

This appendix records all the known bugs of the different revisions of the Schizo design and its functionally equivalent derivative.

## *Products Rights Notice:*

Copyright © 1991-2007 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, California 95054, U.S.A. All Rights Reserved

You understand that these materials were not prepared for public release and you assume all risks in using these materials. These risks include, but are not limited to errors, inaccuracies, incompleteness and the possibility that these materials infringe or misappropriate the intellectual property right of others. You agree to assume all such risks.

THESE MATERIALS ARE PROVIDED BY THE COPYRIGHT HOLDERS AND OTHER CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS (INCLUDING ANY OF OWNER'S PARTNERS, VENDORS AND LICENSORS) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THESE MATERIALS, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Sun, Sun Microsystems, the Sun logo, Solaris, OpenSPARC T1, OpenSPARC T2 and UltraSPARC are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd. The Adobe logo is a registered trademark of Adobe Systems, Incorporated. Part of the products covered by these materials may be derived from the Berkeley BSD systems licensed by the University of California. Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product described in these materials. This distribution may include materials developed by third parties who have intellectual property rights therein. Products covered by and information contained in these materials may be controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists may be prohibited.

## A.1 Useful information

### A.1.1 Identifying Schizo family revisions

The Safari Device ID register, which is at offset 0x0.0000.0000 in any Safari device's address space, is used to identify Schizo parts and determine their revision.

The MT (Module Type) field (bits 9:4) contains the value 0x15 to identify all Schizo parts and all functionally equivalent derivatives (such as ELE).

The revision of parts in the Schizo family is further identified by the MID (Manufacturer ID) and MR (Module Revision) fields (bits 15:10 and 3:0 respectively).

Separate from the Safari Device ID register, and accessible via an IEEE 1149.1 standard JTAG port, each revision in the Schizo family also has a unique JTAG ID.

Table A-1 enumerates the values of these fields and the revision history of parts in the Schizo family.

Table A-1 Schizo Family Revision Identification

Part	Revision	Release Date	MID	MR	JTAG ID	Identifiable Markings
ELE		06/05/03	0x2B	0x7	0x16973321	Sun Part# 100-6818-02
Schizo	2.5	10/25/02	0x2A	0x7	0x307BE06D	Sun Part# 100-4736-07
Schizo	2.4	11/30/01	0x2A	0x6	0x207BE06D	Sun Part# 100-4736-06
Schizo	2.3	06/12/01	0x2A	0x5	0x107BE06D	Sun Part# 100-4736-05
Schizo	2.2	06/26/00	0x2A	0x4	0x1824C06D	Sun Part# 100-4736-04
Schizo	2.1	12/14/99	0x2A	0x3	0x1424C06D	Sun Part# 100-4736-03

On a running system, the ``prtconf -pv`` or ``prtdiag -v`` commands can be used to show the MR value for any Schizo parts. ``prtconf`` can also show the MID value. Examples of both of these commands are shown in Code Example A-1. In both cases, the reported "version" corresponds to the MR field value. For ``prtconf`` the "implementation" corresponds to the MID. There is no generic way to get the JTAG IDs for a running system, although there may be system specific methods.

NOTE: ELE is a Schizo 2<sup>nd</sup> source. It is electrically and functionally equivalent to Schizo 2.5.

Code Example A-1 Using prtconf and prtdiag to view revision information

```
# /usr/platform/sun4u/sbin/prtdiag -v
...
IO ASIC revisions:
-----
          Port
Model    ID  Status Version
-----  -  -
Schizo   8   ok      4
...

# /usr/sbin/prtconf -pv
...
Node 0xf006ca40
...
  name: 'pci'
  device_type: 'pci'
...
  implementation#: 0000002a
  version#: 00000003
  portid: 00000008
...
```

---

**Note** – There may be slight variations in the output on different systems, such as printing “pci” instead of “Schizo” for ‘prtdiag’ output.

---

### A.1.2 Required kernel patch revisions

The following kernel patch levels are required to ensure that workarounds are enabled for all known Schizo errata (use ‘showrev’ or ‘uname -rv’ to see):

- Solaris 10: FCS version
- Solaris 9: 112233-13
- Solaris 8: 117000-03

## A.2 External (User visible) Errata Table

Table A-2 Schizo Family External Errata Table

Errata	Schizo					ELE	Description
	2.1	2.2	2.3	2.4	2.5		
1	✓	Fixed	Fixed	Fixed	Fixed	Fixed	Performance counter irregularities
2	✓	Fixed	Fixed	Fixed	Fixed	Fixed	PCI discard timeouts (DTO) falsely reported
3	✓	✓	Fixed	Fixed	Fixed	Fixed	Context flush of active context causes PCI hang
4	✓	✓	Fixed	Fixed	Fixed	Fixed	Alternating streaming DMA access pattern hangs PCI
5	✓	✓	✓	Fixed	Fixed	Fixed	SSM mode streaming DMA read performance
6	✓	✓	Fixed	Fixed	Fixed	Fixed	Streaming DMA read with master wait states may return bad data
7	✓	✓	Fixed	Fixed	Fixed	Fixed	Streaming DMA write with empty data phases hangs PCI
8	Feature	Feature	Feature	Feature	Feature	Feature	Livelock when two DMA masters access same streaming page
9	Feature	Feature	Feature	Feature	Feature	Feature	DMA writes allowed to read-only streaming page
10	Feature	Feature	Feature	Feature	Feature	Feature	PCI Target initial latency violated
11	Feature	Feature	Feature	Feature	Feature	Feature	PCI arbiter not robust for odd REQ/GNT behavior
12	✓	✓	✓	Fixed	Fixed	Fixed	Parity error on address phase of 64-bit wide PIO
13	Feature	Feature	Feature	Feature	Feature	Feature	Inappropriate PCI command used for 64-byte PIO reads
14	✓	✓	✓	Fixed	Fixed	Fixed	64-bit parity errors not detected during PIO reads

Table A-2 Schizo Family External Errata Table

Errata	Schizo					ELE	Description
	2.1	2.2	2.3	2.4	2.5		
15	✓	✓	✓	Fixed	Fixed	Fixed	PERR driven one cycle too long
16	N/A	N/A	N/A	✓	Fixed	Fixed	Enable/disable for DMA write parity error interrupt
17	✓	✓	✓	✓	Fixed	Fixed	Safari protocol error

✓ = Errata applies to this revision  
 n/a = Errata was not present in these versions  
 Fixed = Bug fixed or fix carried forward from previous version  
 Feature = Irregular feature that will not be changed in silicon

### A.3 Detailed External Errata Summary

#### Errata #1: Performance counter irregularities

Bug ID(s): 4297402, 4297406, 4297413, 4308921

**Symptom:**

Certain performance counter events are not counted correctly.

**Description:**

The following events are not counted correctly when selected for use with Schizo's performance counter (typically accessed via the 'busstat' command).

1. DMA bus cycle count (event 0x05) - On a 33MHz bus, this event will not ever be incremented.
2. DMA word transfer count (event 0x06) - This event incorrectly includes PIO transfers.
3. PIO bus cycle count (event 0x07) - This has two problems. The first is that instead of counting the cycles in the PIO transaction, this event actually counts the cycles that Schizo's internal block is given a PCI grant. The second problem is that on a 33MHz bus, each cycle is counted twice.

This is fixed in Schizo 2.2.

**Workaround:**

None

## *Errata #2: PCI discard timeouts (DTO) falsely reported*

Bug ID(s): 4304514

**Symptom:**

If discard timeouts are enabled, spurious DTO errors are reported. If disabled, PCI bus hangs may occur with non-compliant devices.

**Description:**

For each PCI bus, Schizo has four consistent DMA read buffers. When a PCI master requests data, a read buffer is allocated, and the master is retried until the buffer has the desired data from system memory. The buffer is freed once the data is delivered to the master.

To deal with masters that never complete a transaction when data is ready (such masters are non-compliant according to the PCI spec), Schizo implements a discard timeout for each buffer. When enabled, after the timeout period, the buffer is freed and a DTO interrupt is generated.

Due to a bug in the DTO logic, if the PCI bus is running at 66MHz, spurious DTOs can be periodically reported as long as there is consistent DMA read activity.

This is fixed in Schizo 2.2.

**Workaround:**

In versions of Schizo with this bug, discard timeouts are disabled. Non-compliant PCI cards (that might issue a transaction and fail to retry it until it's completed) should not be used in such a system.

## *Errata #3: Context flush of active context causes PCI hang*

Bug ID(s): 4386803, 4424664, 4427954, 4450523

**Symptom:**

A DMA read transaction on the PCI bus is retried indefinitely by Schizo.

**Description:**

Under some conditions when the OS does a context flush and there are streaming DMA reads active for pages with the same context number, the streaming cache can get into an inconsistent state that will cause it to retry a transaction indefinitely.

The conditions required to cause the problem are as follows:

- The OS must be flushing a context X
- PCI device 1 must have outstanding streaming DMA reads in pages that are also in context X
- PCI device 2 (which may be the same as PCI device 1) must have an outstanding streaming DMA read for a page that is not in context X

If all of this occurs, then there is a possibility that the state in the streaming cache for PCI device 2's transaction becomes corrupt, causing the streaming cache to subsequently retry that transaction indefinitely. The bug does not occur every time the above conditions are met, but only occasionally when the timing of the events meets some additional criteria that won't be covered here.

This is fixed in Schizo 2.3.

**Workaround:**

One workaround for this bug is to disable the Schizo nexus driver from doing context flushes altogether. This will cause the nexus driver to use less efficient page flushing instead. This workaround has not been implemented by default.

Another workaround is for a device to do consistent DMA instead.

The main workaround is for devices to avoid the conditions that cause this bug. This will typically involve changes to the device driver so that it only does the DDI calls that can cause flushes (`ddi_dma_sync()` or `ddi_dma_unbind_handle()`) when DMA by the device is known to be inactive.

There is an alternative workaround for device drivers where this is not practical. There is a mechanism for a device driver to communicate to the nexus driver that it may flush an active context. The nexus driver will then disable context flushing for just that device (and do page flushing instead). To use this workaround, the driver needs to define the property "active-dma-flush" prior to calling `ddi_dma_alloc_handle()`.

## *Errata #4: Alternating streaming DMA access pattern hangs PCI*

Bug ID(s): 4390218

**Symptom:**

A DMA read transaction on the PCI bus is retried indefinitely by Schizo.

**Description:**

If a device alternates accesses to two different, non-consecutive parts of the same streaming DMA page, under certain conditions, the state of Schizo's streaming cache can become corrupt, causing the device to be retried indefinitely.

One of the required conditions for this behavior is that an unrelated streaming DMA read to the last cacheline of a page needs to appear on the bus while the alternating accesses to the same page are occurring. Another required condition is that the alternating accesses to the same page use either Memory Read Line or Memory Read Multiple commands on the PCI bus. In addition, certain internal events having to do with the return of DMA read data have to line up precisely.

This is fixed in Schizo 2.3

**Workaround:**

There are a few possible workarounds for this bug:

- Avoid accessing different parts of a streaming page at the same time
- Map a page that requires simultaneous access to different areas as consistent instead of streaming
- Have the device use Memory Read commands under these conditions
- Change the way internal events line up by changing PCI timing (for example, disable bus parking to cause consecutive transactions to be further apart in time).

## *Errata #5: SSM mode streaming DMA read performance*

Bug ID(s): 4394819, 4469864

**Symptom:**

An SSM system exhibits lower than expected streaming DMA read performance.

**Description:**

When in SSM mode, there is a bug that causes Schizo's streaming cache to limit itself to a single outstanding data fetch from memory at a time (instead of the usual maximum of two per page or four total outstanding fetches).

This effectively prevents the streaming cache from trying to stay two cachelines ahead of the PCI device. Because of the less aggressive prefetching, the system memory latency is more apparent to a PCI device, and the end result is significantly reduced streaming DMA read performance.

This is partially fixed in Schizo 2.3. Instead of always being limited to a single outstanding fetch, Schizo is limited from the end of any streaming cache sync operation (periodically performed by the nexus driver) to any subsequent streaming DMA write.

This is completely fixed in Schizo 2.4.

**Workaround:**

None

### *Errata #6: Streaming DMA read with master wait states may return bad data*

Bug ID(s): 4397653

**Symptom:**

A streaming DMA read crosses a page boundary and returns incorrect data.

**Description:**

Under a very limited set of circumstances, one of which is that a PCI device insert master wait states in the middle of an ongoing DMA transaction, Schizo can provide incorrect data for a streaming DMA read.

The full set of circumstances required is as follows:

- PCI device does a DMA read transaction to a streaming page
- The transaction is in 32-bit mode on the PCI bus, and the starting address of the transaction is not a multiple of 8
- The data phase for the address that is at offset 0x1ff8 in the page is part of the transaction, and this data phase is delayed by at least one master wait state
- The PCI device continues the transaction past the page boundary at offset 0x2000

Under these conditions, starting from when the device crosses into the new page until it disconnects, Schizo will deliver incorrect data (from an earlier cacheline).

This is fixed in Schizo 2.3.

**Workaround:**

There is no workaround for this problem other than to avoid the situation that causes it, which can only be done by a PCI device and/or its device driver. Ideally, this would be done by ensuring that a PCI device does not ever insert master wait states in the middle of a transaction. If that is not possible to guarantee, then the device should not use streaming DMA because of the potential of data corruption.

To date, no known PCI device tested by Sun has exhibited the master wait state behavior that could trigger this bug - the bug was only found during simulations, and not in lab testing of Schizo.

### *Errata #7: Streaming DMA write with empty data phases hangs PCI*

Bug ID(s): 4408474

**Symptom:**

PCI bus hangs on a DMA write with Schizo asserting DEVSEL but not TRDY.

**Description:**

Under certain conditions, a PCI device that starts a transaction with an empty data phase (a data phase in which no byte enables are asserted) may never see a TRDY from Schizo, which can cause the PCI bus to hang.

The full set of conditions required are as follows:

- PCI device is doing streaming mode DMA writes
- PCI device ends one DMA write transaction in the middle of a 64-byte cacheline
- The next DMA write transaction to that page has the following properties:
  - Starts on an address that is a multiple of 8
  - Is done in 32-bit mode
  - Starts in the same cacheline where the previous transaction left off
  - Has no byte enables asserted for the first data phase of the transaction

If all these conditions are met, Schizo will assert DEVSEL for the final DMA write, but it will not assert TRDY. If the device does not implement any sort of timeout, this will effectively hang the PCI bus in a busy state.

This is fixed in Schizo 2.3.

**Workaround:**

There is no workaround for this problem other than to avoid the situation that causes it. If this cannot be ensured by the PCI device doing the DMA transaction, the device driver should not use streaming mode for its DMA.

Devices that sit behind PCI to PCI bridges may have to follow additional rules to ensure that the bridge does not introduce a transaction beginning with an empty data phase.

### *Errata #8: Livelock when two DMA masters access same streaming page*

Bug ID(s): 4446551

**Symptom:**

Two consecutive PCI transactions to the same streaming DMA page alternate indefinitely without progress

**Description:**

Schizo is designed to perform special performance optimizations for DMA pages that are marked streaming. Performance on a streaming page works best when the actual DMA access pattern is “streaming” - that is, when the DMA transactions to that page access consecutively increasing addresses with no gaps, either within a transaction (via deassertion of PCI byte enables), or between transactions (by starting a new transaction somewhere other than where the previous one left off).

Normally, if the streaming guidelines are not followed, Schizo will still handle DMAs as expected, but with significantly poorer performance.

There is a scenario, however, where two (or more) DMA transactions can be presented to Schizo in a repeated fashion such that each transaction is always retried by Schizo, and never makes progress by having a data transfer.

The scenario requires two DMA reads that target the same streaming page, but which address different cachelines within that page. These two DMA read transactions could be from different devices on the same PCI bus, or they could conceivably be issued by

the same device. Both transactions must be “active” on the PCI bus simultaneous - that is, they must have been issued, and must continue to be reissued when Schizo terminates them with a retry, and they must alternate on the PCI bus.

The first DMA read will cause Schizo to fetch data from memory to satisfy the DMA. Prior to obtaining the data, Schizo will retry either of the DMA read transactions as often as they are issued on the PCI bus. After Schizo has obtained the data, if the second DMA transaction shows up on the PCI bus prior to the first one, Schizo will throw away the data for the first DMA read and fetch data from memory to satisfy the second DMA (streaming DMA transactions to the same page share a common data buffer).

This process can now be repeated: when data for the second DMA arrives in Schizo, if the first DMA is the next transaction seen by Schizo, the data will be thrown away and a new data fetch started.

If the transaction continue to be alternated without significant variations in timing, this process can be repeated indefinitely. Neither DMA transaction makes any progress, and this situation can be called a livelock.

There is no plan to fix this in any Schizo version.

**Workaround:**

The only real workarounds for this problem are to avoid the situations that cause it. In particular, if a single device needs to exhibit this type of behavior, it should do its DMA in consistent mode (at least for affected pages). If that is not advisable for performance reasons, the device will need to either keep retrying a single DMA read until data is transferred before moving to another section of the same streaming page, or it will need to manage the retries of the transactions in some other manner to avoid a pattern that causes indefinite livelock.

For the case where two separate devices are issuing the DMA reads, the same end result is required in order to work around the problem, but it is more difficult since the two devices probably can’t coordinate activities with each other. In most cases this is not a supported configuration (DMA addresses given to one device driver should not be shared with a different driver), and the best solution is to map the page in consistent mode.

## *Errata #9: DMA writes allowed to read-only streaming page*

Bug ID(s): 4448450

### **Symptom:**

A streaming DMA write is accepted by Schizo when the DMA address was mapped read-only.

### **Description:**

A device driver may request a DMA address to be mapped read-only. It does this by including the flag `DDI_DMA_WRITE` in a call to `ddi_dma_buf_bind_handle()` or `ddi_dma_addr_bind_handle()`. Note that `DDI_DMA_WRITE` is not a typo, and means that the direction of transfer is from memory to I/O. When this is done, Schizo will terminate any DMA writes that address that page with a PCI target abort.

Under rare conditions, Schizo can allow a DMA write to a streamable read-only page to proceed. The conditions required are that a DMA read to the same page must have recently been issued on the PCI bus (and retried exactly once so far by Schizo), that read must be the first access to this page (since the mapping was allocated), and one of Schizo's internal blocks must be busy servicing a PIO request.

There is no plan to fix this in any Schizo version.

### **Workaround:**

There is no workaround available for this problem. Devices that use read-only mappings must simply live with the very slightly reduced protection against programming and/or hardware faults.

## *Errata #10: PCI Target initial latency violated*

Bug ID(s): 4456145

### **Symptom:**

Schizo takes longer than 16 cycles to assert TRDY as a target.

### **Description:**

The PCI Specification (Revision 2.1) specifies in section 7.5.2 that for 66 MHz operation, a host bridge must respond (with either TRDY or STOP) within 16 cycles of FRAME assertion.

Under certain conditions of heavy load, Schizo will occasionally violate this requirement. Schizo may instead take up to 23 cycles to respond.

This is not expected to present a problem for several reasons:

- Occurrence should be infrequent enough that the extra cycles the bus is kept busy have a negligible effect on performance.
- Since a transaction can always be terminated by Schizo with a retry, the PCI master really has no guarantee on when data is actually going to be available, so this extra latency doesn't change how the master needs to behave in terms of latency.
- PCI devices are required to functionally support (possibly with performance degradation) devices with higher than 16 cycle initial latency (section 3.5.1.1)
- Even if this were to present a problem where a device fails because the system exceeds the maximum REQ to GNT latency it could tolerate, the system could be tuned via the PCI master latency timer registers in each device to reduce the maximum possible latency.

There is no plan to fix this in any Schizo version

**Workaround:**

None

### *Errata #11: PCI arbiter not robust for odd REQ/GNT behavior*

Bug ID(s): 4466942

**Symptom:**

PCI bus hangs - no transactions are seen even when devices wish to use the bus.

**Description:**

The PCI specification expects PCI devices to keep their REQ asserted until they have a chance to begin a transaction (unless they no longer wish to do a transaction). In addition, when they receive a GNT and an idle bus, devices are expected to begin their transaction within 16 clocks at the most. Both of these expectations are detailed in section 3.4.1 - Arbitration Signaling Protocol.

Schizo's PCI arbiter is not as robust as it could be when devices violate these expectations. There are a couple of REQ behaviors a device can exhibit which will cause Schizo to continuously grant the same device and stall any other progress on the PCI bus.

This behaviors depend on a feature of Schizo's arbiter where it raises the priority for devices that have been retried recently. There are two cases where this feature, along with a particular REQ behavior by a device, will lock up the bus. Both cases require the device to issue a previous transaction that is terminated in a retry by Schizo. Then, either:

1. The device repeatedly asserts REQ in a 3-cycle pattern (one or two cycles asserted, two or one cycle deasserted), without ever starting a transaction

If this is timed correctly, Schizo's arbiter will repeat the same three state sequence over and over. GNT will be asserted to the device for one cycle of each iteration, but because REQ is withdrawn, GNT is also withdrawn.

2. The device asserts REQ indefinitely without ever starting a transaction.

In this case, Schizo's arbiter will get into a longer 18 clock sequence over and over. Each iteration, GNT is asserted to the device for 16 clocks cycles. Note that a previously retried transaction is a prerequisite. A device that simply has a REQ stuck asserted but which has never done a transaction will not cause this behavior.

There is no plan to fix this in any Schizo version.

**Workaround:**

There are two possible workarounds. The feature in Schizo that increases priority for retried devices can be disabled (by setting DIS\_RTY\_ARB bit in PCI diagnostic register).

Also, with certain devices that exhibit behavior 1 above, disabling PCI bus parking can prevent the devices from getting into the loop that causes the problem.

## *Errata #12: Parity error on address phase of 64-bit wide PIO*

Bug ID(s): 4482600

**Symptom:**

System panics due to a device asserting SERR as a result of an address parity error caused by Schizo.

**Description:**

Version 2.1 of the PCI specification is somewhat misleading about the requirements for driving parity on the 64-bit extension of the bus during the address phase of a transaction.

The description of the parity signal PAR64, in section 2.2.9 on page 17 of the PCI specification, only requires that PAR64 be valid for transactions that assert REQ64 when they are using a Dual Address Cycle command. However, the description in section 3.10 on page 109 requires PAR64 to be valid during the address phase for any transaction using REQ64, regardless of the command type.

Since the requirements do not conflict, the correct thing to do is to use the stricter requirement from section 3.10 for generating PAR64. However, some versions of Schizo do not do this.

When Schizo as a master requests a 64-bit transaction, it fails to drive correct parity on PAR64 during the address phase. Any other 64-bit device on the PCI bus will see REQ64 asserted, and may detect and report the address parity error, typically by asserting SERR and causing a fatal system error.

In fact, Schizo does not drive the 64-bit extension at all during the address phase. This makes the actual behavior of a given transaction dependent on the previous state of the bus. If the bus was idle for a long period, pullups on the 64-bit extension will cause PAR64 to definitely provide incorrect parity. If a 64-bit transaction was recently completed, the pullups may not have changed the value on the 64-bit extension, and PAR64 may be correct.

Schizo requests a 64-bit transaction using REQ64 under the following conditions:

- Schizo is issuing a PIO (either read or write) to a device on the PCI bus
- The size of the PIO transaction (as received from the CPU or other device on the system interconnect) is greater than 8 bytes.

While many devices will typically use only PIOs of at most 8 bytes, and will thus not run into these conditions, graphics cards are known to do 64-byte block stores (often by passing a device address to a system routine that does 64-byte block stores).

The target of the PIO and the device that detects and reports the address parity error will not necessarily be the same. Also, the target of the PIO does not need to be a 64-bit PCI device in order for the problem to occur (the device reporting the error must, of course, be a 64-bit device in order to see the bad parity on PAR64).

This is fixed in Schizo 2.4.

**Workaround:**

One workaround for this problem is to have devices avoid checking parity on the 64-bit extension during the address phase of a transaction unless a Dual Address Cycle is used (which Schizo will never use as a master). Since the 64-bit extension carries no useful information under these conditions, it is safe to skip this parity check.

Some devices, however, may not be able to disable parity checking in this fashion. In that case, an alternate workaround is to reconfigure the PCI cards in the system so that devices which can receive 64-bit wide PIOs are not on the same bus as devices that check 64-bit parity during address phases.

Another workaround is to modify the driver of the device receiving 64-bit wide PIOs so that all accesses to it are 8 bytes or less (this may have an impact on performance).

### *Errata #13: Inappropriate PCI command used for 64-byte PIO reads*

Bug ID(s): 4488517

**Symptom:**

Schizo issues 64-byte PIO reads using PCI Memory Read command instead of Memory Read Line.

**Description:**

When Schizo issues a 64-byte PIO read operation on the PCI bus, it will use the PCI command Memory Read. While this is not illegal, it is a change from the previous Sun host bridge that Schizo is based on, which used a Memory Read Line command.

In addition, the PCI specification recommends using Memory Read Line for transactions of this size (more than one DWORD up to the next cacheline boundary).

Since this behavior is legal, there should be no functional problems. There may be a slight performance penalty for not using the most efficient PCI command, depending on the device that is the target of the PIO read.

There is no plan to fix this in any Schizo version.

**Workaround:**

None.

## *Errata #14: 64-bit parity errors not detected during PIO reads*

Bug ID(s): 4490597

**Symptom:**

Data corruption on a PIO read goes undetected which could lead to a wide variety of system behaviors.

**Description:**

When Schizo does 64-bit wide data transfers during a PIO read, it does not check parity on the upper 32-bits of the bus properly, and it will ignore parity errors on these signals.

See the description of Errata #10 for a discussion of the conditions needed for Schizo to issue a 64-bit wide PIO.

This is fixed in Schizo 2.4.

**Workaround:**

The only workaround is for the affected device driver to avoid PIO reads that will be issued in 64-bit wide mode on the PCI bus by Schizo.

## *Errata #15: PERR driven one cycle too long*

Bug ID(s): 4495878

**Symptom:**

An unreported parity error may cause unreliable system operation.

**Description:**

According to the PCI spec, two cycles after the last data phase of a transaction, the agent responsible for driving PERR drives it to a 1 or 0 depending on whether a parity error was detected in the last data phase. The following cycle, that agent actively drives PERR to a 1, and the cycle after that, it stops driving PERR.

The earliest the next device may drive PERR is the fifth cycle after the last data phase of the previous transaction.

This provides a clean turn-around cycle on PERR - no new device starts to drive on the same cycle another device is turning off its driver.

Due to a bug in Schizo, on all DMA writes, Schizo drives PERR for one extra cycle at the end of the transaction. This eliminates the normal turn-around cycle on PERR.

If the next transaction follows immediately on the bus, is a write, has a parity error, and the target of the transaction is a device that implements “fast” DEVSEL timing, then that target will be trying to assert PERR at the same time Schizo is turning off its PERR driver. Normally, this is still not a problem, but in a system with marginal timing, this could then result in the PERR assertion being missed.

Because the PERR assertion was missed, no asynchronous fault will be generated for the bad PIO write, and the system will continue to operate, possibly incorrectly, as if no error had occurred.

This is fixed in Schizo 2.4.

**Workaround:**

None

### *Errata #16: Enable for DMA write parity error interrupt*

Bug ID(s): 4723762

**Symptom:**

In Schizo 2.4 an interrupt was added to flag write DMA parity errors. When run in conjunction with a Sun GigaSwift Ethernet card (Cassini ASIC based), this interrupt caused the system to panic due to parity errors initiated by that card. This RFE adds an enable for this interrupt, defaulting it to disabled to be compatible with previous revs of Schizo 2.2/2.3. Once the GigaSwift Ethernet driver is modified to not generate parity errors, this interrupt can be re-enabled.

**Description:**

Hardware currently implemented in 2.5 as disabled upon reset. Uses idle check diagnostic port override bit (14).

\* DMAW parity error interrupt behavior - bit 14 of idle check

diagnostic register.

\* bit 14 = 0 => 2.3 style behavior, don't interrupt on DMA writes

with parity errors.

- \* bit 14 = 1 => 2.4 style behavior, interrupt on any DMA writes with parity errors.
- \* Power on reset value is 0.

This is fixed in Schizo 2.5.

**Workaround:**

- \* Update GigaSwift Ethernet driver to change cache line size to 64 bytes.
- \* Nexus to ignore interrupt for 2.4

### *Errata #17: Safari protocol error*

Bug ID(s): 4709566 (also XMITS bug 4708465)

**Symptom:**

A cache-coherence protocol violation could lead to possible system crashes or data corruption.

**Description:**

Under certain conditions, Schizo can fail to cancel a writeback transaction that should be cancelled due to a prior foreign invalidating command.

In the following sequence of events, Schizo does not cancel a writeback when it should.

*Table 0-1* Sequence of events for bug 4709566

<b>time</b>	<b>device</b>	<b>trans ID</b>	<b>cmd/addr</b>	<b>Notes</b>
m	Schizo	XX	RTO A	Prerequisite for the later WB
n	Dev X	ID1	RTO A	Must hit in Schizo's merge buffer

Table 0-1 Sequence of events for bug 4709566

time	device	trans ID	cmd/addr	Notes
n+1	Schizo	XX	RTO B	
n+2	Schizo	XX	WB A	Due to foreign RTO at time n, Schizo needs to cancel this
n+3	Any	ID1	ANY	TransID must match ID used at time n for bug to occur.

The failure to cancel the writeback could result in data corruption. It could also cause multiple devices to consider themselves the owner of a cacheline, which could later be detected and flagged as a fatal system error.

This is fixed in Schizo 2.5.

**Workaround:**

For Serengeti /StarCat machines the AR can limit consecutive grants which will prevent the above condition from occurring by guaranteeing Schizo can never drive back-to-back transactions.

## A.4 Internal (OS visible) Errata Table

Table A-3 Schizo Family Internal Errata Table

Errata	Schizo			ELE			Description
	2.1	2.2	2.3	2.4	2.5		
I-1	Feature	Feature	Feature	Feature	Feature	Feature	PIO writes to CSR complete out of order
I-2	Feature	Feature	Feature	Feature	Feature	Feature	Manufacturing Test Bug
I-3	✓	Fixed	Fixed	Fixed	Fixed	Fixed	Virtual to physical address translation corruption
I-4	✓	Fixed	Fixed	Fixed	Fixed	Fixed	HW/Manufacturing Test Bug
I-5	Feature	Feature	Feature	Feature	Feature	Feature	Streaming cache LRU update incorrect
I-6	✓	Fixed	Fixed	Fixed	Fixed	Fixed	Streaming cache prefetches incorrectly based on PCI read command
I-7	✓	Fixed	Fixed	Fixed	Fixed	Fixed	REQ64 not deasserted properly after software reset
I-8	✓	✓	Fixed	Fixed	Fixed	Fixed	Dynamic reconfiguration and PCI arb parking
I-9	Feature	Feature	Feature	Feature	Feature	Feature	Manufacturing Test Bug
I-10	✓	✓	Fixed	Fixed	Fixed	Fixed	Transaction ordering in SSM mode
I-11	Feature	Feature	Feature	Feature	Feature	Feature	Minor performance issues
I-12	Feature	Feature	Feature	Feature	Feature	Feature	Diagnostic register access to streaming cache
I-13	✓	✓	Fixed	Fixed	Fixed	Fixed	Consistent Sync/Flush operation
I-14	Feature	Feature	Feature	Feature	Feature	n/a	Thermal diode temperature offset
I-15	Feature	Feature	Feature	Feature	Feature	Feature	Manufacturing Test Bug
I-16	✓	✓	✓	Fixed	Fixed	Fixed	ERR_SLOT and ERR_SLOT_LOCK bits not working
I-17	Feature	Feature	Feature	Feature	Feature	Feature	Not all PCI address parity errors are reported
I-18	✓	✓	✓	Fixed	Fixed	Fixed	DMA write with parity error causes subsequent DMA corruption
I-19	✓	✓	✓	Fixed	Fixed	Fixed	Generate PCI interrupt for DMA write parity error

Table A-3 Schizo Family Internal Errata Table

Errata	Schizo					ELE	Description
	2.1	2.2	2.3	2.4	2.5		
I-20	n/a	n/a	✓	Fixed	Fixed	n/a	Manufacturing Test Bug
I-21	Feature	Feature	Feature	Feature	Feature	Feature	DMA writes with parity errors in passthru mode cause corruption
I-22	✓	✓	✓	✓	Fixed	Fixed	[Reclassified as External Errata #17]
I-23	Feature	Feature	Feature	Feature	Feature	Feature	System deadlocks when using consistent DMA flush/sync

✓ = Errata applies to this revision  
 n/a = Errata was not present in these versions  
 Fixed = Bug fixed or fix carried forward from previous version  
 Feature = Irregular feature that will not be changed in silicon

## A.5 Detailed Internal Errata Summary

### Errata I-1: PIO writes to CSR complete out of order

BugID(s): 4228656

**Symptom:**

Various effects, such as duplicate interrupts.

**Description:**

PIO writes to different subblocks in the PCI leaf will complete asynchronously. For example, clearing error status in the main PCI CSR register can occur after the clearing of associated MDU interrupt registers even if software issues the PIOs in the correct order. This can cause a false interrupt to be signalled.

There is no plan to fix this in any Schizo version.

**Workaround:**

Any ordering requirements of PIO writes between PCI subblocks must be managed by software. In the above instance, a PIO read of the PCI CSR must complete between the PCI CSR PIO write to clear the error and the MDU PIO write to clear the interrupt.

*Errata I-2: Not Applicable - Manufacturing Test Bug*

*Errata I-3: Virtual to physical address translation corruption*

BugID(s): 4333399

**Symptom:**

Possible data corruption. Also, PCI devices may get unexpected target aborts during DMA, and system may report Unmapped Safari address errors from Schizo.

**Description:**

When the interval timer for a consistent DMA read buffer expires, indicating a DTO, at the same time that the PCI device is retrying the transaction, the PCI virtual address gets sent as a physical address without IOMMU translation when making the DMA read request. The physical address will almost certainly be for the wrong page of memory, and may even be nonexistent.

Because of Errata #2, it is not actually necessary that a real DTO occur (i.e. false DTOs caused by Errata #2 can lead to bad physical addresses being used).

This is fixed in Schizo 2.2.

**Workaround:**

Schizo's PCI interval timer must be disabled to prevent DTOs from being reported so that this bug doesn't occur.

*Errata I-4: Not Applicable - HW/Manufacturing Test Bug*

*Errata I-5: Streaming cache LRU update incorrect*

BugID(s): 4341930

**Symptom:**

Theoretical minor performance impact.

**Description:**

The streaming cache least-recently-used entry logic (LRU) is not updated in all cases where it should be.

From the specification, the LRU should be updated (i.e. an entry should move to the end of the LRU list and become “most recently used”) whenever:

- That entry is allocated for a new page
- A PCI transaction hits that entry

The actual hardware behavior is that it will update the LRU:

- At the end of any write transaction where data is transferred
- For read transactions, most of the time when the STC issues a read request to memory (but not all the time: if the STC decides at the beginning of a read transaction that memory fetches are required, the first one will not cause an LRU update).

There is no data to suggest that the actual LRU update algorithm is better or worse than the intended algorithm.

There is no plan to fix this in any Schizo version.

**Workaround:**

None

### *Errata I-6: Streaming cache prefetches incorrectly based on PCI read command*

BugID(s): 4342004

**Symptom:**

Streaming DMA read performance is not as high as expected.

**Description:**

Schizo’s streaming cache uses the PCI read command type to determine how best to match prefetches with the master’s expected transfer behavior.

A design flaw causes the streaming cache to use the command from the previous transaction instead of the current one for this decision (non-read command types end up being interpreted as Memory Reads in this case).

There are a number of ways that this bug can manifest as performance problems:

1. If two devices on the PCI bus are alternating bus cycles, with one doing memory reads and the other doing memory read lines.
2. If two devices on the PCI bus are alternating bus cycles with one doing memory read lines and the other doing memory writes.
3. If a single device on the PCI bus has more than one active stream, and is alternating bus cycles as in #1 or #2.

If, however, there is only a single device active at a time, and it doesn't change command type frequently, then there is little to no impact.

When the streaming cache prefetch behavior is not matched to the PCI master's transfer characteristics, performance can suffer for one of two reasons:

1. Too much prefetching is done - the PCI master doesn't read full cache lines, and many prefetches will end up being thrown away.
2. Not enough prefetching is done - the streaming cache doesn't keep ahead of a PCI device that is transferring large bursts, and is forced to disconnect.

Quantitatively, simulations have shown that under worst case conditions (not expected in actual operation), overall bus utilization can be reduced by 9% with this bug, and a high-performance device's performance could be reduced by more than 20%.

This is fixed in Schizo 2.2.

**Workaround:**

None

*Errata I-7: REQ64 not deasserted properly after software reset*

BugID(s): 4344023

**Symptom:**

Theoretically, if a 32-bit PCI device did a DMA transaction soon enough after the reset, Schizo might mistake it for a 64-bit transaction request, causing data corruption.

**Description:**

On a 33MHz PCI bus, when a software reset of the PCI bus is performed, 50% of the time, Schizo will fail to drive the REQ64 signal high after reset is deasserted (Schizo drives it low during reset to signal that the PCI bus is 64 bits wide).

A required pullup on REQ64 eventually (within 100 cycles) will bring the signal to a valid deasserted level. Within this window, if a 32-bit master talks to a 64-bit target, the target may see REQ64 erroneously asserted.

This is fixed in Schizo 2.2.

**Workaround:**

Software must ensure that no DMA can take place within the 100 clock window after a soft reset of the PCI bus (it can disable arbitration in Schizo).

### *Errata I-8: Dynamic reconfiguration and PCI arb parking*

BugID(s): 4355941

**Symptom:**

System panics due to PCI SERR during dynamic reconfiguration operations.

**Description:**

For system dynamic reconfiguration (DR) to work, the PCI devices must not receive PCI bus grant while the address network is paused. Otherwise, the devices will retry the PCI cycles until their retry timer rolls over, and a PCI SERR would occur when this happens.

To support DR without requiring the system DR controller to access Schizo CSRs via the Safari bus, a "Halt PCI Arbitration" bit located in the PCI shadow scan chain register was added to Schizo 2.1.

Unfortunately, this function does not work properly in Schizo 2.2 if PCI bus parking is also enabled. With parking enabled, it is possible for a PCI device to still see its grant asserted after the Halt PCI Arbitration bit has been set, which can lead to excessive retries during the DR operation, and a PCI SERR.

This is fixed in Schizo 2.3.

**Workaround:**

Disable PCI bus parking.

## *Errata I-9: Not Applicable - Manufacturing Test Bug*

## *Errata I-10: Transaction ordering in SSM mode*

BugID(s): 4396517, 4400010, 4400254

**Symptom:**

SSM systems encounter data corruption (for DMA writes) due to incorrect enforcement of ordering constraints.

**Description:**

For SSM systems only, there are several ordering constraints that Schizo does not properly guarantee. In each case, it's a DMA write ordering constraint where Schizo should delay some operation until all previous DMA writes have fully completed. Instead, Schizo only delays until all previous DMA writes have issued (which is sufficient for non-SSM systems but not for SSM systems).

The ordering constraints affected are the following:

1. Streaming cache synchronization operation should guarantee that all previous streaming DMA writes have completed before setting synchronization flag.
2. Consistent mode synchronization operation should guarantee that all previous consistent DMA writes have completed before setting synchronization flag.
3. Interrupt operation should guarantee that all previous consistent DMA writes have completed before issuing interrupt packet.

Because of incorrect ordering, it's possible for incorrect (stale) data to be used by a device driver.

In order to actually see the problem, the SSM agent must actually reorder the relevant transactions issued by Schizo (and due to driver delays, one of the DMA writes must actually be delayed by a significant amount of time).

This is fixed in Schizo 2.3.

**Workaround:**

On SSM systems, software needs to put in waits after each of these synchronization events that are longer than the maximum "reorder delay" that can be seen by Schizo.

## *Errata I-11: Minor performance issues*

BugID(s): 4396524, 4404296

**Symptom:**

None

**Description:**

Two scenarios have been identified where Schizo's streaming cache does not behave exactly as anticipated, which leads to a very slight and very short term performance loss. Typically, the performance loss would not even be noticeable in a system. The only way to notice the scenario is either in simulation, or if Schizo's PCI and Safari buses are both being monitored with an analyzer.

Scenario 1 (bug 4404296) is as follows:

- PCI device 1 is doing streaming DMA (reads or writes)
- PCI device 2 is doing streaming DMA reads
- PCI device 1 does a transaction that ends at offset 0x1FC0 in its page
- PCI device 2 is the next device to do a streaming DMA

The result of this bug is that Schizo will not prefetch the data two cachelines ahead of PCI device 2's transaction at this point in time. It will still be correctly fetched later in time, however.

Scenario 2 (bug 4396524) is as follows:

- PCI device is doing streaming DMA reads
- PCI device uses Memory Read Line or Memory Read Line Multiple command for a transaction
- That transaction is in 32-bit mode and starts at an address that is not a multiple of 8
- That transaction continues to the end of the 64-byte cacheline

The result of this bug is that Schizo will incorrectly throw away a prefetch for data two cachelines ahead of the problem transaction. This will then have to be refetched at a later time.

There is no plan to fix this in any Schizo version.

**Workaround:**

There is no workaround for these behaviors, and none is required because the bugs responsible for the behaviors really have no discernible impact in a system.

### *Errata I-12: Diagnostic register access to streaming cache*

BugID(s): 4411508, 4421955

**Symptom:**

System panics due to timeout on a PIO read, or DMA data read corruption is observed.

**Description:**

Accessing diagnostic registers in Schizo's streaming cache block while DMA is active can occasionally cause one of two problems. Diagnostic PIO reads can cause Schizo to hang, leading to a system bus timeout, which will typically cause a machine panic. Other diagnostic PIO accesses (reads or writes) can cause corruption of streaming DMA read data.

There is no plan to fix this in any Schizo version.

**Workaround:**

The workaround for this is to avoid accessing these registers while DMA is active. This workaround is handled at the OS level in the Schizo nexus driver. There is no need to access these diagnostic registers in a normal system. They are mainly provided for power-on self-test use, so this workaround should have no impact on Schizo based systems, and is already in place.

### *Errata I-13: Consistent Sync/Flush operation*

BugID(s): 4422321, 4450040, 4454775

**Symptom:**

If nexus driver is doing consistent sync/flush operations, system may hang, or may behave erratically due to dropped/duplicated interrupts. If not doing consistent sync/flush operations, PCI devices behind PCI-PCI bridges may see data corruption due to ordering problems.

**Description:**

There are several problems with the operation of the consistent sync/flush operation in Schizo:

1. There is a possible deadlock if a sync/flush and an interrupt occur at the same time a PIO read of an interrupt register is done. This leads to a complete hang of Schizo.
2. A spurious duplicate interrupt can be sent if a flush/sync operation lines up properly with receipt of an interrupt signal.
3. An interrupt can fail to be delivered if a sync/flush operation follows it immediately, and the original interrupt is NACK'd by the target CPU.

This is fixed in Schizo 2.3.

**Workaround:**

In versions of Schizo with these bugs, consistent sync/flush operations should not be done. Some sort of delay should be used instead to minimize the possibility of improper ordering causing a problem.

### *Errata I-14: Thermal diode temperature offset*

BugID(s): 4428947

**Symptom:**

Schizo thermal diode reports an excessively high temperature.

**Description:**

There are two issues with the Schizo thermal diode:

First the diode has two inputs. When used with the Maxim MAX1617\_DXN part, only one is used. The second unused input (Z\_DIODE\_DDRV2) needs to be tied to the output Z\_DIODE\_DDRV0, or else leakage current gets introduced into the diode, causing a non linear reading from the maxim part at very high and low temperatures.

The second problem is a 25 degree offset that is due to the thermal diode in Schizo not being placed in the die close enough to the actual input/output pins of the package. This adds resistance to the input/output paths of the diode, affecting the measurement.

There is no plan to fix this in any Schizo version.

**Workaround:**

Make sure Z\_DIODE\_DDRV2 input is tied off properly, and subtract 25C from the readings obtained from the Schizo thermal diode.

*Errata I-15: Not Applicable - Manufacturing Test Bug**Errata I-16: ERR\_SLOT and ERR\_SLOT\_LOCK bits not working*

Bug ID(s): 4460200

**Symptom:**

Schizo's ERR\_SLOT bits do not properly indicate the owner of the PCI bus when an error occurred.

**Description:**

Instead of logging the current PCI master device when a PCI related error occurs, Schizo logs the device that is currently receiving a grant. When multiple requests are active this can cause Schizo to log an incorrect and misleading value into ERR\_SLOT.

There is also a related problem with the ERR\_SLOT\_LOCK bit, but this bit's functionality is redundant, and starting with Schizo 2.4 it no longer exists. The ERR\_SLOT field is locked from the time an error occurs until all PCI error status bits are cleared by software.

This is fixed in Schizo 2.4.

**Workaround:**

The value of ERR\_SLOT should be treated as potentially incorrect. Also, software should always retrieve the value of ERR\_SLOT before clearing any error status.

*Errata I-17: Not all PCI address parity errors are reported*

Bug ID(s): 4491685

**Symptom:**

A PCI device may report unexpected master aborts. A PCI analyzer may report address parity errors that the system does not flag.

**Description:**

Schizo does not report all possible address parity errors that occur on the PCI bus. Instead, it only reports parity errors when the address that it sees (which is possibly corrupt), appears to be a DMA address.

The PCI spec says that Schizo may signal address parity errors under various conditions, but does not require it. However, it was originally intended that Schizo would report all address parity errors.

Other devices may detect the same parity error and signal it via SERR. If Schizo was the intended target, the transaction should end up being master-aborted on the PCI bus, which the master presumably will report as an error. If some other device does claim the transaction, that device should signal the address parity error (multiple errors would have to be assumed for an incorrect device to both claim the transaction and not see an address parity error).

There is no plan to fix this in any Schizo version.

**Workaround:**

None

### *Errata I-18: DMA write with parity error causes subsequent DMA corruption*

Bug ID(s): 4495539

**Symptom:**

Data corruption on consistent mode DMA transfers.

**Description:**

If the STOP\_DATA bit in Schizo's PCI Diagnostic register is set to 1, Schizo is supposed to drop consistent mode DMA write transactions with parity errors instead of writing the data to memory. Due to a bug in the way this is handled, under certain conditions, Schizo's internal state can get corrupted, leading to data corruption (e.g. data written to incorrect address).

Possible enabling conditions include heavy consistent DMA write traffic, as well as the use of fast back-to-back PCI transactions for consistent DMA writes.

This is fixed in Schizo 2.4.

**Workaround:**

The STOP\_DATA bit in Schizo's PCI Diagnostic register (previously named DIS\_BADECC) should be set to 0 for versions of Schizo that have this bug.

### *Errata I-19: Generate interrupt for DMA write parity error*

Bug ID(s): 4500920

**Symptom:**

Data corruption on DMA writes.

**Description:**

Originally Schizo was designed to leave the reporting of parity errors during DMA to the PCI master (as dictated by the PCI spec). This was felt to be unwise for RAS reasons in case a PCI device neglected to respond correctly to a parity error, so starting with Schizo 2.4, in addition to anything the PCI master might do, Schizo will generate its own PCI interrupt when a parity error is detected during a DMA write. The new interrupt is identified by the status bit DMA\_WR\_PERR in the PCI Control/Status register (which was formerly the unnecessary bit ERR\_SLOT\_LOCK).

This is fixed in Schizo 2.4.

**Workaround:**

None. Prior to Schizo 2.4, if a PCI master behaves improperly with respect to parity errors on DMA writes, it is possible to get data corruption.

### *Errata I-20: Not Applicable - Manufacturing Test Bug*

### *Errata I-21: DMA write with parity error causes subsequent DMA corruption*

Bug ID(s): 4642710

**Symptom:**

Data corruption on consistent mode DMA transfers if IOMMU passthru mode is used.

**Description:**

This is related to Errata I-18 (a special case that was not covered by the fix for Errata I-18).

If the STOP\_DATA bit in Schizo's PCI Diagnostic register is set to 1, Schizo is supposed to drop consistent mode DMA write transactions with parity errors instead of writing the data to memory. Due to a bug in the way this is handled, under certain conditions, Schizo's internal state can get corrupted, leading to data corruption (e.g. data written to incorrect address).

All of the following conditions must be true for the bug to occur:

- Schizo must be in IOMMU passthru mode (i.e. IOMMU disabled)
- Schizo must receive 2 fast back-to-back DMA transactions on the PCI bus
- The first of these transactions must be a DMA write with a parity error
- The PCI diag bit STOP\_DATA must be set to 1

Under these conditions, the second DMA transaction uses the physical address of the first DMA, leading to potential data corruption.

There is no plan to fix this in any Schizo version.

**Workaround:**

IOMMU passthru mode is a legacy feature that is not used or needed. It should not be used - the IOMMU should be enabled whenever DMA is being performed.

Should passthru mode ever be used, the STOP\_DATA bit in Schizo's PCI Diagnostic register must be set to 0 to avoid this bug.

### *Errata I-22: Safari protocol error*

This has been reassigned as External Errata 17. To prevent confusion, I-22 won't be reused.

### *Errata I-23: System deadlocks when using consistent DMA flush/sync*

Bug ID(s): 4897386

**Symptom:**

Non-cacheable PIO queue timeouts (NCPQ\_TO) reported by the CPU, leading to fatal reset or dstop, depending on platform.

**Description:**

The following sequence is an example of the events required to cause this deadlock. Some variations on this sequence can also lead to the problem.

- *PCIIntA*: Schizo receives a PCI interrupt, and queues it up for the Safari bus, targeting *cpu0*
- *CDMASync1*: A write to Schizo's consistent DMA flush/sync register is issued on Safari
- *PIOWrite1*: An additional PIO write or read targeting an interrupt register in the same PCI core is issued on Safari
- *PIORead1*: A PIO read from *cpu0* to the same PCI core (can either be to the PCI bus, or a Schizo internal PCI register) is issued on Safari
- *IntB*: An interrupt targeting *cpu0* (but not *PCIIntA*) appears on Safari
- *IntA*: Schizo's interrupt corresponding to *PCIIntA* appears on Safari

The Safari transactions mentioned above do not have to be in consecutive cycles, but do have to appear in the indicated order.

It can lead to a deadlock as follows:

- Because of the design of the interrupt block in Schizo, the *CDMASync1* write will not complete internally until any interrupt that has already been queued has been ACKed (this is the main architectural flaw that introduces an unnecessary dependency between interrupts and PIO completion). *PCIIntA* has already been queued in this scenario.
- The subsequent *PIOWrite1* targeting an interrupt register will be stalled in Schizo's PCI core until the interrupt block processes *CDMASync1*.
- The PCI core handles PIO operations in order. Since *PIOWrite1* is stalled, the subsequent *PIORead1* will also not be processed.
- While waiting for an outstanding load to complete (*PIORead1*), *cpu0* cannot process new interrupts.
- The first interrupt targeting *cpu0*, *IntB*, can be stored and ACKed, but until *cpu0* begins processing new interrupts, subsequent interrupts, including *IntA* will be NACKed.

- Schizo is waiting for `cpu0` to ACK `IntA` before it will complete `PIORead1` from `cpu0`; `cpu0` is waiting for the response to `PIORead1` before it will ACK `IntA`. Deadlock.

Due to Errata I-13, the consistent DMA flush/sync operation cannot be used prior to Schizo version 2.3, so this bug cannot occur in those versions, although the underlying hardware problem still exists. The bug can be observed in all Schizo versions starting with 2.3.

There is currently no plan to fix this in any Schizo version.

**Workaround:**

The following sequence of events needs to be used as a replacement for the consistent mode DMA flush/sync operation in order to avoid the bug scenario:

Instead of doing a PIO write to the consistent DMA flush/sync register, do a PIO write to the interrupt "clear" register of an unused interrupt, setting it to the received state and causing Schizo to issue a dummy interrupt. Waiting for the interrupt to be received by a CPU is then equivalent to waiting for the sync flag to be set in memory.

Two spare interrupts that are unused across all Sun platforms have been identified for this purpose: INOs 53 and 54 (decimal). The Schizo nexus driver has been modified to use this alternate sync operation with these spare INOs (one for each PCI leaf, now referred to as ACDMA and BCDMA interrupts). The following kernel patch levels are needed for the Schizo nexus driver with the workaround:

- Solaris 10: FCS version
- Solaris 9: 112233-13
- Solaris 8: 117000-03

