To: T10 Technical Committee
From: Steven Fairchild, HP (steve.Fairchild@hp.com)
Date: 17 September 2002
Subject: Expander Configuration Details

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 DISCOVERY, 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 Discovery 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 which needs to be populated with expander route entries to properly resolve connection requests.

direct routing: The routing capability of an expander device that resolves a connection request to a single end device.

edge expander device set: A set of edge expander devices that are bounded by a subtractive routing port from one member of the set.

expander route entry: A single destination SAS address and routed phy identifier reference within an expander route table. The expander route entry is referenced by using the phy identifier and expander route index variable.

expander route index: A variable used in combination with the phy identifier to select a single destination SAS address reference within an expander route table.

expander route table: A table of expander route entries internal to an expander device that is referenced by the expander device to resolve connection requests.

subtractive routing: The routing capability of an edge expander device that contains a single expander port used to route any connection request that is not resolved within the edge expander device.

table routing: The routing capability of an expander device that contains one or more expander ports used to route connection requests that are resolved by an expander route table.

Change 2: SMP Frame Changes

9.4.4.2 REPORT GENERAL function

{change response format to mark expander route slots as reserved, due to decision to make expander route slots == number of phys}

Table 78 defines the response format.

Table 78. 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	~~(MSB)~~	~~EXPANDER ROUTE SLOTS~~						
5	Reserved							~~(LSB)~~
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 SLOTS field contains the maximum number of route slots for an expander device. Expander devices shall support this field. Other device types shall return a zero value in this field, indicating that no expander route table is supported. If an expander device supports an expander route table, then the number of expander route slots shall be at least equal to the number of phys on the expander device.~~

The EXPANDER ROUTE INDEXES field contains the maximum number of route indexes for an expander device. Expander devices shall support this field. Other device types shall return a zero value in this field, indicating that no expander route table is supported.

If an edge expander device supports an expander route table, then the number of expander route indexes shall be sufficient to provide an expander route table entry for each phy of each member of the edge expander device set that is downstream from the edge expander device.

If a fanout expander device supports an expander route table, then the number of expander route indexes shall be greater than or equal to the maximum number of phys to support the maximum supported devices in an edge expander device set (see 4.1.9).

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

The CONFIGURABLE ROUTE TABLE bit indicates whether 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 and shall have defined values for the ~~EXPANDER ROUTE SLOTS and~~ EXPANDER ROUTE INDEXES. An expander device without a configurable route table shall have the configurable route table bit set to zero and may have defined values for the ~~expander route slots and~~ EXPANDER ROUTE INDEXES.

9.4.4.5 REPORT ROUTE INFORMATION function

{change request and response format to mark expander route slots as reserved, due to decision to make expander route slots == number of phys}

The REPORT ROUTE INFORMATION function returns the route table information for a specific expander ~~route slot~~ phy identifier and expander route index within an expander device. Expander devices shall

support this function if the Report General function has a non-zero value~~s for EXPANDER ROUTE SLOTS and~~ for EXPANDER ROUTE INDEXES. 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	~~(MSB)~~	~~EXPANDER ROUTE SLOT~~						
5	Reserved							~~(LSB)~~
6	(MSB)	EXPANDER ROUTE INDEX						
7								(LSB)
8	Reserved							
~~9~~	~~Ignored~~							
~~10~~								
9	PHY IDENTIFER							
10	Ignored							
11	Reserved							
12	(MSB)	CRC						
15								(LSB)

~~The EXPANDER ROUTE SLOT field indicates the route slot for which the Report Route information is being requested. The value must be in the range of 0 to EXPANDER ROUTE SLOTS or a function result unknown function shall occur.~~

The EXPANDER ROUTE INDEX field indicates the route index for which the REPORT ROUTE INFORMATION is being requested. The value must be in the range of zero to EXPANDER ROUTE INDEXES (see 9.4.4.2) or a function result UNKNOWN FUNCTION shall occur. A phy within an expander device that does not support table routing shall return a function result of INDEX DOES NOT EXIST for any EXPANDER ROUTE INDEX of the phy.

The PHY IDENTIFIER field contains the phy identifier for which the REPORT ROUTE INFORMATION is being requested. The value must be in the range of zero to NUMBER OF PHYS (see 9.4.4.2) or a function result of UNKNOWN FUNCTION shall occur. A phy within an expander device that does not support table routing shall return a function result of INDEX DOES NOT EXIST for any EXPANDER ROUTE INDEX of the phy.

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	~~(MSB)~~	~~EXPANDER ROUTE SLOT~~						
5	Reserved							~~(LSB)~~
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 SLOT field indicates the route slot for which the Report Route information has been requested.~~

The EXPANDER ROUTE INDEX field indicates the expander route index for which the REPORT ROUTE INFORMATION has been requested.

The PHY IDENTIFIER field contains the phy identifier routed by this table entry.

The ROUTE ENTRY DISABLED field indicates the content of the frame is valid for routing. The 1b value indicates the route has been determined to be in violation of connection rules.

The ROUTED SAS ADDRESS field contains the SAS address routed by this table entry.

9.4.4.6 DISCOVER function

{correct typographical errors introduced in sas-01b }

The DISCOVER function returns the physical link configuration information for the specified phy identifier. The function response ~~information~~ provides information from the IDENTIFY address frame received by the phy, as well as the ~~addressing support provided~~ routing method 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 87. DISCOVER request

{no change to table 87}

The PHY IDENTIFIER field indicates the phy (see 4.2.6) for which information shall be reported. The value must be in the range of zero to NUMBER OF PHYS (see 9.4.4.2) or a function result of UNKNOWN FUNCTION shall occur.

The CRC field is defined in 9.4.2.

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 (00h)							
2	FUNCTION RESULT							
3	Reserved							
4	Ignored							
7								
8	Reserved							
9	PHY IDENTIFIER							
10	Ignored							
11	Reserved							
12	~~ROUTE ENTRY DISABLED~~ Ignored	ATTACHED DEVICE TYPE			ROUTING METHOD			
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)	ATTACHED SAS ADDRESS						
23								(LSB)
24	(MSB)	SAS ADDRESS						
31								(LSB)
32	PROGRAMMED MINIMUM PHYSICAL LINK RATE				HARDWARE MINIMUM PHYSICAL LINK RATE			
33	PROGRAMMED MAXIMUM PHYSICAL LINK RATE				HARDWARE MAXIMUM PHYSICAL LINK RATE			
34	Ignored							
35								
36	Reserved							
39								
40	(MSB)	CRC						
43								(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 for which physical configuration link information is being returned.

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

Table 90. Route method

Code	Route method
0h	Direct routing
1h	Subtractive routing
2h	Table routing
All others	Reserved

The ROUTE METHOD field shall be statically defined and shall not change based on the attached device phy.

Expander phy supporting direct routing shall only route connection requests to attached device.

If multiple phys within an expander device supporting direct routing are attached to end devices, the attached SAS addresses define the width of the SAS port.

Expander phy supporting subtractive routing shall, when attached to an expander device phy, route connection requests that are not resolved by expander device phys supporting direct routing or table routing.

Expander phy supporting subtractive routing shall, when attached to an end device phy, follow the rules of a phy with direct routing support.

If multiple phys within an expander device supporting subtractive routing are attached to expander devices, they shall terminate at attached phys with identical SAS addresses, defining a single wide SAS port.

If multiple phys within an expander device supporting subtractive routing are attached to expander devices that do not terminate at identical SAS addresses, the discovery process (see 4.1.9.2) application client shall indicate a vendor unique attachment error, providing the expander device phys contributing to the attachment error.

Expander phy supporting table routing shall, when attached to an expander device phy, route connection requests not resolved by direct routing by referencing an expander route table.

Expander phy supporting table routing shall, when attached to an end device phy, follow the rules of a phy with direct routing support.

The ATTACHED DEVICE TYPE field indicates the type of attached device ~~containing the phy~~ (see 7.7.2), and is defined in table 48.

~~A ROUTE ENTRY DISABLED bit of zero indicates the content of the frame is valid for routing. A ROUTE ENTRY DISABLED bit of one indicates the route table entry has been determined to be in violation of attachment rules and shall not be used for routing a connection request.~~

The PHYSICAL LINK RATE field indicates the physical link rate for this 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	Rate unknown
1h	Phy does not exist
2h	Disabled
3h	Failed
4h	Spinup hold OOB
5h	1,5 Gbps
6h	3,0 Gbps
7h – Fh	Reserved

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 phy.
The SAS ADDRESS field contains the SAS address of this phy.
The HARDWARE MAXIMUM PHYSICAL LINK RATE field indicates the maximum physical link rate supported by the phy.
The PROGRAMMED MAXIMUM PHYSICAL LINK RATE field indicates the maximum physical link rate set by the PHY CONTROL function.
The HARDWARE MINIMUM PHYSICAL LINK RATE field indicates the minimum physical link rate supported by the phy.
The PROGRAMMED MINIMUM PHYSICAL LINK RATE field indicates the minimum physical link rate set by the PHY CONTROL function.

~~The ATTACHED PHY IDENTIFIER field indicates the PHY IDENTIFIER value received during the link reset sequence.~~
~~The ATTACHED DEVICE TYPE field indicates the DEVICE TYPE value received during the link reset sequence.~~
~~If SAS PROTOCOL is set to one, the ATTACHED SAS ADDRESSfield indicates the SAS address received during the link reset sequence. If SAS PROTOCOL is set to zero, this field contains the unique SAS address assigned by the expander device.~~
~~The PHY RESET PROBLEM bit is set to one if the phy state machine obtained dword synchronization for at least one rate during speed negotiation (either SAS or SATA), but the speed negotiation sequence failed. The phy may have retried the negotiation and succeeded, but the bit remains set to one. The bit remains set to one until the phy is reset without any failures.~~

9.4.4.9 CONFIGURE ROUTE INFORMATION function

{change request and response format to mark expander route slots as reserved, due to decision to make expander route slots == number of phys}

The CONFIGURE ROUTE INFORMATION function sets the expander route entry ~~table information~~ for a specific ~~expander route slot~~ phy identifier and expander route index within the expander route table of a configurable 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 Report General function has the CONFIGURABLE ROUTE TABLE field set.

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	~~(MSB)~~	~~EXPANDER ROUTE SLOT~~						
5	Reserved							~~(LSB)~~
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 SLOT field indicates the route slot for which the Configure Route information is being configured. The value must be in the range of zero to EXPANDER ROUTE SLOTS or a function result unknown function shall occur.~~

The EXPANDER ROUTE INDEX field indicates the route index for which the CONFIGURE ROUTE INFORMATION is being configured. The value must be in the range of zero to EXPANDER ROUTE INDEXES (see 9.4.4.2) or a function result UNKNOWN FUNCTION shall occur.

The PHY IDENTIFIER field indicates the phy for which the CONFIGURE ROUTE INFORMATION is being configured. The value must be in the range of zero to NUMBER OF PHYS (see 9.4.4.2) or a function result of UNKNOWN FUNCTION shall occur. If PHY IDENTIFIER specifies a phy that does not support table routing a function result of INDEX DOES NOT EXIST shall occur.

The DISABLE ROUTE ENTRY bit when set to zero indicates the content of the frame is valid for routing. The DISABLE ROUTE ENTRY bit set to one indicates the route table entry is in violation of attachment rules and shall not be used for routing connection requests.

The ROUTED SAS ADDRESS field contains the SAS address to be configured in the expander route entry.

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 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 classes of expander devices:

a) edge expander devices: expander devices that contain subtractive routing capability on at most one port; and
b) fanout expander devices: expander devices that do not contain subtractive routing capability.

Both classes of expanders may support direct routing and table routing capability.

4.1.7.1 Edge expander device set

Edge expander devices may be grouped into edge expander device sets. The edge expander device sets are bounded by the direct routing ports attached to end devices and the port supporting subtractive routing that is: attached to the port of a fanout device; attached to the subtractive routing port on another edge expander device; or attached to an end device.

The table routing ports of an edge expander device provide attachments to other edge expander devices in the edge expander device set.

The number of initiator devices and target devices attached to an edge expander device set may not exceed the maximum number of initiator devices and target devices for an edge expander device (see 4.1.9).

Figure X shows an edge expander device set.

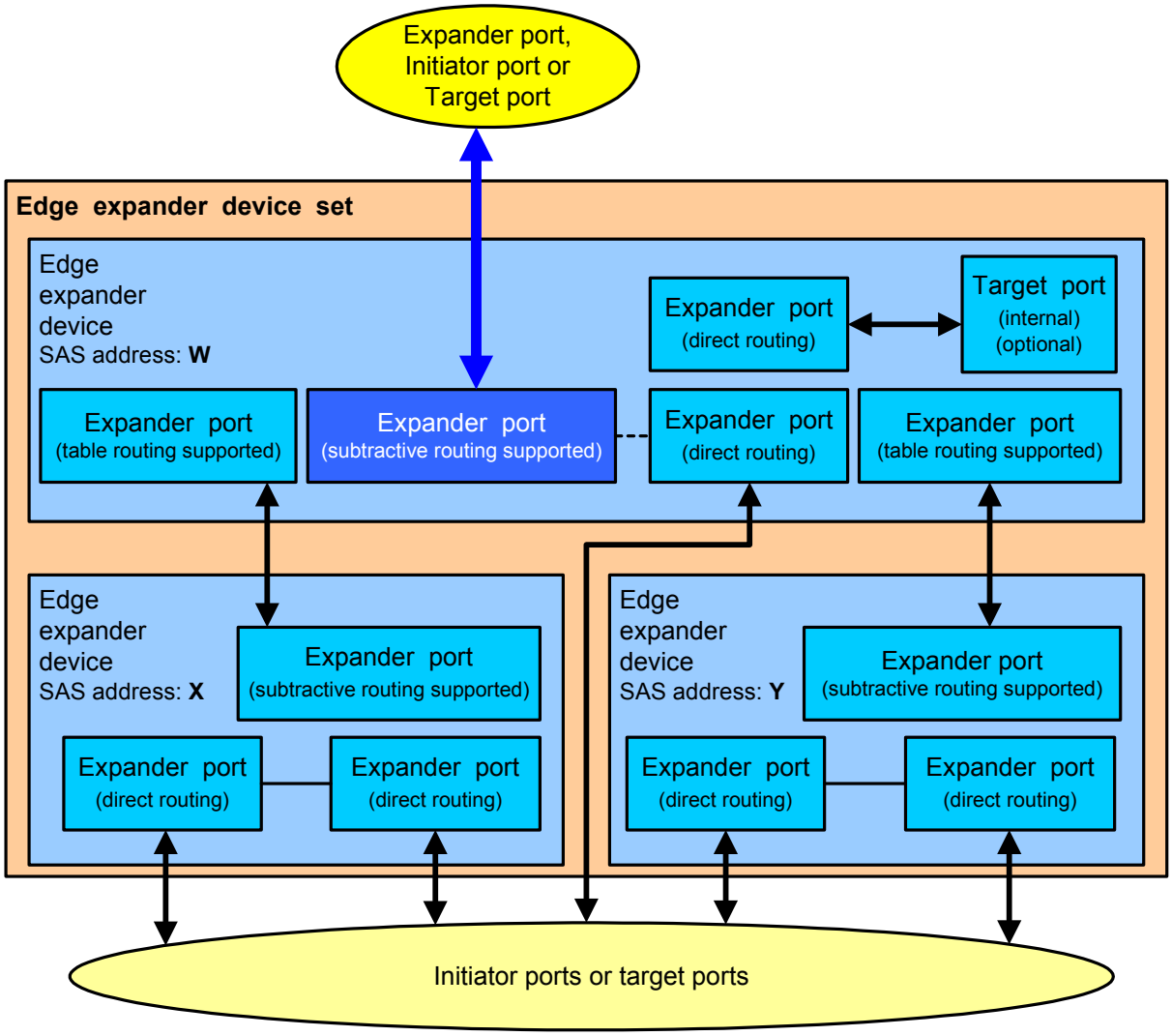

Figure X – Edge expander device set

4.1.7.2 Configurable expander device

Expander devices may have configurable routing capability. Expander devices with configurable routing capability depend on the application client in an initiator device to use the discovery process (see 4.1.9.2) to populate an expander route table (see 4.1.9.1) prior to the expander device being fully capable of routing connection requests.

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 *may be attached to no more than one fanout expander device. Each edge expander device set may be attached to up to 64, initiator devices or target devices.*

An edge expander device set *may be attached to another* edge expander device set *only if there are no other expanders in the SAS domain.*

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

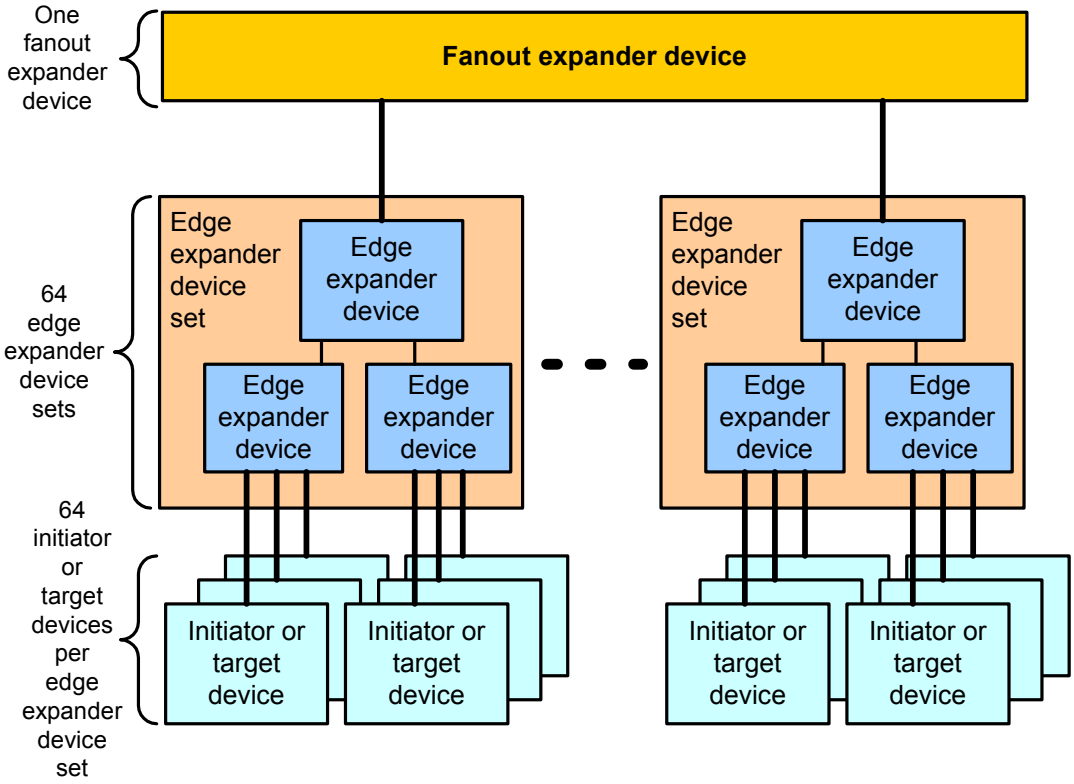

Figure 13 – Maximum expander topology

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

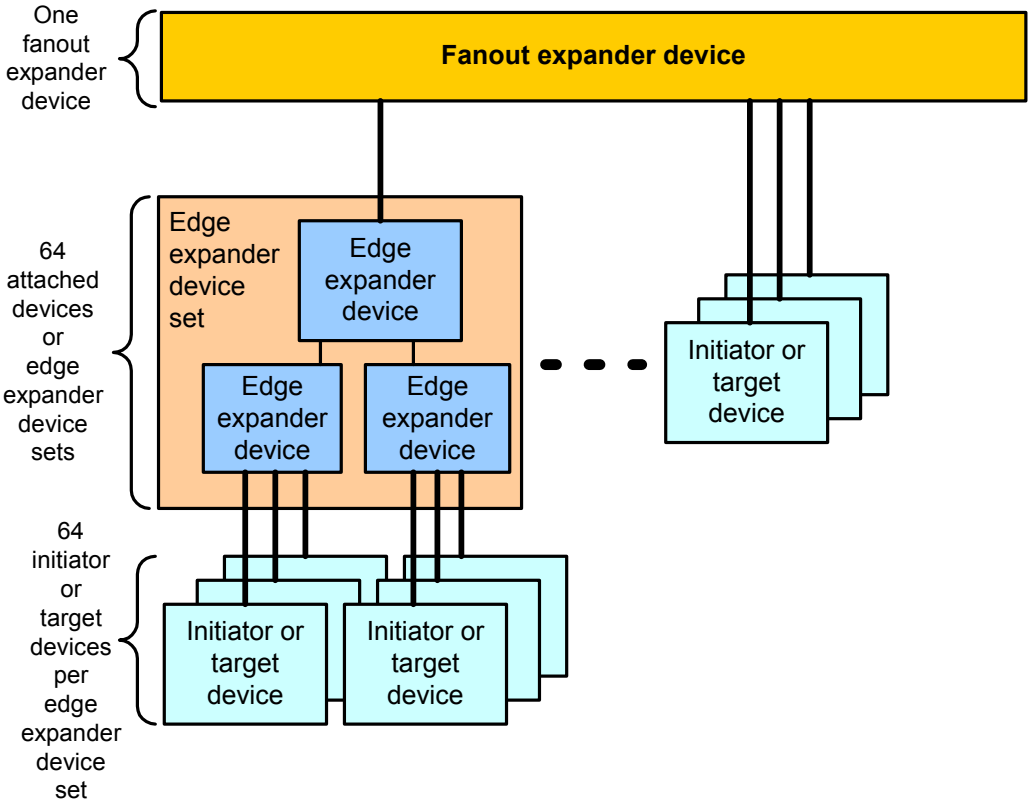

Figure 14 – Fanout expander topology

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.

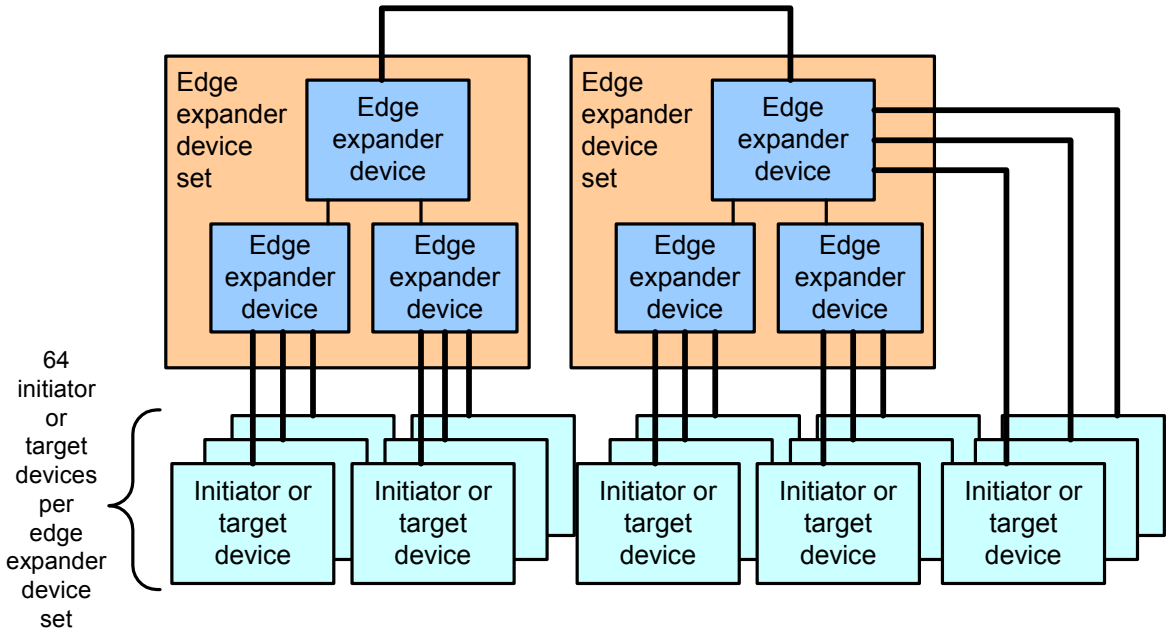

Figure 15 – Edge expander device set topology

4.1.9.1 Expander device connection request routing

The expander connection manager within an expander device (see 4.8.4) shall support at least one of the following route methods to route connection requests to expander ports within the expander device:

a) direct routing: expander port that only routes connection requests to attached devices.
b) subtractive routing: expander port defined as routing connection requests that are not resolved using the direct routing or table routing methods. An edge expander device shall have at most one statically defined subtractive routing port. A fanout expander device shall not support this routing method. An expander port defined as supporting subtractive routing shall also support direct routing to attached initiator or target ports.
c) table routing: expander port defined as routing connection requests that are not resolved using the direct routing method by referencing an expander route table to route connection requests. An expander device may have one or more statically defined table routing ports. An expander port defined as supporting table routing shall also support direct routing to attached initiator or target ports.

The ROUTE METHOD field of the SMP DISCOVER response shall indicate the route method supported by an expander port. The ROUTE METHOD field shall be statically defined and shall not change based on the attached device port.

An expander device with ports that support differing route methods shall determine how to route a connection request with the following precedence:

a) route to a port with direct routing to the SAS address indicated by the connection request; or
b) route to a port with table routing where the expander routing table contains the SAS address indicated by the connection request; or
c) route to a port with subtractive routing; or
d) return an OPEN-REJECT (NO DESTINATION) to the connection request source.

The supported attachments between expander ports based on the expander device type and port routing methods are:

a) edge expander port with subtractive routing capability attached to an edge expander port with subtractive routing capability; or
b) edge expander port with subtractive routing capability attached to an edge expander port with table routing capability; or
c) edge expander port with subtractive routing capability attached to a fanout expander port with table routing capability

The discovery process shall tolerate the unsupported attachment of a fanout expander port with table routing capability to a fanout expander port with table routing capability, but the details of the implementation are beyond the scope of this specification.

During the discovery process (see 4.1.9.2) if an application client in an initiator device detects an unsupported expander port attachment, it shall report and/or record an attachment error in a vendor unique manner, providing the SAS address and phy identifier of the attachment in error. Any devices located downstream from the unsupported port attachment with respect to the discovering application client are unreachable.

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 an organized structure that provides an association between destination SAS addresses and expander phy identifiers. Each association is an expander route entry.

An expander device may indicate its capability of maintaining the expander route entries by the CONFIGURABLE ROUTE TABLE field in the SMP REPORT GENERAL response (see 9.4.4.2).

The NUMBER OF PHYS and EXPANDER ROUTE INDEXES fields in the SMP REPORT GENERAL response (see 9.4.4.2) define the size of the expander route table.

An initiator device references a specific expander route entry within an expander route table by using the PHY IDENTIFIER and EXPANDER ROUTE INDEX fields in the SMP REPORT ROUTE INFORMATION (see 9.4.4.5) or SMP CONFIGURE ROUTE INFORMATION (see 9.4.4.9) functions.

Figure X shows a logical representation of an expander route table with the elements identified.

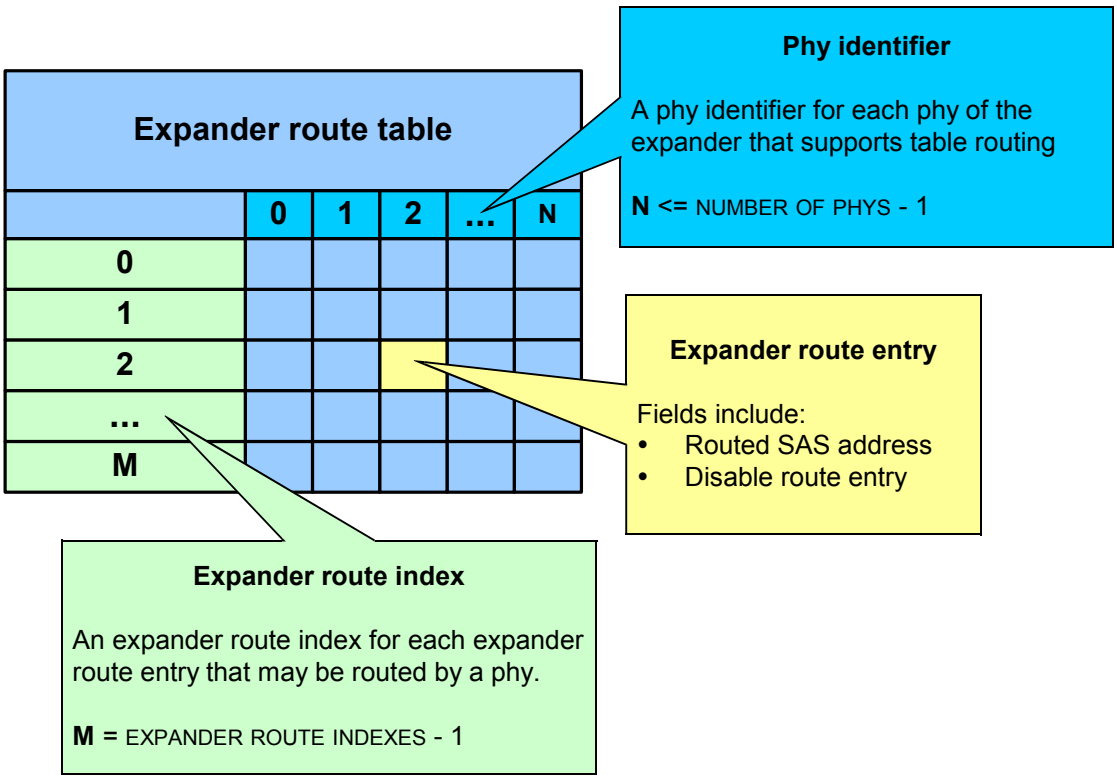

Figure X – Expander route table example

The number of devices attached to edge expander device sets and to the fanout expander determines the maximum extent of the SAS domain (see 4.1.9). The number of phys necessary to support the maximum device count within the SAS domain is less deterministic, since it depends on the number of wide ports and the number of phys within each wide port.

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

An expander device is considered downstream of another expander device when the expander device subtractive routing port is attached to the table routing port of another edge or fanout expander device.

To avoid an overflow of the fanout expander route index during the discovery process (see 4.1.9.2) the minimum number of fanout expander route indexes per phy identifier shall be either:

a) the sum of all downstream edge expander device phys within an attached edge expander device set; or
b) the product of:
    a) the maximum number of devices within an edge expander device set; and
    b) the number of phys within each port.

In the event of an overflow during the discovery process, the application client within the initiator device performing the discovery process shall report and/or record in a vendor unique method that an expander route index overflow occurred, providing:

a) the SAS address of the expander; and
b) the expander phy identifier of the overflow; and
c) the list of attached SAS addresses that will be unreachable. If any of the unreachable attached SAS addresses is for an edge expander device, then any devices attached to that edge expander device will be unreported.

The order of the expander route entries in the expander route indexes is deterministic based on the topology. Figure Y shows the expander route index order.

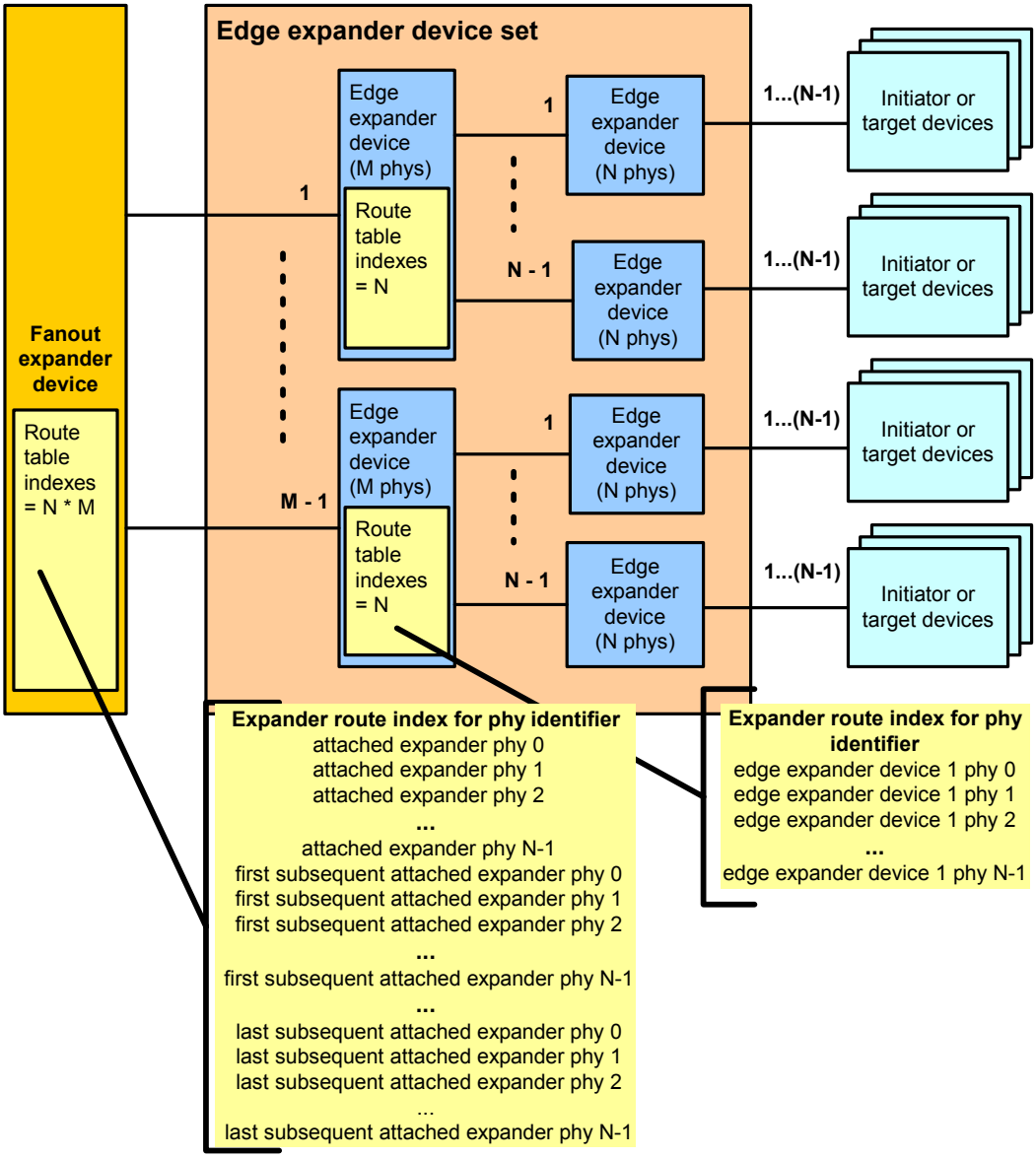

Figure Y – Expander route index order

4.1.9.2 Discovery process

The discovery process is the method an application client within an initiator device uses to complete a level order traversal of the SAS domain to identify the initiator devices, target devices and expander devices within the expander topology. The order of traversal should be:

1) expander device to which the initiator port is attached; and

2) every device attached to that expander device; and

3) if another expander device is found, every device attached to that expander device.

The discovery process begins with the application client within an initiator device determining that an expander device is attached. The application client makes this determination by checking the DEVICE TYPE field in the IDENTIFY frame from the attached device.

If the DEVICE TYPE is either an edge device or fanout device, then the application client shall determine whether the expander device requires configuration (see 4.1.9.1.1). If the expander device is

configurable then the application client also determines the size of the expander route table to configure (see 4.1.9.1.1).

The application client shall then do a level order traversal of the topology by opening SMP connections to the attached expander device and querying each phy in ascending order (from zero to NUMBER OF PHYS – 1) using the SMP DISCOVER function. The application client shall construct a topology table including the information from the SMP DISCOVER response for each phy queried.

When the application client completes the querying of each phy in the attached expander device, the query process is repeated for each subsequent expander device phy encountered.

Figure X shows an example topology, which is used to describe the discovery process.

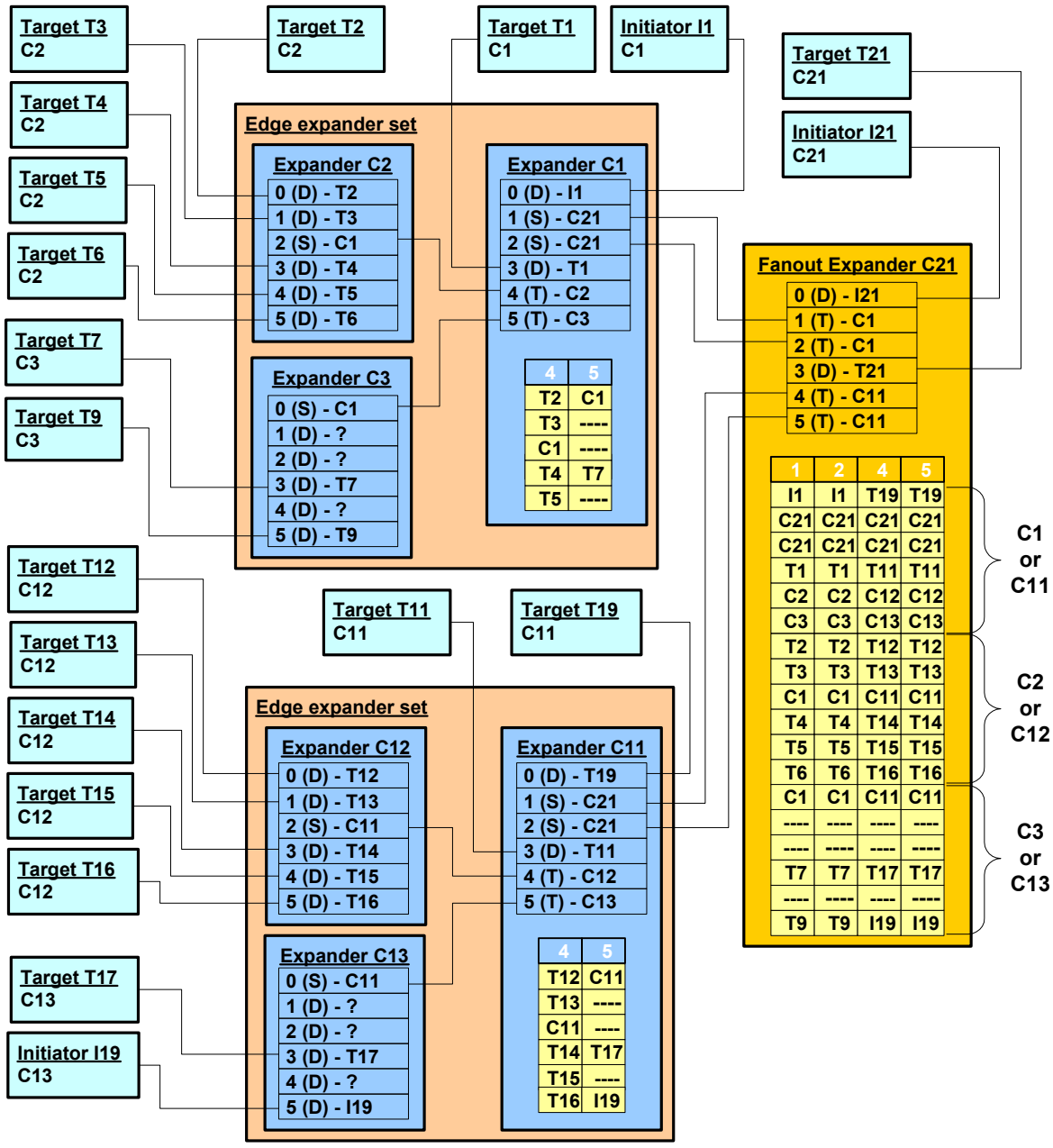

Figure X – Expander topology example

Using Figure X as an example, the application client within initiator device (I1) shall have completed the equivalent of Table X after traversing the edge expander device set with edge expander device members, C1, C2 and C3.

Expander Route Entry	Expander SAS Address	Attached phy Identifier	Attached SAS Address	Device Type	Address Decode
0	C1	0	I1	end	
1	C1	1	C21	edge	subtractive
2	C1	2	C21	edge	subtractive
3	C1	3	T1	end	
4	C1	4	C2	edge	table (configurable)
5	C1	5	C3	edge	table (configurable)
6	C2	0	T2	end	
7	C2	1	T3	end	
8	C2	2	C1	edge	subtractive
9	C2	3	T4	end	
10	C2	4	T5	end	
11	C2	5	T6	end	
12	C3	0	C1	edge	subtractive
13	C3	1		none	
14	C3	2		none	
15	C3	3	T7	end	
16	C3	4		none	
17	C3	5	T9	end	

Table X – Initiator topology table for example initiator device, I1

For configurable expander devices, the SMP CONFIGURE ROUTE INFORMATION function is then used to configure the expander route table before proceeding to the next attached expanders. This step is required so that the configurable expander devices are supplied with sufficient information in the expander route table to be capable of routing subsequent connection requests to the successive expander devices.

Using Figure X as an example, the initiator device (I21) shall have completed the equivalent of Table Y after traversing the fanout expander (C21) and encountering the configurable edge expander device sets (C1, C2, C3 and C11, C12, C13).

Expander Route Entry	Expander SAS Address	Attached phy Identifier	Attached SAS Address	Device Type	Address Decode
0	C21	0	I21	end	
1	C21	1	C1	fanout	table (configurable)
2	C21	2	C1	fanout	table (configurable)
3	C21	3	T21	end	
4	C21	4	C11	fanout	table (configurable)
5	C21	5	C11	fanout	table (configurable)
6	C1	0	I1	end	
7	C1	1	C21	edge	subtractive
8	C1	2	C21	edge	subtractive
9	C1	3	T1	end	
10	C1	4	C2	edge	table (configurable)
11	C1	5	C3	edge	table (configurable)
12	C1	0	I1	end	
13	C1	1	C21	edge	subtractive
14	C1	2	C21	edge	subtractive
15	C1	3	T1	end	
16	C1	4	C2	edge	table (configurable)
17	C1	5	C3	edge	table (configurable)
18	C11	0	T19	end	
19	C11	1	C21	edge	subtractive
20	C11	2	C21	edge	subtractive
21	C11	3	T11	end	
22	C11	4	C12	edge	table (configurable)
23	C11	5	C13	edge	table (configurable)
24	C11	0	T19	end	
25	C11	1	C21	edge	subtractive
26	C11	2	C21	edge	subtractive
27	C11	3	T11	end	
28	C11	4	C12	edge	table (configurable)
29	C11	5	C13	edge	table (configurable)

Table Y – Initiator topology table for example initiator device, I21

Once the initiator device has completed the construction of the topology table, it must configure the route tables for each of the configurable expander devices encountered during the discover process.

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

Change 5: Phy identifier

4.2.6 Phy identifier

Each phy in a device shall be assigned a unique 8 bit identifier. The phy identifier is used for management functions.

Phy identifiers shall be greater than or equal to 00h and less than 40h.

Change 6: Expander device model

4.8.4.1 SAS address mapping

The Expander Function shall provide SAS address to phy identifier mapping using the SAS address and device type, which are exchanged during phy initialization.

Fanout expander devices must perform additional discovery (see 7.6.3) for each attached edge expander device in order to establish an expander route table.

An initiator device must perform the discovery process (see 4.1.9.2) to identify and configure any expander devices with a configurable expander route table.

Change 7: Discover rules

7.6.2 Initiator device specific rules

After identifying that it is attached to an expander device after a link reset sequence, or after receiving a CHANGE primitive sequence, the application client within an initiator device shall perform a discovery process (see 4.1.9.2).

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

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

If during the discovery 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 identifying that it is attached to an edge expander device after a link reset sequence, or after receiving a CHANGE primitive sequence, the application client within a fanout expander device that does not require an application client within an initiator device for configuration shall follow the initiator device specific rules (see 7.6.2) to perform a discovery process. Then the fanout device shall follow the edge expander device specific rules (see 7.6.4).

A fanout expander device that requires initiator device configuration shall follow the edge expander device specific rules (see 7.6.4).

7.6.4 Edge expander device specific rules

After completing the identify sequence, an edge expander device is capable of routing connection requests to attached devices.

An edge expander device that requires an application client within an initiator device for configuration is dependent on at least one application client within an initiator device to complete the discovery process (see 4.1.9.2) before it is fully capable of routing connection requests beyond attached devices.

Change 8: Domain changes

7.9 Domain changes

SAS initiator ports scan the domain during a discovery 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.

~~*Proposal note: After examining the startup process, the recommendation is to remove the restriction on expander devices to not transmit CHANGE to more than one physical link per expander port (statement in red above). The restriction requires complexity in the expander that is unnecessary, since the startup process will typically require the initiator devices to handle multiple CHANGE requests from the same expander phy within short durations as attached devices come ready.*~~

Proposal Note: There is a SAS Change Notification Proposal being drafted by Jim Reif to cover this issue.

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 9: 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 10: 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            96
#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
};

// RoutingMethod
enum RoutingMethod
{
  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                    // see 7.7.2
{
  // 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, must be initialized before being used
struct SMPRequest                  // see 9.4.2
{
  // 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 RoutingMethod: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                  // see 9.4.3
{
  // 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 explicitly */
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-r01b document assuming 02-359 is approved; the paragraph references are to SAS-r01b

Basic assumptions

- change primitives will initate a rediscovery/configuration sequence
- 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
- 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
- configuration of a single expander occurs before proceeding to subsequent expanders attached
- the Attached structure is filled in following OOB and is available from the initialization routines
- 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) + 512)

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

/* obtained from the attached phy, in this simulation it is provided in a text file, SAS-02-359Example.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 one of these tables */
struct TopologyTable *SASDomain[MAXIMUM_INITIATORS] = { 0 };

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

/* this is the configuration chain it contains the list of expanders 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(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(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));

      /* this check is done to make sure we don't overrun our basic assumptions */
      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(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 TableChain */
quadword FirstSASAddress(void)
{
  ChainEntry = 0;

  return(TableChain[ChainEntry].SASAddress);
}

/* this routine will return the next entry on the TableChain */
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(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.RoutingMethod == SUBTRACTIVE) &&
          (discover->Results.AttachedDeviceType == EDGE)) ||
        (discover->Results.AttachedDeviceType == FANOUT))
      {
        if(!AlreadyTraversed(discover->Results.AttachedSASAddress))
        {
          nextExpander = DiscoverExpander(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.RoutingMethod == 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(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.RoutingMethod == TABLE) &&
    ((discover->Results.AttachedDeviceType == EDGE) ||
     (discover->Results.AttachedDeviceType == FANOUT)))
  {
    /* add the expander to the configuration chain */
    PushSASAddress(discover->Results.SASAddress,
                   expander,
                   phyCount);

    nextExpander = DiscoverExpander(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(SourceSASAddress,
                         sasAddress,
                         &SMPRequestFrame,
                         &SMPResponseFrame,
                         CONFIGURE_ROUTE_INFORMATION,
                         routeIndex,
                         phyIdentifier,
                         0,
                         nextDiscover.Results.AttachedSASAddress);
            }
          }
        }

        /* descend to the next expander */
        DiscoverAndConfigure(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 */
  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(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(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(Iam[Initiator].SASAddress,
                           attachedDeviceSet,
                           0);

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