SCSI-2 CAM Discussion Paper

Title: Minor Problems with CAM
Author: Bob Huntsman/Al Youngwerth, Extended Systems
Date: June 27, 1991

References

1. X3T9.2/90-186, *SCSI-2 Common Access Method - Transport and SCSI Interface Module*, rev 2.4, I. Dal Allan (Ed.)

2. X3T9.2/91-96, *Clarifying the relationship of XPT to SIM*, Bob Huntsman, June 27, 1991

Discussion

This paper lists some minor problems noted in our review of [1] that should be corrected, if possible. Due to time constraints, we were unable to produce specific change proposals by the July working group, but may be able to do so by August.

We chose to forward this list of concerns to X3T9.2 now, in the hope that some of them, if not all, can be corrected prior by the August meeting.

The problems are not listed in any particular order.

Problem #1:

Normative versus informative information.

The offical defining part of a standard contains normative information - that is information on defining the subject of the standard. Annexes contain informative information - information that is very useful when trying to understand the standard, but not part of the standard. A suggestion: label the Annexes as:

ANNEX A (Informative only) ...

to emphasize this distinction.

Furthermore, some of the examples that are currently in the main body probably should be moved to an annex. For example, the coding example in 7.3.2.1 and 7.3.2.2 (and maybe 7.3.1.1) appear to be examples, and not standardized code.

Problem #2:

"Should" versus "shall"

[1] uses the terminology "should" when "shall" is probably intended. The distinction is important because "should" implies a conforming implementation has a choice, while "shall" specifies there is no choice.

For example, (and there are many others besides this one), section 7.3.1 says, "...XPT/SIM modules should be loaded as character drivers". As it now reads, if an implemention chooses to violate this rule, it still conforms. I believe that in most cases, when "should" is used in [1], "shall" was intended, i.e., the behavior is not intended to be optional in a conforming implementation.

Bottom line: I think the whole documented ought to be reviewed for occurances of "should". If "should" was really intended, the rule should be written in terms of OPTIONS. If "shall" was intended, the wording needs to be corrected.

Problem #3:

The general section of the proposed standard describe a model containing distinct XPTs and SIMs. This model is attractive, because a provider of an HBA need only provide a SIM for each operating environment to be supported, and need not understand the intricacies of an XPT.

Unfortunately, the DOS section contains a very big disappointment: every SIM writer now has to write the XPT. Why break the model for DOS? Since the current specification already specifies a software interrupt calling convention, it is very easy to follow the general model and provide separate modules for XPT's and SIM's.

Having DOS follow the same model is important, because it makes it MUCH easier for SIM writers (and XPT writers for that matter) write a single piece of code with a few conditionals for various operating environments.

I think that an independent XPT for DOS is as reasonable and desirable as an independent XPT is for all of the other operating environments. The general model will work fine for DOS as it does for UNIX, NOVELL, OS/2, etc. (The fact DOS is not a multi-tasking OS doesn't seem to be particular relevant as to whether or not XPTs are distinct from SIMs...)

Furthermore, with respect to DOS, the current specification is over-specific in some unimportant areas, and under-specific in some important areas:

It is overspecific in that:

As long as the software interrupt interface is used, from an inter-operability point of view, it doesn't really matter if the module is loaded as a character driver or a TSR. Pathologically, a TSR and a DOS character driver are nearly indistinguishable, in that they become part of DOS. (NOTE: TSR's have been known to be problamatic under DOS. However, it is TSR's that use HOT KEYS and chained DOS interrupts that cause problems. The proposed DOS XPT or SIM is very "clean".) Loading a driver as a TSR is very reliable and has one significant advantage over a device driver: Under certain circumstances, it is completely removable, thus supporting the notion of removable SIMs. Consequently, this useful implementation of an XPT/SIM should not be arbitrarily prohibited unless it can be shown inter-operability is compromised.

The bottom line here is that the loading mechanism used under DOS neither promotes nor inhibits

interoperability. The requirement that a conforming DOS XPT/SIM be MUST be implemented as a combined module and MUST be implemented as a character device driver should be eliminated.

The DOS OS section is underspecified in that neither the binary format for the CCB's nor the mechanism that a driver would use to determine the software interrupt being used are specified. If interoperability is to be promoted by CAM, these two details need to be well specified (This is probably true for all operating environments - see problem #10.)

Problem #4:

In Section 7.3.2.2 in a coding example, the return code in AX is shifted left 8 bits. I couldn't determine from the rest of the standard why that was necessary...

Problem #5:

In the Unix OSD section, reference is made to two structures, cam_conftbl[] and cam_edt[]. Although I have been able to "guess" there structure and usage by looking at both the Unix section and the provided CAM.H Unix file, it would be useful if these structures were clearly defined.

Problem #6:

Certain values of a CCB are identified as OUTPUT parameters. But which values is a conforming implementation required to check and which can be ignored? Obviously the function code must be checked. But what about "address of this CCB"? If that field is bad, is a conforming implementation required to returned an error or allowed to process the CCB anyway? The proposed standard does not adequately address this subject.

Problem #7:

In clause 7.1.6.1 and in many other places, the following phrase is used:

"...and the return code is either Success or Failure."

It is not clear to me whether "return code" refers to the function return code (which I assume is returned immediately) or a CCB status code. I assume the former is correct, but I think the exact meaning should be clarified. The numerical value of SUCCESS and FAILURE are undefined (i.e. does a return code of 0 indicate SUCCESS or FAILURE). How should a driver treat a return code of FAILURE - how does it compare to a CAM STATUS failure code?

Problem #8:

"xpt_ccb_malloc()" and "xpt_ccb_free()" are only specified under the UNIX environment. Is it true that a UNIT implementation MUST use ONLY CCBs acquired by xpt_ccb_malloc()? (This is implied, since it appears the xpt_ccb_free() is automatically called).

If I implement xpt_ccb_malloc in a non-UNIX environment, am I conforming or violating the CAM

standard?

Problem #9:

Table 6-1 defines opcode bits returned by an ASYCNCH CALLBACK. If would be useful if a certain range of the unspecified bits were reserved for vendor specific purposes. For example, if a vendor supports a vendor specific XPT function code, he may have some asynchronous requirements. Perhaps 4 of the undefined bits could be reserved for vendor specific purposes: 0x8000, 0x04000, 0x02000, 0x01000 should be reserved for vendor specific implementations.

Problem #10:

The second paragraph of Clause 8., "CAM Control Blocks" reads:

"The sequence of the fields in the data structures will be consistant between vendors, but not necessarily the binary contents. The size and definition of the fields in the data structures can vary between operating systems and hardware platforms, but the vendors are expected to provide compiler definitions which can be used by third-party attachments".

Question: Should the binary contents of a CCB be allowed to vary from vendor to vendor FOR A GIVEN OPERATING ENVIRONMENT?

Clearly the last sentence of Clause 8 quoted above is intended to restrict the binary content of a CCB, but it does it very indirectly. In essence, if an XPT is defined for a particular operating environment, and the implementor of that XPT makes available a compiler definition for the particular environment, then the binary definition of the CCB is nailed down.

This reflects in different way the problem addressed by [2]: Currently an XPT must be written, specified, and distributed, before a SIM writer can write a SIM, because the binary version of the CCB (or the "compiler definition file" of clause 8) is not available in [1] until the XPT is implemented.

I think the entire industry is better served if XPTs are completely specified by X3T9.2 (including calling conventions and binary representations of all fields) instead of asking, waiting, and hoping OS vendors will do it. OS vendors are not going to be motivated until CUSTOMERS ask for it, and customers are not going to ask for it until CAM is widely accepted. CAM is not going to be widely accepted until it is widely supported. This cycle can only be broken if somebody besides OS vendors can initially provide XPT’s.

Bottom line: CCB binary formats and calling conventions need to be completely specified in this document by X3T9.2. This will allow those who are motivated to make CAM succeed (primarily SCSI device makers) to provide XPTs for critical operating environments, that will "plug and play" with SIMs of other SCSI device manufacturers. Eventually, OS vendors will "bless" a popular XPT for its environment, and the desired interoperabilty goal of SCSI and CAM will be realized.

Problem #11:

[1] really needs an index. As a new reader, it is very difficult to locate all revelvent sections for a particular topic.