Minutes for CAM working group
Accredited Standards Committee
X3, Information Processing Systems

Doc. No.: X3T10/95-174
Date: March 21, 1995
Project: CAM-3/990D
Ref. Doc.: X3T10/792D-R12

To: Membership of X3T10

From: William Dallas

Subject: Minutes of March 1995 CAM-3 Working Group Meeting

1. Opening Remarks
2. Attendance and Membership
3. Approval of Agenda
4. Discussion of CAM-3 data types, alignment, and mandatory services.
5. Discussion of the comments from Marc Gauthier.
6. Actions of the working group
7. Meeting Schedule
8. Adjournment

Results of Meeting

1. Opening Remarks

William Dallas the Editor of the CAM document, called the meeting to order at 9:20 am, Tuesday, January 10 1994. He thanked Skip Jones of QLOGIC for hosting the meeting.

As is customary, the people attending introduced themselves. A copy of the attendance list was circulated for attendance.

It was stated that the meeting had been authorized by X3T10 and would be conducted under the X3 rules.

The minutes of this meeting will be posted to the SCSI BBS and the SCSI Reflector and will be included in the next committee mailing.

2. Attendance and Membership

Attendance at working group meetings does not count toward minimum attendance requirements for X3T10 membership. Working group meetings are open to any person or company to attend and to express their opinion on the subjects being discussed.

The following people attended the meeting:

Name	Org	Phone Number -or- Electronic Mail Address
Lansing Sloan	LLNL	ljsloan@llnl.gov
Ed Chim	ADAPTEC	Echim@corp.adaptec.com
Dal Allan	ENDL	dal_allan@mcimail.com
Bill Dallas	Digital Equipment Corp.	dallas@zk3.dec.com

4 people present

3. Approval of Agenda

The meeting agenda was approved.

4. Discussion of CAM-2 data types, alignment, and mandatory services.

Bill Dallas presented what he is proposing for the CAM-3 data types and CAM-3 structure size definitions. The group spent a fair amount of time discussing pointer size definitions. Bill was instructed to make changes to the proposal as specified by the working group. Attached is the result of the working group changes.

5. Discussion of the comments from Marc Gauthier.

The working group discussed the comments from Marc Gauthier on Bill Dallas's general proposal of January 1995. The responses from the working group to Marc's comments are attached. The notation is as follows:
All lines proceeded by >> are Bill's proposal
All lines proceeded by > are Marc's comments
All other lines are the working groups response

6. Actions of the Working Group

See Attachments

7. Meeting Schedule

The next meeting of the CAM Working Group is planned for Tuesday May 9, 1995 in Harrisburg, Pa. The meeting is expected to run from 9:00am - 12:00am. The topics shall be CAM-1 changes and CAM-2 features/functionality.

8. Adjournment

The meeting was adjourned at 12:30 pm. on Tuesday, March 7, 1994.

Attachments:

CAM-3 Data Type and Structure Size Definitions

To allow easier transportability (not binary comparability) of CAM-3 peripheral drivers, and SIM/HBA's between different machine platforms and operating systems, a cam types definition file (cam_types.h) shall be provided by the supplier of the XPT. The cam_types.h header file shall define all CAM-3 defined data type sizes as specified by this International standard.

The supplier of the XPT shall also the supply a CCB structure definitions file (cam.h) which shall use the defined data types and CAM_boundary rules as specified by CAM-3. The term "CAM_boundary" is a defined term and refers to the address alignment of a data type within a structure and the structure itself. (refer to the Definitions Clause X.X and Structure Member CAM_boundary rules Clause X.X for further details). The CCBs structure names, member names and sizes specified by CAM-3 shall be reflected in the cam.h file.

CAM_boundaries (address boundaries) of the CAM-3 defined structures and its members is specified by CAM-3. It shall be the responsibility of the XPT supplier to preserve those CAM_boundaries as specified by the CAM_boundary rules. The CAM_boundary rules allows CCB structures and members to be aligned to a known offset for a 16, 32, 64, etc bit processors regardless of the platform or O.S. defined pointer size.

Data Type Sizes

CAM-3 defines the storage sizes and whether the storage class is signed or unsigned for the structures it defines. The supplier of the XPT shall define the storage class in the cam_types.h as follows:

CAM_U8	-	An unsigned a 8 bit quantity
CAM_S8	-	An signed a 8 bit quantity
CAM_U16	-	An unsigned a 16 bit quantity.
CAM_S16	-	An signed a 16 bit quantity.
CAM_U32	-	An unsigned a 32 bit quantity.
CAM_S32	-	An signed a 32 bit quantity.

Pointer sizes within CAM-3 structures shall be a power of 32 bits and shall be the size that the O.S. defines for its pointer size rounded up to a power of 32 bits. The following is an example of pointer size rules:

OS Pointer Size	CAM-3 Structure Storage Size
1 to 32 bits	32 bits
33 to 64 bits	64 bits
65 to 128 bits	128 bits

Structure Member CAM_boundary rules

The defined CAM-3 data types and pointers declared within a defined CAM-3 structure shall be naturally aligned to their addressed boundary.

The XPT supplier shall ensure the CAM_boundary by padding the structure so that the member aligns with its address boundary. If the next defined member data type does not naturally align then the XPT supplier shall ensure this CAM_boundary by padding the structure so that the member naturally aligns with it address boundary.

All structures and structures within structures shall be aligned to the naturally aligned pointer boundary.

All arrays within a structure shall be aligned to the naturally aligned pointer boundary.

The XPT (transport functionality)

This clause describes the set of services (functions) the XPT shall provide and may optionally provide to the peripheral drivers and the SIM/HBAs. The supplier of the XPT (usually the O.S. provider) shall provide the XPT services listed as mandatory and may optionally provide other XPT services listed as optional. The XPT shall indicate in the PATH INQUIRY CCB the optional services it provides (see PATH INQUIRY CCB clause for further details).

The implementation of an XPT service is in a vendor unique manner but shall conform the following:

- syntax of the XPT service call
- return value of the XPT service call
- behavior specified for the XPT service

The declarations of the XPT services and flags shall be in the xpt_services.h file. The xpt_services.h file shall have a #define SERVICE_NAME() definition for all services both mandatory and optional. The supplier of the XPT may implement any service as function call or as a macro.

> Note: The #define SERVICE_NAME() is to prevent compiling problems for the optional services that the XPT does not provide. The XPT supplier should stub those services it does not provide.

The intention of the mandatory XPT services is to provide a common set of functionality for both the peripheral drivers and the SIMs/HBAs that allows for easier transportability across operating systems and platforms.

XPT CAM-3 Services (mandatory)

The CAM-3 XPT shall provide the mandatory services detailed in this International standard and may provide the optional services.

CAM_U8 XPT_ISR( )

The XPT_ISR service shall provide a means for components of the CAM-3 subsystem to determine if the CAM-3 component is in interrupt service context. If the O.S. does not provide the functionality to determine interrupt context then the XPT_ISR service shall always return the value that indicates the caller is in interrupt service context.

Returns:

0X00:

Indicates that the caller is not in interrupt context.

0X01:

Indicates that the caller is in interrupt context.

CAM_U8 XPT_MEM_ALLOC(addr *, cast, CAM_U32 size, CAM_U32 flags)

The XPT_MEM_ALLOC service shall provide a means for the peripheral drivers, SIMs and the XPT to allocate memory for buffers. The XPT_MEM_ALLOC service shall allocate at least the specified size in CAM_U8s (e.g., bytes) of memory, if memory can be allocated at this time. The caller of the service shall not call the service with the XPT_WAITOK flag set to a one when in interrupt service context. Interrupt context shall be verified and the XPT_WAITOK flag set according before requesting the XPT_MEM_ALLOC service.

The XPT shall indicate in the PATH INQUIRY CCB whether memory pointers returned (e.g., addr *) are physical or a virtual addresses (see PATH INQUIRY CCB clause for further details).

addr *:

Specifies the memory pointer that may contain the pointer to the allocated memory (e.g. pointer to contain the starting address to the allocated memory). The service shall set the storage specified (addr *) to a non NULL value if memory was allocated by this service or NULL if memory could not be obtained at this time.

cast:

Specifies that data type of the addr argument and the type of memory pointer (e.g., addr *) placed in the storage (e.g., addr).

CAM_U32 size:

Specifies the size of memory in CAM_U8s (e.g., bytes) to allocate.

CAM_U32 flags:

Bit 0: - XPT_WAITOK

The flag set to a one specifies that if the requested size can not be allocated at this time the service may block (e.g., suspend the execution of the caller) waiting for memory resources, if the service can not provide the memory resource.

The flag set to a zero (0) specifies that if the requested size can not be allocated at this time the service shall not block.

Note: The service does not have to provide synchronization of requests for callers that are blocked (e.g., when resources become available the first caller that was blocked does not necessarily get the resource).

Bit 1: - XPT_BZERO
The flag set to a one specifies that if the requested memory has been allocated for the caller, before return to the caller, the service shall set each CAM_U8 (e.g., byte) of the allocated memory to a zero (0).

Bit 2: - XPT_PHY_CONTIG
The flag set to a one specifies that the memory allocated shall be physically contiguous. If the service can not at any time provide physically contiguous memory the storage (e.g., addr) shall be set to NULL and the returned value shall be CAM_PROVIDE_FAIL (0X16). The XPT shall indicated in the PATH INQUIRY CCB whether the service supports this feature.

Returns:

CAM_PROVIDE_FAIL (0X16):
If the service can not at any time provide the requested functionality.

CAM_BUSY (0X05):
The service can not satisfy this request at this time.

CAM_REQ_CMPLT (0X01):
The service completed the request as specified and the addr * argument contains the pointer to the allocated memory.

Example:

```
struct xyz *xyz_ptr;
CAM_U8 status;

status = XPT_MEM_ALLOC( &xyz_ptr,  struct xyz *,  sizeof(struct xyz), (XPT_WAITOK |
XPT_BZERO));
```

void XPT_MEM_FREE(addr *)

The XPT_MEM_FREE service shall provide a means for the peripheral drivers, SIMs and the XPT to free (e.g., return to the O.S.) memory buffers allocated by the XPT_MEM_ALLOC service.

addr *
Shall specify the memory pointer that points to the allocated memory to be freed (e.g. returned to the O.S.). The argument addr * shall contain a pointer to memory that was previously allocated in a call to the XPT_MEM_ALLOC service. There shall be only one call to the XPT_MEM_FREE service for each memory buffer allocated by the XPT_MEM_ALLOC service.

Example:

```
struct xyz *xyz_ptr;
CAM_U8 status;

status = XPT_MEM_ALLOC( &xyz_ptr,  struct xyz *,  sizeof(struct xyz), (XPT_WAITOK |
XPT_BZERO));

if (status == CAM_REQ_CMPLT){
        XPT_MEM_FREE( xyz_ptr);
}
```

CCB * xpt_ccb_alloc( )

The xpt_ccb_alloc service shall provide the means for the peripheral drivers, SIMs and the XPT to allocate CCBs for use. It shall be the responsibility of the XPT to ensure that the pointer of the allocated CCB returned from the xpt_ccb_alloc service call shall point to a memory buffer large enough to contain any of the possible XPT/SIM function request CCBs. The xpt_ccb_alloc service shall return a NULL pointer if memory resources are not immediately available.

The returned CCB shall be properly initialized for use as a SCSI I/O request CCB. The SIM private data area shall have been already set up to be used by the XPT and SIM, and shall not be modified by the peripheral driver. The allocated CCB can only be used (i.e., sent to the XPT), once. Once the CCB has completed it shall be returned using the xpt_ccb_free service.

Returns:

Non NULL *:

Indicates that the service has allocated a CCB for the caller.

NULL *:

Indicates that the service could not allocate a CCB at this time.

void xpt_ccb_free(CCB *)

The xpt_ccb_free service shall provide the means for the peripheral drivers, SIMs and the XPT to free CCBs after use. The xpt_ccb_free service takes a pointer to the CCB that the caller has finished with so it can be returned to the CAM subsystem.

CCB *:

Pointer to the CCB to be freed to the CAM subsystem.

Returns:

None:

CAM_U32 xpt_action(CCB *)

All CAM CCB requests to the XPT or a SIM/HBA are placed through this service call. The CAM Status information for callback on completion CCBs shall be obtained at the callback point via the CAM status fields. The CAM Status information for non callback on completion - non immediate CCBs shall be obtained by polling the CAM status field for a non Request in Progress status. The CAM Status information for immediate CCBs shall be obtain on return from the service call by examining the CAM Status field.

CCB *:

Pointer to a CCB that shall be transported/passed to a SIM/HBA or the XPT.

Returns:

For Immediate CCBs:

Any Valid CAM Status

For Queued CCBs

Request In Progress - Indicates that the CCB has been accepted.

Any other valid CAM Status - Indicates that the CCB has not been accepted

The ultimate of any CCB shall be obtained from the CAM status field of the CCB

pm_offset_t phy_addr * XPT_VIRT_TO_PHYS(vm_offset_t addr *, map *)

The XPT_VIRT_TO_PHYS service shall convert the passed virtual addr * argument to its associated physical address and return that address. The map * argument is O.S. dependent (see O.S. dependent clauses for further details). The caller shall pass a null map * value if the map * is not available to the caller (e.g., usually a kernel virtual I/O transfer does have an associated map). The XPT_VIRT_TO_PHYS service shall ensure that if a map * is passed, that map * is associated with this virtual address.

vm_offset_t addr *:
: Pointer of the virtual address that shall be converted to a physical address.

map *:
: Pointer to the O.S. dependent map structure if available. Refer to the O.S. specific clauses for more information.

Returns:

All values:
: The value is the associated physical address.

Example:

```
routine ( CCB_SCSIIO *ccb)
{
pm_offset_t *p_addr;
p_addr = XPT_VIRT_TO_PHYS( (vm_offset_t)ccb->cam_data_ptr, ccb->cam_req_map);
}
```

CAM_U32 XPT_PAGE_SIZE(vm_offset_t addr *, map *)

The XPT_PAGE_SIZE service shall return the virtual page size that is associated with the virtual address (addr *). The map * argument is O.S. dependent (see O.S. dependent clauses for further details). The caller shall pass a null map * value if the map * is not available to the caller (e.g., usually a kernel virtual I/O transfer does have an associated map). The XPT_PAGE_SIZE service shall ensure that if a map * is passed, that map * is associated with this virtual address.

vm_offset_t addr *:
: Pointer of the virtual address that shall be converted to a physical address.

map *:
: Pointer to the O.S. dependent map structure if available. Refer to the O.S. specific clauses for more information.

Returns:

All values:
: The value is the page size for the associated virtual address.

CAM_U32 XPT_PDRV_REG(CAM_U8 addr *)

The XPT_PDRV_REG service shall provide the means for peripheral drivers to register with the XPT. The XPT_PDRV_REG service shall return a unique number (the driver registration number) for a peripheral driver that calls this service. The argument (addr *) shall point to a NULL terminated string that represents the peripheral drivers name (e.g., “cam_xyz_disk_driver”).

The service shall ensure that the drivers name has not been already assigned a peripheral driver registration number. This may be accomplished by comparing the peripheral drivers name to stored peripheral driver names which have assigned driver registration number. If a match is found the returned value of this service is zero (0).

A peripheral driver should ensure that it registers with the XPT only once at peripheral driver initialization. The assigned peripheral driver registration number shall be used by the peripheral driver when requesting an advisory lock on a SCSI Logical unit. The peripheral driver shall call the XPT_PDRV_UNREG service when it is unloaded.

CAM_U8 addr *:

Shall point to a NULL terminated string that represents the peripheral drivers name

Returns:

A positive value:

The value is the peripheral driver registration number for this peripheral driver.

A zero (0)

The service already has a existing registration that matches this peripheral drivers name.

void XPT_PDRV_UNREG(CAM_U32 pdrv_reg_num)

The XPT_PDRV_UNREG service shall provide the means for peripheral drivers to de-register with the XPT. The XPT_PDRV_UNREG service shall break all associations made when this peripheral driver registration number was assigned. This peripheral driver registration number shall be available for re-assignment upon return of this service.

CAM_U32 pdrv_reg_num:

The peripheral driver registration number to disassociate with a peripheral driver (e.g., number available for re-assignment).

Returns:

None

CAM_U8 XPT_UNIT_LOCK(CCB_HEADER *, CAM_U32 pdrv_reg_num, CAM_U32 flags)

The XPT_UNIT_LOCK service shall provide an advisory lock service for the peripheral drivers to lock a identified Logical unit. The argument CCB_HEADER * passed by the caller shall be a pointer to a CCB_HEADER and the CCB_HEADER shall have a valid Connect Id.

CCB_HEADER *

Pointer to a valid CCB_HEADER

CAM_U32 pdrv_reg_num

The callers peripheral drivers registration number obtained for the XPT_PDRV_REG service.

CAM_U32 flags

Bit 0:

Logical unit lock requested is a shared lock. Multiple peripheral drivers may operate with this Logical unit.

Bit 1:

Logical unit lock requested is an exclusive lock. A single peripheral driver shall only be granted the advisory lock until released.

The service shall behave as follows:

Maintain a count on a per Logical unit basis for shared locks active for a Logical unit.

Maintain a indication that an exclusive lock has been granted for the identified Logical unit.

Ensure that when an exclusive lock is requested that there a no shared or exclusive locks currently against the identified Logical unit.

Ensure that when an shared lock is requested that there a no exclusive locks currently against the identified Logical unit.

Ensure that for a lock request that both lock types are not requested.

Ensure that the identified Logical unit is represented in the edt_table[] as a valid Logical unit.

The service may store the peripheral driver registration for a granted lock against a Logical unit.

The peripheral drivers shall ensure that they only call the service once for the granted lock (e.g., when they are ready to operate with the identified Logical unit) and shall release the lock (e.g., XPT_UNIT_UNLOCK) before requesting another lock against the identified Logical unit. The peripheral driver shall not operate with any Logical unit which it has not be granted an lock.

Returns:

- Zero (0x0):
  - The requested advisory lock has been granted.
- A positive value (lock denied)
  - Bit 0 = 1:
    - A exclusive lock has been requested and an exclusive lock already exists.
  - Bit 1 = 1;
    - A exclusive lock has been requested and shared lock(s) already exists.
  - Bit 2 = 1;
    - A shared lock has been requested and an exclusive lock already exists.
  - Bit 3 = 1;
    - The identified Logical unit does not exist in the edt_table[]. A Scan Logical Unit function is suggested.

CAM_U8 XPT_UNIT_UNLOCK(CCB_HEADER *, CAM_U32 pdrv_reg_num, CAM_U32 flags)

The XPT_UNIT_UNLOCK service shall provide an advisory unlock service for the peripheral drivers to unlock a Logical unit. The argument CCB_HEADER * passed by the caller shall be a pointer to a CCB_HEADER and the CCB_HEADER shall have a valid Connect Id.

- CCB_HEADER *
  - Pointer to a valid CCB_HEADER
- CAM_U32 pdrv_reg_num
  - The callers peripheral drivers registration number obtained for the XPT_PDRV_REG service.
- CAM_U32 flags
  - Bit 0:
    - Logical unit lock release requested is a shared lock.
  - Bit 1:
    - Logical unit lock release requested is an exclusive lock.

The service shall behave as follows:

Ensure that the type of lock being requested to be released is currently active against the Logical unit.

Decrement the count on a per Logical unit basis for shared locks active for a Logical unit when releasing a shared lock.

Release the indication that an exclusive has been granted for the identified Logical unit when releasing an exclusive lock.

The peripheral drivers shall ensure that they only call the service once for the granted lock (e.g., when they are ready to no longer operate with the identified Logical unit).

Returns:

- Zero (0x0):
  - The requested advisory lock has been release.
- A positive value (release of lock denied)
  - Bit 0 = 1:
    - A lock release has been requested and no locks exists.
  - Bit 1 = 1;
    - A exclusive lock release has been requested and shared lock(s) already exists.
  - Bit 2 = 1;
    - A shared lock release has been requested and an exclusive lock already exists.

Marc Gauthier’s comments:
>>
>> The CAM-2 working will be meeting on January 10, 1995 to discuss the
>> options for CCB structure sizing and some general rules. The CAM-2
>> working group will then present to the general SCSI working group its
>> thoughts on the structure and sizing of the CCBs and the reasoning for its
>> selection. It is hoped that the general SCSI working will comment on the
>> selection (e.g., approve, disapprove, enhance the CCB, sizing and
>> general rules or have specific ideas).
>>
>> The migration of the CAM specification which is based upon the SCSI-2
>> specification to the CAM-2 specification and will be based on the SCSI-3
>> presents a number of problems. The largest problem that SCSI-3
>> presents to CAM is the converting from a dense addressing space to a
>> extremely sparse addressing space. The expansion of the target and
>> LUN from 8 bits to 64 bits will break all the currently written CAM
>> software. The term break used in the previous sentence means that the
>> currently written software will have to be modified and restructured to
>> conform with CAM-2 and SCSI-3.
>
>I concur; not only because the new CAM-2 CCBs will have bigger fields,
>and different layout, but also because the assumption that one can
>organise active logical units by arrays indexed by SCSI ID and LUN
>is no longer true. Even simple things like scanning the bus for
>devices, ie. trying every possible ID (and LUN for responding IDs)
>won't work in many environments. Scanning for reachable devices
>likely cannot be done by client/application software anymore, but
>would have to be handled through CAM; appropriate primitives
>(CAM commands) for getting a list of active devices need to be
>adapted or added.

The CAM-3 working group will be addressing the scanning issues and if you have any specific proposals to address this issue we would like to consider them.

>
>> Some other problems with the CAM specification today is its lack of
>> expansion capability for new features to both the CAM and SCSI
>> specifications and its assumption of a 32 bit processor world.
>> The CAM-2 working group will be working towards a specification
>> that is easily expanded, fully dynamic and processor word size
>>independent.
>>
>> The following are some ideas that should help in solving some of these
>> issues.
>>
>> Processor word size independence:
>> To allow transportability (not binary comparability) of CAM-2 peripheral
>> drivers, and SIM/HBA's between different machine platforms and
>> operating systems, a cam types definition file (cam_types.h) will be
>> provided by the supplier of the xpt. The cam_types.h header file shall
>> define the CCB structures member sizes. While the CCB's size varies
>> based on machine word size, they are of fixed size and structure
>> member offsets are fixed for a specific machine word size.

>
>Your example definitions seem to imply, however, that the size would
>stay the same for all architectures but only the way in which it is
>defined varies (eg. defining two U_32's instead of one U_64).
>Any change in size would be only due to an architecture's natural
>alignment, and that could likely be normalized with proper padding.
>

Yes, that is correct that the structure members are defined the same
and the member sizes would remain the same. The CCB size difference
would only be for alignment and this is due to the fact the most
machine architectures today have restrictions on accessing storage
elements not proper boundaries. This is a little wasteful of memory
but it allows a easier transport of drivers across platforms.

As for defining members (other then pointers) for a maximum of 32 bits
(CAM_U32) this is also done to allow for easier transport of code between
machines (no changes are needed in the code for casting).

>> The supplier of the xpt shall also the supply a CCB definitions file (cam.h)
>> which shall use the defined CCBs as specified by CAM-2. The CCBs
>> specified in CAM-2 shall specify specific member names of the CCB and
>> the size of each member.
>
>Why does it have to be supplied by the xpt vendor, and not be the same
>for all architectures of a given word size? (not many word sizes to deal
>with, likely only 32 and 64 bits, though 16 bits and 128 bits might be
>considered; each could have standard definition...?).

The XPT is the anchor of CAM and there is one XPT per system, but there
are multiple peripheral drivers and SIM/HBAs. Having the XPT define
the CCBs allows for just 1 definition file which prevents mismatched
CCB definitions. Also having the a supplied definition file allows
for applications to be compiled on a given machine that understand
CAM (e.g., a passthru driver that lets an application send commands
to a device).

The working group feels that if we define 1 implementation (32 or 64 bits)
as a reference and state the data type and alignment rules there is less chance
of conflicts within the document, and the result is then same if we defined a number of platform
implementations.

>
>> Alignment (address boundaries) of the CCB and its members shall also be
>> specified by CAM-2. It shall be the responsibility of the xpt supplier to
>> preserve those alignments as specified by the alignment rules. The
>> alignment rules allows CCB members to at a known offset for a 16, 32,
>> 64, etc bit processors.
>>
>> I have chosen the C language to represent the types definition file.
>> The cam_types.h file contents shall contain type definitions of the follow:
>> typedef char   I_8;   /* 8 bits */
>

>You should say "signed char" rather than "char", which may be signed
>or unsigned depending on compiler and compiler options. Actually, it is
>probably safer to say "signed" in other signed definitions below.

The editor took your idea and implemented data type rules in a more generic fashion.

```
>
>>  typedef unsigned char    U_8;          /* 8 bits */
>>  typedef short  I_16;                  /* 16 bits     */
>>  typedef unsigned short   U_16;         /* 16 bits     */
>> /* For different machine word sizes ( e.g., 32 or 64 bit )
>> #ifdef 32_BIT
>>  typedef  long I_32;                    /* 32 bits          */
>>  typedef unsigned long  U_32;           /* 32 bits          */
>> #endif /* 32_BIT */
>> #ifdef 64_BIT
>> typedef int I_32                        /* 32 bits     */
>> typedef unsigned int U_32               /* 32 bits     */
>>
```

>> Alignment Rules:
>>   All (x)_8 (chars) shall being on a 8 bit address boundary and shall
>>   be 8 bits in length.
>>
>>   All (x)_16 (shorts) shall being on a on a 16 bit address boundary
>>   and shall be 16 bits in length.
>
>Probably shouldn't say (shorts) or even (chars) as these don't necessarily
>have the given number of bits. ("Octets" should by definition
>however always consist of eight bits.)

The editor believes he has resolved this based on your comments.
Please refer to the CAM-3 Data Type and Structure Size Definitions proposal, if you have further comments please feel free to make them.

>
>>   All x_32 (32 bits) shall being on a 32 bit address boundary and
>>   shall be 32 bits in length.
>
>Shouldn't the same apply for 64 bits? Why not simply a general rule,
>that any basic type should be aligned according to its size?
>

The working group believes that the specification of a 64 bit data type other than pointers will have transportability issues. The working group has choose not to define a 64 bit data type.

>>   All pointers shall begin on a machine word address boundary shall
>>   be of machine word size.
>
>Careful, is machine word size necessarily the size of a pointer?
>On 8086's (and 68000's?), I would think machine word size is 16,
>but pointers are 32 bits. (A general rule would sidestep this issue.)
>
The working group and the editor believes they has resolved this issue.
Please refer to the CAM-3 Data Type and Structure Size Definitions proposal, if you have further comments please feel free to make them.

>> All structures and unions and arrays shall begin and end on a
>> machine word boundary. If they don't they shall be padded out to
>> the machine word boundary.
>>
>> All CCBs shall begin on a machine word boundary.
>
>Should alignment to cache lines also be considered? In high-performance
>implementations on certain hardware, cache-alignment might simplify
>low-level code where it has to deal with explicit cache management.
>Of course this may make sense with a 68040's 16-byte cache lines, but
>probably not with 1024+ byte cache lines in certain multi-level caching
>schemes.
>

The working group believes it should look at this in general terms (e.g., where we don't destroy performance for the various cache schemes), but the working group maybe some what limited in our options due to trying to preserve compatibility we CAM-1

>> Due to compiler differences between machine platforms and
>> operating systems if the next defined member type does not align
>> with the specified alignment then it shall be padded to force
>> alignment.
>>
>> Padding shall use the following type:
>> U_8 :8; /* Alignment Padding */
>
>Couldn't bigger sizes also be used as long as padding with them is aligned
>to their size? And allow using arrays of the biggest type for long padding
>sections aligned to the biggest type? Otherwise the above gets rather
>verbose with long padding areas.
>

The working group has specified general rules and it is up to the XPT supplier to define padding as long as they maintain the rules.
Please refer to the CAM-3 Data Type and Structure Size Definitions proposal, if you have further comments please feel free to make them.

>> I have looked at a number of different options for the CAM-2 CCBs from
>> a GPP similar structure arrangement to what has been proposed/talked in
>> the distant past CAM-2 Working Group meetings. I believe the following
>> CCB Header definitions are the most optimal for CAM-2. The below
>> CAM-2 header definitions allow for CAM and CAM-2 peripheral drivers
>> and SIM/HBAs to co-exist in a system also allowing for backwards
>> compatibility.
>>
>> I have rejected the GPP similar structure arrangement due to the following
>> reasons:
>> To large of a departure from the current definitions.
>>
>> The use of too many data pointers. The use of pointers to describe
>> data is very flexible but has many drawbacks. The allocation of the

```
>>      storage areas impacts O.S. performance the more you allocate the
>>      greater the impact.  Has a tendency to be wasteful of a critical
>>      resource (system memory) where most operating systems have
>>      power of 2 kernel memory allocators so when you ask for a buffer of
>>      36 bytes you get a 64 byte bucket.
>
>I would be interested to have a look a look at this "GPP" proposal, if
>possible -- at least to see the history of what was rejected and
>understand the above reasons for its rejection.  And perhaps avoid
>repeating anything, though I somewhat doubt I would have come up
>with extra pointers... (???)
```

GPP is available on the SCSI BBS.

```
>
>
>> CAM (CAM-1) CCB header:
>> /* Common CCB CAM header definition.
>>  * For 32 bit machines
>>  */
>> typedef struct ccb_header {
>>      void *my_addr;              /* The address of this CCB */
>>      U_16 cam_ccb_len;           /* Length of the entire CCB */
>>      U_8 cam_func_code;          /* XPT function code */
>>      U_8 cam_status;             /* Returned CAM subsystem status */
>>      U_16 cam_hrsvd0;            /* Reserved */
>>      U_8 cam_path_id;            /* Path ID for the request */
>>      U_8 cam_target_id;          /* Target device ID */
>>      U_8 cam_target_lun;         /* Target LUN number */
>>      U_32 cam_flags;             /* Flags for operation of the subsys */
>> }CCB_HEADER;                     /* structure ends on 32 bit boundary */
>>
>> CAM 2 CCB header:
>> /* Common CCB CAM 2 header definition.
>>  * For 32 bit machines
>>  */
>> typedef struct ccb_header2 {
>
>You might give thought to making identifiers unique in the first 8 chars,
>for those older compilers that don't take any more into account.  This
>hasn't been done in the past (e.g. cam_target_id vs. cam_target_lun),
>but we've had to change structure definitions in our environment to
>follow our portability guidelines.
```

The working group believes that we are within the ANSI C rules for identifiers (the editor will verify). The actual field names have not been defined yet since they were not defined in CAM-1. There currently is an informative annex that defined member names but the member names still need to be resolved.

```
>
>>      void *my_addr;              /* The address of this CCB */
>>      U_16 cam_ccb_len;           /* Length of the entire CCB */
>>      U_8 cam_func_code;          /* XPT function code/CAM 2 signifier */
>>      U_8 cam_status;             /* Returned CAM subsystem status */
>>      U_32 cam_path_mid;          /* Path ID for the request (Most
>>                                   * significant)
>>                               */
>>      U_32 cam_path_lid;           /* Path ID for the request (Least
>>                               * significant)
>>                               */
>
>Here I would rather not see large integers split up explicitly at this level.
>At least not if expressing them in C.  Please!  For one, it forces a byte-order
>(something which has not so far been discussed).  Although a machine might
>be mostly 32 bit, it may have instructions that help it deal with 64 bits,
>and that should be left to its natural order if possible.  It also makes things
>more consistent.  I would rather, in 32 bit environments, define something
>like:
>
>    typedef struct I_64 {
>       I_32   msb;
>       I_32   lsb;
>    } I_64;          /* 64-bit signed integer, as a structure */
>
>    typedef struct U_64 {
>       U_32   msb;
>       U_32   lsb;
>    } U_64;          /* 64-bit unsigned integer, as a structure */
>
>You could then define struct ccb_header2 more cleanly and naturally
>using U_64's.  One might even keep the same definition for various
>architectures (compiler-specific variations might still occur because
>of padding, etc. considerations, but it is easier to only have to deal with
>writing special header files for exceptional compilers, which has to be
>done in any case anyway).
```

The working group believes that the specification of a 64 bit data type other than pointers will have transportability issues. The working group has choose not to define a 64 bit data type. The breaking up of a 64 bit type into 2 components having a least significant half and most significant half does not force byte ordering in CAM. All ordering of the an data entity that is broken up (e.g., a 64 bit address) is consistent with the ordering of SCSI CDBs. Byte ordering is forced at the protocol level (e.g., FCP, SIP etc.).

```
>
>If one wants to avoid the potential confusion of the "U_64" type with
>a real integer, one might name it something else like "U_64_struct"
>or whatnot.  However these pseudo U_64's can be treated like integers
>to the extent that they can be assigned and declared as storage (the only
>thing to watch out for is passing structures as parameters by value to
>functions, which may work differently on different compilers or
>environments).
```

Yes but as your code moves from a 64 bit implementation to a 32 bit implementation transportability issues arise when dealing with a 64 bit data type where 32 bits is the largest data type supported.

```
>
>Little arithmetic will probably ever be done with
>U_64's; if I understand/recall correctly, scsi ids and luns are
>mostly treated as "magic cookies" in SCSI-3.  As for path ids, what
>they are is yet to be defined here, though if 64 bits are ever to be
>used, they will almost certainly be magic cookies as well (modulo the
```

>all 1's path id for XPT, and without excluding the possibility of other
>special values). I expect that 64 bit path ids will be used by
>providing another primitive for explicitly scanning for path ids.
>That is, XPT would provide another command, "get next path id"
>which takes a path id as a parameter: the XPT would return the
>next path id following the one provided, or the first path id if given
>a special value (say, the XPT all 1's path id). The XPT would
>determine the format of the 64-bit path ids, and would format
>them such that the "get next path id" primitive can be executed
>efficiently (or, and this may be more flexible in the end,
>require that to scan for path ids requires a context, and that the
>same context be used for getting successive path ids, similarly
>to the way opendir() and readdir() work in ANSI/POSIX C).
>This primitive would be trivial to implement in
>current CAM-1 software in migrating them to CAM-2, if they
>just use the least significant 8 bits (or any 'n' lsbits) of path ids
>as an index into an array of SIMs as is currently done in CAM-1.
>So why not define such a "get next path id" primitive right away?
>It doesn't add complexity, yet allows for immediate expansion and
>compatibility with arbitrary 64-bit path ids (to the discretion
>of the XPT of a given architecture/environment). Thus instead
>of saying "we define path ids as taking up 64-bits but only the
>8 lsbits are significant at this time" as below, we make it
>trivial to implement path ids that only use the 8 lsbits but allow
>full use of all 64 bits. Will not some
>similar mechanism be needed for scanning for SCSI IDs and LUNs,
>since they can no longer be simply scanned numerically? Having
>some similar mechanism for discovering path ids right away
>would be more consistent, and simple to understand.
>Has any design been put forward for a method of scanning for
>SCSI IDs and LUNs? Maybe "get next SCSI ID" and/or
>"get next LUN" primitives, with scanning context as suggested
>above for scanning path ids? Or is there something totally
>different needed, some implementation requirements I just
>haven't considered? It just seems the obvious solution.

The above issue has not been worked as of yet, bit a method needs to be developed along the lines of what you have suggested.

```
>
>>      U_32 cam_target_mid;          /* Target device ID (Most significant) */
>>      U_32 cam_target_lid;         /* Target device ID (Least significant) */
>>      U_32 cam_target_mlun;        /* Target LUN number  (Most
>>                                  * significant)
>>                                  */
>>      U_32 cam_target_mlun;        /* Target LUN number  (Least
>>                                  * significant)
>>                                  */
>>      U_32 cam2_func_code;          /* The Real CAM 2 function code. */
>>      U_32 cam_flags;              /* Flags for operation of the subsys */
>>      U_8 :8;                    /* Reserved for expansion */
>>      U_8 :8;                    /* Reserved for expansion */
>>      U_8 :8;                    /* Reserved for expansion */
>>      U_8 :8;                    /* Reserved for expansion */
>>      U_32 cam_target_flags         /* Target mode flags */
>>      U_8 :8;                 /* Reserved for expansion */
>>      U_8 :8;                 /* Reserved for expansion */
>>      U_8 :8;                 /* Reserved for expansion */
>>      U_8 :8;                 /* Reserved for expansion */
>>      U_8 :8;                 /* Reserved for expansion */
>>      U_8 :8;                 /* Reserved for expansion */
>>      U_8 :8;                 /* Reserved for expansion */
>>      U_8 :8;                 /* Reserved for expansion */
>>      U_8 :8;                 /* Reserved for expansion */
>>      /* The above reservation of 8 bytes is done to allow for future routing
>
>(nitpicking) There's one too many byte (9 are present).
```

A number of people mentioned that the editor can't count.

```
>
>>       * of the request over a communication path (e.g., network).  The
>>       * boundaries of  what a host (system) is has rapidly changed from
>>       * what it was 2 years ago. While I haven't thought this out completely
>>       * yet (maybe 16 bytes is needed source and destination addresses) I
>>       * have placed it here as a marker to provoke some thought into it.
>>       */
>> }CCB_HEADER2;           /* structure ends on 32 bit boundary */
>
>With the above suggestions this structure would look like:
>
>/* Common CCB CAM 2 header definition.
> * For 32 bit machines
> */
>typedef struct ccb2_header {
>     void *my_addr;               /* The address of this CCB */
>     U_16 cam_ccb_len;            /* Length of the entire CCB */
>     U_8 cam_func_code;           /* XPT function code/CAM 2 signifier */
>     U_8 cam_status;              /* Returned CAM subsystem status */
>     U_64 cam_path_id;            /* Path ID for the request */
>     U_64 cam_target_id;          /* Target device ID */
>     U_64 cam_target_lun;         /* Target LUN number */
>     U_32 cam2_func_code;         /* The Real CAM 2 function code. */
>     U_32 cam_flags;              /* Flags for operation of the subsys */
>     U_32 cam_pad0;               /* Reserved for expansion */
>     U_32 cam_target_flags        /* Target mode flags */
>     U_64 cam_pad1;               /* Reserved for expansion */
>} CCB2_HEADER;         /* structure ends on 32 bit (and 64-bit) boundary;
>                         would also end on 64-byte (512-bit) boundary if
>                         an additional 64 bits were added */
>
>> Note there is a restriction that there can only be 0x0 to 0xef SIM/HBAs in
>> CAM-2. This restriction will be lifted when CAM-3 (if there is one) is
>> defined. When this occurs it is expected that there all SIM/HBA's must be
>> migrated to CAM-2.
>
>There is no need to be so restrictive to maintain a smooth upgrade path.
>Likely initial implementations may only use 8 bits, but a method that
>allows use of full 64 bits can be defined right away.  See comments above.
>Perhaps what should be said is that at most 0xf0 SIM/HBAs may be
```

>registered at any time in CAM *if* it supports CAM-1 CCBs (one can
>imagine newer environments that could reasonably support CAM-2
>without support for CAM-1, if all drivers have to be adapted to this
>new OS/environment anyway). Let it be clear that this requirement
>is solely a consequence of support for both CAM-1 and CAM-2 CCBs
>at the same time by an XPT. One can also imagine that an XPT could
>support both types of CCBs by maintaining an array (of no more than
>0xf0 elements) of HBA/SIMs each with their own full 64-bit path id,
>and taking the 64-bit path id in CAM-2 CCBs and the 8-bit index in
>CAM-1 CCBs. One might ask why one would do that. Perhaps if there
>are more than 0xf0 SIM/HBAs, only the subset of SIM/HBAs that are
>accessed by CAM-1 peripheral drivers would have an entry in the
>array and be accessible by CAM-1 CCBs. After all, if we don't expect
>more than 0xf0 SIM/HBAs, why have 64 bits for path ids in the
>first place?

There are many possibilities -- we can restrict CAM-1 SIM/HBAs to
0x0 - 0xfe pathids and CAM-3 SIMs/HBA's to 0x100 upwards.

This needs to be discussed much more fully by the working group.

```
>
>> Rules for the CAM 2 XPT:
>>      The XPT shall support both CAM and CAM-2 CCBs.
>>
>>      The XPT shall in xpt_action() determine if this is a CAM or CAM-2
>>      function and route accordingly.
>>
>>      The XPT shall be addressed by a cam_path_id of 0xFF (CAM) and
>>      a cam_path_mid of 0xFFFFFFFF with a cam_path_lid of
>>      0xFFFFFFFF (CAM-2). This means the CAM-2 XPT can be
>>      addressed by both addresses so peripheral drivers can determine if
>>      this is a CAM or CAM-2 XPT.
>>
>>      The XPT shall report in the CAM Path Inquiry CCB that it is a
>>      CAM-2 XPT.
>>
>> Rules for the SIM/HBAs:
>>      The SIM/HBAs shall report in the CAM Path Inquiry CCB that it is a
>>      CAM or CAM-2 SIM/HBA.
>>
>>      The SIM/HBAs that support CAM-2 shall report in the CAM-2 Path
>>      Inquiry CCB that it supports CAM-2 CCBs and it shall report if it
>>      supports CAM CCBs (optional).
>>
>>      The SIM/HBA shall determine through a CAM Path Inquiry CCB that
>>      it is a CAM or CAM-2 XPT.
>
```

>By "determine", do you mean "report"? (If yes, it seems redundant
>with the preceding paragraph, if not, it seems contradictory with it...?)

The above rules were just a stab at getting some ideas down on paper and needs to be clarified/expanded.
The above rules have not been approved by the working group and are just the beginnings of a framework.

>
>Also, the XPT (presumably) always determines the path id, including
>64-bit path ids for CAM-2. This would mean that if the SIM/HBA
>supports CAM-2, it should learn of its 64-bit path id(s) from the XPT.
>Ie. xpt_bus_register() should return a 64-bit path id (by reference
>in 32-bit systems where functions cannot return more than 32 bits at
>a time), and xpt_bus_deregister() should take a 64-bit path id as
>parameter (again, possibly by reference if needed).

The how a CAM-3 SIM/HBA registers and deregisters has not been specified as of yet (the editor believes another set of routines is needed for CAM-3 SIMs).
>
>> Rules for CAM-2 peripheral drivers:
>>
>>   Peripheral drivers shall determine through the CAM Path Inquiry
>>   CCB if the XPT is a CAM or CAM-2 XPT.
>>
>>   Peripheral drivers shall determine through the CAM Path Inquiry
>>   CCB if the addressed SIM/HBA is a CAM or CAM-2 SIM/HBA.
>
>Presumably to know whether its CCBs are supported, or to know which
>CCB to use in talking to the XPT if it (the PD) supports both.
>

Yes that is correct.

>>   A peripheral driver shall support CAM or CAM-2 and optionally both.
>
>
>I should mention that we are currently designing a SCSI support
>architecture for our realtime message-passing multiprocessor OS, Harmony
>(see <http://wwwsel.iit.nrc.ca/harmony.html>). Because it is fully
>message-passing, without necessarily shared memory between processes
>(eg. on multiprocessor systems connected by means other than a common bus),
>CAM request structures don't fit naturally (or efficiently) into messages.
>So I have completely redesigned CAM structures to provide us with a
>native SCSI interface between PD and SIM/HBA that is more easily and
>efficiently translated into messages -- without, incidentally, losing
>efficiency or simplicity for non-message-passing environments.
>As such they are no longer CAM, but may be useful in helping design
>CAM-2 structures.

We must take care in designing the CAM-3 structure to preserve
some compatibility. If you would like a CAM definition for
message passing operating systems please write it up and the working will consider your proposal.

>
>The main difference is that "input" and "output" fields have been
>identified and separated. This way output fields (sent by peripheral
>driver to SIM/HBA [via XPT]) can be sent in one contiguous message,
>and "input" fields be received just as efficiently as well. Target-mode
>only fields have also been separated and are only present in messages
>exchanged by target-mode peripheral drivers (the design for target-mode
>support is still incomplete however). Fields that are common to all
>commands of a linked command chain had also been separated from fields

>that are unique (or partially unique) for each linked command, however
>linked commands are in general seldom encountered and are not usually
>among the commands that form the bulk of I/O requests such as READ and
>WRITE (is there experience to the contrary?). So support for optimizing
>messages when linked commands are present is being dropped, since any
>added complexity for optimizing linked commands will most likely slow
>down the more general case of unlinked commands. I will show example
>of these structures later in a subsequent message.

From the editors experience linked commands are very rare.

>
>One thing that has been avoided is duality of meaning of certain fields,
>such as a data pointer being either a pointer to a block of data or a
>pointer to a scatter/gather list. Rather, only a pointer to a scatter/
>gather list is present, but a single scatter/gather element (pointer +
>size) is present in the I/O request structure, which the scatter/gather
>list pointer can point to in the simple case of a single block of data
>being transferred. A similar thing is being done for the command data
>bytes (CDB). However, one point that is still unclear to me is whether
>SCSI-3 commands can ever exceed 16 bytes. If not, then the simplest
>thing to do is to provide 16 bytes for the CDB and not support
>interpretation of the CDB as a pointer to the real CDB's. If they can,
>and there is no fixed and reasonably short bound on the size of the CDB,
>then what we'd do is provide 16 bytes for the CDB and a separate pointer
>to the CDB. The pointer would always be used, and may point either to
>the 16-byte CDB array within the request structure, or to an external
>array of command data bytes if more than 16 bytes are needed.
>Avoiding such duality makes the code easier to understand, and *more
>efficient* (since code doesn't have to check the meaning of a field;
>it simply always accesses the same, more general field).

Vendor unique commands can be greater then 16 bytes. The duality of fields does present some overhead checking in the SIM/HBAs but it also allows flexibility in the peripheral drivers to accomplish their tasks.

>
>I would have liked to attend the CAM WG meeting, however this is
>beyond the scope of our budget at this time. I am, nevertheless,
>very much interested in what happens in the development of CAM-2.

We will keep you posted and I appreciate your interest, ideas and
criticisms (it is the only way to develop a complete software standard).

>
>
>Regards,
>
>-Marc
>
>--
>Marc E. Gauthier
>Software Engineering Lab, Institute for Information Technology (SEL,IIT)
>National Research Council Canada, Bldg M-50, Ottawa ON Canada K1A