

|  |
| --- |
|  |

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **Port Numbering API’s** |
| **Authors** | **DELL** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **02/09/2015** |
| **SAI-Version** | **V0.9.2** |

**Contents**

[List of Changes i](#_Toc416338492)

[1 Overview 1](#_Toc416338493)

[1.1 Application Port to SAI port Mapping 1](#_Toc416338494)

[1.2 Setup Lane mode for a SAI port 1](#_Toc416338495)

[1.3 Lane Numbering 1](#_Toc416338496)

[2 Specification 2](#_Toc416338497)

[2.1 Changes to saitypes.h 2](#_Toc416338498)

[2.1.1 New definitions 2](#_Toc416338499)

[2.1.2 Update to sai\_attribute\_value\_t 3](#_Toc416338500)

[2.2 Changes to saiswitch.h 3](#_Toc416338501)

[2.2.1 New attributes 3](#_Toc416338502)

[2.3 Changes to saiport.h 3](#_Toc416338503)

[2.3.1 New Définitions 3](#_Toc416338504)

[2.3.2 New attributes 4](#_Toc416338505)

[3 API Flow 4](#_Toc416338506)

[3.1 Get Physical Lane Information 4](#_Toc416338507)

[3.2 BREAKOUT/IN 4](#_Toc416338508)

[4 Examples 5](#_Toc416338509)

[4.1 Port Mapping 5](#_Toc416338510)

[4.1.1 Switch Initialize 5](#_Toc416338511)

[4.1.2 Get Port HW details 6](#_Toc416338512)

[4.1.3 Single Lane Port’s HW Info Get 6](#_Toc416338513)

[4.2 Handling Port Breakout 7](#_Toc416338514)

[4.2.1 Get Port’s supported Breakout Mode(s) 7](#_Toc416338515)

[4.2.2 Breakout: 1 to 4 8](#_Toc416338516)

[4.2.3 Breakout with invalid Inputs: 9](#_Toc416338517)

[4.2.4 Un-breakout 4 to 1 10](#_Toc416338518)

[4.2.5 Un-breakout with invalid Inputs 11](#_Toc416338519)

[4.2.6 Single Lane Port’s breakout Mode 12](#_Toc416338520)

[5 Appendix 12](#_Toc416338521)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 0.9.2 | Proposal for Port Numbering – Base Version | Version 1 | 02/16/15 |
| 0.9.2 | Updated Overview, Added Examples, Modified Appendix | Version 2 | 04/07/15 |
| 0.9.2 | Updated with Review Comments | Version 3 | 04/09/15 |

License

© 2014 Microsoft Corporation, Dell Inc., Facebook, Inc, Broadcom Corporation, Intel Corporation, Mellanox Technologies Ltd.

As of September 9, 2014, the following persons or entities have made this Specification available under the Open Web Foundation Final Specification Agreement (OWFa 1.0), which is available at <http://www.openwebfoundation.org/legal/the-owf-1-0-agreements/owfa-1-0>

Microsoft Corporation, Dell Inc., Facebook, Inc, Intel Corporation, Mellanox Technologies Ltd.

You can review the signed copies of the Open Web Foundation Agreement Version 1.0 for this Specification at <http://opencompute.org/licensing/>, which may also include additional parties to those listed above.

Your use of this Specification may be subject to other third party rights. THIS SPECIFICATION IS PROVIDED "AS IS." The contributors expressly disclaim any warranties (express, implied, or otherwise), including implied warranties of merchantability, noninfringement, fitness for a particular purpose, or title, related to the Specification. The entire risk as to implementing or otherwise using the Specification is assumed by the Specification implementer and user. IN NO EVENT WILL ANY PARTY BE LIABLE TO ANY OTHER PARTY FOR LOST PROFITS OR ANY FORM OF INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION OR ITS GOVERNING AGREEMENT, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND WHETHER OR NOT THE OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE FOLLOWING IS A LIST OF MERELY REFERENCED TECHNOLOGY: Microprocessor technology, semiconductor manufacturing technology, operating system technology (including without limitation networking operating system technology), emulation technology, graphics technology, video technology, integrated circuit packaging technology and the like, compiler technologies, object oriented technology, optical/RF communications technology including chip I/O and driver technology, bus technology, memory chip technology (including, without limitation, NAND memory, NOR memory, resistive RAM (RRAM), seek scan probe (SSP) memory, nonvolatile memory (including without limitation, memory based on chalcogenide materials, phase change memory (PCM), one or more stacked layers of memory cells, embedded PCM memories, non-volatile cache memory, solid state drives, SRAM, embedded DRAM, ferro-electric memory, and polymer memory)) and/or health-related and medical technology. IMPLEMENTATION OF THESE TECHNOLOGIES MAY BE SUBJECT TO THEIR OWN LEGAL TERMS.

# Overview

This document describes the APIs to obtain physical lane information for a SAI port as well as setup the lane mode for a SAI port.

## Application Port to SAI port Mapping

Adapter Host usually has the notation of Application Port (App Port), which refers to the different ports that are used in the system like Front-Panel, Fabric. For front panel, it would be the actual physical location of the port in the box. SAI ports represent the switch logical ports that the Adapter Host can use. Since SAI port number is opaque to Adapter Host and different SAI implementation may choose its own numbering scheme, the Adapter Host needs to map an Application Port to a SAI port.

Since the Adapter Host has knowledge of physical lane information of an Application port based on the platform, this proposal allows SAI to also expose such physical lane information of a SAI port. Therefore, the Adapter Host can associate an Application port with a SAI port using such physical lane information.

In this proposal, the physical lane information in a switching ASIC is defined as the lane number. Every ASIC has a numbering scheme to uniquely number every physical lane. For example, an ASIC has 128 physical lanes, and they are then numbered from 1 to 128.

## Setup Lane mode for a SAI port

A switch SAI port can correspond to a physical communication lane or multiple physical lanes. The switch allows user to bundle several physical lanes into one switch logical port (non-breakout mode), and can also unbundle them (Breakout mode). New SAI ports can be created during the breakout operation, and existing ports can be deleted during the un-breakout operation.

For lane breakout, the application can use SAI\_PORT\_ATTR\_BREAKOUT\_MODE to know current breakout mode and do a break out. As for the un-breakout operation, due to certain switching chip constraints, the application may needs extra information from the ASIC vendors to know exactly what lanes can be bundled together.

The proposal provides API to do the (un)breakout operations for a SAI port, and provides event notification to indicate such SAI add/removal event. The proposal also provides query API to get a list of all current SAI ports available in the system.

Before we do (un)breakout operations for a SAI port(s), the Adaptor Host should remove all the port settings on the SAI port(s) object and other objects like VLAN, FDB, L3, ACL, QOS, Sample packet, Mirror etc.

During SAI initialization, SAI notifies all available SAI ports though the port event notification.

## Lane Numbering

The Lane numbering will vary from vendor to vendor. The vendor should propose a mapping that would map their internal numbering based on the architecture to a 32 bit lane number. There are many ways to map, some suggestions are given below

Some NPU’s may have fixed port and each lane may be represented by a HW port. In that case the mapping is simple – **Lane Number 🡪 HW Port**

The NPU may have Port Blocks and lanes within port blocks. These may be serially increasing. In such cases the Lane number can be derived thru a simple formula. In a 64 lane system we can have 16 port blocks numbered 0-15 and within each port block we can have internal lane numbers 1-4. We can have a simple formula to derive the lane number. **Lane Number = (Port Block Num \* 4) + Internal Lane number**

The NPU has <Block, Lane> and Block, Lane are not increasing serially. For this we would have 24 or 16 bits allocated for Block and remaining for Lane and encode to **32 bit Lane number - <HW Block (24 bits), Lane Number (8 bits)>**

# Specification

## Changes to saitypes.h

### New definitions

/\*

\* To enable Break in/out on a system port(s)

\*

\* **breakout\_mode** – Type of (un)breakout we want to do

\* **port\_list** – List of all ports to be breakout or un-breakout. For breakout operation, it could be 1 port to 2 ports, 1 port to 4 ports, and 2 ports to 4 ports. For un-breakout, it could be 2 ports to 1 port, 4 ports to 2 ports, and 2 ports to 1 port.

\*/

typedef struct \_sai\_port\_breakout\_t

{

sai\_port\_breakout\_mode\_type\_t breakout\_mode;

sai\_object\_list\_t port\_list;

} **sai\_port\_breakout\_t**;

/\*

\* Different breakout modes supported

\*/

typedef enum \_sai\_port\_breakout\_mode\_type\_t

{

SAI\_PORT\_BREAKOUT\_MODE\_1\_LANE = 1 ,

SAI\_PORT\_BREAKOUT\_MODE\_2\_LANE = 2,

SAI\_PORT\_BREAKOUT\_MODE\_4\_LANE = 4,

SAI\_PORT\_BREAKOUT\_MODE\_MAX

} **sai\_port\_breakout\_mode\_type\_t**;

### Update to sai\_attribute\_value\_t

…

…

**sai\_port\_breakout\_t** portbreakout ;

} sai\_attribute\_value\_t;

## Changes to saiswitch.h

### New attributes

typedef enum \_sai\_switch\_attr\_t

{

/\* READ-ONLY \*/

/\* Get the port list \*/

SAI\_SWITCH\_ATTR\_PORT\_LIST, /\* **sai\_object\_list\_t \*/**

/\* Write Only \*/

/\* Set Breakout for a port \*/

SAI\_SWITCH\_ATTR\_PORT\_BREAKOUT, /\* **sai\_port\_breakout\_t \*/**

..

..

## Changes to saiport.h

### New Définitions

typedef enum \_sai\_port\_event\_t

{

SAI\_PORT\_EVENT\_ADD, /\*A new active port created \*/

SAI\_PORT\_EVENT\_DELETE /\* An existing port is invalided and can be deleted \*/

} **sai\_port\_event\_t** ;

/\*

\* Routine Description:

\* Port event notification

\* Arguments:

\* [in] port\_id - port id

\* [in] port\_event - port event associated with this port

\*

\* Return Values:

\* None

\*/

typedef void (\*sai\_port\_event\_notification\_fn)(

\_In\_ sai\_object\_id\_t port\_id,

\_In\_ sai\_port\_event\_t port\_event

);

### New attributes

/\*

\* HW details of a SAI port. This information

\* will help the application to map a SAI port to a user port

\* We provide the list of lanes associated with the port

\*

\*/

typedef enum \_sai\_port\_attr\_t

{

/\* READ-ONLY \*/

/\* The hardware lane list (sai\_u32\_list\_t)\*/

SAI\_PORT\_ATTR\_HW\_LANE\_LIST /\* **sai\_u32\_list\_t**\*/

/\* The breakout capabilities supported by the SAI port\*/

SAI\_PORT\_ATTR\_BREAKOUT\_MODE, /\* **sai\_s32\_list\_t**,

Value - **sai\_port\_breakout\_mode\_type\_t \*/**

/\* current breakout mode (sai\_port\_breakout\_mode\_type\_t) \*/

SAI\_PORT\_ATTR\_CURRENT\_BREAKOUT\_MODE,

…

…

# API Flow

## Get Physical Lane Information

* sai\_initialize\_switch()
* SAI notifies the list of SAI ports thru SAI\_PORT\_EVENT\_ADD
* Application can optionally call SAI\_SWITCH\_ATTR\_PORT\_LIST attribute get to get the list of SAI ports in case it misses the PORT\_ADD\_EVENT
* Applications calls Port Attribute GET with
  + SAI\_PORT\_ATTR\_HW\_LANE\_LIST,

to get HW lane mapping for each of the SAI port. The application uses this information to map a SAI port to an application port

## BREAKOUT/IN

* Call SAI\_SWITCH\_ATTR\_PORT\_BREAKOUT attribute set to enable BREAKOUT/IN. The port(s) to BREAKOUT/IN is passed
* A set of port events are generated with SAI\_PORT\_EVENT\_ADD for the new ports created and SAI\_PORT\_DELETE\_PORT for the existing ports that are invalid
* Applications calls Port Attribute GET with
  + SAI\_PORT\_ATTR\_HW\_LANE\_LIST,

to get HW mapping for each of the new SAI ports and updates its mapping and starts using the new ports and stops using the removed ports

# Examples

## Port Mapping

**Example 1:** Consider the below table, where the NPU has 4 physical block with 4 lanes on each and supports Breakout mode 1X and 4X. The lanes are numbered 1-16

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Application Port** | **Lane Number** | **No of Lanes** | **SAI port**  **(UOID)** | **Breakout Supported Modes** |
| 1 | 1, 2, 3, 4 | 4 | 100 | SAI\_PORT\_BREAKOUT\_MODE\_1\_LANE  SAI\_PORT\_BREAKOUT\_MODE\_4\_LANE |
| 4 | 9,10,11,12 | 4 | 104 | SAI\_PORT\_BREAKOUT\_MODE\_1\_LANE  SAI\_PORT\_BREAKOUT\_MODE\_4\_LANE |
| 8 | 13,14,15,16 | 4 | 108 | SAI\_PORT\_BREAKOUT\_MODE\_1\_LANE  SAI\_PORT\_BREAKOUT\_MODE\_4\_LANE |
| 12 | 5, 6, 7, 8 | 4 | 112 | SAI\_PORT\_BREAKOUT\_MODE\_1\_LANE  SAI\_PORT\_BREAKOUT\_MODE\_4\_LANE |

### Switch Initialize

sai\_api\_initialize (0, NULL);

sai\_api\_query (SAI\_API\_SWITCH, &sai\_switch\_api);

sai\_switch\_api->initialize\_switch (0, NULL, NULL, &switch\_notifications);

#### Output: Port event callback notification

switch\_notifications->on\_port\_event (100, SAI\_PORT\_EVENT\_ADD);

switch\_notifications->on\_port\_event (104, SAI\_PORT\_EVENT\_ADD);

switch\_notifications->on\_port\_event (108, SAI\_PORT\_EVENT\_ADD);

switch\_notifications->on\_port\_event (112, SAI\_PORT\_EVENT\_ADD);

### Get Port HW details

sai\_api\_query (SAI\_API\_PORT, &sai\_port\_api);

sai\_object\_id\_t port\_id = 100 ; (repeat for port 104, 108, 112)

sai\_attribute\_t \*sai\_attr\_get;

sai\_attr\_get = calloc (1, sizeof(sai\_attribute\_t));

sai\_attr\_get[0].id = SAI\_PORT\_ATTR\_HW\_LANE\_LIST;

sai\_attr\_get[0].value.u32list.count = 4;

sai\_attr\_get[0].value.u32list.list = calloc(4, sizeof(uint32\_t)) ;

sai\_port\_api->get\_port\_attribute (port\_id, 1, &sai\_attr\_get) ;

#### output for port 100

sai\_attr\_get.value.u32list.count = 4

sai\_attr\_get.value.u32list.list[0] = 1

sai\_attr\_get.value.u32list.list[1] = 2

sai\_attr\_get.value.u32list.list[1] = 3

sai\_attr\_get.value.u32list.list[3] = 4

#### output for port 108

sai\_attr\_get.value.u32list.count = 4

sai\_attr\_get.value.u32list.list[0] = 13

sai\_attr\_get.value.u32list.list[1] = 14

sai\_attr\_get.value.u32list.list[2] = 15

sai\_attr\_get.value.u32list.list[3] = 16

**Example 2:** Consider a NPU, which doesn’t support Breakout Mode, has fixed speed ports and do not expose the lane information. In such cases the physical information is defined only by the single lane number which will be same as the HW port

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Application Port** | **Lane Number** | **No of Lanes** | **SAI port** | **Breakout Supported Modes** |
| 1 | 2 | 1 | 100 | SAI\_PORT\_BREAKOUT\_MODE\_1\_LANE |
| 2 | 4 | 1 | 101 | SAI\_PORT\_BREAKOUT\_MODE\_1\_LANE |
| 3 | 1 | 1 | 102 | SAI\_PORT\_BREAKOUT\_MODE\_1\_LANE |
| 4 | 3 | 1 | 103 | SAI\_PORT\_BREAKOUT\_MODE\_1\_LANE |

### Single Lane Port’s HW Info Get

sai\_api\_query (SAI\_API\_PORT, &sai\_port\_api);

sai\_object\_id\_t port\_id = 100 ; (repeat for port 101, 102, 103)

sai\_attribute\_t \*sai\_attr\_get;

sai\_attr\_get = calloc (1, sizeof(sai\_attribute\_t));

sai\_attr\_get[0].id = SAI\_PORT\_ATTR\_HW\_LANE\_LIST;

sai\_attr\_get[0].value.u32list.count = 4;

sai\_attr\_get[0].value.u32list.list = calloc(4, sizeof(uint32\_t)) ;

sai\_port\_api->get\_port\_attribute (port\_id, 1, &sai\_attr\_get) ;

#### output for port 100

sai\_attr\_get.value.u32list.count = 1

sai\_attr\_get.value.u32list.list[0] = 2

## **Handling Port Breakout**

 A 40G or a 100G port can be broken down into 10G ports. Some NPU supports this to be done dynamically. There are 2 cases

* Breakout : 40G/100G to 10G port
* Un-breakout: 10G ports back to 40G/100G

1. Get supported breakout Mode
2. Configure the supported breakout mode

### Get Port’s supported Breakout Mode(s)

sai\_attribute\_t sai\_attr\_get;

memset (&sai\_attr\_get, 0, sizeof(sai\_attribute\_t));

sai\_attr\_get.id = SAI\_PORT\_ATTR\_BREAKOUT\_MODE;

sai\_attr\_get.value.s32list.count= 1;

sai\_port\_breakout\_mode\_t  \*mode\_list = NULL ;

mode\_list = calloc (1, sizeof(sai\_port\_breakout\_mode\_t)) ;

sai\_attr\_get.value.s32list.list =  mode\_list ;

sai\_port\_api)->get\_port\_attribute (port\_id, 1, &sai\_attr\_get) ;

#### Get API Failure: SAI\_STATUS\_BUFFER\_OVERFLOW

If return value is SAI\_STATUS\_BUFFER\_OVERFLOW, then memory allocated for the list is insufficient; so re-invoke the API after allocating the updated size in s32list.count.

Output: Updated list count

sai\_attr\_get.value.s32list.count= 2

#### Re-invoke the API with the required memory,

sai\_attribute\_t sai\_attr\_get;

memset(&sai\_attr\_get, 0, sizeof(sai\_attribute\_t));

sai\_attr\_get.id = SAI\_PORT\_ATTR\_BREAKOUT\_MODE;

sai\_port\_breakout\_mode\_t  \*mode\_list = NULL ;

sai\_attr\_get.value.s32list.count= 2

mode\_list = calloc(2 , sizeof(sai\_port\_breakout\_mode\_t)) ;

sai\_attr\_get.value.s32list.list =  mode\_list ;

sai\_port\_api)->get\_port\_attribute (port\_id, 1, &sai\_attr\_get) ;

#### Output

sai\_attr\_get.value.s32list.count= 2;

sai\_attr\_get.value.s32list.list[0] = SAI\_PORT\_BREAKOUT\_MODE\_1\_LANE;

sai\_attr\_get.value.s32list.list[1] = SAI\_PORT\_BREAKOUT\_MODE\_4\_LANE;

Getting the breakout modes supported on a port doesn’t necessarily provide the supported speeds on the port, rather it provides the different lane combinations supported on the mode. For instance, on a 100G port which might possibly support 4x25G and 4x10G, SAI\_PORT\_BREAKOUT\_MODE\_4\_LANE support only means it support 4 individual sub-lanes on the port; in reality it might support only 4x25G mode but not 4x10G.

### Breakout: 1 to 4

The Adapter host calls SAI to breakout passing the SAI 100/40G port to breakout. The SAI handles the request and returns new SAI ports available to use. (Example) Adapter host requests for Break out of port SAI port 104. SAI handles the request and returns the 4 new SAI ports that the Adapter Host can use - 104, 105, 106, 107

#### Set Breakout Mode

sai\_attribute\_t sai\_attr\_set;

memset(&sai\_attr\_set, 0, sizeof(sai\_attribute\_t));

sai\_attr\_set.id = SAI\_SWITCH\_ATTR\_PORT\_BREAKOUT ;

sai\_attr\_set.value.portbreakout.mode = SAI\_PORT\_BREAKOUT\_MODE\_4\_LANE;

sai\_attr\_set.value.portbreakout.port\_list.object\_count = 1 ;

sai\_object\_list\_t \*port\_list = NULL;

port\_list = calloc (1, sizeof(sai\_object\_id\_t)) ;

port\_list[0] = 104;

sai\_attr\_set.value.portbreakout.object\_list =  port\_list;

sai\_port\_api)->set\_switch\_attribute (&sai\_attr\_set) ;

#### Port event callback notification

switch\_notifications->on\_port\_event(104, SAI\_PORT\_EVENT\_DELETE) ;

switch\_notifications->on\_port\_event (104, SAI\_PORT\_EVENT\_ADD);

switch\_notifications->on\_port\_event (105, SAI\_PORT\_EVENT\_ADD);

switch\_notifications->on\_port\_event (106, SAI\_PORT\_EVENT\_ADD);

switch\_notifications->on\_port\_event (107, SAI\_PORT\_EVENT\_ADD);

#### Get new port’s HW details

sai\_object\_id\_t port\_id = 104; (repeat for 105, 106, 107)

sai\_attribute\_t \*sai\_attr\_get;

sai\_attr\_get = calloc (1, sizeof(sai\_attribute\_t));

sai\_attr\_get[0].id = SAI\_PORT\_ATTR\_HW\_LANE\_LIST;

sai\_attr\_get[0].value.u32list.count = 4;

sai\_attr\_get[0].value.u32list.list = calloc(4, sizeof(uint32\_t)) ;

sai\_port\_api->get\_port\_attribute (port\_id, 1, &sai\_attr\_get) ;

#### Output

Port 104 :

sai\_attr\_get.value.u32list.count = 1

sai\_attr\_get.value.u32list.list[0] = 9

port 105:

sai\_attr\_get.value.u32list.count = 1

sai\_attr\_get.value.u32list.list[0] = 10

port 106:

sai\_attr\_get.value.u32list.count = 1

sai\_attr\_get.value.u32list.list[0] = 11

port 107:

sai\_attr\_get.value.u32list.count = 1

sai\_attr\_get.value.u32list.list[0] = 12

### Breakout with invalid Inputs:

#### Not supported Mode

sai\_attribute\_t sai\_attr\_set;

memset(&sai\_attr\_set, 0, sizeof(sai\_attribute\_t));

sai\_attr\_set.id = SAI\_SWITCH\_ATTR\_PORT\_BREAKOUT ;

sai\_attr\_set.value.portbreakout.mode = SAI\_PORT\_BREAKOUT\_MODE\_2\_LANE;

sai\_attr\_set.value.portbreakout.port\_list.port\_count = 1 ;

sai\_object\_list\_t \*port\_list = NULL;

port\_list = calloc (1, sizeof(sai\_object\_id\_t)) ;

port\_list[0] = 104;

sai\_attr\_set.value.portbreakout.port\_list =  port\_list;

sai\_port\_api)->set\_switch\_attribute (&sai\_attr\_set) ;

##### Output

SAI\_STATUS\_NOT\_SUPPORTED as SAI\_PORT\_BREAKOUT\_MODE\_2\_LANE is not supported by port 104.

### Un-breakout 4 to 1

This is a simpler case. The SAI would disable the breakout and the 10G ports are disabled and only the SAI port that is active after disabling breakout is returned. For the example above, when we disable breakout SAI would return back Port 104 to be used as 100/40G

#### Set un-Breakout Mode

sai\_attribute\_t sai\_attr\_set;

memset(&sai\_attr\_set, 0, sizeof(sai\_attribute\_t));

sai\_attr\_set.id = SAI\_SWITCH\_ATTR\_PORT\_BREAKOUT;

sai\_attr\_set.value.portbreakout.mode = SAI\_PORT\_BREAKOUT\_MODE\_1\_LANE;

sai\_attr\_set.value.portbreakout.port\_list.port\_count = 4;

sai\_object\_list\_t \*port\_list = NULL;

port\_list = calloc (1, sizeof(sai\_object\_id\_t));

port\_list[0] = 104;

port\_list[1] = 105;

port\_list[2] = 106;

port\_list[3] = 107;

sai\_attr\_set.value.portbreakout.port\_list =  port\_list;

sai\_port\_api)->set\_switch\_attribute (&sai\_attr\_set);

#### Port event callback notification

switch\_notifications->on\_port\_event (104, SAI\_PORT\_EVENT\_DELETE);

switch\_notifications->on\_port\_event (105, SAI\_PORT\_EVENT\_DELETE);

switch\_notifications->on\_port\_event (106, SAI\_PORT\_EVENT\_DELETE);

switch\_notifications->on\_port\_event (107, SAI\_PORT\_EVENT\_DELETE);

switch\_notifications->on\_port\_event (104, SAI\_PORT\_EVENT\_ADD);

#### Get new port’s HW details

sai\_attribute\_t \*sai\_attr\_get;

sai\_attr\_get = calloc (1, sizeof(sai\_attribute\_t));

sai\_object\_id\_t port\_id = 104 ;

sai\_attr\_get[0].id = SAI\_PORT\_ATTR\_HW\_LANE\_LIST;

sai\_attr\_get[0].value.u32list.count = 4;

sai\_attr\_get[0].value.u32list.list = calloc(4, sizeof(uint32\_t)) ;

sai\_port\_api->get\_port\_attribute (port\_id, 1, &sai\_attr\_get) ;

#### output for port 104

sai\_attr\_get.value.u32list.count = 4

sai\_attr\_get.value.u32list.list[0] = 9

sai\_attr\_get.value.u32list.list[1] = 10

sai\_attr\_get.value.u32list.list[1] = 11

sai\_attr\_get.value.u32list.list[3] = 12

After un-breakout, get API would fail for with SAI\_STATUS\_INVALID\_PORT\_NUMBER for port’s 105, 106, 107

### Un-breakout with invalid Inputs

#### Not supported Mode

sai\_attribute\_t sai\_attr\_set;

memset(&sai\_attr\_set, 0, sizeof(sai\_attribute\_t));

sai\_attr\_set.id = SAI\_SWITCH\_ATTR\_PORT\_BREAKOUT;

sai\_attr\_set.value.portbreakout.mode = SAI\_PORT\_BREAKOUT\_MODE\_2\_LANE;

sai\_attr\_set.value.portbreakout.port\_list.port\_count = 4;

sai\_object\_list\_t \*port\_list = NULL;

port\_list = calloc (1, sizeof(sai\_object\_id\_t));

port\_list[0] = 104;

port\_list[1] = 105;

port\_list[2] = 106;

port\_list[3] = 107;

sai\_attr\_set.value.portbreakout.port\_list =  port\_list;

sai\_port\_api)->set\_switch\_attribute (&sai\_attr\_set);

##### Output

1. SAI\_STATUS\_INVALID\_PARAMETER; SAI\_PORT\_BREAKOUT\_MODE\_2\_LANE would need a port\_count of 2; its 4 in above example
2. SAI\_PORT\_BREAKOUT\_MODE\_2\_LANE with correct no. of ports: if this mode is not supported on these ports, it would result in SAI\_STATUS\_NOT\_SUPPORTED error.

#### Wrong combination of ports in list

sai\_attribute\_t sai\_attr\_set;

memset(&sai\_attr\_set, 0, sizeof(sai\_attribute\_t));

sai\_attr\_set.id = SAI\_SWITCH\_ATTR\_PORT\_BREAKOUT;

sai\_attr\_set.value.portbreakout.mode = SAI\_PORT\_BREAKOUT\_MODE\_4\_LANE;

sai\_attr\_set.value.portbreakout.port\_list.port\_count = 4;

sai\_object\_list\_t \*port\_list = NULL;

port\_list = calloc (1, sizeof(sai\_object\_id\_t));

port\_list[0] = 104;

port\_list[1] = 105;

port\_list[2] = 106;

port\_list[3] = 108;

sai\_attr\_set.value.portbreakout.port\_list =  port\_list;

sai\_port\_api)->set\_switch\_attribute (&sai\_attr\_set);

##### Output

SAI\_STATUS\_NOT\_SUPPORTED as the ports in the port\_list are not part of the same physical block.

### Single Lane Port’s breakout Mode

For single lane ports that doesn’t breakout mode as in example 2, port attribute get SAI\_PORT\_ATTR\_BREAKOUT\_MODE would return SAI\_PORT\_BREAKOUT\_MODE\_1\_LANE alone. So it wouldn’t be possible to configure any other mode on these ports. Calling switch attribute SAI\_SWITCH\_ATTR\_PORT\_BREAKOUT with any breakout mode other than 1X would result in SAI\_SATUS\_NOT\_SUPPORTED.

# Appendix