**To:** X3T9.2 Committee Membership

**From:** Edward A. Gardner, Digital Equipment Corporation

**Subject:** Dealing with SBP Target Resets

This proposal describes a mechanism for dealing with SBP target resets, specifically for ensuring that initiators can reliably recognize when an SBP target has been reset. For this discussion a target reset is any event that results in an SBP target losing its internal context, in particular its knowledge of logged in initiators. This discussion focusses on situations where only the target is reset. The P1394 bus is not reset, so node offsets remain constant, nor are initiators reset. The cause of the target reset is irrelevant to this discussion. The target reset may be spontaneous, such as the target detecting a transient internal error and having to reset itself, as well as target resets explicitly initiated by some initiator.

**Problem Scenario**

Consider an Initiator A that has logged in with a target as (for example) Initiator ID 1 and has been accessing that target. During a brief period of inactivity, perhaps lasting a few seconds, the target resets. Another initiator, say Initiator B, discovers the target has reset, logs in, and is assigned the same Initiator ID 1 by the target. Note that the target reset has erased all context about previously logged in initiators, so the target cannot “remember” that Initiator ID 1 had been assigned to Initiator A.

Subsequently Initiator A ends its period of inactivity and taps the target. The tap message contains Initiator ID 1, which is logged in and valid. The tap message is indistinguishable from valid tap messages sent by Initiator B, and will be erroneously accepted by the target. The target thinks it is talking to a single initiator, when in fact it is talking to two initiators. Bad things happen.

The problem is that all the identifiers so far defined in SBP are uncontrollably dynamic. The Initiator ID byte is assigned by the target for the convenience of the target. It cannot indicate whether or not taps are from the same initiator across target resets.

**Proposed Solution**

The simplest robust solution is to have the Initiator ID byte value assigned by the initiators, not by the target. An initiator would propose an Initiator ID value in its login request, rather than the target assigning an Initiator ID and returning it in the status message or a buffer. The target should return a distinct status code if the proposed Initiator ID is already in use.

I expect there should also be a way for hosts to determine which Initiator IDs are currently logged in, for use by bus monarchs or similar configuration management software. One way to do this might be a function that returns a list or bitmap of Initiator IDs that are currently logged in. Alternately a function that “polls” a specified Initiator ID and returns whether or not the ID is logged in, without altering the login status of that ID.

Multiple initiators would be responsible for requesting non-conflicting Initiator IDs. The mechanism for doing so is outside the scope of SBP. Many multi-host systems have some concept of a host number, which could be directly used as the host’s Initiator ID. Alternately some sort of global resource allocator (e.g., the monarch) could be used to assign Initiator IDs.

Again, SBP simply requires that different initiators have distinct Initiator ID values, it does not specify the means for accomplishing this.

The cost of this solution is that, since targets no longer assign Initiator Id values, a target cannot assign Initiator ID values such that the Initiator ID value itself is an index into the target's internal initiator context slots. The actual target impact depends on whether the target is using hardware or firmware to distinguish different initiators:

1. If the target distinguishes initiators in hardware, then it is providing separate tap register addreses for each initiator. The target uses some portion of the tap register address to distinguish between initiators. That portion of the tap register address is (in effect) the target's internal initiator context slot index. The target merely verifies that the Initiator ID value in the command block matches the expected value, it is not used to locate the target's internal initiator context slot. This style of target implementation is not affected by this proposal.
2. If the target distinguishes initiators in firmware, then it must map the Initiator ID value into an internal initiator context slot index. A fully general way to do this is with a 256 entry translation table. The target's firmware uses the Initiator ID value as an index into the translation table, the table entry contains the internal initiator context slot index. The size of a table entry depends on how many initiators the target supports, but in no case need be larger than a byte. The translation table itself would reside in ordinary RAM, not specialized memory.

Hybrid schemes, where some taps are distinguished with hardware and the remainder with firmware, would follow much the same structure. Those taps that use hardware to distinguish the initiator would not be affected. Those taps that use firmware to distinguish the initiator would use a translation table. In any case the maximum cost is 256 bytes of ordinary RAM and a few words of ROM to perform the translation. More specialized implementations that use even fewer resources are also possible.

**Alternate Solutions**

Fundamentally there are only three classes of solutions to this problem. The first and simplest is to have an identifier that is assigned and managed external to the target device. Since the identifier is managed outside the target device, it remains valid across target resets. The proposed solution is one instance of this class.

The proposed solution uses the externally managed (that is, initiator assigned) identifier as the only initiator identifier. An alternate approach would be to have two initiator identifiers, one assigned by the target and one by the initiator. However, the additional complexity doesn't seem warranted, since all it saves is the 256 byte translation table. An additional argument against this is that it would consume one of the increasingly scarce reserved bytes in the command block.

Parallel SCSI uses another approach within this first class. The SCSI bus ID is assigned and managed external to target devices, specifically when the bus is configured. Similarly, we could use the P1394 node address to identify initiators, except this has been previously discussed and rejected. Among its problems is the fact that translating a 16-bit node address to an internal initiator context slot index requires considerably more resources than translating an 8-bit value.

The second class of solutions is to have targets assign initiator identifiers and remember them across target resets. To be practical targets would have to remember initiator identifiers across all target resets, which means they would have to be stored in non-volatile memory. Such solutions can be made to work, but are quite complex and absurd for protocols such as SBP.

The third class of solutions is what will occur if we ignore this problem. Whenever an initiator detects that initiator identifiers have become invalid, it must ensure that all other initiators are notified before it re-logs in and continues operation. That is, whenever an initiator discovers that a target has reset, it must notify all other initiators and wait for an acknowledgement from all other initiators, before it can re-log in and continue operation. This is a complex and lengthy recovery. Furthermore, having to wait for acknowledgements before proceeding compromises high availablity, a major goal of multi-initiator systems, yet the only alternative is compromising data integrity.

This leads to why it is mandatory that SBP address this problem with a solution such as the one I proposed. If we ignore this problem, then the only standard approach for multi-initiator systems is the third class of solutions. But their consequences are intolerable for multi-initiator systems. The result is that every implementor of multi-initiator systems will be compelled to invent a vendor unique extension similar in concept to what I proposed. And every vendor will develop their own extension, different in details even though similar in concept, leading to the same problems of different device versions for each vendor as we have today. Addressing this problem in SBP today is necessary to avoid (or at least reduce) such interoperability problems.