

# Troubleshooting Guide

*For Watchdog Resets*



THE NETWORK IS THE COMPUTER™

SunService Division  
A Sun Microsystems, Inc. Business  
901 San Antonio Road  
Palo Alto, CA 94303 USA  
415 960-1300 fax 415 969-9131

Part No.: 805-3497-01  
September 1997

# Contents

---

<b>1. Watchdog Resets</b> .....	<b>1</b>
What is a Watchdog Reset? .....	1
What is a System Watchdog Reset? .....	2
What Causes Watchdog Resets? .....	3
Diagnostic Techniques .....	3
Patches Which Address Watchdog Resets .....	4
Specials: Debug Kernels & Bad Modules .....	6
References on SunSolve .....	7
Infodoc 14269 .....	7
Infodoc 11834 .....	7
Infodoc 14133 .....	7
SRDB 3242 .....	8
SRDB 4692 .....	8
SRDB 4893 .....	8
SRDB 5209 .....	8
Infodoc 2025 .....	8
Recommended Reading .....	9
<b>2. Error Management</b> .....	<b>11</b>
Error Reporting Mechanisms .....	11

Bus Errors	11
Types of Bus Errors	11
Bus Errors on Prefetch Operations	12
Bus Errors on Instruction Fetches	12
Bus Errors on Data Loads	12
Bus Errors on Synchronous Data Stores	13
Bus Errors on Asynchronous Data Stores	13
Bus Errors on Block Copy Operations	14
Bus Errors on MMU TLB Operations	14
Bus Errors on SBus DVMA Read Operations	14
Bus Errors on SBus DVMA Write Operations	15
Interrupts	15
Local Level-15 Interrupt	15
Broadcast Level-15 Interrupt	15
Resets	15
CPU Watchdog Reset	16
System Watchdog Reset	16
Types of errors	16
Software Errors	16
Hardware-corrected Errors	16
Memory Correctable ECC Error	16
Recoverable Errors	17
XDBusTime-out	17
XDBus Rejection	18
Memory Uncorrectable ECC Error	18
External Cache Parity Error	18
Processor Write Parity Error	19
DeMap Time-out	19

SBus Time-outs	20
SBus PTE Errors	20
SBus Rejection	20
SBusParity Errors	20
SBus External Page Table Parity Error	21
Fatal Errors	21
XDBus Parity Error	22
XDBus Arbitration Parity Error	22
XDBus Arbitration Time-out	22
XBus ParityError	22
Cache Consistency Errors	23
WriteSingle Time-out	23
Multiple DeMaps	24
Device-specific Errors	24
Critical Errors	24
AC Failure	24
Temperature Warning	24
Fan Failure	25
DC Failure	25
<b>3. General and Hard Hangs</b>	<b>27</b>
Troubleshooting System Hangs	27
Checking for a Resource Deprivation Hang	28
CPU Power	28
Virtual Memory	28
Physical Memory	29
Kernel Memory	29
Disk I/O	30
Generating Core Files	30

Analyzing System Hang Core Files .....	30
Kernal Problems .....	32
2.4 Deadman Kernels .....	32
Testing .....	32
Usage .....	33
Warnings .....	33
Availability .....	33
2.4 Debug Kernels .....	34
Testing .....	34
Warnings .....	34
Availability .....	34
2.5 Deadman Kernels .....	34
Testing .....	35
Usage .....	35
<b>4. Acronyms .....</b>	<b>37</b>
Sun4d Prtdiag Acronyms .....	37
IOC Error Messages .....	38
MQH Errors .....	39
BW Errors .....	39
Acronyms Used to Report Data from Cache Controllers .....	41
<b>5. Troubleshooting .....</b>	<b>43</b>
<b>6. Troubleshooting Commands .....</b>	<b>49</b>
wd-dump(watchdog dump) .....	50
.registers .....	52
.locals .....	52
ctrace .....	53
.psr (Process Status Register) .....	54

Summary .....	54
<b>7. Dragon (1000/2000) Hard Hangs .....</b>	<b>55</b>
<b>8. Setting Up and Using TIP .....</b>	<b>61</b>
Common Problems with TIP .....	63



# Watchdog Resets

---

Watchdog Resets are probably the most difficult computer failure to diagnose. They can be caused by both hardware and software. They also leave the system in a state which limits the amount of investigative work that can be done. Forced crash dumps are not guaranteed to prove anything. In general, Watchdog Resets are a nuisance!

Topics addressed below:

- What is a Watchdog Reset?
- What is a System Watchdog Reset?
- What Causes Watchdog Resets?
- Diagnostic Techniques
- Patches Which Address Watchdog Resets
- Specials: Debug Kernels & Bad Modules
- References on SunSolve
- Recommended Reading

---

## What is a Watchdog Reset?

On SPARC Version 8 systems, the ET (Enable Traps) bit of the Processor Status Register, or PSR, is cleared (set to 0) whenever a trap is being handled. As described in the “The SPARC Architecture Manual: Version 8,” page 74:

"While ET = 0

- Interrupting traps cannot occur, and all interrupt requests are ignored, even if bp\_IRL (Interrupt Request Level) = 15.

- If a precise trap occurs, or if there is an attempt to execute an instruction that can invoke a deferred trap and there is a pending deferred-trap exception, the processor halts execution and enters the `error_mode` state.
- If a deferred trap occurs which was caused by an instruction that began execution while `ET = 0`, the deferred trap causes the processor to halt execution and enter the `error_mode` state.
- Any deferred-trap exception that was caused by an instruction that began execution while `ET = 1` is ignored."

"What occurs after `error_mode` is entered is implementation-dependent; typically the processor triggers an external reset, causing a reset trap."

`Error_mode` is implemented on Sun systems as a *Watchdog Reset*.

On SPARC Version 9 systems, a similar methodology is used. This involves use of a new Processor State Register, `PSTATE`. The `PSTATE_RED` bit is used in much the same way that the V8 PSR `ET` flag is used. However, there's more.

V9 hardware has more than one trap register, one for each "trap level", with a minimum of four trap levels. The processor enters `error_state` when a trap occurs while the processor is already at its maximum supported trap level, that is, when the `TL` (Trap Level) = `MAXTL` (Maximum Trap Level).

What does a Watchdog Reset look like when it occurs? Something like this will be seen on the system console:

```

Watchdog Reset
Type help for more information
<#2> ok

```

The `<#2>` shows which CPU encountered the Watchdog Reset. On some systems, this may be shown as `{2}` instead.

---

## What is a System Watchdog Reset?

A System Watchdog Reset is a special condition where the system hardware identifies a situation which is "fatal". The write up in the Sun4D Technical Manual helps clarify the difference:

### 7.1.3.1 CPU Watchdog Reset

*A CPU watchdog reset is initiated when a trap condition occurs while traps are disabled and the MMU Control register NF bit is not set. The CPU branches to the instruction at physical address 0xF F000 0000. The WD bit in the CC Reset register is set to 1 on a watchdog reset.*

*A watchdog reset affects only one CPU and has no other effect on the system.*

#### *7.1.3.2 System Watchdog Reset*

*When a fatal error is detected, a system watchdog reset is initiated. A system watchdog reset affects all CPUs and I/O devices. Writes in progress may be lost, but the state of main memory is not altered (main memory continues to be refreshed after a system watchdog reset).*

When a System Watchdog Reset occurs, the system will drop down into POST diagnostics. Forget about getting a crash dump. It won't be any of help, even if you did succeed in getting one.

System Watchdog Resets are caused by hardware faults.... 99.999% of the time.

---

## What Causes Watchdog Resets?

Unfortunately, Watchdog Resets can be triggered by bad hardware and/or bad software. As you can imagine, diagnostic work is needed to figure out what caused the Watchdog Reset.

The SMCC European CTE team provided me with a loadable module which causes Watchdog Resets. The method used in their code to trigger the event is straight forward. They directly manipulate the PSR ET bit, then take a trap. Boom! You've got a Watchdog Reset caused by software.

However, faulty hardware can trigger all sorts of problems. This includes panics in perfectly good code, hard hangs, and of course, Watchdog Resets.

---

## Diagnostic Techniques

Watchdog Resets present their biggest challenge in that the best diagnostic information is collected from the Open Boot Prom, an area not frequented by most engineers.

The OBP commands dump out rather raw data, which then must be interpreted. Some of the data may have to be checked against the live system after the next reboot. Other data will have to be compared to information in the appropriate hardware (CPU) architecture manuals.

To help get less "raw" information, the `obpsym` (Open Boot Prom Symbols) module should be loaded up \*prior\* to the next Watchdog Reset. On Solaris 2.5+, this is done by forceloading module `misc/obpsym` in `/etc/system`. On earlier OS's, a special "obpsym" module provided by CTE will have to be loaded onto the system.

Also, when a system is encountering Watchdog Resets, it is a good idea to set the OBP variable "watchdog-reboot?" to false, to prevent automatic reboots.

Occasionally crash dumps can be forced after a Watchdog Reset. The value of the image, however, is not as dependable as one created due to a real software panic. However, sometimes good information can be pulled from the crash dump files.

---

**Note** – Attempting to force a crash dump should only be done after the OBP level diagnostic commands have been completed. Why? Because forcing a crash radically changes the state of the system.

---

Specific diagnostic commands can be found in section six.

---

## Patches Which Address Watchdog Resets

Watchdog Resets can be caused by software problems. Some of those corrected by patches are listed here. This is probably not the full list, but is a good start. It was initially generated by searching for "Watchdog" in the patch readme files, and then updated from there.

SunOS 5.5.1 (Solaris 2.5.1)

- 103640... Kernel patch

SunOS 5.4 (Solaris 2.4)

- 101945... Kernel patch

SunOS 5.3 (Solaris 2.3)

- 101318... Kernel patch
- 101836... SS5 Audio Jumbo Patch, for Solaris 2.3, Edition II
- 102182... SS5 Audio Jumbo Patch, for Solaris 2.3, Edition III (HW 5/94)
- 102193... SS5 Audio Jumbo Patch, for Solaris 2.3, Edition IV (HW 8/94)

- 103479... Point patch for Solaris 2.3 sd driver

#### SunOS 5.4 A+ Edition 1.0

- 101777... Amdahl A+ kernel patch

#### SunOS 5.2 (Solaris 2.2)

- 100999... Kernel patch

#### SunOS 5.1 (Solaris 2.1)

- 100884... Kernel and NFS patch
- 100938... fd problems with status return and boot failures on clones
- 100987... Fixes to kernel/drv/cg6

#### SunOS 4.1.4 (Solaris 1.1.2)

- 104177... System Watchdog Resets in sendsig

#### SunOS 4.1.3\_U1 (Solaris 1.1.1)

- 101443... esp: SCSI Errors - "ILLEGAL bit set" Watchdog Reset
- 101508... Sun4m kernel patch
- 104176... System Watchdog Resets in sendsig

#### SunOS 4.1.3 / 4.1.3C (Solaris 1.1)

- 100726... sun4m jumbo patch for kernel performance and memory bugs
- 101245... Merging SES/B related and "illegal bit" fixes
- 101408... SS10-51 or SS600-51 may hard hang or Watchdog Reset
- 101603... panics and Watchdog Resets when using aioread/aiocancel on NFS file
- 102161... sun4m audio patch for SS5

#### SunOS 4.1.2 (Solaris 1.0)

- 100542... IPI-Galaxy jumbo patch; sun4m patch for Watchdog Reset
- 100575... MP machines do not perform as well as 4/4XX equivalent

#### SunOS 4.1.1

- 100232... Sparcstation 2 crashes or whatchdog resets
- 100275... VME master bus accesses can time out during heavy IO
- 100330... Kernelmap causing system hangs or running out of mbuf

#### SunOS 4.1

- 100076... Pages stolen after Copy On Write

#### SunOS 4.0.3

- 100017... Breakpoints in kadb cause Watchdog Resets
- 100020... Numerous streams and serial I-O fixes
- 100298... sun4/490 only: VME master bus accesses can time out underheavy IO

#### CMW 1.0

- 101058... Watchdog Resets on system running 4.1.1 with CMW 1.0  
DBE 1.2; IDD 1.0; ODS 1.0
- 100614... sun4m jumbo patch for Watchdog Reset (for SunOS 4.1.2)  
IDP 1.0
- 100727... sun4m jumbo patch (for SunOS 4.1.2)  
FDDI 1.1
- 100319... Watchdog Reset in Sun4/490 FDDI->Ethernet router  
Online DiskSuite 3.0
- 103926... POINT PATCH Watchdog Reset from ERROR\_MODE\_RESET (for Solaris 2.3, 2.4, 2.5)  
Solstice DiskSuite 4.0
- 102580... Jumbo patch (for Solaris 2.3 and higher)  
SPARCstorage Array 2.0
- 103290... Point patch for Solaris 2.4 11/94, HW395  
SPARCstorage Array 1.0, 2.0, 2.1, 2.1
- 103351... Jumbo patch for Solaris 2.3  
TCX/XGL
- 101923... TCX Jumbo patch (for Solaris 2.4)

Again, this is not the definitive list of patches which address Watchdog Resets, but it's a good starting point. I'll endeavor to keep this list up-to-date as I get more information. Let me know if you know of other patches which should be added to this list.

---

## Specials: Debug Kernels & Bad Modules

Members of the SMCC European CTE team have lots of interesting technical goodies available on the net. Specifically, I recommend you check out Chris Gerhard's home page. Not only will you find some debug and deadman kernels which will help diagnosis system problems, he's got some modules which will break systems, ... obviously for educational purposes only!

Any time you can spend practicing diagnostic skills will help you later on when you've got a customer on the phone with a dead system.

---

## References on SunSolve

There are lots of references to Watchdog Resets throughout the SunSolve databases. Here are some snippets and summaries of what you can find if you have a dig around. (Yes, some of these are rather old)

### Infodoc 14269

*52) What causes a Watchdog Reset? How can it be distinguished from a panic?*

A Watchdog Reset is when a system takes a trap in the middle of handling another trap, while it has its trap handling disabled. There is a small window, immediately after the system takes a trap, when initial processing must be done and another trap cannot be accepted. The system cannot do anything with the new trap because trap handling is disabled, so the system quits.

Watchdog resets leave the system at the OK prompt without panicking or syncing the file systems. Typing "sync" can be done; the likelihood of it working is small. One can issue a few commands to dump registers and state to the screen (so they can be copied down onto paper), or just reboot the machine.

### Infodoc 11834

*What is a Watchdog Reset?*

A Watchdog Reset is an unrecoverable situation that forces the CPU to reset. It is caused as a result of the machine trapping while handling a trap with the "Enable Traps" bit in the Processor Status Register (PSR) being disabled. The reason traps have been disabled is that no other traps should occur until the first trap has been handled. But because a second trap has occurred and the cpu cannot handle it the machine resets.

Typically this indicates a software problem, but may be hardware related.

### Infodoc 14133

*The system did a Watchdog Reset.*

Watchdog Resets are usually hardware related. They occur when a processor gets a second trap while in the middle of initial processing of a first trap, during the period when trap handling is disabled. During this period, the system does not know what to do with the second trap, so it just stops. Software can cause this, but most Solaris

bugs in this vain have been fixed. A problematic piece of hardware, on the other hand, could cause spurious traps to be sent, some in the middle of processing of legitimate traps.

*The system leaves no messages when it reboots.*

This could be a Watchdog Reset on a system with its obprom "watchdog-reboot?" flag set to true.

### **SRDB 3242**

Watchdog resets can be seen on 4/490 and 4/470 systems running 4.1.1 and FDDI. A workaround is provided in the SRDB.

### **SRDB 4692**

Watchdog resets can occur during diskless boot up if the swap device file selected points to a non-existent or incorrect location.

### **SRDB 4893**

Watchdog reset window underflows can occur on ELC systems running 4.1.1. Fixed by patch 100323-02.

### **SRDB 5209**

Mixing 4mb SIMMs with 1mb SIMMs incorrectly on 4.1.1 systems can cause Watchdog Resets and window underflow errors.

### **Infodoc 2025**

A bug in the sun4c cache chip could cause Watchdog Resets on SunOS 4.1.1. Fixed by patch 100232. Corrected in SunOS 4.1.2.

---

## Recommended Reading

Check out:

- The SPARC Architecture Manuals (V8 and V9)
- The OpenBoot Command Reference Manual



## Error Management

---

This chapter describes the errors that can occur in Sun-4D systems and how they are handled. Errors which are strictly internal to the processor are not described in detail here. Refer to "The SPARC Architecture Manual", Version 8, Sun Microsystems, November 1989 for details.

---

## Error Reporting Mechanisms

This section describes how the different errors are reported to the programmer by the processor.

### Bus Errors

Bus errors are issued to the processor when the processor does a reference to virtual space or physical space which cannot be satisfied for hardware reasons (this excludes MMU protection violations, for example). This section describes how bus errors are handled by the processor and DVMA devices.

### Types of Bus Errors

In Sun-4D systems, there are four codes available for bus errors:

TO: Time Out; this code is returned when the addressed location does not return any answer after some fixed amount of time. BE: Bus Error; this code is returned when the addressed location rejects the required action because it is illegal. UC:

Uncorrectable Error; this code is returned when the addressed location rejects the required action because of an internal failure. UD: Undefined Error [7]); this code is for certain hardware errors.

## Bus Errors on Prefetch Operations

If a bus error is notified on a prefetch operation, it is completely ignored by the processor.

A prefetch is a bus operation which is not explicitly required by the executing program. The cases for prefetch by the processor unit in Sun-4D systems are:

- prefetch by the processor into the internal instruction or data caches.
- prefetch by the CC into the external cache.
- instruction fetches performed before it is known whether the instruction will be executed (an instruction which has been fetched may be discarded due to an intervening branch or trap).

## Bus Errors on Instruction Fetches

An instruction fetch accesses an instruction which will be executed. If a bus error is notified on an instruction fetch, an `instruction_access_exception` trap will be taken by the CPU when that instruction reaches the execution stage.

The Processor Fault Status register will contain the bus error code in the (UD, UC, TO, BE) bits, FT=5, AT=2/3, FAV=0. The virtual address of the error is the error PC, which is saved in register 11 by normal trap processing. The Processor Fault Address register is not updated.

## Bus Errors on Data Loads

A data load is issued by any load instruction when it is executed. If a bus error is notified during a data load, a `data_access_exception` trap will be taken by the CPU.

The Processor Fault Status register will contain the bus error code in the (UD, UC, TO, BE) bits, FT=5, AT=0-3, FAV=1. If the access was through an ASI other than the MMU-related ASIs (0x08-0x0B, 0x20-0x2F), the CS bit in the Processor Fault Status register will be set. The virtual address of the error is saved in the Processor Fault Address register.

## Bus Errors on Synchronous Data Stores

A synchronous data store is issued by the `ldstwb`, `ldstuba`, `swpb` and `swapa` instructions to any ASI, or `stb` instructions to ASIs other than `0x08-0x0B` and `0x20-0x2F` (for ASIs `0x08-0x0B` and `0x20-0x2F`, see "Bus Errors on Asynchronous Data Stores"). If a bus error is notified during a synchronous data store, a `data_access_exception` trap will be taken by the CPU.

The Processor Fault Status register will contain the bus error code in the (UD, UC, TO, BE) bits, FT=5, AT=4-7, FAV=1. If the access was through an ASI other than the MMU-related ASIs (`0x08-0x0B`, `0x20-0x2F`), the CS bit in the Processor Fault Status register will be set. The virtual address of the error is saved in the Processor Fault Address register.

## Bus Errors on Asynchronous Data Stores

An asynchronous data store is issued by any store instruction except `ldstwb`, `ldstuba`, `swpb` and `swapa` to any ASI, and `stb` to an ASI other than `0x08-0x0B` and `0x20-0x2F` (stores to those ASIs use the MMU to access physical space and are handled like normal `st` instruction). The effect of bus errors on asynchronous data stores depends on whether this is an early or late error, and on the state of the processor store buffer.

An early error is an error which is notified to the processor before the bus operation is effectively launched by the CC. There are two cases of early store errors:

- Memory store which misses in the external cache and for which the cache receives a bus error indication from memory when it tries to load the missing cache block.
- Processor bus parity error (see Section 7.2.3.5, "Processor Write Parity Error").

A late error is an error which is notified to the processor after the operation has been launched (and acknowledged to the processor by the CC). Late errors occur only on stores to I/O space.

If the processor store buffer is disabled and an early error is notified, it is handled as for a synchronous data store.

If the processor store buffer is enabled and an early error is notified, the CPU will take a `data_store_error` trap. The SB bit of the Processor Fault Status register will be set, but the bus error code is not logged. The Processor Fault Address register is not updated.

If a late error is notified (independently of the store buffer state), a level-15 interrupt will be issued to the CPU. The CC Error register will contain the error information, with the AE bit set to one and the DCmd subfield of CCOP set to 0100112.

Note that the value of the MC bit of the CC Control register does not affect error handling of store operations.

## Bus Errors on Block Copy Operations

Block copy operations are initiated by using the CC Stream registers. If a bus error is notified during a block copy operation, a level-15 interrupt will be issued to the CPU. The CC Error register will contain the error information, with the AE bit set.

Note that block copy errors may be differentiated from late asynchronous store errors by the value logged in subfield DCmd of field CCOP in the CC Error register, which is 0001012 or 0101012 for Block\_Load errors, 0011112 or 0101112 for Block\_Store errors, and 0100112 for asynchronous store errors.

## Bus Errors on MMU TLB Operations

An MMU TLB operation is a memory reference issued by the processor MMU to load entries into its TLB or to set the Modified or Referenced bits of a PTE. If a bus error is notified during an MMU TLB operation, the CPU will take either an instruction\_access\_exception trap or a data\_access\_exception trap depending on whether the operation was initiated using ASIs 0x08-0x09 (including instruction fetch) or any other ASI (including normal load and store operations).

- If an instruction\_access\_exception trap is taken, the Processor Fault Status register will contain the bus error code in the (UD, UC, TO, BE) bits, FT=4, FAV=0. The virtual address of the error is the error PC, which is saved in register 11 by normal trap processing. The MMU FAR is not updated.

If a data\_access\_exception trap is taken, the Processor Fault Status register will contain the bus error code in the (UD, UC, TO, BE) bits, FT=4, FAV=1. If the access was through an ASI other than the translation ASIs (0x8-0xB, 0x20-0x2F), the CS bit in the Processor Fault Status register will be set. The virtual address of the error is saved in the MMU FAR.

## Bus Errors on SBus DVMA Read Operations

When a bus error is signaled in response to a DVMA read operation (i.e. transfer from main memory to the I/O device), the SBus device receives an SBus ErrAck indication (Refer to SBus Specification, Rev. B0, Sun Microsystems, 1990, for details on SBus). The behavior is the same in consistent or stream mode.

The device should then terminate the DVMA operation and indicate an error condition to the driver. This behavior is device-dependent and is not specified by the Sun-4D system architecture.

## Bus Errors on SBus DVMA Write Operations

When a bus error is signaled in response to a DVMA write operation (i.e. transfer from the I/O device to main memory), the SBus device does not receive an SBus error indication as the write operation is acknowledged before being performed. Instead, the WEC or WES bit in the Slot Configuration register for the corresponding SBus device will be set. WEC is used if the error occurred during a consistent mode access, WES is used if the error occurred during a stream mode access. It is the driver's responsibility to check the status of this bit after a DVMA transfer (typically when the buffer is demapped from the I/O page tables).

## Interrupts

Interrupts are issued to the processor to notify error conditions which are asynchronous with its normal operations.

### Local Level-15 Interrupt

A local level-15 interrupt is issued to the processor when an asynchronous error is detected by CC. This may correspond to an error which was initiated by an action from the CPU (for example, a late error in an asynchronous data store, see "Bus Errors on Asynchronous Data Stores" on page 13"), or to an error which was initiated by an action from another processor or I/O device, but which was detected locally by CC (for example, a cache parity error on data owned by this external cache, see "External Cache Parity Error" on page 18).

When a local level-15 interrupt is issued, the error information is logged in the CC Error register.

### Broadcast Level-15 Interrupt

A broadcast level-15 interrupt can be issued by any device to signal of an error condition which needs to be logged. Error information is stored in the corresponding's device error register.

## Resets

Not supplied.

## CPU Watchdog Reset

A CPU watchdog reset is initiated when a trap condition occurs while traps are disabled and the MMU Control register NF bit is not set. The CPU branches to the instruction at physical address 0xF F000 0000. The WD bit in the CC Reset register is set to 1 on a watchdog reset.

A watchdog reset affects only one CPU and has no other effect on the system.

## System Watchdog Reset

When a fatal error is detected, a system watchdog reset is initiated. A system watchdog reset affects all CPUs and I/O devices. Writes in progress may be lost, but the state of main memory is not altered (main memory continues to be refreshed after a system watchdog reset).

---

# Types of errors

## Software Errors

Errors which do not originate in a hardware malfunction are classified as software errors. All such errors are detected by the processor and are reported only to that CPU. Refer to the processor specification: The Viking Microprocessor (T.I. TMS390Z50) User Documentation, Release 2.0, Sun Microsystems, November 1990, for details regarding error handling in the processor.

## Hardware-corrected Errors

Hardware-corrected errors are always signaled by a broadcast level-15 interrupt for error logging purposes. No recovery action is normally necessary.

## Memory Correctable ECC Error

When the memory subsystem detects a single-bit error on a read, the MQH rewrites the corrected data into main memory and delivers the corrected word to the requestor. The MQH also issues a level-15 broadcast interrupt with INTSID=0x02 if it is the first occurrence of a single-bit error, i.e. if the error is logged in the Correctable Error Address and Data registers. This means that an interrupt is issued

if the SBE bit is set to one because of the error and if the ECI bit of the MQH Control and Status register is set. When MQH detects the first occurrence of a single-bit error it keeps the address, data, ECC, and syndrome for the double-word which was corrected.

If multiple single-bit errors occur during the same subblock access only one interrupt is issued because only the first error is logged.

If one or multiple single-bit error(s) and one or multiple uncorrectable ECC error(s) occur in the same 64-byte sub-block, the INTSID is instead set to 0x04.

A correctable memory ECC error can be generated for diagnostics purposes by forcing incorrect generation through the MQH ECC Diagnostic and Control register.

## Recoverable Errors

Recoverable errors which have a hardware cause are usually signaled by a bus error indication to the requesting device and a level-15 interrupt (which may be broadcast). Error recovery is normally handled by the trap routines, while error logging is done by the level-15 interrupt handler.

Recoverable errors which do not have a hardware cause are signaled only by a bus error indication.

## XDBusTime-out

A recoverable XDBus time-out occurs in the following cases:

1. Read or Write access to a non-existent I/O space location from a processor (DVMA devices cannot access I/O space).
2. Read access to a non-existent memory space location from a processor or a DVMA device.
3. Write access to a non-existent memory location from a processor or a DVMA device if the external cache (respectively the IOC) is enabled (in which case it is actually the read miss which will cause the bus error).
4. DeMap time-out (Processor STA to ASI 0x03. See “DeMap Time-out” on page 19.

The effect of a recoverable XDBus time-out is to return a bus error indication TO to the requesting processor or DVMA device. This error may be forced by referencing an address known to be inexistent.

---

**Note** – XDBus time-outs are handled by BW in the processor unit and by SBI and IOC in the I/O unit. SBI handles time-outs which correspond to operations which do not affect the IOC cache, whereas IOC is responsible for time-outs involving the IOC cache. This is not visible to programmers except when time-out values are specified during system initialization.

---

Note that accessing an inexistent SBus device will result in a XDBus time-out error if the addressed system board is not present, and in an SBus time-out error if the addressed system board exists, but the SBus slot is not populated. The only programmer-visible difference is the time it takes to return the time-out.

## XDBus Rejection

If an access to a control register cannot be performed, a bus error indication BE or TO will be issued to the requesting processor. Typical error cases may be accessing with the wrong size specification, or issuing a swap operation on a register for which it is not supported. This error may be forced by performing an illegal operation.

## Memory Uncorrectable ECC Error

When the memory subsystem detects an uncorrectable ECC error, the MQH notifies a bus error indication UC to the requesting CPU or DMA device. The MQH also issues a level-15 broadcast interrupt with INSTID=0x03 if the error is logged in the Uncorrectable Error Address and Data registers. This means that an interrupt is issued if the error causes the UE bit to be set in the Uncorrectable Error Address register. In this case, MQH keeps the address, data, ECC, and syndrome for the double-word in error.

If multiple uncorrectable errors are detected in the same subblock access only a single interrupt is issued because only the first error is logged.

If one or multiple correctable error(s) and one or multiple uncorrectable ECC error(s) occur in the same 64-byte block, the interrupting INTSID is instead set to 0x04.

A uncorrectable memory ECC error can be generated for diagnostics purposes by forcing incorrect generation through the MQH ECC Diagnostic and Control register.

## External Cache Parity Error

A parity error in the external cache may be discovered in three cases, which are handled differently.

1. If a processor or DVMA device reads data from another processor's external cache because it is the current owner of this data and a parity error is detected, the reader will receive a bus error indication UC and the processor attached to the owner cache will receive a local level-15 interrupt. The error information is logged in the Error register of the owner CC, with CP=1.
2. If CC detects a parity error when trying to write back a line to main memory due to the replacement algorithm, it will write incorrect data back to memory and raise a local level-15 interrupt to its processor. The error information is logged in the CC Error register, with CP=1.
3. If a processor detects an external cache parity error during a read from its own external cache, the error condition will be handled as if a bus error indication UC had been received by the processor for the read operation. In addition, the Processor Fault Status register P bit will be set.

The first 2 cases may be distinguished by the value logged in subfield DCmd of field CCOP in the CC Error register: DCmd is 0x06 in case 2 only.

A cache parity error can be generated by forcing incorrect parity generation through the CC Control register or the Processor Control register.

## Processor Write Parity Error

If a parity error is detected by CC during a write from the processor, an early bus error indication UD will be issued to the processor. In addition, the VP bit in the CC Error register will be set.

A processor write parity error can be generated by forcing the processor to issue incorrect parity on writes using the Processor Control register.

## DeMap Time-out

A DeMap operation issued by a processor (broadcast TLB flush, performed by a store alternate to ASI 0x03) may receive a time-out bus error, resulting in a trap. See “Bus Errors on Synchronous Data Stores” on page 13. This error will occur if the store buffer of another (or "target") processor in the system becomes disabled after the DeMap operation is initiated and before the target processor has finished performing it, or certain hardware failure modes of a target processor.

Since this error reflects a temporary condition and does not necessarily indicate an hardware error, software should retry the DeMap operation on a DeMap time-out a few times before considering an unrecoverable error occurred.

## SBus Time-outs

If SBI detects a time-out on an SBus master access, it will issue a bus error indication to the requesting processor. An SBus time-out error can be forced by accessing an SBus slot known to be unpopulated on an existing system board (this may not be possible in all configurations).

## SBus PTE Errors

If a DVMA operation accesses an invalid PTE, SBI returns ErrAck to the device and logs the error in the IPTE bit of the Slot Configuration register for that device. The device must terminate the DVMA operation, log the error internally, and issue an interrupt (the details are device-dependent).

If a DVMA write operation accesses a valid PTE which does not indicate write permission, SBI returns ErrAck to the device and logs the error in the WPE bit of the Slot Configuration register for that device. The device must terminate the DVMA operation, log the error internally and issue an interrupt (the details are device-dependent).

This error can be forced by writing adequate values in a PTE.

## SBus Rejection

If the SBI detects an ErrAck on an SBus master access, it will issue a bus error indication BE to the requesting processor. This error may be forced in the loopback mode.

If the SBI detects a LateError indication on an SBus master access, it will issue a bus error indication UC to the requesting processor. This error may be forced through the SBI Control register.

## SBusParity Errors

There are 5 cases of SBus parity errors:

1. Master read: if parity is enabled for the slot and SBI detects an error, SBI issues a bus error indication UC to the requesting processor. No information is logged in SBI.
2. Master write: if the SBus device checks parity and detects an error, the device must issue a LateError indication to SBI, which will be transformed by SBI into a bus error indication with code UC to the requesting processor. The device may

also log the error internally and issue an interrupt (this is device-dependent). No information is logged in SBI. This error may be forced through the FPE bit of the SBI Control register.

3. DVMA read: if the SBus device checks parity and detects an error, the device must terminate the DVMA operation, log the error internally and issue an interrupt (the details are device-dependent). SBI does not log any information. This error may be forced through the FPE bit of the SBI Control register.
4. DVMA write: if parity is enabled for the slot and SBI detects an error, SBI will issue a LateError signal to the device and log the error in the SDPE bit of the Slot Configuration register. The device must terminate the DVMA operation, log the error internally and issue an interrupt (the details are device-dependent). This error may be forced in the loopback mode.
5. DVMA address: if parity is enabled for the slot and SBI detects an error during the address phase, SBI will issue ErrAck to the device and log the error in the SAPE bit of the Slot Configuration register. The device must terminate the DVMA operation, log the error internally and issue an interrupt (the details are device-dependent). This error may be forced in the loopback mode.

## SBus External Page Table Parity Error

If an external page table entry used during a DVMA operation has incorrect parity, SBI will return ErrAck to the device, log the error in the PPE bit of the Status register, and issue a broadcast level-15 interrupt with INTSID=0x05. The device must terminate the DVMA operation, log the error internally and issue an interrupt (the details are device-dependent). This error can be forced through the FXP bit of the SBI Control register.

## Fatal Errors

All fatal errors initiate a system watchdog reset if no Service Processor is attached to the system. When a Service Processor is attached to the system fatal errors cause the XDBus traffic to be suspended. Fatal errors correspond to hardware errors in which proper system operation cannot be guaranteed.

---

**Note** – In current Sun-4D systems there is no plan to support a Service Processor and all fatal errors are handled through a system watchdog reset.

---

The reason for a fatal error is logged in the XDBus CSR of the device which reported the error (BW, MQH, or IOC). The reason may be an error detected by an associated XBus device (CC for BW, SBI for IOC), in which case the corresponding error register must also be checked.

Fatal errors can be disabled on a per-chip basis through the EER bit of the XDBus Control and Status registers of BW, MQH and IOC. This feature is intended for system bringup and diagnostics and should not be used during normal operation.

## XDBus Parity Error

XDBus parity errors may be detected by BW, MQH and IOC. The DPE bit of the XDBus CSR for the detecting device is set, and the data and parity bits are recorded. As a side effect, the parity logging in BICs is stopped, which allows error analysis software to locate the source of the parity error and the byte on which it occurred.

This error can be forced using JTAG boundary scanning.

## XDBus Arbitration Parity Error

XDBus arbitration parity errors may be detected by BW, MQH, IOC, BARB and CARB.

If the error is detected by BW, MQH, or IOC, the GPE bit of the XDBus CSR for the detecting device is set, and the arbitration signals are recorded.

If the error is detected by BARB or CARB, a bit corresponding to the arbitration port in error will be set in the corresponding error register (accessible only through JTAG).

This error can be forced using JTAG boundary scanning.

## XDBus Arbitration Time-out

XDBus arbitration time-outs may be detected by BW, MQH and IOC. The GTO bit of the XDBus CSR for the detecting device is set. The time-out duration is parameterized through the GTOL field of the corresponding XDBus CSR.

This error can be forced by using the TTO bit in the BW XDBus CSR.

## XBus ParityError

XBus parity errors may be detected by BW, CC, IOC, and SBI.

If the error is detected by BW or IOC, the CDE field of the corresponding XDBus CSR will be set to a chip-specific value. No additional data is logged.

If the error is detected by CC, the CDE field of the XDBus CSR in the corresponding BW will be set to a device-dependent value to indicate a CC error and the Error register of CC will indicate an XBus parity error. If the error is detected by SBI, the CDE field of the XDBus CSR in the corresponding IOC will be set to a device-dependent value to indicate an SBI error and the error register of SBI will indicate an XBus parity error.

Those errors may be forced by forcing XBus signals through JTAG boundary scanning.

## Cache Consistency Errors

Cache consistency errors may be detected by BW, CC or IOC.

If the error is detected by BW or IOC, the IE bit of the CDE field of the XDBus CSR is set to one and detailed error information is logged in the BW DynaData register.

If the error is detected by CC, the IE bit of the CDE field of the XDBus CSR in the corresponding BW is set to one and the CCE bit of the BW's DynaData register is set to one to indicate a CC error. The CC Error register will indicate a cache consistency error.

Those errors may be forced by modifying directly the tags in CC, BW or IOC in inconsistent ways.

## WriteSingle Time-out

This error is generated when no reply is received after a XDBus memory write operation. If the external cache is enabled, the only case where this can happen is if an MQH accepted a BlockRead (cache line fill) for an address and later ignored a write to that address. Note that, if the external cache is disabled, any write to memory space will issue this error.

The error is logged in the BW, with code 0x1 in the CDE field of the XDBus CSR and the WSTO bit set in the DynaData register. The time-out value is controlled by the RDTOL field of the BW XDBus CSR.

The same error can occur in the IOC for consistent mode stores to shared data. In that case, the error is logged in the IOC as bit STO of the CDE field of the XDBus CSR. The time-out value is controlled by the RDTOL field of the IOC XDBus CSR.

## Multiple DeMaps

This error is generated when two or more DeMap operations are active at the same time. It is detected by the BWs.

## Device-specific Errors

Those errors can be issued by any device in the system. They correspond to a failure of internal consistency checks.

Refer to the individual unit description for a list of device-specific errors.

## Critical Errors

Critical errors require immediate system shutdown and power-off. They are notified through level-15 broadcast interrupts. Note that, since those interrupts are issued via the BootBus, they do not provide an INTSID and are dispatched only to the processor which currently holds the BootBus semaphore 0, or to both processors if none of them holds the semaphore.

## AC Failure

If an AC power failure is detected (loss of line power), a level-15 interrupt is issued to all BootBuses and the ACInt bit of the BootBus Status\_2 register is set in all system boards. This interrupt cannot be masked individually. When the interrupt is issued, the maximum guaranteed system operation time is 5 ms.

## Temperature Warning

If the temperature raises above normal operational level on a given system board, a level-15 interrupt is issued to the BootBus on that board and the TmpInt bit of the BootBus Status\_2 register is set in the corresponding board. The processor which holds Semaphore 0 receives the interrupt. This interrupt can be masked for by using the EnTmp bit of the BootBus Control register. The system should be shut down in a timely manner when a temperature warning is notified.

## Fan Failure

If a fan failure is detected, a level-15 interrupt is issued to all BootBuses and the FanInt bit of the BootBus Status\_2 register is set in all system boards. The processor which holds Semaphore 0 receives the interrupt. This interrupt can be masked for each BootBus by using the EnFan bit of the BootBus Control register. The system should be shut down in a timely manner when a fan failure is notified.

## DC Failure

On SPARCcenter 2000 systems with dual power supplies, a DC failure on one power supply will cause a level-15 interrupt to be issued to all BootBuses. The DCInt bit of the BootBus Status\_2 register is also set on all system boards. The processor which holds Semaphore 0 receives the interrupt. This interrupt can be masked for each BootBus by using the EnDC bit of the BootBus Control register.



## General and Hard Hangs

---

---

### Troubleshooting System Hangs

Part of the difficulty in working with system hangs is knowing whether or not the system is actually hung. Sometimes it appears that the system is hung when in fact only a particular application is hung. To help determine whether the system is actually hung and to help diagnose the problem, ask these questions:

1. Can you rlogin or telnet to the system?
2. Can you ping the system?
3. Does the mouse track in the window?
4. What changes have been made to the system recently?
5. How often do the hangs occur?
6. What are the circumstances under which the hang occurs?
7. Can the hang be reproduced on command?
8. What is necessary to get out of the hang (i.e. can the machine be L1-A'd)?

---

## Checking for a Resource Deprivation Hang

The most common cause of a system hang is that the system has run out of resources. The first thing to normally do is to run some performance tools to see whether that is in fact the case.

The following can be placed in a file and then invoked from cron every 15 minutes to help determine if the system is CPU bound, I/O bound, or memory bound.

```
date >> file1
vmstat 30 10 >> file1
date >> file2
iostat -xtc 30 10 >> file2
date >> file3
/usr/ucb/ps -aux >> file3
date >> file4
echo kmastat | crash >> file4
date >> file5
echo "map kernelmap" | crash >> file5
```

Most users will want to change the file names to be placed in some absolute pathname location.

## CPU Power

In the vmstat command output, look to see what the run queue size is (first column). If the run queue has more than 3 processes waiting per CPU (i.e. more than 3 for a 1 CPU system, more than 6 for a 2 CPU system, etc.), then this bears watching. If the run queue has more than 5 processes waiting per CPU, there is insufficient CPU power in the system.

## Virtual Memory

If over time, the amount of memory specified in the swap column goes down and does not recover, then there is a probability that there is a memory leak on the system. To determine if it is a kernel memory leak, look at the output of the two

crash commands (described later). To determine which application has a memory leak, look at the SZ column of the ps output. This column indicates the size of a process's data and stack in kilobytes.

If over time, the swap column goes down and recovers, look at the lowest value in the swap column. If this value goes below 4000, then the system is in danger of running out of virtual memory space. More swap space should be added to the system.

## Physical Memory

The sr (scan rate) column of the vmstat output indicates the rate at which pages are being scanned in order to find needed pages for current processes. If this rate is over 200 for prolonged periods of time, the machine is out of physical memory. This machine could benefit by additional physical memory.

## Kernel Memory

The kernel had a limited amount of memory which it uses for kernel data allocations. This memory is commonly referred to as the kernel heap. The maximum size of the kernel heap is fixed depending on machine architecture and the amount of physical memory. If a machine runs out of kernel memory, it will usually hang. To see if this is the case, look at the output of the crash kernelmap command. This command shows how many segments of kernel memory exist and how large each segment is. If there only 1 and 2 page segments left, the kernel has run out of memory (even if there are a hundred of them).

The crash kmastat command shows how much memory has been allocated to which bucket. Prior to Solaris 2.4, this showed only 3 buckets making it difficult to tell which bucket was hogging the memory (if any). Starting with 2.4, kmastat breaks memory allocation down to many different buckets. If one of these buckets has several MB of memory, and there are kernel memory allocation failures, there is probably a memory leak involving the large bucket.

In order to diagnose a memory leak problem it is possible to turn on some flags in Solaris 2.4 and above (see SRDB 12172). With these flags turned on, once the kmastat command shows significant growth in the offending bucket, L1-A should be used to stop the machine and create a core file. SunService can use this core file to help determine the cause of the leak.

## Disk I/O

To check for disks which are overly busy, look at the iostat output. The columns of interest are %b (% of time the disk is busy) and svc\_t (average service time in milliseconds). If %b is greater than 20% and svc\_t is greater than 30ms, this is a busy disk. If there are other disks which are not busy, the load should be balanced. If all disks are this busy, additional disks should be considered.

There is no direct way to check for an overloaded SCSI bus, but if the %w column (% of time transactions are waiting for service) is greater than 5%, then the SCSI bus may be overloaded.

Information about what levels to check for the various performance statistics is taken from "Sun Performance and Tuning" by Adrian Cockcroft, ISBN 0-13-149642-5.

Additional performance gathering scripts can be gotten from Infodoc 2242 for Solaris 2.x and Infodoc 11365 for SunOS 4.x.

---

## Generating Core Files

If looking at the performance statistics is not enough to diagnose the problem, it is necessary to get a core file. Infodoc 11837 describes how to do this.

If it is not possible to get a core file, then the situation is called a hard hang. Contact SunService for information on diagnosing hard hang situations.

## Analyzing System Hang Core Files

Once a core file is obtained, the first information to look at is a threadlist generated by the adb command.

```
$adb -k unix.NUM vmcore.NUM | tee threadlist.NUM
physmem xxxxx
$<threadlist
```

The value NUM in the above commands will be replaced by the number of the core files (e.g. unix.1, vmcore.1). If this is a large system, go get a cup of coffee while the threadlist runs (it can take up to 15 minutes).

Once the threadlist has been gotten, use a text editor to look at the threadlist file generated. An example of some threadlist output follows:

```

===== thread_id          e0182000
p0:
p0:      process args=   sched
t0:
t0:      lwp             proc             wchan
           e01c9898      e01d6a48      0
t0+0x34: sp             pc
           e0181ee8      sched+0x3f0
?(?) + effff090
main(0x0,0x3c,0x2,0xe01a7c00,0xe01d6a48,0xe01a7cb8)

===== thread_id          e0e81ec0
p0:
p0:      process args=   sched
0xe0e81ec0: lwp             proc             wchan
           0             e01d6a48      0
0xe0e81ef4: sp             pc
           e0e9fec0      poll_obp_mbox+0xbc
?(?) + effff090
poll_obp_mbox(0xfc,0xfc,0x0,0x742c,0xffffffff,0xe01c5e5c)
level14_handler(0xe0e81d54) + 4
L14_front_end(?) + 68
splclock(0x404000e6)
disp_getwork(0xe01c4a48,0x404000e6,0x0,0xffffffff,0xe10baec0,0xf7469030) + c
idle(0xe01c4a48,0x0,0xe01c4a4c,0xe01c4b64,0x0,0x0) + cc

===== thread_id          e0ea2ec0
p0:
p0:      process args=   sched
0xe0ea2ec0: lwp             proc             wchan
           0             e01d6a48      0
0xe0ea2ef4: sp             pc
           e0ea2610      complete_panic+0xd0
?() + effff090
data address not found

```

This output will have a brief description of information from the thread structure followed by a stack trace for each thread in the system. Threads which show stack traces with data address not found have typically been swapped out.

Looking through this threadlist, look for stack traces which show `mutex_enter`, `rw_enter`, or `biowait` as the top routine. Also look for threads trying to do kernel memory allocation (`kmem_alloc*`).

If there are many threads waiting on the same mutex (look at `wchan`), see what thread owns the mutex and what it's doing. Follow this trail to see why the machine may be hung.

If there are many threads trying to do `kmem_alloc`, there is probably a lack of kernel memory. See the kernel memory section above for information about setting kernel memory flags. Once a corefile with kernel memory flags set is obtained, check with SunService on how to proceed.

If there are many threads waiting on `biowait`, check the buffers being waited for to see if they are all doing I/O to the same disk. Maybe a controller is hung or not operating correctly.

---

## Kernel Problems

I have various kernels that can be used to help debug kernel problems. Deadman kernels are used when systems hard hang. That is the system does not respond to `<BREAK>` or `L1-A`. The deadman kernels are built from the source that was used to build the Kernel Jumbo Patch with the deadman code added to that.

Before you load a deadman kernel, make sure the following has been done to try and debug the problem:

1. Confirm that the state of the system, does it respond to a ping?
2. If it is a server with a key switch make sure the key switch is not in the safe mode.
3. Make sure that the system does not have the consulting special `CONSULT-ZSBRK`.
4. If the system has a framebuffer and keyboard and does not respond to `L1-a (Stop-A)` then try unplugging the keyboard and plugging it back in. This will generate a hardware intrurrpt that should abort the system unless it is truly hard hung.

### 2.4 Deadman Kernels

The 2.4 `sun4d` deadman kernel is a direct port of the `sun4d` deadman code from 2.5. Deadman kernels are used when systems hard hang. That is the system does not respond to `<BREAK>` or `L1-A`. The deadman kernels are built from the source that was used to build the Kernel Jumbo Patch with the deadman code added to that. Please note this code does not use the deadman routine that is included in the base Solaris 2.4 release.

### Testing

I have only done limited testing of these kernels, this consists of:

1. Boot an SC1000 with a storage array, with and without kadb.
2. Boot an SC1000 with a storage array, with the deadman code enabled, with and without kadb.
3. Load a driver that causes the system to hard hang and make sure the system does indeed drop into the debugger or prom depending on whether running kadb or not.

If you are sending this to a customer it is YOUR responsibility to test the kernel before you send it. If you are unable to do this testing then get the customers exact configuration details and log an escalation so that CTE can do the testing.

## Usage

To use the deadman kernel you should move `/kernel/unix` to `/kernel/unix.orig` and then copy the deadman kernel to be `/kernel/unix`. Then add this line to `/etc/system`:

```
set snoopng=1
```

When set up if the clock thread fails to run for 2 seconds then the system will abort into the prom or kadb.

## Warnings

As the deadman timer uses the level 14 interrupt, which is normally used when profiling the kernel, it is not possible to use the deadman kernel when profiling the kernel. This is unlikely to cause any problems, as only kernel developers would normally be profiling the kernel.

## Availability

The Deadman kernels can be got via anonymous ftp from `otis.uk:~gerhard/deadman`. Currently I only have deadman kernels for Solaris 2.4 and the sun4d kernel architecture. After pulling the correct kernel make sure it matches the checksum given in the `cksums` file. Solaris 2.5 and above has deadman support for sun4d in by default.

This directory contains deadman and debug kernels for SunOS 5.4, sun4d systems. The Deadman kernels are called `unix-deadman-XX` where `XX` is the revision of the Kernel Jumbo Patch that they were built against. The deadman kernels are for use on systems with the same kernel jumbo patch version.

## 2.4 Debug Kernels

Debug kernels have numerous assertions enabled. These assertions check that conditions that should be true are true when the system is executing.

### Testing

I have only done very limited testing of these kernels, this consists of:

1. Boot an SC1000 with a storage array, with and without kadb.

If you are sending this to a customer it is YOUR responsibility to test the kernel before you send it. If you are unable to do this testing then get the customers exact configuration details and log an escalation so that CTE can do the testing.

### Warnings

There is a bug in the veritas volume manager that means the debug kernel will always panic with an assertion failed. See bug 1185904.

### Availability

This directory contains debug kernels for SunOS 5.4 are called unix-debug-XX where XX is the revision of the Kernel Jumbo Patch that they were built against. After pulling the correct kernel make sure it matches the checksum given in the cksums file.

## 2.5 Deadman Kernels

I have modified the 2.5 source and built a working 2.5 kernel for sun4m. As mentioned above 2.5 already contains the deadman kernel for sun4d. The 2.5 deadman kernel comes as 2 parts, a genunix and a unix YOU NEED BOTH. Each file as a -XX postfix that refers to the version of the kernel jumbo patch against which this deadman kernel was built. This deadman kernel modifies the deadman routine that is already in the kernel.

The current version (06) only supports modification of the snooping variable from /etc/system, it also will not support kernel profiling even when snooping is not set. I intend to fix this in the future.

## Testing

I have only done limited testing of this kernel, this consists of:

1. Boot a uni processor SS10, load the hard hang driver and confirm that it will deadman.
2. Boot a uniprocessor SS20, load the hard hang driver and confirm that it will deadman.
3. Boot a dual processor SS10, load the hard hang driver and confirm that it will deadman.
4. Boot a dual processor 4/600.load the hard hang driver and confirm that it will deadman.

If you are sending this to a customer it is YOUR responsibility to test the kernel before you send it. If you are unable to do this testing then get the customers exact configuration details and log an escalation so that CTE can do the testing.

## Usage

To use the deadman kernel:

1. Move /kernel/genunix to /kernel/genunix.orig
2. Copy the genunix file to be /kernel/genunix.
3. Move /platform/`uname -i`/kernel/unix to /platform/`uname -i`/kernel/unix.orig
4. Copy the unix file to /platform/`uname -i`/kernel/unix.
5. Add this line to /etc/system:

```
set snooping=1
```



## Acronyms

---

---

### Sun4d Prtdiag Acronyms

Acronym	Definition
BW	Bus Watcher - This chip interfaces between the External cache controller for the CPU and the XDBus.
IOC	Input Output Controller. Controls interface to the XDBus. It effectively links the SBus interface to the XDBUs.
MQH	Memory Queue Handler. This is the memory controller for the sun4d systems.
MXCC	External Cache Controller
SBI	SBus interface chip. The chip is the SBus controller
XBus	The bus that connects bus watchers and CPUs, also the bus that connects the IO controllers and the SBus interface chip
XDBus	The main bus that is the backplane for the sun4d systems

---

## IOC Error Messages

Acronym	Definition
DOHU	XDBus Reply Output Queue Underflow
DOHO	XDBus Reply Output Queue Overflow
DOLU	XDBus Request Output Queue Underflow
DOLO	XDBus Request Output Queue Overflow
BDOHU	XDBus Reply Output Buffer Underflow
BDOHO	XDBus Reply Output Buffer Overflow
BDOLU	XDBus Request Output Buffer Underflow
BDOLO	XDBus Request Output Buffer Overflow
DIHU	XDBus Reply Input Queue Underflow
DIHO	XDBus Reply Input Queue Overflow
DILU	XDBus Request Input Queue Underflow
DILO	XDBus Request Input Queue Overflow
DEVIDFU	Underflow of Device ID register file
DEVIDFO	Overflow of Device ID register file
XINQU	XBus Input Cache Queue Underflow
XINQO	XBus Input Cache Queue Overflow
DSTRQU	Datastore Queue Underflow
DSTRQO	Datastore Queue Overflow
ILE	Invalid Line Error
PSE	Pending State Error
RBE	Readblock Error
XIOERR	Fatal Error detected by the SBI
XPE	XBus Parity error

---

## MQH Errors

Acronym	Definition
OQFIFO	Finite State machine error
CMDQUE	Finite State machine error
MEM_MSTR	Finite State machine error
MEM_SLV	Finite State machine error
MCRAM_CTR	Finite State machine error
CMDQUE	Overflow : Command Queue Overflow

---

## BW Errors

Acronym	Definition
URE	An undefined reply error packet was received
IOWSCE	An IO Write Single Count Error
WSKBCE	A WriteSingleInvalidate, WriteSingleUpdate, SwapSingleInvalidate, SwapSingleUpdate, WriteBlock, or Interrupt Count Error
RM1CE	A Read Miss 1 Count Error
RM2CE	A Read Miss 2 Count Error
WMCE	A Write Miss Count Error
IOWBCE	A IOWriteBlock Count Error
SRCE	A Stream Read Count Error
IORCE	An IOReadSingle or IOSwapSingle Count Error
UXC	Undefined XBus command
UGE	Unexpected Grant Error
ICMFT	Incorrect Command Field in Memfault cycle
CCE	Cache Controller Error
XPE	XBus Parity Error

Acronym	Definition
SGT	Spurious Grant
WSTO	WriteSingleInvalidate, WriteSingleUpdate, SwapSingleInvalidate, SwapSingleUpdate, WriteBlock, or Interrupt Timeout
INVFIFO	Invalidate FIFO Overflow. Control flow error
DREFIFO	XDBus Request FIFO Underflow. Control flow error
DREFIFO	XDBus Request FIFO Overflow. Control flow error
DRPFIFO	XDBus Reply FIFO Underflow. Control flow error
DRPFIFO	XDBus Reply FIFO Overflow. Control flow error
IDFIFO	ID FIFO Underflow. Control flow error
IDFIFO	ID FIFO Overflow. Control flow error
CPFIFO	Control Packets FIFO Underflow. Control flow error
CPFIFO	Control Packets FIFO Overflow. Control flow error
FBRFIFO	FlushBlock Requests FIFO Underflow. Control flow error
FBRFIFO	FlushBlock Requests FIFO Overflow. Control flow error
RBRFIFO	ReadBlock Requests FIFO Underflow. Control flow error
RBRFIFO	ReadBlock Requests FIFO Overflow. Control flow error
XBFIFO	XBus FIFO Underflow. Control flow error
XBFIFO	XBus FIFO Overflow. Control flow error
MDMP	Multiple Demap Error
PFMAE	Pending Flush Monitor Allocation Error
PFMBE0	Pending Flush Monitor Block 0 Error
PFMBE1	Pending Flush Monitor Block 1 Error
PFMCE	Pending Flush Monitor Contention Error
TPE	Tags Parity Error
FPE	Flags Parity Error
UVA	Unspecified Victim Address
IOWS	Invalid Owned Write Single

---

## Acronyms Used to Report Data from Cache Controllers

Acronym	Definition
ERR	Error code
PA	Physical address
CCOP	Cache controller operation code

---



## Troubleshooting

---

This section is in some ways the most difficult of all. Once you see you have some sort of problem on your SS1000 or SC2000. The next step is where do I start or what can I do to find the problem that is causing not only the system problems. But the administrators of these systems as well.

Well, instead of waiting till something happens with your 4d system. The first and best course of action is to do some preventive maintenance.

When I work with customers in dealing with these 4d systems, one of the first things I ask them is how important is this system to you. If this is a high profile system, that needs to be basically up all the time(sounds HA to me). Then we need to take steps in case there are problems.

You just installed your SS1000 or SC2000. The first thing to do is run `prtdiag -v > "somefile_name"`. Why do we want to do this, simple, we want to get a snapshot of the system.

The second thing we want to do is, and this is if your running Solaris 2.5.1 is to run `prtdiag -l`. This is a new option to `prtdiag` with 2.5.1. What this does is get the last fatal error messages out of the system and stick it in `/var/adm/messages`.

Next would be a level 0 dump of the system

I know this is a new system, but a little sanity checking will go a long way, trust me.

Next, if your running Solaris 2.5.1, is to put this `prtdiag -l` in your cron file, and how often you want it too execute is up to you.

After that, go to your startup scripts and "TURN" on `savecore`. If your system panics and crashes, when it comes up if there is a core file to generate, it will dump it to that location and we can do a core analysis of it. If you do NOT have `savecore` turned on. We'll discuss later how you can get around it.

Install SunVTS 2.0 on your 4d system, again how often you want to run this is up to you. At the minimum, once a week is what I would suggest. Do you think I am trying to preach preventive maintenance, in a word "YES".

Next up would be Symon 1.1, this version of Symon includes support for your 4d system, this a an excellent tool for monitoring your system. With Symon, you can customize the default rules or create your own rules. Depending your severity of the problem, you can have your rule, email or page the needed folks to respond to your systems problems.

A note about SunVTS, if you are doing ANY type of disk check and your using SDS, make sure you set the environment variable before you startup SunVTS, otherwise you WILL trash your SDS disks!!!!!! This is documented in the SunVTS User Guide.

Okay, we have done all the preventive maintenance stuff and we're working just fine. One early Sunday morning, say about 1:30am your 4d system has a fit.

What to do, what are your options at this point?

You're at the system now, keyboard doesn't respond. Can't login from another system. You can power cycle your system, not a good idea at this point. We still have a couple of option open to us.

See Section Eight for TIP

We can cause the system to panic, this is done by one of two ways. A real simple way is to unplug your keyboard from your system and plug it back in, this will panic your system. If your have savecore turned on, you will get a core dump. The otherway is to modify your kernel and create a "fan failure". See Section Seven for Fan Failure.

If you didn't turn on savecore, once you caused your system to panic and reboot. As soon as you get a login prompt. Login and do the following. Make sure you're root.

```
savecore /var/crash/`uname -n`
```

You must run savecore very soon after booting - before the swap space containing the crash dump is overwritten by programs currently running.

If your system suffered from a watchdog reset and is sitting at the "ok" prompt. Type sync, this will more then likely cause your system to panic and dump core since we have turned on savecore.

But before typing sync, please run the commands from section six concerning troubleshooting commands.

Your system is up now at the login in prompt, were do we go from here. Login into your system and if your running Solaris 2.5.1, run prtdiag -l, remember that this will pull the last fatal error message out of the system with a time stamp and stick it in

```
/var/adm/messages.
```

Grab the /var/adm/messages file. Check the last actions before the reboot. We now have a core file to look at, /var/adm/messages, we'll need a listing of showrev -p to see the patch level there running at. With this information we can have the Kernel Group do an analysis of the core dump.

How about the hardware?

We are gonna run something called the "POST DAEMON" test, this is common to both the SS1000 and the SC2000. There is also a version on the UE X0000 servers.

It seems that very few people who work on these 4d system know about the POST DAEMON test. This little known feature will become of your best friend.

How to startup the POST DEMON test:

1. Power on system, ascii terminal on ttya (POST starts)
2. The letter "s" key is typed on the ascii terminal

Message will appear below

```
*** Toggle Stop POST Flag = 1 ***
POST diagnostics completes
OBP POST DEMON starts
The following menu will appear(excuse the caveman type graphics)
=====
=DEMON
=0A>Select one of the following functions
=0A>  '0'   System Parameters
=0A>  '1'   Read/Write device
=0A>  '2'   Software Reset
=0A>  '3'   NVRAM Management
=0A>  '4'   Error Reporting
=0A>  '5'   Analyze Error logs
=0A>  '6'   NVRAM SIMM tests
=0A>  'r'   Return to selftest

Command ==>
=====
```

What are the features?

- System Parameters

There are 7 sub menus in this one.

- Read/Write devices

This sub-menu allows you to read and write to system board ASICs. THOROUGH KNOWLEDGE of how the system physical addresses are to the ASIC is required.

- Software Reset

A convenient way of restarting POST without exiting the DEMON.

- NVRAM Management

This sub-menu is used to manage the memory SIMM test results in the NVRAM error logs. It allows you to view and erase the data.

- Error Reporting

This sub-menu is used to print out data saved on the last system WATCHDOG reset. A dump of the system board registers.

- Analyze Error Logs

Analyze NVRAM error logs on every system board. Potentially valuable if there is a POST failure.

- NVRAM SIMM Tests

This sub-menu is provided to users to test the NVRAM SIMMS. This is only a battery test.

- Return to selftest

Terminate the DEMON and the OBP firmware continues on with its normal routine of system configuration.

Continuing on with POST DEMON test:

```

Command ==> 0 (main DEMON menu)
=====
System Paramaters
=0A>Select one of the following functions
=0A>  '0'   Set POST Level
=0A>  '1'   Dump Device Table
=0A>  '2'   Display System
=0A>  '3'   Dump Board Registers
=0A>  '4'   Dump Component IDs
=0A>  '5'   Clear Error logs
=0A>  'r'   Return

Command ==> 2
=====

```

(0=failed,1=passed,blank=untested/unavailable)  
(sbus 1=card present, 0=card not present, x=failed)

Slot	cpuA	bw0	bw1	CpuB	bw0	bw1	ioc0	ioc1	sbi	mqh0	mqh1	mem	sbus	xd1	xd0
0	1	1	1	0	1	1	1	1	1	1	1	128	1011	1	1
1	1	1	1	1	1	1	1	1	1	1	1	128	0000	1	1

Memory Group Status

(0=failed,1=passed,m=simm missing,c=simm mismatch,blank=unpopulated)

```
+---+-----+-----+-----+-----+
Slot|xd0_g0|xd0_g1|xd1_g0|xd1_g1|
+---+-----+-----+-----+-----+
  0 |  1   |  1   |  1   |  1   |
  1 |  1   |  1   |  1   |  1   |
+---+-----+-----+-----+-----+
```

As you can see from the above grids, "CpuB" on system board in slot 0 has failed, as well as an sbus card on system board 0, sbus slot 1.

Now, if you took a snap shot of your system earlier. Based on this grid and what has failed. You can determine what FRU(s) need to be replaced.

---

**Note** – NEVER perform the "clear error logs" function.

---

This destroys the POST error information stored in EVERY NVRAM on all system boards!

If engineering needed to get the board back to the labs for annalysis, this is one place there gonna look for information.

Continuing with the POST DEMON testing:

```
Command ==> 2
OA>Initiating Software Reset...

Command ==> 5
OA>
-----Error Log Analysis for Board 0-----
OA>
-----Error Log Analysis for Board 1-----
OA>Hit any key to continue
OA>

Command ==> r

tty-a not found
tty-b not found

SPARCcenter 20000, No keyboard
ROM Rev. 2.25, 256 MB Memory installed, Serial #5243196
Ethernet address 8:0:20:11:7d:d7, Host ID: 8050013c
```

type help fpr more information

<#0> ok

What did we just do?

- DEMON Command 2, Software Reset

This allows you to restart POST without resetting the stop POST flag. You are still in the DEMON after POST completes.

- DEMON Command 5, Analyze Error logs

You get a summary of any errors on the system boards during POST diagnostics testing.

- DEMON Command r, Return to Selftest

The firmware continues on with its normal routine, which is to check the system hardware configuration and then to boot.

Notes:

cpuA bw0 bw1

This is the CPU A path to the two XD backplane buses. bw0 and bw1 are the bus watcher chips.

cpuB bw0 bw1

This is the CPU B path to the two XD backplane buses. bw0 and bw1 are the bus watcher chips.

ioc0 ioc1 sbi

This is the internal SBus paths to the two XD backplane buses. ioc and ioc1 are the i/o cache chips; sbi is the sbus interface that communicates with the sbus cards.

mqh0 mqh1

These are the two memory controllers(memory queue handlers, or MQH), one for XDBus 0 and one for XDBus 1.

xd1 xd0

These are the two internal XDBus path.

bic barb

bic is the bus interface chip; barb is the board arbiter chip

Bootbus(bb)

This is the internal system board path to on-board 8 bit data devices that include: UARTs, TODC/NVRAM, LEDS, SRAM, EPROM and JTAG

## Troubleshooting Commands

---

On the Sun4d systems you have a handful of commands to use to help you in trying to pin down what might have caused you system to suffer a watchdog reset or a hard hang.

Because you are running the commands from OBP, you cannot save them to a file per say. What I would suggest is to setup your Sun4d system to handle "tip". This way you can run it from another system and save the output when your done. Otherwise it is the pencil and paper method.

What commands are at your disposal?

---

Command	Definition
.registers	Displays many of the kernal internal CPU registers
.locals	Dumps out the registers in the current register "window." These are the registers that were in use at the time of the crash.
.psr	Prints the Processors Status Register contents in a readable format
ctrace	Displays the return stack(similar to a \$c command in adb)
wd-dump	Displays watchdog information on the console, including the PC value, which is the location of the instruction that caused the crashed

---

Let's take a look at them one by one.

---

## wd-dump(watchdog dump)

The wd-dump command dumps out the contents of several registers internal to the Sun4D hardware (including several which can be displayed via other OBP commands). This command also dumps out the SRMMU (SPARC Reference Memory Management Unit) registers. Here's an example:

```
<#5> ok wd-dump
Reset reason type: 1
CC_control: ce  CC_reset: 4  CC_status: 0 8  CC_error: 0 0
BW_control: 2020  BW_csr: 1a042 8000dd00  BW_data: 0 0
BW2_control: 2020  BW2_csr: 1a052 8000dd00  BW2_data: 0 0
BB_control: ef  BB_status1: 42  BB_status2: 2  BB_status3: 2f
BB_sema0_stat: 5  BB_sema1_stat: 4
IU_psr: 404000c2  IU_pc: f74000bc  IU_npc: f74000c0  IU_y: 0  IU_tbr: e0040060
IU_wim: 40  IU_g[8]: 0 0 8000000 ffffffff 980 f82d9a40 1 f824e140
SRMMU_control: 3017001  SRMMU_context: 1faf  SRMMU_cxtbptr: 1ffc000
SRMMU_faultaddr: df72aabc
SRMMU_faultstatus: 20000
<#5> ok
```

Notice that the wd-dump output actually contains the complete .registers output as well (although not in as nice a format). Don't ask customers to do both commands. There is no need.

From the wd-dump output, write down and later use:

- PC and nPC
- SRMMU fault address
- SRMMU fault status

The wd-dump(watchdog dump) displays the following registers on a Sun4d system:

---

Cache Controller	control register reset register status register error register
Bus Watcher (both BW1 and BW2)	control register Dynabus control and status register Dynabus register
Boot Bus	control register status_1 register status_2 register status_3 register semaphore 0 register semaphore 1 register

---

The above registers are described in Sun4d Architecture manual.:

---

IU	PSR, PS, NPC, Y, TBR, WIM, g[0]-g[7]
SRMMU	control register context register context tbl pointer register fault register fault status register

---

The above registers are described in The SPARC Architecture Manual Version 8.

---

## .registers

The `.registers` command dumps out the contents of the global (%g) window registers, the PC (where we died) and the next PC, and the contents of 4 processor registers.

Here is an example:

```
<#5> ok .registers
      %g0      %g1      %g2      %g3      %g4      %g5      %g6      %g7
      0        0      8000000  ffffffff  980 f82d9a40      1 f824e140
      PC      nPC      Y      PSR      WIM      TBR
f74000bc f74000c0      0 404000e3      40 e0040060
<#5> ok
```

The Y (Multiply/Divide) register is the least likely to be of use to us. The PSR (Processor Status Register) is better viewed via the `.psr` command. The WIM (Window Invalid Mask) is not of much use.

From the `.registers` output, write down and later use:

- The TBR (Trap Base Register) value
- The contents of %g7. This points to the thread which was running.

---

## .locals

The `.locals` command shows the contents of input (%i), output(%o), and local (%l) registers. Here's an example:

```
<#5> ok .locals
      0        1        2        3        4        5        6        7
IN:    0 f7400088      10c e01cd434      0        0 e3667dc8 e00dd394
LOC:   ffffffffdf 404000c3 e00dcaf4 f722cf48      40        40 f7400078 f7400184
OUT:   e34a5a70 e01b6c48 f80d7c38 f80d7c40      67 e34a5a70 e3667d68 e008bfdc
<#5> ok
```

These registers are of interest as they are the registers which were visible to the running process at the time of the Watchdog Reset.

From the ctrace output, write down and later use:

- %i registers -- Possibly still calling arguments to the current routine
- %l registers -- Working registers for the current routine

- %o registers -- Parameters possibly being set for the next routine call

---

## ctrace

The following example is taken from a sun4d system which was encouraged to Watchdog Reset by a kernel module which played with the PSR ET bit.

```
Watchdog Reset
Type help for more information
<#5> ok
<#5> ok ctrace
PC: f74000bc watchdog:_init+34
Last leaf: call e008c1b0 lookup_one          from e008bfdc kobj_lookup+8
             0 w %o0-%o5: ( e34a5a70 e01b6c48 f80d7c38 f80d7c40      67 e34a5a70 )

jmp1 f7400088 watchdog:_init                from e00dd394 modinstall+120
     1 w %o0-%o5: (          0 f7400088      10c e01cd434          0          0 )

call e00dd274 modinstall                    from e00dcb14 mod_hold_installed_mod+44
     2 w %o0-%o5: ( f8122480          0          0          0 e01acc00 f7400088 )

call e00dcad0 mod_hold_installed_mod        from e00dbef4 modctl_modload+5c
     3 w %o0-%o5: ( f8122480          0 e3667eec          0          16 e00dbed0 )

jmp1 e00dc358 modctl                        from e006c2a4 syscall_ap+6c
     4 w %o0-%o5: ( f82d9df0 e3667f50      be0 f82d9a40      f3 f82c0898 )

jmp1 ffffffff from e006c770 syscall_trap+104
     5 w %o0-%o5: (          0          0 dffffa88          2 e00dc358 f82d9a40 )

<#5> ok
```

This output shows us similar info to what adb's \$c output would provide plus more. First, we have the prompt which tells us which CPU hit the Watchdog Reset. In this case, it was CPU #5. We have the PC address where we died, in something called "watchdog" at offset init+34. We have the "who called whom" stack traceback list, as well as the routine names. In this case, "modctl" called "mod\_hold\_installed\_mod", which called "modinstall", which called "watchdog". In ()'s, we have the contents of the 6 input registers for each call. These are the routine calling arguments.

If we didn't have the "obpsym" module activated, we would have to do a lot of work in adb after rebooting and HOPE that the addresses which we were jumping from and to contain the same information on the newly rebooted system.

From the ctrace output, write down and later use:

- CPU number
- PC information
- Stack trace back
- First couple arguments for each call

---

## .psr (Process Status Register)

(SPARC V8 only) The .psr command is used on SPARC V8 systems only to dump the contents of the Process Status Register. Here is an example of the output:

```
<#13> .psr
CWP: 6  ET: 1  PS: 1  S: 1  PIL: 0  EF: 0  EC: 0  ICC: nZvc  VER: 0  IMPL:
```

If your at the kadb prompt, you can type the following commands to get some information off the system. The reason being is that there is a chance that getting a core dump will fail.

\$c (current stack trace)  
\$<threadlist (dump all threads in the system)  
\$<cpus (show what all the cpus are doing)

---

## Summary

Diagnosing watchdog reset problems is not always an easy task to say the least. Watchdog resets ONLY occur when the system is processing traps, you can guarantee that it is going to be in the kernel trap handling code.

When you are in the PROM, you cannot access the kernel and therefore the symbol table is not available to the PROM. All the data from the PROM commands will be in hex. Once the system is back up and running, you can run adb on the system, and by using the command "address/i to display the instruction and location of each address taken from the stack trace.

Also by running the POST DEMON test on the 4d system, you can find out if hardware is the caused of your problem. The reason being that watchdog problems are caused by problematic software, which can be related or caused by a specific piece of hardware on your 4d system.

## Dragon (1000/2000) Hard Hangs

---

For Sparcserver 2000 or 1000 systems, it is possible to force a coredump even though the system is not responding to a keyboard abort sequence, but only if you have gone through a special procedure to handle the situation. This obviously is only useful if you are experiencing frequent hard hangs and need to obtain a core dump. The procedure relies on the fact that the kernel includes special code to detect a failure of the cooling system (the fan) and print a warning to the console in this case. The fan status is associated with the highest possible interrupt priority, which virtually guarantees that the interrupt will be processed even though the system appears to be hung completely.

---

**Note** – READ NOTICE AT BOTTOM OF THIS DOCUMENT BEFORE YOU PROCEED!

---

The basic procedure is to find the appropriate place in the kernel code to set a breakpoint, boot kadb and the kernel in single user mode, set the breakpoint, and let the system run. When the hang occurs, you can pull the power plug on the fan, thus generating a "fan failure" interrupt, and forcing a transfer of control to the system kernel debugger, kadb.

Finding the location is a bit complex; basically we need to identify a particular instruction location down in an interrupt service routine. In a function called "intr15\_bbus", the code looks like:

```
/*
 * if we're the 1st to notice, then handle it
 */
if (!handler_fan.detected && lock_try(&handler_fan.detected))
{
    DEBUG_INC(intr15_count_fan);
    handler_fan.detected_cpu = cpu_id;
```

```

/*
 * handler_fan.detected will be owned by the softint
 */
--->>>          if (!disable_fanfail)
TRIGGER_INTR(handler_fan);
}

```

Where the arrow points is the location to set the breakpoint. This is very close to the end of the function. To do this, you need to use `adb` to look at the kernel and obtain a location which you can use later in `kadb`.

Get the address: run the command:

```
adb /kernel/unix
```

(no special flags needed, nor do you need to do this as root).

To identify the appropriate instructions, you need to find the address of the variable which is being tested ("disable\_fanfail") and look for that particular address in the instruction stream. Get the address with the command:

```
disable_fanfail=X
```

This will print the value of the variable address in hex. For this writeup, we will use the value:

```

disable_fanfail=X
                e00eb7c0

```

(The contents are unimportant; all we want is the address). Now you need to locate the address in the instructions at the end of the `intr15_bbus()` function.

SPARC assembly language can be displayed in `adb` with the "i" command. What you are looking for is a "sethi" instruction, which is used to load up a register with the \*upper\* bits of a constant address. One of the restrictions of the assembly language is that it takes two separate instructions to put a full 32-bit address into a register. At this point, you are looking for a sethi instruction which takes the upper bits of the desired address. For this example, look for a constant which contains "0xe00eb???", where the '?' indicates digits we don't care about. (In actual fact, the sethi instruction loads in the upper 22 bits of the address, so the actual constant you should see in the instruction is 0xe00eb400).

Since this is at the end of the function, the easiest way to locate the code is to start at the next function and back up. The following function name is "intr15\_obp", so start there and look at the preceding locations. Use the "?ia" command to print out the instructions with their addresses.

The command `intr15_obp-50,18?ia` will go 0x50 bytes back from `intr15_obp` and will display 0x18 instruction codes.

```

intr15_obp-50,18?ia
intr15_bbus+0x22c: orcc      %g0, %o0, %g0
intr15_bbus+0x230: be       intr15_bbus + 0x274
intr15_bbus+0x234: sethi    %hi(0xe00eb400), %o7
intr15_bbus+0x238: sethi    %hi(0xe00d0c00), %o5
intr15_bbus+0x23c: ld      [%o7 + 0x3c0], %o7
intr15_bbus+0x240: orcc      %g0, %o7, %g0
intr15_bbus+0x244: bne      intr15_bbus + 0x274
intr15_bbus+0x248: st       %i0, [%o5 + 0x110]
intr15_bbus+0x24c: sethi    %hi(0xe00ffc00), %i0
intr15_bbus+0x250: sethi    %hi(0xe00d0c00), %i5
intr15_bbus+0x254: ld      [%i5 + 0x114], %i5 ! -0x1ff2f2ec
intr15_bbus+0x258: or       %l0, 0x3e0, %l0
intr15_bbus+0x25c: mov      0xff, %l1
intr15_bbus+0x260: mov      %i5, %o1
intr15_bbus+0x264: stb      %l1, [%i5 + %l0]
intr15_bbus+0x268: ld      [%g7 + 0x58], %o0
intr15_bbus+0x26c: call     xmit_cpu_intr
intr15_bbus+0x270: ld      [%o0], %o0
intr15_bbus+0x274: ret
intr15_bbus+0x278: restore
intr15_obp:      save      %sp, -0x60, %sp
intr15_obp+4:    call     poll_obp_mbox
intr15_obp+8:    restore
intr15_mxcc:     save      %sp, -0x60, %sp
intr15_mxcc+4:

```

In this sequence, you will notice 4 different sethi instructions, only one of which looks close:

```

intr15_bbus+0x234: sethi    %hi(0xe00eb400), %o7

```

This uses register "%o7" to hold the value of the constant. To be sure we have the right place, find the next instruction which utilizes %o7:

```

intr15_bbus+0x23c: ld      [%o7 + 0x3c0], %o7

```

This uses the value in %o7 plus 0x3c0, which we hope should be the address of the variable we are interested in. You can compute this to verify:

```

0xe00eb400+0x3c0?X
disable_fanfail:

text address not found

```

This printed an error message which we can ignore, but note that the address corresponding to that hex sum is indeed the variable `disable_fanfail`. You have the correct 'sethi' instruction. You will want to set the breakpoint at this location, `"intr15_bbus+0x234"`.

You now need to run through the following steps:

- Make sure the keyswitch (on the front panel) is in a position that will allow you to halt the machine from the keyboard. The "locked" setting will prevent users from aborting the machine from the console.
- Ensure that `savecore` is enabled. Edit `/etc/init.d/syssetup` and verify that the comments have been removed from the `savecore` command line. Also make sure that there is enough disk space to hold the core.
- Reboot the system, but boot the debugger instead of the regular kernel. You can supply boot flags to `kadb` and get the kernel to come up in single user mode:

```
# halt
ok boot kadb -s
```

- Let it come all the way up to "System Maintenance Mode" and give a shell prompt. At this point, killing/aborting the system will not drop to the prom, but will enter the debugger. Kill it:

```
L1-A (or 'break')
```

you should see a prompt `"kadb X"`, indicating "kadb is running on CPU X".

- At this time, set the breakpoint:

```
intr15_bbus+0x234:b
```

provide the address and the command `":b"` to set a breakpoint.

- Verify that the breakpoint is really there:

```
$b
```

This should print out a "table" of breakpoints, which contains exactly one at the magic address.

- Continue from `kadb` and get back to the kernel:

```
:c
```

- Hit return and verify that the system is alive and displaying normal single-user shell prompts.
- Control-D and let the system come up in normal operating mode. At this time, the regular kernel is running along with the debugger (which is permanently resident in memory) with a breakpoint set. The disk copy of the kernel is unchanged at this point, so any normal reboot will proceed as expected, and will NOT have the breakpoint set.
- If/when a hang occurs, pull the plug on the cooling fan in the 2000 system cabinet. This generates a high-priority (level 15) interrupt, which hits the breakpoint and enters `kadb`. (Put the plug back!)

- From the kadb prompt, you *could* look around at the kernel, but the most likely requirement is going to be for a core dump. Get from kadb out to the boot prom by exiting kadb itself:

```
$q
```

- This should get an "ok" prompt, at which point a sync command will cause the system to panic and dump core.

```
ok sync
```

- A "panic: zero" should occur, with the associated messages about dumping various pages to disk.
- The system will reboot normally and run savecore automatically to get a copy of the crash.

This is a one-shot procedure. The system, when it reboots, will NOT have any breakpoints set, so any panics or halts will result in a normal response.

As an alternative, if you don't wish to use kadb to examine anything in memory, breakpoints may be set directly from the prom as long as you have a hex (not symbolic) address to work with. Using the above procedure, print the address in hex as a last step:

```
intr15_bbus+0x234=X
```

This will print out the real address.

Drop into the prom (L1-A), set the breakpoint, and continue. (The advantage of this is that you can do it on a live system without rebooting, as long as it's pretty idle).

Verify that the breakpoint has been set correctly by looking at the contents of memory with adb:

```
# adb -k /dev/ksyms /dev/mem
intr15_bbus+0x234/i
```

The instruction printed by this should *not* be the 'sethi' which was originally there, but a 'ta' (TRAP) instruction. Setting a breakpoint involves actually replacing the code at that point by a special instruction to go to the kernel (or prom). This verifies that the new code is in place.

---

**Note** – You can only do this on the SC2000. For the SS1000, should add that if really desperate, the fans can be stopped by inserting a tool that will fit thru the holes in the sheetmetal and stop the fans. That doing so will likely require the fans to be replaced, so only do if really desperate...

---



## Setting Up and Using TIP

---

You can use the TTYA or TTYB ports on your SPARC system to connect to a second Sun workstation. This workstation can be either the same type of SPARC system or a different type of Sun workstation or server system. By connecting two systems in this way, you can use a shell window on the Sun workstation as a terminal to your SPARC system. (See the online tip manpage for detailed information about terminal connection to a remote host.)

The TIP method is recommended (over simply connecting to a dumb terminal), since it lets you use windowing and operating system features when working with the boot PROM. A communications program or another non-Sun computer can be used in the same way, if the program can match the output baud rate used by the PROM TTY port.

---

**Note** – In the following pages, "SPARC system" refers to your system, and "Sun workstation" refers to the system you are connecting to your system.

---

Use the following procedure to set up the TIP connection.

1. Connect the Sun workstation TTYB serial port to your SPARC system TTYA serial port using a serial connection cable. Use a 3-wire Null Modem Cable, and connect wires 3-2, 2-3, and 7-7. (Refer to your system installation manual for specifications on null modem cables.)
2. At the Sun workstation, add the following lines to the `/etc/remote` file.

If you are running a pre-Solaris 2.0 version of the operating environment, type:

```
hardware:\
      :dv=/dev/ttyb:br#9600:el=^C^S^Q^U^D:ie=%$:oe=^D
```

If you are running version 2.0 or 2.1 of the Solaris operating environment, type:

```
hardware:\
      :dv=/dev/term/b:br#9600:el=^C^S^Q^U^D:ie=%$:oe=^D:
```

3. In a shelltool window on the Sun workstation, type:

```
hostname% tip hardwire
connected
```

The Shell Tool window is now a TIP window directed to the Sun workstation TTYB.

---

**Note** – Use a Shell Tool, not a Command Tool; some TIP commands may not work properly in a Command Tool window.

---

4. At your SPARC system, enter the Forth Monitor so that the ok prompt is displayed.

If you do not have a video monitor attached to your SPARC system, connect the SPARC system TTYA to the Sun workstation TTYB and turn on the power to your SPARC system. Wait for a few seconds, and press Stop-A to interrupt the power-on sequence and start the Forth Monitor. Type n to get to the ok prompt. Unless the system is completely inoperable, the Forth Monitor is enabled, and you can continue with the next step in this procedure.

5. If you need to redirect the standard input and output to TTYA, type:

```
ok ttya io
```

There will be no echoed response.

6. Press Return on the Sun workstation keyboard. The ok prompt appears in the TIP window.

Typing ~# in the TIP window is equivalent to typing Stop-A at the SPARC system.

---

**Note** – Do not type Stop-A from a Sun workstation being used as a TIP window to your SPARC system. Doing so will abort the operating system on the workstation. (If you accidentally type Stop-A, you can recover by immediately typing either c at the > prompt or go at the ok prompt.)

---

7. When you are finished using the TIP window, end your TIP session and exit the window:

- a. Redirect the input and output to the screen and keyboard, if needed.

- b. In the TIP window, type:

```
ok ~.
hostname%
```

---

**Note** – Note - When entering ~ (tilde character) commands in the TIP window, ~ must be the first character entered on the line. To ensure that you are at the start of a new line, press Return first.

---

## Common Problems with TIP

This section describes solutions for TIP problems occurring in pre-Solaris 2.0 operating environments.

Problems with TIP may occur if:

- The lock directory is missing or incorrect.

There should be a directory named `/usr/spool/uucp`. The owner should be `uucp` and the mode should be `drwxr-sr-x`.

- TTYB is enabled for logins.

The status field for TTYB (or the serial port you are using) must be set to off in `/etc/ttytab`. Be sure to execute `kill -HUP 1` (see `init(8)`) as root if you have to change this entry.

- `/dev/ttyb` is inaccessible.

Sometimes, a program will have changed the protection of `/dev/ttyb` (or the serial port you are using) so that it is no longer accessible. Make sure that `/dev/ttyb` has the mode set to `crw-rw-rw-`.

- The serial line is in tandem mode.

If the TIP connection is in tandem mode, the operating system sometimes sends XON (^S) characters (particularly when programs in other windows are generating lots of output). The XON characters are detected by the Forth word `key?`, and can cause confusion. The solution is to turn off tandem mode with the `~s !tandem` TIP command.

- The `.cshrc` file generates text.

TIP opens a sub-shell to run `cat`, thus causing text to be attached to the beginning of your loaded file. If you use `dl` and see any unexpected output, check your `.cshrc` file.

You can certainly use other devices to TIP into the Sun4d system. For example there is the HP palmtop, most any laptops, pc's, macs. What it comes down to is that the software on these system use to TIP to the Sun4d systems needs to match the baud rate of that serial port.

9600,8,n,1 if your communication software can handle these settings then your good to go and you can TIP into the system.

Because you are using a shelltool for tip, we need to capture the data from the TIP session. Do the following to capture the data from the TIP session.

```
# script /tmp/foo
# tip hardwire
(do needed work)
# ctrl-d
```

Use vi /tmp/foo to look at data from TIP session. Don't forget to quit TIP session.