To: T10 Technical Committee
From: Steven Fairchild, HP (steve.Fairchild@hp.com)
Date: 11 October 2002
Subject: Expander Configuration Details

Revision r3:
- Incorporate comments from editor's meeting, 10/3 and 10/4
- Incorporated comments from call, 10/1/02

Revision r2:
- Added the definition of an end device.
- Changed port references to phy in route descriptions
- Changed topology figure

Revision r1:
- Changed to expander device set, rather than expander components building expander devices.
- Changed figures to follow common look and feel of SAS specification.
- Changes to DISCOVER, REPORT ROUTE INFORMATION and CONFIGURE ROUTE INFORMATION to correct typos and make consistent with change to expander device sets.

The purpose for this document is to define the topology addressing details for the edge and fanout expanders. Most of the wording is to clarify existing concepts, with the exception of a new section on configuration of expander route tables within edge and fanout devices.

Also, the current SAS specification imposes rules about the number of edge devices or fanout devices that may be connected to each other, implying that topologies will fail if the rules are broken. The recommendation is to not restrict the interconnection of any end, edge or fanout device. During the Discover process, when an initiator determines that an illegal topology has been created, it may take appropriate action. Disabling illegal links and reporting un-reachable SAS addresses as necessary.

Change 1: Definitions added to 3.1

configurable expander device: An expander device that contains an expander route table that is configured with expander route entries.

direct routing attribute: The attribute of an expander phy that allows it to be used by the expander connection manager to route a connection request to an end device.

direct routing method: The method the expander connection manager uses to establish a connection with an end device.

edge expander device set: A group of one or more edge expander devices (see 4.1.9.x).

end device: An initiator or target device that is not contained within an expander device.

expander connection manager: from Tim's proposal.

expander route entry: A single destination SAS address and a method to enable or disable the route entry within an expander route table.

expander route index: A value used in combination with the phy identifier to select a expander route entry within an expander route table.

expander route table: A table of expander route entries within an expander device. The table is used by the expander function to resolve connection requests (see 4.1.9.x).

subtractive routing attribute: The attribute of an edge expander phy that allows it to be used by the expander connection manager to route connection requests not resolved using the direct routing method or table routing method.

subtractive routing method: The method the expander connection manager uses to route connection requests not resolved using the direct routing method or table routing method.

table routing attribute: The attribute of an expander phy that allows it to be used by the expander connection manager to route connection requests using an expander route table.

table routing method: The method the expander connection manager uses to route connection requests using an expander route table.

Change 2: SMP Frame Changes

9.4.4.2 REPORT GENERAL function

Table 87 defines the response format.

Table 87. REPORT GENERAL response

Byte	7	6	5	4	3	2	1	0
0	SMP FRAME TYPE (41h)							
1	FUNCTION (00h)							
2	FUNCTION RESULT							
3	Reserved							
4	Reserved							
5								
6	(MSB) EXPANDER ROUTE INDEXES							
7								(LSB)
8	Reserved							
9	NUMBER OF PHYS							
10	Reserved							CONFIGURABLE ROUTE TABLE
11	Reserved							
12	(MSB) CRC							
15								(LSB)

The EXPANDER ROUTE INDEXES field contains the maximum number of route indexes for the expander device. Expander devices shall support this field. Other device types shall not support this field.

If an edge expander device supports an expander route table, then the number of expander route indexes for each phy shall be greater than or equal to the number of phys downstream from the edge expander device phy.

If a fanout expander device supports an expander route table, then the number of expander route indexes shall be 64 (see 4.1.9).

The NUMBER OF PHYS field contains the number of phys in the expander device.

The CONFIGURABLE ROUTE TABLE bit indicates the expander device has an expander route table that shall be configured. An expander device with a configurable route table shall have the CONFIGURABLE ROUTE TABLE bit set to one. An expander device without a configurable route table shall have the CONFIGURABLE ROUTE TABLE bit set to zero.

9.4.4.5 REPORT ROUTE INFORMATION function

The REPORT ROUTE INFORMATION function returns the expander route entry from the expander route table within an expander device. Expander devices shall support this function if the EXPANDER ROUTE INDEXES field is non-zero in the REPORT GENERAL function. This function is used primarily as a diagnostic tool to resolve topology issues.

Table 84 defines the request format.

Table 84. REPORT ROUTE INFORMATION request

Byte	7	6	5	4	3	2	1	0
0	SMP FRAME TYPE (40h)							
1	FUNCTION (04h)							
2	Reserved							
3								
4	Reserved							
5								
6	(MSB) EXPANDER ROUTE INDEX							
7								(LSB)
8	Reserved							
9	PHY IDENTIFER							
10	Ignored							
11	Reserved							
12	(MSB) CRC							
15								(LSB)

The EXPANDER ROUTE INDEX field indicates the expander route index for the route table entry being requested. If the value is not within the range of zero to EXPANDER ROUTE INDEXES (see 9.4.4.2) the expander device shall return an UNKNOWN FUNCTION function result.

The PHY IDENTIFIER field indicates the phy identifier for the route table entry being requested. If the value is not within the range of zero to NUMBER OF PHYS (see 9.4.4.2) the expander device shall return an UNKNOWN FUNCTION function result.

If the phy with the indicated PHY IDENTIFIER does not have the table routing attribute (see 4.x.x.x) the expander device shall return an INDEX DOES NOT EXIST function result for any EXPANDER ROUTE INDEX with the indicated PHY IDENTIFIER.

Table 85 defines the response format.

Table 85. REPORT ROUTE INFORMATION response

Byte	7	6	5	4	3	2	1	0
0	SMP FRAME TYPE (41h)							
1	FUNCTION (04h)							
2	FUNCTION RESULT							
3	Reserved							
4	Reserved							
5								
6	(MSB) EXPANDER ROUTE INDEX							
7								(LSB)
8	Reserved							
9	PHY IDENTIFIER							
10	Ignored							
11	Reserved							
12	ROUTE ENTRY DISABLED	Ignored						
13	Ignored							
15								
16	(MSB) ROUTED SAS ADDRESS							
23								(LSB)
24	Ignored							
35								
36	Reserved							
39								
40	(MSB) CRC							
43								(LSB)

The FUNCTION RESULT field is defined in 9.4.3. The FUNCTION RESULT field values specific to this function are defined in table 86.

Table 86 – Function results for REPORT ROUTE INFORMATION

Code	Meaning	Description
11h	INDEX DOES NOT EXIST	Expander route index does not exist; rest of data is invalid.
All others	Reserved	

The EXPANDER ROUTE INDEX field contains the expander route index for the expander route entry.

The PHY IDENTIFIER field contains the phy identifier for the expander route entry.

The ROUTE ENTRY DISABLED bit indicates whether the expander connection manager shall use the expander route entry to route connection requests. If the ROUTE ENTRY DISABLED bit is 0b, then the expander connection manager shall use the route table entry to route connection requests. A 1b value indicates the route table entry is in violation of connection rules and shall not be used by the expander connection manager to route connection requests.

The ROUTED SAS ADDRESS field contains the routed SAS address of the expander route entry.

9.4.4.6 DISCOVER function

The DISCOVER function returns the physical link configuration information for an expander device phy. The function response provides information from the IDENTIFY address frame received by the phy, as well as the routing attribute supported by the phy. This function shall be implemented by all expander devices and shall not be implemented by other types of devices.

Table 87 defines the request format.

Table 96. DISCOVER request

Byte	7	6	5	4	3	2	1	0
0	SMP FRAME TYPE (40h)							
1	FUNCTION (10h)							
2	Reserved							
3	Reserved							
4	Ignored							
7								
8	Reserved							
9	PHY IDENTIFIER							
10	Ignored							
11	Reserved							
40	(MSB)							
43	CRC							(LSB)

The PHY IDENTIFIER field indicates the phy identifier for the link configuration information being requested. If the value is not within the range of zero to NUMBER OF PHYS (see 9.4.4.2) the expander device shall return an UNKNOWN FUNCTION function result.

Table 88 defines the response format.

Table 88. DISCOVER response

Byte	7	6	5	4	3	2	1	0
0	SMP FRAME TYPE (41h)							
1	FUNCTION (10h)							
2	FUNCTION RESULT							
3	Reserved							
4	Ignored							
7								
8	Reserved							
9	PHY IDENTIFIER							
10	Ignored							
11	Reserved							
12	Ignored	ATTACHED DEVICE TYPE			ROUTING ATTRIBUTE			
13	Reserved				PHYSICAL LINK RATE			
14	Reserved				ATTACHED SSP INITIATOR	ATTACHED STP INITIATOR	ATTACHED SMP INITIATOR	Reserved
15	Reserved				ATTACHED SSP TARGET	ATTACHED STP TARGET	ATTACHED SMP TARGET	ATTACHED SATA TARGET
16	(MSB)							
23	ATTACHED SAS ADDRESS							(LSB)
24	(MSB)							
31	SAS ADDRESS							(LSB)
32	PROGRAMMED MINIMUM PHYSICAL LINK RATE				HARDWARE MINIMUM PHYSICAL LINK RATE			
33	PROGRAMMED MAXIMUM PHYSICAL LINK RATE				HARDWARE MAXIMUM PHYSICAL LINK RATE			
34	Vendor Specific							
35								
36	Reserved							
39								
40	(MSB)							
43	CRC							(LSB)

The FUNCTION RESULT field is defined in 9.4.3. There are no function specific field values.

The PHY IDENTIFIER field indicates the phy identifier for the physical configuration link information.

The ROUTING ATTRIBUTE field indicates the routing attribute and routing method supported by the phy and is defined in table 90.

Table 90. Routing attribute

Code	Routing attribute	Routing methods supported
0h	Direct routing	Direct routing for attached end devices
1h	Subtractive routing	Subtractive routing for attached expander devices, and direct routing for attached end devices
2h	Table routing	Table routing for attached edge expander devices, and direct routing for attached end devices
All others	Reserved	Reserved

The ROUTING ATTRIBUTE field shall not change based on the attached device type. The routing method used by the expander connection manager shall change based on the attached device type as described in Table 90.

The ATTACHED DEVICE TYPE field indicates the type of device attached to the phy (see 7.7.2), and is defined in table 48. If no device is attached to the phy then a zero value will be returned.

The PHYSICAL LINK RATE field indicates the physical link rate for the phy negotiated during the link reset sequence and is defined in table 91. The physical link rate may be outside the programmed minimum physical link rate and programmed maximum physical link rate if they have been changed since the link reset sequence.

Table 91. Physical Link Rate

PHYSICAL LINK RATE	Physical link rate
0h	Reserved [a]
1h	Phy does not exist
2h	Disabled
3h	Failed
4h	Spinup hold OOB
5h	1,5 Gbps
6h	3,0 Gbps
7h – Fh	Reserved
a For use by application client when rate is undefined.	

The ATTACHED STP INITIATOR bit indicates the STP INITIATOR value received during the link reset sequence.
The ATTACHED STP TARGET bit indicates the STP TARGET value received during the link reset sequence.
The ATTACHED SSP INITIATOR bit indicates the SSP INITIATOR value received during the link reset sequence.
The ATTACHED SSP TARGET bit indicates the SSP TARGET value received during the link reset sequence.
The ATTACHED SMP INITIATOR bit indicates the SMP INITIATOR value received during the link reset sequence.
The ATTACHED SMP TARGET bit indicates the SMP TARGET value received during the link reset sequence.
The ATTACHED SATA TARGET bit indicates a SATA target is attached.
The ATTACHED SAS ADDRESS field contains the SAS address of the attached device port. If no device is attached, then a zero value shall be returned.
The SAS ADDRESS field contains the SAS address of this device port.
The HARDWARE MAXIMUM PHYSICAL LINK RATE field indicates the maximum physical link rate supported by the phy (see Table 92). The value shall be greater than or equal to the value of the HARDWARE MINIMUM PHYSICAL LINK RATE field.

Table 92. Maximum or minimum physical link rate

Maximum or minimum physical link rate	Physical link rate
0h – 4h	Reserved
5h	1,5 Gbps
6h	3,0 Gbps
7h – Fh	Reserved

The PROGRAMMED MAXIMUM PHYSICAL LINK RATE field indicates the maximum physical link rate (see Table 91) set by the PHY CONTROL function. The value shall not exceed the HARDWARE MAXIMUM PHYSICAL LINK RATE.

The HARDWARE MINIMUM PHYSICAL LINK RATE field indicates the minimum physical link rate supported by the phy (see Table 92). The value shall be less than or equal to the value of the HARDWARE MAXIMUM PHYSICAL LINK RATE field.

The PROGRAMMED MINIMUM PHYSICAL LINK RATE field indicates the minimum physical link rate (see Table 91) set by the PHY CONTROL function. The value shall not exceed the HARDWARE MAXIMUM PHYSICAL LINK RATE.

9.4.4.9 CONFIGURE ROUTE INFORMATION function

The CONFIGURE ROUTE INFORMATION function sets the expander route entry for the expander route table within an expander device. Expander devices that do not have a configurable route table or end devices shall not support this function. Expander devices shall support this function if the CONFIGURABLE ROUTE TABLE field is set in the REPORT GENERAL function.

Table 100 defines the request format.

Table 100. CONFIGURE ROUTE INFORMATION request

Byte	7	6	5	4	3	2	1	0
0	SMP FRAME TYPE (40h)							
1	FUNCTION (80h)							
2	Reserved							
3								
4	Reserved							
5								
6	(MSB) EXPANDER ROUTE INDEX							
7								(LSB)
8	Reserved							
9	PHY IDENTIFIER							
10	Reserved							
11	Reserved							
12	DISABLE ROUTE ENTRY	Ignored						
13	Ignored							
15								
16	(MSB) ROUTED SAS ADDRESS							
23								(LSB)
24	Ignored							
35								
36	Reserved							
39								
40	(MSB) CRC							
43								(LSB)

The EXPANDER ROUTE INDEX field indicates the expander route index for the route table entry being configured. If the value is not within the range of zero to EXPANDER ROUTE INDEXES (see 9.4.4.2) the expander device shall return an UNKNOWN FUNCTION function result.

The PHY IDENTIFIER field indicates the phy identifier for the route table entry being configured. If the value is not within the range of zero to NUMBER OF PHYS (see 9.4.4.2) the expander device shall return an UNKNOWN FUNCTION function result.

If the phy with the indicated PHY IDENTIFIER does not have the table routing attribute (see 4.x.x.x) the expander device shall return an INDEX DOES NOT EXIST function result for any EXPANDER ROUTE INDEX with the indicated PHY IDENTIFIER.

The DISABLE ROUTE ENTRY bit indicates whether the expander connection manager shall use the expander route entry to route connection requests. If the ROUTE ENTRY DISABLED bit is 0b, then the expander connection manager shall use the route table entry to route connection requests. A 1b value indicates the route table entry is in violation of connection rules and shall not be used by the expander connection manager to route connection requests.

The ROUTED SAS ADDRESS field contains the routed SAS address for the expander route entry being configured.

Table 101 defines the response format.

Table 101. CONFIGURE ROUTE INFORMATION response

Byte	7	6	5	4	3	2	1	0
0	SMP FRAME TYPE (41h)							
1	FUNCTION (80h)							
2	FUNCTION RESULT							
3	Reserved							
4	(MSB)							
7				CRC				(LSB)

The FUNCTION RESULT field is defined in 9.4.3. The FUNCTION RESULT field values specific to this function are defined in table 102.

Table 102 – Function results for CONFIGURE ROUTE INFORMATION

Code	Meaning	Description
11h	INDEX DOES NOT EXIST	Expander route index does not exist; rest of data is invalid.
All others	Reserved	

Change 3: Expander devices

4.1.7 Expander devices

Expander devices are part of the service delivery subsystem. Expander devices shall contain two or more external expander ports. Expander devices may also include initiator ports and target ports (e.g., an expander device may include an embedded SCSI enclosure services target port) attached to internal expander ports. Expander ports may support being attached to SAS initiator ports, SAS and/or SATA target ports, and to other expander ports.
Figure 10 shows an expander device and the type of ports it may support.

<figure 10 from sas spec, expander device>

There are two types of expander devices differentiated by the routing attributes of their phys, edge expander devices and fanout expander devices. Edge expander devices may contain phys with the subtractive routing attribute. A fanout expander device shall not contain phys with the subtractive routing attribute. Edge expander devices and the fanout expander device may also contain phys with the direct routing and table routing attributes.

4.1.7.1 Edge expander device set

One or more edge expander devices may be grouped into edge expander device sets. The edge expander device sets are bounded by the direct routing phys that are either attached to end devices or not attached to any device. Edge expander device sets are further bounded by the phys supporting subtractive routing that are:

a) attached to the phys of a fanout device;
b) attached to the phys supporting subtractive routing on another edge expander device set;
c) attached to an end device; or,
d) unattached.

The phys that support table routing within an edge expander device provide attachments to other edge expander devices in the edge expander device set.

The number of devices attached to an edge expander device set may not exceed 64. (see 4.1.9).

Figure X shows an edge expander device set.

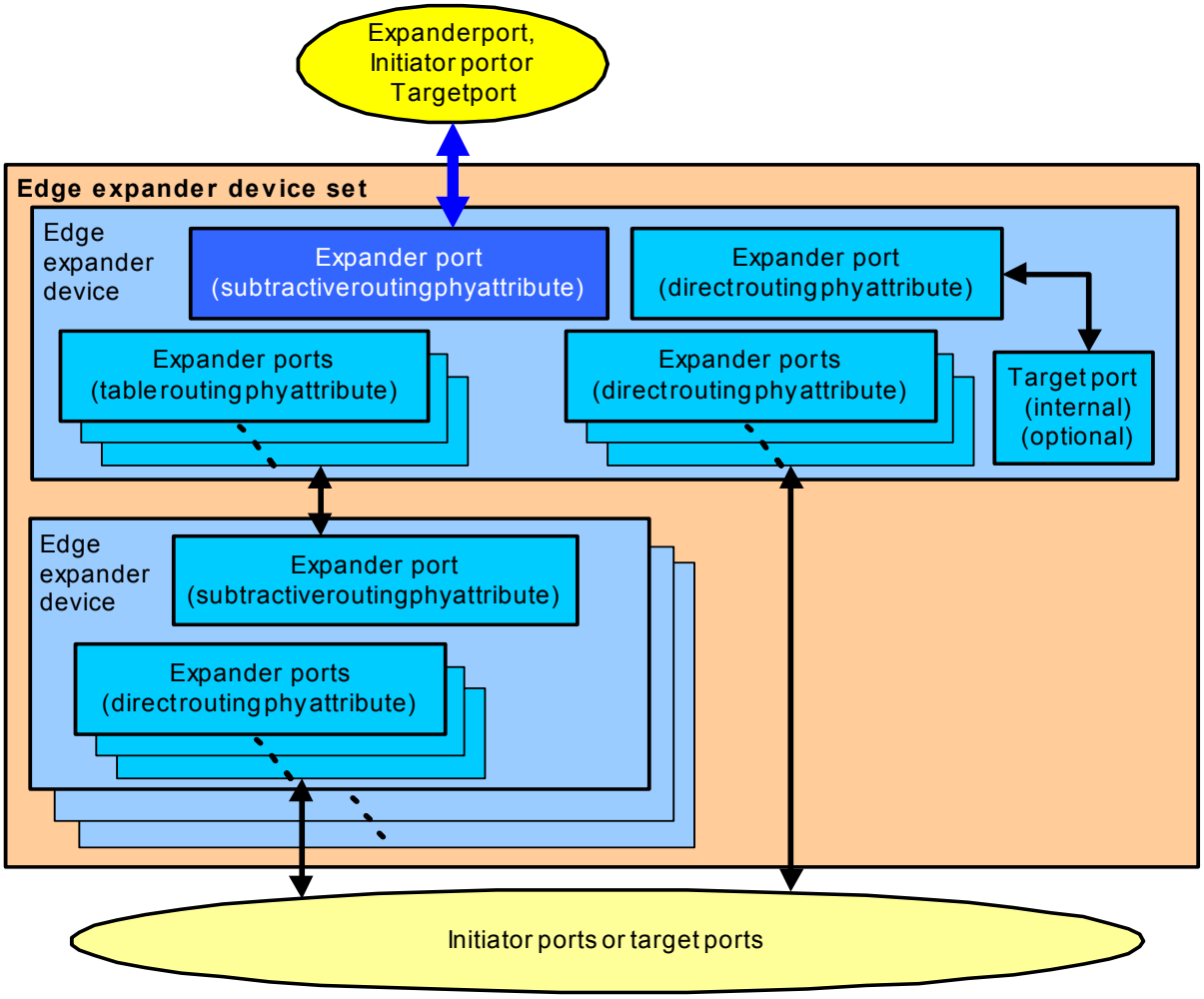

Figure X – Edge expander device set

4.1.7.2 Configurable expander device

Expander devices may have a configurable route table. Expander devices with a configurable route table depend on the application client within one or more initiator devices to use the discover process (see 4.1.9.2) to configure the expander route table (see 4.1.9.1). The expander route table is used by the expander connection manager to route connection requests to expander phys with the table routing attribute.

Change 4: Expander topologies

4.1.9 Expander topologies

The SAS domain consists of initiator devices, target devices, and expander devices.

No more than one fanout expander device shall be included in a SAS domain. The fanout expander device may be attached to up to 64 edge expander device sets, initiator devices, or target devices.

Each edge expander device set shall contain no more than 64 physical links to edge expander devices, initiator devices, or target devices. Each edge expander device set shall not be attached to more than one fanout expander device. An edge expander device set may be attached to one other edge expander device set if that is the only other edge expander device set in the SAS domain and there are no fanout expanders in the SAS domain.

The number of edge expander devices and the phy route attributes of edge expander devices within an edge expander device set shall be established when the edge expander device set is configured.

Figure 13 shows a SAS domain with the maximum number of expander devices.

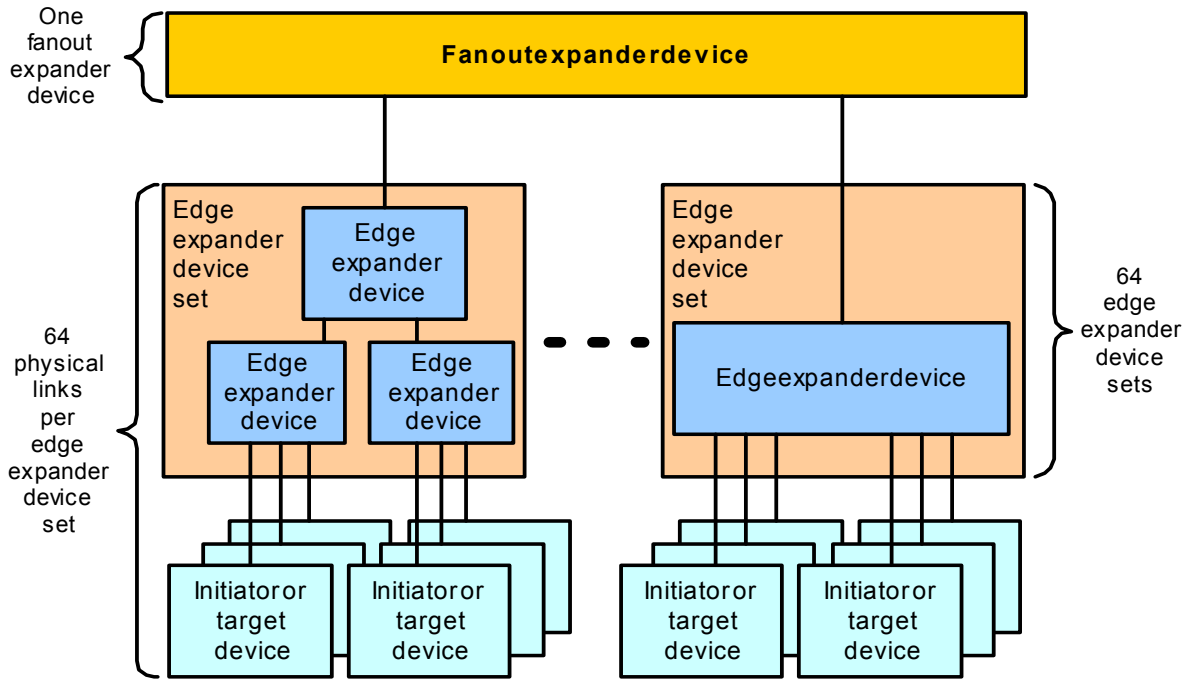

Figure 13 – Maximum expander topology

Initiator devices and target devices may be attached directly to the fanout expander device, as shown in figure 14.

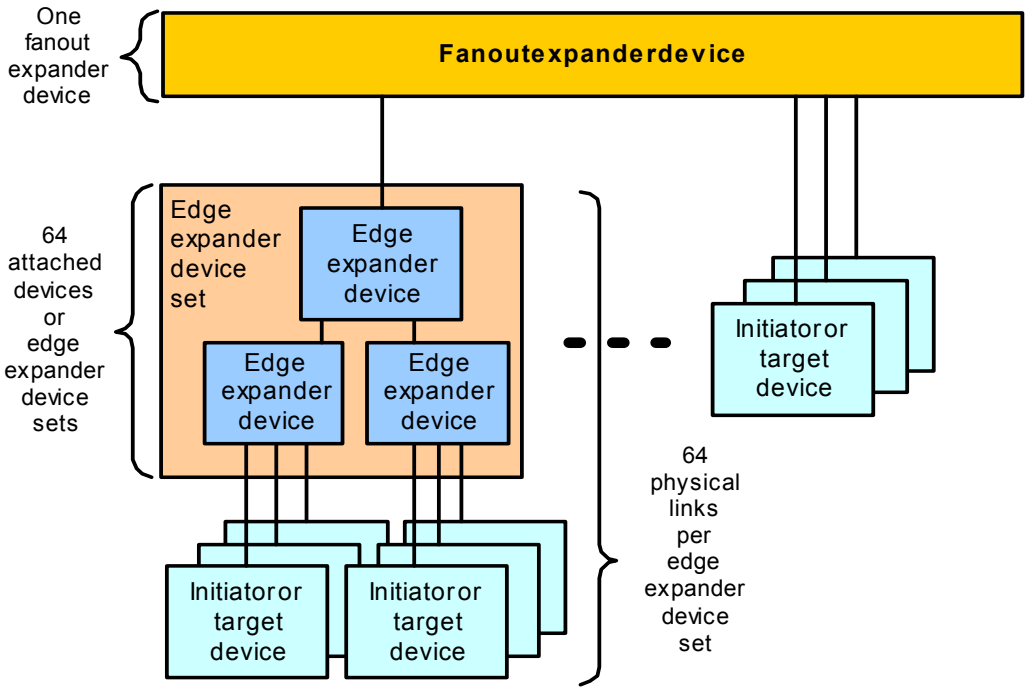

Figure 14 – Fanout expander topology

Initiator devices and target devices may be attached to any of the edge expander devices within an edge expander device set and at most, two edge expander device sets may be attached together without a fanout expander, as shown in figure 16.

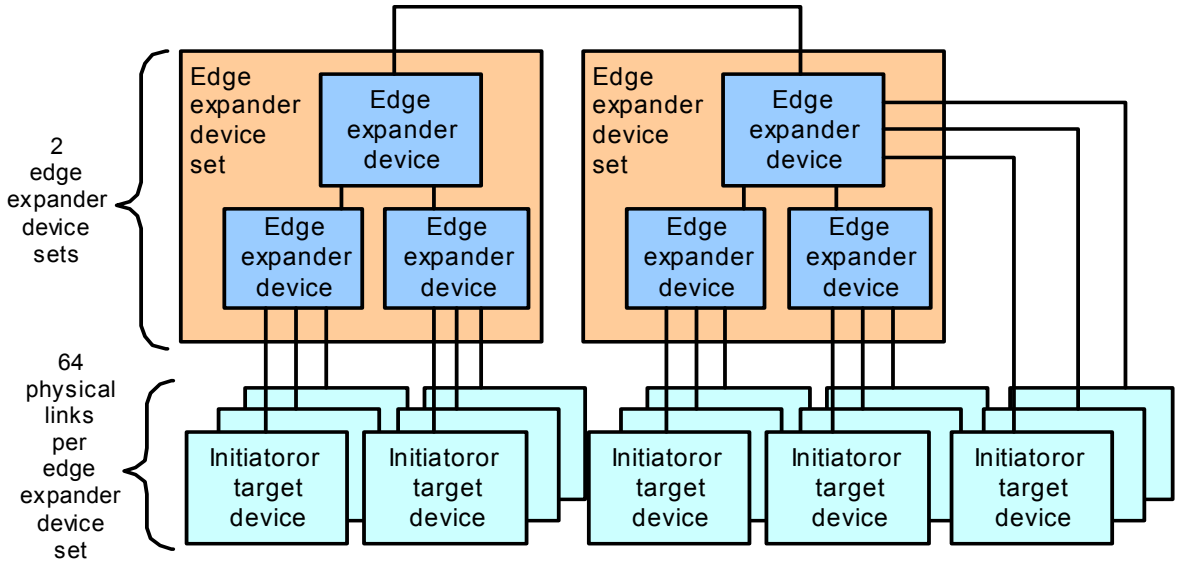

Figure 15 – Edge expander device set topology

4.1.9.1 Expander device connection request routing

Each expander phy in an expander device shall support one of the following routing attributes:

a) direct routing attribute;
b) table routing; or
c) subtractive routing.

The routing attributes allow the expander connection manager to determine which routing method is used to route connection requests to the expander phy.

A phy that has the direct routing attribute allows the expander connection manager to use the direct routing method. The direct route method routes connection requests to attached end devices.

A phy that has the table routing attribute allows the expander connection manager to use the table routing method, or the direct routing method to route connection requests. The table routing method routes connection requests to attached expander devices using an expander route table. An expander device may have zero or more phys with the table routing attribute.

A phy that has the subtractive routing attribute allows the expander connection manager to use the subtractive routing method, or the direct routing method to route connection requests. The subtractive routing method routes unresolved connection requests to an attached expander device. An edge expander device shall have at most one defined port with phys that have the subtractive routing attribute. A fanout expander phy shall not have the subtractive routing attribute.

An edge expander device may only use phys with the table routing attribute to attach to phys with the subtractive routing attribute in other edge expander devices within an edge expander device set.

If multiple phys within an expander device have subtractive routing attributes and are attached to expander devices, they shall attach to phys with identical SAS addresses (i.e., the same expander port).

If multiple phys within an expander device have subtractive routing attributes and are attached to expander devices that do not have identical SAS addresses, the application client that is processing the discover process (see 4.1.9.2) shall report an error in a vendor-specific manner.

The expander connection manager of an expander device containing phys that have different route attributes shall determine how to route a connection request using the following precedence:

1) route to a phy with a direct routing attribute when the destination SAS address in the connection request matches the attached SAS address;
2) route to a phy with a table routing attribute when the destination SAS address in the connection request matches the attached SAS address in the expander route entry and the disable route entry bit in the expander route entry is set to zero;
3) route to a phy with a subtractive routing attribute; or
4) return an OPEN-REJECT (NO DESTINATION) to the connection request source SAS address.

If the destination SAS address of a connection request matches the attached SAS address of an expander route entry and the DISABLE ROUTE ENTRY bit in the expander route entry is set to one, then the expander connection manager shall ignore the expander route entry.

The discover process shall allow the following attachments between expander phys:

a) edge expander phy with subtractive routing attribute attached to an edge expander phy with subtractive routing attribute;
b) edge expander phy with subtractive routing attribute attached to an edge expander phy with table routing attribute; or
c) edge expander phy with subtractive routing attribute attached to a fanout expander phy with table routing attribute.

During the discover process (see 4.1.9.2) if an application client in an initiator device detects an illegal expander phy attachment, it shall report an error in a vendor-specific manner.

4.1.9.1.1 Expander route table

An expander device that supports the table routing method (see 4.1.9.1) shall contain an expander route table. The expander route table is a structure that provides an association between destination SAS addresses and expander phy identifiers. Each association represents an expander route entry.

An expander device indicates its ability to maintain the expander route entries and define the size of the expander route table (see 9.4.4.2).

An initiator device may reference a specific expander route entry within an expander route table (see 9.4.4.5 and 9.4.4.9).

Figure X shows a representation of an expander route table.

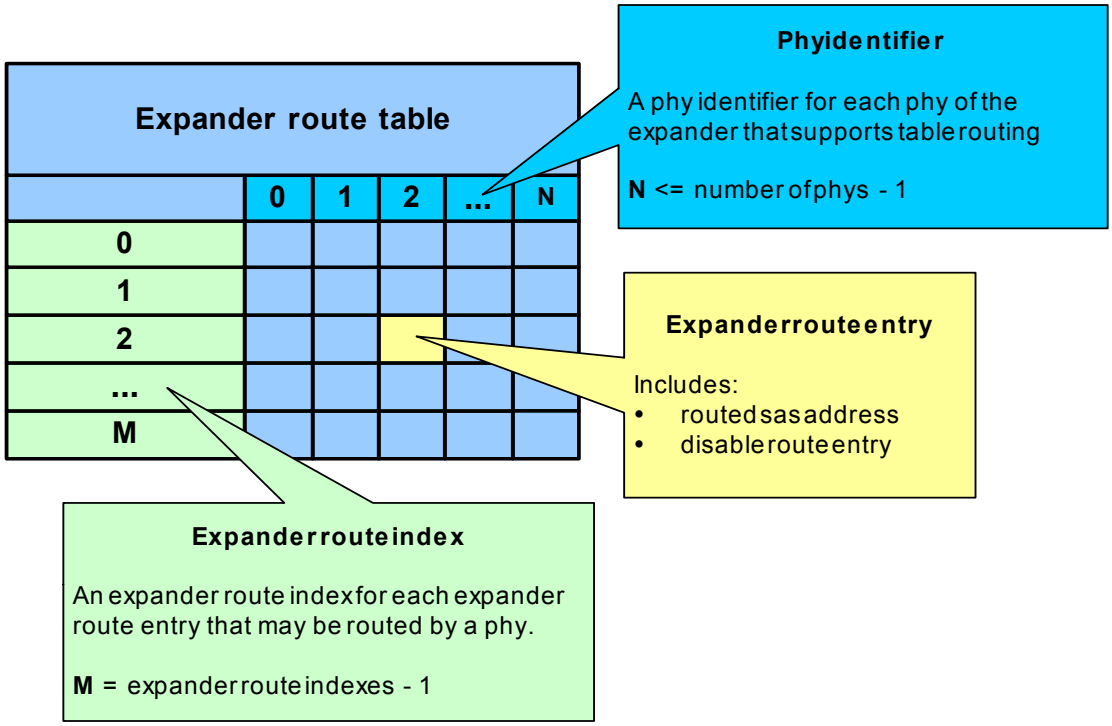

Figure X – Expander route table example

The number of end devices that may be attached to an edge expander device set is dependent on the number of expander route entries in the expander route table of the edge expander devices.

To avoid an overflow of an edge expander route index during the discover process (see 4.1.9.2) the number of edge expander route indexes per phy identifier shall be greater than or equal to the sum of all cascaded edge expander device phys addressable through the edge expander device phy.

An expander device is considered cascaded to another expander device when the expander device table routing phy is attached to the subtractive routing phy of another edge expander device.

Fanout expander devices may not be cascaded.

During the discover process (see 4.1.9.2) if an application client in an initiator device detects an overflow of the edge expander route index, it shall report an error in a vendor-specific manner.

An expander route table entry that references the SAS address of the expander itself (self-reference) shall have the DISABLE ROUTE ENTRY bit set in the expander route table entry.

4.1.9.1.2 Expander route index order

The recursive table AA defines the expander route index order of routed SAS addresses for an expander phy with the table routing attribute and cascaded edge expander devices.

Table AA Routed SAS addresses for a phy.

Expander Route Index	Routed SAS Address Description
0	Routed SAS address of the device attached to phy 0 of the attached edge expander device.
1	Routed SAS address of the device attached to phy 1 of the attached edge expander device.
2	Routed SAS address of the device attached to phy 2 of the attached edge expander device.
N	Routed SAS address of the device attached to phy N of the attached edge expander device.
Routed SAS addresses for phys with the table routing attribute and a cascaded edge expander device.	
N + 1	Routed SAS addresses for the first phy of the attached edge expander with the table routing attribute and a cascaded edge expander device.
…	
n	Routed SAS addresses for the last phy of the attached edge expander with the table routing attribute and a cascaded edge expander device.

As an example, figure Y is used in association with Table Y to show the expander route index order for the edge expander, E0.

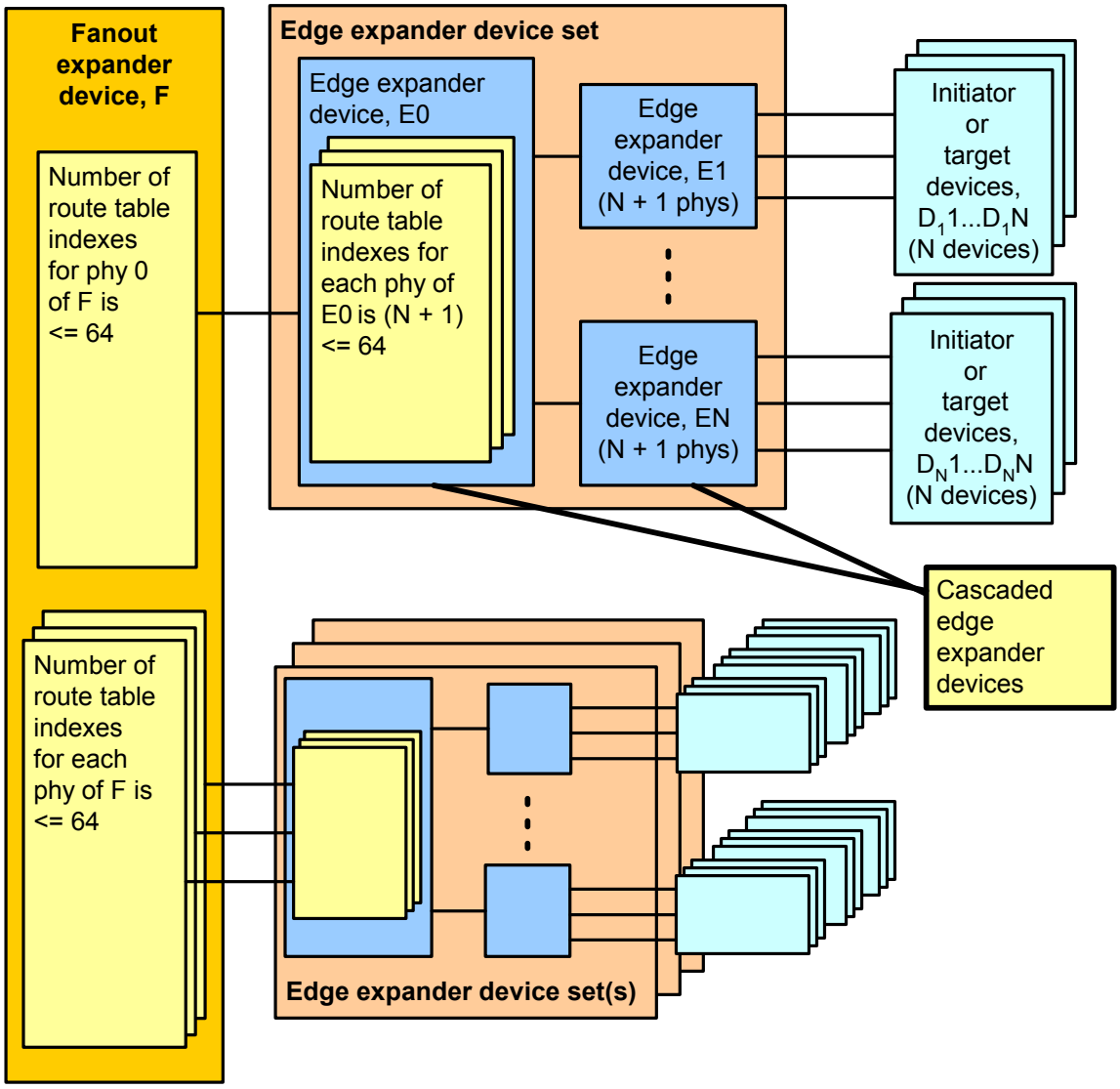

Figure Y – Expander route index order

Table Y – Expander route indexes for phy 0 of edge expander E0.

Expander Route Index	Routed SAS Address Description
0	Routed SAS address, (e.g., E0) of the device attached to phy 0 of the attached edge expander, (e.g., E1). In figure Y the DISABLE ROUTE ENTRY bit is set for this entry; because, E0 is a self-reference.
1	Routed SAS address, (e.g., $D_1$1) of the device attached to phy 1 of the attached edge expander, (e.g., E1).
2	Routed SAS address, (e.g., $D_1$2) of the device attached to phy 2 of the attached edge expander, (e.g., E1).
N	Routed SAS address, (e.g., D_1N) of the device attached to phy N of the attached edge expander, (e.g., E1).

Table Z in association with Figure Y shows the expander route index order for the fanout expander F.

Table Z – Fanout expander, F expander route indexes for phy 0

Expander Route Index	Routed SAS Address Description
0	Routed SAS address, (e.g., F) of the device attached to phy 0 of the attached edge expander, (e.g., E0). In figure Y the DISABLE ROUTE ENTRY bit is set for this entry; because, F is a self-reference.
1	Routed SAS address, (e.g., E1) of the device attached to phy 1 of the attached edge expander, (e.g., E0).
2	Routed SAS address, (e.g., E2) of the device attached to phy 2 of the attached edge expander, (e.g., E0).
N	Routed SAS address, (e.g., EN) of the device attached to phy N of the attached edge expander, (e.g., E0).
Routed SAS addresses for phys with table routing attribute and a cascaded expander device.	
N + 1	Routed SAS addresses for an edge expander, (e.g., EN). This edge expander (e.g., EN) is in turn attached to a cascaded edge expander, (e.g., E0) through phy 0.
n	Routed SAS addresses for an edge expander, (e.g., EN). This edge expander (e.g., EN) is in turn attached to a cascaded edge expander, (e.g., E0) through phy N.

4.1.9.2 Discover process

The discover process shall perform a traversal of the SAS domain to identify the initiator devices, target devices and expander devices. The order of traversal shall be to discover:

1) the expander device to which the initiator port is attached;
2) every device attached to that expander device; and
3) as each expander device is found, every device attached to that expander device.

The discover process is repeated until all expander devices have been traversed.

The discover process begins with the application client of an initiator device determining that an expander device is attached (see 9.4.4.x).

If either an edge device or fanout device is attached, then the discover process determines if the expander device requires configuration (see 9.4.4.x). If the expander device requires configuration, then expander route entries are configured with the SAS addresses of the devices routed to the phys with the table routing attribute. The order of the expander route entries is described in 4.1.9.1.2.

The result of the discover process is that the application client has the necessary information to communicate with each device in the SAS domain and each configurable expander device is configured with the expander route entries to allow routing of connection requests through the SAS domain.

An example algorithm used to perform the discover process is provided in Annex X.

Change 5: Discover rules

7.6.2 Initiator device specific rules

After a link reset sequence, or after receiving a CHANGE primitive sequence, the application client within an initiator device shall perform a discover process (see 4.1.9.2).

When this is done after a link reset sequence, this allows the application client within an initiator device to discover all the devices in the SAS domain. When this is done after a CHANGE, this allows the application client within an initiator device to determine what has changed in the SAS domain.

The discover information may be used to select link rates for connection requests.

If during the discover process (4.1.9.2) the application client within the initiator device detects a port with a SAS address it has already encountered, it has found a routing loop. It shall disable routing of destination SAS addresses through the expander port used to reach this previously encountered port to break the loop. The DISABLE EXPANDER ROUTE field in a SMP CONFIGURE ROUTE INFORMATION function request is used to disable the expander port of an expander device.

7.6.3 Fanout expander device specific rules

After completing the identify sequence, the expander connection manager within a fanout expander device shall be capable of routing connection requests to attached devices.

After a link reset sequence, or after receiving a CHANGE primitive sequence, the application client within a fanout expander device that does not have a configurable expander route table shall follow the initiator device specific rules (see 7.6.2) to perform a discover process.

The expander connection manager of a fanout expander device that has a configurable expander route table is dependent on the completion of the discover process (see 4.1.9.2) for routing connection requests using the table routing method.

7.6.4 Edge expander device specific rules

After completing the identify sequence, the expander connection manager within an edge expander device shall be capable of routing connection requests to attached devices.

The expander connection manager of an edge expander device that has a configurable expander route table is dependent on the completion of the discover process (see 4.1.9.2) for routing connection requests using the table routing method.

Change 6: Domain changes

7.9 Domain changes

SAS initiator ports scan the domain during a discover process (see 4.1.9.2) to search for initiator devices, expander devices and target devices after power on or receiving a CHANGE primitive sequence.

The CHANGE primitive sequence shall only be sent outside of a connection. The expander device shall transmit CHANGE to one physical link attached to each expander port. The expander device should not transmit CHANGE to more than one physical link per expander port.

CHANGE shall not be transmitted to any phy that is part of the expander port that is the cause for sending CHANGE.

Expander devices shall transmit CHANGE for the following reasons:

a) after an expander phy has lost bit sychronization;

b) after the link reset sequence completes; and

c) after the expander device receives CHANGE.

CHANGE shall only be sent outside of connections. CHANGE may be sent by initiator ports to force expander devices to exchange SAS addresses, but should not be sent by target ports.

An expander device is not required to queue multiple CHANGE indications for the same expander port. If a second CHANGE indication is requested before the first indication has been transmitted, the second indication may be dropped.

An edge expander device that detects CHANGE shall follow the edge device specific rules (see 7.6.4).

A fanout expander device that detects CHANGE shall follow the fanout device specific rules (see 7.6.3) to discover the topology.

An initiator port that detects CHANGE shall follow the initiator device specific rules (see 7.6.2) to discover the topology.

Change 7: Connections

7.12.4.2 Edge expander devices

When an edge expander device receives a connection request, it compares the destination SAS address to the SAS addresses of the devices to which each of its phys is attached.

If they match, then the expander device shall arbitrate for access to one of the matching physical links and pass along the connection request.

If they do not match, but an expander device is attached and the request did not come from that expander device, the connection request shall be forwarded to the expander device.

If they do not match, and no expander device is attached or an expander device is attached and the request came from that expander device, the edge expander device shall reply with OPEN_REJECT (NO DESTINATION).

If the output port indicated by the routing table (see table 55) matches the input port, it shall reply with OPEN_REJECT (BAD DESTINATION).

When two edge expander devices are attached, this means requests to non-existent devices return OPEN_REJECT (BAD DESTINATION) rather than OPEN_REJECT (NO DESTINATION). When a fanout expander device is involved, an OPEN_REJECT (NO DESTINATION) is sent.

Table 55 shows the routing table maintained by an edge expander device.

<table in SAS spec>

7.12.4.3 Fanout expander devices

When a fanout expander device receives a connection request, it compares the destination SAS address to the SAS addresses of the devices to which each of its phys is attached. For each port, which is attached to an edge expander, it also compares the SAS addresses to which edge expander device reported being attached.

If it finds a match, it shall arbitrate for access to one of the matching physical links and pass along the connection request.

If it does not find a match, it shall reply with OPEN_REJECT (NO DESTINATION). If the output port indicated by the routing table matches the input port, it shall reply with OPEN_REJECT (BAD DESTINATION).

Table 56 shows the routing table maintained by a fanout expander device.

<table in SAS spec>

Change 8: Example discover/configure code

ANNEX X – Example discover process algorithm (Informative)

SASDiscoverSimulation.h

```
// assume the maximum number of phys in an expander device is 16
#define MAXIMUM_EXPANDER_PHYS          16
#define MAXIMUM_EXPANDER_INDEXES       64
#define MAXIMUM_INITIATORS             8

// see 9.4.1
#define SMP_REQUEST_FRAME              0x40
#define SMP_RESPONSE_FRAME             0x41

#define REPORT_GENERAL                 0x00
#define DISCOVER                       0x10
#define CONFIGURE_ROUTE_INFORMATION    0x80

#define SMP_REQUEST_ACCEPTED           0x00
#define SMP_UNKNOWN_FUNCTION           0x01
#define SMP_REQUEST_FAILED             0x02

// DeviceTypes
enum DeviceTypes
{
   END = 0,
   EDGE,
   FANOUT
};

// RoutingAttribute
enum RoutingAttribute
{
   DIRECT = 0,
   SUBTRACTIVE,
   TABLE
};

// RouteFlag
enum DisableRouteEntry
{
   ENABLED = 0,
   DISABLED
};

// PhyLinkRate(s)
enum PhysicalLinkRate
{
   RATE_UNKNOWN = 0,
   PHY_DOES_NOT_EXIST,
   PHY_DISABLED,
   PHY_FAILED,
   SPINUP_HOLD_OOB,
   GBPS_1_5,
   GBPS_3_0
};
```

```
// provide the simple type definitions
typedef unsigned char byte;
typedef unsigned word;
typedef unsigned long dword;
typedef unsigned _int64 quadword;

// defines for the protocol bits
#define SATA   0x01
#define SMP    0x02
#define STP    0x04
#define SSP    0x08

// the structures assume a char bitfield is valid, this is compiler dependent
// defines would be more portable, but less descriptive

// the Identify frame is exchanged following OOB, for this
// code it contains the identity information for the attached device
// and the initiator application client
struct Identify
{
   // byte 0
   byte AddressFrame:4;       // for an identify frame the value is 0
   byte DeviceType:3;
   byte IgnoredByte0Bit7:1;

   // byte 1
   byte IgnoredByte1;

   // byte 2
   union
   {
      struct
      {
         byte ReservedByte2Bit0:1;
         byte SMPInitiator:1;
         byte STPInitiator:1;
         byte SSPInitiator:1;
         byte ReservedByte2Bit4_7:4;
      };
      byte InitiatorBits;
   };

   // byte 3
   union
   {
      struct
      {
         byte IgnoredByte3Bit0:1;
         byte SMPTarget:1;
         byte STPTarget:1;
         byte SSPTarget:1;
         byte ReservedByte3Bit4_7:4;
      };
      byte TargetBits;
   };

   // byte 4-11
   byte IgnoredByte4_11[8];

   // byte 12-19
   quadword SASAddress;
```

```
   // byte 20-21
   byte IgnoredByte20_21[2];

   // byte 22-27
   byte ReservedByte22_27[8];

   // byte 28-31
   dword CRC[4];
};

// request specific bytes for SMP Report General function
struct SMPRequestReportGeneral
{
   // byte 4-7
   byte CRC[4];
};

// request specific bytes for SMP Discover function
struct SMPRequestDiscover
{
   // byte 4-7
   byte IgnoredByte4_7[4];

   // byte 8
   byte ReservedByte8;

   // byte 9
   byte PhyIdentifier;

   // byte 10
   byte IgnoredByte10;

   // byte 11
   byte ReservedByte11;

   // byte 12-15
   byte CRC[4];
};

// the ConfigureRouteInformation structure is used to provide the
// expander route entry for the expander route table, it is intended
// to be referenced by SMP RequestConfigureRouteInformation
struct ConfigureRouteInformation
{
   // byte 12
   byte IgnoredByte12Bit0_6:7;
   byte DisableRouteEntry:1;           // if a routing error is detected
                                       // then the route is disabled by
                                       // setting this bit

   // byte 13-15
   byte IgnoredByte13_15[3];

   // byte 16-23
   quadword RoutedSASAddress;          // identical to the AttachedSASAddress
                                       // found through discovery

   // byte 24-35
   byte IgnoredByte24_35[12];
```

```
   // byte 36-39
   byte ReservedByte36_39[4];

   // byte 40-43
   dword CRC[4];
};

// request specific bytes for SMP ConfigureRouteInformation function
struct SMPRequestConfigureRouteInformation
{
   // byte 4-7
   byte ReservedByte4_5[2];

   // byte 6-7
   word ExpanderRouteIndex;

   // byte 8
   byte ReservedByte8;

   // byte 9
   byte PhyIdentifier;

   // byte 10
   byte IgnoredByte10;

   // byte 11
   byte ReservedByte11;

   // byte 12-15
   struct ConfigureRouteInformation Configure;
};

// generic structure referencing an SMP Request, is initialized
// before being used
struct SMPRequest
{
   // byte 0
   byte SMPFrameType;                  // always 40h for SMP Frame request

   // byte 1
   byte Function;                      // 00h ReportGeneral
                                       // 10h Discover
                                       // 80h ConfigureRouteInformation

   // byte 2-3
   byte ReservedByte2_3[2];

   // bytes 4-n
   union
   {
      struct SMPRequestDiscover Discover;
      struct SMPRequestReportGeneral ReportGeneral;
      struct SMPRequestConfigureRouteInformation ConfigureRouteInformation;

// remaining structures would be necessary for a complete implementation
// for this example however they are comments only
//
//      struct SMPRequestSATACapabilities SATACapabilities;
//      struct SMPRequestReportManfacturerInformation
//             ManufacturerInformation;
//      struct SMPRequestReportRouteInformation RouteInformation;
```

```
//      struct SMPRequestReportPhyErrorLog PhyErrorLog;
//      struct SMPRequestReportPhySATA PhySATA;
//      struct SMPRequestPhyControl PhyControl;
//      struct SMPRequestPhyMarginControl PhyMarginControl;

   } Request;
};

// request specific bytes for SMP Report General response, intended to be
// referenced by SMPResponse
struct SMPResponseReportGeneral
{
   // byte 4-5
   byte ReservedByte4_5;

   // byte 6-7
   word ExpanderRouteIndexes;

   // byte 8
   byte ReservedByte8;

   // byte 9
   byte NumberOfPhys;

   // byte 10
   byte ConfigurableRouteTable:1;
   byte ReservedByte10Bit1_7:7;

   // byte 11
   byte ReservedByte11;

   // byte 12-15
   byte CRC[4];
};


// the Discover structure is used to retrieve expander port information
// it is intended to be referenced by the SMPResponseDiscover structure
struct Discover
{
   // byte 12
   byte RoutingAttribute:4;
   byte AttachedDeviceType:3;
   byte RouteEntryDisabled:1;

   // byte 13
   byte PhysicalLinkRate:4;
   byte ReservedByte13Bit4_7:4;

   // byte 14
   union
   {
      struct
      {
         byte ReservedByte2Bit0:1;
         byte AttachedSMPInitiator:1;
         byte AttachedSTPInitiator:1;
         byte AttachedSSPInitiator:1;
         byte ReservedByte14Bit4_7:4;
      };
      byte InitiatorBits;
```

```
   };

   // byte 15
   union
   {
      struct
      {
         byte AttachedSATATarget:1;
         byte AttachedSMPTarget:1;
         byte AttachedSTPTarget:1;
         byte AttachedSSPTarget:1;
         byte ReservedByte15Bit4_7:4;
      };
      byte TargetBits;
   };

   // byte 16-23
   quadword AttachedSASAddress;

   // byte 24-31
   quadword SASAddress;

   // byte 32
   byte HardwareMinimumPhysicalLinkRate:4;
   byte ProgrammedMinimumPhysicalLinkRate:4;

   // byte 33
   byte HardwareMaximumPhysicalLinkRate:4;
   byte ProgrammedMaximumPhysicalLinkRate:4;

   // byte 34-35
   byte VendorSpecific[2];

   // byte 36-39
   byte ReservedByte36_39[4];

   // byte 40-43
   dword CRC[4];
};

// response specific bytes for SMP Discover, intended to be referenced by
// SMPResponse
struct SMPResponseDiscover
{
   // byte 4-7
   byte IgnoredByte4_7;

   // byte 8
   byte ReservedByte8;

   // byte 9
   byte PhyIdentifier;

   // byte 10-11
   byte ReservedByte10_11;

   // byte 12-43
   struct Discover Results;
};

// response specific bytes for SMP Configure Route Information,
```

```
// intended to be referenced by SMPResponse
struct SMPResponseConfigureRouteInformation
{
   // byte 4-7
   byte CRC[4];
};

// generic structure referencing an SMP Response, must be initialized
// before being used
struct SMPResponse
{
   // byte 0
   byte SMPFrameType;                  // always 41h for SMP responses

   // byte 1
   byte Function;

   // byte 2
   byte FunctionResult;

   // byte 3
   byte ReservedByte3;

   // bytes 4-n
   union
   {
      struct SMPResponseDiscover Discover;
      struct SMPResponseReportGeneral ReportGeneral;
      struct SMPResponseConfigureRouteInformation ConfigureRouteInformation;

// remaining structures would be necessary for a complete implementation
// for this example however they are comments only
//
//      struct SMPRequestSATACapabilities SATACapabilities;
//      struct SMPRequestReportManfacturerInformation
//             ManufacturerInformation;
//      struct SMPRequestReportRouteInformation RouteInformation;
//      struct SMPRequestReportPhyErrorLog PhyErrorLog;
//      struct SMPRequestReportPhySATA PhySATA;
//      struct SMPRequestPhyControl PhyControl;
//      struct SMPRequestPhyMarginControl PhyMarginControl;

   } Response;
};

// this structure is how this simulation obtains it's knowledge about the
// initiator port that is doing the discover, it is not defined as part of
// the standard...
// the simulation assumes that whatever is marked as the initiator in the
// ini file will perform the discover process on all the phys of the device
struct ApplicationClientKnowledge
{
   quadword SASAddress;
   byte NumberOfPhys;
   byte InitiatorBits;
   byte TargetBits;
};

// the TopologyTable structure is the summary of the information gathered
// during the discover process, the table presented here is not concerned
// about memory resources consumed, production code would be more concerned
```

```
// about specifying necessary elements explictly
struct TopologyTable
{
   struct TopologyTable *Next;

   // information from REPORT_GENERAL
   struct SMPResponseReportGeneral Device;

   // information from DISCOVER
   struct SMPResponseDiscover Phy[MAXIMUM_EXPANDER_PHYS];

   // route index for the configuration chain functions
   word RouteIndex[MAXIMUM_EXPANDER_PHYS];

   //
   // in production code there would also be links to the necessary device
   // information like end device; vendor, model, serial number, etc.
   // the gathering of that type of information is not done here...
   //
};
```

SASDiscoverSimulation.cpp

```
//
// this is a simple simulation and code implementation of the initiator based
// expander discovery and configuration proposal, 02-359

// there is no attempt to handle phy errors, arbitration issues, etc.
// production level implementation would have to handle errors appropriately

// structure names used are equivalent to those referenced in the
// SAS-02a document

// basic assumptions
//
// 1. change primitives will initate a rediscovery/configuration sequence
// 2. table locations for SASAddresses are deterministic for a specific
//    topology only, when the topology changes, the location of a SASAddress
//    in an ASIC table cannot be assumed
// 3. a complete discovery level occurs before the configuration of the level
//    begins, multiple passes are required as the levels of expanders
//    encountered between the intiator and the end devices is increased
// 4. configuration of a single expander occurs before proceeding to
//    cascaded expanders
// 5. the Attached structure is filled in following OOB and is available from
//    the initialization routines
// 6. the Iam structure is provide by the application client

#include <malloc.h>
#include <memory.h>

#include "SASDiscoverSimulation.h"

#define ADDRESS_CHAIN_LEVELS  16
#define MAXIMUM_ROUTE_ENTRIES (64*64)

// loaded by the application client, in this simulation it is provided
// in a text file, 02-359r3b SASDomainExample.ini
struct ApplicationClientKnowledge Iam[MAXIMUM_INITIATORS] = { 0 };

// obtained following OOB from the attached phy, in this simulation
```

```
// it is provided in a text file, 02-359r3b SASDomainExample.ini
struct Identify Attached[MAXIMUM_INITIATORS] = { 0 };

// buffers used to request and return SMP data
struct SMPRequest SMPRequestFrame = { 0 };
struct SMPResponse SMPResponseFrame = { 0 };

// resulting discover information will end up in this table
struct TopologyTable *SASDomain[MAXIMUM_INITIATORS] = { 0 };

// this is the function used to send an SMPRequest and get a response back
extern byte SMPRequest(byte PhyIdentifier,
                       quadword Source,
                       quadword Destination,
                       struct SMPRequest *SMPRequestFrame,
                       struct SMPResponse *SMPResponseFrame,
                       byte Function,
                       ...);

// this is the configuration chain it contains the list of expander between
// the programming initiator and the discover extent, each expander in the
// chain must be configured when a new SASaddress is encounter from the top
// of the chain to the bottom
static struct ConfigurationChain
{
   quadword SASAddress;
   struct TopologyTable *Expander;
   byte PhyIdentifier;
} TableChain[ADDRESS_CHAIN_LEVELS] = { 0 };

static int ChainIndex = 0;
static int ChainEntry = 0;

// this is the upstream chain and contains the list of expanders that have
// already been traversed upstream, this reduces the number of times we
// go through the configuration
static quadword UpstreamChain[ADDRESS_CHAIN_LEVELS] = { 0 };

// this the route entry chain it contains the list of route entries that
// have already been configured, this reduces duplication of configuration
// requests
static struct RouteEntryChain
{
   quadword SASAddress;
   word RouteIndex;
   byte PhyIdentifier;
} ConfiguredChain[MAXIMUM_ROUTE_ENTRIES] = { 0 };

// this function gets the report general and discover information for
// a specific expander
struct TopologyTable *DiscoverExpander(byte PhyIdentifier,
                                       quadword SourceSASAddress,
                                       quadword DestinationSASAddress,
                                       struct TopologyTable *Expander)
{
   struct TopologyTable *expander;
   byte phyCount;
   int error = 1;

   // allocate space to retrieve the expander information
   expander = (struct TopologyTable *)
```

```
          calloc(1,
                 sizeof(struct TopologyTable));

// make sure we only do this if the allocation is successful
if(expander)
{
   // get the report general information for the expander
   SMPRequest(PhyIdentifier,
              SourceSASAddress,
              DestinationSASAddress,
              &SMPRequestFrame,
              &SMPResponseFrame,
              REPORT_GENERAL);

   // don't worry about too much in the 'else' case for this example,
   // production code must handle
   if(SMPResponseFrame.FunctionResult == SMP_REQUEST_ACCEPTED)
   {
      // copy the result into the topology table
      memcpy((void *)&(expander->Device),
             (void *)&SMPResponseFrame.Response.ReportGeneral,
             sizeof(struct SMPResponseReportGeneral));

      if(expander->Device.NumberOfPhys <= MAXIMUM_EXPANDER_PHYS)
      {
         // now walk through all the phys of the expander
         for(phyCount = 0;
             (phyCount < expander->Device.NumberOfPhys);
             phyCount++)
         {
            // get the discover information for each phy
            SMPRequest(PhyIdentifier,
                       SourceSASAddress,
                       DestinationSASAddress,
                       &SMPRequestFrame,
                       &SMPResponseFrame,
                       DISCOVER,
                       phyCount);

            // don't worry about the 'else' case for this example,
            // production code must handle
            if(SMPResponseFrame.FunctionResult == SMP_REQUEST_ACCEPTED)
            {
               // clear the error flag
               error = 0;

               // copy the result into the topology table
               memcpy((void *)&(expander->Phy[phyCount]),
                      (void *)&SMPResponseFrame.Response.Discover,
                      sizeof(struct SMPResponseDiscover));
            }
            else
            {
               // something happened so just bailout on this expander
               error = 1;

               // release the memory we allocated for this...
               free(expander);
               break;
            }
         }
```

```
         }
      }

      // if we did not have an error, then link in the information from this
      // expander into the topology table
      if(!error &&
         Expander)
      {
         Expander->Next = expander;
      }
   }

   // the expander pointer is the error return, a null indicates something
   // bad happened...
   return(expander);
}

// this routine will add a SASAddress to the configuration chain
void PushSASAddress(quadword SASAddress,
                    struct TopologyTable *Expander,
                    byte PhyIdentifier)
{
   TableChain[ChainIndex].SASAddress = SASAddress;
   TableChain[ChainIndex].Expander = Expander;
   TableChain[ChainIndex].PhyIdentifier = PhyIdentifier;

   if(ChainIndex < ADDRESS_CHAIN_LEVELS)
   {
      ChainIndex++;
   }
}

// this routine will remove a SASAddress from the configuration chain
void PopSASAddress(void)
{
   if(ChainIndex)
   {
      ChainIndex--;
   }

   TableChain[ChainIndex].SASAddress = 0;
   TableChain[ChainIndex].Expander = 0;
   TableChain[ChainIndex].PhyIdentifier = 0;
}

// this routine will reset the ChainEntry and return the first entry on
// the Chain
quadword FirstSASAddress(void)
{
   ChainEntry = 0;

   return(TableChain[ChainEntry].SASAddress);
}

// this routine will return the next entry on the Chain
quadword NextSASAddress(void)
{
   if(ChainEntry < ChainIndex)
   {
      ChainEntry++;
   }
```

```
   return(TableChain[ChainEntry].SASAddress);
}

// this routine will get the route index from the expander structure
word IndexForSASAddress(void)
{
   struct TopologyTable *expander = TableChain[ChainEntry].Expander;
   word index = 0;

   index = expander->RouteIndex[TableChain[ChainEntry].PhyIdentifier]++;

   return(index);
}

// this routine will get the phy identifier from the configuration chain
byte PhyForSASAddress(void)
{
   return(TableChain[ChainEntry].PhyIdentifier);
}

// this routine keeps track of which SASAddresses have already been
// traversed upstream and avoid duplicate configuration passes
byte AlreadyTraversed(quadword SASAddress)
{
   byte chain = 0;
   byte itsHere = 0;

   while(UpstreamChain[chain])
   {
      if(UpstreamChain[chain] == SASAddress)
      {
         itsHere = 1;
         break;
      }

      if(chain < ADDRESS_CHAIN_LEVELS)
      {
         chain++;
      }
   }

   UpstreamChain[chain] = SASAddress;

   return(itsHere);
}

// this routine keeps track of which expanders have been configured and
// avoids duplicate configuration requests
byte AlreadyConfigured(quadword SASAddress,
                       word RouteIndex,
                       byte PhyIdentifier)
{
   word chain = 0;
   byte itsHere = 0;

   while(ConfiguredChain[chain].SASAddress)
   {
      if((ConfiguredChain[chain].SASAddress == SASAddress) &&
         (ConfiguredChain[chain].RouteIndex == RouteIndex) &&
         (ConfiguredChain[chain].PhyIdentifier == PhyIdentifier))
```

```
      {
         itsHere = 1;
         break;
      }

      if(chain < MAXIMUM_ROUTE_ENTRIES)
      {
         chain++;
      }
   }

   ConfiguredChain[chain].SASAddress = SASAddress;
   ConfiguredChain[chain].RouteIndex = RouteIndex;
   ConfiguredChain[chain].PhyIdentifier = PhyIdentifier;

   return(itsHere);
}

// this function is recursive and discovers then configures as necessary
// the expanders it finds that are "downstream"
struct TopologyTable *DiscoverAndConfigure(byte PhyIdentifier,
                                           quadword SourceSASAddress,
                                           struct TopologyTable *Start,
                                           byte Upstream)
{
   struct TopologyTable *expander = Start;
   struct TopologyTable *nextExpander;
   struct TopologyTable *attachedDeviceSet = 0;

   struct SMPResponseDiscover *discover;
   struct SMPResponseDiscover *configure;

   quadword sasAddress;
   byte phyCount;
   word indexOffset;
   byte phyOffset;

   // if the upstream flag is set, then we're moving upstream looking for
   // the boundary of the device set
   if(Upstream)
   {
      // walk through all the phys of the expander
      for(phyCount = 0;
          (phyCount < expander->Device.NumberOfPhys);
          phyCount++)
      {
         // this is just a pointer helper
         discover = &(expander->Phy[phyCount]);

         // look for phys with edge or fanout devices attached...
         if(((discover->Results.RoutingAttribute == SUBTRACTIVE) &&
             (discover->Results.AttachedDeviceType == EDGE)) ||
            (discover->Results.AttachedDeviceType == FANOUT))
         {
            if(!AlreadyTraversed(discover->Results.AttachedSASAddress))
            {
               nextExpander
               = DiscoverExpander(PhyIdentifier,
                                  SourceSASAddress,
                                  discover->Results.AttachedSASAddress,
                                  expander);
```

```
            // if we successfully got the information from the next
            // expander then link it in the topology table...
            if(nextExpander)
            {
               // if the attached device has a subtractive phy, then
               // stop going upstream, we have two expander device sets
               // connected without a fanout expander, save the address
               // of the attachedDeviceSet, and do the configuration of
               // it as a "separate domain"...
               if(nextExpander->Phy[phyCount].Results.RoutingAttribute
                  == SUBTRACTIVE)
               {
                  attachedDeviceSet = nextExpander;
               }
               // the attached device does not have a subtractive phy, so
               // continue to move upstream until we hit the edge of the
               // device set...
               else
               {
                  // go upstream to the next expander
                  attachedDeviceSet = DiscoverAndConfigure
                                      (PhyIdentifier,
                                       SourceSASAddress,
                                       nextExpander,
                                       1);

                  expander->Next = nextExpander;
               }
            }
         }
      }
   }
}

// this is a maximal decent traversal with a configuration stage
// at each transition to a new level, if a configuration is required
// by the expander

// walk through all the phys of the expander looking for table routes
// only, we'll do configuration as we find them...
for(phyCount = 0;
    (phyCount < expander->Device.NumberOfPhys);
    phyCount++)
{
   // this is just a pointer helper
   discover = &(expander->Phy[phyCount]);

   // look for phys with edge or fanout devices attached...
   if((discover->Results.RoutingAttribute == TABLE) &&
      ((discover->Results.AttachedDeviceType == EDGE) ||
       (discover->Results.AttachedDeviceType == FANOUT)))
   {
      // add the expander to the configuration chain
      PushSASAddress(discover->Results.SASAddress,
                     expander,
                     phyCount);

      nextExpander = DiscoverExpander
                     (PhyIdentifier,
                      SourceSASAddress,
```

```
                discover->Results.AttachedSASAddress,
                expander);

// if we successfully got the information from the next expander
// then link it in the topology table...
if(nextExpander)
{
   byte nextPhyCount = 0;

   // loop through the phys of the nextExpander, gathering
   // the SASAddresses for this expander
   for(nextPhyCount = 0;
       (nextPhyCount < nextExpander->Device.NumberOfPhys);
       nextPhyCount++)
   {
      // configure the expanders between the initiator and
      // the destination
      for(sasAddress = FirstSASAddress();
          sasAddress;
          sasAddress = NextSASAddress())
      {
         word routeIndex = IndexForSASAddress();
         byte phyIdentifier = PhyForSASAddress();

         // this avoids duplicate configure cycles as we walk the
         // topology
         if(!AlreadyConfigured(sasAddress,
                               routeIndex,
                               phyIdentifier))
         {
            struct SMPResponseDiscover *nextDiscover;

            // this is just a helper pointer to reduce line length
            nextDiscover = &(nextExpander->Phy[nextPhyCount]);

            // configure the indexes for the nextExpander in the
            // AttachedExpander
            SMPRequest(PhyIdentifier,
                       SourceSASAddress,
                       sasAddress,
                       &SMPRequestFrame,
                       &SMPResponseFrame,
                       CONFIGURE_ROUTE_INFORMATION,
                       routeIndex,
                       phyIdentifier,
                       0,
                       nextDiscover->Results.AttachedSASAddress);
         }
      }
   }

   // descend to the next expander
   DiscoverAndConfigure(PhyIdentifier,
                        SourceSASAddress,
                        nextExpander,
                        0);

   // remove the last expander from the configuration chain
   PopSASAddress();

   expander->Next = nextExpander;
```

```
         }
      }
   }

   return(attachedDeviceSet);
}

// this routine will append the leaf to the tree domain
void ConcatenateDomains(struct TopologyTable *Tree,
                        struct TopologyTable *Leaf)
{
   while(Tree)
   {
      if(Tree->Next == 0)
      {
         Tree->Next = Leaf;
         break;
      }

      Tree = Tree->Next;
   }
}

// this routine clear the state variables that control execution, this allows
// recursion to work, and allows the different initiator passes to work...
void InitializeForRecursion(void)
{
   word chain = 0;

   for(chain = 0;
       chain < MAXIMUM_ROUTE_ENTRIES;
       chain++)
   {
      if(chain < ADDRESS_CHAIN_LEVELS)
      {
         TableChain[chain].SASAddress = 0;
         TableChain[chain].Expander = 0;
         TableChain[chain].PhyIdentifier = 0;

         UpstreamChain[chain] = 0;
      }

      ConfiguredChain[chain].SASAddress = 0;
      ConfiguredChain[chain].RouteIndex = 0;
      ConfiguredChain[chain].PhyIdentifier = 0;
   }

   ChainIndex = 0;
   ChainEntry = 0;
}

// the application client for the initiator device would make a call to
// this function to begin the discover process...
// to simplify the setup for the simulation, the DiscoverProcess will get
// the Initiator number to allow multiple initiators...
void DiscoverProcess(byte Initiator)
{
   // clear the state variables
   InitializeForRecursion();

   // check to see if an expander is attached
```

```
   // this example ignores the other phys on initiators with multiple phys,
   // the results would be the same, just the determination of wide ports
   // and independent SASDomains would need to be handled...
   if((Attached[Initiator].DeviceType == EDGE) ||
      (Attached[Initiator].DeviceType == FANOUT))
   {
      // expander is attached, so begin to walk the topology, building
      // the necessary tables for each expander
      SASDomain[Initiator] = DiscoverExpander(0,
                                              Iam[Initiator].SASAddress,
                                              Attached[Initiator].SASAddress,
                                              0);
   }

   // make sure we got the information from the attached expander before
   // going on...
   if(SASDomain[Initiator])
   {
      struct TopologyTable *attachedDeviceSet;

      // go upstream on the subtractive phys until we discover that we are
      // attached to another subtractive phy or a fanout expander
      // then begin the discover process from that point, this works
      // because any new address that we find will naturally move upstream
      // due to the subtractive addressing method
      // if during the discover and configuration cycle, it is determined
      // that there are two device sets connected, then a second discover and
      // configuration cycle is required for the other device set
      attachedDeviceSet = DiscoverAndConfigure(0,
                                               Iam[Initiator].SASAddress,
                                               SASDomain[Initiator],
                                               1);

      // if two device sets are connected, then the attached device set
      // has to be discovered and configured separately, but there is no
      // need to do the upstream check...
      if(attachedDeviceSet)
      {
         DiscoverAndConfigure(0,
                              Iam[Initiator].SASAddress,
                              attachedDeviceSet,
                              0);

         // put the domains together
         ConcatenateDomains(SASDomain[Initiator],
                            attachedDeviceSet);
      }
   }
}
```