

|  |
| --- |
|  |

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](#_Toc411863648)

[1 Overview 1](#_Toc411863649)

[1.1 SAI Port Representation 1](#_Toc411863650)

[1.2 Handling Remote Ports 1](#_Toc411863651)

[2 Specification 2](#_Toc411863652)

[2.1 Changes to saitypes.h 2](#_Toc411863653)

[2.1.1 New definitions 2](#_Toc411863654)

[2.1.2 Update to sai\_attribute\_value\_t 2](#_Toc411863655)

[2.2 Changes to saiswitch.h 2](#_Toc411863656)

[2.2.1 New attributes 2](#_Toc411863657)

[2.3 Changes to saiport.h 3](#_Toc411863658)

[2.3.1 New Définitions 3](#_Toc411863659)

[2.3.2 New attributes 3](#_Toc411863660)

[3 API Flow 4](#_Toc411863661)

[3.1 Get System Ports 5](#_Toc411863662)

[3.2 BREAKOUT/IN 5](#_Toc411863663)

[4 Appendix 5](#_Toc411863664)

[4.1 Application Port 5](#_Toc411863665)

[4.2 HW Port 6](#_Toc411863666)

[4.3 SAI Port 6](#_Toc411863667)

[4.4 Logical Port Mapping 6](#_Toc411863668)

[4.5 HW Port Mapping 6](#_Toc411863669)

[4.6 Various Steps Involved 8](#_Toc411863670)

[4.7 Port Mappings 8](#_Toc411863671)

[4.7.1 Adapter Host 8](#_Toc411863672)

[4.7.2 Adapter (SAI) 8](#_Toc411863673)

[4.8 Handling Breakout 8](#_Toc411863674)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 0.9.2 | Proposal for Port Numbering – Base Version |  | 02/16/15 |

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

In general each NPU may have its own NPU numbering. It can be

* Only HW port number corresponding to each physical port
* HW port number corresponding to each physical port and a logical port mapping to this HW port number to provide an easy access to the application

We need an abstraction of this numbering from application so a transparent numbering scheme is needed so applications view of a port remains the same irrespective of the NPU Vendor

* SAI would provide a set of logical ports to the application. The application would use them transparently without trying to understand how a SAI port is represented
* On the other hand the application needs to map each of the SAI port to a user port. So for each SAI port, the application would get the HW details of the port which could be a simple port number or be lane numbering. Since the application has knowledge of port mapping of a user port to actual HW port/lane, it would use the information to map the SAI port to a user port
* Break out/in is a common feature that needs to be supported. Various combinations – 1\*40G, 4 \*40G, 4 \*25G, 1 \*100G will be supported. Based on break out/in a new set of ports may be created and existing ports deleted. So based on the break out/in a new set of port events indicating ADD/Remove is generated
* Once the switch is initialized SAI would generate ADD\_PORT for all the default SAI Ports. Any application that misses that event can still query to get the current list of SAI ports
* For a lane based port numbering, we would use the block number, lane information. For a simple HW port mapping, block number would be used to represent the HW port number and the lane information is absent or not applicable

## **SAI Port Representation**

A SAI port can have the following information embedded in it

* Port Type - Needed if we support more than just Front-Panel Port
* NPU/Fabric ID - Needed in a system with Multi NPU's, we need to program a distinct Fabric ID for each NPU in the system
* NPU Logical or Hardware Port Information

Each SAI implementation may decide how they want the internal port numbering scheme should be

## **Handling Remote Ports**

 In a system where there are more than 1 NPU, there would be a need to switch packets between them or we can have a common LAG across these 2 NPU's. In that case we may need to program the port of the remote NPU from this NPU. Given that the Adapter has visibility of only 1 NPU, there should be a way to program the remote NPU's SAI ports in the given Adapter Instance. If we go with the SAI port representation above, a given adapter can easily deduce remote ports using the NPU/Fabric ID and use that to handle the programming remote ports case. The assumption would be that the Fabric ID is distinct. The Host Adapter or layer above would have the info about all the NPU's in the system and their respective SAI ports

# Specification

## Changes to saitypes.h

### New definitions

/\*

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

\*

\* **breakout\_mode** – Type of breakout/in we want to do

\* **port\_list** – List of ports that needs to be breakout/in. For breakout it could typically be 1

\* port. For break in it would be a set of ports

\*/

typedef struct \_sai\_port\_break\_outin\_t

{

uint32\_t breakout\_mode;

sai\_port\_list\_t port\_list;

} **sai\_port\_break\_outin\_t**;

/\*

\* List of the Breakout Mode’s. A port like 100G may support more Break out modes

\*/

typedef struct \_sai\_port\_break\_outin\_mode\_t

{

uint32\_t num\_modes ;

uint32\_t mode\_list ;

} **sai\_port\_break\_outin\_mode\_t** ;

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

…

…

**sai\_port\_id\_t** portid;

**sai\_port\_break\_outin\_t** break\_outin\_portlist ;

**sai\_port\_break\_outin\_mode\_t**  break\_outin\_mode ;

} 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\_port\_list\_t \*/**

/\* Get the CPU Port \*/

SAI\_SWITCH\_ATTR\_CPU\_PORT, /\* **sai\_port\_id\_t \*/**

/\* Set Breakout for a port \*/

SAI\_SWITCH\_ATTR\_PORT\_BREAK\_IN\_OUT, /\* **sai\_port\_break\_outin\_t \*/**

..

..

## Changes to saiport.h

### New Définitions

typedef enum \_sai\_port\_event\_t

{

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

SAI\_PORT\_DELETE\_EVENT /\* 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 - new event associated with this port

\*

\* Return Values:

\* None

\*/

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

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

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

);

### New attributes

/\*

\* Different breakout modes supported

\*/

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

{

SAI\_PORT\_BREAKOUT\_MODE\_1X ,

SAI\_PORT\_BREAKOUT\_MODE\_2X,

SAI\_PORT\_BREAKOUT\_MODE\_4X,

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

/\*

\* HW details of a SAI port. This information

\* would help the application to map a SAI port to an user port

\* **start\_lane** – A port like 40G/100G is a combination of multiple lanes. start\_lane gives the 1st lane

\* within the block num

\* **end\_lane** – end\_lane gives the last lane within the block num

\* **lane\_active\_bitmap** – In case there are inactive/invalid lanes within the start/end lane. This would

\* provide that indication

\*

\*/

typedef enum \_sai\_port\_attr\_t

{

/\* READ-ONLY \*/

/\* The HW Map corresponding to the SAI port\*/

SAI\_PORT\_ATTR\_HW\_BLOCK\_NUM, /\* **sai\_uint32\_t** \*/

/\* HW Lane Info – Start Lane Number \*/

SAI\_PORT\_ATTR\_HW\_START\_LANE\_NUM, /\* **sai\_uint32\_t** \*/

/\* HW Lane Info – End Lane Number \*/

SAI\_PORT\_ATTR\_HW\_END\_LANE\_NUM, /\* **sai\_uint32\_t** \*/

/\* HW Lane Info – Number of Lanes \*/

SAI\_PORT\_ATTR\_HW\_NUM\_LANE, /\* **sai\_uint32\_t** \*/

/\* HW Lane Info – Active Lane Bitmap \*/

SAI\_PORT\_ATTR\_HW\_ LANE\_ACT\_MAP, /\* **sai\_uint32\_t** \*/

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

SAI\_PORT\_ATTR\_BREAKOUTIN\_MODE, /\* **sai\_port\_break\_outin\_mode\_t** ,

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

…

…

# API Flow

## Get System Ports

* sai\_initialize\_switch()
* Application calls SAI\_SWITCH\_ATTR\_HW\_PORT\_LIST attribute get to get the list of SAI ports
* Applications calls Port Attribute GET with Multiple attributes -
  + SAI\_PORT\_ATTR\_HW\_BLOCK\_NUM,
  + SAI\_PORT\_ATTR\_HW\_START\_LANE\_NUM,
  + SAI\_PORT\_ATTR\_HW\_END\_LANE\_NUM,
  + SAI\_PORT\_ATTR\_HW\_NUM\_LANE,
  + SAI\_PORT\_ATTR\_HW\_ LANE\_ACT\_MAP,
  + SAI\_PORT\_ATTR\_BREAKOUTIN\_MODE

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

## BREAKOUT/IN

* Call SAI\_SWITCH\_ATTR\_PORT\_BREAK\_IN\_OUT attribute set to enable FANOUT/IN. The port(s) to FANOUT/IN is passed
* A set of port events are generated with SAI\_PORT\_ADD\_EVENT for the new ports created and SAI\_PORT\_DELETE\_PORT for the existing ports that are invalid
* Applications calls Port Attribute GET with Multiple attributes -
  + SAI\_PORT\_ATTR\_HW\_BLOCK\_NUM,
  + SAI\_PORT\_ATTR\_HW\_START\_LANE\_NUM,
  + SAI\_PORT\_ATTR\_HW\_END\_LANE\_NUM,
  + SAI\_PORT\_ATTR\_HW\_NUM\_LANE,
  + SAI\_PORT\_ATTR\_HW\_ LANE\_ACT\_MAP,
  + SAI\_PORT\_ATTR\_BREAKOUTIN\_MODE

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

SAI\_PORT\_ADD\_EVENT can be used to inform all the SAI ports after switch initialize for the application know of the SAI ports automatically rather than getting the port list thru SAI\_SWITCH\_ATTR\_PORT\_LIST

# Appendix

Detailed explanation of the port numbering with examples. The various port numbers that would be referenced

## Application Port

This 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

## HW Port

This refers to the physical port in the NPU which we use to access a port in the NPU

## SAI Port

SAI is an interface between the Application and the NPU. SAI would provide a set of system ports for the Adapter Host to use. Each system port would be mapped to an Application Port number using its corresponding HW mapping. Internally SAI uses whatever numbering it prefers, the system or SAI port is what the Adapter host would use to interface with SAI

## Logical Port Mapping

Some NPU's may provide what is called as logical port mapping. The NPU SDK may provide a way by which we can map each of the HW port to a logical number. The logical number mapping may/may not be a 1-1 with the Application port. Any access to the NPU would use this logical number and not the actual HW Port number. This flexibility is provided by NPU's like Broadcom and may not be available on all NPU's

## HW Port Mapping

This refers to the mapping between the Application Port and actual HW port

**(Example)**

* An NPU with 8 ports
* The HW port numbers are 16-47
* Each HW port is a native 40G port with 4 lanes and it can be used as 1 40G port or 4 10G ports

The front-panel ports are connected such a way that

**HW Port Mapping**

**Application Port <-> HW Port**

1<->24

2<->16

3<->28

4<->32

5<->44

6<->36

7<->20

8<->40

**NPU Logical Port Mapping**

**NPU Logical Port<->HW Port**

1<->16, 2<->17, 3<->18, 4<->19

5<->20, 6<->21, 7<->22, 8<->23

9<->24, 10<->25, 11<->26, 12<->27

13<->28, 14<->29, 15<->30, 16<->31

17<->32, 18<->33, 19<->34, 20<->35

21<->36, 22<->37, 23<->38, 24<->39

25<->40, 26<->41, 27<->42, 28<->43

29<->44, 30<->45, 31<->46, 32<->47

**SAI port Numbering** - (Example) Starts with 100 in increments of 4. Each SAI implementation can choose it's SAI port numbering scheme. Anyway that number would be transparently used by the Adapter Host and interpreted only within the SAI. So the numbering can be any scheme

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Application Port** | **HW Port** | **NPU Logical Port** | **SAI Port** | **Speed** |
| 1 | 24 | 9 | 100 | 40G |
| 2 | 16 | 1 | 104 | 40G |
| 3 | 28 | 13 | 108 | 40G |
| 4 | 32 | 17 | 112 | 40G |
| 5 | 44 | 29 | 116 | 40G |
| 6 | 36 | 21 | 120 | 40G |
| 7 | 20 | 5 | 124 | 40G |
| 8 | 40 | 25 | 128 | 40G |

The above table gives a good view of how all the port numbers and mapping are interconnected in our example

## **Various Steps Involved**

* Adapter Host has a knowledge of the HW Mapping, Logical mapping for the NPU in the system
* When the Adapter is initialized, the Adapter Host send the "Logical Port Mapping" to the Adapter if applicable for that NPU. The SAI would use that for Switch Initialize. Typically that information would be needed at the time of switch Initialization
* SAI initializes the switch and knows the list of ports it can support
* The Adapter host requests for the list of SAI port. The Adapter Host does the mapping of the Application port to a SAI port using the HW port mapping information and uses the SAI port to access any of the SAI API's that requires port as an input
* Internally SAI may have a mapping of SAI Port to a logical port which corresponds to the HW port passed at the time of SAI port creation

For the example above

**Adapter Host to program application port 4 it would call SAI API with Port Number 112. The SAI would use logical port 17 to call the NPU SDK to program the port**

## Port Mappings

### Adapter Host

* Mapping between Application Port<->HW Port
* Mapping between NPU Logical Port<->HW Port
* Mapping between Application Port<->SAI Port

### Adapter (SAI)

* Mapping between SAI Port<->NPU Logical Port or HW Port

## **Handling 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

* 40G/100G to 10G port
* 10G ports back to 40G/100G

**40G to 10G**

The Adapter host calls SAI to breakout passing the SAI 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 108. SAI handles the request and returns the 4 new SAI ports that the Adapter Host can use - 108, 109, 110, 111

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Application Port** | **HW Port** | **NPU Logical Port** | **SAI Port** | **Speed** |
| 1 | 24 | 9 | 100 | 40G |
| 2 | 16 | 1 | 104 | 40G |
| 3 | 28  29  30  31 | 13  14  15  16 | 108  109  110  111 | 10G  10G  10G  10G |
| 4 | 32 | 17 | 112 | 40G |
| 5 | 44 | 29 | 116 | 40G |
| 6 | 36 | 21 | 120 | 40G |
| 7 | 20 | 5 | 124 | 40G |
| 8 | 40 | 25 | 128 | 40G |

**10G to 40G**

 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 108 to be used as 40G