Sun4v Hypervisor API

Revision 0.16

March 7, 2005

Sun microsystems

Table of Contents

1 Introduction

This document details the API provided to a sun4v domain by the underlying hypervisor. 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.

2 Hypervisor call conventions

Hypervisor API calls are made through the 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 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

These registers should be considered volatile across the call unless explicitly stated as part of the instruction semantics. All other registers are preserved across the function call, unless explicitly stated otherwise as part of the function semantics.

Some functions may preserve the caller's arguments as an optimization to assist multiple consecutive calls to the same function. This optimization is required only for those hypervisor functions which specify it.

2.2 Fast traps

Fast traps are the preferred mechanism for hypervisor API calls. The calling convention is as follows:

Register	Input	Output
%o5	function number	
%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 value4

These registers should be considered volatile across the call unless explicitly stated as part of the instruction semantics. All other registers are preserved across the function call, unless explicitly stated otherwise as part of the function semantics.

Further arguments may be passed in memory, as defined on a per function call basis.

Typically calling arguments are not preserved, and return values are defined on a per-function basis. Some functions will not provide any return values at all, other functions may never even return.

All fast trap API calls use *sw_trap_number* 0x80 in the Tcc instruction.

2.3 Post hypervisor trap processing

Upon return from the API call, the returned status is given in register %o0. A value of EOK in %o0 implies successful execution of the API call, with any additional result value being returned in %o1. In the event that the API call fails, the non-EOK error code is returned in %o0. In either case optionally any futher information may be being supplied in %o1-%o4 on a per service basis.

If an invalid *sw_trap_number* is issued, or if an invalid function number is specified, the hypervisor will return with EBADTRAP in %o0.

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
CPU_TICK_NPT	0x81
CPU_STICK_NPT	0x82
MMU_MAP_ADDR	0x83
MMU_UNMAP_ADDR	0x84
TTRACE_ADDENTRY	0x85
API_VERSION	0xff

3.2 Function number

The following are the function numbers for %o5 when a fast trap is invoked:

MACH_EXIT	0x00
MACH_DESC	0x01
MACH_SIR	0x02
CPU_START	0x10
CPU_STOP	0x11
CPU_YIELD	0x12
CPU_WATCHDOG	0x13
CPU_QCONF	0x14
CPU_QINFO	0x15
CPU_MYID	0x16
CPU_STATE	0x17
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

3.3 Errors ids

When a hypervisor API returns an error, %o0 will contains 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
EWOULDBLOCK	9	Cannot complete operation without blocking
ENOACCESS	10	No access to specified resource
EIO	11	I/O Error
ECPUERROR	12	CPU is in error state
ENOTSUPPORTED	13	Function not supported
ENOMAP	14	No mapping found
ETOOMANY	15	Too many items specified / limit reached

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 Domain services

These services affect the entire virtual machine domain. Initially, a domain is created in the *idle* state. When the service processor loads a guest (usually the boot firmware) onto a domain, the domain transitions to the *guest* state. The domain may be return to the *idle* state either by a service processor request to the hypervisor, or by the guest software using a hypervisor API.

5.1 API call

5.1.1 api_version

```
trap#            API_VERSION
arg0             api_group
arg1             major_number
arg2             req_minor_number

ret0             status
ret1             act_minor_number
```

The API service enables a guest to request and check for a version of the Hypervisor APIs with which it may be compatible. API services are grouped into sets that are specified by the argument *api_group*, (defined in the table below). For the specified group the guest's requested API major version number is given by the argument *major_number* and a requested API minor version number is given by the argument *req_minor_number*.

If the *major_number* is supported, the actual minor version implemented by the Hypervisor is returned in ret1 (*act_minor_number*). Note that the actual minor version number may be less than, equal to, or greater than the requested minor version number. (See Notes, below).

If the *major_number* is not supported, the Hypervisor returns an error code in ret0, and ret1 is undefined. (See Errors, below.)

The API groups are defined beloow together with their version numbers compliant with this specification.

Group	**Number (api_group)**	**Group Definition**	**Version for this specification**
Common	0x0	sun4v version	1-0
	0x1	core API version	1-0
Technology	0x100	PCI	1-0
	0x101	Virtual I/O (Inter-domain communication)	1-0

API version numbers.

The API version number for the sun4v API group (0x0) shall be 1.0 (major = 1, minor = 0) for implementations compliant with this specification.

The API version number for the ***core API*** group (0x1) shall be 1.0 (major=1, minor=0) for implementations compliant with this specification.

Implementation Notes:

This API uses its own trap number to ensure consistency between future versions of the

API.

A guest may request minor version X, and this API may return minor version Y for a given *major_number* and *api_group*.

If X = Y, then the requested minor version is available.

If the implementation returns minor version Y where Y < X.

The guest must be able to determine if the minor interface version Y offers the required services it needs and proceed accordingly. (This is a guest policy issue.)

If Y > X, then the guest must determine if it can operate with version Y. In most cases, minor version number incremements are assumed to be compatible so in general, the guest may accept Y when Y > X. In this case, the guest may want to print a warning, but that is up to the policy of the guest.

This API uses its own trap number, not for performance reasons, but to ensure its constancy even in the face of new API major versions.

API calls and status information returned from the hypervisor are versioned in group as follows:

The guest inquires of the hypervisor (via this API service) as to whether or not a particular version (major and minor) of a particular interface group is available.

The Hypervisor will return EINVAL or ENOTSUPPORTED if a particular grouping is not recognized, or version of the interface is not available, otherwise the hypervisor will return EOK, and the minor number of the interface it is willing to offer the guest.

In the event of EINVAL or ENOTSUPPORTED, a guest may attempt either a different version, or find some way to continue without the required service interface. Alternatively the guest may simply elect to request termination (exit) of the virtual machine environment.

In the event that EOK is returned, the hypervisor will return the minor number of the available interface that it is will to present to the guest.

It is assumed that increments of the interface minor number are backwards compatible.

A guest may request minor version X, and to a hypervisor capable of minor version Y.

Upon use of this API, the hypervisor will always return minor version Y if X>=Y. The guest is then assumed to be able to determine if interface version Y offers the required services.

Alternatively in the event that X<Y, the hypervisor may elect to emulate version X, thus returning X, or simply return Y. Should the hypervisor return a minor number greater than the version the guest has requested, the guest should have a policy action, such as print a warning.

5.1.1.1 Errors

```
EINVAL              If api_group field is invalid or unsupported
ENOTSUPPORTED       If major number for that api_group is not
supported
EOK                 If api_group and major number match
```

5.1.2 mach_exit

```
trap#               FAST_TRAP
```

```
function#          MACH_EXIT
arg0               exit_code
```

This service stops all CPUs in the virtual machine domain and places the domain into the *idle* state. The *exit_code* is passed to the service processor as the domain's exit status. On systems without a service processor, the domain will undergo a reset, and the boot firmware will be reloaded. The *exit_code* will be passed on in the new machine description. This function will never return to the guest code that invokes it.

5.1.2.1 Errors

This service does not return.

5.1.3 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 8 byte aligned, and a minimum of 64bytes in size. Upon return the actual size of the machine description copied into the given buffer is provided in the ret1 (length) return value.

5.1.3.1 Errors

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

5.1.4 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.

5.1.4.1 Errors

This service does not return.

6 CPU services

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

6.1 CPU list

Operations that are to be be performed on multiple CPUs 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.

To allow for certain optimizations, a special case exists if the CPU list pointer is NULL. In this case the requested API call is performed on the local processor only.

6.2 API calls

6.2.1 cpu_start

```
trap#              FAST_TRAP
function#          CPU_START
arg0               cpu
arg1               pc
arg2               tba
arg3               target_arg0

ret0               status
```

Start CPU *cpu* with *pc* in %pc and *tba* in %tba. 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.

6.2.1.1 Errors

ENOCPU	Invalid *cpu*
EINVAL	Target *cpu* is not in the stopped state
ENORADDR	Invalid *pc* or *tba* real address

6.2.2 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.

6.2.2.1 Errors

ENOCPU	Invalid *cpu*
EINVAL	target *cpu* is the current cpu

EINVAL	target *cpu* is not in the *running* state

6.2.3 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.

6.2.3.1 Errors

No possible error

6.2.4 cpu_watchdog

trap#	FAST_TRAP
function#	CPU_WATCHDOG
arg0	seconds
ret0	status
ret1	seconds remaining from previous interval

This call is used to update a CPU watchdog timeout. arg0 contains the new duration to wait until the timeout occurs. The timeout period is specified in seconds. The hypervisor may fail the service if the timeout interval given is too short. A timeout interval of 0 disables the watchdog service.

Upon returning a status of EOK, the service returns the previously specified timeout period in ret1.

If the supervisor fails to call this function, the hypervisor will inform the service processor that the virtual machine is not responding. On systems without a service processor, the hypervisor may issue a hardware reset and reload the boot firmware.

6.2.4.1 Errors

EOK	successful
EINVAL	timeout period is too short or too long for this platform

6.2.5 cpu_tick_npt

trap#	CPU_TICK_NPT
arg0	npt
ret0	status

This function sets the NPT bit in the %tick register of the current CPU to *npt*. The counter field is not changed.

Programming note: Certain platforms (for example Niagara-1.0) may have issues maintaining clock synchronization as a result of this service, in which case the API service may return ENOTSUPPORTED.

6.2.5.1 Errors

ENOTSUPPORTED

6.2.6 cpu_stick_npt

```
trap#              CPU_STICK_NPT
arg0               npt

ret0               status
```

This function sets the NPT bit in the %stick register of the current CPU to *npt*. The counter field is not changed.

Programming note: Certain platforms (for example Niagara-1.0) may have issues maintaining clock synchronization as a result of this service, in which case the API service may return ENOTSUPPORTED.

6.2.6.1 Errors

ENOTSUPPORTED

6.2.7 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. B*ase* 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 unconfigured 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

6.2.7.1 Errors

ENORADDR	Invalid *base*
EINVAL	Invalid *queue* or, *nentries* not a power of two in number
EBADALIGN	*baseaddr* is not correctly aligned for size

6.2.8 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.

6.2.8.1 Errors

EINVAL	Invalid *queue*

6.2.9 cpu_mondo_send

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

Send a mondo interrupt to CPU list *cpus* 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.

6.2.9.1 Errors

ENORADDR	Invalid *data*
ENOCPU	Invalid CPU in *cpus*

6.2.10 cpu_myid

trap#	FAST_TRAP
function#	CPU_MYID
ret0	status
ret1	state

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

6.2.11 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

6.2.11.1 Errors

ENOCPU	Invalid CPU in cpuid

7 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.

7.1 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 provided for version 1.0 of this specification.

7.1.1 Page sizes

The page size bitmask is indicates with each bit (when set) that the corresponding page size may be legally present in the TSB. 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
etc. ...	

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 TTE:

Value	Page size assumed for index computation
0	8K
1	64K
2	512K

```
3                   4MB
4                   32MB
5                   256MB
etc. ...
```

The index shift value must correspond to the smallest page size specified in the page size bit mask.

7.1.2 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.

7.2 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	Size	Field
0x00	8	Fault type
0x08	8	Fault Address
0x10	8	Fault Context
0x40	8	Data fault type
0x48	8	Data fault status
0x50	8	Data fault context

Each of the fault type fields may be interpreted as follows:

Code	Fault type	Trap type	Instn/Data	Fault Addres Updated	Fault Context Updated	Comments
1	fast miss	fast	I+D	x	x	
2	fast protection	fast	D	x	x	
3	MMU miss	slow	I+D	x	x	
4	invalid RA	slow	I+D	x	x	
5	privileged violation	slow	I+D	x	x	
6	protection violation	slow	I+D	x	x	ifetch from non-executable , or store to non-writeablei

Code	Fault type	Trap type	Instn/Data	Fault Addres Updated	Fault Context Updated	Comments
7	NFO access	slow	I+D	x	x	ifetch from NFO, or non-NF load from NFO
8	so page	slow	D	x	x	(NF load from side-effect page)
9	invalid VA	slow	I+D	x	x	
10	invalid ASI	slow	D	-	-	
11	nc atomic	slow	D	x	x	
12	privileged action	slow	D	x	x	
13	reserved	none	-	-	-	
14	unaligned access	slow	D	x	x	
15	invalid page size	slow	D	x	x	

7.3 API calls

7.3.1 mmu_tsb_ctx0

```
trap#               FAST_TRAP
function#           MMU_TSB_CTX0
arg0                ntsb
arg1                tsbs

ret0                status
```

Configures the TSBs for the current CPU for virtual addresses with context zero. *tsbs* is an array of *ntsbs* TSB descriptions. Each description is for either a direct mapped per-pagesize TSB, or a fully-assiciative TSB that can contains TTE entries of any pagesize. Each TTE must be encoded for context 0.

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

7.3.1.1 Errors

```
ENORADDR            Invalid TSB base
EBADPGSZ            Invalid pagesize
EBADTSB             Invalid associativity or size
EINVAL              Invalid ntsbs
```

7.3.2 mmu_tsb_ctxnon0

```
trap#               FAST_TRAP
function#           MMU_TSB_CTXNON0
arg0                ntsb
arg1                tsbs

ret0                status
```

Configures the TSBs for the current CPU for virtual addresses with non-zero contexts. *tsbs* is an array of *ntsbs* TSB descriptions. Each description is for either a direct mapped per-pagesize TSB, or a fully-assiciative TSB that can contains TTE entries of any pagesize.

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

7.3.2.1 Errors

```
ENORADDR            Invalid TSB base
```

EBADPGSZ	Invalid pagesize
EBADTSB	Invalid associativity or size
EINVAL	Invalid *ntsbs*

7.3.3 mmu_demap_page

trap#	FAST_TRAP
function#	MMU_DEMAP_PAGE
arg0-1	cpus
arg2	vaddr
arg3	ctx
arg4	flags
ret0	status

Demaps any page mapping of virtual address *vaddr* in context *ctx* from TLBs associated with CPU list *cpus*. Any virtual tagged caches are guaranteed to be kept consistent. The flags argument applies the d-map operation to I-TLB entries if bit 1 is set, and in addition applies the demap operation to D-TLB entries if bit 0 is set. At least one of bit 0 and / or bit 1 must be set. For hardware implementations with single unified I and D TLBs, 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.

7.3.3.1 Errors

ENOCPU	Invalid CPU in *cpus*
EINVAL	Invalid vaddr, context or flag value

7.3.4 mmu_demap_ctx

trap#	FAST_TRAP
function#	MMU_DEMAP_CTX
arg0-1	cpus
arg2	ctx
arg3	flags
ret0	status

Demaps context *ctx* from TLBs and any virtually tagged caches on CPU list *cpus*. The flags argument applies the d-map operation to I-TLB entries if bit 1 is set, and in addition applies the demap operation to D-TLB entries if bit 0 is set. At least one of bit 0 and / or bit 1 must be set. For hardware implementations with single unified I and D TLBs, 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.

7.3.4.1 Errors

ENOCPU	Invalid CPU in *cpus*
EINVAL	Invalid context or flag value

7.3.5 mmu_demap_all

trap#	FAST_TRAP
function#	MMU_DEMAP_ALL
arg0-1	cpus
arg2	flags
ret0	status

Demaps all translations from TLBs and virtually tagged caches on CPU list *cpus*. The flags argument applies the d-map operation to I-TLB entries if bit 1 is set, and in addition applies the demap operation to D-TLB entries if bit 0 is set. At least one of bit 0 and/or bit 1 must be set. For hardware implementations with single unified I and D TLBs, 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.

7.3.5.1 Errors

ENOCPU	Invalid CPU in *cpus*
EINVAL	Invalid flag value

7.3.6 mmu_map_addr

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

This API call is intended both for supervisors that do not use TSBs, and for supervisors to specify temporary translation mappings. TTE provides a translation for virtual address *vaddr* in context *ctx* for the calling virtual CPU. The TTE is directed to an appropriate TLB as indicated by the flags:

If bit 0 of flags is set then the TTE is loaded into an appropriate D-TLB for translating vaddr in context ctx. Similarly, if bit 1 of flags is set, the TTE is loaded into an appropriate I-TLB. If both bits are set then the TTE is loaded into both I- and D-TLBs.

The hypervisor may perform an implied flush before installing the TTE on CPUs where installing multiple TTEs with the same virtual tag could damage the CPU.

7.3.6.1 Errors

EINVAL	Invalid value

7.3.7 mmu_map_perm_addr

trap#	FAST_TRAP
function#	MMU_MAP_PERM_ADDR
arg0	vaddr
arg1	ctx
arg2	TTE
arg3	flags
ret0	status

This API call used used to specifiy 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.

TTE to provides a translatation for virtual address *vaddr* in context *ctx* for the calling virtual CPU. There is an implied unmap of all conflicting previous mappings before installing the new TTE.

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 are interpreted as follows:

If bit 0 of flags is set then the TTE is used with an appropriate D-TLB for translating vaddr in context ctx. Similarly, if bit 1 of flags is set, the TTE is used with an appropriate I-TLB. Both bits 0 and 1 may be simultaneously set.

7.3.7.1 Errors

EBADPGSI	Invalid page size value
ENORADDR	Invalid real address in TTE
ETOOMANY	Too many mappings (maximum of 8 reached)

7.3.8 mmu_unmap_addr

```
trap#            MMU_UNMAP_ADDR
arg0             vaddr
arg1             ctx
arg2             flags

ret0             status
```

Demaps virtual address *vaddr* in context *ctx* on this CPU. This function is intended to be used to demap pages mapped with **mmu_map_addr** above. This function is equivalent to invoking **mmu_demap_page** with only the current CPU in the CPU list. The flags argument applies the d-map operation to I-TLB entries if bit 1 is set, and in addition applies the demap operation to D-TLB entries if bit 0 is set. At least one of bit 0 and/or bit 1 must be set. For hardware implementations with single unified I and D TLBs, 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.

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

7.3.8.1 Errors

EINVAL	Invalid value

7.3.9 mmu_unmap_perm_addr

```
trap#            FAST_TRAP
function#        MMU_UNMAP_PERM_ADDR
arg0             vaddr
arg1             ctx
arg2             flags

ret0             status
```

Demaps any permanent page mapping (established via mmu_map_perm_addr) of virtual address *vaddr* in context *ctx* from TLBs associated with CPU list *cpus*. Any virtual tagged caches are guaranteed to be kept consistent. The flags argument applies the demap operation to I-TLB entries if bit 1 is set, and in addition applies the demap operation to D-TLB entries if bit 0 is set. At least one of bit 0 and/or bit 1 must be set. For hardware implementations with single unified I and D TLBs, 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.

7.3.9.1 Errors

ENOCPU	Invalid CPU in *cpus*
EINVAL	Invalid vaddr, context or flag value

7.3.10 mmu_fault_area

```
trap#               FAST_TRAP
function#           MMU_FAULT_AREA
arg0                raddr

ret0                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.

7.3.10.1 Errors

```
ENORADDR            Invalid real address
EBADALIGN           Invalid alignment for fault area
```

7.3.11 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. To avoid complicated address mapping issues, the caller is required to provide a *return_target* address that is a real address if translation is to be disabled, or a virtual address if translation is being enabled. Upon successful completion, the hypervisor will return control to the *return_target* address provided.

7.3.12 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.

7.3.13 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.

7.3.13.1 Errors

```
EINVAL                    Invalid buffer size
```

8 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.

8.1 API calls

8.1.1 mem_scrub

```
trap#                FAST_TRAP
function#            MEM_SCRUB
arg0                 raddr
arg1                 length

ret0                 status
ret1                 length scrubbed
```

Associates a valid ecc code with the memory at *raddr*. Uncorrectable errors are not automatically scrubbed by the hypervisor. Supervisors must use this function to avoid multiple errors from the same line of memory. This API service may also be used to bulk-scrub memory.

The start address and length should be aligned with the main memory detection / correction (ECC) coverage boundary.

The hypervisor may elect to only partially scrub the requested block of memory, in this event it will return EOK in ret0, and the length scrubbed in ret1.

8.1.1.1 Errors

EOK	Success or partial success
ENORADDR	Invalid *raddr*
EBADALIGN	Either the start address or length are not correctly aligned.

8.1.2 mem_sync

```
trap#                FAST_TRAP
function#            MEM_SYNC
arg0                 raddr
arg1                 length

ret0                 status
ret1                 length synched
```

Forces any cached copies of data to be in sync with memory starting at *raddr* for *length*. This function is intended to ensure that memory contents and cached copies of data are in sync. This function may cause cache write-backs, updates or invalidations where necessary, but does not imply a cache flush.

The hypervisor may elect to only partially sync the requested block of memory, in this event it will return EOK in ret0, and the length synced in ret1.

8.1.2.1 Errors

EOK	Success or partial success
ENORADDR	Invalid *raddr*
EBADALIGN	Either the start address or length are not correctly aligned.

9 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.

9.1 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

9.2 API calls

9.2.1 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*).

9.2.1.1 Errors

EINVAL	Invalid devhandle/devino

9.2.2 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).

9.2.2.1 Errors

EINVAL	Invalid sysino

9.2.3 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.*

9.2.3.1 Errors

EINVAL	Invalid sysino or intr_enabled value

9.2.4 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.

9.2.4.1 Errors

EINVAL	Invalid sysino

9.2.5 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*.

9.2.5.1 *Errors*

```
EINVAL              Invalid sysino
```

9.2.6 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*.

9.2.6.1 *Errors*

```
EINVAL              Invalid sysino
```

9.2.7 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*.

9.2.7.1 *Errors*

```
EINVAL              Invalid sysino
ENOCPU              Invalid cpuid
```

10 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.

10.1 API calls

10.1.1 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.

10.1.1.1 Errors

EWOULDBLOCK

10.1.2 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.

10.1.2.1 Errors

EWOULDBLOCK

11 Console services

The hypervisor will provide a virtual console device for systems with service processors. On systems without a service processor, the console may be a native device.

11.1 API calls

11.1.1 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

11.1.1.1 Errors

```
EWOULDBLOCK         No character available
```

11.1.2 cons_putchar

```
trap#               FAST_TRAP
function#           CONS_PUTCHAR
arg0                char

ret0                status
```

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

11.1.2.1 Errors

```
EINVAL              Illegal character
EWOULDBLOCK         Output buffer currently full, would block
```

12 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.

12.1 API calls

12.1.1 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 miniumum size required by the hypervisor for a valid dump buffer.

12.1.1.1 Errors

```
EOK                 Dump buffer was configured, ret1 contains
required size
ENORADDR            Invalid raddr
EBADALIGN           raddr not aligned on 64byte boundary
```

12.1.2 dump_buf_info

```
trap#               FAST_TRAP
function#           DUMP_BUFINFO

ret0                status
ret1                real address of current dump buffer
ret2                size of current dump buffer
```

12.1.2.1 Errors

```
EINVAL              No dump buffer is currently configured
```

13 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.

13.1 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 fail offset is the offset of the next entry to be written.

13.2 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

13.3 API calls

13.3.1 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.

13.3.1.1 Errors

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

13.3.2 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.

13.3.2.1 Errors

EINVAL	No buffer currently definedd

13.3.3 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.

13.3.3.1 Errors

EINVAL	No buffer currently defined

13.3.4 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.

13.3.4.1 Errors

EINVAL	No buffer currently defined

13.3.5 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 and 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.

13.3.5.1 Errors

EINVAL	No buffer currently defined