

|  |
| --- |
|  |

External PHY Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **External PHY Abstraction Interface** |
| **Authors** | **Dell** |
| **Status** | **Draft** |
| **Type** |  |
| **Created** | **8/5/2019** |
| **Version** | **1.0** |

**Contents**

[List of Changes i](#_Toc20210380)

[1 Overview 1](#_Toc20210381)

[1.1 PHY Interfaces 1](#_Toc20210382)

[1.1.1 MII Interface 1](#_Toc20210383)

[1.1.2 SPI Interface 2](#_Toc20210384)

[1.1.3 Power Supply, Clock and Reset 2](#_Toc20210385)

[1.1.4 System Side Interface 2](#_Toc20210386)

[1.1.5 Line Side Interface 2](#_Toc20210387)

[1.2 Use cases of PHY 2](#_Toc20210388)

[1.2.1 GearBox 2](#_Toc20210389)

[1.2.2 Retimer 2](#_Toc20210390)

[1.2.3 Macsec 2](#_Toc20210391)

[1.2.4 Use case examples 2](#_Toc20210392)

[1.3 Functional Blocks of PHY 6](#_Toc20210393)

[1.3.1 Control Interface 6](#_Toc20210394)

[1.3.2 Firmware management 6](#_Toc20210395)

[1.3.3 PHY Configurations 6](#_Toc20210396)

[1.3.4 PHY State 8](#_Toc20210397)

[1.3.5 PHY Statistics 8](#_Toc20210398)

[1.4 Requirements 9](#_Toc20210399)

[1.5 PHY Driver interface 10](#_Toc20210400)

[2 Object model 11](#_Toc20210401)

[2.1 SAI Switch (PHY) Object 12](#_Toc20210402)

[2.2 Lanes 12](#_Toc20210403)

[2.3 SAI Port Object 13](#_Toc20210404)

[2.3.1 SAI Port State 13](#_Toc20210405)

[2.4 SAI Port Connector Object 13](#_Toc20210406)

[3 Specification 15](#_Toc20210407)

[3.1 API Initialization 15](#_Toc20210408)

[3.2 Platform Adaption 17](#_Toc20210409)

[3.2.1 PHY Driver controlling PHY directly by sysfs 17](#_Toc20210410)

[3.2.2 Platform API for PHY Driver to NOS 18](#_Toc20210411)

[3.2.3 Platform API for NOS to PHY 18](#_Toc20210412)

[3.3 Firmware management 19](#_Toc20210413)

[3.4 PHY Initialization 20](#_Toc20210414)

[3.5 Port Configurations 29](#_Toc20210415)

[3.6 Port connector object 38](#_Toc20210416)

[3.7 Warm Boot 39](#_Toc20210417)

[3.8 Configuration example 42](#_Toc20210418)

[3.8.1 PHY1 config file 42](#_Toc20210419)

[3.8.2 PHY2 config file 42](#_Toc20210420)

[3.8.3 Example platform configuration file Configuration example 42](#_Toc20210421)

[3.8.4 Platform initialization from NOS 42](#_Toc20210422)

[3.8.5 Convert Port Ethernet0 from 100G to 40G 53](#_Toc20210423)

[3.8.6 Convert Port Ethernet2 from 100G to 4 10G Front end ports 53](#_Toc20210424)

[3.8.7 Convert Port Ethernet0 from 100G to 4 10G Front end ports 56](#_Toc20210425)

[3.8.8 Example PHY-1 & PHY-2 initialization with Firmware download broadcast 59](#_Toc20210426)

[3.8.9 Example for PHY-1 initialization with PHY hardware access with sysFS 62](#_Toc20210427)

[3.9 Appendix: 63](#_Toc20210428)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
|  | Draft | Dell | 8/5/2019 |
|  | 0.2 | Dell | 9/12/2019 |
|  | 0.3 | Dell | 9/19/2019 |
|  | 1.0 | Dell | 9/23/2019 |
|  |  |  |  |

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.

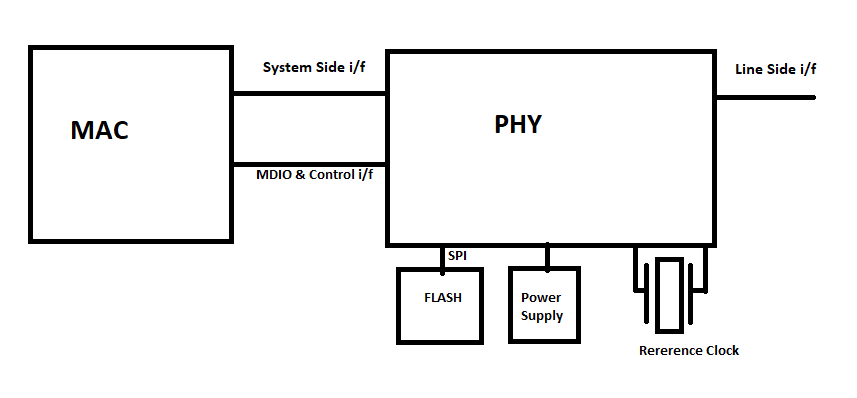
You can review the signed copies of the Open Web Foundation Agreement Version 1.0 for this Specification at <http://opencompute.org/licensing/>, which may also include additional parties to those listed above.

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

The purpose of this document to describe PHY functionality and common interface to manage PHY. PHY support the physical layer functionality. Which is connector between MAC(SerDes) to physical medium such as optical fiber or copper transceivers. Necessity of PHY depends on platform/hardware design. Some platforms may be supported without an PHY(PHY Less) or PHY supports as part of ASIC (Internal PHY) and some cases it might be External PHY. External PHY will be used to serve different purposes like gearbox, retimer, MACSEC and multi gigabit ethernet phy transceivers etc.



## PHY Interfaces

In General External PHY has below interfaces to connect/communicate are peripherals.

### MII Interface

MII(Media Independent inter Interface) is interface to connect MAC and External PHY.

MII has two signal interfaces:

* A Data interface to the Ethernet MAC, for sending and receiving Ethernet frame data.
* A PHY management interface, MDIO, used to read and write the control and status registers of the PHY in order to configure each PHY before operation, download the firmware and to monitor link status during operation.

There can be multiple MDIO interfaces for efficient communications. MDIO frequency must be configured as part of the initialization.

### SPI Interface

SPI (Serial Peripheral Interface) is used for connect the external ROM(Flash) to download the firmware.

### Power Supply, Clock and Reset

Reference clock for RX and TX.

### System Side Interface

System Side is MII interface to connect ASIC MAC and PHY, which is used for packet communication.

### Line Side Interface

Line Side is MII interface to connect PHY and transceivers, which is for external connections and communication.

## Use cases of PHY

### GearBox

A gearbox is essentially a kind of multiplexer/demultiplexer that’s used to convert multiple serial data streams at one rate to multiple streams at another rate.

### Retimer

A retimer is essentially a kind of clock change without effect the rates.

### Macsec

Media Access Control Security (MACsec) is an 802.1AE IEEE industry-standard security technology that provides secure communication for all traffic on Ethernet links. MACsec provides point-to-point security on Ethernet links between directly connected nodes and is capable of identifying and preventing most security threats, including denial of service, intrusion, man-in-the-middle, masquerading, passive wiretapping, and playback attacks.

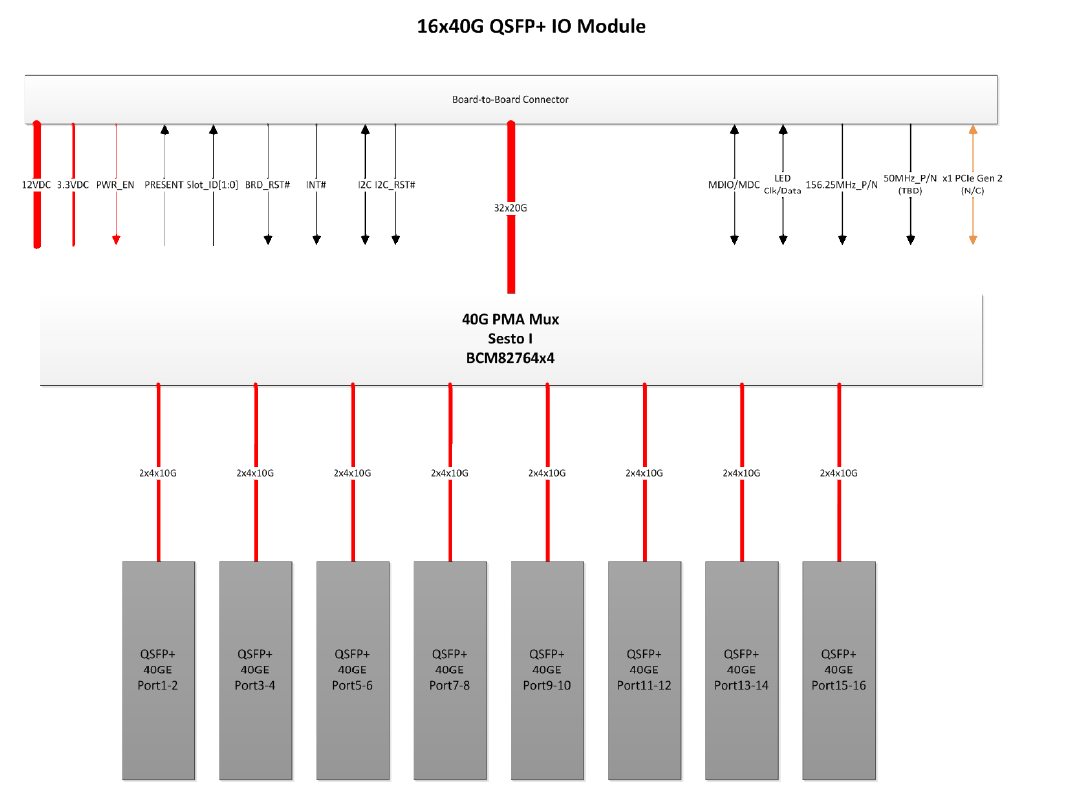
### Use case examples

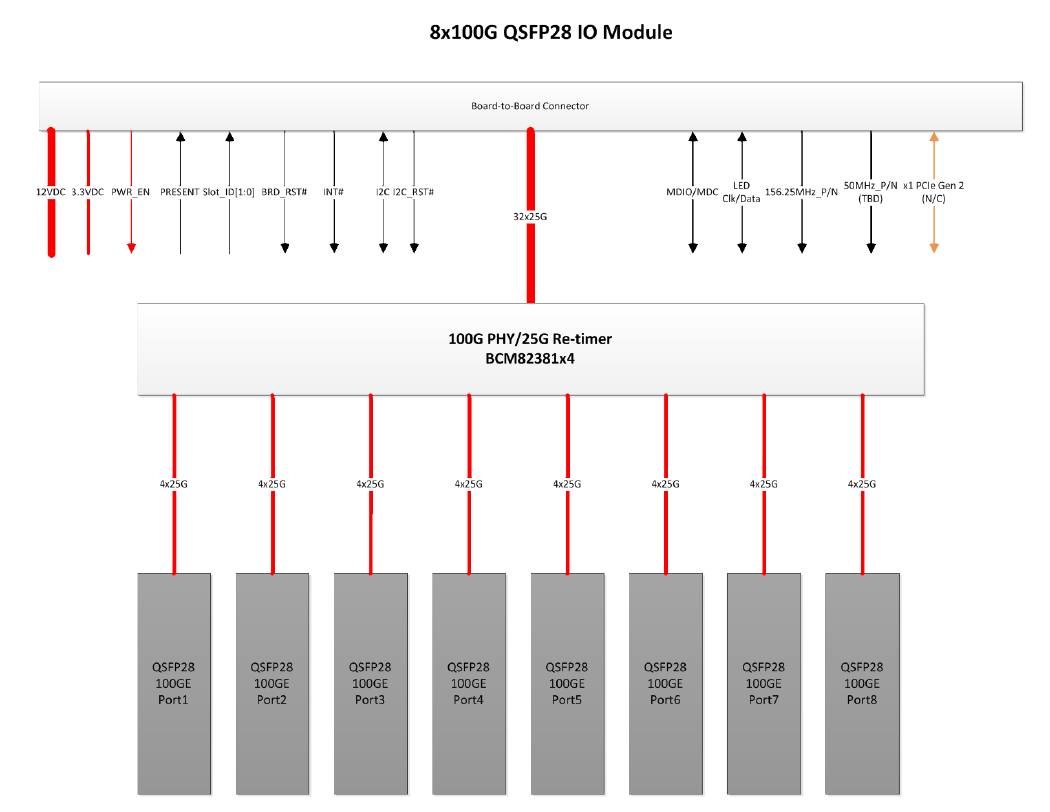
Typical PHY supports all gearbox, retimer, macsec or subset of these. Based on Platform these features will be enable/disable.

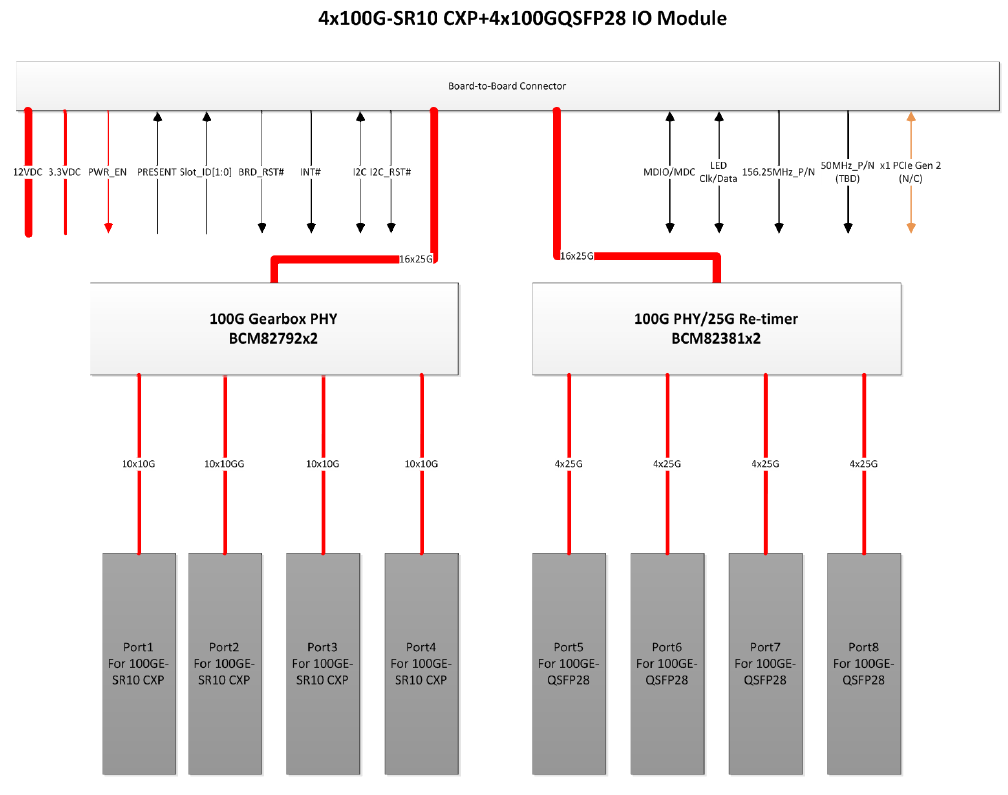
#### Facebook Use case



#### Dell Use case







## Functional Blocks of PHY

### Control Interface

Typical is MDIO interface for access PHY registers. Read/Write register access to control the PHY and some indirect registers to access the system/line side mac sec, PTP and other control blocks.

### Firmware management

Firmware management to bring up PHY is first step. Which requires microcode to be downloaded to PHY. This must be done before any configurations etc.

PHY firmware can be booted by two methods.

From flash or from RAM using MDIO interface, depends on the HW design and mode of operation.

1. From flash chip connected to the external EEPROM using SPI
2. From PHY’s IRAM and DRAM via the MDIO interface

By using MDIO interface firmware download can happen unicast or broadcast method. Broadcast is a method to broadcast firmware on all PHY devices connected to same MDIO bus. In case of unicast each firmware will be downloaded in unicast fashion for each PHY at a time.

PHY firmware download should be done only at first time, to reduce boot time subsequent downloaded should be avoided unless there is PHY firmware version mismatch.

Typical Firmware download methods:

* No Download – Do not download
* Internal - MDIO download
* EPROM – From Flash

Typical Firmware load methods:

* Force – Always download
* Skip - If firmware exits, do not download
* Auto - Version mismatch download firmware

### PHY Configurations

After PHY bootup, To activate PHY or work PHY properly configuration should be done. Below are important configurations.

#### Lane Mapping

Lane mapping is needs to connect the MAC to PHY and PHY to transceivers based on platform HW design. This mapping can be changed

* Init time
* Dynamic change

#### Lane Polarity Settings

To offer routing flexibility on the serdes interface, the phy should be capable to set and get polarity inversions on the line side for both Tx and Rx.

#### Pre-emphasis setting

Pre-emphasis is type of booting the signal to get the proper signal. Should have the capability to set the pre-emphasis on system and line side per lane based and different settings will depend on the mode of operation (NRZ/PAM4).

Typical Analog parameters used to achieve pre-emphasis are

* Pre
* Main
* post
* post2
* Amplitude

#### PHY Enable/disable

PHY to work we need to enable transmit

#### Interface settings

Typical PHY as gear box can have different rates in system side and line. Example 1x100G to 4X25G/4X10G. So interface configuration should be done accordingly.

* Speed
* Interface type
* Reference clock
* Auxiliary modes

#### Auto negotiation

#### FEC

FEC is other important features in PHY due to the interface type differences. In case of gear box, FEC modes can be different in system side and line side based on speed and interface type/user configuration.

#### Loopback

External PHY supports loopback at various layers and system side and line side interfaces.

Loopback types:

* Remote – PMD RX/TX to external connection.
* Local - MAC RX/TX
* Digital loopback – At PMD RX/TX layer after passing rough the all PHY layers.

This loop back options supported per port, port lane, Some these types support only Per lane basis.

#### EEE

PHY may require configuring with EEE.

#### Sync E

Sync E  facilitates the [transference of clock signals](https://en.wikipedia.org/wiki/Time_and_frequency_transfer) over the [Ethernet physical layer](https://en.wikipedia.org/wiki/Ethernet_physical_layer). External PHY may require configuring for Sync E.

#### PTP

PHY support PTP needs controls. scope of this document is not to cover PTP attributes, but this should be extendable based on need.

#### MaC Sec

MacSec should be control by application. Basic control is to enable/disable. Other Attributes related to MacSec in PHY covered in separate proposal.

#### Fanout

This required to convert ports.

#### Cable diagnostics

Should have the capability to enable and disable cable diagnostics if the phy has the capability. For multi gig copper Phy.

#### Link Training

Port link training should be configurable option.

#### PRBS

Should have ability to configure the polynomial (7/9/15/23/31), enable and disable PRBS on both Tx and Rx side. Should have capability to read PRBS counters. Debug puspose

### PHY State

#### PHY Information

This information is to identify PHY information like

* PHY ID
* Firmware revision
* PCS
* PMD – To check signal strength and signal lock information.

#### Port Link Status

Typical ASIC ports link status is notifying the physical connectivity to application, in case of external PHY ASIC port is connected to PHY and PHY is connected to External Transceivers. To get the actual link status applications must get status from ASIC, System side and line side. Once all the link status are up then port oper status should be up.

#### Debug

Need to decide on providing the Debug infra. is shell or what is way?

### PHY Statistics

External PHY will have counters like regular MAC/ASIC ports. Typical counters

* System Side interface counters
* Line Side Interface counters
* FEC counters
* PTP counters
* MacSec counters

## Requirements

Requirement of External PHY Abstraction interface API’s is to support all above functional block of PHY. Some of these settings belongs to PHY initialization, global, per PHY and per lane settings.

As part of PHY Initialization need to provide ASIC port to PHY mapping, PHY address and system side and line side lane information.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Key Attribute | Platform/ Global | Per PHY | Per Port | Per Lane | ASIC Side | System Side IF | Line Side IF | Config Set | Config Get/state Get | Counters |
|  |  |  |  |  |  |  |  |  |  |  |
| Mdio frequency | Y |  |  |  |  |  |  | Y |  |  |
| Register read/Write 32 and 64 bits |  | Y |  |  |  |  |  | Y | Y |  |
| Revision |  | Y |  |  |  |  |  |  | Y |  |
| Firmware version |  | Y |  |  |  |  |  |  | Y |  |
| Firmware download methods |  | Y |  |  |  |  |  |  |  |  |
| Firmware load type |  | Y |  |  |  |  |  |  |  |  |
| Firmware Init BCAST/Unicast |  | Y |  |  |  |  |  |  |  |  |
| Operation Mode (Gear/Retimer/pass) |  | Y |  |  |  |  |  | Y | Y |  |
| Ref Clock/PLL |  | Y |  | Y |  |  |  |  |  |  |
| Admin (Tx cntrol) Enable/disable |  | Y |  | Y |  | Y | Y |  |  |  |
| MaCSec enable/disable |  | Y | Y |  |  |  |  | Y | Y | Y |
| Lane Mapping |  |  | Y | Y |  | Y | Y | Y | Y |  |
| Polarity |  |  |  | Y |  |  | Y | Y |  |  |
| Pre emphasis |  |  |  | Y | Y | Y | Y | Y |  |  |
| Interface type |  |  | Y |  |  | Y | Y | Y | Y | Y |
| Speed |  |  | Y |  |  | Y | Y | Y | Y |  |
| Loopback MAC/System/Line side |  |  |  | Y | Y | Y | Y |  | Y |  |
| FEC |  |  | Y |  | Y | Y | Y | Y | Y | Y |
| Auto Neg |  |  | Y |  | Y | Y | Y | Y | Y |  |
| Link Status |  |  | Y |  | Y | Y | Y |  |  |  |
| Cable diagnostics |  |  | Y |  |  |  | Y |  |  |  |
| PRBS generator/checker |  |  | Y |  |  | Y | Y |  |  |  |
| Eye Scan |  |  | Y |  |  |  |  |  |  |  |
| Fanout-mode(init) |  |  | Y |  |  |  |  |  |  |  |
| Equalization |  |  |  |  |  |  |  |  |  |  |
| Interrupts |  |  |  |  |  |  |  |  |  |  |

## PHY Driver interface

Scope of this document to provide abstraction interface to program external PHY’s.



# Object model

Approach we are taking for PHY interface is object model CRUD. PHY are treated as Layer-1 switch. Proposal to reuse Switch Abstraction Interface (SAI) API’s as common PHY abstraction interface. PHY are treated as Layer-1 switches.

lanes

0

QSFP28

PHY

SWITCH

50G Lane-12

25G Lane-0

25G Lane-1

50G Lane-13

25G Lane-2

25G Lane-3

25G Lane-4

QSFP28

50G Lane-14

25G Lane-6

25G Lane-5

50G Lane-15

25G Lane-7

25G Lane-8

QSFP28

50G Lane-16

50G Lane-17

25G Lane-10

25G Lane-9

25G Lane-11

Pros:

* Every major functional block is defined as object
* Every object will have below API’s
  + Create
  + Delete
  + Set
  + Get
  + Counters
* Each object will have attributes to configure required functionality easily.
* There is no need to add new API to add new functionality.

Primary objects for PHY interface are

## SAI Switch (PHY) Object

This is primary object to control each PHY in platform independently.

This object supports

* PHY Initialization & status
* Firmware information and management
* Global (PHY level) configurations
* Platform Adaption interface

## Lanes

Lane represents physical channel to carry the data. In general, PHY lanes are used to connect to Media and switch. Based on where these lanes are connected will get logical definition as system side and line side lane.

In general, System side lanes to line side lanes are fixed hardware connections inside MUX based on PHY mode of operation. This object provides mapping of this.

Lane numbering: Lane number is local to each PHY. PHY can have multiple lanes, based on requirement lanes are grouped to create one physical port. PHY abstraction driver needs to provide unique number to each lane in PHY to NOS. Based on lane number NOS can determine port is system side port/line side port etc.

**Lane numbering (i.e HW port) local to PHY, should be unique per each lane**.

QSFP

ASIC

PHY

Lane0

Lane4

Lane0

Lane0

Lane1

Lane1

Lane1

Lane5

Lane2

Lane6

Lane20

Lane2

Lane3

Lane7

Lane30

Lane3

Lane Object:

* Provided capability of lane map at ASIC side and line side of PHY.
* Lane polarity flip
* Pre-emphasis
* Dynamic change of system side lane to line side lane cross connect mapping.

Most of Lane parameters are one time during initialization time. To simplify proposal lane level functionality and configurations to addressed in SAI port object and SAI port connector object.

## SAI Port Object

This is representing logical combination of lanes as port. SAI port will provide the configuration option for system and line side port. PHY may require different configuration for system side and line side port. Unlike ASIC, PHY will have 2 sides i.e system and line side port for each front-end port.

PHY driver needs to support of SAI port object for system side with system side lane list and line side with line side lane list. Since PHY lane number is unique in to PHY, NOS will know system side lane numbers and line side lane numbers based on platform file.

### SAI Port State

PHY has system side and line side. Link status is depending on both sides of PHY. NOS needs to consolidate link status from ASIC and PHY port objects. Some PHY’s supports propagating line side link state to system side. When ever system side link status effected ASIC link status will change. So NOS can determine port operation status by ASIC side port state, there is no need to get PHY port state.

Recommendation for PHY driver are

* Support propagation in case hardware supports this capability. But provide initialization time configuration option to enable this or not.
* In case of propagation not supported by PHY then recommendation to notify port state change (SAI\_SWITCH\_ATTR\_PORT\_STATE\_CHANGE\_NOTIFY) by PHY driver for both line and system side.

In case of above options are not supported then NOS will query port state from ASIC, system side and line side to give consolidated status to applications.

## SAI Port Connector Object

This object to create relation between the system side port object and line side port object in PHY driver. Most of the cases PHY internal system side lane map to line side lane map is fixed. But some cases user may need to change this mapping. This Port connector object will be used to present or alter system side lane to line side lanes connection in PHY. PHY driver will make sure traffic from system side lanes going to line side lane numbers.

Below example ASIC Core has 4 serdes lanes, PHY supports 4 system side lanes and 8 line side lanes.

Initially platform initialized with ethernet0 and ethernet4 100G ports. For Ethernet0 physical port system side uses 2x50G(lane 8, 9) and line side uses 4x25G(lane 0,1,2,3). For Ethernet4 physical port system side uses 2x50G(lane 10,11) and line side uses 4x25G(lane 4,5,6,7). Internally PHY set with traffic ingress from 2 system side lanes goes to 4 line side lanes.

Ethernet 0

ASIC Core

Ethernet 4

**PHY**

Ethernet0

Ethernet 4

Now User decide dynamically break out 100G Ethernet0 to 4X25G(ethernet0, ethernet1, ethernet2 and ethernet3) individual ports. Each ASIC lane will be used for each 25G port. To support this combination second physical port will be unusable. This will be remapped as below.

Ethernet0 physical port system side uses 1x25G(lane 8) and line side uses 1x25G(lane 0).

Ethernet1 physical port system side uses 1x25G(lane 9) and line side uses 1x25G(lane 1).

Ethernet2 physical port system side uses 1x25G(lane 10) and line side uses 1x25G(lane 2).

Ethernet3 physical port system side uses 1x25G(lane 11) and line side uses 1x25G(lane 3).

Ethernet0

Ethernet 1

ASIC Core

Ethernet 2

Ethernet 3

ASIC Core

**PHY**

QSFP

QSFP

25G

25G

25G

25G

25G

25G

25G

25G

To remap this internal mapping in PHY driver PHY port connector object will be used. After creating system side and line side port objects each for each physical port, port connector object should be created with members as system side port object id and line side port object id, PHY driver will get system side lane id’s from system side port object and from line side lane id’s from lane side port object and make changes to lane connections if it is modified from their defaults.

This object support from PHY driver is mandatory. This will give view of port.

# Specification

## API Initialization

<https://github.com/opencomputeproject/SAI/blob/master/inc/saistatus.h>

<https://github.com/opencomputeproject/SAI/blob/master/inc/sai.h>

typedef enum \_sai\_api\_t

{

SAI\_API\_UNSPECIFIED = 0, /\*\*< unspecified API \*/

SAI\_API\_SWITCH = 1, /\*\*< sai\_switch\_api\_t \*/

SAI\_API\_PORT = 2, /\*\*< sai\_port\_api\_t \*/

.

.

SAI\_API\_PORT\_CONNECTOR = 40, /\*\*< sai\_phy\_port\_api\_t \*/

} sai\_api\_t;

/\*\*

\* @brief Defines log level

\*/

typedef enum \_sai\_log\_level\_t

{

/\*\* Log Level Debug \*/

SAI\_LOG\_LEVEL\_DEBUG = 0,

/\*\* Log Level Info \*/

SAI\_LOG\_LEVEL\_INFO = 1,

/\*\* Log Level Notice \*/

SAI\_LOG\_LEVEL\_NOTICE = 2,

/\*\* Log level Warning \*/

SAI\_LOG\_LEVEL\_WARN = 3,

/\*\* Log Level Error \*/

SAI\_LOG\_LEVEL\_ERROR = 4,

/\*\* Log Level Critical \*/

SAI\_LOG\_LEVEL\_CRITICAL = 5

} sai\_log\_level\_t;

typedef const char\* (\*sai\_profile\_get\_value\_fn)(

\_In\_ sai\_switch\_profile\_id\_t profile\_id,

\_In\_ const char \*variable);

typedef int (\*sai\_profile\_get\_next\_value\_fn)(

\_In\_ sai\_switch\_profile\_id\_t profile\_id,

\_Out\_ const char \*\*variable,

\_Out\_ const char \*\*value);

/\*\*

\* @brief Method table that contains function pointers for services exposed by

\* adapter host for adapter.

\*/

typedef struct \_sai\_service\_method\_table\_t

{

/\*\*

\* @brief Get variable value given its name

\*/

sai\_profile\_get\_value\_fn profile\_get\_value;

/\*\*

\* @brief Enumerate all the K/V pairs in a profile.

\*

\* Pointer to NULL passed as variable restarts enumeration. Function

\* returns 0 if next value exists, -1 at the end of the list.

\*/

sai\_profile\_get\_next\_value\_fn profile\_get\_next\_value;

} sai\_service\_method\_table\_t;

/\*\*

\* @brief Adapter module initialization call

\*

\* This is NOT for SDK initialization.

\*

\* @param[in] flags Reserved for future use, must be zero

\* @param[in] services Methods table with services provided by adapter host

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

sai\_status\_t sai\_api\_initialize(

\_In\_ uint64\_t flags,

\_In\_ const sai\_service\_method\_table\_t \*services);

/\*\*

\* @brief Retrieve a pointer to the C-style method table for desired SAI

\* functionality as specified by the given sai\_api\_id.

\*

\* @param[in] api SAI API ID

\* @param[out] api\_method\_table Caller allocated method table. The table must

\* remain valid until the sai\_api\_uninitialize() is called.

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

sai\_status\_t sai\_api\_query(

\_In\_ sai\_api\_t api,

\_Out\_ void \*\*api\_method\_table);

/\*\*

\* @brief Uninitialize adapter module. SAI functionalities,

\* retrieved via sai\_api\_query() cannot be used after this call.

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

sai\_status\_t sai\_api\_uninitialize(void);

/\*\*

\* @brief Set log level for SAI API module

\*

\* The default log level is #SAI\_LOG\_LEVEL\_WARN.

\*

\* @param[in] api SAI API ID

\* @param[in] log\_level Log level

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

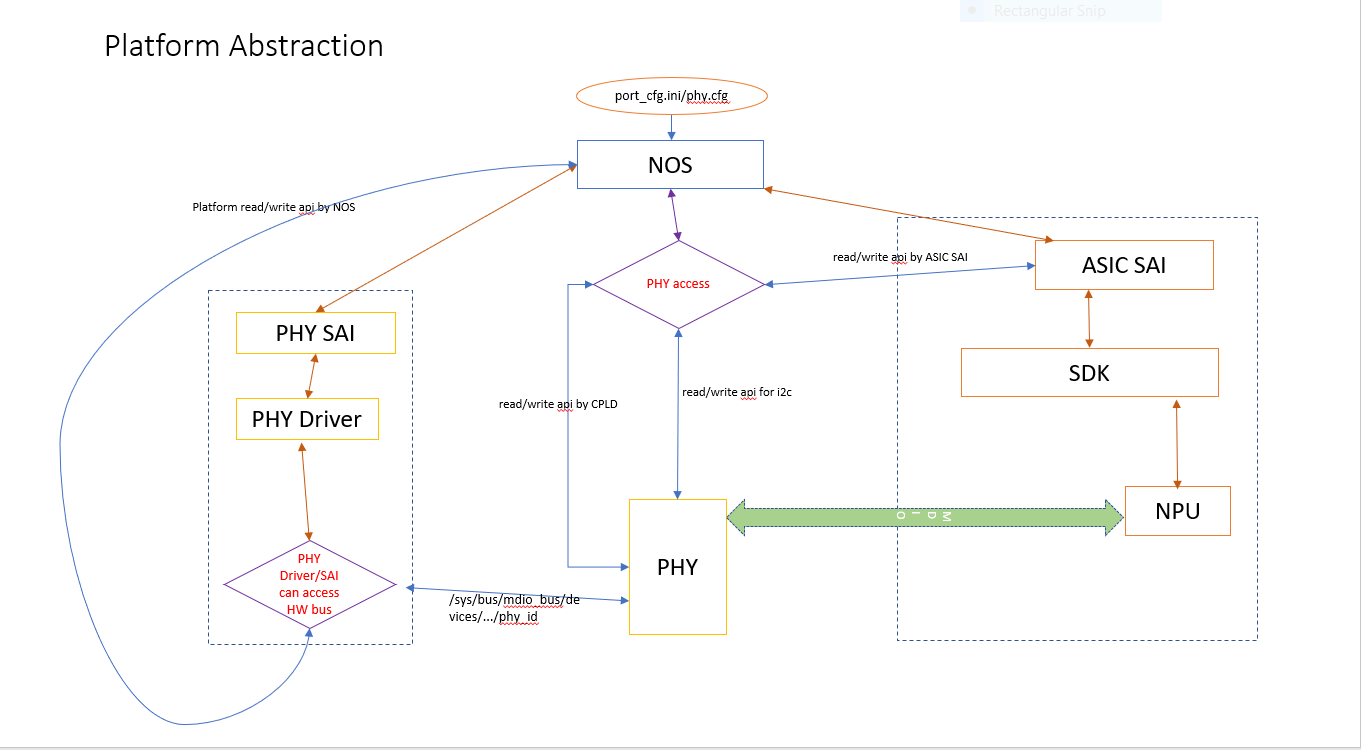
sai\_status\_t sai\_log\_set(

\_In\_ sai\_api\_t api,

\_In\_ sai\_log\_level\_t log\_level);

## Platform Adaption

Platform adaption API needs to provide to interfaces PHY driver with PHY hardware. This API needs to provide by the platform integration. These API will provide base register read /write access to PHY device.



Typical PHY’s are controlled by MDIO interface connected to NPU. But it may be controlled by the I2C/CPLD etc. Proposal should cover both cases. PHY driver can directly control PHY by kernel sysfs, if this is not supported by driver then abstraction will be handled by NOS. Proposal covers both approaches.

### PHY Driver controlling PHY directly by sysfs

PHY driver can access device directly by Linux file system with provided bus(MDIO/I2C etc) address, then PHY driver program PHY. By this option there is no need of NOS to provide call back API’s. If this option is not supported by PHY driver then platform abstraction by call back methods should support.

/\*\*

\* @brief Attribute data for #SAI\_SWITCH\_DEVICE\_ACCESS\_TYPE

\*/

typedef enum \_sai\_switch\_hardware\_access\_bus\_type\_t

{

/\*\* DEVICE is attched to MDIO bus to acccess for programming.\*/

SAI\_SWITCH\_HARDWARE\_ACCESS\_BUS\_TYPE\_MIDO,

/\*\* DEVICE is attched to I2C bus to acccess for programming.\*/

SAI\_SWITCH\_HARDWARE\_ACCESS\_BUS\_TYPE\_I2C,

/\*\* DEVICE is attched to CPLD bus to acccess for programming.\*/

SAI\_SWITCH\_HARDWARE\_ACCESS\_BUS\_TYPE\_CPLD,

} sai\_switch\_hardware\_access\_bus\_type\_t;

### Platform API for PHY Driver to NOS

Provide Platform specific read/write API to PHY SAI. Which will be used to access PHY from PHY driver.

* PHY can access by PHY address
* Each port/lane in PHY can access by port MDIO address.

Implementation of these functions are specific to platform in NOS

/\*\*

\* @brief Provied platform specific PHY register/MDIO read access

\*

\* Passed as a parameter into sai\_initialize\_switch()

\*

\* @param[in] platform\_context Platform context used for bus protection.

\* NOS will provide this to PHY driver, PHY driver will give back

\* to NOS as part of this API.

\* @param[in] phy\_id PHY object id

\* @param[in] device\_addr PHY address(PHY/lane/port MDIO address)

\* @param[in] first\_reg\_addr First register address

\* @param[in] number\_of\_registerss Number of consecutive registers to read

\* @param[Out] reg\_val Register value output

\*/

typedef sai\_status\_t (\*sai\_switch\_register\_read\_fn)(

\_In\_ void \*platform\_context,

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

\_In\_ uint32\_t device\_addr,

\_In\_ uint32\_t start\_reg\_addr,

\_In\_ uint32\_t number\_of\_registers,

\_out\_ uint32\_t \*reg\_val);

/\*\*

\* @brief Provied platform specific PHY register/MDIO write access

\*

\* Passed as a parameter into sai\_initialize\_switch()

\*

\* @param[in] platform\_context Platform context used for bus protection.

\* NOS will provide this to PHY driver, PHY driver will give back

\* to NOS as part of this API.

\* @param[in] phy\_id PHY object id

\* @param[in] device\_addr PHY address(PHY/lane/port MDIO address)

\* @param[in] regiter\_id register address

\* @param[in] first\_reg\_addr First register address

\* @param[in] number\_of\_registerss Number of consecutive registers to write

\* @param[in] reg\_val Register value to be written

\*/

typedef sai\_status\_t (\*sai\_switch\_register\_write\_fn)(

\_In\_ void \*platform\_context,

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

\_In\_ uint32\_t device\_addr,

\_In\_ uint32\_t reg\_addr,

\_In\_ uint32\_t start\_reg\_addr,

\_In\_ uint32\_t number\_of\_registers,

\_In\_ const uint32\_t \*reg\_val);

### Platform API for NOS to PHY

As mentioned PHY can control by NPU, I2C/CPLD.

* NOS will have platform knowledge (platform file) to program PHY.
* Fetch API’s needed for PHY access.
* Call corresponding API’s for PHY register read and write.

#### PHY connected to NPU

ASIC SAI should provide API’s for MDIO read/write as part of switch API query.

{

sai\_switch\_register\_read\_fn mdio\_read;

sai\_switch\_register\_write\_fn mdio\_write;

} sai\_switch\_api\_t;

#### PHY connected to I2C/CPLD

NOS should get read/write API’s form Platform drivers and use it.

## Firmware management

/\*\*

\* @brief Attribute data for #SAI\_SWITCH\_FIRMWARE\_LOAD\_METHOD

\*/

typedef enum \_sai\_switch\_firmware\_load\_method\_t

{

/\*\* Do not download FW. Use already downloaded FW instead \*/

SAI\_SWITCH\_FIRMWARE\_LOAD\_METHOD\_NONE,

/\*\* Download FW internally via MDIO \*/

SAI\_SWITCH\_FIRMWARE\_LOAD\_METHOD\_INTERNAL,

/\*\* Load FW from EEPROM \*/

SAI\_SWITCH\_FIRMWARE\_LOAD\_METHOD\_EEPROM,

} sai\_switch\_firmware\_load\_method\_t;

typedef enum \_sai\_switch\_firmware\_load\_type\_t {

/\*\* Skip firmware download if firmware is already present \*/

SAI\_SWITCH\_FIRMWARE\_LOAD\_SKIP,

/\*\* Always download the firmware specified by firmware load method \*/

SAI\_SWITCH\_FIRMWARE\_LOAD\_FORCE,

/\*\* Check the firmware version. If it is different from current version download the firmware \*/

SAI\_SWITCH\_FIRMWARE\_LOAD\_AUTO,

} sai\_switch\_firmware\_load\_type\_t;

typedef enum \_sai\_switch\_firmware\_broadcast\_method\_t {

/\*\* Firmware downloaded as unicast for each PHY device on MDIO bus \*/

SAI\_SWITCH\_FIRMWARE\_BROADCAST\_NONE,

/\*\* Firmware downloaded as unicast for each PHY device on MDIO bus \*/

SAI\_SWITCH\_FIRMWARE\_BROADCAST\_Enable,

} sai\_switch\_firmware\_broadcast\_method\_t;

Firmware management is multi step process. Firmware download method is broadcast and unicast. Unicast method is direct, Every PHY will download firmware as part of switch create and initializes PHY, there is not dependency with other PHY or BUS.

In case of broadcast method, PHY driver needs below sequence for steps.

* All PHY’s should be out of reset.
* Set download method as broadcast for all PHY connected to same BUS. This will be done as part of create switch with other required parameters like PHY address, FW path etc. But in case of broadcast type PHY is not initialized until FW download initiated. This is like software operation.
* Execute firmware download in any one of the PHY connected to same bus. This step will broadcast firmware to all PHY’s on same bus.
* Stop broadcast on all PHY’s. Since broadcast is enable on BUS, so before end broadcast if we do PHY configuration/initialization will also broadcast.
* Next step is to verify FW status and initialize PHY on all PHY’s.

## PHY Initialization

/\*\*

\* @brief Maximum Hardware ID Length

\*/

#define SAI\_MAX\_HARDWARE\_ID\_LEN 255

/\*\*

\* @brief Maximum Firmware Path Name Length

\*/

#define SAI\_MAX\_FIRMWARE\_PATH\_NAME\_LEN PATH\_MAX

/\*\*

\* @def SAI\_KEY\_INIT\_CONFIG\_FILE

\* Vendor specific and vendor specific format config file for each PHY. This file can have

\* This can have all conifguration parameters which will not change after PHY initlization

\* Like

\* - System and line side lane numbering like

\* 0 - 16 for line side

\* 16- 24 for system side

\* - Lane mapping

\* - Lane swap

\* - Polatiry Flip

\* - Phy mode – Gear box/retimer/passthough

\* - Interfcae mode

\* - IEEE, Fiber etc

\* - Reference clock etc.

\*

\*/

#define SAI\_KEY\_INIT\_CONFIG\_FILE "SAI\_INIT\_CONFIG\_FILE"

/\*\*

\* @def SAI\_KEY\_HW\_PORT\_PROFILE\_ID\_CONFIG\_FILE

\* Vendor specific Configuration file for Hardware Port Profile ID parameters.

\* HW port profile ID can be used to set vendor specific port attributes based on

\* the transceiver type plugged in to the port

\*/

#define SAI\_KEY\_HW\_PORT\_PROFILE\_ID\_CONFIG\_FILE "SAI\_HW\_PORT\_PROFILE\_ID\_CONFIG\_FILE"

/\*\*

\* @brief Attribute data for #SAI\_SWITCH\_ATTR\_OPER\_STATUS

\*/

typedef enum \_sai\_switch\_oper\_status\_t

{

/\*\* Unknown \*/

SAI\_SWITCH\_OPER\_STATUS\_UNKNOWN,

/\*\* Up \*/

SAI\_SWITCH\_OPER\_STATUS\_UP,

/\*\* Down \*/

SAI\_SWITCH\_OPER\_STATUS\_DOWN,

/\*\* Switch encountered a fatal error \*/

SAI\_SWITCH\_OPER\_STATUS\_FAILED,

} sai\_switch\_oper\_status\_t;

/\*\*

\* @brief Attribute Id in sai\_set\_switch\_attribute() and

\* sai\_get\_switch\_attribute() calls

\*/

typedef enum \_sai\_switch\_attr\_t

{

/\*\*

\* @brief Handle for switch profile id.

\*

\* Use this to retrieve the Key-Value pairs as part of switch

\* initialization.

\*

\* @type sai\_uint32\_t

\* @flags CREATE\_ONLY

\* @default 0

\*/

SAI\_SWITCH\_ATTR\_SWITCH\_PROFILE\_ID,

/\*\*

\* @brief Device Information for switch initialization.

\*

\* Hardware information format is based on SAI implementations by vendors.

\* String is NULL terminated. Format is vendor specific.

\* Example: Like PCI location, I2C address, PHY address etc.

\* In case of NULL, First NPU attached to CPU will be initialized.

\* Single NPU case this attribute is optional.

\*

\* @type sai\_s8\_list\_t

\* @flags CREATE\_ONLY

\* @default empty

\*/

SAI\_SWITCH\_ATTR\_SWITCH\_HARDWARE\_INFO,

/\*\*

\* @brief Vendor specific path name of the firmware to load.

\*

\* @type sai\_s8\_list\_t

\* @flags CREATE\_ONLY

\* @default empty

\*/

SAI\_SWITCH\_ATTR\_FIRMWARE\_PATH\_NAME,

/\*\*

\* @brief Set to switch initialization or connect to NPU/SDK.

\*

\* TRUE - Initialize switch/SDK.

\* FALSE - Connect to SDK. This will connect library to the initialized SDK.

\* After this call the capability attributes should be ready for retrieval

\* via sai\_get\_switch\_attribute()

\*

\* @type bool

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY

\*/

SAI\_SWITCH\_ATTR\_INIT\_SWITCH,

/\*\*

\* @brief Platform adaption register/MIDO read API for PHY.

\* callback function passed to the adapter.

\*

\* Use sai\_switch\_register\_read\_fn as read function.

\* This is mandatory function for PHY, if PHY driver not suported MDIO bus

\* access directly by sysfs.

\*

\* @type sai\_pointer\_t sai\_switch\_register\_read\_fn

\* @flags CREATE\_AND\_SET

\* @default NULL

\*/

SAI\_SWITCH\_ATTR\_REGISTER\_READ,

/\*\*

\* @brief Platform adaption register/MDIO write API for PHY.

\* callback function passed to the adapter.

\*

\* Use sai\_switch\_register\_read\_fn as read function.

\* This is mandatory function for PHY, if PHY driver not suported MDIO bus

\* access directly by sysfs.

\* @type sai\_pointer\_t sai\_switch\_register\_write\_fn

\* @flags CREATE\_AND\_SET

\* @default NULL

\*/

SAI\_SWITCH\_ATTR\_REGISTER\_WRITE,

/\*\*

\* @brief Firmware donwload broadcast/unicast

\*

\* Indicates firmware download as unicast for each PHY device on bus.

\*

\* TRUE - BROADCAST

\* FALSE - UNICAST

\*

\* @type bool

\* @flags MANDATORY\_ON\_CREATE

\* @default true

\*/

SAI\_SWITCH\_ATTR\_FIRMWARE\_DOWNLOAD\_BROADCAST,

/\*\*

\* @brief Firmware downlaod method

\*

\* @type sai\_switch\_firmware\_load\_method\_t

\*

\* @flags CREATE\_ONLY

\* @default SAI\_SWITCH\_FIRMWARE\_LOAD\_METHOD\_INTERNAL

\*/

SAI\_SWITCH\_ATTR\_FIRMWARE\_LOAD\_METHOD,

/\*\*

\* @brief Firmware load type auto/force/skip

\*

\* Check the firmware version. If it is different from current version download the firmware and load.

\* Otherwise Always download the firmware specified by firmware load method.

\*

\* @type sai\_switch\_firmware\_load\_type\_t

\* @flags CREATE\_ONLY

\* @default SAI\_SWITCH\_FIRMWARE\_LOAD\_AUTO

\*/

SAI\_SWITCH\_ATTR\_FIRMWARE\_LOAD\_TYPE,

/\*\*

\* @brief Execute Firmware download

\* In case of broadcast method, This attribute should be set on

\* any one of the PHY connected to same bus. Since it broadcast FW will

\* download to all PHY’s.

\*

\* @type bool

\* @flags SET\_ONLY

\* @validonly SAI\_SWITCH\_ATTR\_FIRMWARE\_DOWNLOAD\_BROADCAST == true

\*/

SAI\_SWITCH\_ATTR\_FIRMWARE\_DOWNLOAD\_EXECUTE,

/\*\*

\* @brief End Broadcast on bus

\*

\* Broadcast is enabled for BUS, PHY configurations also will broadcast.

\* End broadcast before initialize PHY’s.

\*

\* @type bool

\* @flags SET\_ONLY

\* @validonly SAI\_SWITCH\_ATTR\_FIRMWARE\_DOWNLOAD\_BROADCAST == true

\*/

SAI\_SWITCH\_ATTR\_FIRMWARE\_BROADCAST\_END,

/\*\*

\* @brief Firmware status verify and complete initializing PHY.

\* NOS should mandatory to set with attribute on switch before doing any other PHY

\* PHY confiurations.

\*

\* @type bool

\* @flags SET\_ONLY

\* @validonly SAI\_SWITCH\_ATTR\_FIRMWARE\_DOWNLOAD\_BROADCAST == true

\*/

SAI\_SWITCH\_ATTR\_FIRMWARE\_VERIFY\_AND\_INIT\_SWITCH,

/\*\*

\* @brief Firmware download and running status

\*

\* Indicates firmware download and running in PHY.

\*

\* TRUE - Firmware running

\* FALSE - Firmware not running.

\*

\* @type bool

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_FIRMWARE\_STATUS,

/\*\*

\* @brief Firmware Major version number

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_FIRMWARE\_MAJOR\_VERSION,

/\*\*

\* @brief Firmware Minor version number

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_FIRMWARE\_MINOR\_VERSION,



/\*\*

\* @brief Switch hardware can access device by file system in driver.

\*

\* @type bool

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_HARDWARE\_ACCESS\_BY\_SYSFS,

/\*\*

\* @brief Switch device access bus type mdio/i2c/cpld

\*

\*

\* @type sai\_switch\_device\_access\_bus\_type\_t

\* @flags MANDATORY\_ON\_CREATE and CREATE\_ONLY

\* @default SAI\_SWITCH\_ATTR\_DEVICE\_ACCESS\_BUS\_MDIO

\* @vlaidnly SAI\_SWITCH\_ATTR\_HARDWARE\_ACCESS\_BY\_SYSFS == true

\*/

/\*\*

\* @brief Switch device access bus type mdio/i2c/cpld

\*

\*

\* @type sai\_switch\_device\_access\_bus\_type\_t

\* @flags MANDATORY\_ON\_CREATE and CREATE\_ONLY

\* @default SAI\_SWITCH\_ATTR\_DEVICE\_ACCESS\_BUS\_MDIO

\* @vlaidnly SAI\_SWITCH\_ATTR\_HARDWARE\_ACCESS\_BY\_SYSFS == true

\*/

SAI\_SWITCH\_ATTR\_HARDWARE\_ACCESS\_BUS\_TYPE,

/\*\*

\* @brief Device address to access MDIO bus in PHY driver. If PHY driver supports this

\* attribute, then it not mandatry for adaptor to pass

\* SAI\_SWITCH\_ATTR\_REGISTER\_WRITE/READ.

\* Platform abstarction is not needed in this case.

\* BUSID:DEVICEID

\*

\* @type sai\_string\_t

\* @flags MANDATORY\_ON\_CREATE and CREATE\_ONLY

\* @vlaidnly SAI\_SWITCH\_ATTR\_HARDWARE\_ACCESS\_BY\_SYSFS == true

\* @default NULL

\*/

SAI\_SWITCH\_ATTR\_HARDWARE\_ACCESS\_ADDRESS,

/\*\*

\* @brief Platform abstarction context gived by adaptor to PHY driver,

\* This will be used to function passed to the adapter context to PHY driver

\* and PHY driver will give this context back to NOS.

\*

\* Use void as pointer.

\*

\* @type sai\_pointer\_t void

\* @flags CREATE\_AND\_SET

\* @default NULL

\* @vlaidnly SAI\_SWITCH\_ATTR\_HARDWARE\_ACCESS\_BY\_SYSFS == true

\*/

SAI\_SWITCH\_ATTR\_PLATFROM\_CONTEXT,

/\*\*

\* @brief Get the port connector list from PHY

\*

\* @type sai\_object\_list\_t

\* @flags READ\_ONLY

\* @objects SAI\_OBJECT\_TYPE\_PORT\_CONNECTOR

\* @default internal

\*/

SAI\_SWITCH\_ATTR\_PORT\_CONNECTOR\_LIST,

/\*\*

\* @brief Support of propogating line side port state to system side port. System side

port state will change ASIC port state. NOS can depends on ASIC port state for

link status.

\*

\* @type bool

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_PORT\_STATE\_PROPOGATE\_FORM\_LINE\_TO\_SYSTEM\_SUPPORTED,

/\*\*

\* @brief Operational status change notification callback

\* function passed to the adapter.

\*

\* Use sai\_switch\_state\_change\_notification\_fn as notification function.

\*

\* @type sai\_pointer\_t sai\_switch\_state\_change\_notification\_fn

\* @flags CREATE\_AND\_SET

\* @default NULL

\*/

SAI\_SWITCH\_ATTR\_SWITCH\_STATE\_CHANGE\_NOTIFY,

/\*\*

\* @brief Shutdown notification callback function passed to the adapter.

\*

\* Use sai\_switch\_shutdown\_request\_notification\_fn as notification function.

\*

\* @type sai\_pointer\_t sai\_switch\_shutdown\_request\_notification\_fn

\* @flags CREATE\_AND\_SET

\* @default NULL

\*/

SAI\_SWITCH\_ATTR\_SWITCH\_SHUTDOWN\_REQUEST\_NOTIFY,

/\*\* @ignore - for backward compatibility \*/

SAI\_SWITCH\_ATTR\_SHUTDOWN\_REQUEST\_NOTIFY = SAI\_SWITCH\_ATTR\_SWITCH\_SHUTDOWN\_REQUEST\_NOTIFY,

/\*\*

\* @brief Port state change notification callback function passed to the adapter.

\*

\* Use sai\_port\_state\_change\_notification\_fn as notification function.

\*

\* @type sai\_pointer\_t sai\_port\_state\_change\_notification\_fn

\* @flags CREATE\_AND\_SET

\* @default NULL

\*/

SAI\_SWITCH\_ATTR\_PORT\_STATE\_CHANGE\_NOTIFY,

/\*\*

\* @brief Number of active(created) ports on the switch

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

/\*\* @ignore - for backward compatibility \*/

SAI\_SWITCH\_ATTR\_PORT\_NUMBER = SAI\_SWITCH\_ATTR\_NUMBER\_OF\_ACTIVE\_PORTS,

/\*\*

\* @brief Maximum number of supported ports on the switch

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_MAX\_NUMBER\_OF\_SUPPORTED\_PORTS,

/\*\*

\* @brief Get the port list

\*

\* @type sai\_object\_list\_t

\* @flags READ\_ONLY

\* @objects SAI\_OBJECT\_TYPE\_PORT

\* @default internal

\*/

SAI\_SWITCH\_ATTR\_PORT\_LIST,

/\*\*

\* @brief List of temperature readings from all sensors.

\*

\* Values in Celsius.

\*

\* @type sai\_s32\_list\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_TEMP\_LIST,

/\*\*

\* @brief The current value of the maximum temperature

\* retrieved from the switch sensors

\*

\* Value in Celsius.

\*

\* @type sai\_int32\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_MAX\_TEMP,

/\*\*

\* @brief The average of temperature readings over all

\* sensors in the switch

\*

\* Value in Celsius.

\*

\* @type sai\_int32\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_AVERAGE\_TEMP,

/\*\*

\* @brief Refresh interval

\*

\* @par The SDK can

\*

\* 1 - Read the counters directly from HW (or)

\* 2 - Cache the counters in SW. Caching is typically done if

\* retrieval of counters directly from HW for each counter

\* read is CPU intensive

\*

\* This setting can be used to

\*

\* 1 - Move from HW based to SW based or Vice versa

\* 2 - Configure the SW counter cache refresh rate

\*

\* Setting a value of 0 enables direct HW based counter read. A

\* non zero value enables the SW cache based and the counter

\* refresh rate.

\*

\* A NPU may support both or one of the option. It would return

\* error for unsupported options

\*

\* Default - 1 sec (SW counter cache)

\*

\* @type sai\_uint32\_t

\* @flags CREATE\_AND\_SET

\* @default 1

\*/

SAI\_SWITCH\_ATTR\_COUNTER\_REFRESH\_INTERVAL,

/\*\*

\* @brief End of attributes

\*/

SAI\_SWITCH\_ATTR\_END,

/\*\* Custom range base value \*/

SAI\_SWITCH\_ATTR\_CUSTOM\_RANGE\_START = 0x10000000,

/\*\* End of custom range base \*/

SAI\_SWITCH\_ATTR\_CUSTOM\_RANGE\_END

} sai\_switch\_attr\_t;

/\*\*

\* @brief Switch shutdown request callback.

\*

\* Adapter DLL may request a shutdown due to an unrecoverable failure

\* or a maintenance operation

\*

\* @objects switch\_id SAI\_OBJECT\_TYPE\_SWITCH

\*

\* @param[in] switch\_id Switch Id

\*/

typedef void (\*sai\_switch\_shutdown\_request\_notification\_fn)(

\_In\_ sai\_object\_id\_t switch\_id);

/\*\*

\* @brief Switch operational state change notification

\*

\* @objects switch\_id SAI\_OBJECT\_TYPE\_SWITCH

\*

\* @param[in] switch\_id Switch Id

\* @param[in] switch\_oper\_status New switch operational state

\*/

typedef void (\*sai\_switch\_state\_change\_notification\_fn)(

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

\_In\_ sai\_switch\_oper\_status\_t switch\_oper\_status);

/\*\*

\* @brief Create switch

\*

\* SDK initialization/connect to SDK. After the call the capability attributes should be

\* ready for retrieval via sai\_get\_switch\_attribute(). Same Switch Object id should be

\* given for create/connect for each NPU.

\*

\* @param[out] switch\_id The Switch Object ID

\* @param[in] attr\_count Number of attributes

\* @param[in] attr\_list Array of attributes

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_create\_switch\_fn)(

\_Out\_ sai\_object\_id\_t \*switch\_id,

\_In\_ uint32\_t attr\_count,

\_In\_ const sai\_attribute\_t \*attr\_list);

/\*\*

\* @brief Remove/disconnect Switch

\*

\* Release all resources associated with currently openled switch

\*

\* @param[in] switch\_id The Switch id

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_remove\_switch\_fn)(

\_In\_ sai\_object\_id\_t switch\_id);

/\*\*

\* @brief Set switch attribute value

\*

\* @param[in] switch\_id Switch id

\* @param[in] attr Switch attribute

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_set\_switch\_attribute\_fn)(

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

\_In\_ const sai\_attribute\_t \*attr);

/\*\*

\* @brief Get switch attribute value

\*

\* @param[in] switch\_id Switch id

\* @param[in] attr\_count Number of attributes

\* @param[inout] attr\_list Array of switch attributes

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_get\_switch\_attribute\_fn)(

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

\_In\_ uint32\_t attr\_count,

\_Inout\_ sai\_attribute\_t \*attr\_list);

/\*\*

\* @brief Switch method table retrieved with sai\_api\_query()

\*/

typedef struct \_sai\_switch\_api\_t

{

sai\_create\_switch\_fn create\_switch;

sai\_remove\_switch\_fn remove\_switch;

sai\_set\_switch\_attribute\_fn set\_switch\_attribute;

sai\_get\_switch\_attribute\_fn get\_switch\_attribute;

sai\_switch\_register\_read\_fn mdio\_read;

sai\_switch\_register\_write\_fn mdio\_write;

} sai\_switch\_api\_t;

## Port Configurations

/\*\*

\* @brief Attribute data for #SAI\_PORT\_ATTR\_TYPE

\*/

typedef enum \_sai\_port\_type\_t

{

/\*\* Actual port. N.B. Different from the physical port. \*/

SAI\_PORT\_TYPE\_LOGICAL,

/\*\* CPU Port \*/

SAI\_PORT\_TYPE\_CPU,

} sai\_port\_type\_t;

/\*\*

\* @brief Attribute data for #SAI\_PORT\_ATTR\_OPER\_STATUS

\*/

typedef enum \_sai\_port\_oper\_status\_t

{

/\*\* Unknown \*/

SAI\_PORT\_OPER\_STATUS\_UNKNOWN,

/\*\* Up \*/

SAI\_PORT\_OPER\_STATUS\_UP,

/\*\* Down \*/

SAI\_PORT\_OPER\_STATUS\_DOWN,

/\*\* Test Running \*/

SAI\_PORT\_OPER\_STATUS\_TESTING,

/\*\* Not Present \*/

SAI\_PORT\_OPER\_STATUS\_NOT\_PRESENT

} sai\_port\_oper\_status\_t;

/\*\*

\* @brief Defines the operational status of the port

\*/

typedef struct \_sai\_port\_oper\_status\_notification\_t

{

/\*\*

\* @brief Port id.

\*

\* @objects SAI\_OBJECT\_TYPE\_PORT, SAI\_OBJECT\_TYPE\_BRIDGE\_PORT, SAI\_OBJECT\_TYPE\_LAG

\*/

sai\_object\_id\_t port\_id;

/\*\* Port operational status \*/

sai\_port\_oper\_status\_t port\_state;

} sai\_port\_oper\_status\_notification\_t;

/\*\*

\* @brief Attribute data for #SAI\_PORT\_ATTR\_INTERNAL\_LOOPBACK\_MODE

\*/

typedef enum \_sai\_port\_internal\_loopback\_mode\_t

{

/\*\* Disable internal loopback \*/

SAI\_PORT\_INTERNAL\_LOOPBACK\_MODE\_NONE,

/\*\* Port internal loopback at PHY module \*/

SAI\_PORT\_INTERNAL\_LOOPBACK\_MODE\_PHY,

/\*\* Port internal loopback at MAC module \*/

SAI\_PORT\_INTERNAL\_LOOPBACK\_MODE\_MAC

} sai\_port\_internal\_loopback\_mode\_t;

/\*\*

\* @brief Attribute data for #SAI\_PORT\_ATTR\_MEDIA\_TYPE

\*/

typedef enum \_sai\_port\_media\_type\_t

{

/\*\* Media not present \*/

SAI\_PORT\_MEDIA\_TYPE\_NOT\_PRESENT,

/\*\* Media type not known \*/

SAI\_PORT\_MEDIA\_TYPE\_UNKNOWN,

/\*\* Media type fiber. Remote advertise medium information as fiber \*/

SAI\_PORT\_MEDIA\_TYPE\_FIBER,

/\*\* Media type copper. Remote advertise medium information as copper \*/

SAI\_PORT\_MEDIA\_TYPE\_COPPER,

/\*\* Back plane \*/

SAI\_PORT\_MEDIA\_TYPE\_BACKPLANE,

} sai\_port\_media\_type\_t;

/\*\*

\* @brief Breakout Mode types based on number

\* of SerDes lanes used in a port

\*/

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

{

/\*\* 1 lane breakout Mode \*/

SAI\_PORT\_BREAKOUT\_MODE\_TYPE\_1\_LANE = 0,

/\*\* 2 lanes breakout Mode \*/

SAI\_PORT\_BREAKOUT\_MODE\_TYPE\_2\_LANE = 1,

/\*\* 4 lanes breakout Mode \*/

SAI\_PORT\_BREAKOUT\_MODE\_TYPE\_4\_LANE = 2,

/\*\* Breakout mode max count \*/

SAI\_PORT\_BREAKOUT\_MODE\_TYPE\_MAX

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

/\*\*

\* @brief Attribute data for #SAI\_PORT\_ATTR\_FEC\_MODE

\*/

typedef enum \_sai\_port\_fec\_mode\_t

{

/\*\* No FEC \*/

SAI\_PORT\_FEC\_MODE\_NONE,

/\*\* Enable RS-FEC - 25G, 50G, 100G ports \*/

SAI\_PORT\_FEC\_MODE\_RS,

/\*\* Enable FC-FEC - 10G, 25G, 40G, 50G ports \*/

SAI\_PORT\_FEC\_MODE\_FC,

/\* Need to enhance for other type of FEC \*/

} sai\_port\_fec\_mode\_t;

/\*\*

\* @brief Attribute data for #SAI\_PORT\_ATTR\_INTERFCE\_TYPE

\* Used for selecting electrical interface with specific electrical pin and signal quality

\*/

typedef enum \_sai\_port\_interface\_type\_t

{

SAI\_PORT\_INTERFCAE\_TYPE\_NONE,

SAI\_PORT\_INTERFCAE\_TYPE\_SR,

SAI\_PORT\_INTERFCAE\_TYPE\_SR4,

SAI\_PORT\_INTERFCAE\_TYPE\_CR,

SAI\_PORT\_INTERFCAE\_TYPE\_CR4,

SAI\_PORT\_INTERFCAE\_TYPE\_KR,

SAI\_PORT\_INTERFCAE\_TYPE\_KR4,

SAI\_PORT\_INTERFCAE\_TYPE\_LR,

} sai\_port\_interface\_type\_t;

/\*\*

\* @brief Attribute Id in sai\_set\_port\_attribute() and

\* sai\_get\_port\_attribute() calls

\*/

typedef enum \_sai\_port\_attr\_t

{

/\*\*

\* @brief Start of attributes

\*/

SAI\_PORT\_ATTR\_START,

/\* READ-ONLY \*/

/\*\*

\* @brief Port Type

\*

\* @type sai\_port\_type\_t

\* @flags READ\_ONLY

\*/

SAI\_PORT\_ATTR\_TYPE = SAI\_PORT\_ATTR\_START,

/\*\*

\* @brief Operational Status

\*

\* @type sai\_port\_oper\_status\_t

\* @flags READ\_ONLY

\*/

SAI\_PORT\_ATTR\_OPER\_STATUS,

/\*\*

\* @brief Breakout mode(s) supported

\*

\* @type sai\_s32\_list\_t sai\_port\_breakout\_mode\_type\_t

\* @flags READ\_ONLY

\*/

SAI\_PORT\_ATTR\_SUPPORTED\_BREAKOUT\_MODE\_TYPE,

/\*\*

\* @brief Current breakout mode

\*

\* @type sai\_port\_breakout\_mode\_type\_t

\* @flags READ\_ONLY

\*/

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

/\*\*

\* @brief Query list of supported port speed(full-duplex) in Mbps

\*

\* @type sai\_u32\_list\_t

\* @flags READ\_ONLY

\*/

SAI\_PORT\_ATTR\_SUPPORTED\_SPEED,

/\*\*

\* @brief Query list of supported port FEC mode

\*

\* @type sai\_s32\_list\_t sai\_port\_fec\_mode\_t

\* @flags READ\_ONLY

\*/

SAI\_PORT\_ATTR\_SUPPORTED\_FEC\_MODE,

/\*\*

\* @brief Query auto-negotiation support

\*

\* @type bool

\* @flags READ\_ONLY

\*/

SAI\_PORT\_ATTR\_SUPPORTED\_AUTO\_NEG\_MODE,

/\*\*

\* @brief Query port supported MEDIA type

\*

\* @type sai\_port\_media\_type\_t

\* @flags READ\_ONLY

\*/

SAI\_PORT\_ATTR\_SUPPORTED\_MEDIA\_TYPE,

/\*\*

\* @brief Operational speed in Mbps

\*

\* If port is down, the returned value should be zero.

\* If auto negotiation is on, the returned value should be the negotiated speed.

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_PORT\_ATTR\_OPER\_SPEED,

/\* READ-WRITE \*/

/\*\*

\* @brief Hardware Lane list

\*

\* @type sai\_u32\_list\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY | KEY

\*/

SAI\_PORT\_ATTR\_HW\_LANE\_LIST,

/\*\*

\* @brief Speed in Mbps

\*

\* On get, returns the configured port speed.

\*

\* @type sai\_uint32\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_AND\_SET

\*/

SAI\_PORT\_ATTR\_SPEED,

/\*\*

\* @brief Auto Negotiation configuration

\*

\* @type bool

\* @flags CREATE\_AND\_SET

\* @default false

\*/

SAI\_PORT\_ATTR\_AUTO\_NEG\_MODE,

/\*\*

\* @brief Admin Mode

\*

\* @type bool

\* @flags CREATE\_AND\_SET

\* @default false

\*/

SAI\_PORT\_ATTR\_ADMIN\_STATE,

/\*\*

\* @brief Media Type

\*

\* @type sai\_port\_media\_type\_t

\* @flags CREATE\_AND\_SET

\* @default SAI\_PORT\_MEDIA\_TYPE\_NOT\_PRESENT

\*/

SAI\_PORT\_ATTR\_MEDIA\_TYPE,

/\*\*

\* @brief INterfcae Type

\*

\* @type sai\_port\_interface\_type\_t

\* @flags CREATE\_AND\_SET

\* @default internal

\*/

SAI\_PORT\_ATTR\_INTERFACE\_TYPE,

/\*\*

\* @brief Port Ref clock in Mhz

\*

\* @type sai\_float\_t

\* @flags CREATE\_ONLY

\* @default ????

\*

\*/

SAI\_PORT\_ATTR\_REF\_CLOCK,

/\*\*

\* @brief Port PRBS Polynomial

\*

\* @type sai\_uint32\_t

\* @flags CREATE\_AND\_SET

\* @default internal

\*/

SAI\_PORT\_ATTR\_PRBS\_POLYNOMIAL,

/\*\*

\* @brief Port failver - Need to look hwo to define this(ashok)

\*

\* @type sai\_uint32\_t

\* @flags CREATE\_AND\_SET

\* @default ??

\*/

SAI\_PORT\_ATTR\_PRBS\_POLYNOMIAL,

/\*\*

\* @brief Forward Error Correction (FEC) control

\*

\* @type sai\_port\_fec\_mode\_t

\* @flags CREATE\_AND\_SET

\* @default SAI\_PORT\_FEC\_MODE\_NONE

\*/

SAI\_PORT\_ATTR\_FEC\_MODE,

/\*\*

\* @brief Port serdes control pre-emphasis

\*

\* List of port serdes pre-emphasis values. The values are of type sai\_u32\_list\_t

\* where the count is number lanes in a port and the list specifies list of values

\* to be applied to each lane.

\*

\* @type sai\_u32\_list\_t

\* @flags CREATE\_AND\_SET

\* @default internal

\*/

SAI\_PORT\_ATTR\_SERDES\_PREEMPHASIS,

/\*\*

\* @brief Port serdes control idriver

\*

\* List of port serdes idriver values. The values are of type sai\_u32\_list\_t

\* where the count is number lanes in a port and the list specifies list of values

\* to be applied to each lane.

\*

\* @type sai\_u32\_list\_t

\* @flags CREATE\_AND\_SET

\* @default internal

\*/

SAI\_PORT\_ATTR\_SERDES\_IDRIVER,

/\*\*

\* @brief Port serdes control ipredriver

\*

\* List of port serdes ipredriver values. The values are of type sai\_u32\_list\_t

\* where the count is number lanes in a port and the list specifies list of values

\* to be applied to each lane.

\*

\* @type sai\_u32\_list\_t

\* @flags CREATE\_AND\_SET

\* @default internal

\*/

SAI\_PORT\_ATTR\_SERDES\_IPREDRIVER,

/\*\*

\* @brief Enable/Disable Port Link Training

\*

\* @type bool

\* @flags CREATE\_AND\_SET

\* @default false

\*/

SAI\_PORT\_ATTR\_LINK\_TRAINING\_ENABLE,

/\*\*

\* @brief End of attributes

\*/

SAI\_PORT\_ATTR\_END,

/\*\* Custom range base value \*/

SAI\_PORT\_ATTR\_CUSTOM\_RANGE\_START = 0x10000000,

/\*\* End of custom range base \*/

SAI\_PORT\_ATTR\_CUSTOM\_RANGE\_END

} sai\_port\_attr\_t;

/\*\*

\* @brief Port counter IDs in sai\_get\_port\_stats() call

\*/

typedef enum \_sai\_port\_stat\_t

{

/\*\* SAI port stat if in octets \*/

SAI\_PORT\_STAT\_IF\_IN\_OCTETS,

/\*\* SAI port stat if in ucast pkts \*/

SAI\_PORT\_STAT\_IF\_IN\_UCAST\_PKTS,

.

.

} sai\_port\_stat\_t;

/\*\*

\* @brief Create port

\*

\* @param[out] port\_id Port id

\* @param[in] switch\_id Switch id

\* @param[in] attr\_count Number of attributes

\* @param[in] attr\_list Array of attributes

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_create\_port\_fn)(

\_Out\_ sai\_object\_id\_t \*port\_id,

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

\_In\_ uint32\_t attr\_count,

\_In\_ const sai\_attribute\_t \*attr\_list);

/\*\*

\* @brief Remove port

\*

\* @param[in] port\_id Port id

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_remove\_port\_fn)(

\_In\_ sai\_object\_id\_t port\_id);

/\*\*

\* @brief Set port attribute value.

\*

\* @param[in] port\_id Port id

\* @param[in] attr Attribute

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_set\_port\_attribute\_fn)(

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

\_In\_ const sai\_attribute\_t \*attr);

/\*\*

\* @brief Get port attribute value.

\*

\* @param[in] port\_id Port id

\* @param[in] attr\_count Number of attributes

\* @param[inout] attr\_list Array of attributes

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_get\_port\_attribute\_fn)(

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

\_In\_ uint32\_t attr\_count,

\_Inout\_ sai\_attribute\_t \*attr\_list);

/\*\*

\* @brief Get port statistics counters. Deprecated for backward compatibility.

\*

\* @param[in] port\_id Port id

\* @param[in] number\_of\_counters Number of counters in the array

\* @param[in] counter\_ids Specifies the array of counter ids

\* @param[out] counters Array of resulting counter values.

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_get\_port\_stats\_fn)(

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

\_In\_ uint32\_t number\_of\_counters,

\_In\_ const sai\_stat\_id\_t \*counter\_ids,

\_Out\_ uint64\_t \*counters);

/\*\*

\* @brief Get port statistics counters extended.

\*

\* @param[in] port\_id Port id

\* @param[in] number\_of\_counters Number of counters in the array

\* @param[in] counter\_ids Specifies the array of counter ids

\* @param[in] mode Statistics mode

\* @param[out] counters Array of resulting counter values.

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_get\_port\_stats\_ext\_fn)(

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

\_In\_ uint32\_t number\_of\_counters,

\_In\_ const sai\_stat\_id\_t \*counter\_ids,

\_In\_ sai\_stats\_mode\_t mode,

\_Out\_ uint64\_t \*counters);

/\*\*

\* @brief Clear port statistics counters.

\*

\* @param[in] port\_id Port id

\* @param[in] number\_of\_counters Number of counters in the array

\* @param[in] counter\_ids Specifies the array of counter ids

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_clear\_port\_stats\_fn)(

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

\_In\_ uint32\_t number\_of\_counters,

\_In\_ const sai\_stat\_id\_t \*counter\_ids);

/\*\*

\* @brief Clear port's all statistics counters.

\*

\* @param[in] port\_id Port id

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_clear\_port\_all\_stats\_fn)(

\_In\_ sai\_object\_id\_t port\_id);

/\*\*

\* @brief Port state change notification

\*

\* Passed as a parameter into sai\_initialize\_switch()

\*

\* @count data[count]

\*

\* @param[in] count Number of notifications

\* @param[in] data Array of port operational status

\*/

typedef void (\*sai\_port\_state\_change\_notification\_fn)(

\_In\_ uint32\_t count,

\_In\_ const sai\_port\_oper\_status\_notification\_t \*data);

\*\*

\* @brief Port methods table retrieved with sai\_api\_query()

\*/

typedef struct \_sai\_port\_api\_t

{

sai\_create\_port\_fn create\_port;

sai\_remove\_port\_fn remove\_port;

sai\_set\_port\_attribute\_fn set\_port\_attribute;

sai\_get\_port\_attribute\_fn get\_port\_attribute;

sai\_get\_port\_stats\_fn get\_port\_stats;

sai\_get\_port\_stats\_ext\_fn get\_port\_stats\_ext;

sai\_clear\_port\_stats\_fn clear\_port\_stats;

sai\_clear\_port\_all\_stats\_fn clear\_port\_all\_stats;

} sai\_port\_api\_t;

## Port connector object

/\*\*

\* @brief Attribute Id in sai\_set\_port\_connector\_attribute() and

\* sai\_get\_port\_connector\_attribute() calls

\*/

typedef enum \_sai\_port\_connector\_attr\_t

{

/\*\*

\* @brief Start of attributes

\*/

SAI\_PORT\_CONNECTOR\_ATTR\_START,

/\*\*

\* @brief System side port id

\*

\* @type sai\_object\_id\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY

\* @objects SAI\_OBJECT\_TYPE\_PORT

\*/

SAI\_PORT\_CONNECTOR\_ATTR\_SYSTEMSIDE\_PORT\_ID = SAI\_PORT\_CONNECTOR\_ATTR\_START,

/\*\*

\* @brief Line Side port id

\*

\* @type sai\_object\_id\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY

\* @objects SAI\_OBJECT\_TYPE\_PORT

\*/

SAI\_PORT\_CONNECTOR\_ATTR\_SYSTEMSIDE\_PORT\_ID,

/\*\*

\* @brief End of attributes

\*/

SAI\_PORT\_CONNECTOR\_ATTR\_END,

} sai\_port\_connector\_attr\_t;

/\*\*

\* @brief Create port connector

\* @param[out] port\_conn\_id Port connector id

\* @param[in] phy\_id Switch id

\* @param[in] attr\_count Number of attributes

\* @param[in] attr\_list Array of attributes

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_create\_port\_connector\_fn)(

\_Out\_ sai\_object\_id\_t \*phy\_port\_id,

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

\_In\_ uint32\_t attr\_count,

\_In\_ const sai\_attribute\_t \*attr\_list);

/\*\*

\* @brief Remove port connector

\*

\* @param[in] port\_conn\_id port connction id

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_remove\_port\_connector\_fn)(

\_In\_ sai\_object\_id\_t port\_conn\_id);

/\*\*

\* @brief Port connectoor methods table retrieved with sai\_api\_query()

\*/

typedef struct \_sai\_port\_connector\_api\_t

{

sai\_create\_port\_connector\_fn create\_port\_connector;

sai\_remove\_port\_connector\_fn remove\_port\_connector;

} sai\_port\_connector\_api\_t;

## Warm Boot

/\*\*

\* @brief Attribute data for #SAI\_SWITCH\_ATTR\_RESTART\_TYPE

\*/

typedef enum \_sai\_switch\_restart\_type\_t

{

/\*\* NPU doesn't support warmboot \*/

SAI\_SWITCH\_RESTART\_TYPE\_NONE = 0,

/\*\* Planned restart only \*/

SAI\_SWITCH\_RESTART\_TYPE\_PLANNED = 1,

/\*\* Both planned and unplanned restart \*/

SAI\_SWITCH\_RESTART\_TYPE\_ANY = 2,

} sai\_switch\_restart\_type\_t;

/\*\*

\* @def SAI\_KEY\_BOOT\_TYPE

\*

\* 0: cold boot. Initialize NPU and external phys.

\* 1: warm boot. Do not re-initialize NPU or external phys, reconstruct SAI/SDK state from stored state.

\* 2: fast boot. Only initialize NPU. SAI/SDK state should not be persisted except for those related

\* to physical port attributes such as SPEED, AUTONEG mode, admin state, operational status.

\*/

#define SAI\_KEY\_BOOT\_TYPE "SAI\_BOOT\_TYPE"

/\*\*

\* @def SAI\_KEY\_WARM\_BOOT\_READ\_FILE

\* The file to recover SAI/NPU state from

\*/

#define SAI\_KEY\_WARM\_BOOT\_READ\_FILE "SAI\_WARM\_BOOT\_READ\_FILE"

/\*\*

\* @def SAI\_KEY\_WARM\_BOOT\_WRITE\_FILE

\* The file to write SAI/NPU state to

\*/

#define SAI\_KEY\_WARM\_BOOT\_WRITE\_FILE "SAI\_WARM\_BOOT\_WRITE\_FILE"

/\*\*

\* @brief Set Type of reboot WARM/COLD

\*

\* Indicates controlled warm restart.

\* Since warm restart can be caused by crash

\* (therefore there are no guarantees for this call),

\* this hint is really a performance optimization.

\* This hint is set as part of the shutdown sequence, before boot.

\* TRUE - Warm Reboot

\* FALSE - Cold Reboot

\*

\* @type bool

\* @flags CREATE\_AND\_SET

\* @default false

\*/

SAI\_SWITCH\_ATTR\_RESTART\_WARM,

/\*\*

\* @brief Warm boot recovery

\*

\* Start warm boot recovery when set to true

\* This hint is set after boot.

\* In case of host adapter restart, host adapter can pass boot type in

\* #SAI\_KEY\_BOOT\_TYPE. In case of host adapter recovery, host adapter can

\* pass a hint about the boot type and recovery, in this flag.

\*

\* @type bool

\* @flags CREATE\_AND\_SET

\* @default false

\*/

SAI\_SWITCH\_ATTR\_WARM\_RECOVER,

/\*\*

\* @brief Type of restart supported

\*

\* @type sai\_switch\_restart\_type\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_RESTART\_TYPE,

/\*\*

\* @brief Minimum interval of time required by SAI for planned restart in milliseconds.

\*

\* Will be 0 for #SAI\_SWITCH\_RESTART\_TYPE\_NONE. The Host Adapter will have to

\* wait for this minimum interval of time before it decides to bring down

\* SAI due to initialize failure.

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_MIN\_PLANNED\_RESTART\_INTERVAL,

/\*\*

\* @brief Nonvolatile storage required by both SAI and NPU in KB

\*

\* Will be 0 for #SAI\_SWITCH\_RESTART\_TYPE\_NONE.

\*

\* @type sai\_uint64\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_NV\_STORAGE\_SIZE,

/\*\*

\* @brief Instruct SAI to execute switch pre-shutdown

\*

\* Indicates controlled switch pre-shutdown as first step of warm shutdown.

\* This hint is optional, SAI application could skip this step and

\* go directly to warm shutdown.

\* This hint should be ignored, if at the time SAI receives this hint,

\* SAI\_SWITCH\_ATTR\_RESTART\_WARM is NOT already set to TRUE.

\* The scope of pre-shutdown is to backup SAI/SDK data, but leave CPU port

\* active for some final control plane traffic to go out.

\* TRUE - Execute switch pre-shutdown for warm shutdown

\* FALSE - No-op, does NOT mean cancelling already executed pre-shutdown

\*

\* @type bool

\* @flags CREATE\_AND\_SET

\* @default false

\*/

SAI\_SWITCH\_ATTR\_PRE\_SHUTDOWN,

## Configuration example

Configuration for below example platform to configure ASIC and PHY.

* ASIC ports connected to 2 External PHY’s.
* PHY-1 supports 4 system side and 8-line side lanes.
* PHY-2 supports 4 system side and 4-line side lanes.
* 2 Physical lanes in ASIC connected to 2 physical lanes of system side in PHY.
* 4 physical line side lanes are connected to QSFP28 port.
* System side lanes supports 50G/20G speed per lane
* Line side lanes supports 25G/10G speed per lane

### PHY1 config file

Format is vendor specific, below is examples



### PHY2 config file



### Example platform configuration file Configuration example

* NOS platform file contains hardware design information
* Number of ASIC’s which library is controlling
* Number of PHY’s which vendor PHY and what library controlling
* Port information like
  + ASIC HW port list
  + PHY information
  + PHY system side HW lane list and speed
  + PHY line side HW lane list and speed



### Platform initialization from NOS

After reading platform configuration file start initializing ASIC’s, PHY’s and ports in system.

#### Example ASIC initialization

typedef std::map<std::string, std::string> sai\_kv\_pair\_t;

typedef std::map<std::string, std::string>::iterator kv\_iter;

static sai\_kv\_pair\_t kvpair;

const char\* profile\_get\_value(sai\_switch\_profile\_id\_t profile\_id,

const char\* variable)

{

kv\_iter kviter;

std::string key;

if (variable == NULL)

return NULL;

key = variable;

kviter = kvpair.find(key);

if (kviter == kvpair.end()) {

return NULL;

}

return kviter->second.c\_str();

}

int profile\_get\_next\_value(sai\_switch\_profile\_id\_t profile\_id,

const char\*\* variable,

const char\*\* value)

{

kv\_iter kviter;

std::string key;

if (variable == NULL || value == NULL) {

return -1;

}

if (\*variable == NULL) {

if (kvpair.size() < 1) {

return -1;

}

kviter = kvpair.begin();

} else {

key = \*variable;

kviter = kvpair.find(key);

if (kviter == kvpair.end()) {

return -1;

}

kviter++;

if (kviter == kvpair.end()) {

return -1;

}

}

\*variable = (char \*)kviter->first.c\_str();

\*value = (char \*)kviter->second.c\_str();

return 0;

}

void kv\_populate(void)

{

kvpair["SAI\_INIT\_CONFIG\_FILE"] = "asic\_file.cfg";

}

main () {

sai\_service\_method\_table\_t sai\_service\_method\_table;

kv\_populate();

sai\_service\_method\_table.profile\_get\_value = profile\_get\_value;

sai\_service\_method\_table.profile\_get\_next\_value = profile\_get\_next\_value;

Step 1: Load ASIC SAI library

void\* asic\_handle = dlopen("lisai\_asic.so", RTLD\_LAZY);

Step2 : Get asic\_api\_initialize & asic\_api\_query API’s

sai\_status\_t (\*asic\_api\_initialize)(uint64\_t flags, const sai\_service\_method\_table\_t \*services);

sai\_status\_t (\*asic\_api\_query) (sai\_api\_t api, void \*\*api\_method\_table);

\*(void\*\*)(&asic\_api\_initialize) = dlsym(handle, "sai\_api\_initialize");

asic\_api\_initialize(0, &sai\_service\_method\_table);

\*(void\*\*)(&asic\_api\_query) = dlsym(handle, "sai\_api\_initialize");

sai\_switch\_api\_t\* asic\_switch\_api = NULL;

**asic\_api\_query(SAI\_API\_SWITCH,(static\_cast<void\*\*> (static\_cast<void\*>(&asic\_switch\_api))));**

Step 3: Initialize NPU

sai\_object\_id\_t switch\_id;

sai\_attr\_set[0].id = SAI\_SWITCH\_ATTR\_INIT\_SWITCH;

sai\_attr\_set[0].value.booldata = 1;

sai\_attr\_set[1].id = SAI\_SWITCH\_ATTR\_SWITCH\_PROFILE\_ID;

sai\_attr\_set[1].value.u32 = 0;

sai\_attr\_set[2].id = SAI\_SWITCH\_ATTR\_SWITCH\_HARDWARE\_INFO;

sai\_attr\_set[2].value.u32 = "0x0001";

sai\_attr\_set[4].id = SAI\_SWITCH\_ATTR\_PORT\_STATE\_CHANGE\_NOTIFY;

sai\_attr\_set[4].value.ptr = (void \*)sai\_port\_state\_evt\_callback;

**asic\_switch\_api->create\_switch (&asic\_switch\_id , attr\_count, sai\_attr\_set)));**

Step 4: Get Default Ports Created as part of switch initialization by config file passsd.

uint32\_t max\_ports = 0;

sai\_get\_attr.id = SAI\_SWITCH\_ATTR\_PORT\_NUMBER;

asic\_switch\_api->get\_switch\_attribute(switch\_id,1, &sai\_get\_attr);

max\_ports = sai\_get\_attr.value.u32;

**get\_attr.id = SAI\_SWITCH\_ATTR\_PORT\_LIST;**

get\_attr.value.objlist.count = 1;

get\_attr.value.objlist.list = (sai\_object\_id\_t \*) calloc(max\_ports,

sizeof(sai\_object\_id\_t));

ret = asic\_switch\_api->get\_switch\_attribute(switch\_id,1, &get\_attr);

for(count = 0; count < get\_attr.value.objlist.count; count++) {

LOG\_PRINT("Port list id %d is %lu \r\n", count, get\_attr.value.objlist.list[count]);

}

}

}

#### Creating ASIC Ports with 100G

As part of ASIC initialization, all ports need to create with default settings. If not SAI provided option to create ports with port API.

Step 1 : Query ASIC port API table.

**asic\_api\_query(SAI\_API\_PORT,(static\_cast<void\*\*> (static\_cast<void\*>(&asic\_port\_api))));**

Step 2: Create SAI ports for Ethernet0, Ethernet1 , Ethernet2

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

sai\_attr\_set[0].value.u32list.count = 2;

port\_lane\_list[0] = 1;

port\_lane\_list[1] = 2;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 100G;

**ret = asic\_port\_api->create\_port(&asic\_port\_id\_0, switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 2;

port\_lane\_list[0] = 3;

port\_lane\_list[1] = 4;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 100G;

**ret = asic\_port\_api->create\_port(&asic\_port\_id\_1, switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 2;

port\_lane\_list[0] = 5;

port\_lane\_list[1] = 6;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 100G;

**ret = asic\_port\_api->create\_port(&asic\_port\_id\_2, switch\_id, 2, sai\_attr\_set);**

Step 3: Set port attributes for Ethernet0, Ethernet4 , Ethernet8

sai\_attr\_set.id = SAI\_PORT\_ATTR\_FEC\_MODE;

sai\_attr\_set.value.s32 = SAI\_PORT\_FEC\_MODE\_FC;

**asic\_port\_api->set\_port\_attribute(asic\_port\_id\_0, &sai\_attr\_set);**

**asic\_port\_api->set\_port\_attribute(asic\_port\_id\_1, &sai\_attr\_set);**

**asic\_port\_api->set\_port\_attribute(asic\_port\_id\_2, &sai\_attr\_set);**

#### Example PHY-1 initialization

NOS will start initializing PHY based on config file.

static sai\_kv\_pair\_t kvpair;

void kv\_populate(int profile)

{

Kvpair[1]["SAI\_INIT\_CONFIG\_FILE"] = "phy\_1\_config\_file";

}

main () {

sai\_service\_method\_table\_t sai\_service\_method\_table;

kv\_populate();

sai\_service\_method\_table.profile\_get\_value = profile\_get\_value;

sai\_service\_method\_table.profile\_get\_next\_value = profile\_get\_next\_value;

Step 1: Platform adaption API for PHY 1. PHY1 is controlled by NPU. Use ASIC SAI provided API.

sai\_status\_t phy\_1\_register\_read(void \*platform\_ctx, sai\_object\_id\_t phy\_id,

uint32\_t device\_addr,

uint32\_t reg\_addr,

uint32\_t \*reg\_val) {

asic\_switch\_api->mdio\_read(switch\_id, device\_addr, reg-addr, reg\_val);

return OK;

}

sai\_status\_t phy\_1\_register\_write(void \*platform\_ctx, sai\_object\_id\_t phy\_id,

uint32\_t device\_addr,

uint32\_t reg\_addr,

uint32\_t reg\_val) {

asic\_switch\_api->mdio\_write(switch\_id, device\_addr, reg-addr, reg\_val);

return OK;

}

Step 1: Load Vendor(Exmaple1) PHY SAI library

void\* phy\_1\_handle = dlopen("libsai\_phy\_example1.so", RTLD\_LAZY);

Step2 : Get Example1 PHY SAI sai\_api\_initialize & asic\_api\_query API’s

sai\_status\_t (\*phy\_1\_api\_initialize)(uint64\_t flags, const sai\_service\_method\_table\_t \*services);

sai\_status\_t (\*phy\_1\_api\_query) (sai\_api\_t api, void \*\*api\_method\_table);

\*(void\*\*)(&phy\_1\_api\_initialize) = dlsym(handle, "sai\_api\_initialize");

phy\_1\_api\_initialize(0, &sai\_service\_method\_table);

\*(void\*\*)(&phy\_1\_api\_query) = dlsym(handle, "sai\_api\_query");

sai\_switch\_api\_t\* phy\_1\_switch\_api = NULL;

**asic\_api\_query(SAI\_API\_SWITCH,(static\_cast<void\*\*> (static\_cast<void\*>(&phy\_1\_switch\_api))));**

Step 3: Initialize PHY

sai\_object\_id\_t phy\_1\_switch\_id;

sai\_attr\_set[0].id = SAI\_SWITCH\_ATTR\_INIT\_SWITCH;

sai\_attr\_set[0].value.booldata = 1;

sai\_attr\_set[1].id = SAI\_SWITCH\_ATTR\_SWITCH\_PROFILE\_ID;

sai\_attr\_set[1].value.u32 = 0;

sai\_attr\_set[2].id = SAI\_SWITCH\_ATTR\_SWITCH\_HARDWARE\_INFO;

sai\_attr\_set[2].value.u32 = “busid:0x1000";

**sai\_attr\_set[3].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_PATH\_NAME;**

**sai\_attr\_set[3].value.str = “/tmp/phyexample1.bin”;**

sai\_attr\_set[4].id = SAI\_SWITCH\_ATTR\_REGISTER\_READ;

sai\_attr\_set[4].value.ptr = (void \*) &phy\_1\_register\_read;

sai\_attr\_set[5].id = SAI\_SWITCH\_ATTR\_REGISTER\_WRITE;

sai\_attr\_set[5].value.ptr = (void \*) &phy\_1\_register\_write;

sai\_attr\_set[10].id = SAI\_SWITCH\_ATTR\_PLATFROM\_CONTEXT;

sai\_attr\_set[10].value.ptr = (void \*)&sai\_platform\_context;

sai\_attr\_set[6].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_DOWNLOAD\_BROADCAST;

sai\_attr\_set[6].value.booldata = 0;

sai\_attr\_set[7].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_LOAD\_TYPE;

sai\_attr\_set[7].value.u32 = SAI\_SWITCH\_FIRMWARE\_LOAD\_AUTOs;

sai\_attr\_set[8].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_LOAD\_METHOD;

sai\_attr\_set[8].value.u32 = SAI\_SWITCH\_FIRMWARE\_LOAD\_METHOD\_INTERNAL;

sai\_attr\_set[9].id = SAI\_SWITCH\_ATTR\_PORT\_STATE\_CHANGE\_NOTIFY;

sai\_attr\_set[9].value.ptr = (void \*)sai\_port\_state\_evt\_callback;

**phy\_1\_switch\_api->create\_switch (&phy\_1\_switch\_id , attr\_count, sai\_attr\_set)));**

Step 4: Get Default Ports Created as part of PHY initialization by config file passsd.

uint32\_t max\_ports = 0;

sai\_get\_attr.id = SAI\_SWITCH\_ATTR\_PORT\_NUMBER;

phy\_1\_switch\_api->get\_switch\_attribute(phy\_1\_switch\_id,1, &sai\_get\_attr);

max\_ports = sai\_get\_attr.value.u32;

**get\_attr.id = SAI\_SWITCH\_ATTR\_PORT\_LIST;**

get\_attr.value.objlist.count = 1;

get\_attr.value.objlist.list = (sai\_object\_id\_t \*) calloc(max\_ports,

sizeof(sai\_object\_id\_t));

ret = phy\_1\_switch\_api\_table->get\_switch\_attribute(phy\_1\_switch\_id,1, &get\_attr);

for(count = 0; count < get\_attr.value.objlist.count; count++) {

LOG\_PRINT("Port list id %d is %lu \r\n", count, get\_attr.value.objlist.list[count]);

}

Step 4: Get Default Port connectors Created as part of PHY initialization by config file passsd.

**get\_attr.id = SAI\_SWITCH\_ATTR\_PORT\_CONNECTOR\_LIST;**

get\_attr.value.objlist.count = 1;

get\_attr.value.objlist.list = (sai\_object\_id\_t \*) calloc(max\_ports,

sizeof(sai\_object\_id\_t));

ret = phy\_1\_switch\_api\_table->get\_switch\_attribute(phy\_1\_switch\_id,1, &get\_attr);

for(count = 0; count < get\_attr.value.objlist.count; count++) {

LOG\_PRINT("Port connector list id %d is %lu \r\n", count, get\_attr.value.objlist.list[count]);

}

#### Creating PHY1 Ports with 100G

As part of PHY1 initialization, all ports need to create with default settings. If not SAI provides option to create ports with port API.

Step 1 : Query PHY port API table.

**phy\_1\_api\_query(SAI\_API\_PORT,(static\_cast<void\*\*> (static\_cast<void\*>(&phy\_1\_port\_api))));**

**phy\_1\_api\_query(SAI\_API\_PORT\_CONNECTOR,(static\_cast<void\*\*> (static\_cast<void\*>(&phy\_1\_port\_connector\_api))));**

Step 2: Create PHY1 System Side ports for Ethernet0, Ethernet4

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

sai\_attr\_set[0].value.u32list.count = 2;

port\_lane\_list[0] = 38;

port\_lane\_list[1] = 39;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 100G;

**ret = phy\_1\_port\_api->create\_port(&phy\_1\_system\_port\_id\_0, phy\_1\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 2;

port\_lane\_list[0] = 40;

port\_lane\_list[1] = 41;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 100G;

**ret = phy\_1\_port\_api->create\_port(&phy\_1\_system\_port\_id\_1, phy\_1\_switch\_id, 2, sai\_attr\_set);**

Step 2: Create PHY1 Line Side ports for Ethernet0, Ethernet4

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

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

port\_lane\_list[0] = 30;

port\_lane\_list[1] = 31;

port\_lane\_list[0] = 32;

port\_lane\_list[1] = 33;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 100G;

**ret = phy\_1\_port\_api->create\_port(&phy\_1\_line\_port\_id\_0, phy\_1\_switch\_id, 2, sai\_attr\_set);**

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

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

port\_lane\_list[0] = 34;

port\_lane\_list[1] = 35;

port\_lane\_list[0] = 36;

port\_lane\_list[1] = 37;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 100G;

**ret = phy\_1\_port\_api->create\_port(&phy\_1\_line\_port\_id\_1, phy\_1\_switch\_id, 2, sai\_attr\_set);**

Step 2-1: Create Port connector for PHY1 Line Side and system side ports for Ethernet0, Ethernet4

sai\_attr\_set[0].id = SAI\_PORT\_CONNECTOR\_ATTR\_SYSTEMSIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_1\_system\_port\_id\_0**;

sai\_attr\_set[1].id = SAI\_PORT\_CONNECTOR\_ATTR\_LINESIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_1\_line\_port\_id\_0**;

**ret = phy\_1\_port\_connector\_api->create\_port\_connector(&phy\_1\_port\_conn\_id\_0, phy\_1\_switch\_id, 2, sai\_attr\_set);**

sai\_attr\_set[0].id = SAI\_PORT\_CONNECTOR\_ATTR\_SYSTEMSIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_1\_system\_port\_id\_1**;

sai\_attr\_set[1].id = SAI\_PORT\_CONNECTOR\_ATTR\_LINESIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_1\_line\_port\_id\_1**;

**ret = phy\_1\_port\_connector\_api->create\_port\_connector(&phy\_1\_port\_conn\_id\_1, phy\_1\_switch\_id, 2, sai\_attr\_set);**

Step 3: Set port attributes for Ethernet0, Ethernet4

Step 3-1: Set interface FEC types different in system and line side ports

sai\_attr\_set.id = SAI\_PORT\_ATTR\_FEC\_MODE;

sai\_attr\_set.value.s32 = SAI\_PORT\_FEC\_MODE\_FC;

**phy\_1\_port\_api->set\_port\_attribute(phy\_1\_system\_port\_id\_0, &sai\_attr\_set);**

**phy\_1\_port\_api->set\_port\_attribute(phy\_1\_system\_port\_id\_1, &sai\_attr\_set);**

sai\_attr\_set.id = SAI\_PORT\_ATTR\_FEC\_MODE;

sai\_attr\_set.value.s32 = SAI\_PORT\_FEC\_MODE\_RS;

**phy\_1\_port\_api->set\_port\_attribute(phy\_1\_line\_port\_id\_0, &sai\_attr\_set);**

**phy\_1\_port\_api->set\_port\_attribute(phy\_1\_line\_port\_id\_1, &sai\_attr\_set);**

#### Example PHY-2 initialization

NOS will start initializing PHY based on config file.

static sai\_kv\_pair\_t kvpair;

void kv\_populate(int profile)

{

Kvpair[2]["SAI\_INIT\_CONFIG\_FILE"] = "phy\_2\_config\_file";

}

main () {

sai\_service\_method\_table\_t sai\_service\_method\_table;

kv\_populate();

sai\_service\_method\_table.profile\_get\_value = profile\_get\_value;

sai\_service\_method\_table.profile\_get\_next\_value = profile\_get\_next\_value;

Step 1: Platform adaption API for PHY 2. PHY2 is controleld over i2c(for example)

sai\_status\_t phy\_2\_register\_read(void \*platform\_ctx, sai\_object\_id\_t switch\_id,

uint32\_t device\_addr,

uint32\_t reg\_addr,

uint32\_t \*reg\_val) {

platform\_i2c\_read(devicd\_addr, reg-addr, reg\_val);

return OK;

}

sai\_status\_t phy\_2\_register\_write(void \*platform\_ctx, sai\_object\_id\_t switch\_id,

uint32\_t device\_addr,

uint32\_t reg\_addr,

uint32\_t reg\_val) {

platform\_i2c\_write(devicd\_addr, reg-addr, reg\_val);

return OK;

}

Step 1: Load Vendor(Exmaple2) PHY SAI library

void\* phy\_1\_handle = dlopen("**libsai\_phy\_example2.so**", RTLD\_LAZY);

Step2 : Get Example1 PHY SAI sai\_api\_initialize & asic\_api\_query API’s

sai\_status\_t (\*phy\_2\_api\_initialize)(uint64\_t flags, const sai\_service\_method\_table\_t \*services);

sai\_status\_t (\*phy\_2\_api\_query) (sai\_api\_t api, void \*\*api\_method\_table);

\*(void\*\*)(&phy\_2\_api\_initialize) = dlsym(handle, "sai\_api\_initialize");

phy\_2\_api\_initialize(0, &sai\_service\_method\_table);

\*(void\*\*)(&phy\_2\_api\_query) = dlsym(handle, "sai\_api\_query");

sai\_switch\_api\_t\* phy\_2\_switch\_api = NULL;

**asic\_api\_query(SAI\_API\_SWITCH,(static\_cast<void\*\*> (static\_cast<void\*>(&phy\_2\_switch\_api))));**

Step 3: Initialize PHY

sai\_object\_id\_t phy\_2\_switch\_id;

sai\_attr\_set[0].id = SAI\_SWITCH\_ATTR\_INIT\_SWITCH;

sai\_attr\_set[0].value.booldata = 1;

sai\_attr\_set[1].id = SAI\_SWITCH\_ATTR\_SWITCH\_PROFILE\_ID;

sai\_attr\_set[1].value.u32 = 0;

sai\_attr\_set[2].id = SAI\_SWITCH\_ATTR\_SWITCH\_HARDWARE\_INFO;

sai\_attr\_set[2].value.u32 = “0x1000";

**sai\_attr\_set[3].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_PATH\_NAME;**

**sai\_attr\_set[3].value.str = “/tmp/phyexample2.bin”;**

sai\_attr\_set[4].id = SAI\_SWITCH\_ATTR\_REGISTER\_READ;

sai\_attr\_set[4].value.ptr = (void \*) phy\_2\_register\_read;

sai\_attr\_set[5].id = SAI\_SWITCH\_ATTR\_REGISTER\_WRITE;

sai\_attr\_set[5].value.ptr = (void \*) phy\_2\_register\_write;

sai\_attr\_set[10].id = SAI\_SWITCH\_ATTR\_PLATFROM\_CONTEXT;

sai\_attr\_set[10].value.ptr = (void \*)&sai\_platform\_context;

sai\_attr\_set[6].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_DOWNLOAD\_BROADCAST;

sai\_attr\_set[6].value.booldata = 0;

sai\_attr\_set[7].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_LOAD\_TYPE;

sai\_attr\_set[7].value.u32 = SAI\_SWITCH\_FIRMWARE\_LOAD\_AUTOs;

sai\_attr\_set[8].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_LOAD\_METHOD;

sai\_attr\_set[8].value.u32 = SAI\_SWITCH\_FIRMWARE\_LOAD\_METHOD\_INTERNAL;

sai\_attr\_set[4].id = SAI\_SWITCH\_ATTR\_PORT\_STATE\_CHANGE\_NOTIFY;

sai\_attr\_set[4].value.ptr = (void \*)sai\_port\_state\_evt\_callback;

**phy\_2\_switch\_api->create\_switch (&phy\_2\_switch\_id , attr\_count, sai\_attr\_set)));**

Step 4: Get Default Ports Created as part of PHY initialization by config file passsd.

uint32\_t max\_ports = 0;

sai\_get\_attr.id = SAI\_SWITCH\_ATTR\_PORT\_NUMBER;

phy\_2\_switch\_api->get\_switch\_attribute(phy\_2\_switch\_id,1, &sai\_get\_attr);

max\_ports = sai\_get\_attr.value.u32;

**get\_attr.id = SAI\_SWITCH\_ATTR\_PORT\_LIST;**

get\_attr.value.objlist.count = 1;

get\_attr.value.objlist.list = (sai\_object\_id\_t \*) calloc(max\_ports,

sizeof(sai\_object\_id\_t));

ret = phy\_2\_switch\_api\_table->get\_switch\_attribute(phy\_2\_switch\_id,1, &get\_attr);

for(count = 0; count < get\_attr.value.objlist.count; count++) {

LOG\_PRINT("Port list id %d is %lu \r\n", count, get\_attr.value.objlist.list[count]);

}

**get\_attr.id = SAI\_SWITCH\_ATTR\_PORT\_CONNECTOR\_LIST;**

get\_attr.value.objlist.count = 1;

get\_attr.value.objlist.list = (sai\_object\_id\_t \*) calloc(max\_ports,

sizeof(sai\_object\_id\_t));

ret = phy\_2\_switch\_api\_table->get\_switch\_attribute(phy\_2\_switch\_id,1, &get\_attr);

for(count = 0; count < get\_attr.value.objlist.count; count++) {

LOG\_PRINT("Port connector list id %d is %lu \r\n", count, get\_attr.value.objlist.list[count]);

}

#### Creating PHY2 Ports With 100G

As part of PHY2 initialization, all ports need to create with default settings. If not SAI provides option to create ports with port API.

Step 1 : Query PHY port API table.

**phy\_2\_api\_query(SAI\_API\_PORT,(static\_cast<void\*\*> (static\_cast<void\*>(&phy\_2\_port\_api))));**

Step 2: Create PHY2 System Side ports for Ethernet8

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

sai\_attr\_set[0].value.u32list.count = 2;

port\_lane\_list[0] = 4;

port\_lane\_list[1] = 5;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 100G;

**ret = phy\_1\_port\_api->create\_port(&phy\_2\_system\_port\_id\_0, phy\_2\_switch\_id, 2, sai\_attr\_set);**

Step 2: Create PHY2 Line Side ports for Ethernet8

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

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

port\_lane\_list[0] = 0;

port\_lane\_list[1] = 1;

port\_lane\_list[0] = 2;

port\_lane\_list[1] = 3;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 100G;

**ret = phy\_2\_port\_api->create\_port(&phy\_2\_line\_port\_id\_0, phy\_2\_switch\_id, 2, sai\_attr\_set);**

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

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

port\_lane\_list[0] = 34;

port\_lane\_list[1] = 35;

port\_lane\_list[0] = 36;

port\_lane\_list[1] = 37;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 100G;

**ret = phy\_2\_port\_api->create\_port(&phy\_1\_line\_port\_id\_1, phy\_1\_switch\_id, 2, sai\_attr\_set);**

Step 2-1: Create Port connector for PHY2 Line Side and system side ports for Ethernet8

sai\_attr\_set[0].id = SAI\_PORT\_CONNECTOR\_ATTR\_SYSTEMSIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_system\_port\_id\_0**;

sai\_attr\_set[1].id = SAI\_PORT\_CONNECTOR\_ATTR\_LINESIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_line\_port\_id\_0**;

**ret = phy\_2\_port\_connector\_api->create\_port\_connector(&phy\_2\_port\_conn\_id\_0, phy\_2\_switch\_id, 2, sai\_attr\_set);**

Step 3: Set port attributes for Ethernet8

Step 3-1: Set interface FEC types different in system and line side ports

sai\_attr\_set.id = SAI\_PORT\_ATTR\_FEC\_MODE;

sai\_attr\_set.value.s32 = SAI\_PORT\_FEC\_MODE\_FC;

**phy\_2\_port\_api->set\_port\_attribute(phy\_2\_system\_port\_id\_0, &sai\_attr\_set);**

sai\_attr\_set.id = SAI\_PORT\_ATTR\_FEC\_MODE;

sai\_attr\_set.value.s32 = SAI\_PORT\_FEC\_MODE\_RS;

**phy\_2\_port\_api->set\_port\_attribute(phy\_2\_line\_port\_id\_0, &sai\_attr\_set);**

#### NOS Mapping

After Initializing NOS will have below objects to control ASIC and PHY

* ASIC Switch object - asic\_switch\_id
* PHY1 switch object – phy\_1\_switch\_id
* PHY2 switch object – phy\_2\_switch\_id
* NOS port objects

Ethernet0 asic\_port\_id\_0, phy\_1\_system\_port\_id\_0, phy\_1\_system\_port\_id\_0, phy\_1\_port\_conn\_id\_0

Ethernet1 asic\_port\_id\_1, phy\_1\_system\_port\_id\_1, phy\_1\_system\_port\_id\_1, phy\_1\_port\_conn\_id\_1

Ethernet2 asic\_port\_id\_2, phy\_2\_system\_port\_id\_0, phy\_2\_system\_port\_id\_0., phy\_2\_port\_conn\_id\_0

### Convert Port Ethernet0 from 100G to 40G

sai\_attr\_set.id = SAI\_PORT\_ATTR\_SPPPED;

sai\_attr\_set.value.u32 = 40G;

asic\_port\_api-> **set\_port\_attribute(asic\_port\_id\_0, &sai\_attr\_set);**

**phy\_1\_port\_api->set\_port\_attribute(phy\_1\_system\_port\_id\_0, &sai\_attr\_set);**

**phy\_1\_port\_api->set\_port\_attribute(phy\_1\_line\_port\_id\_0, &sai\_attr\_set);**

**phy\_2\_port\_api->set\_port\_attribute(phy\_2\_system\_port\_id\_0, &sai\_attr\_set);**

**phy\_2\_port\_api->set\_port\_attribute(phy\_2\_line\_port\_id\_0, &sai\_attr\_set);**

### Convert Port Ethernet2 from 100G to 4 10G Front end ports

To support 4 individual 10G ports, ASIC should have 4 port representation for each front panel.

In PHY2, system side lanes 4,5,6,7 and line side 0,1,2,3 can be mapped 1-1.

Step 1: Remove the exiting port oid’s and port connector oid’s for Ethernet2

**phy\_2\_port\_connector\_api->remove\_port\_connector(phy\_2\_port\_conn\_id\_0);**

**asic\_port\_api->remove\_port(asic\_port\_id\_2);**

**phy\_2\_port\_api->remove\_port(phy\_2\_system\_port\_id\_0);**

**pht\_2\_port\_api->remove\_port(phy\_2\_line\_port\_id\_0);**

Step 2: Create 4 ports in ASIC for each 10G Ethernet2, Ethernet3, Ethernet4, Ethernet5

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 5;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = asic\_port\_api->create\_port(&asic\_port\_id\_2, asic\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 6;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = asic\_port\_api->create\_port(&asic\_port\_id\_3, asic\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 7;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = asic\_port\_api->create\_port(&asic\_port\_id\_4, asic\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 8;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = asic\_port\_api->create\_port(&asic\_port\_id\_5, asic\_switch\_id, 2, sai\_attr\_set);**

Step 3: Create system side PHY ports each 10G Ethernet2, Ethernet3, Ethernet4, Ethernet5

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 4;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_2\_port\_api->create\_port(&phy\_2\_system\_port\_id\_0, phy\_2\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 5;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_2\_port\_api->create\_port(&phy\_2\_system\_port\_id\_1, phy\_2\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 6;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_2\_port\_api->create\_port(&phy\_2\_system\_port\_id\_2, phy\_2\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 7;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_2\_port\_api->create\_port(&phy\_2\_system\_port\_id\_3, phy\_2\_switch\_id, 2, sai\_attr\_set);**

Step 4: Create line side PHY ports each 10G Ethernet2, Ethernet3, Ethernet4, Ethernet5

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 0;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_2\_port\_api->create\_port(&phy\_2\_line\_port\_id\_0, phy\_2\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 1;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_2\_port\_api->create\_port(&phy\_2\_line\_port\_id\_1, phy\_2\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 2;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_2\_port\_api->create\_port(&phy\_2\_line\_port\_id\_2, phy\_2\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 3;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_2\_port\_api->create\_port(&phy\_2\_line\_port\_id\_3, phy\_2\_switch\_id, 2, sai\_attr\_set);**

Step 2-1: Create Port connector for PHY2 Line Side and system side ports each 10G Ethernet2, Ethernet3, Ethernet4, Ethernet5

sai\_attr\_set[0].id = SAI\_PORT\_CONNECTOR\_ATTR\_SYSTEMSIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_system\_port\_id\_0**;

sai\_attr\_set[1].id = SAI\_PORT\_CONNECTOR\_ATTR\_LINESIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_line\_port\_id\_0**;

**ret = phy\_2\_port\_connector\_api->create\_port\_connector(&phy\_2\_port\_conn\_id\_0, phy\_2\_switch\_id, 2, sai\_attr\_set);**

sai\_attr\_set[0].id = SAI\_PORT\_CONNECTOR\_ATTR\_SYSTEMSIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_system\_port\_id\_1**;

sai\_attr\_set[1].id = SAI\_PORT\_CONNECTOR\_ATTR\_LINESIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_line\_port\_id\_1**;

**ret = phy\_2\_port\_connector\_api->create\_port\_connector(&phy\_2\_port\_conn\_id\_1, phy\_2\_switch\_id, 2, sai\_attr\_set);**

sai\_attr\_set[0].id = SAI\_PORT\_CONNECTOR\_ATTR\_SYSTEMSIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_system\_port\_id\_2**;

sai\_attr\_set[1].id = SAI\_PORT\_CONNECTOR\_ATTR\_LINESIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_line\_port\_id\_2**;

**ret = phy\_2\_port\_connector\_api->create\_port\_connector(&phy\_2\_port\_conn\_id\_2, phy\_2\_switch\_id, 2, sai\_attr\_set);**

sai\_attr\_set[0].id = SAI\_PORT\_CONNECTOR\_ATTR\_SYSTEMSIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_system\_port\_id\_3**;

sai\_attr\_set[1].id = SAI\_PORT\_CONNECTOR\_ATTR\_LINESIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_line\_port\_id\_3**;

**ret = phy\_2\_port\_connector\_api->create\_port\_connector(&phy\_2\_port\_conn\_id\_3, phy\_2\_switch\_id, 2, sai\_attr\_set);**

### Convert Port Ethernet0 from 100G to 4 10G Front end ports

Right now, Ethernet0 and Ethernet1 shares the same core in ASIC side. To create 4 10G front panel ports, We need to reuse the ASIC lanes 1,2,3,4.

In such cases

* Need to disable some front panel port. Which is previous Ethernet2.
* Remap system side lanes 38,39,40,41 to line side lanes 30,31,32,33.
* Each lane is individual port.

Step 1: Remove the exiting port oid’s and port connector oid’s for Ethernet0, Ethernet1

**phy\_1\_port\_connector\_api->remove\_port\_connector(phy\_1\_port\_conn\_id\_0);**

**phy\_1\_port\_connector\_api->remove\_port\_connector(phy\_1\_port\_conn\_id\_1);**

**asic\_port\_api->remove\_port(asic\_port\_id\_0);**

**phy\_1\_port\_api->remove\_port(phy\_1\_system\_port\_id\_0);**

**phy\_1\_port\_api->remove\_port(phy\_1\_line\_port\_id\_0);**

**asic\_port\_api->remove\_port(asic\_port\_id\_1);**

**phy\_1\_port\_api->remove\_port(phy\_1\_system\_port\_id\_1);**

**pht\_1\_port\_api->remove\_port(phy\_1\_line\_port\_id\_1);**

Step 2: Create 4 ports in ASIC for each 10G Ethernet0-0, Ethernet0-1, Ethernet0-2, Ethernet0-3

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 1;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = asic\_port\_api->create\_port(&asic\_port\_id\_0\_0, asic\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 2;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = asic\_port\_api->create\_port(&asic\_port\_id\_0\_1, asic\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 3;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = asic\_port\_api->create\_port(&asic\_port\_id\_0\_2, asic\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 4;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = asic\_port\_api->create\_port(&asic\_port\_id\_0\_3, asic\_switch\_id, 2, sai\_attr\_set);**

Step 3: Create system side PHY ports each 10G Ethernet0-0, Ethernet0-1, Ethernet0-2, Ethernet0-3

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 38;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_1\_port\_api->create\_port(&phy\_1\_system\_port\_id\_0\_0, phy\_1\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 39;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_1\_port\_api->create\_port(&phy\_1\_system\_port\_id\_0\_1, phy\_1\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 40;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_1\_port\_api->create\_port(&phy\_1\_system\_port\_id\_0\_2, phy\_1\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 41;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_1\_port\_api->create\_port(&phy\_1\_system\_port\_id\_0\_3, phy\_1\_switch\_id, 2, sai\_attr\_set);**

Step 4: Create line side PHY ports each 10G Ethernet0-0, Ethernet0-1, Ethernet0-2, Ethernet0-3

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 30;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_1\_port\_api->create\_port(&phy\_1\_line\_port\_id\_0\_0, phy\_1\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 31;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_1\_port\_api->create\_port(&phy\_1\_line\_port\_id\_0\_1, phy\_1\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 32;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_1\_port\_api->create\_port(&phy\_1\_line\_port\_id\_0\_2, phy\_1\_switch\_id, 2, sai\_attr\_set);**

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

sai\_attr\_set[0].value.u32list.count = 1;

port\_lane\_list[0] = 33;

sai\_attr\_set[0].value.u32list.list = port\_lane\_list;

sai\_attr\_set[1].id = SAI\_PORT\_ATTR\_SPEED;

sai\_attr\_set[1].value.u32 = 10G;

**ret = phy\_1\_port\_api->create\_port(&phy\_1\_line\_port\_id\_0\_3, phy\_1\_switch\_id, 2, sai\_attr\_set);**

**Step 5: Modify cross connect map with in PHY , i.e Remap system side lane to line side lane by creasting the port connector object.**

sai\_attr\_set[0].id = SAI\_PORT\_CONNECTOR\_ATTR\_SYSTEMSIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_system\_port\_id\_0\_0**;

sai\_attr\_set[1].id = SAI\_PORT\_CONNECTOR\_ATTR\_LINESIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_line\_port\_id\_0\_0**;

**ret = phy\_2\_port\_connector\_api->create\_port\_connector(&phy\_2\_port\_conn\_id\_0\_0, phy\_2\_switch\_id, 2, sai\_attr\_set);**

sai\_attr\_set[0].id = SAI\_PORT\_CONNECTOR\_ATTR\_SYSTEMSIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_system\_port\_id\_0\_1**;

sai\_attr\_set[1].id = SAI\_PORT\_CONNECTOR\_ATTR\_LINESIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_line\_port\_id\_0\_1**;

**ret = phy\_2\_port\_connector\_api->create\_port\_connector(&phy\_2\_port\_conn\_id\_0\_0, phy\_2\_switch\_id, 2, sai\_attr\_set);**

sai\_attr\_set[0].id = SAI\_PORT\_CONNECTOR\_ATTR\_SYSTEMSIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_system\_port\_id\_0\_2**;

sai\_attr\_set[1].id = SAI\_PORT\_CONNECTOR\_ATTR\_LINESIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_line\_port\_id\_0\_2**;

**ret = phy\_2\_port\_connector\_api->create\_port\_connector(&phy\_2\_port\_conn\_id\_0\_1, phy\_2\_switch\_id, 2, sai\_attr\_set);**

sai\_attr\_set[0].id = SAI\_PORT\_CONNECTOR\_ATTR\_SYSTEMSIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_system\_port\_id\_0\_3**;

sai\_attr\_set[1].id = SAI\_PORT\_CONNECTOR\_ATTR\_LINESIDE\_PORT;

sai\_attr\_set.value.objectid = **phy\_2\_line\_port\_id\_0\_3**;

**ret = phy\_2\_port\_connector\_api->create\_port\_connector(&phy\_2\_port\_conn\_id\_0\_3, phy\_2\_switch\_id, 2, sai\_attr\_set);**

### Example PHY-1 & PHY-2 initialization with Firmware download broadcast

NOS will start initializing PHY based on config file.

static sai\_kv\_pair\_t kvpair;

void kv\_populate(int profile)

{

Kvpair[1]["SAI\_INIT\_CONFIG\_FILE"] = "phy\_1\_config\_file";

}

main () {

sai\_service\_method\_table\_t sai\_service\_method\_table;

kv\_populate();

sai\_service\_method\_table.profile\_get\_value = profile\_get\_value;

sai\_service\_method\_table.profile\_get\_next\_value = profile\_get\_next\_value;

Step 1: Platform adaption API for PHY 1. PHY1 is controlled by NPU. Use ASIC SAI provided API.

sai\_status\_t phy\_1\_register\_read(void \*platform\_ctx, sai\_object\_id\_t phy\_id,

uint32\_t device\_addr,

uint32\_t reg\_addr,

uint32\_t \*reg\_val) {

asic\_switch\_api->mdio\_read(platform\_ctx, switch\_id, device\_addr, reg-addr, reg\_val);

return OK;

}

sai\_status\_t phy\_1\_register\_write(void \*platform\_ctx, sai\_object\_id\_t phy\_id,

uint32\_t device\_addr,

uint32\_t reg\_addr,

uint32\_t reg\_val) {

asic\_switch\_api->mdio\_write(platform\_ctx, switch\_id, device\_addr, reg-addr, reg\_val);

return OK;

}

Step 1: Load Vendor(Exmaple1) PHY SAI library

void\* phy\_1\_handle = dlopen("libsai\_phy\_example1.so", RTLD\_LAZY);

Step2 : Get Example1 PHY SAI sai\_api\_initialize & asic\_api\_query API’s

sai\_status\_t (\*phy\_1\_api\_initialize)(uint64\_t flags, const sai\_service\_method\_table\_t \*services);

sai\_status\_t (\*phy\_1\_api\_query) (sai\_api\_t api, void \*\*api\_method\_table);

\*(void\*\*)(&phy\_1\_api\_initialize) = dlsym(handle, "sai\_api\_initialize");

phy\_1\_api\_initialize(0, &sai\_service\_method\_table);

\*(void\*\*)(&phy\_1\_api\_query) = dlsym(handle, "sai\_api\_query");

sai\_switch\_api\_t\* phy\_1\_switch\_api = NULL;

**asic\_api\_query(SAI\_API\_SWITCH,(static\_cast<void\*\*> (static\_cast<void\*>(&phy\_1\_switch\_api))));**

Step 3: Initialize PHY 1

sai\_object\_id\_t phy\_1\_switch\_id;

sai\_attr\_set[0].id = SAI\_SWITCH\_ATTR\_INIT\_SWITCH;

sai\_attr\_set[0].value.booldata = 1;

sai\_attr\_set[1].id = SAI\_SWITCH\_ATTR\_SWITCH\_PROFILE\_ID;

sai\_attr\_set[1].value.u32 = 0;

sai\_attr\_set[2].id = SAI\_SWITCH\_ATTR\_SWITCH\_HARDWARE\_INFO;

sai\_attr\_set[2].value.u32 = “0x1000";

**sai\_attr\_set[3].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_PATH\_NAME;**

**sai\_attr\_set[3].value.str = “/tmp/phyexample1.bin”;**

sai\_attr\_set[4].id = SAI\_SWITCH\_ATTR\_REGISTER\_READ;

sai\_attr\_set[4].value.ptr = (void \*) &phy\_1\_register\_read;

sai\_attr\_set[5].id = SAI\_SWITCH\_ATTR\_REGISTER\_WRITE;

sai\_attr\_set[5].value.ptr = (void \*) &phy\_1\_register\_write;

**sai\_attr\_set[6].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_DOWNLOAD\_BROADCAST;**

**sai\_attr\_set[6].value.booldata = 1;**

**sai\_attr\_set[7].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_LOAD\_TYPE;**

**sai\_attr\_set[7].value.u32 = SAI\_SWITCH\_FIRMWARE\_LOAD\_AUTO;**

**sai\_attr\_set[8].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_LOAD\_METHOD;**

**sai\_attr\_set[8].value.u32 = SAI\_SWITCH\_FIRMWARE\_LOAD\_METH**OD\_INTERNAL;

sai\_attr\_set[9].id = SAI\_SWITCH\_ATTR\_PORT\_STATE\_CHANGE\_NOTIFY;

sai\_attr\_set[9].value.ptr = (void \*)sai\_port\_state\_evt\_callback;

sai\_attr\_set[9].id = SAI\_SWITCH\_ATTR\_PLATFROM\_CONTEXT;

sai\_attr\_set[9].value.ptr = (void \*)&sai\_platform\_context;

**phy\_1\_switch\_api->create\_switch (&phy\_1\_switch\_id , attr\_count, sai\_attr\_set)));**

Step 4: Initialize PHY 2

sai\_object\_id\_t phy\_1\_switch\_id;

sai\_attr\_set[0].id = SAI\_SWITCH\_ATTR\_INIT\_SWITCH;

sai\_attr\_set[0].value.booldata = 1;

sai\_attr\_set[1].id = SAI\_SWITCH\_ATTR\_SWITCH\_PROFILE\_ID;

sai\_attr\_set[1].value.u32 = 0;

sai\_attr\_set[2].id = SAI\_SWITCH\_ATTR\_SWITCH\_HARDWARE\_INFO;

**sai\_attr\_set[2].value.u32 = “0x2000";**

**sai\_attr\_set[3].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_PATH\_NAME;**

**sai\_attr\_set[3].value.str = “/tmp/phyexample2.bin”;**

sai\_attr\_set[4].id = SAI\_SWITCH\_ATTR\_REGISTER\_READ;

sai\_attr\_set[4].value.ptr = (void \*) &phy\_1\_register\_read;

sai\_attr\_set[5].id = SAI\_SWITCH\_ATTR\_REGISTER\_WRITE;

sai\_attr\_set[5].value.ptr = (void \*) &phy\_1\_register\_write;

**sai\_attr\_set[6].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_DOWNLOAD\_BROADCAST;**

**sai\_attr\_set[6].value.booldata = 1;**

**sai\_attr\_set[7].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_LOAD\_TYPE;**

**sai\_attr\_set[7].value.u32 = SAI\_SWITCH\_FIRMWARE\_LOAD\_AUTO;**

**sai\_attr\_set[8].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_LOAD\_METHOD;**

**sai\_attr\_set[8].value.u32 = SAI\_SWITCH\_FIRMWARE\_LOAD\_METH**OD\_INTERNAL;

sai\_attr\_set[4].id = SAI\_SWITCH\_ATTR\_PORT\_STATE\_CHANGE\_NOTIFY;

sai\_attr\_set[4].value.ptr = (void \*)sai\_port\_state\_evt\_callback;

**phy\_1\_switch\_api->create\_switch (&phy\_1\_switch\_id , attr\_count, sai\_attr\_set)));**

Step 5: Start Firmware execute on PHY-1 which is first PHY connected on bus

sai\_set\_attr.id = SAI\_SWITCH\_ATTR\_FIRMWARE\_DOWNLOAD\_EXECUTE;

sai\_attr\_set.value.booldate = 1;

phy\_1\_switch\_api->set\_switch\_attribute(phy\_1\_switch\_id,1, &sai\_set\_attr);

Step 6: Stop broadcast on ALL PHY’s connected to bus

sai\_set\_attr.id = SAI\_SWITCH\_ATTR\_FIRMWARE\_BROADCAST\_END;

sai\_attr\_set.value.booldate = 1;

phy\_1\_switch\_api->set\_switch\_attribute(phy\_1\_switch\_id,1, &sai\_set\_attr);

sai\_set\_attr.id = SAI\_SWITCH\_ATTR\_FIRMWARE\_BROADCAST\_END;

sai\_attr\_set.value.booldate = 1;

phy\_2\_switch\_api->set\_switch\_attribute(phy\_2\_switch\_id,1, &sai\_set\_attr);

Step 7: Firmware verify and initialize PHY’s connected to bus

sai\_set\_attr.id = SAI\_SWITCH\_ATTR\_FIRMWARE\_VERIFY\_AND\_INIT\_SWITCH;

sai\_attr\_set.value.booldate = 1;

phy\_1\_switch\_api->set\_switch\_attribute(phy\_1\_switch\_id,1, &sai\_set\_attr);

sai\_set\_attr.id = SAI\_SWITCH\_ATTR\_FIRMWARE\_VERIFY\_AND\_INIT\_SWITCH;

sai\_attr\_set.value.booldate = 1;

phy\_2\_switch\_api->set\_switch\_attribute(phy\_2\_switch\_id,1, &sai\_set\_attr);

Step 8: Get Default Port connectors Created as part of PHY initialization by config file passsd.

**get\_attr.id = SAI\_SWITCH\_ATTR\_PORT\_CONNECTOR\_LIST;**

get\_attr.value.objlist.count = 1;

get\_attr.value.objlist.list = (sai\_object\_id\_t \*) calloc(max\_ports,

sizeof(sai\_object\_id\_t));

ret = phy\_1\_switch\_api\_table->get\_switch\_attribute(phy\_1\_switch\_id,1, &get\_attr);

for(count = 0; count < get\_attr.value.objlist.count; count++) {

LOG\_PRINT("Port connector list id %d is %lu \r\n", count, get\_attr.value.objlist.list[count]);

}

### Example for PHY-1 initialization with PHY hardware access with sysFS

NOS will start initializing PHY based on config file.

static sai\_kv\_pair\_t kvpair;

void kv\_populate(int profile)

{

Kvpair[1]["SAI\_INIT\_CONFIG\_FILE"] = "phy\_1\_config\_file";

}

main () {

sai\_service\_method\_table\_t sai\_service\_method\_table;

kv\_populate();

sai\_service\_method\_table.profile\_get\_value = profile\_get\_value;

sai\_service\_method\_table.profile\_get\_next\_value = profile\_get\_next\_value;

Step 1: Load Vendor(Exmaple1) PHY SAI library

void\* phy\_1\_handle = dlopen("libsai\_phy\_example1.so", RTLD\_LAZY);

Step2 : Get Example1 PHY SAI sai\_api\_initialize & asic\_api\_query API’s

sai\_status\_t (\*phy\_1\_api\_initialize)(uint64\_t flags, const sai\_service\_method\_table\_t \*services);

sai\_status\_t (\*phy\_1\_api\_query) (sai\_api\_t api, void \*\*api\_method\_table);

\*(void\*\*)(&phy\_1\_api\_initialize) = dlsym(handle, "sai\_api\_initialize");

phy\_1\_api\_initialize(0, &sai\_service\_method\_table);

\*(void\*\*)(&phy\_1\_api\_query) = dlsym(handle, "sai\_api\_query");

sai\_switch\_api\_t\* phy\_1\_switch\_api = NULL;

**asic\_api\_query(SAI\_API\_SWITCH,(static\_cast<void\*\*> (static\_cast<void\*>(&phy\_1\_switch\_api))));**

Step 3: Initialize PHY 1

sai\_object\_id\_t phy\_1\_switch\_id;

sai\_attr\_set[0].id = SAI\_SWITCH\_ATTR\_INIT\_SWITCH;

sai\_attr\_set[0].value.booldata = 1;

sai\_attr\_set[1].id = SAI\_SWITCH\_ATTR\_SWITCH\_PROFILE\_ID;

sai\_attr\_set[1].value.u32 = 0;

sai\_attr\_set[2].id = SAI\_SWITCH\_ATTR\_SWITCH\_HARDWARE\_INFO;

sai\_attr\_set[2].value.u32 = “0x1000";

**sai\_attr\_set[3].id = SAI\_SWITCH\_ATTR\_HARDWARE\_ACCESS\_BUS\_TYPE;**

**sai\_attr\_set[3].value.u32 = SAI\_SWITCH\_HARDWARE\_ACCESS\_BUS\_TYPE\_MIDO;**

**sai\_attr\_set[4].id = SAI\_SWITCH\_ATTR\_HARDWARE\_ACCESS\_ADDRESS;**

**sai\_attr\_set[4].value.u32 = “1:0x1000”;**

sai\_attr\_set[5].id = SAI\_SWITCH\_ATTR\_FIRMWARE\_PATH\_NAME;

sai\_attr\_set[5].value.str = “/tmp/phyexample1.bin”;

**phy\_1\_switch\_api->create\_switch (&phy\_1\_switch\_id , attr\_count, sai\_attr\_set)));**

## Appendix:

Open Items:

Gearbox/Retimer Proposal (Dell)

1. How to put switch and Phy together, Tian will take a look on how it will impact Fboss.  
2. Need to add context to the call back for read and write. Tian will send to Ashok  
3. Credo to send Ashok incremental read API  
4. Allow the gearbox drive to use the MDIO sysfs bus, Guohan to send to Ashok  
5. Do we need a unique lane ID ?  
6. Link scan as open item for now

Response:

Broadcom:

**From:** Apo Gedik <apo.gedik@broadcom.com>   
**Sent:** Friday, September 13, 2019 11:40 AM  
**To:** Daparthi, Ashok; bithika@fb.com  
**Cc:** Craig Gava; Nishant Chadha  
**Subject:** RE: [MoM] MACSEC/Gearbox Meeting-7

[EXTERNAL EMAIL]

PAI team would like to have separate header files but if needed we can support common header, as well.

Facebook:

+ few guys from FB side.

1. How to put switch and Phy together, Tian will take a look on how it will impact Fboss

Sorry, folks, it took much longer than what I initially thought to close on this.

**tl;dr, FBOSS is onboard to extend existing SAI headers to cover retimer/gearbox/macsec use cases.**

The first challenge we got is how to satisfy ODR, when linking PHY SDK+SAI together with Switch SDK+SAI. SAI defines several global functions, so both Switch SDK and PHY SDK will define those functions with the same name. When linking them together in FBOSS, we get ODR violation. In order to work around this, we evaluated two solutions. Both solutions need to use shared library. One solution is to use dlopen()/dlsym(). The other solution is to downgrade all SAI symbols to LOCAL scope in the SDK+SAI shared lib.

The second challenge we spent lots of time debating internally was SAI backward compatibility. For example, in this pull request (<https://github.com/opencomputeproject/SAI/pull/854/files>), SAI\_SWITCH\_ATTR\_WARM\_RECOVER was added in the middle of the enum, which changes values of all enum after that line. That also means that NOS, switch SDK+SAI, and PHY SDK+SAI all need to rebuild using the same SAI version. With N vendors in between to provide Switch SAI and PHY SAI, it will be challenge to have all vendors using the same SAI version (pls correct me if I am wrong here. :-)). NOS can choose a version of SAI to compile switch SAI and PHY SAI. But that SAI version might not be what vendor has tested and qualified. This challenge is not limit to PHY SAI, but also to Switch SAI. But linking multiple PHY SAI together with switch SAI in the same process makes this more challenging.

2. Need to add context to the call back for read and write. Tian will send to Ashok

This shall not block sending the current PHY SAI proposal to the broader community to start review. I will either send the patch to Ashok or to SAI community depending on the status of the review.

Thx,

Tian

MSFT:

Common headers – MACSEC is may be common for ASIC and PHY.

Sonic it is not an issue for having common headers

Dell:

Dell is fine with common headers.