Document: T10/00-410r0 Date: 16 November 2000
To: T10 Committee Membership
From: Edward A. Gardner, Ophidian Designs
Subject: Scatter/Gather and Bi-directional Data Transfers for SRP

Among the comments that I have received regarding SRP is a desire for scatter/gather data transfers. This would appear to be redundant, the design intent of SRP is that data transfers take place to a virtual memory region, with the host adapter mapping from the virtual region to physical memory. However, there appear to be two arguments in favor of a second, SRP layer scatter/gather function:

1. The host adapter may only support a limited number of regions. A VI or Infiniband^{tm} host adapter designed for IPC applications could reasonably support only a few regions, perhaps a dozen or even fewer. If that same hardware were later used for IO as well as IPC, a single region would have to be shared for many outstanding IO operations. That single region might be used to map physical memory, with SRP level scatter/gather used to reference discontiguous physical pages.
2. Virtually non-contiguous data transfers. Databases and logging file systems may wish to write items that are virtually non-contiguous as a single operation.

Note that both of these could be addressed with an improved host adapter. The first requires more host adapter resources. The second requires an adapter that can map virtually non-contiguous memory regions. The best long-term solution is to keep all scatter/gather mapping local to the host adapter, as that provides the best overall system performance. The need for this proposal is driven by occasional or non-performance critical functions, which do not warrant an enhanced adapter, or to allow use of adapters that do not include the necessary enhancements. In all cases I would label SRP layer scatter/gather as a kludge, but it’s a necessary kludge.

Scatter/gather transfers are fundamentally about associating multiple data segments with a single IO operation. Bidirectional IO also involves associating multiple data segments with a single IO operation, one or more for data-out access plus one or more for data-in access. Therefore this proposal addresses both functions.

Note that this is deliberately a minimal proposal. There are four approaches to defining multiple data segments per command:

1. Disallow multiple data segments per command, the approach in the current SRP draft.
2. Place the entire scatter/gather list in initiator memory and have the target fetch all or portions of it as needed. This is the approach taken in this proposal.
3. Place the entire scatter/gather list in the command IU. This has several problems. The initiator and target would have to negotiate the scatter/gather list size (IU size). The target would have to dedicate data structure and message buffer resources for the worst case scatter/gather list for its entire command queue, causing the target to restrict the size of its command queue. Note that each entry in a scatter/gather list (a data segment descriptor) is 16 bytes, the same size as the largest commonly encountered CDB. The reduced command queue will typically hurt performance more than the benefit of not having to fetch the scatter/gather list. While this approach might be optimal if all commands have the same scatter/gather list size (e.g. two data segments), it is not optimal for other environments.
4. Place a portion of the scatter/gather list in the command IU, and have the target fetch the remainder (if necessary) from initiator memory. This is clearly the most complex approach, as well has having some of the memory consumption problems of approach 2.

The question we do not know is how often SRP layer scatter/gather or bi-directional IO will actually be used. I believe they will be used infrequently. In the cases where they are used, the target’s fetch of the scatter/gather list will almost always be overlapped with electro-mechanical access delays, and therefore not affect performance. Therefore this proposal recommends the simplest solution, approaches 1 and 2 above. If subsequent experience with real product implementations proves these assumptions wrong, we can adopt either or both approaches 3 or 4 in an SRP-2. Note that those are clean supersets of this proposal, maintaining backwards compatibility will be straightforward.

SRP currently identifies a command's data buffer using a 16-byte structure in the SRP_CMD information unit, which I'll call a data segment descriptor (table 1). A data segment descriptor identifies a contiguous data segment within initiator memory.

Table 1 - Data segment descriptor

Bit / Byte	7	6	5	4	3	2	1	0
n+0	MSB							
• • •				DATA VIRTUAL ADDRESS				
n+7								LSB
n+8	MSB							
• • •				DATA MEMORY HANDLE				
n+11								LSB
n+12	MSB							
• • •				DATA LENGTH				
n+15								LSB

This proposal defines a indirect data buffer flag (IND). When IND is set to zero, the SRP_CMD information unit contains a data segment descriptor, just as in the current SRP draft.. The data buffer is comprised of the single contiguous data segment identified by that data segment descriptor (figure 2).

When IND is set to one, the SRP_CMD information unit contains a data segment list descriptor (table 2). This is similar to a data segment descriptor, except that the 32-bit DATA LENGTH field is broken into two 16-bit fields. A data segment list descriptor identifies a list of data segment descriptors in initiator memory, which together identify the command's data buffer(s). The TOTAL LIST LENGTH field contains the total length of the data segment descriptor list. For bi-directional commands, the data segment descriptor list is divided into two sublists. The OUT SUBLIST LENGTH field contains the length of the data-out sublist, which describes the data-out buffer. The remainder of the list comprises the data-in sub-list, which describes the data-in buffer. Both length fields are in bytes (i.e., the number of segment descriptors times 16), implying a maximum of 4095 segments in a list or sub-list.

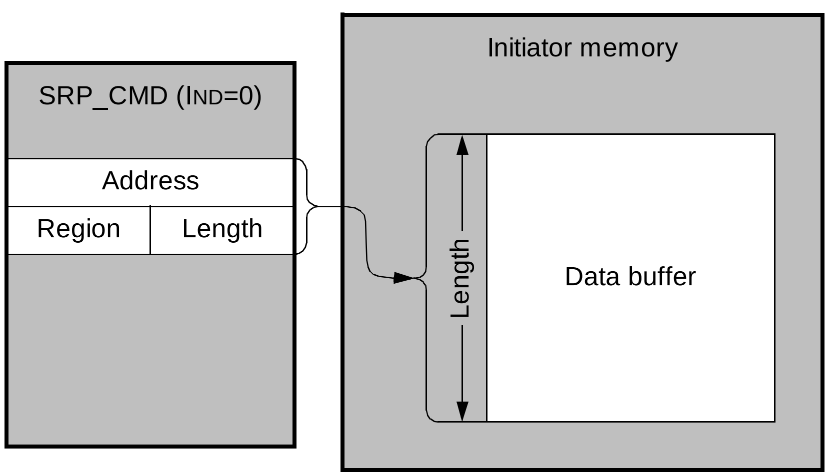

Figure 2 - Direct Data Buffer Mapping

Table 2 - Data segment list descriptor

Bit / Byte	7	6	5	4	3	2	1	0
n+0	MSB							
• • •				DATA VIRTUAL ADDRESS				
n+7								LSB
n+8	MSB							
• • •				DATA MEMORY HANDLE				
n+11								LSB
n+12	MSB			OUT SUBLIST LENGTH				
n+13								LSB
n+14	MSB			TOTAL LIST LENGTH				
n+15								LSB

An SRP_CMD information unit with either RDDATA or WRDATA set to 1 and the other set to 0 is a uni-directional command. For uni-directional commands with IND set to 1, all the segments in the data segment descriptor list concatenated together comprise the command's data-in or data-out buffer (figure 3). The TOTAL LIST LENGTH field indicates the length of the data segment descriptor list. The OUT SUBLIST LENGTH field shall be ignored. The sum of the DATA LENGTH fields in all the segment descriptors is the command byte count defined by SAM-2. A zero DATA LENGTH field in a segment descriptor is valid and indicates that the segment descriptor shall not contribute to the command's data-out or data-in buffer.

An SRP_CMD information unit with both RDDATA and WRDATA set to 1 is a bi-directional command (figure 4). Initiators shall set IND to 1 in all bi-directional commands. A bi-directional command's data segment descriptor list is divided into two sub-lists. The length of the data-out (first) sub-list is specified by OUT SUBLIST LENGTH. The data-in (second) sub-list contains the remaining data segment descriptors from the entire list. The segments in the data-out sub-list, concatenated together, comprise the command's data-out buffer. The segments in the data-in sub-list, concatenated together, comprise the command's data-in buffer. The sum of the DATA LENGTH fields in the segment descriptors of each sub-list is the command's data-out or data-in byte count respectively. A zero DATA LENGTH field in a segment descriptor is valid and indicates that the segment descriptor shall not contribute to the command's data-out or data-in buffer.

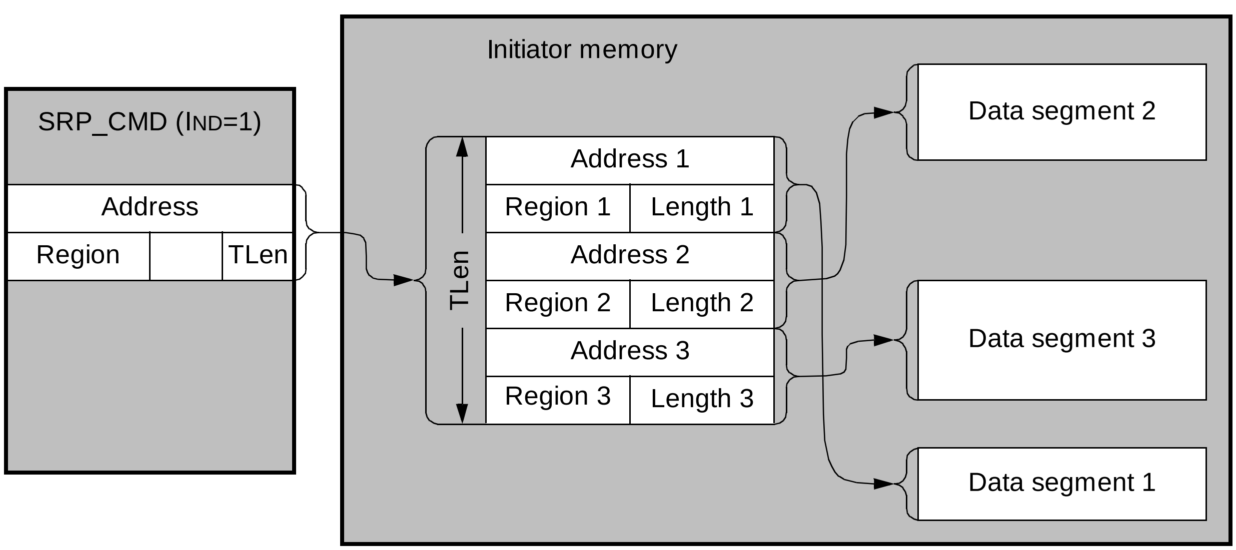

Figure 3 - Uni-directional Indirect Data Buffer Mapping

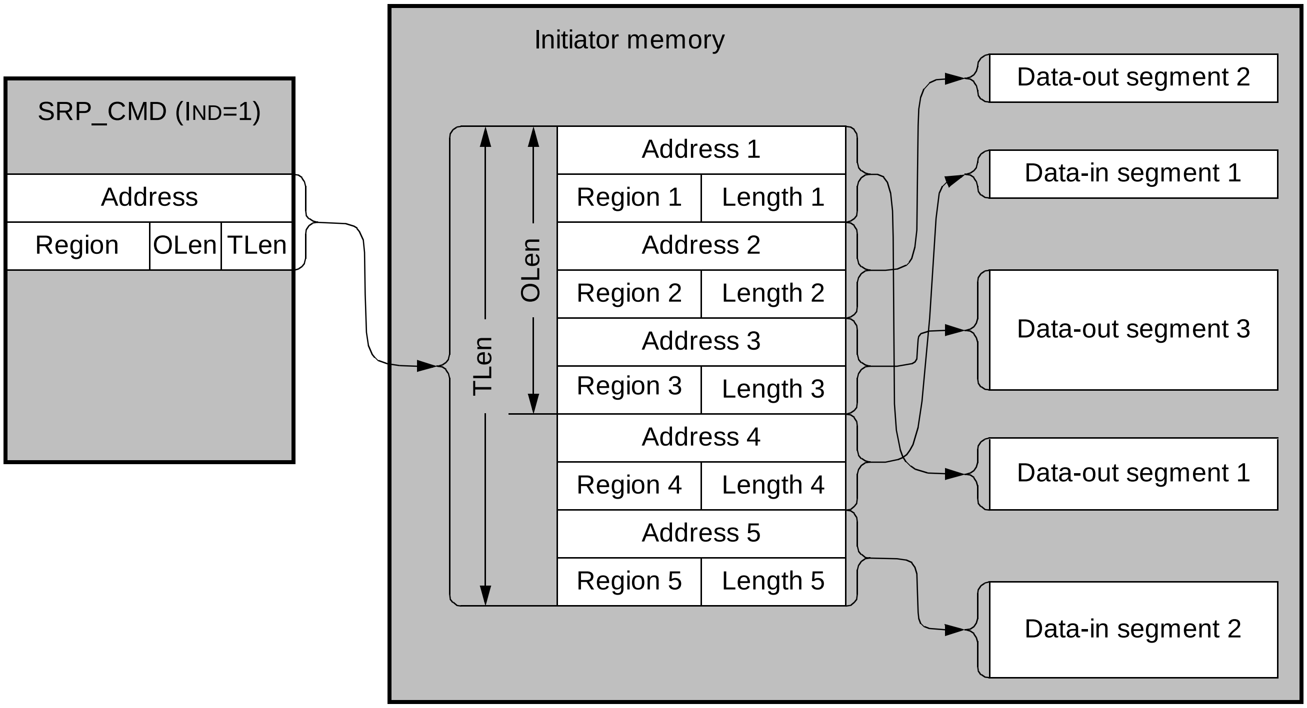

Figure 4 - Bi-directional Indirect Data Buffer Mapping