI-X initialization pattern, following the rising edge of P_RST#, per Table 9-5 in the *PCI-X Addendum to the PCI Local Bus Specification*, Rev. 1.0a, P_RST# to PCI-X initialization pattern hold time, Tprh = 0-50 ns. This hold time specification is also listed in Table 21 of the *Intel® 80321 I/O Processor Datasheet* as TIH3.

There are three signals in the PCI-X initialization pattern that are in violation of the Tprh timing parameter; DEVSEL#, STOP# and TRDY#. The actual minimum hold time for these signals is in the range of 100-500 ps across various temperatures and voltages, versus a minimum hold time specification of 0ns. The implication of this timing violation is, that some systems may not get initialized to the proper PCI-X bus speed and PCI-X mode at the rising edge of P_RST#.

Workaround: Ensure the P_RST# to PCI-X initialization pattern minimum hold time is 1 ns for the DEVSEL#, STOP# and TRDY# signals.

Status: Fixed.

13. The MTTR1 (Core Multi-Transaction Timer) is not operating due to improper behavior of the core internal bus request signal (REQ#)

Problem: The MTTR1 (Core Multi-Transaction Timer) is not operating due to improper behavior of the core internal bus request signal (REQ#). All agents on the bus, except the core, maintain their assertion on REQ# signals upon receiving a retry. When the MCU is busy this means that the core must wait for other agents to complete their transactions before the core gains access to memory. Due to the fact that internal bus agents initiate larger transactions than the core, this issue results in an unbalanced access to the internal bus biased to these other agents (DMA, AAU, ATU, etc.). When operational, the MTTR1 is intended to correct this balance. See Section 11.2.2 of the *Intel® 80321 I/O Processor Developer's Manual* for more information on the MTTR1 function.

Implication: In the case of the MCU internal bus target, this problem is compounded by the many internal bus retries that are issued by the MCU when under heavily loaded conditions. The result is that the internal bus arbiter removes the core access to the bus when the core deasserts REQ#. This condition may result in the core being locked out of accessing the MCU until other internal bus agents have completed their transaction(s) (i.e., when the DMA is in the process of a large block transfer of data, the core may have to wait until the DMA transaction is completed before it would have access to the internal bus to initiate its transaction).

Workaround: No workaround.

Status: NoFix.

14. The MCU supports a page size of 2 Kbytes for 64-bit mode

Problem: The *Intel® 80321 I/O Processor Developer's Manual* (Section 7.1.1 - Table 136 and Section 7.2.2.3) states that the MCU supports a page size of 4 Kbytes for 64-bit mode and 2 Kbytes for 32-bit mode. This is in error.

Implication: The MCU supports a page size of 2 Kbytes for 64-bit mode and for 32-bit mode.

Workaround: No workaround.

Status: NoFix.

15. A logic error in the Memory Controller Unit (MCU) incorrectly reports an ECC Error on memory writes. This error does not corrupt memory contents or data. There are two different conditions that exacerbate the issue.

PART 1: Partial DDR Memory ECC Initialization at Startup

Problem: Data returned from a read to DDR SDRAM is registered at the pad interface as it enters the MCU and is then latched into the MCU Read FIFO. Data at the head of the read FIFO is connected to ECC logic that indicates any data ECC errors. During a DDR SDRAM Read, the MCU Read State Machine (MCU_RSM), by design, reads beyond the targeted area (i.e., the MCU_RSM "over-runs"). The MCU_RSM returns only the data requested to the Mastering Device, but the data that was loaded into the head of the MCU Read FIFO, as a result of the overrun, is evaluated by the ECC logic. Should the over-run data contain an ECC error, an internal flag is set in the MCU, that indicates the MCU Read FIFO is currently "sitting" on an ECC Error.

Since the data sitting at the head of the MCU Read FIFO, at the end of a DDR SDRAM read is not returned to Mastering Device, the MCU MMR logic does not pay attention to any ECC flags set at the end of the read. However, when the next transaction to the MCU immediately following the read is a DDR SDRAM write, and the data at the head of the MCU Read FIFO contains an ECC error, the MCU erroneously loads the ECC MMRs, based upon the data sitting at the MCU Read FIFO from the previous read. The MCU reports the error from the read during the subsequent write. Thus, when the ECC Reporting MMRs are loaded (ECAR), they then contain the address of the write and, the syndrome from the read data and the reported or logged ECC address is incorrect.

It is important to note that all of the ECC circuitry that corrects and reports ECC errors during reads and writes is fully intact and operational. The net effect of this logic error is, that an ECC event is reported for an area of memory the Mastering Device (during the read) did not intend on reading (recall, this error is only triggered because of the MCU_RSM over-run). This error can only be stimulated by a DDR SDRAM write that immediately follows a DDR SDRAM read, where the data at the head of the MCU Read FIFO has an ECC error (this is very unlikely to happen, except in a memory system that has not fully initialized DDR SDRAM).

Workaround: **Part 1:** There are two items in the workaround guidance for this error. Item 1 is how to handle an ECC event reported by the MCU. Item 2 refers to how to minimize the occurrence for a partially scrubbed ("partially initialized") memory system.

Item 1: The guidance in servicing an ECC event for a fully scrubbed DDR SDRAM, is to service each ECC event that is reported by the MCU, even though there exists a very small chance that the ECC event that caused the MCU to report, was not in an area of memory that was intended on being read (in which case the address in the ECAR would be incorrect).

Item 2: For a partially scrubbed memory system, the likelihood of this bug occurring increases greatly for the cases where the over-run of the MCU_RSM goes into un-initialized DDR SDRAM space (this can occur because of the non-linearity of the Internal Bus to DDR SDRAM address translation).

When performing a partial ECC Memory Initialization, the 80321 requires additional memory regions to be initialized. The following additional rules apply:

1. The useable space must be a multiple of 4 Kbyte blocks.
2. All column addresses associated with each used 4 Kbyte block requires the first 256 bytes of the matching 4 Kbyte offset block to also be initialized. The offsets for each of the following examples are derived from the column addressing scheme for the 80321 DDR SDRAM Addressing. Refer to section 7.2.2.2, "DDR SDRAM Addressing", in the *Intel® 80321 I/O Processor Developer's Manual*, for an explanation of 80321 column addressing.

Example 1. 4 Kbyte Initialization Required for 128 Mbyte DDR Memory (Figure 1)

When the program only uses the first 4 Kbyte block, then after initializing the first 4 Kbytes, the first 256 bytes at an offset address of 16 Mbytes must also be initialized.

When using the first 16 Mbytes of memory, the over-run occurs after every 4 Kbytes of memory space has been addressed. From section 7.2.2.2 of the *Intel® 80321 I/O Processor Developer's Manual*, I_AD[24] of the column address gets set, which results in a 16 Mbyte offset.

MA[12:0]	12	11	10	9	8	7	6	5	4	3	2	1	0
Col	-	I_AD[26]	-	**I_AD[24]**	I_AD[11]	I_AD[10]	I_AD[9]	I_AD[8]	I_AD[7]	I_AD[6]	I_AD[5]	I_AD[4]	I_AD[3]

↑

Gets set to '1' on the over-run

Example 2. 12 Kbyte Initialization Required at 20 Mbyte Base Address for 128 Mbyte DDR Memory (Figure 1)

When more than one 4 Kbyte block is used, every 4 Kbyte block of initialized memory has to perform the same 256 byte initialization, at an offset address of 48 Mbytes from each 4 Kbyte block.

When using memory in the 16 Mbyte - 64 Mbyte range, I_AD[24] is already set and the over-run occurs after every 4 Kbytes of memory space has been addressed, and then I_AD[26] of the column address gets set, which results in a 48 Mbyte offset (64 Mbyte – 16 Mbyte).

MA[12:0]	12	11	10	9	8	7	6	5	4	3	2	1	0
Col	-	**I_AD[26]**	-	I_AD[24]	I_AD[11]	I_AD[10]	I_AD[9]	I_AD[8]	I_AD[7]	I_AD[6]	I_AD[5]	I_AD[4]	I_AD[3]

↑

Gets set to '1' on the over-run

Example 3. 4 Kbyte Initialization Required at 112 Mbyte Base Address for 128 Mbyte DDR Memory (Figure 1)

In this example, I_AD[24] and I_AD[26] are set to begin with and then the next over-run clears these two bits, which results in essentially a 16 Mbyte offset or an offset to the beginning of memory, since the 16 Mbyte offset is higher than the total 128 Mbyte address space. The over-run requires initializing the first 256 bytes starting at address 0.

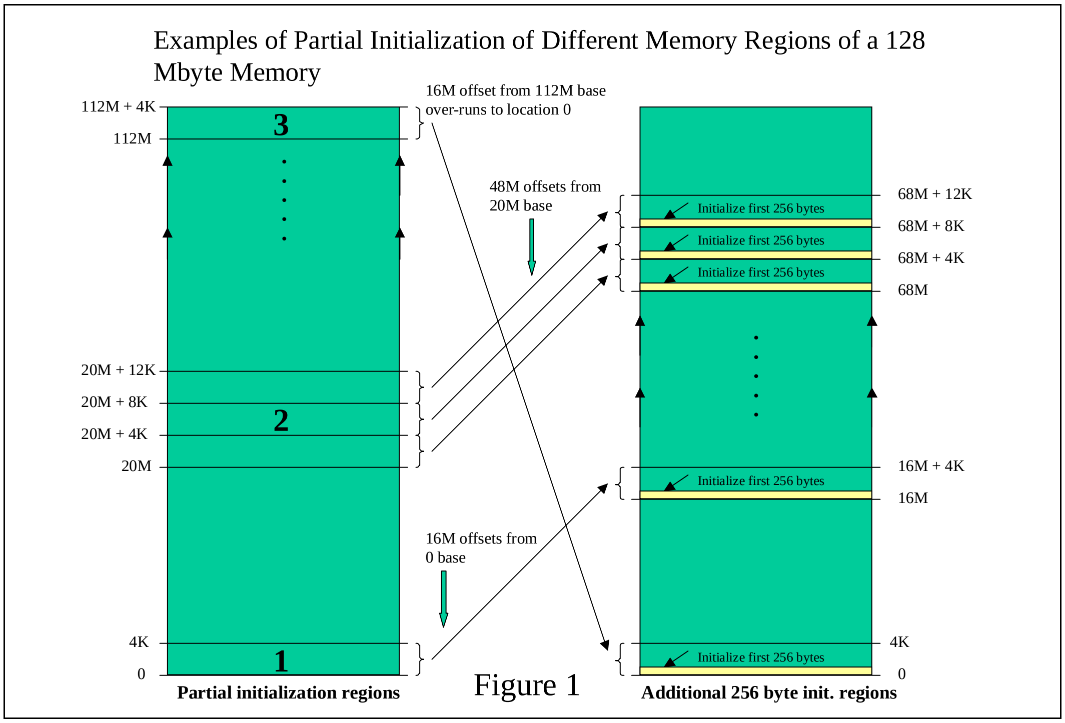
When using memory above 64 Mbytes, I_AD[26] is already set and the over-run occurs after every 4 Kbytes of memory space has been addressed, and then I_AD[24] of the column address gets set. Therefore, I_AD[24] and I_AD[26] are both set, which results in a 16 Mbyte offset (80 Mbyte – 64 Mbyte).

MA[12:0]	12	11	10	9	8	7	6	5	4	3	2	1	0
Col	-	**I_AD[26]**	-	**I_AD[24]**	I_AD[11]	I_AD[10]	I_AD[9]	I_AD[8]	I_AD[7]	I_AD[6]	I_AD[5]	I_AD[4]	I_AD[3]

↑

Both get 'cleared' on the over-run

Figure 1. Partial Memory Initialization Examples


Examples of Partial Initialization of Different Memory Regions of a 128 Mbyte Memory
112M + 4K
112M
3
16M offset from 112M base over-runs to location 0
48M offsets from 20M base
20M + 12K
20M + 8K
20M + 4K
20M
2
16M offsets from 0 base
4K
0
1
Partial initialization regions
Figure 1
Initialize first 256 bytes
Initialize first 256 bytes
Initialize first 256 bytes
68M + 12K
68M + 8K
68M + 4K
68M
16M + 4K
Initialize first 256 bytes
16M
4K
Initialize first 256 bytes
0
Additional 256 byte init. regions

PART 2: Under certain conditions, a location in the MCU Read FIFO can be written to and read from at the same time which may result in the ELOG register being updated incorrectly

Background:The MCU can be partitioned into two blocks; a read block and a write block.

The read block has a sub-block that controls and records the ECC status. When a read is requested, the MCU continues to fetch data after all requested data has been returned to the requesting device. This phenomenon occurs due to an asynchronous clock boundary between the memory bus and the internal bus to support DDR memory. Hence, for a finite period of time, the ECC status logic is receiving information on data that is not returned to the requestor. This is documented in the *Intel® 80321 I/O Processor Developer's Manual*, as normal behavior.

Problem: When a read transaction is completing and a write transaction is requested, an ECC error may be erroneously flagged during the over-run period. The ECC error can be caused either by a true ECC error or due to the skew between the data strobe (DQS) and the internal ECC comparison logic. This issue is associated with the DQS skew.

In normal operation, data into the I/O processor is passed into the MCU Read FIFO and allowed to settle for several cycles before it is read. At the end of a read, the MCU Read FIFO input pointer is cleared so that any over-run data is placed into Address 0 of the FIFO. The Read MCU FIFO output pointer is not cleared until the transaction is completed. When the output pointer ever points at Address 0, read data does not get the normal "several" cycles to stabilize. Thus, a one-cycle path is created directly from the input pad to the ECC calculation and status logic. This path was designed to be one-cycle. However, since the DQS is allowed to vary per DDR specification, the actual path can be constrained to less than one-cycle, further limiting the time the logic has to process the incoming data (i.e., less than one cycle). When DQS moves far enough, erroneous results are presented at the ELOG register.

For read transactions, additional logic prevents the ELOG registers from updating.

Write transactions are retried until the entire MCU has completed the read transaction. However, some logic is enabled during the retried cycle. In this condition, a 1-2 cycle hole exists and the ELOG register can be updated incorrectly.

Workaround: **Part 2:** None.

Note: Although there is a workaround for Part 1, there is not one for Part 2. Therefore, there is not a complete workaround for this erratum.

Status: Fixed.

16. Intel® 80321 I/O Processor/PCI-X Bridge Unexpected Split Completion Error

Problem: In PCI-X mode, when the 80321 does a Memory Block Read, under the following three conditions, the 80321 sets the Unexpected Split Completion Error bit (PCIXSR bit19, DMA-CSR bit2):

1. **The errata condition requires Byte Count Modify bit (BCM) of the Split Completion Transaction Attribute Phase to be set.**

   The Completer sets BCM when the byte count is less than the original request. The Completer may choose to do this when a short data burst would not allow the Requester enough time to recognize a disconnect on an Allowable Disconnect Boundary (ADB). When the data length size is 256 bytes or greater, the Completer does not need to use BCM because the Requester has time to recognize a disconnect. However, some Completers may still choose to split the transaction using BCM.

2. **The Requested Data address range must cross over a 128 byte ADB and the Completer decides to divide the completion transaction using BCM.**

   Not all ADB (naturally aligned 128-byte boundary) address ranges cause this condition because the 80321 divides certain requests into two ADB aligned data requests. The Completer is not allowed to set the BCM on ADB aligned requests. One such condition occurs when the 80321 Maximum Memory Read Byte Count is set to 512 (PCIXCMD bits [3:2]). For this case, the 80321 will divide the request into two ADB aligned requests whenever the ADB crossover address is evenly divisible by 512. When the 80321 Maximum Read Byte Count is set to 1K or greater, then the 80321 will divide requests that cross over 1K address boundaries. Completers are not required to divide the completion transaction into two transactions. They can treat it as an immediate transaction. The address boundaries where most Completers would make divisions are the same boundaries where the 80321 already divides the Request into two ADB aligned requests.

3. **The bits AD[11:8] of the Split Completion Transaction Attribute Phase must have at least one bit set.**

   When these four bits are zero, then the errata will not occur. These bits are defined as Completer Function Number, [10:8], and the low order bit of the Completer Device Number, [15:11].

Workaround: Assumption: The 80321 is in control of all data memory read requests. When a device driver not residing on the 321 were to build and execute a request by reading the 80321 memory and registers, then that driver would have to be changed also. This may not prove practical.

When a Data Read Request from the 80321 would cause the errata condition, the fix to the errata condition is:

a. Divide the request into two aligned data requests. When the original request begins at Adr0 and has a length of BC0, then the new requests would use:

```
Adr1 = Adr0              BC1 = 256 - BC0
Adr2 = Adr1 + BC1        BC2 = BC0 - BC1
```

The workaround could be applied in one of two ways:

b. Each request could be checked before being initiated and then adjusted.
This approach has the advantage of never allowing the error condition to occur, but has the disadvantage of extra overhead in the main stream code and the code will not be required with future stepped components.

c. The fix could be applied only after an error condition has been detected.
This approach has the disadvantage of lost time because of one transfer would be useless and has the advantages that the error recover code may already reside in a single location and the code will not require changing for optimal performance when future stepped components are available.

Status: Fixed.

17. Vih Minimum Input High Voltage (Vih) level for the PCI pins

Problem: The Vih Minimum Input High Voltage (Vih) level for the PCI pins is being tested at 100 mV higher than the minimum Vih level specified in Table 4-3 (DC Specifications for 3.3 V Signaling) of the *PCI Local Bus Specification*, Revision 2.2. This Vih test limit only applies to cold temperature testing specified to be 0°C.

The *PCI Local Bus Specification*, Revision 2.2 specifies the minimum Vih level to be 0.5 Vcc. The Vcc specification is 3.3 V +/- 10% with the minimum Vcc specification (or minimum power level) being tested at 3.0 V. The minimum Vih level per the PCI Specification should therefore be 0.5(3.0 V) or 1.5 V. The 80321 is unable to meet this minimum Vih level at cold temperature testing specified to be 0°C.

Implication: During cold temperature manufacturing testing, 80321 silicon is subjected to a 0°C environment for an extended period of time. During this time the Vih test is implemented and the junction temperature is at or near the test temperature of 0°C. This junction temperature is considered to be far less than the temperature the 80321 silicon would be subjected to in a customer application under operating conditions.

Below is an example calculation showing the expected junction temperature for a customer application operating in an ambient temperature of 0°C:

Tj = junctions temperature, Ta = ambient temperature, qja = junction to ambient thermal resistance of the package, P = power at minimum Vcc

Tj = Ta + (qja * P) where Ta = 0C, qja = 13.94 C/W assuming 200lfm airflow (see Table 11 of the *Intel® 80321 I/O Processor Datasheet*), P = 3.0 W

Tj = 0 + (13.94 C/W * 3.0 C)

Tj = 41.82 C

Workaround: The minimum Vih level for the PCI pins will be tested at the *PCI Local Bus Specification*, Revision 2.2 specification (0.5 Vcc) plus an additional 100 mV that equates to 1.6 V during cold temperature manufacturing testing.

Status: NoFix.

Specification Changes

1. DDR V_{CC} and DDR V_{REF} minimum specifications need to be changed on the A-0 and B-0/B-1 steppings

Issue: DDR V_{CC} and DDR V_{REF} minimum specifications need to be changed in Table 17 of the Intel® 80321 I/O Processor *Advance Information Datasheet* (273518-001) for the A-0 and B-0/B-1steppings.

The DDR voltages specifications from Table 17, in the referenced *Intel® 80321 I/O Processor Advance Information Datasheet*, are shown in Table 1.

Table 1. Published DDR V_{CC} and DDR V_{REF} Values

Symbol	Parameter	Min	Max	Units
V_{CC25}	2.5V DDR Supply Voltage	2.3	2.7	V
V_{REF}	Memory I/O Reference Voltage	$V_{CC25}/2 – 0.05$	$V_{CC25}/2 + 0.05$	V

The new DDR voltages specifications — that supersede the values listed in Table 17 of the referenced datasheet — are shown in Table 2.

Table 2. Revised DDR V_{CC} and DDR V_{REF} Values

Symbol	Parameter	Min	Max	Units
V_{CC25}	2.5V DDR Supply Voltage	2.375	2.7	V
V_{REF}	Memory I/O Reference Voltage	$V_{CC25}/2 – 0.013$	$V_{CC25}/2 + 0.05$	V

2. DDR SDRAM signal timing change, T_{VA3}

Issue: The DDR SDRAM Address and Control write output — valid after CK (T_{VA3}) minimum specification — needs to be changed in Table 22 of the *Intel® 80321 I/O Processor Datasheet* (273518-001) for all steppings.

The *old* T_{VA3} timing specification — in Table 22 of the referenced Intel® 80321 I/O Processor *Datasheet* — is shown in Table 3.

Table 3. Published T_{VA3} Value

Symbol	Parameter	Min	Max	Units
T_{AV3}	Address and Control write output valid after CK	3.5		ns

The *new* T_{VA3} timing specification — that supercedes the value listed in Table 22 of the referenced datasheet — is shown in Table 4.

Table 4. Revised T_{VA3} Values

Symbol	Parameter	Min	Max	Units
T_{AV3}	Address and Control write output valid after CK	3.3		ns

3. P_BMI (AE23) added to B-0/B-1 Steppings

The P_BMI (AE23) signal has been added to the Intel® 80321 I/O processor (B-0/B-1 Stepping). This signal replaces, using an external GPIO pin for Initialization Device Select (IDSEL) control of an I/O device during host configuration cycles.

Issue: I/O Device IDSEL control using the new P_BMI signal:

When the system boots after reset, the host BIOS initiates a PCI bus scan to find all the PCI components installed in the system. The system uses the IDSEL signal to address the I/O device when assigning the necessary resources. Without special control over the IDSEL signal during configuration cycles, the host and the 80321 may both attempt to configure the same I/O device. By taking control of IDSEL, the 80321 can execute configuration cycles to the slave I/O device (SCSI) and properly hide the slave I/O device from the host and operating system initiated configuration cycles.

The 80321 has eight integrated General Purpose Input Output (GPIO) pins, referred to as GPIO[7:0]. These pins, along with the new PCI-X Bus Master Indicator (P_BMI) signal, can be used to control the IDSEL to the I/O device. External circuitry is no longer required other than a simple switch. The output function of the P_BMI signal is controlled by the GPIO Output Data Register (GPOD), Bit 0 as shown in Table 1. The P_BMI signal is always driven and defaults to driving low at power up.

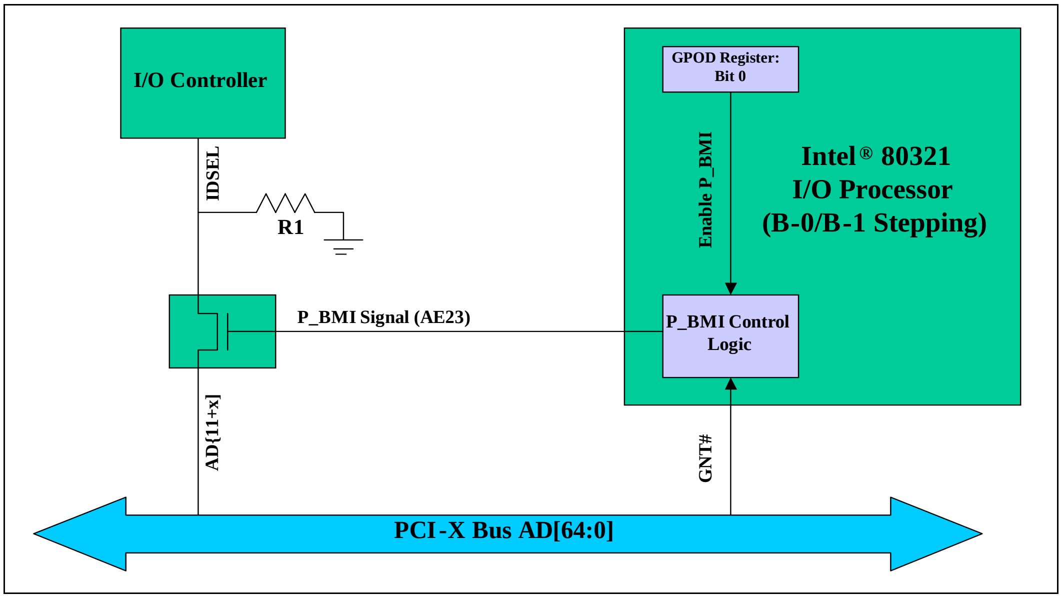
Table 1. GPIO Output Data Register (address = FFFF E7CCh)

Bit	Default	Description
0	0	This bit value is driven on the P_BMI signal when the 80321 has been given control of the bus (granted GNT#) by the bus arbiter.

The IDSEL signal is used as a chip select during configuration cycles initiated by the BIOS, operating system or 80321. The GPOD[0] can be driven low in firmware, thereby disabling the P_BMI signal and hiding the host I/O device from the system, by turning off its IDSEL. When the 80321 intends to perform configuration cycles in the PCI bus segment of the I/O device, the P_BMI signal should be asserted high by driving the GPOD[0] high. The affect of these two operations is, that the I/O device is initialized and controlled by the 80321. More care must be taken with the gate chosen to control IDSEL, since most host bridge controllers do not use PCI address stepping. With IDSEL being a synchronous signal, with respect to CLK, the switch used must be a sub nanosecond propagation delay device (e.g., Pericom PI5C3303). In Figure 2, the P_BMI signal is used to control a mux/demux switch that is used to enable/disable IDSEL to the I/O device.

Note: The host BIOS does not require any modifications to accommodate this implementation. All the responsibility for I/O device configuration and resource falls to the 80321 firmware.

Figure 2. IOP321 P_BMI Signal Implementation for 80321 B-0/B-1 Stepping



Specification Clarifications

1. The Intel® 80321 I/O processor is compliant with the PCI Local Bus Specification, Revision 2.2 but it is not compliant with PCI Local Bus Specification, Revision 2.3

Issue: The Intel® 80321 I/O processor was designed to be compliant with the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a, that calls out compliance with the *PCI Local Bus Specification*, Revision 2.2. Since the release of the 80321, the PCI Special Interest Group has released a new specification revision, *PCI Local Bus Specification*, Revision 2.3.

Status: NoFix. The current stepping of the 80321 is not compliant with *PCI Local Bus Specification*, Revision 2.3 and there are no plans to make it compliant with the *PCI Local Bus Specification*, Revision 2.3 in future steppings.

2. Modifications to the Hot-Debug procedure are necessary for the Intel® 80321 I/O processor when flat memory mapping is not used (Virtual Address = Physical Address)

Issue: The Intel® 80321 I/O processor can implement Hot Debug as stated in the application note "*Hot-Debug for Intel® Xscale™ Core Debug*: "http://developer.intel.com/design/iio/applnots/273539.htm".

However, there can be a conflict for resources when flat memory mapping is not used (Virtual Address = Physical Address).

This is primarily due to the debug implementation within the core that causes the Instruction Memory Management Unit to be disabled when in this Special Debug State.

Status: NoFix. The following are suggested steps to overcome this conflict within a debug environment.

1. Instrument the application code to add an infinite loop before any memory is remapped (physically or virtually).
2. Hook up the JTAG Debugger that supports Hot Debug.
3. Set PC to address passed the loop.
4. The code can now run without the need to reset the application environment.

Note: Once a debug session has ended, you must follow the above steps over again in order to regain debug control.

3. Removed. Does not apply to the Intel® 80321 I/O processor.

4. BAR0 Configuration When Using the Messaging Unit (MU)

Issue: When the BAR0 is configured as a prefetchable register by default and a burst request crosses into or through the range of offsets 40h to 4Ch (i.e., this includes the Circular Queues), the transaction is signaled a Target Abort immediately on the PCI/PCI-X bus, which may be read as an NMI by the host BIOS.

Status: Doc. Do not configure the BAR0 as prefetchable when using BAR0 and the non-prefetchable MU registers (i.e., range of offsets 40h to 4Ch). Configure the BAR0 as non-prefetchable, IABAR0[3], when accessing these non-prefetchable MU registers. Since non-prefetchable memory windows cannot be placed above the 4 Gbyte address boundary, when the Prefetchable Indicator bit, IABAR0[3], is cleared prior to host configuration, also clear the Type Indicator bits, IABAR0[2:1] for 32-bit addressability. When the non-prefetchable MU registers are in use, those memory accesses that require prefetchable operations, use the BAR2 configured as prefetchable.

5. Reading Unpopulated SDRAM Memory Banks

Issue: A hang condition can occur with the 80321 when firmware does a read to unpopulated SDRAM memory and DQS0 is sampled low. In this scenario, putting a load (i.e., scope probe), on the DQS0 signal could trigger DQS0 to be sampled low, which the MCU interprets as the pre-amble and waits for DQS0 to go high. Since the read is to unpopulated memory, nothing drives the DQS0 signal high, therefore the 80321 appears to hang.

Status: Doc. Do not attempt to read from non-existent memory. In some applications, firmware performs a memory scan, typically during boot-up, to determine the total amount o SDRAM installed. Instead, either use the Serial Presence Detect (SPD) mechanism or have it hard coded in firmware. SPD is used to read, via I^2C, from a non-volatile storage device. This device contains data programmed by the DIMM manufacturer, that identifies the module type, various SDRAM organizations and timing parameters. Using SPD or hard coded firmware eliminates the need to do SDRAM sizing in the firmware.

6. 32-bit Writes-to-Unaligned 64-bit Addresses, are Promoted to 64-bit Aligned Writes

Issue: In 80321-based applications that run the PCI bus segment in 32-bit PCI Mode or 64-bit PCI Mode with 32-bit targets, write transactions that are on unaligned 64-bit addresses are promoted to 64-bit aligned writes. The first half of the 64-bit write is on a 64-bit aligned address and has the BE# signals disabled. Therefore, the write is invalid. The second half on the 64-bit write is a valid write with the BE# enabled and the write is to the intended 32-bit address.

Per the *PCI Local Bus Specification*, Revision 2.2, the PCI compliant devices should ignore the first half of the 64-bit write due to the BE# signals being disabled.

Status: For devices that support using the I/O memory window, the 64-bit write does not occur when using the 80321 ATU I/O Window and the only expected 32-bit write occurs. See section 3.2.2.2 of the *Intel® 80321 I/O Processor Developer's Manual* for details.

For memory mapped devices, the only option is to run in PCI-X mode, where the byte count and starting address are consistent with the actual number of bytes to be written (i.e., 4). This is so because, when a 64-bit PCI-X request gets downshifted, the requester can use the starting address/byte count to recognize that the write request does not cross a DWORD address boundary and only perform a single 32-bit wide data cycle.

7. In-order Delivery not guaranteed for data blocks described by a single DMA descriptor

Issue: In-order delivery is not guaranteed for data blocks described by a single DMA descriptor that crosses a 1 KB boundary. This may result in out of order execution of the DMA transfer. When multiple DMA descriptors are used the ordering is maintained with respect to the blocks described by each descriptor. When ordering is important, the ordering needs to be maintained by splitting the relevant pieces of data into multiple DMA descriptors.

Example A 100 byte DMA transfer described by a single descriptor with a source address of 0x3ff8. Since each DMA channel has two 1 KB buffers, the DMA unit breaks this transaction at the 1 KB boundary. Therefore, the first buffer might fetch the 8 bytes from 0x3ff8-0x3fff and the second buffer might fetch the remaining 92 byes from 0x4000-0x405C. Both buffers have the ability to access the internal bus, without preference (i.e., either buffer may gain access first). Therefore, it is possible the 92 bytes of data after the 1 KB boundary could be transferred to the destination before the first 8 bytes. However, the transaction is completed and all data has been copied to the correct address when the descriptor completes (i.e., descriptors are not completed out of order).

Status: When a data delivery sequence is required, descriptors should be used to ensure sequenced arrival (e.g., in the example above), break the data into blocks then use multiple descriptors linked in the correct order to ensure sequential data delivery.

Documentation Changes

1. Table 4 Page 18 second row has incorrect data

Problem: The RCVENO# description of Table 4 states:
"RECEIVE ENABLE OUT must be connected to RCVENI# of the 80321and be trace length matched to DQ[63:0]."

Workaround: Change the description to the following:
"RECEIVE ENABLE OUT - this pin must be connected to RCVENI# of the 80321 and be Trace Length Matched = Average Memory Clock Length + Average DQS Length."

Affected Docs: *Intel® 80321 I/O Processor Design Guide*.

Status: Updated in latest Design Guide Revision.

2. Table 9 (Sheet 3 of 5), page 31 and Table 10 (Sheet 5 of 5), page 38 have incorrect data

Problem: In the Third Signal column, Signal WR# (Ball P23) is incorrect.

Workaround: Change WR# to W/R#.

Affected Docs: *Intel® 80321 I/O Processor Datasheet*.

3. Table 10 (Sheet 2 of 5), page 35 has incorrect data

Problem: In the Third Signal column, Signal P_AD4 (Ball AC7) is incorrect.

Workaround: Change P_AD4 to P_CLK.

Affected Docs: *Intel® 80321 I/O Processor Datasheet*.

4. Section 6.2.2 on page 37 has incorrect data

Problem: The last sentence in the first paragraph states "With conventional PCI mode, a low on **P_M66EN** determines the PCI bus is at 66 MHz."

Workaround: Change the last sentence in the first paragraph to the following: "With conventional PCI mode, a low on **P_M66EN** determines the PCI bus is at 33 MHz."

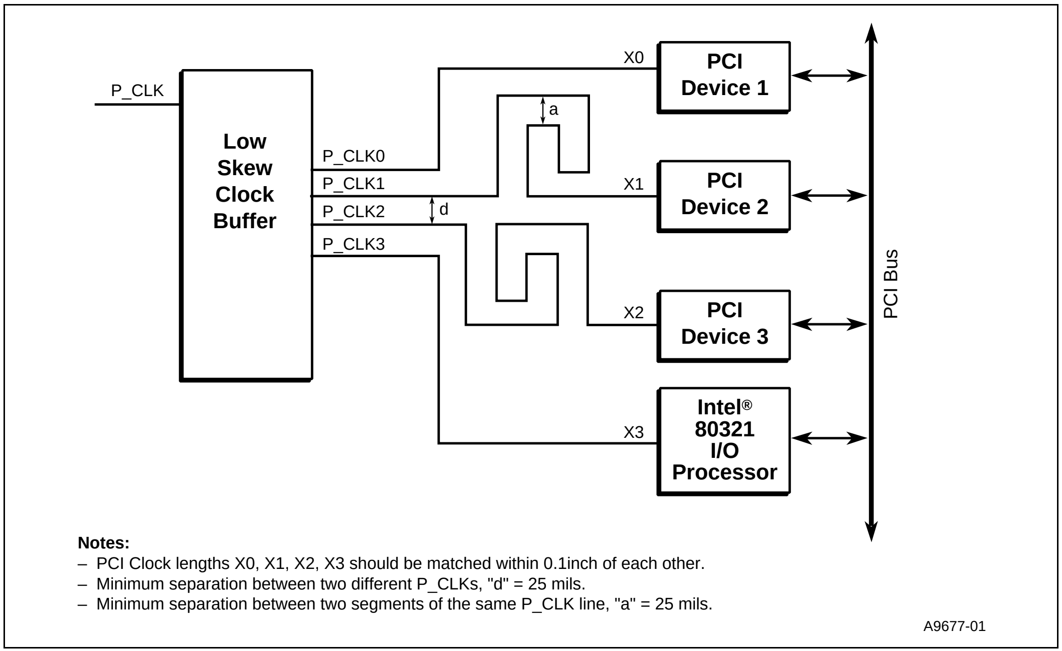
Affected Docs: *Intel® 80321 I/O Processor Design Guide*.

Status: Updated in latest Design Guide Revision.

5. Figure 14 on page 40 has missing text

Problem: The second note shows “...set __CLKs...”. It should read “...set P_CLKs...”.

Workaround: Replace Figure 14 with the following:



Notes:

- PCI Clock lengths X0, X1, X2, X3 should be matched within 0.1inch of each other.
- Minimum separation between two different P_CLKs, "d" = 25 mils.
- Minimum separation between two segments of the same P_CLK line, "a" = 25 mils.

Affected Docs: *Intel® 80321 I/O Processor Design Guide*.

Status: Updated in latest Design Guide Revision.

6. Table 18, page 61 has missing data and incorrect data

Problem: Table 18 is missing data and lists incorrect data.

Workaround: Replace Table 18 with the following:

Symbol	Parameter	Minimum	Maximum	Units
V_{CC25}	2.5 V Supply Voltage for DDR (also referenced as V_{DD25})	2.3	2.7	V
V_{DDQ}	I/O Supply Voltage	2.3	2.7	V
V_{REF}	Memory I/O Reference Voltage	V_{CC25} /2 - 0.05	V_{CC25} /2 + 0.05	V
V_{TT}	DDR Memory I/O Termination Voltage	V_{REF} - 0.04	V_{REF} + 0.04	V

Affected Docs: *Intel® 80321 I/O Processor Design Guide*.

Status: Updated in latest Design Guide Revision.

7. Section 7.6.1 page 75 has incorrect data

Problem: The first paragraph states:
"According to the *PCI Local Bus Specification*, Revision 2.2, PCI_RST# can be asserted when the system power drops as low as 2.5 V. This voltage is too low for the 80321 to be able to execute the power-fail sequence."

Workaround: Change the first paragraph to the following:
"According to the *PCI Local Bus Specification*, Revision 2.2, P_RST# can be asserted when the system power drops as low as 2.5 V. This voltage is too low for the 80321 to be able to execute the power-fail sequence."

Affected Docs: *Intel® 80321 I/O Processor Design Guide*.

Status: Updated in latest Design Guide Revision.

8. Section 7.6.1 page 76 has incorrect data

Problem: The first paragraph states:
"...This PWRDELAY signal remains asserted a few milliseconds after the S_RST# to allow ample time for the power-fail state machine to execute its sequence..."

Workaround: Change the first paragraph to the following:
"...This PWRDELAY signal remains asserted a few milliseconds after the P_RST# to allow ample time for the power-fail state machine to execute its sequence..."

Affected Docs: *Intel® 80321 I/O Processor Design Guide*.

Status: Updated in latest Design Guide Revision.

9. Section 7.6.1 page 77 has incorrect data

Problem: The first paragraph states:
"The latches are cleared when the 80321 drives SCKE[1:0] low with a self-refresh command and are reset when S_RST# is driven from low to high after system power is recovered."

Workaround: Change the first paragraph to the following:
"The latches are cleared when the 80321 drives SCKE[1:0] low with a self-refresh command and are reset when P_RST# is driven from low to high after system power is recovered."

Affected Docs: *Intel® 80321 I/O Processor Design Guide*.

Status: Updated in latest Design Guide Revision.

10. Section 7.6.3 page 78 has missing data and incorrect data

Problem: The first paragraph states:
"...Power to DDR SDRAM is ensured with an automatic switch over to backup battery power when the system power is lost. Refer to the *Intel® 80321 I/O Processor Datasheet* for more information about this function."

Workaround: Change the first paragraph to the following:
"...Power to DDR SDRAM is ensured with an automatic switch over to backup battery power when the system power is lost. Battery backup should maintain power on DDR voltages to prevent data loss. Refer to the *Intel® 80321 I/O Processor Developer's Manual*, section 7.3, for more information about this Power Failure mode."

Affected Docs: *Intel® 80321 I/O Processor Design Guide*.

Status: Updated in latest Design Guide Revision.

11. Section 14.1 page 113 has missing data

Problem: The second paragraph states:
"The equivalent for other analyzers can be substituted. AFuturePlus Systems configuration file with the FS1104 product that matches the pinout in Table 33."

Workaround: Change the second paragraph to the following:
"The equivalent for other analyzers can be substituted. AFuturePlus Systems configuration file with the FS1104 product that matches the pinout in Table 33. Refer to the *PCI-X Addendum to the PCI Compliance Checklist* and the *PCI Compliance Checklist*,, Revision 2.1, available on the www.pcisig.com website for details about the PCI/PCI-X protocol and electrical specification compliance."

Affected Docs: *Intel® 80321 I/O Processor Design Guide*.

Status: Updated in latest Design Guide Revision.

12. Channel Control Register; Channel Enable, page 248

Problem: Note required (see below):

Note: The Channel Enable bit is not autoclearing. When a descriptor is loaded by user software to the NDAR and the Channel Enable bit is set from a prior execution, the Channel Enable bit must be cleared to reset the state of the DMA and the AAU Controller. This does not apply when using append to chain to existing descriptors.

Affected Docs: *Intel® 80321 I/O Processor Developer's Manual*.

Status: Update to Developer's Manual pending.