

# **Sun4v Hypervisor Core API Specification**

Revision 0.20-trim

March 16, 2005



## Table of Contents

|  |    |  |    |
|--|----|--|----|
| 1 Introduction.....                      | 3  | 7.2 API calls.....                                   | 16 |
| 1.1 Related specifications.....          | 3  | 8 MMU services.....                                  | 21 |
| 1.2 Additional specifications.....       | 3  | 8.1 Definition for translation table entry (TTE).... | 21 |
| 2 Hypervisor call conventions.....       | 5  | 8.2 Translation Storage Buffer (TSB) specification.. | 21 |
| 2.1 Hyper-fast traps.....                | 5  | 8.3 MMU flags.....                                   | 22 |
| 2.2 Fast traps.....                      | 5  | 8.4 MMU Fault status area.....                       | 22 |
| 2.3 Post hypervisor trap processing..... | 5  | 8.5 API calls.....                                   | 26 |
| 3 Common definitions.....                | 7  | 9 Cache and Memory services.....                     | 33 |
| 3.1 Trap numbers.....                    | 7  | 9.1 API calls.....                                   | 33 |
| 3.2 Function numbers for FAST_TRAP.....  | 7  | 10 Device interrupt services.....                    | 35 |
| 3.3 Function numbers for CORE_TRAPs..... | 8  | 10.1 Definitions.....                                | 35 |
| 3.4 Error codes.....                     | 8  | 10.2 API calls.....                                  | 35 |
| 3.5 Guest states.....                    | 8  | 11 TOD services.....                                 | 38 |
| 3.6 Initial guest environment.....       | 9  | 11.1 API calls.....                                  | 38 |
| 3.7 Privileged registers.....            | 9  | 12 Console services.....                             | 39 |
| 3.8 Other initial guest state.....       | 11 | 12.1 API calls.....                                  | 39 |
| 4 Machine description.....               | 12 | 13 Core dump services.....                           | 40 |
| 5 API versioning.....                    | 13 | 13.1 API calls.....                                  | 41 |
| 6 Domain services.....                   | 14 | 14 Trap trace services.....                          | 42 |
| 6.1 API call.....                        | 14 | 14.1 Trap trace buffer control structure.....        | 42 |
| 7 CPU services.....                      | 16 | 14.2 Trap trace buffer entry format.....             | 42 |
| 7.1 CPU id and CPU list.....             | 16 | 14.3 API calls.....                                  | 43 |

## 1 Introduction

This document details the calling conventions of the API provided to a sun4v domain by the underlying hypervisor, and the core functions common to all hypervisors. The intended audience for this document is operating system and firmware engineers porting to the sun4v architecture.

The API serves two principal purposes:

1. To enable the supervisor to request services and operations to be performed on its behalf by the hypervisor.
2. To inform the hypervisor of information it expects from the supervisor, for example the size and location of the interrupt delivery queues.

### 1.1 Related specifications

This document should be read in conjunction with the following specifications;

The Sun4v Architecture Specification describes the architectural model of the virtual machine environment provided through a conjunction of platform hardware and hypervisor software. It is to be read in addition to the Level-1 SPARC v9 specification. It supplants and extends the Level-2 SPARC v9 specification in describing the programming model, register and exception interfaces for privileged mode software.

The SunSPARC Specification describes the common hardware specification for SPARC processors. It is of primary interest to hypervisor implementors.

The Machine Description Specification documents the transport format and content by which the virtual machine environment implemented by a hypervisor is described to guest software. Many of the arguments provided to hypervisor API services should be derived from or have their constraints (e.g. maximum and minimum values) described by the machine description.

The Sun4v Error Specification documents the behavioral semantics of sun4v virtual machine environment, as well as the syntax of the error reports provided via the resumable and non-resumable error queue interfaces defined by the sun4v architecture.

### 1.2 Additional specifications

Hypervisor API services are divided into three categories; Core, Technology and, Platform Specific.

#### 1.2.1 Core API services

Core API services are common to all sun4v virtual machine environments.

#### 1.2.2 Technology API services

Technology API services are common to platforms implementing a specific technology requiring a sun4v/hypervisor interface. An example of this category is the PCI IO API specification, which is common to all platforms implementing a virtualized PCI root nexus capability.

### 1.2.3 Platform Specific API services

Platform Specific API services are unique to a platform or platform family only. Examples of such interfaces typically include API services to access performance counters, or processor specific features such as the cryptographic acceleration in Niagara-1.

## 2 Hypervisor call conventions

Hypervisor API calls are made through the use of a trap ( `TCC` ) instruction using *sw\_trap\_numbers* 0x80 and above. The calling convention has two forms; fast-trap and hyper-fast-trap. The principle difference between these two forms is whether the function number is passed in a register or is encoded in the trap instruction itself. The latter is the faster form, but has a limited number of possible functions, and is therefore reserved for performance critical operations only.

### 2.1 Hyper-fast traps

This trap mechanism encodes the API function number (0x80 + a 7bit value) in the `Tcc` instruction's *sw\_trap\_number* itself, and therefore provides the fastest possible method of reaching the actual function implementation. The calling convention is as follows:

| Register | Input      | Output        |
|----------|------------|---------------|
| %o0      | argument 0 | return status |
| %o1      | argument 1 | return value1 |
| %o2      | argument 2 | return value2 |
| %o3      | argument 3 | return value3 |
| %o4      | argument 4 | return value4 |

All arguments and return values are 64-bits unless explicitly stated by the description of a specific API service. Further arguments may be passed in memory, as defined on a per function call basis.

### 2.2 Fast traps

Fast traps are the preferred mechanism for hypervisor API calls. All fast trap API calls use *sw\_trap\_number* 0x80 in the `Tcc` instruction, with the required function number provided as a 64bit value in register %o5. The calling convention is as follows:

| Register | Input           | Output         |
|----------|-----------------|----------------|
| %o5      | function number | undefined      |
| %o0      | argument 0      | return status  |
| %o1      | argument 1      | return value 1 |
| %o2      | argument 2      | return value 2 |
| %o3      | argument 3      | return value 3 |
| %o4      | argument 4      | return value 4 |

All arguments and return values are 64-bits unless explicitly stated by the description of a specific API service. Further arguments may be passed in memory, as defined on a per function call basis.

### 2.3 Post hypervisor trap processing

The following convention is used, unless explicitly described for a particular API service:

- All API services resume executing at the next logical instruction after the service trap as with a *done* instruction.

- All sun4v defined registers are preserved across an API service except as explicitly stated below;
  - Registers providing arguments to an API service (including the function number %o5 for fast traps) should be considered volatile, and their values upon return are undefined unless they are explicitly specified on a per-service basis. Registers not used for passing arguments or returning values are preserved across the API service.
  - Upon return from the API service, the returned status is given in register %o0. A value of zero in %o0 indicates successful execution of the API service, all other values indicate an error status (as defined in section 3.4).
- If an invalid *sw\_trap\_number* is issued, or if an invalid function number is specified, the hypervisor will return with EBADTRAP (as defined in section 3.4) in %o0.
- All 64 bits of the argument or return values are significant.

### 3 Common definitions

#### 3.1 Trap numbers

The following are the *sw\_trap\_numbers* encoded in the Tcc instruction that enters the hypervisor:

|                 |      |
|-----------------|------|
| FAST_TRAP       | 0x80 |
| MMU_MAP_ADDR    | 0x83 |
| MMU_UNMAP_ADDR  | 0x84 |
| TTRACE_ADDENTRY | 0x85 |
| CORE_TRAP       | 0xff |

Unless assigned to technology or platform specific APIs all other trap numbers ( 0x86 to 0xfe inclusive) result in EBADTRAP being returned in %o0 as described in section 2.3.

#### 3.2 Function numbers for FAST\_TRAP

Function numbers for fast-traps are provided in %o5 as a 64-bit value. The following are the function numbers defined for the core API set:

|                      |      |
|----------------------|------|
| MACH_EXIT            | 0x00 |
| MACH_DESC            | 0x01 |
| MACH_SIR             | 0x02 |
| CPU_START            | 0x10 |
| CPU_STOP             | 0x11 |
| CPU_YIELD            | 0x12 |
| CPU_QCONF            | 0x14 |
| CPU_QINFO            | 0x15 |
| CPU_MYID             | 0x16 |
| CPU_STATE            | 0x17 |
| CPU_SET_RTBA         | 0x18 |
| CPU_GET_RTBA         | 0x19 |
| MMU_TSB_CTX0         | 0x20 |
| MMU_TSB_CTXNON0      | 0x21 |
| MMU_DEMAP_PAGE       | 0x22 |
| MMU_DEMAP_CTX        | 0x23 |
| MMU_DEMAP_ALL        | 0x24 |
| MMU_MAP_PERM_ADDR    | 0x25 |
| MMU_FAULT_AREA       | 0x26 |
| MMU_ENABLE           | 0x27 |
| MMU_UNMAP_PERM_ADDR  | 0x28 |
| MMU_TSB_CTX0_INFO    | 0x29 |
| MMU_TSB_CTXNON0_INFO | 0x2a |
| MEM_SCRUB            | 0x31 |
| MEM_SYNC             | 0x32 |
| CPU_MONDO_SEND       | 0x42 |
| TOD_GET              | 0x50 |
| TOD_SET              | 0x51 |
| CONS_GETCHAR         | 0x60 |
| CONS_PUTCHAR         | 0x61 |
| TTRACE_BUFCONF       | 0x90 |
| TTRACE_BUFINFO       | 0x91 |
| TTRACE_ENABLE        | 0x92 |
| TTRACE_FREEZE        | 0x93 |
| DUMP_BUFCONF         | 0x94 |
| DUMP_BUFINFO         | 0x95 |
| INTR_DEVINO2SYSINO   | 0xa0 |

|                 |      |
|-----------------|------|
| INTR_GETENABLED | 0xa1 |
| INTR_SETENABLED | 0xa2 |
| INTR_GETSTATE   | 0xa3 |
| INTR_SETSTATE   | 0xa4 |
| INTR_GETTARGET  | 0xa5 |
| INTR_SETTARGET  | 0xa6 |

Unless assigned to technology specific or platform specific APIs all other function numbers used for fast-traps result in EBADTRAP being returned in %o0 as described in section 2.3.

### 3.3 Function numbers for CORE\_TRAPs

CORE\_TRAP APIs follow the same calling conventions as FAST\_TRAP API services. The following are the function numbers defined for the core API set:

|             |      |
|-------------|------|
| API_VER     | 0x00 |
| API_PUTCHAR | 0x01 |
| API_EXIT    | 0x02 |

CORE\_TRAP function numbers are defined as followed:

API\_VER is defined in section 5.

API\_PUTCHAR is an alias for FAST\_TRAP function CONS\_PUTCHAR.

API\_EXIT is an alias for FAST\_TRAP function MACH\_EXIT.

### 3.4 Error codes

When a hypervisor API returns, unless explicitly described by the API service, the 64-bit value in %o0 will be one of the following error identification values.

|               |    |  |
|---------------|----|--|
| EOK           | 0  | Successful return                          |
| ENOCPU        | 1  | Invalid CPU id                             |
| ENORADDR      | 2  | Invalid real address                       |
| ENOINTR       | 3  | Invalid interrupt id                       |
| EBADPGSZ      | 4  | Invalid pagesize encoding                  |
| EBADTSB       | 5  | Invalid TSB description                    |
| EINVAL        | 6  | Invalid argument                           |
| EBADTRAP      | 7  | Invalid function number                    |
| EBADALIGN     | 8  | Invalid address alignment                  |
| EWOLDBLOCK    | 9  | Cannot complete operation without blocking |
| ENOACCESS     | 10 | No access to specified resource            |
| EIO           | 11 | I/O Error                                  |
| ECPUEERROR    | 12 | CPU is in error state                      |
| ENOTSUPPORTED | 13 | Function not supported                     |
| ENOMAP        | 14 | No mapping found                           |
| ETOOMANY      | 15 | Too many items specified / limit reached   |

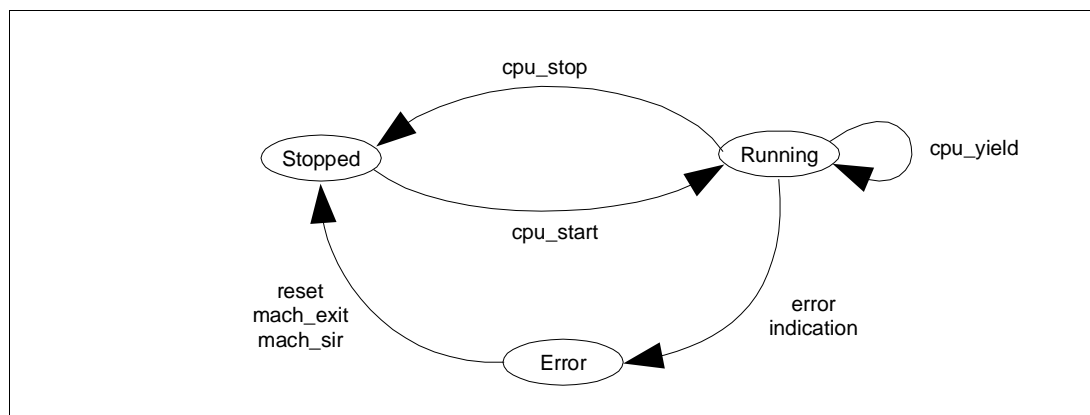
### 3.5 Guest states

As defined by the Sun4v Architecture Specification each virtual CPU can have one of three different states:

|         |  |
|---------|--|
| Stopped | CPU is stopped, not executing code, and may be started via the cpu_start API service |
| Running | CPU is executing   |
| Error   | CPU is in error, and no longer executing code  |

The relationship of these CPU states and hypervisor services may be summarized with the state diagram below:





### 3.6 Initial guest environment

The initial state of each sun4v virtual CPU is defined in the Sun4v Architecture Specification. Initial register state is duplicated here together with initial register configuration performed by the hypervisor for completeness.

### 3.7 Privileged registers

| Register(s) | Initial Value                             |
|-------------|---|
| %cwp        | 0   |
| %cansave    | NWIN-2                                    |
| %cleanwin   | NWIN-2                                    |
| %canrestore | 0   |
| %otherwin   | 0   |
| %wstate     | 0   |
| %pstate     | all 0 except pstate.priv=1, pstate.mm=tso |
| %tl         | MAXPTL (2)                                |
| %gl         | MAXPGL (2)                                |
| %pil        | MAXPIL (0xf)                              |
| %tba        | current rtba                              |
| %tt         | POR                                       |

#### 3.7.1 Non-Privileged Registers

| Register(s)                | Initial Value                          |
|----------------------------|--|
| %g1-%g7                    | 0                                      |
| %i0[%cwp]                  | real address of startup memory segment |
| %i1[%cwp]                  | siz eof startup memory segment         |
| %i2-%i7[%cwp]              | 0                                      |
| %i0-%i7[all other windows] | 0                                      |
| %d0-%d62                   | Binary 0                               |

| Register(s) | Initial Value |
|-------------|---------------|
| %fsr        | 0             |

### 3.7.2 Ancillary State Registers

| Register(s)         | Initial Value |
|---------------------|---------------|
| asr0 (%y)           | 0             |
| asr2 (%ccr)         | 0             |
| asr3 (%asi)         | ASI_REAL      |
| asr4 (%tikc         | >0, npt=0     |
| asr5 (%pc)          | current pc    |
| asr6 (%fprs)        | 0             |
| asr19 (%gsr)        | 0             |
| asr22 (%softint)    | 0             |
| asr24 (%stick)      | >0, npt=0     |
| asr25 (%stick_cmpr) | 0             |

### 3.7.3 Internal memory-mapped registers

| Register(s)                            | Initial Value   |
|--|-----------------|
| ASI_SCRATCHPAD, VA=0x00                | 0               |
| ASI_SCRATCHPAD, VA=0x08                | 0               |
| ASI_SCRATCHPAD, VA=0x10                | 0               |
| ASI_SCRATCHPAD, VA=0x18                | 0               |
| ASI_SCRATCHPAD, VA=0x20                | not implemented |
| ASI_SCRATCHPAD, VA=0x28                | not implemented |
| ASI_SCRATCHPAD, VA=0x30                | 0               |
| ASI_SCRATCHPAD, VA=0x38                | 0               |
| ASI_MMU, VA=0x08 (primary ctx)         | 0               |
| ASI_MMU, VA=0x10 (secondary ctx)       | 0               |
| ASI_MMU, VA=0xn08 (for valid {n} > 0)  | 0               |
| ASI_MMU, VA=0xn10 (for valid {n} > 0)  | 0               |
| ASI_QUEUE, VA=0x3c0 (cpu mondo head)   | 0               |
| ASI_QUEUE, VA=0x3c8 (cpu mondo tail)   | 0               |
| ASI_QUEUE, VA=0x3d0 (dev mondo head)   | 0               |
| ASI_QUEUE, VA=0x3d8 (dev mondo tail)   | 0               |
| ASI_QUEUE, VA=0x3e0 (res. error head)  | 0               |
| ASI_QUEUE, VA=0x3e8 (res. error tail)  | 0               |
| ASI_QUEUE, VA=0x3f0 (nres. error head) | 0               |
| ASI_QUEUE, VA=0x3f8 (nres. error tail) | 0               |

### 3.7.4 CPU-specific Registers

Platform specific performance counters will be configured such that exceptions/interrupts are disabled.

### 3.8 Other initial guest state

MMU state is disabled.

MMU fault status area location is undefined.

TSB info is undefined.

All queue base addresses and sizes are undefined.

One CPU is placed into the running state, all other CPUs are in the stopped state.

## 4 Machine description

*To describe the resources within a virtual machine (or logical domain), a data structure called a machine description is made available to a guest operating system. The machine description content and its binary format is currently described in a separate document - to be combined with this.*

## 5 API versioning

This section describes the API versioning interface available to all privileged code.

*This API interface is to be defined.*

## 6 Domain services

The following services enable privileged software to request information about or to affect the entire virtual machine domain.

### 6.1 API call

#### 6.1.1 mach\_exit

|           |           |
|-----------|-----------|
| trap#     | FAST_TRAP |
| function# | MACH_EXIT |
| arg0      | exit_code |

This service stops all CPUs in the virtual machine domain and places them into the *stopped* state. The 64-bit *exit\_code* may be passed to a service entity as the domain's exit status.

On systems without a service entity, the domain will undergo a reset, and the boot firmware will be reloaded.

This function will never return to the guest that invokes it.

*Note: by convention a exit\_code of zero denotes successful exit by the guest code. A non-zero exit\_code denotes a guest specific error indication.*

##### 6.1.1.1 Errors

This service does not return.

#### 6.1.2 mach\_desc

|           |           |
|-----------|-----------|
| trap#     | FAST_TRAP |
| function# | MACH_DESC |
| arg0      | buffer    |
| arg1      | length    |
| ret0      | status    |
| ret1      | length    |

This service copies the most current machine description into the buffer indicated by the real address in arg0. The buffer provided must be 16 byte aligned, and a minimum of 64bytes in size. In all cases upon return the actual size of the machine description is provided in the ret1 (length) return value.

*Note: A method of determining the appropriate buffer size for the machine description is to first call this service with a buffer length of 0 bytes. The service will always return the machine description size in ret1.*

##### 6.1.2.1 Errors

|           |                                       |
|-----------|---------------------------------------|
| EBADALIGN | Buffer is badly aligned               |
| ENORADDR  | Buffer is to an illegal real address. |

### 6.1.3 mach\_sir

|           |           |
|-----------|-----------|
| trap#     | FAST_TRAP |
| function# | MACH_SIR  |

This service provides a software initiated reset of a virtual machine domain. All CPUs are immediately captured, all hardware devices are returned to the entry default state, and the domain is restarted at the SIR (trap type 0x4) real trap table (rtba) entry point on one of the CPUs. Memory is preserved across this operation.

#### 6.1.3.1 Errors

This service does not return.

## 7 CPU services

CPU's represent devices that can execute software threads. A single chip that contains multiple cores or strands is represented as multiple CPU's with unique CPU identifiers. CPU's are exported to OBP via the machine description (and to Solaris via the device tree). CPU's are always in one of three states: *stopped*, *started*, or *error*.

### 7.1 CPU id and CPU list

Operations that are to be performed on multiple CPU's specify them via a CPU list. A CPU list is an array of 16-bit words in real memory, each of which is a CPU id.

CPU lists are passed through the API as 2 arguments: the first is the number of entries (16-bit words) in the CPU list, and the second is the (real address) pointer to the CPU id list.

### 7.2 API calls

#### 7.2.1 cpu\_start

|           |             |
|-----------|-------------|
| trap#     | FAST_TRAP   |
| function# | CPU_START   |
| arg0      | cpuid       |
| arg1      | pc          |
| arg2      | rtba        |
| arg3      | target_arg0 |
| ret0      | status      |

Start CPU with id *cpuid* with *pc* in %pc and with a real trap base address value of *rtba*. The indicated CPU must be in the *stopped* state. On successful completion, it will be in the *started* state. The target CPU will be supplied with *target\_arg0* in %o0 and *rtba* in %tba.

##### 7.2.1.1 Errors

|          |   |
|----------|---|
| ENOCPU   | Invalid <i>cpuid</i>                            |
| EINVAL   | Target <i>cpuid</i> is not in the stopped state |
| ENORADDR | Invalid <i>pc</i> or <i>rtba</i> real address   |



## 7.2.2 cpu\_stop

|           |           |
|-----------|-----------|
| trap#     | FAST_TRAP |
| function# | CPU_STOP  |
| arg0      | cpu       |
| ret0      | status    |

Stop CPU *cpu*. The indicated CPU must be in the *running* state. On completion, it will be in the *stopped* state. It is not legal to stop the current CPU.

**Programming note:** As this service cannot be used to stop the current cpu, this service may not be used to stop the last running CPU in a domain. To stop and exit a running domain the guest must use the `mach_exit` service.

### 7.2.2.1 Errors

|        |  |
|--------|--|
| ENOCPU | Invalid <i>cpu</i>                                   |
| EINVAL | target <i>cpu</i> is the current cpu                 |
| EINVAL | target <i>cpu</i> is not in the <i>running</i> state |

## 7.2.3 cpu\_set\_rtba

|           |               |
|-----------|---------------|
| trap#     | FAST_TRAP     |
| function# | CPU_SET_RTBA  |
| arg0      | rtba          |
| ret0      | status        |
| ret1      | previous_rtba |

Set the real trap base address of the local cpu to the value of *rtba*. Upon success the previous value of *rtba* is returned in *ret1*.

*Programming note: this service does not affect %tba*

### 7.2.3.1 Errors

|           |   |
|-----------|---|
| ENORADDR  | Invalid <i>rtba</i> real address                    |
| EBADALIGN | <i>rtba</i> is incorrectly aligned for a trap table |

## 7.2.4 cpu\_get\_rtba

|           |               |
|-----------|---------------|
| trap#     | FAST_TRAP     |
| function# | CPU_GET_RTBA  |
| ret0      | status        |
| ret1      | previous_rtba |

Returns the current value of *rtba* in *ret1*.

## 7.2.5 cpu\_yield

|           |           |
|-----------|-----------|
| trap#     | FAST_TRAP |
| function# | CPU_YIELD |
| ret0      | status    |

Suspend execution on the current CPU. Execution will resume when a interrupt (device, stick\_cmpr, or cross-call) is targeted to the CPU. On some CPUs, this API may be used by the hypervisor to save power by disabling hardware strands.

### 7.2.5.1 Errors

No possible error

## 7.2.6 cpu\_qconf

|           |            |
|-----------|------------|
| trap#     | FAST_TRAP  |
| function# | CPU_QCONF  |
| arg0      | queue      |
| arg1      | base raddr |
| arg2      | nentries   |
| ret0      | status     |

Configure queue *queue* to be placed at real address *base*, and of *nentries* entries. *nentries* must be a power of two number of entries. *Base* must be aligned exactly to match the queue size. Each queue entry is 64 bytes long, so for example, a 32 entry queue must be aligned on a 2048 byte real address boundary.

The specified queue is un-configured if *nentries* is 0.

For the current version of this API service the argument queue is defined as follows:

| queue | description               |
|-------|---------------------------|
| 0x3c  | cpu mondo queue           |
| 0x3d  | device mondo queue        |
| 0x3e  | resumable error queue     |
| 0x3f  | non-resumable error queue |

*Programming note: The maximum number of entries for each queue for a specific cpu may be determined from the machine description.*

### 7.2.6.1 Errors

|           |   |
|-----------|---|
| ENORADDR  | Invalid <i>base</i>   |
| EINVAL    | Invalid queue or,<br><i>nentries</i> not a power of two in number or,<br><i>nentries</i> is less than two or too large. |
| EBADALIGN | <i>baseaddr</i> is not correctly aligned for size   |

### 7.2.7 cpu\_qinfo

|           |            |
|-----------|------------|
| trap#     | FAST_TRAP  |
| function# | CPU_QINFO  |
| arg0      | queue      |
| ret0      | status     |
| ret1      | base_raddr |
| ret2      | nentries   |

Return the configuration info for queue *queue*. The *base\_raddr* is the currently defined read address base of the defined queue, and *nentries* is the size of the queue in terms of number of entries.

For the current version of this API service the argument *queue* is defined as follows:

| queue | description               |
|-------|---------------------------|
| 0x3c  | cpu mondo queue           |
| 0x3d  | device mondo queue        |
| 0x3e  | resumable error queue     |
| 0x3f  | non-resumable error queue |

If the specified queue is a valid queue number, but no queue has been defined this service will return success, but with *nentries* set to 0 and *base\_raddr* will have an undefined value.

#### 7.2.7.1 Errors

|        |               |
|--------|---------------|
| EINVAL | Invalid queue |
|--------|---------------|

### 7.2.8 cpu\_mondo\_send

|           |                |
|-----------|----------------|
| trap#     | FAST_TRAP      |
| function# | CPU_MONDO_SEND |
| arg0-1    | cpulist        |
| arg2      | data           |
| ret0      | status         |

Send a mondo interrupt to CPU list *cpulist* with 64 bytes of data pointed to by *data*. *data* must be a 64 byte aligned real address. The mondo data will be delivered to the *cpu\_mondo* queues of the recipient cpus.

In the event of a detected error, cpus in *cpulist* to which the mondo has been successfully delivered will be indicated by having their entry in *cpulist* updated with the value 0xffff.

#### 7.2.8.1 Errors

|             |  |
|-------------|--|
| EBADALIGN   | Mondo data is not 64byte aligned or cpulist is not 2byte aligned |
| ENORADDR    | Invalid data mondo address                                       |
| ENOCPU      | Invalid CPU in cpus  |
| EWOULDBLOCK | Some or all of the listed cpus did not receive the mondo         |

### 7.2.9 cpu\_myid

|           |           |
|-----------|-----------|
| trap#     | FAST_TRAP |
| function# | CPU_MYID  |
| ret0      | status    |
| ret1      | cpuid     |

Return the hypervisor ID handle for the current CPU. Used by a virtual cpu to discover its own identity.

### 7.2.10 cpu\_state

|           |           |
|-----------|-----------|
| trap#     | FAST_TRAP |
| function# | CPU_STATE |
| arg0      | cpuid     |
| ret0      | status    |
| ret1      | state     |

Retrieve the current running state of cpu *cpuid*. The running states are:

|                 |     |                           |
|-----------------|-----|---------------------------|
| CPU_STATE_IDLE  | 0x1 | cpu not started           |
| CPU_STATE_GUEST | 0x2 | cpu running guest code    |
| CPU_STATE_ERROR | 0x3 | cpu is in the error state |

#### 7.2.10.1 Errors

|        |                      |
|--------|----------------------|
| ENOCPU | Invalid CPU in cpuid |
|--------|----------------------|

## 8 MMU services

These hypervisor services control the behavior of address translations handled by the hypervisor.

A basic sun4v guest operating system, need not use any of these services at all. The default/initial operating environment for a guest is with virtual address translation disabled. In this mode all instructions and data references are made with real addresses.

If a guest operating system enables MMU translations, then virtual to real mappings may be specified in one of three different ways; either as permanent mappings, or as mappings that may be evicted and reloaded into system TLBs directly via MMU service functions, or indirectly via Translation Storage Buffers (TSBs). Moreover, with translations enabled, a guest Operating System must declare a Fault Status area for the hypervisor to provide information in the event of a translation fault.

### 8.1 Definition for translation table entry (TTE)

The format of a translation table entry (TTE) is defined in the Sun4v Architecture Specification.

### 8.2 Translation Storage Buffer (TSB) specification

The TSB functions control two sets of TSBs, one for when the virtual address context is zero, and one for when it is not zero. The demap functions remove translations from hardware TLBs. See the Address Model chapter in the sun4v Architecture Specification for more information on TSBs and TLBs.

A TSB description is a memory data structure that defines a single TSB:

| offset | size | contents                                |
|--------|------|---|
| 0      | 2    | page size to use for index shift in TSB |
| 2      | 2    | associativity of TSB                    |
| 4      | 4    | size of TSB in TTEs (16 bytes)          |
| 8      | 4    | context                                 |
| 12     | 4    | page size bitmask                       |
| 16     | 8    | real address of TSB base                |
| 24     | 8    | reserved                                |

The maximum TSB associativity supported is indicated in the machine description. An associativity value of 0xffff is reserved and should not be used for version 1.0 of this specification.

#### 8.2.1 Page sizes

The Sun4v Architecture Specification defines value encodings of page size for translation table entries (TTEs). The page size bitmask indicates which of these encodings may be specified for TTEs within a given TSB. For each bit in the page size bitmask, if set, the sun4v page size may be specified. For example, bit 0 corresponds to an 8KByte page size, bit 1 to a 64K page size, and so on in multiples of 8 of the page size for each bit in the field:

| Bit | Page size |
|-----|-----------|
| 0   | 8K        |
| 1   | 64K       |
| 2   | 512K      |
| 3   | 4MB       |
| 4   | 32MB      |
| 5   | 256MB     |
| 6   | 2GB       |

7 16GB

Bits 8 through 15 are reserved and must be set to zero.

The index shift page size indicates the page size to use for computing the TSB index for TTE retrieval. This value is the same as the page size value that may be specified in an individual sun4v TTE:

| Value | Page size assumed for index computation |
|-------|---|
| 0     | 8K                                      |
| 1     | 64K                                     |
| 2     | 512K                                    |
| 3     | 4MB                                     |
| 4     | 32MB                                    |
| 5     | 256MB                                   |
| 6     | 2GB                                     |
| 7     | 16GB                                    |

Values 8 though 15 are reserved. The index shift value must correspond to the smallest page size specified in the page size bit mask.

### 8.2.2 Context

This description enables TSBs to be defined where the context value for a page-translation is supplied within each entry of the TSB, or where a single value applies to the whole TSB. The latter enables a single TSB to be used for multiple context values (the context field within each TSB entry is required to be zero). The context field within the description selects which of these two modes the TSB is defined to use. If context contains a value between 1 and max\_context that is the context value applied to every entry in the TSB. If a context value of -1 is given in the TSB description, then the TSB is defined to use the context field within each TTE. Any other value supplied in context will return an EINVAL error. max\_context is provided by the machine description for each virtual cpu.

### 8.3 MMU flags

The MMU APIs are designed to function for both instruction and data address translations. Therefore, many of these interfaces take an MMU 'flags' argument in order to specify whether the operation is relevant to instruction or data mappings, or both. To ensure consistency between the MMU services this flags argument is defined here, and as follows:

The flags argument applies the API operation to instruction translations if bit 1 is set, and in addition applies the API operation to data translation entries if bit 0 is set. For every API service requiring a flags argument, at least one of bit 0 and/or bit 1 must be set.

**Implementation note:** For hardware implementations with unified instruction and data functions (for example; TLBs); Mapping an instruction translation entry may also cause an identical data translation entry to be mapped, and vice-versa even if not explicitly requests by the flags argument. Similarly, demapping an instruction translation entry may also cause the data translation entry to be demaped, and vice-versa even if not explicitly requested by the flags setting.

### 8.4 MMU Fault status area

MMU related faults have their status and fault address information placed into a memory region made available by privileged code. Like the TSBs above, the fault status area for **each** virtual processor is declared to the hypervisor via a hypervisor API call.

It is possible for MMU related faults to be delivered either by the hypervisor or directly

by processor hardware if so implemented. For this reason, the MMU fault area is arranged on an aligned address boundary with instruction and data fault fields arranged into distinct 64byte blocks.

The layout of the MMU fault status area is described in the table below:

| Offset (bytes) | Size (bytes) | Field                            |
|----------------|--------------|----------------------------------|
| 0x00           | 8            | Instruction fault type (IFT)     |
| 0x08           | 8            | Instruction fault address (IFA)  |
| 0x10           | 8            | Instruction fault context A(IFC) |
| 0x40           | 8            | Data fault type (DFT)            |
| 0x48           | 8            | Data fault address (DFA)         |
| 0x50           | 8            | Data fault context (DFC)         |

The definition of the values of the instruction and data fault type fields is as follows:

| Code                  | Fault type              |
|-----------------------|-------------------------|
| 1                     | fast miss               |
| 2                     | fast protection         |
| 3                     | MMU miss                |
| 4                     | invalid RA              |
| 5                     | privileged violation    |
| 6                     | protection violation    |
| 7                     | NFO access              |
| 8                     | so page/NFO side effect |
| 9                     | invalid VA              |
| 10                    | invalid ASI             |
| 11                    | nc atomic               |
| 12                    | privileged action       |
| 13                    | reserved                |
| 14                    | unaligned access        |
| 15                    | invalid page size       |
| 16 to -2              | reserved                |
| -1 (0xffffffffffffff) | multiple errors         |

For each MMU related trap, the fault status area is updated as follows; (a blank entry for IFT,IFA,IFC,DFT,DFA or DFC indicates the field is not updated for the particular condition and is therefore undefined, and '●' indicates the field is updated with the relevant fault type, address or context information for the trap).

| sun4v trap type              | Fault type                    | IFT | IFA | IFC | DFT | DFA | DFC | Comments   |
|------------------------------|-------------------------------|-----|-----|-----|-----|-----|-----|--|
| instruction_access_exception | invalid RA (0x4)              | •   | •   |     |     |     |     | instruction fetch to real address out of range                                       |
|                              | privilege violation (0x5)     | •   | •   | •   |     |     |     | non privileged instruction access to privileged page (TTE.p=1)                       |
|                              | NFO access (0x7)              | •   | •   | •   |     |     |     | instruction access to non-faulting load page (TTE.nfo=1)                             |
|                              | invalid VA (0x9)              | •   | •   | •   |     |     |     | instruction virtual access out of range  |
|                              | Invalid TSB entry             | •   | •   | •   |     |     |     | Hardware table walk found an invalid RA in a TTE loaded from a TSB                   |
|                              | Protection violation (0x6)    | •   | •   | •   |     |     |     | Instruction access to page without execute permission                                |
|                              | Multiple error (-1)           | •   |     |     |     |     |     | Hardware encountered multiple errors   |
| instruction_access_MMU_miss  | MMU miss (0x3)                | •   | •   | •   |     |     |     | TSB Miss   |
| data_access_exception        | invalid RA (0x4)              |     |     |     | •   | •   | •   | real address out of range  |
|                              | privilege violation (0x5)     |     |     |     | •   | •   | •   | Non-privileged data access to privileged page (TTE.p=1)                              |
|                              | NFO access (0x7)              |     |     |     | •   | •   | •   | Data access to non-faulting page (TTE.nfo=1) with ASI other than a non-faulting ASI. |
|                              | so page/NFO side effect (0x8) |     |     |     | •   | •   | •   | Non-faulting ASI data access to side-effect page (TTE.e=1)                           |
|                              | invalid VA (0x9)              |     |     |     | •   | •   | •   | Data or branch virtual access out of range   |
|                              | invalid ASI (0xa)             |     |     |     | •   | •   | •   | Invalid ASI for instruction  |
|                              | nc atomic (0xb)               |     |     |     | •   | •   | •   | Atomic access to non-cacheable page (TTE.cp=0)                                       |
|                              | privileged action (0xc)       |     |     |     | •   | •   | •   | Data access by non-privileged software using a privileged or hyper-privileged ASI    |
|                              | invalid page size (0xf)       |     |     |     | •   |     |     |  |
|                              | Multiple error (-1)           |     |     |     | •   |     |     | Hardware encountered multiple errors   |
| data_access_MMU_miss         | MMU miss (0x3)                |     |     |     | •   | •   | •   | TSB Miss   |
| data_access_protection       | protection violation (0x6)    |     |     |     | •   | •   | •   | store to non-writeable ??  |



| sun4v trap type   | Fault type              | IFT | IFA | IFC | DFT | DFA | DFC | Comments   |
|---|-------------------------|-----|-----|-----|-----|-----|-----|--|
| mem_address_not_aligned<br>LDDF_mem_address_not_aligned<br>STDF_mem_address_not_aligned<br>LDQF_mem_address_not_aligned<br>STQF_mem_address_not_aligned | unaligned access (0xe)  |     |     |     |     | ●   | ●   | Data access is not properly aligned                |
|   |                         |     |     |     |     | ●   | ●   |  |
|   |                         |     |     |     |     | ●   | ●   |  |
|   |                         |     |     |     |     | ●   | ●   |  |
|   |                         |     |     |     |     | ●   | ●   |  |
| fast_instruction_access_MMU_miss  | fast miss (0x1)         |     | ●   | ●   |     |     |     | TLB Miss   |
| fast_data_access_MMU_miss   | fast miss (0x1)         |     |     |     |     | ●   | ●   | TLB Miss   |
| fast_data_access_protection   | fast protection (0x2)   |     |     |     |     | ●   | ●   | Store data access to page without write permission |
| privileged_action   | privileged action (0xc) |     |     |     |     | ●   | ●   | Use of privileged ASI when pstate.priv = 0         |

## 8.5 API calls

### 8.5.1 mmu\_tsb\_ctx0

|           |              |
|-----------|--------------|
| trap#     | FAST_TRAP    |
| function# | MMU_TSB_CTX0 |
| arg0      | ntsb         |
| arg1      | tsbdptr      |
| ret0      | status       |

Configures the TSBs for the current CPU for virtual addresses with context zero. *tsbdptr* is a pointer to an array of *ntsbs* TSB descriptions.

A maximum of 16 TSBs maybe specified in the TSB description list.

#### 8.5.1.1 Errors

|          |   |
|----------|---|
| ENORADDR | Invalid TSB base in a TSB descriptor                              |
| EBADPGSZ | Invalid pagesize in a TSB descriptor                              |
| EBADTSB  | Invalid associativity or size in a TSB descriptor                 |
| EINVAL   | Invalid <i>ntsbs</i> , or<br>non-zero context in a TSB descriptor |

### 8.5.2 mmu\_tsb\_ctxnon0

|           |                 |
|-----------|-----------------|
| trap#     | FAST_TRAP       |
| function# | MMU_TSB_CTXNON0 |
| arg0      | ntsb            |
| arg1      | tsbdptr         |
| ret0      | status          |

Configures the TSBs for the current CPU for virtual addresses with non-zero contexts. *tsbdptr* is a pointer to an array of *ntsbs* TSB descriptions.

A maximum of 16 TSBs may be specified in the TSB description list.

#### 8.5.2.1 Errors

|          |   |
|----------|---|
| ENORADDR | Invalid TSB base in a TSB descriptor                          |
| EBADPGSZ | Invalid pagesize in a TSB descriptor                          |
| EBADTSB  | Invalid associativity or size in a TSB descriptor             |
| EINVAL   | Invalid <i>ntsbs</i> , or<br>zero context in a TSB descriptor |

### 8.5.3 mmu\_demap\_page

|           |                 |
|-----------|-----------------|
| trap#     | FAST_TRAP       |
| function# | MMU_DEMAP_PAGE  |
| arg0      | <i>reserved</i> |
| arg1      | <i>reserved</i> |
| arg2      | vaddr           |
| arg3      | context         |
| arg4      | flags           |
| ret0      | status          |

Demaps any page mapping of virtual address *vaddr* in context *context* for the current virtual CPU. Any virtual tagged caches are guaranteed to be kept consistent. The flags argument is defined according to section 8.3; “MMU flags”.

Arguments arg0 and arg1 are reserved and must be set zero.

#### 8.5.3.1 Errors

The implementation of this function is not required to check for all possible errors, and may return the following error codes:

|              |                                      |
|--------------|--------------------------------------|
| EINVAL       | Invalid vaddr, context or flag value |
| ENOTSUPPORED | arg0 or arg1 is non-zero             |

### 8.5.4 mmu\_demap\_ctx

|           |                 |
|-----------|-----------------|
| trap#     | FAST_TRAP       |
| function# | MMU_DEMAP_CTX   |
| arg0      | <i>reserved</i> |
| arg1      | <i>reserved</i> |
| arg2      | context         |
| arg3      | flags           |
| ret0      | status          |

Demaps all mappings for context *context* for the current virtual CPU. Any virtual tagged caches are guaranteed to be kept consistent. The flags argument is defined according to section 8.3; “MMU flags”.

Arguments arg0 and arg1 are reserved and must be set zero.

#### 8.5.4.1 Errors

The implementation of this function is not required to check for all possible errors, and may return the following error codes:

|              |                               |
|--------------|-------------------------------|
| EINVAL       | Invalid context or flag value |
| ENOTSUPPORED | arg0 or arg1 is non-zero      |

### 8.5.5 mmu\_demap\_all

|           |               |
|-----------|---------------|
| trap#     | FAST_TRAP     |
| function# | MMU_DEMAP_ALL |
| arg0      | reserved      |
| arg1      | reserved      |
| arg2      | flags         |
| ret0      | status        |

Demaps all translations for the current virtual CPU. Any virtual tagged caches are guaranteed to be kept consistent. The flags argument is defined according to section 8.3; “MMU flags”.

Arguments arg0 and arg1 are reserved and must be set zero.

#### 8.5.5.1 Errors

The implementation of this function is not required to check for all possible errors, and may return the following error codes:

|             |                          |
|-------------|--------------------------|
| EINVAL      | Invalid flag value       |
| ENOTSUPPRED | arg0 or arg1 is non-zero |

### 8.5.6 mmu\_map\_addr

|       |              |
|-------|--------------|
| trap# | MMU_MAP_ADDR |
| arg0  | vaddr        |
| arg1  | context      |
| arg2  | TTE          |
| arg3  | flags        |
| ret0  | status       |

This API service creates a non-permanent mapping using the TTE to virtual address *vaddr* for *context* for the calling virtual CPU. The flags argument is defined according to section 8.3; “MMU flags”.

*Note: This API call is intended both for privileged code that does not use TSBs, and for privileged code to specify temporary translation mappings.*

#### 8.5.6.1 Errors

The implementation of this function is not required to check for all possible errors, and may return the following error codes:

|          |                                       |
|----------|---------------------------------------|
| EINVAL   | Invalid vaddr, context, or flag error |
| EBADPGSI | Invalid page size value               |
| ENORADDR | Invalid real address in TTE           |

### 8.5.7 mmu\_map\_perm\_addr

|           |                   |
|-----------|-------------------|
| trap#     | FAST_TRAP         |
| function# | MMU_MAP_PERM_ADDR |
| arg0      | vaddr             |
| arg1      | context           |
| arg2      | TTE               |
| arg3      | flags             |
| ret0      | status            |

This API service creates a permanent mapping using the TTE to virtual address *vaddr* for *context* for the calling virtual CPU.

A maximum of 8 such permanent mappings may be specified by privileged code. Mappings may be removed with **mmu\_unmap\_perm\_addr** below.

The flags argument is defined according to section 8.3; “MMU flags”.

*Note: This API call used used to specify address space mappings for which privileged code does not expect to receive misses. For example, this mechanism can be used to map kernel nucleus code and data.*

#### 8.5.7.1 Errors

|          |  |
|----------|--|
| EINVAL   | Invalid vaddr, context, or flag error    |
| EBADPGSI | Invalid page size value                  |
| ENORADDR | Invalid real address in TTE              |
| ETOOMANY | Too many mappings (maximum of 8 reached) |

### 8.5.8 mmu\_unmap\_addr

|       |                |
|-------|----------------|
| trap# | MMU_UNMAP_ADDR |
| arg0  | vaddr          |
| arg1  | context        |
| arg2  | flags          |
| ret0  | status         |

Demaps virtual address *vaddr* in context *context* on this CPU. This function is intended to be used to demap pages mapped with **mmu\_map\_addr**. This service is equivalent to invoking **mmu\_demap\_page** with only the current CPU in the CPU list.

The flags argument is defined according to section 8.3; “MMU flags”.

Attempting to perform an unmap operation for a previously defined permanent mapping will have undefined results.

#### 8.5.8.1 Errors

The implementation of this function is not required to check for all possible errors, and may return the following error codes:

|        |                                      |
|--------|--------------------------------------|
| EINVAL | Invalid vaddr, context or flag value |
|--------|--------------------------------------|

### 8.5.9 mmu\_unmap\_perm\_addr

|           |                     |
|-----------|---------------------|
| trap#     | FAST_TRAP           |
| function# | MMU_UNMAP_PERM_ADDR |
| arg0      | vaddr               |
| arg1      | context             |
| arg2      | flags               |
| ret0      | status              |

Demaps any permanent page mapping (established via `mmu_map_perm_addr`) of virtual address *vaddr* in context *context* for the current virtual CPU. Any virtual tagged caches are guaranteed to be kept consistent.

The flags argument is defined according to section 8.3; “MMU flags”.

#### 8.5.9.1 Errors

|        |                                      |
|--------|--------------------------------------|
| EINVAL | Invalid vaddr, context or flag value |
|--------|--------------------------------------|

### 8.5.10 mmu\_fault\_area

|           |                               |
|-----------|-------------------------------|
| trap#     | FAST_TRAP                     |
| function# | MMU_FAULT_AREA                |
| arg0      | raddr                         |
| ret0      | status                        |
| ret1      | previous mmu fault area raddr |

Configure the MMU fault status area for the calling CPU. A 64 byte aligned real address specifies where MMU fault status information is placed. The return value is the previously specified area, or 0 for the first invocation. Specifying a fault area at real address 0 is not allowed.

#### 8.5.10.1 Errors

|           |                                  |
|-----------|----------------------------------|
| ENORADDR  | Invalid real address             |
| EBADALIGN | Invalid alignment for fault area |

### 8.5.11 mmu\_enable

|           |               |
|-----------|---------------|
| trap#     | FAST_TRAP     |
| function# | MMU_ENABLE    |
| arg0      | enable_flag   |
| arg1      | return_target |
| ret0      | status        |

This function either enables or disables virtual address translation for the calling CPU within the virtual machine domain. If the *enable\_flag* is zero, translation is disabled, any non-zero value will enable translation.

When this function returns, the newly selected translation mode will be active. The argument *return\_target* is a virtual address if translation is being enabled, or *return\_target* is a real address in the event that translation is to be disabled.

Upon successful completion, this API service will return control to the *return\_target* address with the new operating mode. In the event of call failure, the previous operating mode remains, and the service simply returns to the caller with the appropriate error code in *ret0*.

#### 8.5.11.1 Errors

|           |  |
|-----------|--|
| ENORADDR  | Invalid real address when disabling translation  |
| EBADALIGN | <i>return_target</i> is not aligned to an instruction                                      |
| EINVAL    | <i>enable_flag</i> requests current operating mode;<br>(e.g. disable if already disabled). |

### 8.5.12 mmu\_tsb\_ctx0\_info

|           |                   |
|-----------|-------------------|
| trap#     | FAST_TRAP         |
| function# | MMU_TSB_CTX0_INFO |
| arg0      | maxtsbs           |
| arg1      | bufferptr         |
| ret0      | status            |
| ret1      | ntsbs             |

This function returns the TSB configuration as previously defined by **mmu\_tsb\_ctx0** into the buffer provided by *arg1*. The size of the buffer is given in *arg1* in terms of number of TSB description entries.

Upon return, *ret1* contains the number of TSB descriptions previously configured. If the supplied buffer was too small, then *EINVAL* is returned in *ret0*, otherwise *EOK* is returned and *ret1* TSB descriptions have been copied into the buffer. If zero TSBs were configured, then *EOK* is returned with *ret1* containing 0.

### 8.5.13 mmu\_tsb\_ctxnon0\_info

|           |                      |
|-----------|----------------------|
| trap#     | FAST_TRAP            |
| function# | MMU_TSB_CTXNON0_INFO |
| arg0      | maxtsbs              |
| arg1      | bufferptr            |
| ret0      | status               |
| ret1      | ntsbs                |

This function returns the TSB configuration as previously defined by **mmu\_tsb\_ctx0** into the buffer provided by arg1. The size of the buffer is given in arg1 in terms of number of TSB description entries.

Upon return, ret1 contains the number of TSB descriptions previously configured. If the supplied buffer was too small, then EINVAL is returned in ret0, otherwise EOK is returned and ret1 TSB descriptions have been copied into the buffer. If zero TSBs were configured, then EOK is returned with ret1 containing 0.

#### 8.5.13.1 Errors

|        |                     |
|--------|---------------------|
| EINVAL | Invalid buffer size |
|--------|---------------------|



## 9 Cache and Memory services

In general, caches and memory are not exposed to the supervisor, although they are described to it in the machine description.

### 9.1 API calls

#### 9.1.1 mem\_scrub

|           |                 |
|-----------|-----------------|
| trap#     | FAST_TRAP       |
| function# | MEM_SCRUB       |
| arg0      | raddr           |
| arg1      | length          |
| ret0      | status          |
| ret1      | length scrubbed |

This service zeros the memory contents for the memory address range raddr to raddr+length. It also creates a valid error-checking code for the memory address range raddr to raddr+length-1.

This service starts scrubbing at raddr, but may scrub less than length bytes of memory. On success the actual length scrubbed is returned in ret1.

The arguments raddr and length must be aligned to an 8K page boundary or must contain the start address and length from a sun4v error report.

*Note: There are two uses for this function: The first use is to block clear and initialize memory and the second is to scrub an uncorrectable error reported via a resumable or non-resumable trap. The second use requires the arguments to be equal to the raddr and length provided in a sun4v memory error report.*

##### 9.1.1.1 Errors

|           |   |
|-----------|---|
| ENORADDR  | Invalid raddr   |
| EBADALIGN | Either the start address or length are not correctly aligned. |
| EINVAL    | length == 0   |

## 9.1.2 mem\_sync

|           |               |
|-----------|---------------|
| trap#     | FAST_TRAP     |
| function# | MEM_SYNC      |
| arg0      | raddr         |
| arg1      | length        |
| ret0      | status        |
| ret1      | length synced |

For the memory address range *raddr* to *raddr+length-1*, this service forces the next access within that range to be fetched from main system memory.

This service starts syncing at *raddr*, but may sync less than *length* bytes of memory. On success the actual length synced is returned in ret1.

The arguments *raddr* and *length* must be aligned to an 8K page boundary.

### 9.1.2.1 Errors

|           |   |
|-----------|---|
| ENORADDR  | Invalid <i>raddr</i>  |
| EBADALIGN | Either the start address or length are not correctly aligned. |
| EINVAL    | <i>length</i> == 0  |

## 10 Device interrupt services

Device interrupts are allocated to system bus bridges by the hypervisor, and described to the boot firmware in the machine description. OBP then describes them to Solaris via the device tree. The services described here are the generic interrupt services only, it is expected that the system bus nexus drivers will have additional APIs for functions that are specific to that bridge.

### 10.1 Definitions

These definitions apply to the following services:

- cpuid** A unique opaque value which represents a target cpu.
- devhandle** Device handle. The device handle uniquely identifies a sun4v device. It consists of the the lower 28-bits of the hi-cell of the first entry of the sun4v device's "reg" property as defined by the Sun4v Bus Binding to Open Firmware.
- devino** Device interrupt number. Specifies the relative interrupt number within the device. The value is the same as the values in the "interrupts" property or "interrupt-map" property in the sun4v device. The unique combination of devhandle and devino are used to identify a specific device interrupt.
- sysino** System Interrupt Number. A 64-bit unsigned integer representing a unique interrupt within a virtual machine.

**intr\_state** A flag representing the interrupt state for a given sysino. The state values are defined as:

| Name           | Value | Definition                     |
|----------------|-------|--------------------------------|
| INTR_IDLE      | 0     | Nothing Pending                |
| INTR_RECEIVED  | 1     | Interrupt received by hardware |
| INTR_DELIVERED | 2     | Interrupt delivered to queue   |

**intr\_enabled** A flag representing the 'enabled' state for a given sysino. The state values are defined as:

| Name          | Value | Definition         |
|---------------|-------|--------------------|
| INTR_DISABLED | 0     | sysino not enabled |
| INTR_ENABLED  | 1     | sysino enabled     |

### 10.2 API calls

### 10.2.1 intr\_devino\_to\_sysino

|           |                    |
|-----------|--------------------|
| trap#     | FAST_TRAP          |
| function# | INTR_DEVINO2SYSINO |
| arg0      | devhandle          |
| arg1      | devino             |
| ret0      | status             |
| ret1      | sysino             |

Converts a device specific interrupt number given by the arguments *devhandle* and *devino* into a system specific ino (*sysino*).

#### 10.2.1.1 Errors

|        |                          |
|--------|--------------------------|
| EINVAL | Invalid devhandle/devino |
|--------|--------------------------|

### 10.2.2 intr\_getenabled

|           |                 |
|-----------|-----------------|
| trap#     | FAST_TRAP       |
| function# | INTR_GETENABLED |
| arg0      | sysino          |
| ret0      | status          |
| ret1      | intr_enabled    |

Returns state in *intr\_enabled* if the interrupt defined by *sysino* is enabled (1) or disabled (0).

#### 10.2.2.1 Errors

|        |                |
|--------|----------------|
| EINVAL | Invalid sysino |
|--------|----------------|

### 10.2.3 intr\_setenabled

|           |              |
|-----------|--------------|
| trap#     | FAST_TRAP    |
| function# | INTR_ENABLED |
| arg0      | sysino       |
| arg1      | intr_enabled |
| ret0      | status       |

Sets the 'enabled' state of the interrupt defined by the argument *sysino* to the state defined by the argument *intr\_enabled*.

#### 10.2.3.1 Errors

|        |                                      |
|--------|--------------------------------------|
| EINVAL | Invalid sysino or intr_enabled value |
|--------|--------------------------------------|

### 10.2.4 intr\_getstate

|           |               |
|-----------|---------------|
| trap#     | FAST_TRAP     |
| function# | INTR_GETSTATE |
| arg0      | sysino        |
| ret0      | status        |
| ret1      | intr_state    |

Returns the current state of the interrupt given by the *sysino* argument.

#### 10.2.4.1 Errors

|        |                |
|--------|----------------|
| EINVAL | Invalid sysino |
|--------|----------------|

**10.2.5 intr\_setstate**

|           |               |
|-----------|---------------|
| trap#     | FAST_TRAP     |
| function# | INTR_SETSTATE |
| arg0      | sysino        |
| arg1      | intr_state    |
| ret0      | status        |

Sets the current state of the interrupt given by the *sysino* argument to the value given in the argument *intr\_state*.

Note: Setting the state to INTR\_IDLE clears any pending interrupt for *sysino*.

**10.2.5.1 Errors**

|        |                |
|--------|----------------|
| EINVAL | Invalid sysino |
|--------|----------------|

**10.2.6 intr\_gettarget**

|           |                |
|-----------|----------------|
| trap#     | FAST_TRAP      |
| function# | INTR_GETTARGET |
| arg0      | sysino         |
| ret0      | status         |
| ret1      | cpuid          |

Returns the *cpuid* that is the current target of the interrupt given by the *sysino* argument.

The *cpuid* value returned is undefined if the target has not been set via *intr\_settarget*.

**10.2.6.1 Errors**

|        |                |
|--------|----------------|
| EINVAL | Invalid sysino |
|--------|----------------|

**10.2.7 intr\_settarget**

|           |                |
|-----------|----------------|
| trap#     | FAST_TRAP      |
| function# | INTR_SETTARGET |
| arg0      | sysino         |
| arg1      | cpuid          |
| ret0      | status         |

Set the target cpu for the interrupt defined by the argument *sysino* to the target cpu value defined by the argument *cpuid*.

**10.2.7.1 Errors**

|        |                |
|--------|----------------|
| EINVAL | Invalid sysino |
| ENOCPU | Invalid cpuid  |

## 11 TOD services

The TOD is maintained by the hypervisor on a per-domain basis. Setting the TOD in one domain does not affect any other domain.

Time is described by a single unsigned 64-bit word equivalent to a `time_t` for the Unix `time(2)` system call. The word contains the time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds.

### 11.1 API calls

#### 11.1.1 `tod_get`

|           |             |
|-----------|-------------|
| trap#     | FAST_TRAP   |
| function# | TOD_GET     |
| ret0      | status      |
| ret1      | time-of-day |

Returns the current time-of-day. May block if TOD access is temporarily not possible.

##### 11.1.1.1 Errors

EWOULDBLOCK

#### 11.1.2 `tod_set`

|           |           |
|-----------|-----------|
| trap#     | FAST_TRAP |
| function# | TOD_SET   |
| arg0      | tod       |
| ret0      | status    |

The current time-of-day is set to the value specified in `arg0`. May block if TOD access is temporarily not possible.

##### 11.1.2.1 Errors

EWOULDBLOCK

## 12 Console services

This section describes the API services provided for a guest console.

### 12.1 API calls

#### 12.1.1 cons\_getchar

|           |              |
|-----------|--------------|
| trap#     | FAST_TRAP    |
| function# | CONS_GETCHAR |
| ret0      | status       |
| ret1      | character    |

Returns a character from the console device. If no character is available then an EWOULDBLOCK error is returned. If a character is available, then the returned status is EOK and the character value is in ret1. A virtual BREAK is represented by the 64-bit value -1

##### 12.1.1.1 Errors

|             |                        |
|-------------|------------------------|
| EWOULDBLOCK | No character available |
|-------------|------------------------|

#### 12.1.2 cons\_putchar

|           |              |
|-----------|--------------|
| trap#     | FAST_TRAP    |
| function# | CONS_PUTCHAR |
| arg0      | char         |
| ret0      | status       |

Write a character to the console device. Currently only character values between 0 and 255 may be used.

##### 12.1.2.1 Errors

|             |   |
|-------------|---|
| EINVAL      | Illegal character                         |
| EWOULDBLOCK | Output buffer currently full, would block |

## 13 Core dump services

When privileged code in a domain crashes/panics it may provide a capability to dump its internal state for later debugging. Such “core dumps” can be provided back to Sun from the field to help diagnose field problems. However the hypervisor virtualizes much of the platform hardware, thus obscuring information about the physical resources that can be useful in diagnosing hardware bugs.

Instead of adding a core dumping capability to the hypervisor, this API allows the domain's privileged code to dump platform and hypervisor-specific information as part of its own core dumping procedure. Privileged code allocates a section of its own memory space and informs the hypervisor that this may be used as a “dump buffer” for the hypervisor to place hypervisor specific debug/dump information.

Once declared, a dump buffer can be used at any time by the hypervisor to record private debug information, thus avoiding having such logs within the hypervisor itself.

The required size of the dump buffer is provided to the domain as part of the initial machine description.

During a core-dump operation, a guest requests that the hypervisor update any information in the dump buffer in preparation to being dumped as part of the domain's memory image.

Dump buffer information is highly platform and hypervisor specific. The format and content of the buffer are hypervisor private and should not be considered useable by sun4v code. Some platform hypervisors may provide no dump buffer information for security reasons.



## 13.1 API calls

### 13.1.1 dump\_buf\_conf

|           |                           |
|-----------|---------------------------|
| trap#     | FAST_TRAP                 |
| function# | DUMP_BUFCONF              |
| arg0      | raddr                     |
| arg1      | size                      |
| ret0      | status                    |
| ret1      | size of buffer on success |

This function declares a domain dump buffer to the hypervisor. The *raddr* supplies the real base address of the dump-buffer and must be 64byte aligned.

The *size* field specifies the size of the dump buffer allocated, and may be larger than the minimum size specified in the machine description.

A size of 0 unconfigures the dump buffer.

This function may be called any number of times so that a guest may relocate a dump buffer, or create “snapshots” of any dump-buffer information. Each call to `dump_buf_conf` implicitly performs a synchronization so as to be atomic with the declaration of the new dump buffer.

If *raddr* is an illegal or badly aligned real address, then any currently active dump buffer is disabled (equivalent to passing a size of 0) and an error is returned.

In the event that the call fails, `ret1` contains the minimum size required by the hypervisor for a valid dump buffer.

#### 13.1.1.1 Errors

|               |  |
|---------------|--|
| EOK           | Dump buffer was configured, <code>ret1</code> contains |
| required size |  |
| ENORADDR      | Invalid <i>raddr</i>                                   |
| EBADALIGN     | <i>raddr</i> not aligned on 64byte boundary            |

### 13.1.2 dump\_buf\_info

|           |                                     |
|-----------|-------------------------------------|
| trap#     | FAST_TRAP                           |
| function# | DUMP_BUFINFO                        |
| ret0      | status                              |
| ret1      | real address of current dump buffer |
| ret2      | size of current dump buffer         |

#### 13.1.2.1 Errors

|        |  |
|--------|--|
| EINVAL | No dump buffer is currently configured |
|--------|--|

## 14 Trap trace services

The hypervisor provides limited trap tracing capability for privileged code running on each virtual CPU.

Privileged code provides a round-robin trap trace queue within which the hypervisor writes 64 byte entries detailing hyperprivileged traps taken on behalf of privileged code. This is provided as a debugging capability for privileged code.

The hypervisor provides limited trap tracing capability for privileged code running on each virtual CPU.

### 14.1 Trap trace buffer control structure

The trap trace control structure is 64 bytes long and placed at the start (offset 0) of the trap trace buffer.

The format of the control structure is as follows:

| Offset | Size | Field definition |
|--------|------|------------------|
| 0x00   | 8    | Head offset      |
| 0x08   | 8    | Tail offset      |
| 0x10   | 0x30 | Reserved         |

The head offset is the offset of the most recently completed entry in the trap-trace buffer. The tail offset is the offset of the next entry to be written.

### 14.2 Trap trace buffer entry format

Trap trace entries all have the following format:

| Offset | Size | Name                 | Description                         |
|--------|------|----------------------|-------------------------------------|
| 0 x0   | 1    | TTRACE_ENTRY_TYPE    | Indicates hypervisor or guest entry |
| 0x01   | 1    | TTRACE_ENTRY_HPSTATE | Hyper-privileged state              |
| 0x02   | 1    | TTRACE_ENTRY_TL      | Trap level                          |
| 0x03   | 1    | TTRACE_ENTRY_GL      | Global register level               |
| 0x04   | 2    | TTRACE_ENTRY_TT      | Trap type                           |
| 0x06   | 2    | TTRACE_ENTRY_TAG     | Extended trap identifier            |
| 0x08   | 8    | TTRACE_ENTRY_TSTATE  | Trap state                          |
| 0x10   | 8    | TTRACE_ENTRY_TICK    | Tick                                |
| 0x18   | 8    | TTRACE_ENTRY_TPC     | Trap PC                             |
| 0x20   | 8    | TTRACE_ENTRY_F1      | Entry specific                      |
| 0x28   | 8    | TTRACE_ENTRY_F2      | Entry specific                      |
| 0x30   | 8    | TTRACE_ENTRY_F3      | Entry specific                      |
| 0x38   | 8    | TTRACE_ENTRY_F4      | Entry specific                      |

For each entry the TTRACE\_ENTRY\_TYPE field value is defined as follows:

| Value | Name              | Description                             |
|-------|-------------------|---|
| 0x00  | TTRACE_TYPE_UNDEF | Entry content undefined                 |
| 0x01  | TTRACE_TYPE_HV    | Hypervisor trap entry                   |
| 0xff  | TTRACE_TYPE_GUEST | Guest entry via ttrace_addentry service |

## 14.3 API calls

### 14.3.1 ttrace\_bufconf

```

trap#           FAST_TRAP
function#       TTRACE_BUFCONF
arg0            raddr

```

The trap trace buffer and entry format is described in Section “” below.

```

arg1            size

ret0            status
ret1            ret_size

```

This function requests hypervisor trap tracing and declares a virtual cpu's trap trace buffer to the hypervisor. The *raddr* supplies the real base address of the trap trace queue and must be 64byte aligned.

The *size* field specifies the size of the buffer allocated. A size of zero disables trap tracing for the calling virtual cpu. The buffer allocated must be sized for a power of two number of 64 byte trap trace entries plus an initial 64 byte control structure. For further detail see Section “” below.

This function may be called any number of times so that a virtual cpu may relocate a trap trace buffer, or create “snapshots” of information.

If *raddr* is an illegal or badly aligned real address, then trap tracing is disabled (equivalent to passing a size of 0) and an error is returned.

Upon failure this service call will return the minimum size of buffer required in *ret1*, and will return the actual size to be used on success.

#### 14.3.1.1 Errors

```

ENORADDR       Invalid raddr, or size too small
EBADALIGN      raddr not aligned on 64byte boundary

```

### 14.3.2 ttrace\_bufinfo

|           |                |
|-----------|----------------|
| trap#     | FAST_TRAP      |
| function# | TTRACE_BUFINFO |
| ret0      | status         |
| ret1      | raddr          |
| ret2      | size           |

This function returns the size and location of the previously declared trap-trace buffer. In the event that no buffer was previously declared, or the buffer disabled (e.g. via a `ttrace_bufconf` call with a size of zero), this call will fail.

#### 14.3.2.1 Errors

|        |                              |
|--------|------------------------------|
| EINVAL | No buffer currently definedd |
|--------|------------------------------|

### 14.3.3 ttrace\_enable

|           |                       |
|-----------|-----------------------|
| trap#     | FAST_TRAP             |
| function# | TTRACE_ENABLE         |
| arg0      | enable                |
| ret0      | status                |
| ret1      | previous enable state |

This function enables (or disables) trap tracing, returning the previously enabled state in `ret1`. Future systems may define various flags for the enable argument (`arg0`), for the moment a guest should pass `(uint64_t)-1` to enable, and `(uint64_t)0` to disable all tracing - which will ensure future compatibility.

#### 14.3.3.1 Errors

|        |                             |
|--------|-----------------------------|
| EINVAL | No buffer currently defined |
|--------|-----------------------------|

### 14.3.4 ttrace\_freeze

|           |                       |
|-----------|-----------------------|
| trap#     | FAST_TRAP             |
| function# | TTRACE_FREEZE         |
| arg0      | frozen (boolean)      |
| ret0      | status                |
| ret1      | previous freeze state |

This function freezes (or unfreezes) trap tracing, returning the previously freeze state in `ret1`. Future systems may define various flags for the enable argument (`arg0`), for the moment a guest should pass `(uint64_t)-1` to enable, and `(uint64_t)0` to freeze all tracing - which will ensure future compatibility.

#### 14.3.4.1 Errors

|        |                             |
|--------|-----------------------------|
| EINVAL | No buffer currently defined |
|--------|-----------------------------|

### 14.3.5 ttrace\_addentry

|           |                 |
|-----------|-----------------|
| trap#     | FAST_TRAP       |
| function# | TTRACE_ADDENTRY |
| arg0      | tag (16-bits)   |
| arg1      | data word 0     |
| arg2      | data word 1     |
| arg3      | data word 2     |
| arg4      | data word 3     |
| ret0      | status          |

This function adds an entry to the trap trace buffer. Upon return only arg0/ret0 is modified - none of the other registers holding arguments are volatile across this hypervisor service.

#### 14.3.5.1 Errors

|        |                             |
|--------|-----------------------------|
| EINVAL | No buffer currently defined |
|--------|-----------------------------|