ADIN3310/ADIN6310 Software Driver User Guide

INTRODUCTION

This hardware reference manual is an overview of the TSN driver library providing examples of using the various features of the ADIN6310/ADIN3310 TSN switch, referred to as the Scalable Ethernet Switch (SES). The driver is designed to provide an API used for initialization, configuration and packet transmission and reception for the switch. The switches command and control messages are transferred via Ethernet, SPI or Quad-SPI.

The switch is a managed switch, where the host can be connected over SPI or MAC interface. The driver exposes various switch and TSN related configuration options. Using the driver API, the host can manage the Switch. It can set various switch & TSN related options like static entries, scheduling, preemption, VLAN tables, in addition to configuration of redundancy features.

A simplified block diagram of SES driver in a host application is shown below. The switch has an embedded microcontroller (Packet Assist Engine), that helps to manage the switch. The Packet Assist Engine runs a number of stacks and effectively offloads the connected Host processor. The firmware running on the packet assist engine is developed by Analog Devices. The TSN Driver library includes the firmware and the APIs in source code format for configuration of the switch functionality.

The driver package is available as a download from the switch product page, the header files detail the functions and parameters used for hardware configuration. This hardware reference manual should be consulted in conjunction with the Switch data sheet and user guide.

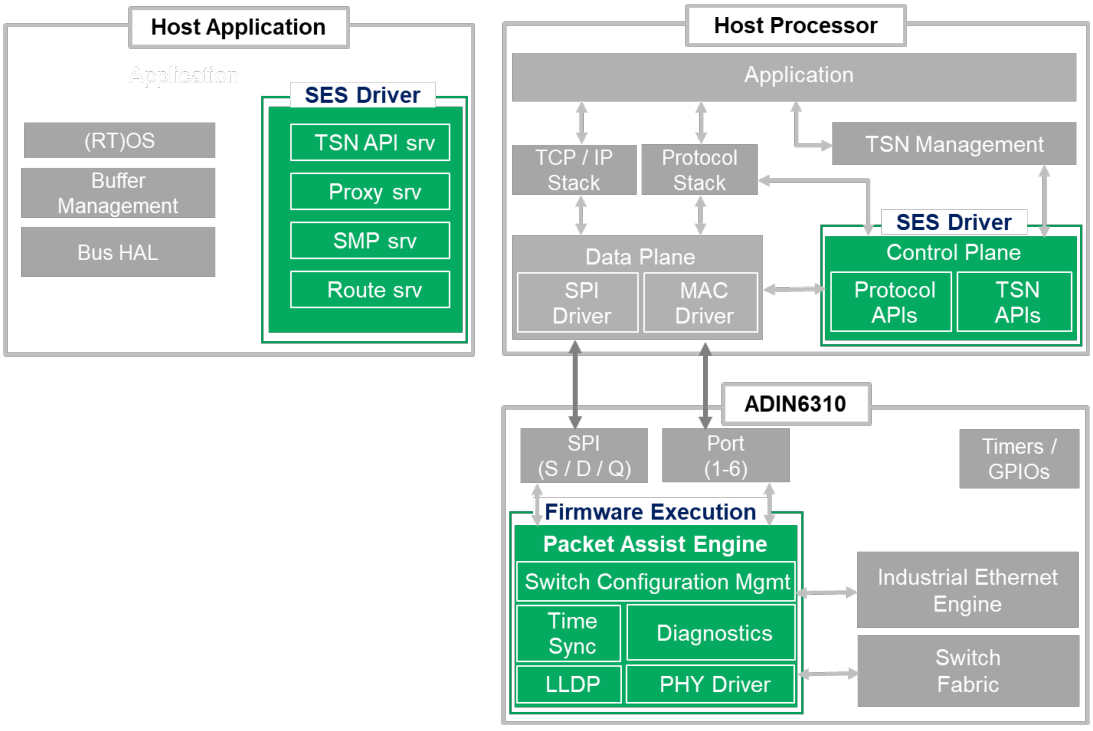

Figure 1. Driver Block Diagram

TABLE OF CONTENTS

TABLE OF CONTENTS

REVISION HISTORY

5/2026—Rev. B to Rev. C

TABLE OF CONTENTS

7/2025—Rev. A to Rev. B

TABLE OF CONTENTS

2/2025—Rev. 0 to Rev. A

TABLE OF CONTENTS

10/2024—Revision 0: Initial Version

DRIVER CONTENTS

The Switch driver package contains the software driver, supporting examples, driver documentation, and appropriate licenses for use of the driver. The driver software is provided as source code.

FOLDER STRUCTURE

The Switch driver package has the following folder structure:

- **Bin**: The 'bin' folder contains the firmware binary image. This image can be flashed onto the switch using various programming methods.
- **Doc**: The 'doc' folder contains driver documentation in compressed HTML format. The Doxygen format contains detail on all the APIs, enumerations, and variables. This folder also includes the driver release notes.
- **Examples**: The 'examples' folder contains three reference examples of the driver ported to the following two platforms
  - ses-windows-port-srv: Windows port for both serial interface & MAC/Ethernet Interface
  - ses-linux-port-srv: Linux with Ethernet interface
  - Linux User space driver with examples: ses-example-app, ses-tsn-configuration-application
- **Lic**: The 'lic' folder contains licensing documents related to the software driver provided in this package.
- **Src**: The 'src' folder has source code for the driver. It is divided into following:
  - ses-proxy-srv
  - ses-route-srv
  - ses-tsn-api-srv: this folder contains the header & source files for switch configuration/TSN configuration.
  - smp-stk
  - tsn-model-srv
- **wireshark**: The 'wireshark' folder contains the Switch Messaging Protocol (SMP) dissector.

DRIVER API

The driver APIs are used to configure the switch and are part of the 'ses-tsn-api-srv' folder. As the Switch is a managed switch, there will always be a host processor in the system configuring the switch functionality. The host processor will issue calls to these APIs to send command (configuration) packets to the switch over the desired host interface (choice of SPI or MAC interface). The switch supports hardware strapping to define where the host resides, refer to the Strapping and Configuration section in the switch data sheet. Refer to the Doxygen files in the "doc" folder for a full listing of APIs in conjunction with this document.

HOST COMMUNICATION WITH THE SWITCH

Communication Over Ethernet

The switch uses a Switch Messaging Protocol (SMP) protocol for communication with the host. The SMP messages are encapsulated within an Ethernet frame and are identified by the Ethertype value 0xAAC5 (big endian). An older Ethertype value, 0x88B5, may also be encountered. All fields defined as part of the SES-SMP protocol use little-endian byte ordering.

The SMP protocol appears as the payload of an Ethernet frame. Each SMP message consists of a size field, followed by one or more TLV (Type-Length-Value) fields. The Type field is composed of a 1-byte service code followed by a 1-byte function code. The Length field is a 2-byte unsigned value, and the Value field contains the payload, whose structure depends on the specific message.

The available SMP message types (service codes) for supported functions are defined in the smp-stk->SMP_services.h file. The final TLV in each SMP frame is always 0x00 (a service code of 0x00 with no additional data). This terminating TLV does not follow the general TLV structure and serves as an explicit end marker.

Switch Messaging Protocol Dissector

The driver library includes a Wireshark dissector plugin that provides decode support for the Switch Messaging Protocol (SMP). This dissector translates raw SMP Ethernet frames into a readable format, allowing users to inspect SMP message contents directly within Wireshark.

The dissector is intended as a debugging and validation tool. It enables user to analyze SMP communication between the host and the switch by decoding message headers, service and function codes, TLV fields, and payload data. This is particularly useful when troubleshooting host-switch communication, validating firmware update behavior, or verifying correct driver operation.

DRIVER CONTENTS

The use of the dissector is optional. Under normal operation, users are not required to understand or interact with SMP messages, as all SMP messaging and response handling are managed internally by the driver.

Install the Plugin

Wireshark searches specific directories for plugins at startup.

To install the dissector, copy the Lua files included in the driver library to Wireshark's user-specific plugin directory.

Common Location (Windows):

- %APPDATA%\Wireshark\plugins

If the directory does not exist, create in manually.

After copying the files, launch Wireshark. If Wireshark is already running, reload the plugins by selecting:

- Analyze → Reload Lua Plugins (or Press CTRL + Shift + L)

NOTE: Wireshark 4.3.x or newer is required.

Communication Over SPI

When the host uses SPI as preferred interface, the switch uses the TIMER0 hardware pin as an interrupt to indicate availability of data to the host. If the host uses Ethernet as preferred interface, then the TIMER0 pin does not act as interrupt pin and is available as a timer pin. For SPI mode, the host should connect the TIMER0 pin to an available GPIO pin on the host. A rising edge event on this pin should prompt the host to read the SMP message from the switch. In this interrupt service routine, the HAL interface read function must be called to read data from the switch and then call 'SES_ReceiveMessage()' API of the driver. Ideally dedicate a separate thread for this event or have a ISR. An example is provided as part of the examples folder in the driver package.

As soon as an interrupt from the switch is received, the host must trigger a HAL interface read of 4 bytes (uint32_t). The response from the switch for this interface read call would result in the length of bytes available to read from the switch. The host must call another HAL read with this length of bytes to the switch. Finally, the response data of this call must be passed to "SES_ReceiveMessage()'.

SPI Channel Reset

Resetting the SPI Read and Write Channels

The switch provides APIs to reset the SPI read or write channel to recover from SPI communication issues.

The following API is provided for this purpose:

```
SpiReadWriteReset(int tblIndex, bool read);
```

- int tblIndex - Identifies the SPI interface instance.
- bool read - Set to true to reset the SPI read channel, or false to reset the SPI write channel.

For example, if an error occurs during an SPI read operation, the TIMER0/INT pin will remain asserted. To clear this condition and restore normal operation, the SPI read channel can be reset using:

```
SpiReadWriteReset(tblIndex, true);
```

This action also deasserts the TIMER0/INT line.

Similarly, if an issue occurs during an SPI write operation, the corresponding write channel can be reset using:

```
SpiReadWriteReset(tblIndex, false);
```

DRIVER CONTENTS

Software Interface Statistics

This example demonstrates how to call SES_GetSoftwareInterfaceStatistics() to retrieve the current software-level Ethernet and SPI interface counters from the switch. These statistics are useful for interface health monitoring and for debugging communication issues such as dropped messages, buffer allocation failures, SPI command errors, and bus errors.

SPI Statistics Example

```
void PrintSoftwareInterfaceSPIStatistics_Example(void)
{
  SES_ethStatistics_t ethStats = { 0 };
  SES_spiStatistics_t spiStats = { 0 };
  int32_t status;

  /* When clearLocal is set to 1, the statistics will be cleared following the call. If clearLocal is
set to 0, then the statistics will accumulate. */
  status = SES_GetSoftwareInterfaceStatistics(&ethStats, &spiStats, 0);
  if (status != SES_OK) {
    printf("SES_GetSoftwareInterfaceStatistics failed: %ld\n", (long)status);
    return;
  }

  printf("\nSPI statistics\n");
  printf(" SPI mode : %lu\n", (unsigned long)spiStats.spiMode);
  printf(" SPI enabled : %lu\n", (unsigned long)spiStats.spiEnabled);
  printf(" SPI IRQ count : %lu\n", (unsigned long)spiStats.spiIrqCount);
  printf(" Sent messages : %lu\n", (unsigned long)spiStats.sentMessageCount);
  printf(" Received messages : %lu\n", (unsigned long)spiStats.receivedMessageCount);
  printf(" Invalid command errors : %lu\n", (unsigned long)spiStats.errorInvalidCommandCount);
  printf(" Bus errors : %lu\n", (unsigned long)spiStats.errorBusErrorCount);
  printf(" Read DMA errors : %lu\n", (unsigned long)spiStats.errorReadQDMAerrorCount);
  printf(" Write DMA errors : %lu\n", (unsigned long)spiStats.errorWriteQDMAerrorCount);
}
```

Ethernet Statistics Example

```
void PrintSoftwareInterfaceETHStatistics_Example(void)
{
  SES_ethStatistics_t ethStats = { 0 };
  SES_spiStatistics_t spiStats = { 0 };
  int32_t status;

  /* When clearLocal is set to 1, the statistics will be cleared following the call. If clearLocal is
set to 0, then the statistics will accumulate. */
  status = SES_GetSoftwareInterfaceStatistics(&ethStats, &spiStats, 0);
  if (status != SES_OK) {
    printf("SES_GetSoftwareInterfaceStatistics failed: %ld\n", (long)status);
    return;
  }

  printf("Ethernet statistics\n");
  printf(" Sent messages : %lu\n", (unsigned long)ethStats.sentMessageCount);
  printf(" RX low priority port0 : %lu\n", (unsigned long)ethStats.receivedLPMessageCount[0]);
  printf(" RX high priority port0 : %lu\n", (unsigned long)ethStats.receivedHPMessageCount[0]);
  printf(" Buffer allocation failures : %lu\n", (unsigned long)ethStats.bufferGetFailCount);
  printf(" Encoder send failures : %lu\n", (unsigned long)ethStats.sendFailCount);
```

DRIVER CONTENTS

```
  printf(" Collector receive failures : %lu\n", (unsigned long)ethStats.receiveFailCount);
  printf(" Encoder not ready count : %lu\n", (unsigned long)ethStats.encoderNotReadyCount);
}
```

FIRMWARE LOAD/UPDATE

The switch supports secure firmware updates over the host interface via the driver (either SPI or Ethernet firmware updates are supported). The firmware is provided as part of the TSN library package, each release of the driver software is paired with firmware for the packet assist engine. Devices are supplied unprogrammed, without firmware loaded. The first action the host must take is to load firmware. The firmware image will be less than 500 kB. The firmware & bootloader is included as part of the driver package, see the character array in SES_firmware.c & SES_bootloader.c files in ses-tsn-api-srv. The images can reside in host's memory and there would need to be sufficient space to accommodate, or alternatively have separate memory available to the host.

To ensure proper operation of the switch, the driver supports an automatic firmware update capability. On configuration of a device (by calling SES_AddDevice) the operational firmware version is checked and automatically updated if the value does not match the expected version. It is recommended that customers who disable this capability by defining the preprocessing macro below perform the version check manually and ensure the operational firmware matches the host version. Mismatched configuration will produce undesirable effects.

The host can disable this automatic firmware check / update process by enabling the following macro in the file "\src\ses-tsn-api-srv\inc\SES_firmware_update.h":

```
#define DISABLE_AUTOMATIC_FW_UPDATE
```

The automatic firmware update mechanism supports two methods for accessing firmware; reading it from a file or directly from the firmware character array (SES_firmware.c). By default, the firmware character array is part of the driver library, and firmware will be retrieved from this location during the update process when the driver is first run. The alternative approach of reading firmware from a file is useful when the host has firmware located elsewhere such as in external flash.

The firmware character array is part of the driver library, it can be included or excluded using the macro in SES_firmware_update.h `#define LINK_FIRMWARE_ARRAY .`

When the LINK_FIRMWARE_ARRAY macro is enabled, the automatic firmware update process retrieves the firmware from the character array. Conversely, if the macro is not defined, the firmware is read from the specified file path in SES_firmware_update_examples.h.

An example demonstrating how to read firmware from the file system is provided in the SES_firmware_update_examples.c file. This configuration offers flexibility in choosing the firmware source depending on the build requirements.

When a newly installed device is powered up, its bootloader checks and requests firmware from the host. The host must respond to these requests from the bootloader and support the process of loading firmware. This process will automatically start updating the firmware and the device will reset after a successful firmware update.

Pre-requisites on the host side

- The Application shall implement the call back function (callBackFunc_p) for handling the firmware update process event handling and progress monitoring.
- Only a valid firmware must be used to perform firmware update, invalid firmware will be rejected.
- The application must create memory block for the valid binary and share the pointer as an argument to the SES_ImageLoader function. Firmware can be passed as one single block or in blocks of 256, 512 or 1024.
- Include switch default destination address (79:c6:bb:ff:fe:00) in host address filtering

FIRMWARE UPDATE PROCESS

The firmware update process happens in stages. As a first step, the Switch is entered into programming mode. Initially, a secondary bootloader (part of driver) is loaded on to the switch. This happens in two stages: transferring header and transferring image. At each of these stages, the header / image is validated, and an appropriate status is sent via the callback up to the host application.

After the secondary bootloader is transferred, it requests a firmware image. The firmware is also transferred in two stages: transferring header and transferring image. Again, both are validated, and appropriate status is sent via the callback.

If there is an erroneous event during any of these stages, this will be indicated with appropriate status via callback.

The classification of headers and images happens behind the scenes (in the driver), and the host need not worry about it. If automatic firmware updates are allowed (DISABLE_AUTOMATIC_FW_UPDATE in SES_firmware_update.h is commented out), from the host point of view, it just needs to do a standard initialization for new firmware to be loaded. Alternatively, host can call one of the image loader APIs for manually update. The host does need to implement a callback to read the status as indicated by the SES during firmware update.

FIRMWARE LOAD/UPDATE

Figure 2 shows an example flow diagram during a normal firmware update:

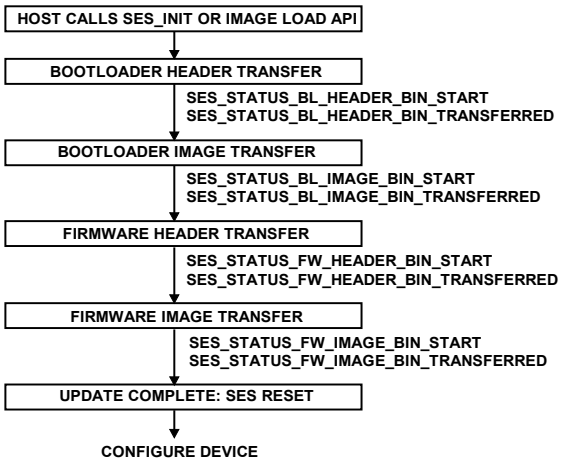

Figure 2. Firmware Update Flow

The following examples shows pseudo-code of call to the image loader APIs and callback implementation. Firmware can be loaded as a single block or provided in multiple blocks, the first example shows the firmware image being passed as a single block.

```
/*This example illustrate how to update the switch firmware by embedding the firmware character array
into the build,
It will read the firmware from charecter array and update the SES firmware
*/

/*To use the example below, ensure LINK_FIRMWARE_ARRAY is not disabled */
void SES_ImageLoaderSingleblock(void) {

    /*int8_t SES_ImageLoader(sesID_t sesId,
                      SES_fwUpdateStatusCallback_tp statusCallback_p,
                      SES_fwUpdateRequestSetup_tp requestSetup_p,
                      void *param_p,
                      SES_getImgBlk_tp imgBlkRequestFunc_p);*/

    int8_t rv = 0;
    sesID_t sesId = 0;

    printf("Updating firmware in single block \n");
    printf("Firmware API call status :: %d\n",SES_ImageLoader(sesId, SES_ImageLoader_callback, NULL,
NULL, GetImageBlockFromCharacterArray));
}
```

The following shows the callback implementation.

```
//=============================================================================
static void test_SES_ImageLoader_callback(uint8_t status) {

    switch(status) {
```

FIRMWARE LOAD/UPDATE

```
    // ERROR STATUSES
        case SES_IMAGE_SIZE_INVALID:
        case SES_DEVICE_NOT_RESPONDING:
        case SES_FWU_BUSY:
        case SES_SEC_HDR_VALIDATE_FAILED_RESPONSE:
        case SES_IMG_HASH_INVALID_RESPONSE:
        case SES_ABORTED_BY_DEVICE: {
            // Error states
            break;
        }
    // FIRMWARE UPLOAD STATUSES
        case SES_STATUS_BL_HEADER_BIN_START:
        case SES_STATUS_BL_HEADER_BIN_TRANSFERED:
        case SES_STATUS_BL_IMAGE_BIN_START:
        case SES_STATUS_BL_IMAGE_BIN_TRANSFERED:
        case SES_STATUS_FW_HEADER_BIN_START:
        case SES_STATUS_FW_HEADER_BIN_TRANSFERED:
        case SES_STATUS_FW_IMAGE_BIN_START:
        case SES_STATUS_FW_IMAGE_BIN_TRANSFERED: {
            // Firmware update status
            break;
        }
        default: {
            // Any other state. Should not get here!
            break;
        }
    } // switch(status)
}
```

FIRMWARE UPDATE FROM FILE

The following example shows firmware being provided in multiple blocks. The supported block sizes are 256, 512 or 1024 bytes, all blocks should be the same size, with exception of the last block, which can be less than or equal to the chosen block size.

```
int32_t SES_FirmwareUpdateFileSytemExample() {
/* sesId                    SES ID
*  statusCallback_p         Firmware update status callback function pointer
*  requestSetup_p           Optional function pointer to application specific setup function
*  param_p                  Optional parameter to be passed to setup callback
*  imgBlkRequestFunc_p      Function pointer to the image block retrieval function
*/

    /*int8_t SES_ImageLoader(sesID_t sesId,
        SES_fwUpdateStatusCallback_tp statusCallback_p,
        SES_fwUpdateRequestSetup_tp requestSetup_p,
        void* param_p,
        SES_getImgBlk_tp imgBlkRequestFunc_p);*/

    int8_t rv;
    sesID_t sesId = 0;

    char* path = "bin/firmware/ses-app.bin";
    rv = SES_ImageLoader(sesId, SES_ImageLoader_callback, SES_ImageLoadFromFileSetup, path,       GetI▶
mageBlockFromFile);
```

FIRMWARE LOAD/UPDATE

```
    return rv;
}
```

There are reference examples included in SES_firmware_update_examples.c showing how to read firmware from the file system or the character array. The examples are provided as a reference and modifications will be required to adapt to different platforms. The two examples are as follows:

- GetImageBlockFromCharacterArray: this is used when the firmware and bootloader character arrays are directly linked into the SES driver code by defining LINK_FIRMWARE_ARRAY.
- GetImageBlockFromFile: uses a file system implementation to read in blocks of data from binary files representing the bootloader and firmware.

To enable reading firmware and bootloader files from the file system, an application-specific file operation porting layer must be implemented by the user. The following functions need to be implemented by the user as part of this porting layer:

- stat_wrapper: Application-specific implementation of the stat function.
- fopen_wrapper: Application-specific implementation of the fopen function.
- fread_wrapper: Application-specific implementation of the fread function.
- fclose_wrapper: Application-specific implementation of the fclose function.

INITIAL FIRMWARE LOAD – IMAGE REQUESTS

Firmware Load Over Ethernet

When the host interface is strapped for Ethernet, a new unprogrammed device will send image requests to the host over the Ethernet interface. These messages image requests have a source address of 7a:c6:bb:ff:fe:00 and a destination address of 79:c6:bb:ff:fe:00, they will be interpreted by the driver and initiate the firmware load over the host interface. All switches will have these same source and destination addresses.

Example of an image request frame:

Message #1: image request:

- Destination MAC address Multicast address: 79:c6:bb:ff:fe:00
- Source MAC address Unicast address: 7a:c6:bb:ff:fe:00
- Ethertype 0x88b5 or 0xAAC5
- Frame:

79 c6 bb ff fe 00 7a c6 bb ff fe 00 88 b5 11 00

08 00 0c 00 00 00 00 00 00 00 00 00 53 45 43 48

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00

Message #2: image request (NAK):

- Destination MAC address Multicast address: 79:c6:bb:ff:fe:00
- Source MAC address Unicast address: 7a:c6:bb:ff:fe:00
- Ethertype 0x88b5 or 0xAAC5
- Frame:

79 c6 bb ff fe 00 7a c6 bb ff fe 00 88 b5 05 00

08 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

FIRMWARE LOAD/UPDATE

00 00 00 00 00 00 00 00 00 00 00 00

Firmware Load Over SPI

When the host interface is hardware strapped for SPI mode (Single/Dual/Quad), the TIMER0/INT pin acts as an interrupt to the host to tell the host the switch is ready/has some message to read.

When the device has no image loaded, the Timer0/INT signal goes high approx. every 1.25second with a low pulse duration of 2 ms. This prompts the host to read from the switch, which initiates the firmware load process. Once the firmware load has started, the operation of the Timer0/INT reverts to normal operation, going high when there is a message for the host to read from the switch. Example of an image request over SPI is shown in Figure 3.

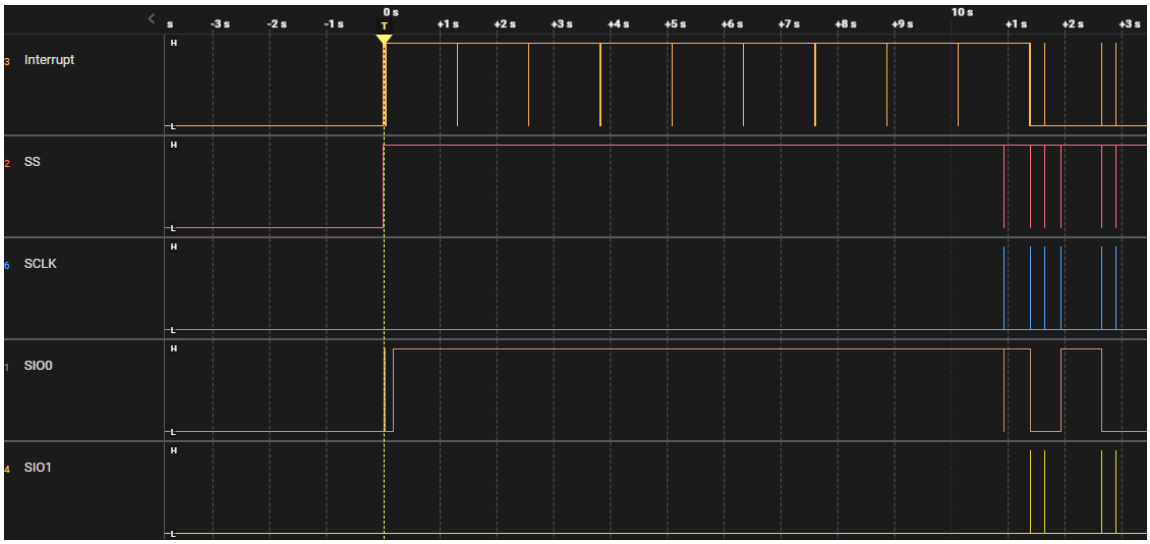

Figure 3. Overview of firmware image request over SPI

FIRMWARE VERSION

The driver library and firmware are paired and should always match. When a new version of driver and firmware is run, the automatic firmware update process can update firmware running on the switch. If the firmware update fails (or) the Switch already has the latest firmware, it would continue with the existing firmware, skipping the automatic firmware update process.

Firmware version can be read back from the device using the SES_GetFirmwareInfo() API, it will return the firmware component information in the form SC0000519-00X-xxx where the "X" corresponds to the major release version and "xxx" the minor release. The version information can be cross referenced against the software release notes included in the driver package to understand changes and known issues.

FIRMWARE UPDATE AND SECURITY REVISIONS

The security versions for the switch firmware may change during the development and to ensure there are no issues with rolling back to previous versions of firmware with different security versions, a number of APIs were included in SES_switch.h.

The added APIs are used to perform checks on the security version of the switch to the rollback counter (SES_CheckSecurityVersion()), if needed the SES_UpdateSecurityVersion() can update the rollback counter and SES_GetSecurityVersionInfo() enables the host to get the firmware security version. SES_CheckSecurityVersion() is called as part of the SES_AddDevice() routine during initial device configuration.

NOTE: If a call to SES_CheckSecurityVersion() returns an error, then users should call SES_UpdateSecurityVersion in order to update the security version on the device. This will set the rollback counter to the user flash security version value from the secure boot header.

FIRMWARE ERASE

Calling the SES_ImageLoader API triggers a complete firmware reload sequence. As part of this process, the device firmware is erased before the new firmware image is requested and programmed. The firmware erase phase corresponds to the following state:

- SMP_FLASH_ERASE_SUCCESS

FIRMWARE LOAD/UPDATE

This state is defined in the SES_firmware_update_control source code, within the HandleProgrammingMessage function. If the firmware update process is exited or halted while the system is in this state, the existing firmware will have already been erased.

Once the firmware has been successfully erased, the switch begins issuing image request messages. For an Ethernet host, these requests are sent over Ethernet. For a SPI host, the requests are indicated through TIMER0/INT interrupts.

NOTE: This behavior is intended for development and debugging purposes only.

PORTING AND USING THE DRIVER

The TSN Driver library software is OS agnostic. Because the driver must run on a host (the switch is a managed switch), be it a PC or an embedded microcontroller, porting examples are provided so that any OS and hardware related functions can be implemented by the user. In a general sense, the porting APIs are divided into following categories:

- (RT)OS Semaphore: create, acquire, release, and delete
- Buffer Management: allocate, and free memory
- HAL interface: initialize, send, and receive

The image below, depicts typical application flow and various components involved. There are some blocks that must be implemented by the user based on his/her host device.

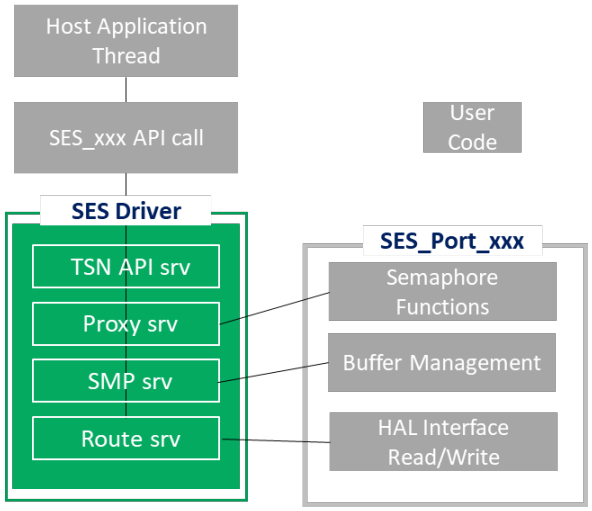

Figure 4. Typical Application Flow

Two example implementations of porting layer are provided as reference in the examples folder. For porting the driver to a target environment, pick the appropriate examples and follow the below steps.

1. Take a copy of one of the example folders (ses-windows- port-srv or ses-linux- port-srv)
2. Rename the copied folder to **ses-<platform>-<interface>-port-srv**
3. Rename **ses-<platform>-<interface>-port-srv\src\SES_PORT_XXXX.c** to **SES_PORT_<interface>.c**
   a. **<platform>**: Windows, Linux, FreeRTOS, Zephyr OS, etc.,
   b. **<interface>**: Ethernet, SPI, QuadSPI, USB, Serial etc.,
4. Implement the APIs defined in **SES_PORT_<interface>.c** as per target platform and interface.
5. The API declarations are provided in **src\ses-port-srv\inc\SES_PORT_api.h**

PORTING LAYER APIS

The following APIs contained in the SES_port_api.h must be implemented by the user:

- Memory Management
  - int32_t **SES_PORT_Malloc** (void **buf_pp, int size);
    - Allocates a buffer of at least the requested size and returns its address via buff_pp. Returns SES_PORT_OK on success, SES_PORT_BUF_ERR if the input pointer is null.
  - int32_t **SES_PORT_Free** (void *buf_p);
    - Releases a previously allocated buffer. Returns SES_PORT_OK on success, SES_PORT_BUF_ERR if the input pointer is null.
- Semaphore Management
  - void* **SES_PORT_CreateSemaphore** (int initCount, int maxCount);
    - Creates a semaphore with the specified initial and maximum count. Returns a handle to the semaphore or NULL on failure.

PORTING AND USING THE DRIVER

- int32_t **SES_PORT_WaitSemaphore** (int* semaphore_p, int timeout);
  - Waits (takes) the semaphore referenced by semaphore_p, with a timeout. Returns SES_PORT_OK on success, SES_PORT_TIMEOUT on timeout, or SES_PORT_INVALID_PARAM if the handle is invalid.
- int32_t **SES_PORT_SignalSemaphore** (int* semaphore_p);
  - Signals (gives) the semaphore referenced by semaphore_p. Returns nonzero on success, zero on failure, or SES_PORT_INVALID_PARAM if the handle is invalid.
- int32_t **SES_PORT_DeleteSemaphore** (int* semaphore_p);
  - Deletes the semaphore referenced by semaphore_p. Returns nonzero on success, zero on failure, or SES_PORT_INVALID_PARAM if the handle is invalid.

- Interface Management (Typedefs for function pointers)
  - **SES_PORT_init_t**
    - Initializes a hardware interface (Ethernet or SPI). Takes interface-specific parameters and returns the interface type and MAC address. Returns SES_PORT_OK on success.
  - **SES_PORT_release_t**
    - Releases an initialized hardware interface. Returns SES_PORT_OK on success, SES_PORT_ERROR on error, or SES_PORT_NOT_INITIALIZED if not initialized.
  - **SES_PORT_sendMessage_t**
    - Sends a message over a hardware interface. Returns SES_PORT_OK on success, or error codes on failure.
  - **SES_PORT_updateFilter_t**
    - Updates the frame filter for a hardware interface, adding or removing MAC addresses or Ethertypes. Returns SES_PORT_OK on success, or error codes on failure.
  - **SES_PORT_getInterfaceInfo_t**
    - Retrieves a comma-separated string of interface information. Returns SES_PORT_OK on success, or error codes if the buffer is too small or memory allocation fails.
  - **SES_PORT_handleError_t**
    - Handles a reported SPI error by executing the appropriate recovery actions, such as resetting the affected SPI read or write channel, based on the specific error types detected.

Additional Porting Layer APIs

The following APIs are contained in the SES_port_api.h which are already defined:

- Message Receive Interface
  - **SES_ReceiveMessage**
    - The SES_ReceiveMessage API is defined in the route services and should be called when a message is ready to be processed.
- Timeout Interface
  - **SES_PORT_HandleTimeout**
    - Handles RPC timeouts and cleans up expired response buffers. Should be called periodically. Returns SES_PORT_OK on success.

The SES_PORT_HandleTimeout is intended for monitoring RPC timeout conditions and handling the cleanup of response buffers associated with timed-out or pending RPC transactions. A periodic task can invoke this API to identify and clear response buffers that have exceeded their timeout thresholds, ensuring proper resource management and avoiding stale entries. The frequency of each API call depends on how many ADIN3310/ADIN6310 devices is the host communicating with and the bandwidth of the host system. The frequency should increase the more devices the host is communicating with or if the host is configuring the device frequently. Response buffers are dynamically allocated when RPC messages are sent, allowing the system to correctly match incoming response messages with their corresponding blocking API calls.

NOTE: If the user fails to call SES_PORT_HandleTimeout, any RPC messages that are sent but do not receive a response (e.g., due to a timeout condition like SES_TIMEOUT) will result in the loss of the corresponding response buffer. Over time, if too many response buffers are lost, communication with the SES interface may be disrupted, as no new buffers can be allocated.

PORTING AND USING THE DRIVER

A simple task/thread can be created where user can check and perform the cleanup activity for pending RPC responses. The below code is calling cleanup activity on every 100mSec.

```
void SES_TimeOutTask_Example(void)
{
    while(1) {
        //check RPC timeout and perform cleanup for pending RPC responses
        SES_PORT_HandleTimeout(100);
    }
}
```

Message Tap Debug and Data Capture

A debug and data capture option is supported between the host and the switch. This feature allows users to log SPI or Ethernet request and response traffic and capture it in Wireshark-compatible Packet Capture (PCAP) format for analysis and debugging.

To support this functionality, the following porting layers have been provided:

- SES_PORT_message_tap.h
- SES_PORT_spi_analysis.h

To capture data transactions between the host and the switch, the macro "ENABLE_MESSAGE_TAP" must be defined. When this macro is enabled, all relevant message traffic is recorded into a PCAP file. The provided example implementation records and stores these packets in a standard PCAP format.

By default, the capture file is stored at the following location:

- message_log/capture.pcap

The default file path and related configurations are defined in SES_PORT_message_tap.h. The generated PCAP file can be opened using Wireshark and is intended to assist users in diagnosing configuration and communication issues between the host and the switch.

NOTE: The example implementation uses Windows-specific I/O functionality. For deployment on an embedded target, the file I/O implementation must be adapted accordingly.

Message Tap APIs (SES_PORT_message_tap.h)

This header defines the interface for a diagnostic message-tap facility that capture SPI or Ethernet commands sent to the switch and records them as PCAP files. This functionality is part of the platform porting layer and provides the following APIs:

- SES_PORT_LogMessage
  - Appends a raw packet, including the Ethernet header, to an existing PCAP file.
- SES_PORT_LogHeaderlessMessage
  - Appends a raw packet without an Ethernet header to an existing PCAP file.
  - A dummy Ethernet header is added internally.
- SES_PORT_CreatePcapFile
  - Creates a new PCAP file at a specified path and writes the global PCAP header.
- SES_PORT_DeletePcapFile
  - Deletes an existing PCAP file.

SPI Analysis (SES_PORT_spi_analysis.h)

This header defines the interface for instrumenting SPI bus traffic between the host and the switch. It allows users to install callback hooks that are invoked around SPI transactions, enabling real-time monitoring, analysis, or controlled modifications of data transfers.

The following interfaces are provided:

- SES_PORT_SpiWriteHook_t
  - Callback type invoked before an SPI write, allowing inspection or modification of the outgoing data and length.

PORTING AND USING THE DRIVER

- SES_PORT_SpiHeaderReadHook_t
  - Callback type invoked after an SPI header read completes, providing access to the received header data.
- SES_PORT_SpiDataReadHook_t
  - Callback type invoked after an SPI data read completes, providing access to the received payload data.
- SES_PORT_SPI_EnableReadOperations
  - Enables or disables SPI read operations on a given hardware interface handle.
  - It will block if a read operation is currently in progress.
- SES_PORT_SPI_InstallOperationHooks
  - Registers callback hooks for SPI write, header-read, and data-read operations.
  - Hooks may be selectively omitted by passing NULL.

BUILD ENVIRONMENT

The Switch driver can be built as 32-bit or 64-bit. Toolchains used by ADI for testing were Visual Studio version 2015, 2022 Community version, 2026 Community version, and Linux GCC version 11.30.

To build the driver along with host application, the following directories must be on the included path of the project:

- Any one of the ses-<platform>-<interface>-port-srv/inc
- ses-proxy-srv/inc
- ses-route-srv/inc
- ses-tsn-api-srv/inc
- smp-stk/inc
- tsn-model-srv/inc

The files in the following directories must be added to the build source:

- Any one of the ses-<platform>-<interface>-port-srv/src
- ses-proxy-srv/src
- ses-route-srv/src
- ses-tsn-api-srv/src
- smp-stk/src

Finally, follow the build steps as per the platform build steps.

NOTE: The SPI porting layer example included in the driver package is based on libFT4222 and uses FTDI USB to SPI evaluation module (UMFT4222EV). If using Raspberry Pi SPI interface or other platforms, some changes are required in the porting layer see Porting to Raspberry Pi SPI Interface (SPIDEV).

The Switch drivers allows feature sets to be excluded at build time. When a specific feature is chosen to be excluded, all components associated with the feature must be removed consistently from the driver build. This means excluding the feature's API set from the internal registration mechanism (SES_api_sets.c), and ensuring the corresponding source files are not compiled. Removing a feature's source files without updating the internal API set registration will result in linker errors, while removing a feature from the registration without excluding its source files will result in unused code.

EXAMPLE PROJECT

To help the user become familiar with the switch product prior to porting to own host platform, there is an example project available to run on a Windows platform. It is available on Github and linked from the ADIN6310/ADIN3310 product page. This project serves as a foundational template that demonstrates key features, workflows, and best practices for using the switch. The Windows platform is only a vehicle to demonstrate capability. For real applications, user will connect their own host to the switch and port the driver to that platform. The examples provided in this user guide are included the example project.

Overview

The Windows Project is an example that showcases:

PORTING AND USING THE DRIVER

- **Core Features**: How to integrate and use key features of the switch in a Windows application.
- **Setup and Configuration**: Step-by-step guidance on configuring the project for the network features using the configuration file.
- **Customizable Code**: Modular components that can be easily modified to suit different use cases, in terms of which kind of hardware interface (Ethernet or SPI) is used and what switch capability is required in the network.

Getting Started

1. **Download the Project**: The example project is available to download from the product page.
2. **System Requirements**: Ensure your system meets the following prerequisites:
   - Operating System: Windows 10 or later
   - Software: [e.g., Visual Studio or other dependencies]
   - Other Requirements: [e.g. drivers or third-party libraries], refer to the readme section in the repository for this.
3. **Open the Project**: Follow these steps to open the sample project:
   - Launch [IDE or editor, e.g., Visual Studio].
   - Open the solution file (e.g., SampleProject.sln) located in the project directory.
   - Modify the configuration for the ADIN3310/ADIN6310 hardware being used and host information.
4. **Run the Application**:
   - Build the project using the [build settings].
   - Connect to the hardware over SPI or Ethernet
   - Run the project to exercise the switch functionality

After exploring the example project the user will have better understanding of the switch capability and can transition to developing own application on desired host.

LINUX USER SPACE DRIVER

Included in the examples folder is a user-space Linux driver compatible with Linux-based operating systems and can be used to configure the Switch. There are two examples provided, which can be compiled on a Linux host once all prerequisite toolchains are installed. These examples demonstrate how to build the user space driver on a Linux platform.

1. **ses-example-app**:

   This example demonstrates how to establish communication between the host and the switch and performs a very basic configuration of the switch using the port configuration parameters. The application is performed exclusively through the driver APIs and does not require any external network configuration tools such as NETCONF or Netopeer.

   Application Flow

   During execution, the application first extracts all port-related configuration data from the XML file using helper functions implemented in

   - ses-example-app/src/EX_parse_board_settings.c

   Using the extracted configuration data, the application then performs basic switch initialization by calling the following APIs in sequence:

   - EX_GetBoardSettings: Retrieves per-port configuration information from example-eval-adin6310.xml.
   - SES_Init: Performs software initialization of the switch interface.
   - SES_AddHwInterface: Adds the hadrware interface used by the host to communicate with the switch.
   - SES_AddDevice: Adds an SES device and performs basic device initialization
   - SES_InitializePorts: Initiazes the switch ports associated with the specified MAC address.
   - Exit: Terminates the application after succesful completion of the basic configuration.

2. **ses-tsn-configuration-application**:

   This example demonstrates an advanced configuration workflow that focues on Time Sensitive Networking (TSN) features. In addition to performing the initial switch configuration, the application sets up a sysrepo interface, enabling switch management through NETCONF using a Netopeer client.

   Similar to the basic example, the application uses the eval-adin6310.xml file for initial switch bring-up and basic port configuration. Once the switch is configured, the application initializes a sysrepo session and remains active to listen for and process configuration requests received from the Netopeer client.

   Sysrepo Datastores

   The application uses the following sysrepo datastores as part of the host application configuration model:

   - **Startup Datastore**

     Stores the configuration loaded during system startup and represents the intended configuration state of the device.
   - **Running Datastore**

     Contains the active configuration currently applied to the device. This datastore is typically modified during normal operation.
   - **Candidate Datastore**

     Used to stage configuration changes prior to committing them to the running datastore, allowing validation and consistency checks before application.

   Application Functionality

   The application performs the following operations:

   - Initializes switch ports based on the configuration specified in the XML file.
   - Enables Precision Time Protocol (PTP) with a single instance using the peer-to-peer delay mechanism.
   - Enableds Link Layer Discovery Protocol (LLDP).
   - Starts the sysrepo service and waits for NETCONF requests from the Netopeer client.

The following steps outline the build process for compiling the examples in a Linux platform. The examples have been tested with the specific versions of dependencies listed in the README.txt file included in the example folder. Compatibility with other versions is not guaranteed and may impact ability to build the application.

LINUX USER SPACE DRIVER

SES-EXAMPLE-APP BUILD AND EXECUTION

Detailed instructions for the complete build process, including the installation of required dependencies, are provided in the README.txt file located in the respective example folders (e.g., ses-example-app, ses-tsn-configuration-app).

1. **Organize Source Files**: Copy all driver-related source files, along with the Linux driver porting layer and example applications, into a single directory. After merging the directory should contain all these subfolders :
   - linux-file-system-srv
   - linux-hwpl-srv
   - linux-ospl-srv
   - ses-example-app
   - ses-linux-port-srv
   - ses-tsn-configuration-app
   - sysrepo-glue-srv
   - ses-proxy-srv
   - ses-route-srv
   - ses-tsn-api-srv
   - smp-stk
   - tsn-model-srv
2. **Install Required Tools:** Ensure the following dependencies are resolved:
   - pcap: Tested with version 1.10.1 `sudo apt-get install libpcap-dev`
   - libyang: Clone libyang from source. Tested with version 2.0.164
     - while installing libyang, also install PCRE2 by using this command: `sudo apt install -y libpcre2-dev`
     - `git clone --branch v2.0.164 https://github.com/CESNET/libyang.git`
     - `cd libyang`
     - `mkdir build && cd build`
     - `cmake .. && make`
     - `sudo make install`
   - libFT4222: Tested with version 1.4.4.44
     - `curl https://ftdichip.com/wp-content/uploads/2021/01/libft4222-linux-1.4.4.44.zip -o libft4222-linux-1.4.4.44.zip && unzip libft4222-linux-1.4.4.44.zip && tar -xvzf libft4222-linux-1.4.4.44.tgz`
     - `./install4222.sh`
3. **Build**:
   - `cd ses-example-app/project/cmake/`
   - `cmake -S . -B build/`
   - `cmake --build build`
4. **Run Example Application**: Once the build is successful, use the following command to run the example application:

```
ses-example-app -f <xml-filepath> -p <interface-selection> [-m <yang-modules-folder>] [-s <0,1,2,4>
SPI-mode] -i <production> [-l SPI interface info]
```

   Example for Ethernet interface:

   - `./ses-example-app -f example-eval-6310.xml -p d8:3a:dd:13:ad:db -m sysrepo-glue-srv/modules -s 0 -i production`

   Example for SPI interface:

   - `./ses-example-app -f example-eval-6310.xml -p 0 -m sysrepo-glue-srv/modules -s 1 -i production -l`
5. Upon successful execution, the following output will be displayed:

LINUX USER SPACE DRIVER

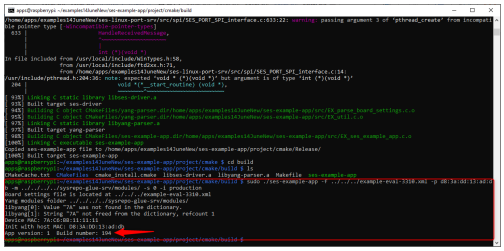

Figure 5. Terminal showing successful execution of ses-example-app

SES-TSN-CONFIGURATION-APP BUILD AND EXECUTION

1. **Install Required Tools:** To build and execute "***ses-tsn-configuration-app***" following dependencies must be installed along with the dependencies resolved in previous step:
   - sysrepo: Tested with version 2.1.42
     - `git clone --branch v2.1.42 https://github.com/sysrepo/sysrepo`
     - `cd sysrepo`
     - `mkdir build && cd build`
     - `cmake .. && make`
     - `sudo make install`
   - libssh
     - `git clone https://git.libssh.org/projects/libssh.git`
     - `cd libssh`
     - `mkdir build`
     - `cd build`
     - `cmake -DWITH_GCRYPT=OFF ..`
     - `make`
     - `sudo make install`
   - libnetconf2
     - `sudo apt-get install libssh-dev uncrustify libcurl4-openssl-dev`
     - `git clone --branch v2.1.7 https://github.com/CESNET/libnetconf2`
     - `cd libnetconf2`
     - `mkdir build && cd build`
     - `cmake .. && make`
     - `sudo make install`
2. **Build**
   - `cd ses-tsn-configuration-app/project/cmake/`
   - `cmake -S . -B build/`
   - `cmake --build build`
3. **Run Application**: Once the build is successful, use the following command to run the application:

```
ses-tsn-configuration-app -f <xml-filepath> -o <output-filepath> -p <interface-selection> [-s
<0,1,2,4> SPI-mode] -i < production> [-l SPI interface info]
```

Example for Ethernet interface:

- `./ses-tsn-configuration-app -f eval-adin6310.xml -o test -p d8:3a:dd:13:ad:db -s 0 -i production`

Example for SPI interface (UMFT4222EV):

LINUX USER SPACE DRIVER

```
./ses-tsn-configuration-app -f eval-adin6310.xml -o test -p 0 -s 1 -i production -l
```

DEVICE CONFIGURATION

There are two approaches to enable TSN functionality and configure the switch. The first approach is by calling respective functions in API and second uses NETCONF.

Configuration using Driver API:

The driver APIs can be utilized to configure the switch in a Linux environment. In the "ses-example-app" folder, the source file "src/EX_ses_example_app.*c*" can be modified to add the necessary API calls. After making these modifications, the project can be recompiled following the previous build process, and then used to configure the switch. Examples for the various features are included in this reference manual.

Configuration using NETCONF:

NETCONF protocol defines a mechanism for network device management, retrieving configuration information and modifying device with new configuration data It uses remote procedure call (RPC) messages encoded in XML for communication between server and client. The NETCONF server is the network device, while client is the application running as part of network manager.

YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF, NETCONF remote procedure calls, and NETCONF notifications. YANG is used to model the operations and content layers of NETCONF.

Netopeer2 is a server for implementing network configuration management based on the NETCONF Protocol. The Netopeer2 server uses sysrepo as a NETCONF datastore implementation.

Sysrepo is a YANG-based configuration and operational state data store for Unix/Linux applications. Applications can use sysrepo to store their configuration data modeled by YANG model. Sysrepo and libyang are an open-source project based on Linux only, documentation available online here: https://www.sysrepo.org/.

The TSN driver library examples folder includes sysrepo-glue-srv. This asset provides the glue code/translation layer required to interface sysrepo datastores with the Switch TSN Driver API, see Figure 6.

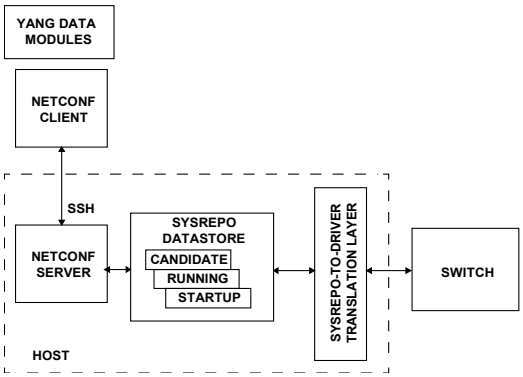

Figure 6. NETCONF, SYSREPO and glue

To configure the switch using a NETCONF client/server, it is necessary to install Netopeer2.

Install netopeer2:

1. `git clone -b v2.1.16 https://github.com/CESNET/netopeer2.git`
2. `cd netopeer2 && chmod +x scripts/setup.sh && chmod +x scripts/merge_*.sh && mkdir build && cd build && cmake -DCMAKE_BUILD_TYPE:String=Release .. && make && sudo make install`

Once installation is complete, using the following steps to configure the switch:

1. Run the ses-tsn-configuration-app as discussed earlier.
2. Start Netopeer2 server with command: `sudo netopeer2-server -v3 -d`

LINUX USER SPACE DRIVER

3. Once server is started, the terminal output should display similar to Figure 7

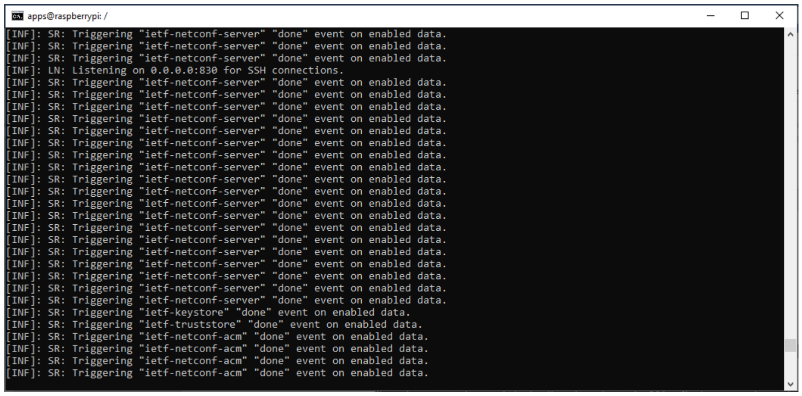

Figure 7. Running NETCONF

1. To start Netopeer client, use following commands: `sudo netopeer2-cli` and `connect --login <username> --host <hostname> --port 830`

The NETCONF Netopeer client can be started either on same host in different terminal or on other devices using SSH connection. In this example, netopeer-server, netopeer-client and ses-tsn-configuration-app are running on Raspberry pi in different terminals and Raspberry pi is connected to windows PC via SSH.

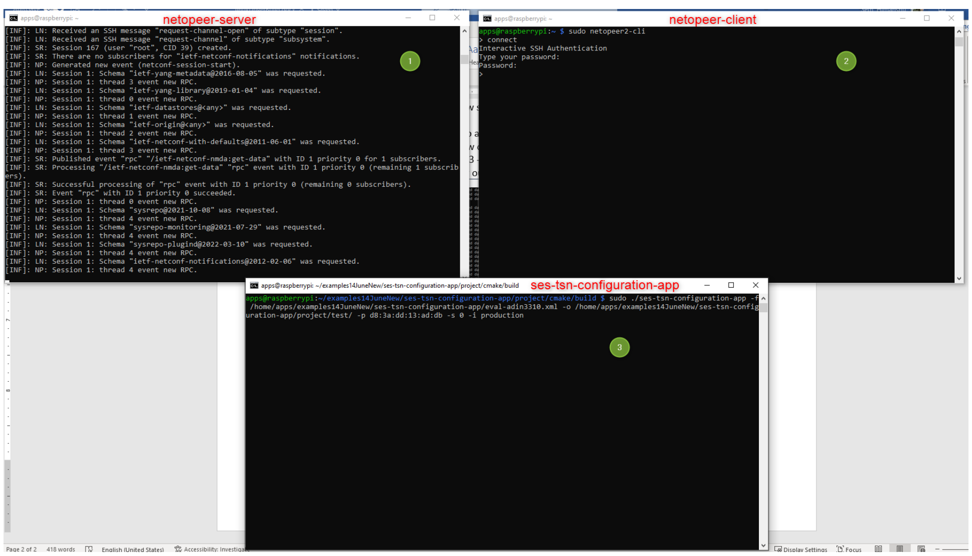

Figure 8. NETCONF Server, Client, ses-tsn-configuration-app

Netopeer-client will perform SSH key authentication and ask for the root password if connected as root user. Once client is connected, the configuration can be performed using get and edit operations. In the terminal window shown in Figure 9 the running configuration is read and preemption is enabled in Port0:

LINUX USER SPACE DRIVER

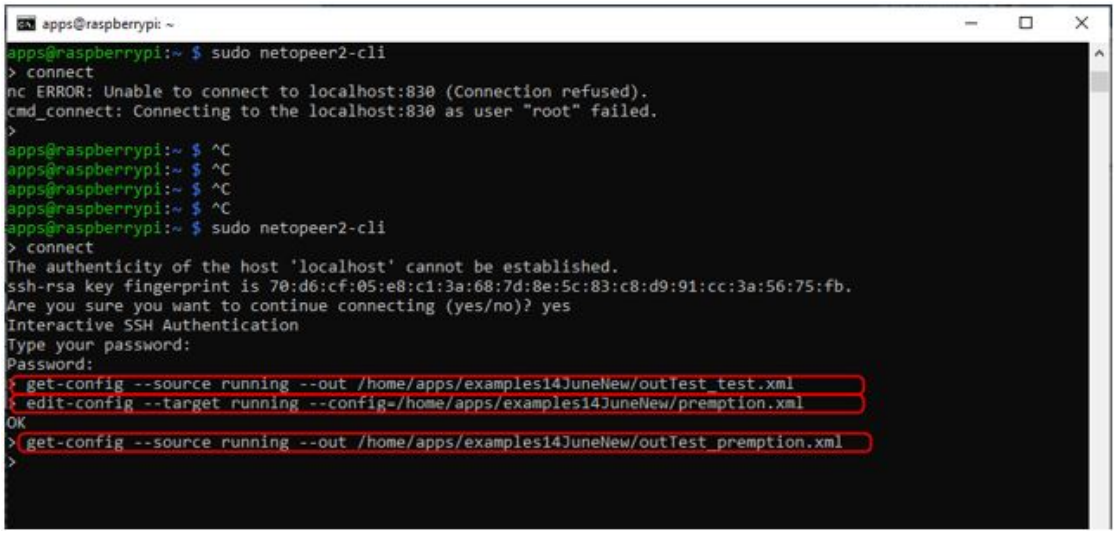

Figure 9. NETCONF read and edit Preemption configuration

Once edit-config is successful, the running config can be retrieved and compared as shown in the XML files Figure 10.

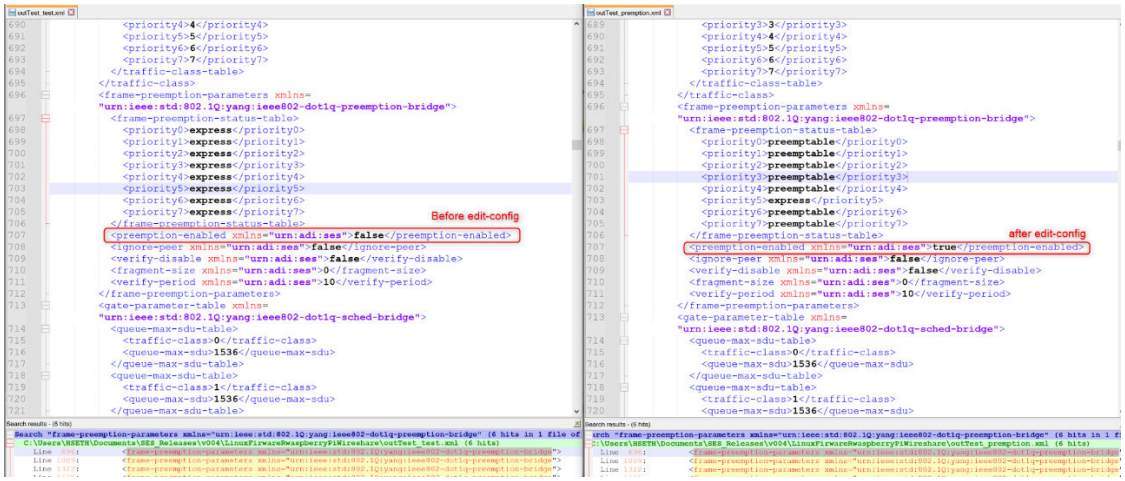

Figure 10. Preemption Configuration edit

PORTING TO RASPBERRY PI SPI INTERFACE (SPIDEV)

The TSN driver includes porting example for how to port to the on-board Raspberry Pi SPI interface. This uses the ioctl, gpiod, and spidev libraries.

The Raspberry Pi provides only a single SPI interface for external communication. The Quad-SPI (QSPI) interface is reserved for RPI internal use and is not available for general-purpose external communication.

NOTE:

The Raspberry Pi SPI interface must be enabled before use. When interfacing the switch with the Raspberry Pi, the VDDIO rail must be configured to 3.3V. The required jumper settings are as follows:

- EVAL-ADIN3310: P2(1-2)
- EVAL-ADIN6310: P3(1-2), P4(2)-P5(1), P33(OPEN).

The SPI examples are included in the TSN driver library under the “ses-linux-port-srv/src/spi” folder:

1. **Using the FT4222 SPI Dongle:** SES_PORT_FT4222_interface.c

LINUX USER SPACE DRIVER

2. **Using SPIDEV:** SES_PORT_SPIDEV_interface.c

In these reference examples, the following functions are implemented to support communication with the switch:

- SES_PORT_SPI_Init
- SES_PORT_SPI_Release
- SES_PORT_SPI_SendMessage
- SES_PORT_SPI_GetInterfaceInfo

To build with SPIDEV, in addition to all the dependencies mentioned in the Ses-example-app Build and Execution section, libgpiod must be installed to support switch communication over SPIDEV:

1. libgpiod (only required with SPIDEV=ON): Tested with version 2.1.13

- Check if libgpiod is installed: `dpkg -l | grep libgpiod`
- Uninstall libgpiod if installed: `sudo apt-get remove libgpiod-dev libgpiod2`
- `git clone https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git --branch v2.1.3`
- `cd libgpiod`
- `sudo apt install autotools-dev autoconf-archive libtool pkg-config`
- `sudo ./autogen.sh --enable-tools=yes`
- `sudo make`
- `sudo make install`

NOTE: While installing the libgpiod, ensure to uninstall/clear any previously installed library version

Build and execute “ses-example-app”:

Build steps: Ensure DSPIDEV=ON to build with SPIDEV support

- `cd ses-example-app/project/cmake/`
- `cmake -DSPIDEV=ON -S . -B build`
- `cmake --build build`

Execute: In this example, /dev/spidev0.0 is used as the SPI interface, and GPIO pin 5 is used for the interrupt which can be configured in -p and -g parameters.

```
ses-example-app -f <xml-filepath> -g <path of the gpio chip and pin number> -p <interface-selection> [-m <yang-modules-folder>] [-s <0,1,2,4> SPI-mode] -i <development, production> [-l SPI interface info]
```

Example: `sudo ./ses-example-app -f example-eval-6310.xml -g /dev/gpiochip0:5 -p /dev/spidev0.0 -m sysrepo-glue-srv/modules/ -s 1 -i production -l`

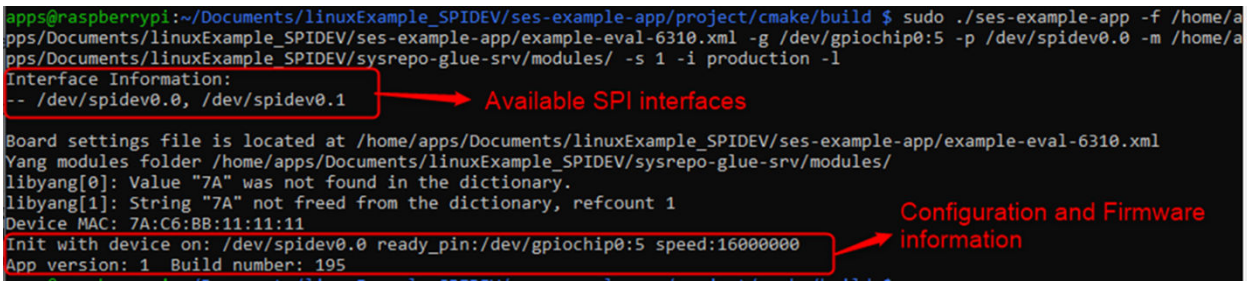

Figure 11. Example output when running ses-example-app

Build and execute “ses-tsn-configuration-app”:

Ensure that all dependencies mentioned in the Ses-tsn-configuration-app Build and Execution section are resolved.

LINUX USER SPACE DRIVER

Build steps: Ensure DSPIDEV=ON to build with SPIDEV support

- `cd ses-tsn-configuration-app/project/cmake/`
- `cmake -DSPIDEV=ON -S . -B build`
- `cmake --build build`

Execute: In this example, /dev/spidev0.0 is used as the SPI interface, and GPIO pin 5 is used for the interrupt which can be configured in -p and -g parameters.

```
ses-tsn-configuration-app -f <xml-filepath> -o <output-filepath> -g< path of the gpio chip and pin number> -p <interface-selection> [-s <0,1,2,4> SPI-mode] -i <development, production> [-l SPI interface info]
```

Example: `sudo ./ses-tsn-configuration-app -f eval-adin6310.xml -o /home/apps -g /dev/gpiochip0:5 -p /dev/spidev0.0 -s 1 -i production -l`

YOCTO BUILD

The Yocto project is an open-source project that provides tools and processes for developing custom Linux-based systems. It enables the creation of tailored Linux distributions by selecting and configuring necessary components, thereby minimizing footprint and maximizing performance. The yocto recipes for ses-example-app and ses-tsn-configuration-app are provided in “yocto-recipes” in the example folder.

This section describes how to build a yocto image for Raspberry pi 4 hardware which includes the ses-example-app and its dependencies. The yocto image can be built on Ubuntu using the following steps:

Building Yocto image with Ses-example-app:

1. Setting up build system for Debian:
   a. `sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat cpio python3 python3-pip python3-pexpect xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev pylint3 xterm lz4`
2. Set up a project and source folder:
   a. `mkdir yocto`
   b. `cd yocto`
   c. `mkdir sources`
   d. `cd sources`
3. **Getting meta layers:** clone poky meta layer for basic yocto functionality
   a. `git clone https://git.yoctoproject.org/poky -b nanbield`
4. Clone the meta layers for Raspberry Pi and its dependencies:
   a. `git clone https://git.yoctoproject.org/meta-raspberrypi -b nanbield`
   b. `git clone https://git.openembedded.org/meta-openembedded -b nanbield`
5. **Initialize the build environment:** Return to yocto project folder and source/initialize the build environment, which generates a build folder inside the yocto project folder.
   a. `source sources/poky/oe-init-build-env`
6. **Configure BitBake layer:** Provide source code repos for ses-sample-application in recipes-sesdriver/sesdriver/files.
   a. Add meta layers to bblayers.conf
      - `../sources/meta-raspberrypi \`
      - `../sources/meta-openembedded/meta-oe \`
      - `../sources/meta-openembedded/meta-multimedia \`
      - `../sources/meta-openembedded/meta-networking \`
      - `../sources/meta-openembedded/meta-python \`

LINUX USER SPACE DRIVER

- `../sources/ses-example-app`

b. Select the target machine by adding the following line to local.conf
- `MACHINE ?= "raspberrypi4"`

c. Set systemd as init manager in local.conf
- `INIT_MANAGER = "systemd"`

d. To add a root password to the image provide the following lines:
- `INHERIT += "extrausers"`
- `EXTRA_USERS_PARAMS = "usermod -p '$(openssl passwd root)' root;"`
- `LICENSE_FLAGS_ACCEPTED = "synaptics-killswitch ftdi"`

e. To build the recipe with your selected image add the following line to local.conf:, or provide an additional .bbappend layer for the desired image including this recipe.
- `IMAGE_INSTALL:append = " sesexampleapp"`

7. **Provide source files:** Add the necessary source files from the src folder of driver library in the files folder of ses-example-app/recipes-se-sexampleapp/sesexampleapp/files:

a. ses-example-app
b. ses-linux-port-srv
c. ses-proxy-srv
d. ses-route-srv
e. ses-tsn-api-srv
f. smp-stk
g. sysrepo-glue-srv
h. tsn-model-srv

8. **Building the yocto image:**

a. To start building a yocto image go to the yocto project folder and source/initialize the build environment and build the desired image:
- `source sources/poky/oe-init-build-env`
- `bitbake core-image-base`

b. To only build sesexample app:
- `source sources/poky/oe-init-build-env`
- `bitbake sesexampleapp`

c. Successful build of the Yocto image is shown in

Figure 12. Terminal Window Yocto Build successful output

LINUX USER SPACE DRIVER

The final image can be found under:

- tmp/deploy/images/repberrypi4/core-image-base-raspberrypi4.wic.bz2

To flash the image onto the Raspberry Pi, first decompress the image file, then proceed with the flashing process. Use the following credentials to login:

- Username: root
- Password : root

ses-example-app is placed in /usr/bin/ yang modules are placed in /usr/share/yang/modules/ The switch board settings files are placed in /usr/share/ses-x310/.

1. **Run Example Application:** Once the build is successful, use the following command to run the example application:

```
ses-example-app -f <xml-filepath> -p <interface-selection> [-m <yang-modules-folder>] [-s <0,1,2,4>
SPI-mode] -i <production> [-l SPI interface info]
```

Example for Ethernet interface:

- `ses-example-app -f /usr/share/ses-x310/eval-adin6310.xml -p d8:3a:dd:13:ad:db -m /usr/share/yang/modules/ -s 0 -i production`

Example for SPI interface:

- `ses-example-app -f /usr/share/ses-x310/eval-adin6310.xml -p 0 -m /usr/share/yang/modules/ -s 1 -i production -l`

Building Yocto Image with ses-tsn-configuration-app:

To build the Yocto image with the ses-tsn-configuration-app, follow the previous steps up to "**Initialize the build environment**" and then proceed with the additional steps below:

1. **Configure BitBake layer:** Provide source code repos for ses-sample-application in recipes-sesdriver/sesdriver/files/.
   - **a.** Add meta layers to bblayers.conf
     - `../sources/meta-raspberrypi \`
     - `../sources/meta-openembedded/meta-oe \`
     - `../sources/meta-openembedded/meta-multimedia \`
     - `../sources/meta-openembedded/meta-networking \`
     - `../sources/meta-openembedded/meta-python \`
     - `../sources/ ses-tsn-configuration-app`
   - **b.** Select the target machine by adding the following line to local.conf
     - MACHINE ?= "raspberrypi4"
   - **c.** Set systemd as init manager in local.conf
     - INIT_MANAGER = "systemd"
   - **d.** To add a root password to the image provide the following lines:
     - INHERIT += "extrausers"
     - EXTRA_USERS_PARAMS = "usermod -p '$(openssl passwd root)' root;"
     - LICENSE_FLAGS_ACCEPTED = "synaptics-killswitch ftdi"
   - **e.** To build the recipe with your selected image add the following line to local.conf:, or provide an additional .bbappend layer for the desired image including this recipe.
     - IMAGE_INSTALL:append = " libpcap libft4222 sestsnconfigurationapp sysrepo netopeer2-server openssh openssl"
   - **f.** To provide libraries and header files to build the application on the target image append the following to local.conf:
     - IMAGE_INSTALL:append = " libpcap-dev libft4222-dev sestsnconfigurationapp sysrepo-dev netopeer2-server openssh openssl cmake packagegroup-core-buildessential"

LINUX USER SPACE DRIVER

2. **Provide source files:** Add the necessary source files from the src folder of library in the files folder of ses-tsn-configuration-app/recipes-sestsnconfigurationapp/sestsnconfigurationapp/files:
   - linux-file-system-srv
   - linux-hwpl-srv
   - linux-ospl-srv
   - ses-tsn-configuration-app
   - ses-linux-port-srv
   - ses-proxy-srv
   - ses-route-srv
   - ses-tsn-api-srv
   - smp-stk
   - sysrepo-glue-srv
   - tsn-model-srv
3. **Building the yocto image:**
   a. To start building a yocto image, go to the yocto project folder and source/initialize the build environment and build the desired image:
      - source sources/poky/oe-init-build-env
      - bitbake core-image-base
   b. To only build sesexample app:
      - source sources/poky/oe-init-build-env
      - bitbake sestsnconfigurationapp
   c. The final image can be found under:
      - tmp/deploy/images/repberrypi4/core-image-base-raspberrypi4.wic.bz2
   d. To flash the image onto the Raspberry Pi, first decompress the image file, then proceed with the flashing process.
   e. Use the following credentials to login
      - Username: root
      - Password : root
   f. ses-tsn-configuration-app is placed in /usr/bin/ yang modules are placed in /usr/share/yang/modules/ Switch board settings files are placed in /usr/share/ses-x310/
4. **Run Example Application:** Once the build is successful, use the following command to run the example application:

```
ses-tsn-configuration-app -f <xml-filepath> -o <output-filepath> -p <interface-selection> [-s
<0,1,2,4> SPI-mode] -i < production> [-l SPI interface info]
```

   a. Example for ethernet interface:
      - ses-tsn-configuration-app -f /usr/share/ses-x310/eval-adin6310.xml -o test -p d8:3a:dd:13:ad:db -s 0 -i production
   b. Example for SPI interface:
      - ses-tsn-configuration-app -f /usr/share/ses-x310/eval-adin6310.xml -o test -p 0 -s 1 -i production -l
5. To configure the switch using NETCONF server, refer the “Configuration using NETCONF” from previous section. The netopeer server is already installed with yocto build.

DRIVER EXAMPLES

The following are some examples routines for common features when using the switch. Note that *all* code snippets in this document are under the following license:

MULTI-SES SUPPORT

The driver supports use cases where a single host processor can configure multiple Switch devices. The porting layer can support multiple interfaces and the data structures describing the interface and holding interface handle used by the hardware drivers.

The routing layer contains the data structure associating a Switch device index to an interface index. This inserts an operation between the SMP and Port that is responsible for creating message header for routing to the proper device.

In the select layer, a device index parameter identifies which device to interact with. The APIs for multi-SES configuration are included throughout the header files with the prefix SES_MX and an additional sesID_t parameter.

The SES_MX APIs can be called with a sesID_t of 0 if only one SES is used, alternatively, simply use the corresponding single API call. Throughout this user guide, reference will be made to the single APIs unless it's a specific use case where multi-ses configuration is used.

Further detail on the Multi-SES, review the SES_Interface_Management.h header file.

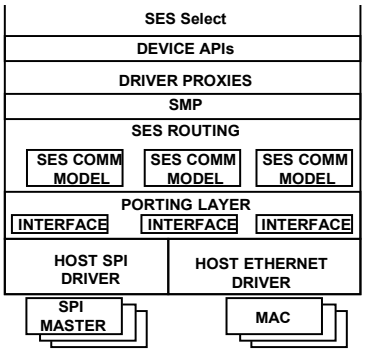

Figure 13. Multi-SES capability.

The initialization sequence is described briefly here and in more detail in the following examples

- Control and structure initialization (SES_Init)
- Initialize the interfaces (SES_AddHwInterface)
- Add devices to the routing layer (SES_AddDevice)
  - Discover and configure locally attached SES devices
    - Configuration Assign MAC and define ports
  - Auto-update firmware if mismatch is detected
- Port Configuration (SES_InitializePorts)
- Once all devices are configured use modified APIs (SES_MX_..) with device index to perform configuration

The application is responsible for configuring static table of local devices prior to calling SES_AddDevice for chained chips. In a star topology, create a static table entry, enable a port, add the next device, change the static table entry and repeat for devices on additional ports.

DRIVER EXAMPLES

SWITCH & PORT INITIALIZATION

The Switch is a managed switch, therefore requires some interaction over the host interface before any traffic can cross the switch. When the switch powers up, it samples its host strapping pins to identify where the host interface resides and what type of host interface (SPI or Ethernet). If it is a new device from the factory, it will be unprogrammed. The switch will send image requests to the host to initiate the firmware update process. If it has been previously programmed, then it waits for the host to communicate over the defined host port and tell the switch how to configure the remaining ports. The initialization sequence of the Switch involves four API calls, the sequence is shown in the reference example below.

SES_InitializePorts() is the API call used to initialize the ports on the switch. This API is called once during the startup sequence. Applications and devices can maintain the port configuration settings for all ports persistently and apply such settings during the startup sequence. Without calling this function none of the ports will work except the host port in Ethernet interface mode. The API internally configures and initializes all the ports and corresponding PHYs. If the PHYs are not "Known PHYs" only ports are configured and initialized. The corresponding configuration and initialization of unknown or unmanaged PHYs must be handled by the application directly or through the switch via the generic MDIO read/write APIs. Some of the configuration settings are fixed based on the target hardware configuration and some of the settings can be changed. For example, PHY type is fixed based on the hardware configuration. Speed and Duplex can be changed based on the preferred settings.

The following APIs in this section can be viewed in the SES_switch.h header file in the TSN driver library package.

Initialize Using Ethernet Host Interface

The following example shows how to initialize the ADIN6310 (6-port device) using an Ethernet host, with all ports configured for Gigabit speeds, this configuration matches the typical configuration for the EVAL-ADIN6310EBZ evaluation board.

Firstly, the SES_Init() API must be called, followed by a call to the SES_AddHwInterface() with the MAC address for the host, pointers to the driver interface configuration and the interface ID. Directly after this, the SES_AddDevice() is called with the Interface ID, a user defined MAC address for the Switches Packet assist engine (primary MAC address) and the device ID(in event of the host interfacing to more than one Switch device).

```
#define SES_PORT_COUNT   (6)
int32_t rv = 0;

/*
Port configuration for ADIN6310 {Port enable, MII, RxDelay, txDelay, clk selection, link polarity, PHY
type, {autoneg, pullupctrl, phyAddr, Speed, Duplex, Crossover}}
*/
const SES_portInit_t portConfiguration[6] = {
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1300, {true, 0, 0, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 1, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 2, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 4, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 8, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 9, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}}
};

/*Example of primary MAC address assigned to the switch Packet Assist engine*/
uint8_t sesPrimaryMac[6] = { 0x7a, 0xc6, 0xbb, 0x11, 0x11, 0x11 };
/*Host MAC address connected to the switch*/
uint8_t hostMac[6] = { 0x00, 0x00, 0x00, 0x99, 0x99, 0x99 };
uint32_t interfaceId = NULL;
```

DRIVER EXAMPLES

```
sesID_t deviceId = NULL;

SES_driverFunctions_t driverFunctions = {
    .init_p = SES_PORT_ETH_Init,
    .release_p = SES_PORT_ETH_Release,
    .sendMessage_p = SES_PORT_ETH_SendMessage,
    .updateFilter_p = SES_PORT_ETH_UpdateFilter
};
rv = SES_Init();
if (rv == SES_PORT_OK) {
    rv = SES_AddHwInterface(hostMac, &driverFunctions, &interfaceId);
    if (rv == SES_PORT_OK) {
        rv = SES_AddDevice(interfaceId, sesPrimaryMac, &deviceId);
        if (rv == SES_PORT_OK) {
            rv = SES_InitializePorts(SES_PORT_COUNT, portConfiguration);
        }
    }
}
```

Initialize Using Standard SPI Host Interface

The following example shows how to initialize one switch using a host connected over Single SPI interface. Note the SPI hardware used here is an FTDI USB to SPI interface (UMFT4222EV) which is capable of single SPI and Quad SPI.

The SES_PORT_SPI_HandleError defines the recommended recovery action for each supported SPI error condition. A detailed sample implementation is provided in the SES_PORT_SPI_interface.c source file. This implementation demonstrates how different SPI error scenarios can be handled using the recommended recovery actions.

```
#define SES_PORT_COUNT   (6)
int32_t rv = 0;
/*
Port configuration for ADIN6310 {Port enable, MII, RxDelay, txDelay, clk selection, link polarity, PHY
type, {autoneg, pullupctrl, phyAddr, Speed, Duplex, Crossover}}
*/
const SES_portInit_t portConfiguration[6] = {
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1300, {true, 0, 0, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 1, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 2, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 4, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 8, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 9, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}}
};

uint8_t sesPrimaryMac[6] = { 0x7a, 0xc6, 0xbb, 0x11, 0x11, 0x11 }; // Primary MAC address of SES -
example
uint32_t interfaceId = NULL;
sesID_t deviceId = NULL;
```

DRIVER EXAMPLES

```
SES_driverFunctions_t driverFunctions = {
.init_p = SES_PORT_SPI_Init,
.release_p = SES_PORT_SPI_Release,
.sendMessage_p = SES_PORT_SPI_SendMessage,
.handleError_p = SES_PORT_SPI_HandleError
};

SES_PORT_ft4222InitParams_t initParameter = {
.baseLocationId = SES_PORT_SPI_SINGLE_FT4222,
.chipSelect = 0,
.spiMode = SES_PORT_SPI_singleMode
};

rv = SES_Init();
if (rv == SES_PORT_OK) {
    rv = SES_AddHwInterface(&initParameter, &driverFunctions, &interfaceId);
    if (rv == SES_PORT_OK) {
        rv = SES_AddDevice(interfaceId, sesPrimaryMac, &deviceId);
        if (rv == SES_PORT_OK) {
            rv = SES_InitializePorts(SES_PORT_COUNT, portConfiguration);
        }
    }
}
```

Initialize Using QSPI Host Interface

The following example shows how to initialize one switch using a host connected over Quad SPI interface. Note the SPI hardware used here is an FTDI USB to SPI interface (UMFT4222EV) which is capable of single SPI and Quad SPI.

```
#define SES_PORT_COUNT   (6)
int32_t rv = 0;
/*
Port configuration for ADIN6310 {Port enable, MII, RxDelay, txDelay, clk selection, link polarity, PHY
type, {autoneg, pullupctrl, phyAddr, Speed, Duplex, Crossover}}
*/
const SES_portInit_t portConfiguration[6] = {
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1300, {true, 0, 0, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 1, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 2, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 4, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 8, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 9, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}}
};

uint8_t sesPrimaryMac[6] = { 0x7a, 0xc6, 0xbb, 0x11, 0x11, 0x11 }; // Primary MAC address of SES -
example
uint32_t interfaceId = NULL;
sesID_t deviceId = NULL;
```

DRIVER EXAMPLES

```
SES_driverFunctions_t driverFunctions = {
.init_p = SES_PORT_SPI_Init,
.release_p = SES_PORT_SPI_Release,
.sendMessage_p = SES_PORT_SPI_SendMessage,
.handleError_p = SES_PORT_SPI_HandleError
};

SES_PORT_ft4222InitParams_t initParameter = {
.baseLocationId = SES_PORT_SPI_QUAD_FT4222,
.chipSelect = 0,
.spiMode = SES_PORT_SPI_quadMode
};

rv = SES_Init();
if (rv == SES_PORT_OK) {
    rv = SES_AddHwInterface((&)initParameter, (&)driverFunctions, (&)interfaceId);
    if (rv == SES_PORT_OK) {
        rv = SES_AddDevice(interfaceId, sesPrimaryMac, (&)deviceId);
        if (rv == SES_PORT_OK) {
            rv = SES_InitializePorts(SES_PORT_COUNT, portConfiguration);
        }

    }
}
```

Initialize Multiple Switches in a Chain, First Device SPI Host Interface, Second Over Ethernet

The following example shows how to initialize two switches, where the host is SPI interface to the first switch and the second switch is daisy-chained from one of the ports of the first switch.

```
#define SES_PORT_COUNT   (6)
int32_t rv = 0;

const SES_portInit_t portConfiguration[] = {
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1300, {true, 0, 0, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 1, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 2, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 4, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 8, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 9, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}}
    };

uint8_t device_a_mac[6] = { 0x7A, 0xC6, 0xBB, 0x11, 0x11, 0x11 };
uint8_t device_b_mac[6] = { 0x7A, 0xC6, 0xBB, 0x22, 0x22, 0x22 };
int host_interfaceId = NULL;
sesID_t device_a_id = NULL;
sesID_t device_b_id = NULL;
```

DRIVER EXAMPLES

```
SES_driverFunctions_t driverFunctions = {
    .init_p = SES_PORT_SPI_Init,
    .release_p = SES_PORT_SPI_Release,
    .sendMessage_p = SES_PORT_SPI_SendMessage    };

SES_PORT_ft4222InitParams_t device_a_param = {
    .baseLocationId = SES_PORT_SPI_SINGLE_FT4222,
    .chipSelect = 0,
    .spiMode = SES_PORT_SPI_singleMode
};

rv = SES_Init();
if (rv == SES_PORT_OK) {
    rv = SES_AddHwInterface(&device_a_param, &driverFunctions, &host_interfaceId);
    if (rv == SES_PORT_OK) {
        rv = SES_AddDevice(host_interfaceId, device_a_mac, &device_a_id);
        if (rv == SES_PORT_OK) {
            rv = SES_MX_InitializePorts(device_a_id, SES_PORT_COUNT, portConfiguration);
        }
    }
}
Sleep(5000); // Wait time to bring link up between Switch 1 and Switch 2 boards
// Switch 2 - Ethernet Host interface
if (rv == SES_PORT_OK) {
    rv = SES_AddDevice(host_interfaceId, device_b_mac, &device_b_id);
    if (rv == SES_PORT_OK) {
        rv = SES_MX_InitializePorts(device_b_id, SES_PORT_COUNT, portConfiguration);
    }
}
```

Initialize Multiple Switches in a Star Configuration

The following example shows how to initialize multiple switches from one host, where there are multiple switches devices configured in a star topology. In this example, all devices are connected over Ethernet interface.

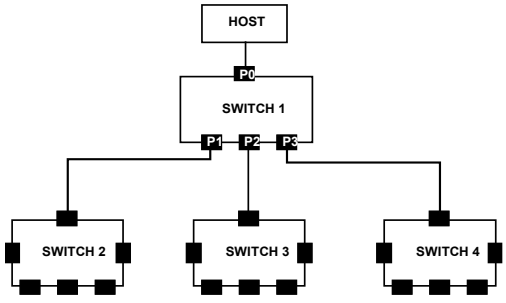

Figure 14. Multi-SES Star configuration

DRIVER EXAMPLES

In a star topology, create a static table entry, enable a port, add the next device, change the static table entry and repeat for devices on additional ports as shown in the example below.

```
void SES_Init_Star_Config_Test(void) {

    uint8_t macAddr_p[6] = { 0x7A, 0xC6, 0xBB, 0xFF, 0xFE, 0x00 }; // Default MAC address for SES
    int16_t vlanId = 0x0FFF;

    uint8_t pc_addr[6] = { 0xAC, 0x91, 0xA1, 0x91, 0xAC, 0x97 }; // host mac address
    uint8_t device_a_mac[6] = { 0x7A, 0xC6, 0xBB, 0x11, 0x11, 0x11 };
    uint8_t device_b_mac[6] = { 0x7A, 0xC6, 0xBB, 0x22, 0x22, 0x22 };
    uint8_t device_c_mac[6] = { 0x7A, 0xC6, 0xBB, 0x23, 0x23, 0x23 };

    int host_interfaceId = NULL;
    sesID_t device_a_id = NULL;
    sesID_t device_b_id = NULL;
    sesID_t device_c_id = NULL;


    const SES_portInit_t portConfiguration[6] = {
        { 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1300, {true, 0, 0, SES_phySpeed1000, SES_phyDu▶
plexModeFull, SES_autoMdix}},
        { 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 1, SES_phySpeed1000, SES_phyDu▶
plexModeFull, SES_autoMdix}},
        { 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 2, SES_phySpeed1000, SES_phyDu▶
plexModeFull, SES_autoMdix}},
        { 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 4, SES_phySpeed1000, SES_phyDu▶
plexModeFull, SES_autoMdix}},
        { 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 8, SES_phySpeed1000, SES_phyDu▶
plexModeFull, SES_autoMdix}},
        { 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 9, SES_phySpeed1000, SES_phyDu▶
plexModeFull, SES_autoMdix}}
    };

    SES_driverFunctions_t driverFunctions = {
        .init_p = SES_PORT_ETH_Init,
        .release_p = SES_PORT_ETH_Release,
        .sendMessage_p = SES_PORT_ETH_SendMessage,
        .updateFilter_p = SES_PORT_ETH_UpdateFilter
    };

         // Configure, Switch 1
    if (SES_PORT_OK == SES_Init()) {
        rv = SES_AddHwInterface(pc_addr, &driverFunctions, &host_interfaceId);
        printf("SES_AddHwInterface - %d\n", rv);
        if (SES_PORT_OK == SES_AddDevice(host_interfaceId, device_a_mac, &device_a_id)) {
            printf("SES_AddDevice Success \n");
            if (SES_PORT_OK == SES_MX_InitializePorts(device_a_id, SES_PORT_COUNT, portConfiguration))
{
                printf("SES_MX_InitializePorts\n");
            }
        }
    }

    // Install default MAC address in Static table to configure Port 1, Switch 2, then configure
```

DRIVER EXAMPLES

```
    if (SES_PORT_OK == SES_AddStaticTableEntry(&macAddr_p, vlanId, 0x02)) {
        Sleep(2000);
        if (SES_PORT_OK == SES_AddDevice(host_interfaceId, device_b_mac, &device_b_id)) {
            if (SES_PORT_OK == SES_MX_InitializePorts(device_b_id, SES_PORT_COUNT, portConfiguration))
{
                printf("Switch 2 configured Success!!\n");
                printf("Remove Static Entry :: %d\n", SES_RmStaticTableEntry(&macAddr_p, vlanId, 0));
            }
        }
    }

    // Install default MAC address in Static table to configure Port 2, Switch 3, then configure
    if (SES_PORT_OK == SES_AddStaticTableEntry(&macAddr_p, vlanId, 0x04)) {
        Sleep(2000);
        if (SES_PORT_OK == SES_AddDevice(host_interfaceId, device_c_mac, &device_c_id)) {
            if (SES_PORT_OK == SES_MX_InitializePorts(device_c_id, SES_PORT_COUNT, portConfiguration)){
                printf("Switch 3 configured Success!!\n");
                printf("Remove Static Entry :: %d\n", SES_RmStaticTableEntry(&macAddr_p, vlanId, 0));
            }
        }
    }
}
```

MAC Address Assignment

Irrespective of how the host interface is connected, the switch will be initialized with the primary MAC address being assigned to the Packet Assist Engine's MAC address and all the ports (Port 0 – 5, in case of a 6-port Switch) will be automatically assigned incremental MAC addresses. The primary MAC address gets used for communications to/from the connected Host Processor and as Source MAC Address for non-port-specific communications. Per-port addresses are used as Source MAC Address for link-local communications such as LLDP, MSTP and gPTP messaging. The frames used for these purposes are not forwarded by neighboring bridges.

For example:

- Let's say the primary MAC address (supplied as a parameter to SES_AddDevice() is 7A:C6:BB:11:11:11, this address gets assigned to the internal packet assist engine)
- Then, each port will be automatically assigned the following MAC addresses:
  - Port 0: 7A:C6:BB:11:11:12
  - Port 1: 7A:C6:BB:11:11:13
  - Port 2: 7A:C6:BB:11:11:14
  - Port 3: 7A:C6:BB:11:11:15
  - Port 4: 7A:C6:BB:11:11:16
  - Port 5: 7A:C6:BB:11:11:17
- The user can reconfigure the port MAC addresses if needed by calling SES_SetMacAddress() with the desired MAC address.

The MAC addresses shown above are examples addresses. When implementing the switch in a user application, the expectation is user would assign the switch with MAC addresses from their own pool of IEEE MAC addresses.

DRIVER EXAMPLES

Port Configuration Structure

The API call to initialize the SES is 'SES_InitializePorts()'. This API call will initialize SES's ports with the port configuration structure supplied as a parameter to it. The port configuration structure has the following members:

Table 1. Port Configuration Structure

Parameter Type	Parameter Name	Description
uint8_t	enablePort	Desired port and PHY state at initialization
SES_miiMode_t	mii	Inform the port about MII mode used. One of: ▶ SES_rmiiMode: RMII Fast Ethernet ▶ SES_rgmiiMode: RGMII Gigabit Ethernet ▶ SES_sgmiiMode: SGMII ▶ SES_sgmiiMode1000BaseSxLx: SGMII fiber optic Gigabit Ethernet ▶ SES_sgmiiMode1000BaseKx: SGMII backplane Gigabit Ethernet ▶ SES_sgmiiMode100BaseFx: SGMII Fast Ethernet over fiber optic cable
SES_miiClkConfig_t	miiClkDelays	Configure DLL delays for clock source. The miiClkDelays member parameters in the port configuration structure must match with that being set in the PHY. Its members are: ▶ uint8_t enableRxDelay: configure the RXC DLL delays 0 = disabled, 1 = enabled (For RGMII interface only) ▶ uint8_t enableTxDelay: configure the TXC DLL delays 0 = disabled, 1 = enabled (For RGMII interface only) ▶ uint8_t clkSelection: configure clock selection 0 = internal (SES 50MHz clock used and provided to Px_TXC pin for PHY), 1 = external 50MHz clock used for both SES and PHY (Applies to RMII interface only).
uint8_t	linkStatusPolarity	Informs the switch of the PHY link status polarity. The switch expects the link status input to be active low. This parameter defines whether the PHY link status is active high or active low and determines whether the switch needs to instruct the PHY to invert link status polarity via MDIO, ensuring that the PHY output matches the switch's expect behavior. Automatic polarity inversion is supported for ADI PHYs. The ADIN1300 defaults to an active high link status. Valid Values: ▶ 1: Active low ▶ 0: Active high Passing a value of 1 indicates that the PHY link status signal is already active low, which matches the switch expectation. No action is required from the switch. Passing a value of 0 indicates that the PHY link status signal is active high. In this case, the switch instructs the PHY to invert the link status polarity by configuring the appropriate PHY registers via MDIO, so that the signal presented to the switch is active low. **NOTE**: Automatic inversion of the PHY link status polarity by the switch is supported only for supported ADI PHYs. For non-ADI PHYs, the user must ensure that the PHY link status output matches the switch's active low requirement.
SES_phyType_t	phyType	Inform the switch of the PHY type used. One of: ▶ SES_phyADIN1100: ADI 10BASE-T1L PHY, use for both ADIN1100 and ADIN1101 ▶ SES_phyADIN1200: ADI 10/100 PHY ▶ SES_phyADIN1300: ADI 10/100/1000 PHY, use for both ADIN1300 and ADIN1320. EVAL-ADIN6310EBZ uses this ▶ SES_phyUnmanaged: User Managed PHY. The switch does not manage the PHY, instead, the user is fully responsible for configuring the PHY and ensuring that the negotiated link speed, duplex, and interface mode exactly match the switch port configuration. The host can configure through switch MDIO read/write APIs. Any mismatch between the PHY and switch settings may

DRIVER EXAMPLES

Table 1. Port Configuration Structure (Continued)

Parameter Type	Parameter Name	Description
SES_phyConfiguration_t	phyConfig	result in traffic loss or forwarding failures, as the switch does not perform any validation in unmanaged mode. **NOTE**: EVAL-ADIN3310EBZ and EVAL-ADIN6310EBZ uses ADIN1300 PHYs by default. The EVAL-ADIN6310T1LEBZ evaluation board uses a combination of ADIN1100 and ADIN1300 PHY. Data structure to set port attributes ▶ bool autoNegEnable : Used to enable/disable auto-negotiation in the external PHY (ADIN1200/ADIN1300) or in the switch for SGMII mode. ▶ In SGMII mode, some copper SFP modules may require auto-neg disabled ▶ Does not apply for ADIN1100 and ADIN1101PHY ▶ unit8_t phyPullupCtrl: Switch has internal pullup/down resistors on RXD pins which are provided to configure unique PHY addresses for ADIN1200/ADIN1300 PHYs on all ports ▶ 0 = internal, disabled after PHY config complete ▶ 1 = external ▶ 2 = internal and not disabled. ▶ For PHYs other than those listed, use external PHY address strapping ▶ uint16_t phyAddr: inform the Switch about the PHY address ▶ SES_phySpeed_t speed: desired speed of phy/port at initialization ▶ SES_phyDuplexMode_t duplexMode: desired duplex mode of phy/port at initialization ▶ SES_plyMdiMdixConfig_t crossover: desired crossover mode of phy/port at initialization

The following examples show useful port configurations for different possible switch use cases.

RGMII Mode

The below example sets up the switch ports in RGMII modes for all ports (this configuration matches how the switch is typically configured on the EVAL-ADIN6310EBZ Evaluation board). In this case, all 6 ports have ADIN1300 PHYs connected to the ports over RGMII interface. Note, when the host interface is SPI, then the configuration for port 0 is fully used. When the Host is configured as an Ethernet port, then the hardware strapping configurations configure the port and the parameters shown here are ignored.

```
//{Port enable, MII, RxDelay, txDelay, clk selection, link polarity, PHY type, {autoneg, pullupctrl,
phyAddr, Speed, Duplex, Crossover}}
const SES_portInit_t initializePorts_p[] = {
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1300, {true, 0, 0, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 1, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 2, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 4, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 8, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 9, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}}
};
```

Note that the *miiClkDelays* member parameters in the port configuration structure must agree with that being set in the PHY, RXC and TXC delays should only be introduced once in the path, on either side, typically they will be configured on the transmitting side. The ADIN1300

DRIVER EXAMPLES

default hardware strapping configures RGMII mode with both TXC and RXC delay enabled, therefore for these examples, we are relying on the PHY enabling the delay for both and no delays are necessary on the switch side.

SGMII Modes

ADIN6310 supports SGMII (copper/fiber/backplane) connectivity on four of the six ports (port indices 1, 2, 3 and 4). To initialize the required ports in SGMII mode, the “SES_miiMode_t” member of “SES_portInit_t” should be passed the required mode. When using SGMII interface, the switch port Link pin must be driven or pulled low to enable the port, as a low indicates link up on the port.

In the EVAL-ADIN6310EBZ, the SFP module LOS signal can be routed (hardware jumper on ports that support SGMII) to drive the switch port LINK input pin low to indicate a link UP on the port.

If using SGMII copper PHYs, review whether the SFP supports auto-negotiation capability. If auto-negotiation is not supported, ensure the parameter for autoNegEnable is configured as false. When connecting the switch SGMII interface to another SGMII MAC interface, autoNegEnable should be configured to false and both sides configured with matching speeds. Auto-negotiation is not supported when both sides are MAC interface.

The following example configures Port 1 as SGMII mode for a copper SFP module, Ports 2-3 as SGMII for fiber SFP module, Port 4 is configured for backplane, 1000BASE-KX and Port 5 is configured for RGMII with an ADIN1300 PHY connected.

For direct RMII/RGMII MAC-MAC connections, the speed should be configured correctly on both sides of the link and the parameter autoNegEnable does not matter.

```
//{Port enable, MII, RxDelay, txDelay, clk selection, link polarity, PHY type, {autoneg, pullupctrl,
phyAddr, Speed, Duplex, Crossover}}
const SES_portInit_t initializePorts_p[] = {
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1300, {true, 0, 0, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_sgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 1, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_sgmiiMode1000BaseSxLx, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 2, SES_phySpeed1000,
SES_phyDuplexModeFull, SES_autoMdix}},
{ 1, SES_sgmiiMode1000BaseSxLx, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 4, SES_phySpeed1000,
SES_phyDuplexModeFull, SES_autoMdix}},
{ 1, SES_sgmiiMode1000BaseKx, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 8, SES_phySpeed1000, SES_phy▶
DuplexModeFull, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 0, SES_phyADIN1300, {true, 0, 9, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}}
};
```

Field Switch Evaluation Board With ADIN1100 10BASE-T1L PHYs

The following example of port initialization structure may be used with the EVAL-ADIN6310T1LEBZ board. This version of hardware uses 2 x ADIN1300 PHYs on Ports 2 & 3 and 4 x ADIN1100 PHYs on Ports 0, 1, 4, 5. The ADIN1100 PHYs are connected over RGMII interface. For this version of hardware, the LINK_ST from each PHY includes an inverter in the path. Note the “linkStatusPolarity” being active low; this must be modified based on specific board design. Also, notice the ‘phyPullupCtrl’ field set to 1 – External pull-up, as for this evaluation board, external pull-ups are used for PHY addressing.

When configuring ADIN1100 PHYs, the auto-neg parameter does not make any changes to the PHY auto-negotiation capability.

```
//{Port enable, MII, RxDelay, txDelay, clk selection, link polarity, PHY type, {autoneg, pullupctrl,
phyAddr, Speed, Duplex, Crossover}}
const SES_portInit_t initializePorts_p[] = {
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1100, {true, 1, 0, SES_phySpeed10, SES_phyDuplexMode▶
```

DRIVER EXAMPLES

```
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1100, {true, 1, 1, SES_phySpeed10, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1300, {true, 1, 6, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1300, {true, 1, 5, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1100, {true, 1, 2, SES_phySpeed10, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1100, {true, 1, 3, SES_phySpeed10, SES_phyDuplexMode▶
Full, SES_autoMdix}}
};
```

ADIN3310 – 3-port Configuration

```
//{Port enable, MII, RxDelay, txDelay, clk selection, link polarity, PHY type, {autoneg, pullupctrl,
phyAddr, Speed, Duplex, Crossover}}
const SES_portInit_t initializePorts_p[] = {
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1300, {true, 0, 0, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1300, {true, 0, 1, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyADIN1300, {true, 0, 2, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}}
};
```

SES_phyUnmanaged

To use a PHY other than ADI PHYs (ADIN1100, ADIN1200, or ADIN1300), use the “SES_phyUnmanaged” parameter instead of “SES_phyADIN1300”.

Note, the PHY addressing may now be different as another PHY likely has different PHY Address strapping pins and the internal switch pull resistors on the RXD pins may not suit. Review the PHY address strapping relevant to the PHY used. The PHY should ensure it drives the switch port Px_LINK input with an active low link indictor. When using non-ADI PHYs, the Switch does not manage the PHYs directly, therefore the host needs to manage/configure the PHYs as needed, see Reading/Writing PHY Registers Over MDIO section.

This example shows the same PHY addressing as before, but relies on the user providing external strapping to configure the addresses accordingly.

```
//{Port enable, MII, RxDelay, txDelay, clk selection, link polarity, PHY type, {autoneg, pullupctrl,
phyAddr, Speed, Duplex, Crossover}}
const SES_portInit_t initializePorts_p[] = {
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyUnmanaged, {true, 1, 0, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyUnmanaged, {true, 1, 1, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyUnmanaged, {true, 1, 2, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyUnmanaged, {true, 1, 4, SES_phySpeed1000, SES_phyDuplexMode▶
```

DRIVER EXAMPLES

```
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyUnmanaged, {true, 1, 8, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}},
{ 1, SES_rgmiiMode, { 0, 0, 0 }, 1, SES_phyUnmanaged, {true, 1, 9, SES_phySpeed1000, SES_phyDuplexMode▶
Full, SES_autoMdix}}
};
```

DEFAULT FORWARDING CONFIGURATION

The default forwarding configuration of the switch for all traffic is to forward on all ports in cut-through mode operation. This behavior applies for unknown traffic, for which no matching entry exists in the lookup tables.

The SES_GetUnicastMissReturn() and SES_GetMulticastMissReturn() APIs allow the user to query the current forwarding. The default behavior can be modified using the SES_SetUnicastMissReturn() and SES_SetMulticastMissReturn() APIs.

```
/* Example of getting the default Miss return for unicast on Port 0 */
int32_t sesGetUnicastMissReturn_Example(void) {
    int32_t rv = 0;
    SES_mac_t mac = SES_macPort0;
        uint16_t portMap;
        bool cutThrough;
        uint8_t lookupDone;
        uint8_t txFilter;
        bool txFilterValid;
        uint8_t rxFilter;
        bool rxFilterValid;
    rv = SES_GetUnicastMissReturn(mac, &portMap, &cutThrough,
            &lookupDone, &txFilter, &txFilterValid, &rxFilter, &rxFilterValid);
    return rv;
}
```

```
/*
* Example of configuring default Unicast Miss Return for incoming traffic
* on Port 0 to egress only on Port 1 and for switching behaviour to be Store & Forward
*/
int32_t sesSetUnicastMissReturn_Example(void) {
    int32_t rv = 0;
    SES_mac_t mac = SES_macPort0;
        /* Forward only to Port 1 */
        uint16_t portMap = 0x2;

        /* Enable Store & Forward mode */
        bool cutThrough = false;

        /* No further lookup required */
        uint8_t lookupDone = 0x11;
        uint8_t txFilter = 0;
        bool txFilterValid = false;
        uint8_t rxFilter = 0;
        bool rxFilterValid = false;
        rv = SES_SetUnicastMissReturn(mac, portMap, cutThrough,
            lookupDone, txFilter, txFilterValid, rxFilter, rxFilterValid);
        return rv;
}
```

DRIVER EXAMPLES

CUT THROUGH VS STORE AND FORWARD OPERATION

The SES_SetStoreAndForwardMask() API provides user ability to configure the switch for store and forward mode, the SES_GetStoreAndForwardMask() allows user readback the current configuration. The default operation is Cut through operation for all queues on all ports. The example below configures Port 0 with store and forward operation on all queues.

```
/*Example of configuring Store and Forward operation on Port 0 on all queues */
int32_t ses_storeForwardExample() {

    int32_t rv = SES_OK;
    SES_mac_t port = SES_macPort0;

    /* set store and forward mode on allqueues */
    SES_queueFrameSwitching_t storeandforward = { 0,0,0,0,0,0,0,0 };
    rv = SES_SetStoreAndForwardMask(port, &storeandforward);

    return rv;
}
```

LOOKUP TABLES

The Forwarding table consists of a total of 2048 entries, the first 128 entries are allocated to Static entries and remaining 1918 entries are available for dynamic/learned entries. The dynamic entries are placed higher up in the table compared to the static entries. If the switch has learned a MAC address based on the traffic crossing it and the host later installs a static entry for that same MAC address, the dynamic table should be flushed, otherwise the existing dynamic entry will be hit first and the new static entry will get ignored.

Adding a Static Table Entry

The Static table APIs allow user to add/read/remove static entries to/from the lookup table. Adding a static table entry makes the switch aware of the ports on which a specific MAC address and specific VLANs are possibly reachable, when an frame ingresses on a port with MAC and VLAN information that match an entry in the table, then the forwarding information returned is used to decide which port(s) the frame is egressed on. Static entries are inserted and removed by the host. Dynamic entries are learned by the switch based on the traffic crossing the switch. Valid entries in the dynamic cable can be read back using SES_ReadDynamicTable() and entries can be flushed from the table using SES_FlushDynamicTable() or SES_FlushDynamicTableEx() see Reading the Dynamic Table (Learned Entries) section.

The switch default miss behavior (no entry in shared static or dynamic table) is for frames to be broadcasted on all ports. This behavior can be changed through API writes as discussed in Default Forwarding Configuration.

To enter a static entry, there is a basic API, SES_AddStaticTableEntry() and a more detailed API SES_AddStaticTableEntryEx(). The basic API places the entry at some free location in the table. There is no ability to read this type of entry back as there is no index returned with location of where it was stored. When installing an entry which has a VLAN tag, the VLAN table must also be configured for the VLAN ID and ports of interest to allow traffic forward based on the static entry.

Examples of inserting a basic static table entries as follows:

```
int32_t ses_addSimpleStaticEntry() {

    /* Simple example of adding a static entry */
    int32_t rv = SES_OK;
    uint8_t macAddr[6] = { 0x11, 0x22, 0x33, 0x44, 0x55, 0x66 };

    /* VLAN ID 0xF */
    int16_t vlanId = 0xF;
```

DRIVER EXAMPLES

```
    /* Egress on Port 4 */
    uint8_t portMap = 0x10;
    rv = SES_AddStaticTableEntry(&macAddr, vlanId, portMap);
    return rv;
}
```

The more detailed API, allows user greater configurability over the table entry and returns an index for location. This type of table entry can be read back using the SES_ReadStaticTableEntry() API, when in normal TSN mode of operation. When the device is configured for PRP or HSR mode of operation, it is not possible to read static entries with this API. The following example shows use of the more detailed static table entry.

```
/* Example of adding a detailed static entry */

int32_t rv = SES_OK;
/*low priority = 0, entry will be placed towards bottom of table*/
uint8_t lookupPriority = 0;
/*MAC address for entry to be installed */
uint8_t macAddr[6] = { 0x22, 0x33, 0x44, 0x55, 0x66, 0x77 };
/*Mask to be applied to the MAC address match information, 0-> don't care, 1->match */
uint8_t macMask[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
/*VLAN ID for entry, use 4095 for untagged*/
uint16_t vlanId = 0x0F;
/*mask for bits in vlanID to match, 0-> don't care, 1-> match bit */
uint16_t vlanMask = 0xFFF;
/*set to 0 to match on destination address, or 1 to match on source*/
uint8_t sourceEntry = 0;
/*override a blocked/learning state on a port*/
uint8_t override = 0;
/*set to 0 to use destination lookup result or 1 to use source lookup result for port map*/
uint8_t sourceOverride = 0;
/*Used in HSR/PRP, if set, treat frame as a supervisory frame in PRP or HSR*/
uint8_t hsrOrPrpSupervisory = 0;
/*indicate type of lookup, destination lookup with option to also perform source, extended or combina▶
tion */
SES_dynTblLookup_t lookupType = SES_dynamicTblLookupBasic;
/*set of ports matching frame should egress. Egress on Port 4 */
uint8_t portMap = 0x10;
/*forward frame to assist engine, not intended for normal operation*/
uint8_t sendToAE = 0;
/*used in PRP mode, if set, the frame is from a DANP*/
uint8_t danp = 0;
/*if set, treat frame as cut-through on receiving port and transmit port if all other conditions are
met. Store and Forward operation if set to 0. */
uint8_t cutThrough = 0;
uint8_t syntTimestamp = 0;
uint8_t localTimestamp = 0;
/*used as part of FRER. For normal operation pass dynamicTblNoSequenceOp*/
SES_dynTblSeqMgmt_t sequenceMgmt = dynamicTblNoSequenceOp;
int rxSequenceSet = 0;
/*transmit transform entry to associate with matching frame receiveFilter */
int transmitFilter = SES_IGNORE_FIELD;
/*receive filter to associate with matching frame*/
int receiveFilter = SES_IGNORE_FIELD;
/*pointer to location where static entry index is stored*/
```

DRIVER EXAMPLES

```
int32_t index;

rv = SES_AddStaticTableEntryEx(lookupPriority, &macAddr, &macMask, vlanId, vlanMask, sourceEntry, over▶
ride, sourceOverride, hsrOrPrpSupervisory, lookupType,
portMap, sendToAE, danp, cutThrough, syntTimestamp, localTimestamp, sequenceMgmt,
rxSequenceSet, transmitFilter, receiveFilter, &index);
```

Example of reading a static entry back from the device at index location 120 as follows:

```
/* Example of reading a static entry back from the device at index location 120 */
int32_t rv = SES_OK;
uint8_t macAddr[6];
uint8_t macMask[6];
uint16_t vlanId;
uint16_t vlanMask;
int sourceEntry;
int override;
int sourceOverride;
int hsrOrPrpSupervisory;
SES_dynTblLookup_t lookupType;
uint8_t portMap;
int sendToAE;
int danp;
int cutThrough;
int syntTimestamp;
int localTimestamp;
SES_dynTblSeqMgmt_t sequenceMgmt;
int rxSequenceSet;
int transmitFilter;
int receiveFilter;
int index = 120;

rv = SES_ReadStaticTableEntry(index, &macAddr, &macMask,
        &vlanId, &vlanMask, sourceEntry,
        override, sourceOverride, hsrOrPrpSupervisory,
        &lookupType, &portMap, sendToAE,
        danp, cutThrough, syntTimestamp,
        localTimestamp, &sequenceMgmt, &rxSequenceSet,
        &transmitFilter, &receiveFilter);


printf("Result %d, vlanID %d, msk 0x%04X, srcEntry %d, override %d, srcOverride %d, hsrOrPrpSuperviso▶
ry %d, lookupType %d, portMap %d, danp %d, cutThrough %d \r\n",
rv, vlanId, vlanMask, sourceEntry, override, sourceOverride, hsrOrPrpSupervisory, lookupType, portMap,
danp, cutThrough);

printf("mac %02X:%02X:%02X:%02X:%02X:%02X\r\n", macAddr[0], macAddr[1], macAddr[2], macAddr[3], mac▶
Addr[4], macAddr[5]);
printf("msk %02X:%02X:%02X:%02X:%02X:%02X\r\n", macMask[0], macMask[1], macMask[2], macMask[3], mac▶
Mask[4], macMask[5]);
```

When reading back a static table entry, the MAC address mask values returned by the switch may differ from the values originally configured by the user. This behavior is expected and results from how the switch internally handles mask values.

DRIVER EXAMPLES

Internally, when both the MAC address bit and the corresponding mask bit are set to 1, the entry is interpreted as a don't care condition. To ensure correct behavior, the switch adjust the mask values that are programmed to the switch registers. As a result, the mask returned when reading back a static entry reflects the adjusted mask, rather than the exact value provided by the user.

Example

If the user programs a static table entry with the following values:

- **MAC Address**: 10:11:12:13:14:15
- **MAC Mask** (Configured by User): FF:FF:FF:FF:FF:FF

When the entry is read back, the returned mask value may appear as:

- **MAC Mask** (Read back): EF:EE:ED:EC:EB:EA

ADDING A SOURCE LOOKUP ENTRY

The example demonstrates the use of an static table entry to perform a source look up. The SES_SetRxPortLookupMode API configures Port 1 to operate in source lookup mode. The installed static table entry is configured to match on source MAC address.

When a frame is received on Port 1, the switch performs a source MAC address lookup. If the source MAC matches the installed entry, the switch routes the frame to Port 2 according to the static table configuration.

```
int32_t sesSourceLookupEntryWithRxPortLookup_Example(void)
{
  int32_t rv = SES_PORT_ERROR;

  uint8_t macAddr[6] = { 0x00, 0x00, 0x00, 0x11, 0x11, 0x11 };
  int32_t index;

  rv = SES_SetRxPortLookupMode(
      SES_macPort1, // mac - port 1
      SES_RX_PORT_SRC_LOOKUP // lookupMode - source lookup mode
  );

  rv = SES_AddStaticTableEntryEx(
      1, // lookupPriority - high
      macAddr, // macAddr_p - MAC address
      NULL, // macMask_p - Exact match
      SES_DYNTBL_NO_VLAN, // vlanId - no VLAN
      0x0000, // vlanMask
      1, // sourceEntry - match on source address
      0, // override - do not override
      1, // sourceOverride - source lookup result for port mapping
      0, // hsrOrPrpSupervisory - not HSR/PRP supervisory frame
      SES_dynamicTblLookupSrc, // lookupType - destination lookup done. Go for source address lookup
      0x04, // portMap - forward to port 2
      0, // sendToAE - do not send to AE
      0, // danp - not from a DANP
      0, // cutThrough - not cut through
      0, // syntTimestamp - no synthetic timestamp
      0, // localTimestamp - no local timestamp
      dynamicTblNoSequenceOp, // sequenceMgmt - no operation
      0, // rxSequenceSet
      SES_DYNTBL_NO_ENTRY, // transmitFilter - no association
      SES_DYNTBL_NO_ENTRY, // receiveFilter - no association
      &index
```

DRIVER EXAMPLES

```
  );
  printf("Add Static Table Entry : %d, Index: %d\n", rv, index);

  return rv;
}
```

The example demonstrates a two-stage lookup, where a destination MAC lookup entry triggers a source MAC lookup.

- Entry 1 (Destination Entry) matches the frame's destination MAC and sets lookup type to SES_dynamicTblLookupSrc, which instructs the switch to perform a source lookup as the next step.
- Entry 2 (Source Entry) matches the frame's source MAC and is evaluated if the destination entry matches first.

This approach enables forwarding based on both destination and source MAC address, with optional control over whether the source entry overrides the portmap defined by the destination entry.

```
/**
* Example: Source lookup triggered by a destination lookup.
*
* Two static table entries are created:
* 1. Destination entry - matches destination MAC, sets lookupType to
* SES_dynamicTblLookupSrc to match into a source address lookup entry.
* 2. Source entry - matches source MAC of the incoming frame. Only
* evaluated if the destination entry matched first.
*/
int32_t sesSourceLookupEntry_Example(void)
{
  int32_t rv = SES_PORT_ERROR;

  /* --- Destination Lookup Entry --- */
  /* Matches destination MAC; matches next to a source lookup via SES_dynamicTblLookupSrc */
  uint8_t lookupPriority = 0;
  uint8_t macAddrDst[6] = { 0x00, 0x00, 0x00, 0x11, 0x11, 0x11 };
  uint8_t macMask[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
  int16_t vlanId = 0x0FFF;
  uint16_t vlanMask = 0; /* VLAN field wildcarded (ignored in lookup) */
  uint8_t sourceEntry = 0; /* 0 = destination entry */
  uint8_t override = 0;
  uint8_t sourceOverride = 0;
  uint8_t hsrOrPrpSupervisory = 0;
  SES_dynTblLookup_t lookupType = SES_dynamicTblLookupSrc; /* Next: do source lookup next */
  uint8_t portMap = 0x02; /* Forward to port 1 */
  uint8_t sendToAE = 0;
  uint8_t danp = 0;
  uint8_t cutThrough = 1;
  uint8_t syntTimestamp = 0;
  uint8_t localTimestamp = 0;
  SES_dynTblSeqMgmt_t sequenceMgmt = dynamicTblNoSequenceOp;
  int rxSequenceSet = 0;
  int transmitFilter = SES_IGNORE_FIELD;
  int receiveFilter = SES_IGNORE_FIELD;
  int32_t index_dst;

  rv = SES_AddStaticTableEntryEx(
    lookupPriority,
    macAddrDst,
```

DRIVER EXAMPLES

```
    macMask,
    vlanId,
    vlanMask,
    sourceEntry,
    override,
    sourceOverride,
    hsrOrPrpSupervisory,
    lookupType,
    portMap,
    sendToAE,
    danp,
    cutThrough,
    syntTimestamp,
    localTimestamp,
    sequenceMgmt,
    rxSequenceSet,
    transmitFilter,
    receiveFilter,
    &index_dst
  );
  if (rv < SES_PORT_OK)
    printf("SES_AddStaticTableEntryEx - Destination Lookup : %d\n", rv);

  /* --- Source Lookup Entry --- */
  /* Triggered by the destination entry above; matches source MAC */

  uint8_t macAddrSrc[6] = { 0x00, 0x00, 0x00, 0x22, 0x22, 0x22 };
  sourceEntry = 1; /* 1 = source entry */
  // sourceOverride = 1; /* Optional: Set to 1 to use the portmap defined in the source lookup */
  lookupType = SES_dynamicTblLookupBasic; /* No further matching */
  int32_t index_src;

  rv = SES_AddStaticTableEntryEx(
    lookupPriority,
    macAddrSrc,
    macMask,
    vlanId,
    vlanMask,
    sourceEntry,
    override,
    sourceOverride,
    hsrOrPrpSupervisory,
    lookupType,
    portMap,
    sendToAE,
    danp,
    cutThrough,
    syntTimestamp,
    localTimestamp,
    sequenceMgmt,
    rxSequenceSet,
    transmitFilter,
    receiveFilter,
    &index_src
  );
  if (rv < SES_PORT_OK)
```

DRIVER EXAMPLES

```
    printf("SES_AddStaticTableEntryEx - Source Lookup : %d\n", rv);

  return rv;
}
```

ADDING AN EXTENDED TABLE ENTRY

The extended lookup table supports up to 256 entries. The purpose of this lookup table is to allow deeper inspection into the frame to identify different types of traffic to perform filtering and transforms.

The decision to perform an extended lookup is indicated by SES_dynTblLookup_t parameter in a normal static entry or alternatively can apply to all frames per port using SES_SetRxPortLookupMode().

The helper routines SES_AddExtendedStaticRoute_Simple (), SES_AddExtendedStaticRoute_IPv4TcpUdp (), and SES_AddExtendedStatic-Route_IPv6TcpUdp () can be used to add extended table entries. These three routines simplify the process of installing an extended table entry or an IPv4/IPV6 entry.

Alternatively, for other Ethertypes of interest, the SES_SetExtLookupRule() and SES_InstallExtLookupEntry () can be used to add an extended lookup rule and install an extended table entry. If user tries to install a rule that already exists, the API will return that the rule already exists. Firmware takes care of installing rules for IPv4 & IPv6, therefore user doesn't need to install these rules and can directly use the helper API calls.

The following example shows adding an extended table entry using SES_AddExtendedStaticRoute_Simple () API. The API matches the ether type and data of the frame.

```
/* Example of adding an extended table entry, note that to perform extended table lookup, user *must
also either install a static table entry passing  passing SES_dynamicTblLookupExt for the *lookup mode
or alternatively enable extended table lookup for the port using *SES_SetRxPortLookupMode()
*/

/* Egress on Port 4 */
int8_t portMap = 0x10;
int8_t cutThrough = 0;
/* Ethertype to match in frame */
uint16_t ethertype = 0xA000;
/* data to match in frame */
uint8_t data[14] = { 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,
                     0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA };
/* mask information, match on first two bytes */
uint8_t mask[14] = { 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00,
                     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };

int txFilter = SES_IGNORE_FIELD;
int rxFilter = SES_IGNORE_FIELD;
int saveSyntonized = 0;
int saveFreerunning = 0;
int srcOverride = 1;
int override = 0;
int supervisory = 0;
int seqRecovery = SES_IGNORE_FIELD;
int sequenceRecoveryMode = 0;
/* route number of the added route */
int routeNum;
rv = SES_AddExtendedStaticRoute_Simple(ethertype, &data, &mask, txFilter, rxFilter, portMap, save▶
Syntonized, saveFreerunning, cutThrough, override, srcOverride, supervisory, seqRecovery, sequenceReco▶
```

DRIVER EXAMPLES

```
veryMode, &routeNum);
```

An existing extended table entry can be updated or removed. By calling the SetExtendedLookupEntryPortMap() API, user can revise the port map for an existing route. Additionally, the SES_UninstallExtLookupEntry () and SES_RemoveExtLookupRule () API can remove an extended table entry and extended lookup rule. Both APIs require the route number returned when the entry was placed in the table to be known and passed to the API.

```
/*  Example of updating the port map of an existing extended table entry */

/* Ethertype to match in frame */
uint16_t ethertype = 0xA000;
uint8_t data[] = { 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,
                      0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA };
/* mask information, match on first two bytes */
uint8_t mask[] = { 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00,
                      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }; /
int txFilter = SES_IGNORE_FIELD;
int rxFilter = SES_IGNORE_FIELD;
int portMap = 0x0002;
int saveSyntonized = 0;
int saveFreerunning = 0;
int cutThrough = 1;
/*  Use Extended table portmap return information */
int override = 0;
int srcOverride = 1;
int supervisory = 0;
int seqRecovery = SES_IGNORE_FIELD;
int sequenceRecoveryMode = 0;
/* route number of the added route */
int routeNum;
rv = SES_AddExtendedStaticRoute_Simple(ethertype, &data, &mask,
    txFilter, rxFilter, portMap,
    saveSyntonized, saveFreerunning, cutThrough,
    override, srcOverride, supervisory,
    seqRecovery, sequenceRecoveryMode, &routeNum);
/* new port map for the route */
portMap = 0x0010;

rv = SetExtendedLookupEntryPortMap(routeNum, portMap);
```

Configuring Transmit Transform

Replacing Source or Destination MAC Address

The example demonstrates how to update either the source or destination MAC address for egressing traffic using a port specific transmit transform API, SES_TxPortSetTxXformEntry. In the example, the source MAC address of the egress traffic on Port 1 and Port 2 will be replaced by the destAddr if the destination MAC matches the destinationAddr configured in the static entry. To update the destination MAC address instead, the srcAddrReplace parameter in SES_TxPortSetTxXformEntry must be set to 0.

DRIVER EXAMPLES

Note: Either the source or the destination MAC address can be updated at a time, not both. Details about the Transmit Transform API can be found in the SES_frer.h header and SES_frer.c code files.

```
void updateMACAddr_Example()
{
    uint8_t destinationAddr[6] = { 0xF8,0xE4,0x3B,0x10,0x49,0xC9 };
    uint8_t Addr_mask[6] = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };
    uint64_t tempAddr[6] = { 0xaa,0xaa,0xaa,0xaa,0xaa,0xaa };
    uint64_t destAddr = 0;
    int32_t rv = 0;

    int32_t txAddrXformIndex = SES_GetTxTransformIndex(0x06);

    printf("Transform Index %d\n", txAddrXformIndex);

    //Convert 8bit to 64bit
    for (int i = 0; i < 6; i++) {
        destAddr |= ((uint64_t)tempAddr[i] << (8 * (5 - i)));
    }

    rv = SES_TxPortSetTxXformEntry(
        SES_macPort1, // int port
        txAddrXformIndex, // int entry
        0, // int timeSize
        0, // int timeOffset
        0, // int timeSourceFree
        0, // int peerDelaySelect
        SES_noTimeOperation, // SES_DRV_txPortTimeOperation_t timeOperation
        0, // int srfIndex
        0, // int srfEnable
        SES_noTagOperation, // SES_DRV_txPortTagOperation_t tagOperation
        destAddr, // uint64_t destAddress
        1, // int destAddrSwapEn
        0, // int vlanPCP
        0, // int vlanDEI
        0, // uint16_t vlanID
        SES_dynamicTblVlanNoTagOp, // vlanOperation
        1, // int srcAddrReplace
        0); // portIdReplace

    printf("Tx transform :: %d\n", rv);

    rv = SES_TxPortSetTxXformEntry(
        SES_macPort2, // int port
        txAddrXformIndex, // int entry
        0, // int timeSize
        0, // int timeOffset
        0, // int timeSourceFree
        0, // int peerDelaySelect
        SES_noTimeOperation, // SES_DRV_txPortTimeOperation_t timeOperation
        0, // int srfIndex
        0, // int srfEnable
        SES_noTagOperation, // SES_DRV_txPortTagOperation_t tagOperation
        destAddr, // uint64_t destAddress
        1, // int destAddrSwapEn
        0, // int vlanPCP
```

DRIVER EXAMPLES

```
        0, // int vlanDEI
        0, // uint16_t vlanID
        SES_dynamicTblVlanNoTagOp, // SES_DRV_txPortVLANOperation_t vlanOperation
        1, // int srcAddrReplace
        0); // portIdReplace

    printf("Tx transform :: %d\n", rv);

    rv = SES_AddStaticTableEntryEx(
        1, // lookupPriority
        destinationAddr, // MACaddr_p
        Addr_mask, // MACmask_p
        SES_DYNTBL_NO_VLAN, // vlanID
        0, // vlanMask
        0, // sourceEntry
        0, // override
        0, // sourceOverride
        0, // hsrOrPrpSupervisory
        SES_dynamicTblLookupBasic, // lookupType
        0x06, // portMap
        0, // sendToAE
        0, // danp
        0, // cutThrough
        0, // syntTimestamp
        0, // localTimestamp
        dynamicTblNoSequenceOp, // sequenceMgmt
        0, // rxSequenceSet
        txAddrXformIndex, // transmitFilter
        SES_DYNTBL_NO_NTRY, // receiveFilter
        NULL); // index_p

        printf("Static Entry :: %d\n", rv);
}
```

IPV4/IPV6 EXISTING RULES AND EXTENDED TABLE ENTRIES

The firmware installs rules for IPv4 (Ethertype 0x0800) and IPv6 (Ethertype 0x86DD) entries based on the following offsets and sizes. As the rule exists already, user does not need to install another rule and can simply use the helper functions to install IPv4/IPv6 extended table entries as shown below.

```
/*SES_SetExtLookupRule() API description includes the following information about the existing rules
created by firmware for IPv4/IPv6 entries. */

* @note   If a rule with the same Ethertype has already been set, and the
 *         field count, offsets and sizes match, the set request will be ignored.
 *         Otherwise a parameter error will be returned.
 *
 *         IPV4 and IPV6 rules are installed at the startup of AE. The rules are as follows:
 *         IPv4 superset rule offsets and sizes
 *         This rule will use a total of 2 extended table rows
 *         0: DSCP: offset:1, size:1
 *         1: Protocol offset:9, size:1
 *         2: Src and Dest Addr offset:12, size:8
 *         3: Src and Dest Port offset:20, size:4
 *         4: IPv4 UDP data inspect offset:28, size:7
```

DRIVER EXAMPLES

```
 *
 *         IPv6 superset rule offsets and sizes
 *         This rule will use a total of 4 extended table rows
 *         0: DSCP: offset:0, size:2 (needs to be masked 0x0F,0xC0)
 *         1: Protocol offset:6, size:1
 *         2: Src Addr offset:8, size:16
 *         3: Dst Addr offset:24, size:16
 *         4: Src and Dest Port offset:40, size:4
 *         5: IPv6 UDP data inspect offset:48, size:8
 *         6: IPv6 TCP data inspect offset:60, size:9
```

Example of IPv4 Extended Entry

The following example shows how to install an IPv4 entry into the extended table using the helper function SES_AddExtendedStatic-Route_IPv4TcpUdp (). For IPv4, as a rule has already been installed by firmware, a rule does not need to be installed. The SES_AddExtended-StaticRoute_IPv4TcpUdp () function calls to the SES_InstallExtLookupEntry () API to install the entry. To perform extended table lookup, user must also either install a static table entry passing SES_dynamicTblLookupExt to the lookup mode or alternatively enable extended table lookup for the port using SES_SetRxPortLookupMode().

```
/* Example of adding an IPv4 extended table entry using SES_AddExtendedStaticRoute_IPv4TcpUdp helper
function */

uint8_t dscp = 0x00;
/* 0x11 for UDP */
uint8_t nextProto = 0x11;
/* IP address 10:10:10:10 */
uint32_t srcIpAddrArr[4] = { 10, 10, 10, 10 };
uint32_t srcIpAddr = (srcIpAddrArr[0] << 24) | (srcIpAddrArr[1] << 16) | (srcIpAddrArr[2] << 8) |
(srcIpAddrArr[3]);
/* IP address 10:10:10:20 */
uint32_t dstIpAddrArr[4] = { 10, 10, 10, 20 };
uint32_t dstIpAddr = (dstIpAddrArr[0] << 24) | (dstIpAddrArr[1] << 16) | (dstIpAddrArr[2] << 8) |
(dstIpAddrArr[3]);
uint16_t srcPortArr[2] = { 0x00, 0x00 };
uint16_t srcPort = (srcPortArr[0] << 8) | (srcPortArr[1]);
uint16_t dstPortArr[2] = { 0x00, 0x00 };
uint16_t dstPort = (dstPortArr[0] << 8) | (dstPortArr[1]);
int portMap = 0x02;
int routeNum;

/* Call the Switch driver API to make an entry in the extended lookup table */
rv = SES_AddExtendedStaticRoute_IPv4TcpUdp(
        dscp,    // Differentiated Services Code Point Field to match
        nextProto,    // Protocol field to match, 6 for TCP, 17 for UDP
        srcIpAddr,    // source IPv4 address
        dstIpAddr,    // destination IPv4 address
        srcPort,    // source port
        dstPort,    // destination port
        SES_IGNORE_FIELD,    // tx filter
        SES_IGNORE_FIELD,    // rx filter
        portMap,    // ports to forward the frame to
        0,    // no syntonized timestamp
```

DRIVER EXAMPLES

```
        0,    // no local timestamp
        0,    // Store & Forward entry
        0,    // set to 0 to not override blocked or learning state on a port
        1,    // set to 1 to use Extended table portmap return information
        0,    // not a supervisory frame
        SES_IGNORE_FIELD,    // no sequence recovery
        0,    // no sequence recovery
        &routeNum);
```

Alternatively, the same IPv4 entry can be installed using the SES_InstallExtLookupEntry() API as shown below.

```
/* Example of installing the same IPv4 entry */

int32_t ses_install_extended_lookup_entry_ipv4_example(void) {
    int32_t rv = SES_PORT_ERROR;

        /* Example of adding an IPv4 extended table entry using SES_InstallExtLookupEntry*/
    uint16_t ethertype = 0x0800;

        /*
        * DCSP
        * Next protocol
        * Source IP address and Destination IP address
        * Source port and Destination port number
        */
    uint16_t fieldOffsets[] = { 1, 9, 12, 20};
    uint16_t fieldSizes[] = { 1, 1, 8, 4};


    int fieldCount = sizeof(fieldSizes)/sizeof(fieldSizes[0]);

        /*
        * DCSP
        * Next protocol
        * Source IP address
        * Destination IP address
        * Source port
        * Destination port number
        */

    uint8_t fieldData[] = { 0,
                0x11,
                10, 10, 10, 10,
                10, 10, 10, 20,
                0x00, 0x00,
                0x00, 0x00};

    uint8_t fieldMask[] = { 0xFC,
                0xFF,
                0xFF, 0xFF, 0xFF, 0xFF,
                0xFF, 0xFF, 0xFF, 0xFF,
                0xFF, 0xFF,
                0xFF, 0xFF};
```

DRIVER EXAMPLES

```
    int portMap = 0x02;
    int extendedTableIndex;

        rv = SES_InstallExtLookupEntry(ethertype,    // ethertype
            fieldCount,        // fieldCount
            fieldOffsets,        // fieldOffsets
            fieldSizes,        // fieldSizes
            fieldData,        // fieldData
            fieldMask,        // fieldMask
            SES_IGNORE_FIELD,    // txFilter
            SES_IGNORE_FIELD,    // rxFilter
            portMap,    // portMap
            0,    // saveSyntonized
            0,    // saveFreerunning
            0,    // cutThrough
            0,    // override, set to 0 to not override blocked or learning state on a port
            1,    // srcOverride, set to 1 to use Extended table portmap return information
            0,    // supervisory
            SES_IGNORE_FIELD,    // seqRecovery
            0,    // sequenceRecoveryMode
            &extendedTableIndex);    // extendedTableIndex

    return rv;
}
```

Example of IPv6 Extended Entry

The following example shows how to install an IPv6 entry into the extended table using the helper function SES_AddExtendedStatic-Route_IPv6TcpUdp (). For IPv6, as a rule has already been installed by firmware, a rule does not need to be installed. The SES_AddExtended-StaticRoute_IPv6TcpUdp () function calls to the SES_InstallExtLookupEntry () API to install the entry. To perform extended table lookup, user must also either install a static table entry passing SES_dynamicTblLookupExt to the lookup mode or alternatively enable extended table lookup for the port using SES_SetRxPortLookupMode()

```
/* Example of adding an IPv6 extended table entry using SES_AddExtendedStaticRoute_IPv6TcpUdp helper
function */
uint8_t diffSer = 0x00;
uint8_t nextHeader = 0x11;
/* IPv6 address 11:2233:4455:6677:8899:aabb:ccdd:eeff */
uint8_t srcIpAddr[16] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
                         0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff};

/* IPv6 address 11:2233:4455:6677:8899:aabb:ccdd:ee00 */
uint8_t dstIpAddr[16] = { 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
                        0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0x00 };

uint16_t srcPort = 0;
uint16_t dstPort = 0;
int portMap = 0x02;
int routeNum;
/* Call the ADINx310 driver API to make an entry in the extended lookup table */
```

DRIVER EXAMPLES

```
rv = SES_AddExtendedStaticRoute_IPv6TcpUdp(diffSer,
    nextHeader,     // Protocol field to match, 6 for TCP, 17 for UDP
    &srcIpAddr,     // source IPv6 address
    &dstIpAddr,     // destination IPv6 address
    srcPort,        // source port
    dstPort,        // destination port
    SES_IGNORE_FIELD,// tx filter
    SES_IGNORE_FIELD,// rx filter
    portMap,        // ports to forward the frame to
    0,              // no syntonized timestamp
    0,              // no local timestamp
    0,              // Store & Forward entry
    0,              // set to 0 to not override blocked or learning state on a port
    1,              // set to 1 to use Extended table portmap return information
    0,              // not a supervisory frame
    SES_IGNORE_FIELD,// no sequence recovery
    0,              // no sequence recovery
    &routeNum);
```

Similarly, the IPv6 extended table entry can be installed using the SES_InstallExtLookupEntry API as follows:

```
/* Example of adding an IPv6 extended table entry using SES_InstallExtLookupEntry() API*/

int32_t ses_install_extended_lookup_entry_ipv6_example(void) {
int32_t rv = SES_PORT_ERROR;

uint16_t ethertype = 0x86DD;

/*
* Differentiated services
* Next header
* Source IP address
* Destination IP address
* Source port number
* IPv6 UDP data inspect
* IPv6 TCP data inspect
*/

uint16_t fieldOffsets[] = { 0,
                            6,
                            8,
                            24,
                            40,
                            48,
                            60 };

uint16_t fieldSizes[] = { 2,
                          1,
                          16,
                          16,
                          4,
                          8,
                          9 };
```

DRIVER EXAMPLES

```
int fieldCount = sizeof(fieldSizes)/sizeof(fieldSizes[0]);

uint8_t fieldData[2 + 1 + 16 + 16 + 2 + 2];
/* Differentiated services */
fieldData[0] = (0 >> 2);
fieldData[1] = (0 << 6);
/* Next header field to match, 6 for TCP, 17 for UDP*/
fieldData[2] = 0x11;
uint8_t SrcIpv6Addr[] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff};
uint8_t DstIpv6Addr[] = { 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0x00 };
/* Source IP address */
memcpy(&fieldData[3], SrcIpv6Addr, sizeof(SrcIpv6Addr));
/* Destination IP address */
memcpy(&fieldData[3 + sizeof(SrcIpv6Addr)], DstIpv6Addr, sizeof(DstIpv6Addr));

uint8_t SrcPrt[] = {0x00, 0x00};
uint8_t DstPrt[] = {0x00, 0x00};

/* Source port and destination port number */
fieldData[3 + sizeof(SrcIpv6Addr) + sizeof(DstIpv6Addr)] = SrcPrt[0];
fieldData[3 + sizeof(SrcIpv6Addr) + sizeof(DstIpv6Addr) + 1] = SrcPrt[1];
fieldData[3 + sizeof(SrcIpv6Addr) + sizeof(DstIpv6Addr) + 2] = DstPrt[0];
fieldData[3 + sizeof(SrcIpv6Addr) + sizeof(DstIpv6Addr) + 3] = DstPrt[1];


uint8_t fieldMask[] = { 0x0F, 0xC0, // Differentiated services
                        0xFF, // Next protocol
                        0xFF, 0xFF, 0xFF, 0xFF, // Source IP address
                        0xFF, 0xFF, 0xFF, 0xFF,
                        0xFF, 0xFF, 0xFF, 0xFF,
                        0xFF, 0xFF, 0xFF, 0xFF,
                        0xFF, 0xFF, 0xFF, 0xFF, // Destination IP address
                        0xFF, 0xFF, 0xFF, 0xFF,
                        0xFF, 0xFF, 0xFF, 0xFF,
                        0xFF, 0xFF, 0xFF, 0xFF,
                        0xFF, 0xFF, 0xFF, 0xFF, // Source port and destination port number
                        0x00, 0x00, 0x00, 0x00, // IPv6 UDP data inspect
                        0x00, 0x00, 0x00, 0x00,
                        0x00, 0x00, 0x00, 0x00, // IPv6 TCP data inspect
                        0x00, 0x00, 0x00, 0x00, 0x00 };

int portMap = 0x04;
int extendedTableIndex;

rv = SES_InstallExtLookupEntry(ethertype, // ethertype
    fieldCount, // fieldCount
    fieldOffsets, // fieldOffsets
    fieldSizes, // fieldSizes
    fieldData, // fieldData
    fieldMask, // fieldMask
    SES_IGNORE_FIELD, // txFilter
    SES_IGNORE_FIELD, // rxFilter
    portMap, // portMap
    0, // saveSyntonized
```

DRIVER EXAMPLES

```
    0, // saveFreerunning
    0, // cutThrough
    0, // override, set to 0 to not override blocked or learning state on a port
    1, // srcOverride, set to 1 to use Extended table portmap return information
    0, // supervisory
    SES_IGNORE_FIELD, // seqRecovery
    0, // sequenceRecoveryMode
    &extendedTableIndex); // extendedTableIndex

return rv;
}
```

USING EXTENDED TABLE TO REPRIORITIZE (PSFP STREAM GATE) BASED ON ETHERTYPE

The static table, extended table and PSFP functionality can be used to reprioritize traffic into a different transmit queue based on the extended lookup information. This feature uses the Stream filter/gate functionality (see SES_psfp.h) and the reprioritization is applied at the ingress port.

There are two possible approaches to achieving this. The first approach is to perform an extended lookup on all ingressing frames and the second approach performs an extended lookup only on frames that match a static table entry.

Approach 1: Perform Extended Lookup on All Ingressing Frames and Reprioritizing Matched Frames

This approach performs an extended lookup on all frames and adds a stream filter to reprioritize all traffic with 0x88b6 ethertype to transmit port Queue 7 (highest priority). The example shows the functions for the stream entry, the extended table entry and configuring the lookup mode for the receive port to perform an extended lookup on all frames.

```
/**
*     @brief Helper function to set port stream filter and stream gate
*
*     @param[in]   mac: MAC identifier of the Rx port where the stream filter must be applied
*     @param[in]   ipv: The desired priority value
*     @param[in]   rxStreamGateIndex: Index for the stream gate that needs to be created. 0-15 are the
valid values
                Note: Use SES_PSFP_GetStreamGateIndex API to get an available stream gate ID
*     @param[in]   rxFilterIndex: Index for the stream filter that needs to be created. 0-31 are the
valid values
                   Note: Use SES_PSFP_GetStreamFilterIndex API to get an available stream filter ID
*
*     @return
*       - #SES_OK      operation succeeded
*       - #SES_ERROR   unable to set stream filter
*/
int32_t AddStreamFilterEntry(SES_mac_t mac,
    uint8_t ipv,
    uint8_t rxStreamGateIndex,
    uint8_t rxFilterIndex)
{
    int32_t rv = SES_ERROR;
    int32_t i;
    static int routeNum = 0;
    TSN_ieee802_dot1q_psfp_stream_filter_instance_table_config_t streamFilterInstanceTable;
```

DRIVER EXAMPLES

```
    TSN_ieee802_dot1q_psfp_stream_gate_instance_table_config_t streamGateInstanceTable;

    /* Clear the contents of the structure before configuring */
    memset(&streamFilterInstanceTable, 0, sizeof(TSN_ieee802_dot1q_psfp_stream_filter_instance_ta▶
ble_config_t));
    memset(&streamGateInstanceTable, 0, sizeof(TSN_ieee802_dot1q_psfp_stream_gate_instance_table_con▶
fig_t));

    /* Configure the stream filter and stream gates */
    /* Stream filter parameters */
    streamFilterInstanceTable.stream_filter_instance_id = rxFilterIndex; //filterIndex
    streamFilterInstanceTable.priority_spec_type.priority_spec = 0; //pcp
    streamFilterInstanceTable.stream_handle_spec.stream_handle = rxFilterIndex; //streamID
    streamFilterInstanceTable.max_sdu_size = 0xFFFF; //maxSDU
    streamFilterInstanceTable.flow_meter_ref = 0xFF; //flowMeterID
    streamFilterInstanceTable.stream_gate_ref = rxStreamGateIndex; //gateID
    streamFilterInstanceTable.stream_blocked_due_to_oversize_frame_enabled = 0; //blockEnable
    /* Stream gate parameters */
    streamGateInstanceTable.stream_gate_instance_id = rxStreamGateIndex;
    streamGateInstanceTable.gate_enable = true;
    streamGateInstanceTable.config_change = true;
    streamGateInstanceTable.admin_gate_states = 1;
    streamGateInstanceTable.admin_ipv = 0x0;
    streamGateInstanceTable.admin_control_list_length = 0x01;
    streamGateInstanceTable.admin_control_list[0].ipv_spec = ipv; //ipv Set
    streamGateInstanceTable.admin_control_list[0].gate_state_value = 1; //gateOpen
    streamGateInstanceTable.admin_control_list[0].time_interval_value = 1000000; // gate nanoseconds

    streamGateInstanceTable.admin_cycle_time.numerator = 1; //numerator in seconds
    streamGateInstanceTable.admin_cycle_time.denominator = 1000; //denominator in seconds, Admin cycle
time = 1ms (1/1000)

    /* Call the ADINx310 driver API to add stream gate entry */
    rv = SES_PSFP_SetStreamGateActive(mac, &streamGateInstanceTable);

    /* Call the ADINx310 driver API to add stream filter entry */
    rv = SES_PSFP_SetStreamFilterActive(mac, &streamFilterInstanceTable);

    return rv;
}

/**
*     @brief Helper function to add an entry in the extended table
*
*     @param[in]   portMap: egress port indicator
*     @param[in]   rxFilterIndex: Streamfilter ID that needs to be applied on the frame that gets a
hit in extended lookup table
*     @param[in]   ethertype: ethertype to match for this extended static route
*     @param[in]   cutThrough: 0 -> don't use cut-through forwarding
*                           1 -> use cut-through
*     @param[out]  routeNum_p: pointer to location in which route number
*                  will be copied
*
*     @return
*       - #SES_OK      operation succeeded
*       - #SES_ERROR   unable to set gate parameters
```

DRIVER EXAMPLES

```
*/
int32_t AddExtendedTableEntry(uint8_t portMap,
    uint8_t rxFilterIndex,
    uint16_t ethertype,
    uint8_t cutThrough,
    int32_t* routeNum_p)
{
    int32_t rv = SES_ERROR;
    /* If only interested to match on ethertype, then set Field data to zero */
    uint8_t fieldData[14];
    memset(&fieldData[0], 0, sizeof(fieldData));
    uint8_t fieldMask[14];
    memset(&fieldMask[0], 0, sizeof(fieldData));
    /* Call the ADINx310 driver API to make an entry in the extended lookup table */
    rv = SES_AddExtendedStaticRoute_Simple(ethertype, //ethertype
                &fieldData, //data_p
                &fieldMask, //mask_p
                SES_IGNORE_FIELD, //txFilter
                rxFilterIndex, //rxFilter
                portMap, //portMap
                0, //saveSyntonized
                0, //saveFreerunning
                cutThrough, //cutThrough
                0, //override, set to 0 to not override blocked or learning state on a port
                1, //srcOverride, set to 1 to use Extended table portmap return information
         0, //supervisory
                SES_IGNORE_FIELD, //seqRecovery
                0, //sequenceRecoveryMode
                routeNum_p); //routeNum_p
    return rv;
}

//pseudo code
#include "SES_switch.h"
void main(void) {

    int32_t rv = -1;
    SES_mac_t mac = SES_macPort0;//port to apply rx filter at (ingress port)
    uint8_t ipv = 7;//highest priority. Puts the frame in Q7 of the egress port
    uint8_t portMap = 0x04; //Egress port Port2
    uint16_t ethertype = 0x88b6;
    int32_t routeNum = 0;
    uint8_t cutThrough = 0; // store-forward mode for psfp operation.
    uint8_t rxFilterIndex = SES_PSFP_GetStreamFilterIndex(0x01); //Get available stream filter ID for
Port0
    uint8_t rxStreamGateIndex = SES_PSFP_GetStreamGateIndex(0x01); //Get available stream gate ID for
Port0
    uint32_t regValue = 0;

    /* Call helper function to set a stream filter */
    rv = AddStreamFilterEntry(mac, ipv, rxStreamGateIndex, rxFilterIndex);

    /* Call helper function to add an entry in the extended lookup table */
    rv |= AddExtendedTableEntry(portMap, rxFilterIndex, ethertype, cutThrough, &routeNum);

    /* Call the Switch driver API for enabling the extended lookup for all frames on the chosen Rx
```

DRIVER EXAMPLES

```
port*/
    rv |= SES_GetRxPortLookupMode(mac, &regValue);
    if (rv == 0) {
        regValue |= SES_RX_PORT_EXT_LOOKUP;
        rv = SES_SetRxPortLookupMode(mac, regValue);
    }
}
```

Approach 2: Perform Extended Lookup and Reprioritization Only on Frames That Match an Entry in the Static Table

This approach only performs an extended lookup on frames that match an existing entry in the static table. Similar to the first example, a stream filter is added to reprioritize all traffic with 0x88b6 ethertype to transmit port Queue 7 (highest priority). The example shows the static table configuration for the receive port to perform the extended lookup only on frames of interest and uses the AddStreamFilterEntry and AddExtendedTableEntry helper functions from the Approach 1: Perform Extended Lookup on All Ingressing Frames and Reprioritizing Matched Frames section. In this example the reprioritization is being performed based on the Static table information and the Ethertype.

```
/**
* This example uses the AddStreamFilterEntry and
* AddExtendedTableEntry helper functions used above in approach 1.
*/


void main(void) {

    int32_t rv = -1;
    SES_mac_t mac = SES_macPort0;//port to apply rx filter at (ingress port)
    uint8_t ipv = 7;//highest priority. Puts the frame in Q7 of the egress port
    int16_t vlanId = 1; //vlan id
    uint8_t portMap = 0x04; //Egress port Port2
    uint16_t ethertype = 0x88b6;
    int32_t routeNum = 0;
    int32_t index;
    uint8_t cutThrough = 0; // store-forward mode for psfp operation.
    uint8_t rxFilterIndex = SES_PSFP_GetStreamFilterIndex(0x01); //Get available stream filter ID for
Port0
    uint8_t rxStreamGateIndex = SES_PSFP_GetStreamGateIndex(0x01); //Get available stream gate ID for
Port0
    uint8_t macAddr[6] = { 0x11,0x22,0x33,0x44,0x55,0x66 }; //sample mac address
    uint8_t macMask[6] = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };

    /* Call helper function to set a stream filter */
    rv = AddStreamFilterEntry(mac, ipv, rxStreamGateIndex, rxFilterIndex);

    /* Call helper function to add an entry in the extended lookup table */
    rv |= AddExtendedTableEntry(portMap, rxFilterIndex, ethertype, cutThrough, &routeNum);

    /* Call the Switch driver API for adding a static table entry */
    rv |= SES_AddStaticTableEntryEx(1,                          // lookup priority
        &macAddr,                   // mac address
        &macMask,                   // mac mask
        SES_DYNTBL_NO_VLAN,         // vlan id
```

DRIVER EXAMPLES

```
        SES_DYNTBL_VLAN_EXACT_MATCH, // vlan mask
        0,                           // source entry
        0,                           // override
        0,                           // source override
        0,                           // hsr prp supervisory
        SES_dynamicTblLookupExt,     // lookup type -> extended table lookup
        portMap,                     // port map
        0,                           // send to ae
        0,                           // danp
        cutThrough,                  // cut through -> store and forward operation
        0,                           // synt timestamp
        0,                           // local timestamp
        ETH_dynamicTblNoSequenceOp,  // sequencemgmt
        0,                           // rx sequence set
        SES_IGNORE_FIELD,            // transmit filter
        SES_IGNORE_FIELD,            // receive filter
        &index);                       // index*

}
```

READING THE DYNAMIC TABLE (LEARNED ENTRIES)

The Forwarding table consists of a total of 2048 entries, 128 entries are allocated to Static entries and 1918 entries are available for dynamic/learned entries. It is possible to read back the MAC addresses learned by the switch using the SES_ReadDynamicTable() API.

The SES_GetDynTblEntryUsage() API provides detail on how many entries are in the table including the 128 static entries. It is not possible to read the static entries back through the Dynamic table read, instead use the specific SES_ReadStaticTableEntry() API. The SES_ReadDynamicTable() API supports reading back the full contents of the dynamic table and supports a maximum of 50 table entries per call. To read the full dynamic table, iterative calls are required. The following example demonstrates how to read the full dynamic table in blocks of 50 entries. During dynamic table read operation, learning is disabled until the read operation is completed. New entries will not be learned during this time.

```
void Ses_ExampleReadDynamicTable(void) {

    /*Read 50 entries*/
    int count = 0;
    int startIndex = 0;
    int lastIndex = 49;

    /*Array to store 50 entries */
    SES_dynTblEntry_t dynTblEntry[50];
    uint16_t validEntries_p;
    uint16_t entryCount;

    /*Get valid table entries */
    printf("Entry count %d\n", SES_GetDynTblEntryUsage(&entryCount));
    printf("Entry count :: %d\n", entryCount);

    /* Condition check if entries are less than 50 */
    if ((entryCount - (int16_t)128) < lastIndex) {
        lastIndex = (entryCount - (int16_t)128) - 1;
    }
```

DRIVER EXAMPLES

```
    /* Read table until all the valid entries are read */
    while (lastIndex < ((entryCount - (int16_t)128))) {

        /* initial 128 entries are blocked by Static entries */
        if (SES_OK != SES_ReadDynamicTable(startIndex, lastIndex, dynTblEntry, &validEntries_p)) {
            printf("Reading Dynamic Table failed!!\n");
            break;
        }

        for (int i = 0; i <= (lastIndex - startIndex); i++) {
            printf("Index :: %d\t", i);
            printf("Dynamic Table MacAddress % 02x % 02x % 02x % 02x % 02x % 02x \n\n",
                dynTblEntry[i].macAddress[0],
                dynTblEntry[i].macAddress[1],
                dynTblEntry[i].macAddress[2],
                dynTblEntry[i].macAddress[3],
                dynTblEntry[i].macAddress[4],
                dynTblEntry[i].macAddress[5]);
        }
        /* Control variables to maintain starts and last index */
        if (lastIndex + 1 == (entryCount - (int16_t)128)) {
            break;
        }
        startIndex = lastIndex + 1;
        if ((startIndex + 49) < (entryCount - (int16_t)128)) {
            lastIndex = startIndex + 49;
        }
        else {
            lastIndex = (entryCount - (int16_t)128) - 1;
        }
    }
}
```

READING/WRITING PHY REGISTERS OVER MDIO

There are a number of APIs for Read/Write PHY registers are used for configuring and controlling the PHYs as per their register map defined in datasheets. For ADI PHYs or “Known PHYs” (ADIN1100, ADIN1200, and ADIN1300) using these APIs may be helpful for accessing any specific PHY features not included in the switch configuration or for debugging purposes. From a PHY configuration point of view the ADI PHYs can be managed directly by the switch as part of the initial switch configuration.

For PHYs that are not in the list of "Known PHYs", these PHYs must be managed by the host through the switch MDIO interface. In this case, the MDIO APIs must be called in combination with switch driver corresponding port configuration APIs. For example, when SES_WritePhyReg() is called to configure the speed and duplex settings of the PHY, SES_SetPhyConfig must be called to inform SES about the speed and duplex that was set on to the PHY. This allows the host to directly control both the umanaged PHY and the switch port configuration to ensure they match.

MDIO read/write APIs are included in the ‘SES_switch.h’ file. MDIO access is available to all PHYs connected to the SES switch MDIO bus. The SES switch MDIO bus is a leader interface and can access PHYs at the port/PHY address locations provided to the SES_InitializePorts() API. The API needs to know what port the PHY is connected to and interfaces directly to that PHY address.

When an ADI PHY is connected (ADIN1100, ADIN1200, ADIN1300), the SES_ReadPhyReg() and SES_WritePhyReg() APIs can directly access extended register space. When interfacing with the ADIN1100 PHY, the device address is important and must be passed for the particular register space of interest.

DRIVER EXAMPLES

In the event other vendor PHYs are used in the application with the switch, access to extended space must be done through the specific pointer/data registers for that PHY.

```
//Example of reading PHY register on Port2
static int32_t test_SES_ReadPhyReg(void) {
    int32_t rv = SES_OK;
    SES_mac_t port = SES_macPort2;
    uint32_t reg_addr = 0x00000001;
    uint16_t data;
    rv = SES_ReadPhyReg(port, reg_addr, &data);
    return rv;
}
//=============================================================================
//Example of writing to extended space directly for ADIN1300 PHY on Port 3. When writing to the
ADIN11200/ADIN1300 PHYs, the device address can be left as 0, as the switch firmware takes care of
access to the extended space.
static int32_t test_SES_WritePhyReg(void) {
    int32_t rv = SES_OK;
    SES_mac_t port = SES_macPort3;
    uint32_t reg_addr = 0x00008E27; //FLD_EN Enhanced link detect register
    uint16_t data = 0x0000; //Disable Enhanced link detect function
    rv = SES_WritePhyReg(port, reg_addr, data);

}
```

```
//Example of reading extended register through pointer/data register for ADIN1300 PHY
static int32_t test_SES_ReadPhyRegExtended(void) {
    int32_t rv = SES_OK;
    SES_mac_t port = SES_macPort2;
    uint32_t reg_addr_pt = 0x00000010; //Pointer register for ADIN1300 PHY
    uint32_t reg_addr_dt = 0x00000011; //Pointer register for ADIN1300 PHY
    uint16_t data;
    uint16_t ext_reg = 0x8E27;
    rv = SES_WritePhyReg(port, reg_addr_pt, ext_reg); //Pass the register of interest to pointer
register

    rv = SES_ReadPhyReg(port, reg_addr_dt, &data); //Read from the data register
}
```

```
//Example of writing to vendor specific registers for the ADIN1100 PHY
static int32_t test_SES_WritePhyRegADIN1100(void) {
    int32_t rv = SES_OK;
    SES_mac_t port = SES_macPort1;
    uint16_t addr; // address to write
    uint16_t dev_addr; // Device address for register grouping
    uint32_t reg_addr; // Combined device address + register address
    uint16_t data = 0x0; // Disable Frame Checker
    dev_addr = 0x001F; //Vendor Specific register section 2
    addr = 0x8C8001; // Frame Checker register
        reg_addr = (uint32_t)dev_addr << 16 | (uint32_t)addr;
        rv = SES_WritePhyReg(port, reg_addr, &data);
```

DRIVER EXAMPLES

```
}


//Example of reading vendor specific registers for the ADIN1100 PHY
static int32_t test_SES_ReadPhyRegADIN1100(void) {
    int32_t rv = SES_OK;
    SES_mac_t port = SES_macPort1;
    uint16_t addr; // address to write
    uint16_t dev_addr; // Device address for register grouping
    uint32_t reg_addr;  // Combined device address + register address
    uint16_t data;

    dev_addr = 0x001E; //Vendor Specific register section 1
    addr = 0x8C82; // LED Control Register
        reg_addr = (uint32_t)dev_addr << 16 | (uint32_t)addr;
        rv = SES_ReadPhyReg(port, reg_addr, &data)
```

CVLAN CONFIGURATION

VLAN APIs are included in the 'SES_vlan.h' file. The switch supports C-VLANs (0 through 4095) and provides user ability to configure the VLAN table learning/forwarding capability, remap VLANs on all ports and reprioritization VLANs. VLAN 4095 is used for untagged traffic.

The default VLAN table behavior is for No Learn/No Forward for all VLAN IDs, with exception of untagged and VID 0 for all ports. During initial configuration, user will need to configure the VLAN table directly or configure ports as Trunk or Access type in order to forward traffic with different VLAN IDs across the switch.

The switch supports two methods of configuring the device for VLAN support. The first is port based Trunk/Access configuration and the second is directly configuring the VLAN table for individual VIDs/ports.

Trunk/Access Port Type Configuration

A trunk port can support traffic for multiple different VLANs, while an Access port only supports a single VLAN. Traffic egressing an Access port is untagged, so has the VLAN tag removed before it transmits out. Traffic ingressing an Access port has the Access port VLAN tag with VID & PCP inserted into the frame.

When using the SES_SetVlanPortType API to configure a switch port for Trunk/Access function, the API call takes care of the configuration of the VLAN table and any insertion/removal of VLAN tags required as shown below for each VLAN Port type.

Trunk Port Type: SES_SetVlanPortType(mac, Vid_start, Vid_end, VlanHeaderPcp, TRUNK);

- Switch sets the VLAN Mode to Learn & Forward for range of VIDs from **vid_start** to **vid_end.** vlanHeaderPcp only applies for Access port.
- Switch enables Unicast and Multicast Return Tx filters with TX Transform entry id.
- Switch creates Static Entry for handling untagged traffic.

Access Port Type: SES_SetVlanPortType(mac, Vid, Vid, VlanHeaderPcp, ACCESS);

- Switch sets the VLAN Mode to Learn & Forward for **vid**, Access ports only support 1 VID, so start = end, priority can be configured using vlanHeaderPcp.
- Switch enables VLAN Header Insertion for **vid**. Untagged traffic sent into the Access port gets a tag inserted with vid.
- Switch enables VLAN Remap for VLAN ID = 0 (Priority tagged packets) mapped to **vid**.
- Switch creates TX transform entry with txPortRemoveVLAN option. Traffic egressing the Access port has the tag removed before transmitting.
- Switch enables Unicast and Multicast Return Tx filters with TX Transform entry id.

DRIVER EXAMPLES

Not a Member: SES_SetVlanPortType(mac, 0, 0, 0, NOTMEMBER);

- Switch sets the VLAN mode for all VLANs to No Learn & No Forward
- Switch disables VLAN Header Insertion
- Switch disables Unicast and Multicast Return Tx filters
- Switch disables VLAN Remapping
- Switch removes Transform Entry and Static Entry

The following example and Figure 15 shows the VLAN configuration for a number of ports with a mix of Trunk and Access Port configurations. Traffic only forwards between ports that belong to the same VIDs.

Trunk to Trunk Port: Tagged traffic (VID 2-5) coming from the Trunk port 0 to Trunk Port 5, egresses on Port 5 with the VLAN tag. Untagged traffic ingressing, egresses on the other port untagged.

Trunk to Access Port: VID traffic (VID2) ingressing the Trunk port, egresses on Access ports 1 & 2 with the VLAN tag removed.

Access to Trunk Port: Untagged traffic ingressing Access ports 1 & 2 have a VLAN tag with VID 2 inserted into the frame and transmit out Port 0/5 with the VLAN tag.

Access Port to Access Port: Untagged traffic ingressing Port 1 has a tag inserted on the Rx side and tag removed on the Tx side, the untagged frame is transmitted out Port 2. Tagged traffic received on Port 1 has to have VLAN tag VID2 and will transmit out of Port 2 Access port with the tag removed.

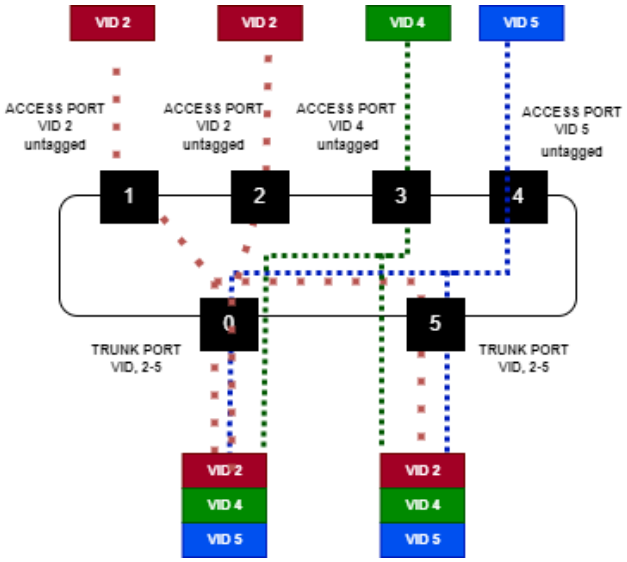

Figure 15. VLAN Trunk/Access Port overview

```
void portConfigNormalMode() {
uint8_t vlanHeaderPcp = 0;

//Port 0 & 5, VID 2-5, VLAN Tagged traffic: Trunk Ports, priority doesn't matter for Trunk ports
SES_SetVlanPortType(SES_macPort0, (uint16_t)2, (uint16_t)5,vlanHeaderPcp, SES_vlanTrunk);
SES_SetVlanPortType(SES_macPort5, (uint16_t)2, (uint16_t)5, vlanHeaderPcp, SES_vlanTrunk);

//Port 1 & 2, VID 2, priority 3, Access Port
vlanHeaderPcp = 3;
SES_SetVlanPortType(SES_macPort1, (uint16_t)2, (uint16_t)2, vlanHeaderPcp, SES_vlanAccess);
SES_SetVlanPortType(SES_macPort2, (uint16_t)2, (uint16_t)2, vlanHeaderPcp, SES_vlanAccess);
      //Port 3, VID 4, priority 4 Access Port
vlanHeaderPcp = 4;
```

DRIVER EXAMPLES

```
SES_SetVlanPortType(SES_macPort3, (uint16_t)4, (uint16_t)4, vlanHeaderPcp, SES_vlanAccess);
      //Port 4, VID 5, priority 5 Access Port
vlanHeaderPcp = 5;
SES_SetVlanPortType(SES_macPort4, (uint16_t)5, (uint16_t)5, vlanHeaderPcp, SES_vlanAccess);
}
```

VLAN Table: Example CVLAN Configuration

This is the alternative method to configuring the switch VLAN table and should only be used if Trunk/Access port configuration is not suitable for the application use case. The following example code shows how to set the port mode for learning and forwarding for all active VIDs on a specific port in the first section and for no learn and no forwarding operation in the second portion. Applying a mode or map to any VID aside from “vlanNoLearn_NoForward” will add it to the active VLAN list. A maximum of 64 VLANs can be configured on the switch.

Note: when the host interface uses one of the MAC interfaces as the host communications/control path, do not disable forwarding on the host port interface for all VIDs using the SES_SetVlanPortModeAll() API as this will also stop messages forwarding from the Host to the Packet Assist engine.

Note: The SES_SetVlanPortModeAll() API will only apply the mode to the active VIDs.

```
/*The following will set the mode as SES_vlanLearn_Forward for all VIDs on Port2. */

int32_t rv = SES_OK;
SES_mac_t mac = SES_macPort2;
SES_vlanMode_t mode = SES_vlanLearn_Forward;
rv = SES_SetVlanPortModeAll(mac, mode);

/*The following will set the mode as no Learn and no forward for all VIDs on Port2. Do not disable
forwarding on Host connected ethernet port. */

int32_t rv = SES_OK;
SES_mac_t mac = SES_macPort2;
SES_vlanMode_t mode = SES_vlanNoLearn_NoForward;
rv = SES_SetVlanPortModeAll(mac, mode);
```

The following is an example to set port mode as ‘SES_vlanLearn_Forward’ for a specific VID on a specific port.

```
/* The following will set the mode as SES_vlanLearn_Forward for VID = 10 on Port2. */
int32_t rv = SES_OK;
SES_mac_t mac = SES_macPort2;
uint16_t VID = 10;
SES_vlanMode_t mode = SES_vlanLearn_Forward;
rv = SES_SetVlanPortMode(mac, VID, mode);
```

The following is an example to set port mode as ‘SES_vlanLearn_Forward’ for a specific VID on several (or) all ports.

```
/* The following will set the mode as SES_vlanLearn_Forward for VID = 10 on
 * Port2 and Port3.
 * Map to set SES_vlanLearn_Forward on Port2 and Port3:
 * Reserved|port 5|port 4|port 3|port 2|port 1|port 0|
 * | 0000  |  00  |  00  |  11  |  11  |  00  |  00  |
```

DRIVER EXAMPLES

```
 */
int32_t rv = SES_OK;
uint16_t VID = 10;
uint16_t map = 0x00F0;
rv = SES_SetVlanMode(VID, map);
```

Inserting a VLAN Tag

The following example shows how to insert a VLAN tag into untagged traffic. Inserting a VLAN tag uses a Transmit Transform and a static table entry. Details about the Transmit Transform API can be found in the SES_frer.h header and SES_frer.c code files.

```
/* First find an available Tx Transform Index for configured portMap using SES_GetTxTransformIndex().
Then create a Tx Transform entry with that index for each port of interest in portMap with VLAN ID,
VLAN PCP, VLAN DEI and use Insert VLAN Tag Operation with API call SES_TxPortSetTxXformEntry().
Now map the Tx Transform entry index to transmitFilter on Static Entry using SES_AddStaticTableEn▶
tryEx(). */

        SES_mac_t mac = SES_macPort4;
    int timeSize = 0;
    int timeOffset = 0;
    int timeSourceFree = 0;
    int peerDelaySelect = 0;
    SES_timestampOperation_t timeOperation = SES_noTimeOperation;
    int srfIndex = 0;
    int srfEnable =0;
    SES_tagOperation_t tagOperation = SES_noTagOperation;
    uint64_t destAddress[6] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
    int destAddrSwapEn = 0;
    int vlanPcp = 7; // Priority of 7
    int vlanDei= 0;
    uint16_t vlanId = 1; // VLAN ID of 1
    SES_dynTblVlanTagOp_t vlanOperation = SES_dynamicTblVlanInsertTag; // Insert VLAN Tag
    int srcAddrReplace = 0;
    int portIdReplace = 0;

// Get Tx Transform index for Port 4
    int32_t filterindex = SES_GetTxTransformIndex(0x10); //Get Tx Transform index for Port4
    int entry = filterindex;

    result = SES_TxPortSetTxXformEntry(mac,entry,timeSize,timeOffset, timeSourceFree,peerDelay▶
Select,timeOperation,srfIndex,srfEnable,        tagOperation,destAddress,destAddrSwapEn,vlanPcp,vlan▶
Dei,vlanId,vlanOperation,srcAddrReplace,portIdReplace);
```

```
/* Now install a static entry with the macMask as all 0's, so all traffic going out that port has VLAN
tag inserted */
    uint8_t lookupPriority = 0;
    uint8_t macAddr_p[6] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x77 };
    uint8_t macMask_p[6] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };/
    vlanId = 4095;
    uint16_t vlanMask = 0xFFF;
    uint8_t sourceEntry = 0;
```

DRIVER EXAMPLES

```
    uint8_t override = 0;
    uint8_t sourceOverride = 0;
    uint8_t hsrOrPrpSupervisory = 0;
    SES_dynTblLookup_t lookupType = SES_dynamicTblLookupBasic;
    uint8_t portMap = 0x10;
    uint8_t sendToAE = 0;
    uint8_t danp = 0;
    uint8_t cutThrough = 0;
    uint8_t syntTimestamp = 0;
    uint8_t localTimestamp = 0;
    SES_dynTblSeqMgmt_t sequenceMgmt = dynamicTblNoSequenceOp;
    int rxSequenceSet = 0;
    int transmitFilter = filterindex; // Tx Transform index to insert VLAN tag
    int receiveFilter = -1;
    int index_p;

    result = SES_AddStaticTableEntryEx(lookupPriority, &macAddr_p, &macMask_p, vlanId,
            vlanMask, sourceEntry, override, sourceOverride,
            hsrOrPrpSupervisory, lookupType, portMap, sendToAE,
            danp, cutThrough, syntTimestamp, localTimestamp,
            sequenceMgmt, rxSequenceSet, transmitFilter, receiveFilter,
            &index_p);
```

Removing a VLAN Tag

Similarly, a transform can be used to remove a VLAN tag. The switch expects a minimum sized VLAN tagged frame to be 68 bytes. If user is ingressing frames of 64-bytes which include a VLAN tag and configuring the switch to remove the VLAN tag directly or using VLAN access port, the switch will deliberately corrupt the frame on egress. This is a known errata and is captured in the Silicon Anomaly section of the ADIN3310/ADIN6310 datasheet.

TIME SYNCHRONIZATION

Time Synchronization APIs are included in the 'SES_time_synchronization.h' file.

The switch currently supports the following Time Synchronization profiles:

- IEEE 802.1 AS-2020
- IEEE 1588-2019 Delay Request-Response (End-to-End, E2E)
- IEEE 1588-2019 Annex I.4 Peer-to-Peer Default PTP profile

Support for IEEE C37.238-2017 will be added in future software updates.

The gPTP stack is not enabled by default and must be explicitly enabled if the user intends to use any of the Time Synchronization profiles with the stack running on the packet assist engine.

Enabling time synchronization requires the following sequence:

1. Call the SES_PtpStart() API to enable the gPTP stack.
2. Configure CMLDS.
3. Create the clock instance by identifying the clock identity, domain number, clock number, and associated link ports.

When an ADI PHY is used, the firmware automatically applies the corresponding PHY Tx/Rx delay parameters based on the PHY type and link speed.

NOTE: For non-ADI PHYs, the Tx/Rx delay parameters must be provided by the user.

IEEE 802.1AS PROFILE

The following examples show the configuration of the switch PTP stack for AS 2020 profile.

If only one time domain is enabled, user has choice of Peer-to-Peer delay mechanism (TSN_ptp_delay_mechanism_p2p) or Common Mean Link Delay Service (CMLDS) using parameter TSN_ptp_delay_mechanism_common_p2p. CMLDS provides the mean propagation delay and neighbor rate ratio to all active domains. When using AS2020 with multiple domains, CMLDS should be enabled.

The following example shows the Time Synchronization configuration for one PTP instance with multiple link ports and the application of the P2P delay mechanism. The example can be modified to use the CMLDS delay by updating the delay mechanism from TSN_ptp_delay_mechanism_p2p to TSN_ptp_delay_mechanism_common_p2p.

```
void SES_Single_AS_PtpInstance_Example(void) {

    if (SES_OK == SES_PtpStart()) {
        /* Configure CMLDS, Clock identity */
        const TSN_ptp_init_cmlds_ds_t initDs = { {0x7A, 0xC6, 0xBB, 0x11, 0x11, 0x11, 0xff, 0xff } };
        printf("SES_PtpInitCmlds :: %d\n", SES_PtpInitCmlds(&initDs));

        /* Create instance with all ports */
        TSN_ptp_instance_type_t instanceType = TSN_ptp_instance_type_ptp_relay;

        /* initDs_p parameters:: clock_identity, clock_number, domain_number */
        TSN_ptp_init_instance_ds_t initDs_p = { {0x7A, 0xC6, 0xBB, 0x11, 0x11, 0x11}, 0, 0 };
        uint32_t instanceIndex_p;

        /* Enabling PTP on each port  */
        uint16_t numberPtpPorts = 6;
        uint16_t linkPortNumber[6] = { 1, 2, 3, 4, 5, 6 };
        printf("Create PTP instance ADIN6310 status :: %d\n", SES_PtpCreatePtpInstance(instanceType,
numberPtpPorts, linkPortNumber, &initDs_p, &instanceIndex_p));

        /* Initialize default_ds */
        uint32_t instanceIndex = 0;
```

TIME SYNCHRONIZATION

```
        TSN_ptp_default_ds_t default_ds;
        TSN_ptp_current_ds_t current_ds;
        TSN_ptp_port_ds_t port_ds;

        printf("SES_PtpGetDefaultDs - %d\n", SES_PtpGetDefaultDs(instanceIndex, &default_ds));
        default_ds.instance_enable = 1;
        default_ds.gm_capable = 0;
        printf("SES_PtpSetDefaultDs - %d\n", SES_PtpSetDefaultDs(instanceIndex, &default_ds));

        /* Enable Instance 1 PTP Ports with P2P delay mechanism */
        printf("SES_PtpGetPortDs :: %d\n", SES_PtpGetPortDs(instanceIndex, 0, &port_ds));
        port_ds.delay_mechanism = TSN_ptp_delay_mechanism_p2p;
        port_ds.port_enable = 1;

        printf("PTP Ports 2 :: %d\n", numberPtpPorts);
        for (int i = 0; i < numberPtpPorts; i++) {
            printf("Set Port DS for Port :: %d :: %d\n", i, SES_PtpSetPortDs(instanceIndex, i,
&port_ds));
        }
    }
}
```

The following examples show the Time Synchronization configuration for switches with two instances and configured for CMLDS delay mechanism.

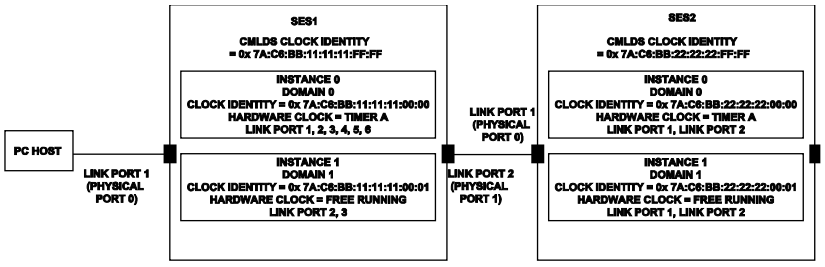

Figure 16. Example Time Synchronization Configuration for two switches with two instances on different link ports

```
int32_t ses_test_multiple_ptp_instance_example(void) {
    int32_t rv = 0;

    if (SES_OK != SES_PtpStart())
        return;

    /*Configure CMLDS, Clock identity */
    const TSN_ptp_init_cmlds_ds_t initDs = {
        .clock_identity = { 0x7a, 0xc6, 0xbb, 0x11, 0x11, 0x11, 0xff, 0xff }
    };
    rv = SES_PtpInitCmlds(&initDs);
    printf("SES_PtpInitCmlds - %d\n", rv);

    /* Create instance with six ports */
    uint16_t numberPtpPorts1 = 6;
    uint16_t linkPortNumber1[6] = { 1, 2, 3, 4, 5, 6 };
```

TIME SYNCHRONIZATION

```
    TSN_ptp_init_instance_ds_t initDs1 = {
        .clock_identity = { 0x7a, 0xc6, 0xbb, 0x11, 0x11, 0x11, 0x00, 0x00 },
        .clock_number = 0,  //Timer A
        .domain_number = 0
    };
    uint32_t* instanceIndex1;

    rv = SES_PtpCreatePtpInstance(TSN_ptp_instance_type_ptp_relay, numberPtpPorts1, &linkPortNumber1,
&initDs1, &instanceIndex1);
    printf("SES_PtpCreatePtpInstance - %d\n", rv);

    /* Initialize default_ds */
    TSN_ptp_default_ds_t default_ds;
    rv = SES_PtpGetDefaultDs(instanceIndex1, &default_ds);
    default_ds.instance_enable = 1;
    rv = SES_PtpSetDefaultDs(instanceIndex1, &default_ds);
    printf("SES_PtpSetDefaultDs - %d\n", rv);

    uint16_t numberPtpPorts2 = 2;
    uint16_t linkPortNumber2[2] = { 1, 2 };
    TSN_ptp_init_instance_ds_t initDs2 = {
        .clock_identity = { 0x7a, 0xc6, 0xbb, 0x11, 0x11, 0x11, 0x00, 0x01 },
/* Second instance must run from free running clock */
        .clock_number = 0x80000000,
        .domain_number = 1
    };

    uint32_t* instanceIndex2;

    rv = SES_PtpCreatePtpInstance(TSN_ptp_instance_type_ptp_relay, numberPtpPorts2, &linkPortNumber2,
&initDs2, &instanceIndex2);
    printf("SES_PtpCreatePtpInstance - %d\n", rv);

    rv = SES_PtpGetDefaultDs(instanceIndex2, &default_ds);
    default_ds.instance_enable = 1;
    rv = SES_PtpSetDefaultDs(instanceIndex2, &default_ds);
    printf("SES_PtpSetDefaultDs - %d\n", rv);

    /*Initialize port_ds*/
    TSN_ptp_port_ds_t port_ds;
    rv = SES_PtpGetPortDs(instanceIndex1, 0, &port_ds);
    printf("SES_PtpGetPortDs 1 - %d\n", rv);

    /*Enable Instance 1 PTP Ports for all 6 ports*/
    port_ds.delay_mechanism = TSN_ptp_delay_mechanism_common_p2p;
    port_ds.port_enable = 1;
    rv = SES_PtpSetPortDs(instanceIndex1, 0, &port_ds);
    printf("SES_PtpSetPortDs 1 - %d\n", rv);
    rv = SES_PtpSetPortDs(instanceIndex1, 5, &port_ds);
    printf("SES_PtpSetPortDs 6 - %d\n", rv);

    /*Enable Instance 2 PTP Ports for ports 1 & 2*/
    port_ds.delay_mechanism = TSN_ptp_delay_mechanism_common_p2p;
    port_ds.port_enable = 1;
    rv = SES_PtpSetPortDs(instanceIndex2, 0, &port_ds);
    printf("SES_PtpSetPortDs 1 - %d\n", rv);
```

TIME SYNCHRONIZATION

```
    rv = SES_PtpSetPortDs(instanceIndex2, 1, &port_ds);
    printf("SES_PtpSetPortDs 2 - %d\n", rv);

    printf("ses_test_create_multiple_ptp_instance_example, rv - %d\n", rv);
    return rv;
}
```

If configuring the PTP instance to be connected to an IEEE802.1AS 2011 device, ensure that the delay mechanism is configured to be TSN_ptp_delay_mechanism_p2p and domain number 0. The following example shows the required API calls to configure the PTP stack with backward compatibility support for IEEE802.1AS 2011.

```
 int32_t ses_test_as2011_example(void) {
    int32_t rv = 0;

    if (SES_OK != SES_PtpStart())
        return;
    /*Configure CMLDS, Clock identity*/
    const TSN_ptp_init_cmlds_ds_t initDs = { {0x7a, 0xc6, 0xbb, 0x11, 0x11, 0x11, 0xff, 0xff } };
    printf("SES_PtpInitCmlds :: %d\n", SES_PtpInitCmlds(&initDs));


    uint16_t numberPtpPorts = 6;
    uint16_t linkPortNumber[6] = { 1, 2, 3, 4, 5, 6 };
    TSN_ptp_init_instance_ds_t init_s = {
        .clock_identity = { 0x7a, 0xc6, 0xbb, 0x11, 0x11, 0x11, 0x00, 0x00 },
        .clock_number = 0,
        .domain_number = 0
    };
    uint32_t instanceIndex;
    rv = SES_PtpCreatePtpInstance(TSN_ptp_instance_type_ptp_relay, numberPtpPorts, &linkPortNumber,
&init_s, &instanceIndex);

    /* Initialize default_ds*/
    TSN_ptp_default_ds_t default_ds;
    rv = SES_PtpGetDefaultDs(instanceIndex, &default_ds);
    default_ds.instance_enable = 1;
    rv = SES_PtpSetDefaultDs(instanceIndex, &default_ds);
    printf("SES_PtpSetDefaultDs - %d\n", rv);

    /*Initialize port_ds with peer to peer delay*/
    TSN_ptp_port_ds_t port_ds;
    rv = SES_PtpGetPortDs(instanceIndex, 0, &port_ds);
    port_ds.delay_mechanism = TSN_ptp_delay_mechanism_p2p,
    port_ds.port_enable = 1;

    rv = SES_PtpSetPortDs(instanceIndex, 0, &port_ds);
    rv = SES_PtpSetPortDs(instanceIndex, 1, &port_ds);
    rv = SES_PtpSetPortDs(instanceIndex, 2, &port_ds);
    rv = SES_PtpSetPortDs(instanceIndex, 3, &port_ds);
    rv = SES_PtpSetPortDs(instanceIndex, 4, &port_ds);
    rv = SES_PtpSetPortDs(instanceIndex, 5, &port_ds);

    return rv;
```

```
}
```

Reading PTP Parameters

The example demonstrates how to read various parameter related to the configured time sync profile, port data structure, and CMLDS parameter.

```
/*
* The following example demonstrates how to read various parameters
* related to the configured Time Sync profile, Port Data Structure,
* and CMLDS parameters.
*/
void getPtpParameters(void) {

    //PTP default structure
    TSN_ptp_default_ds_t default_ds;

    //PTP current structure
    TSN_ptp_current_ds_t current_ds;

    //PTP Port structure
    TSN_ptp_port_ds_t port_ds;

    //CMLDS port structure
    TSN_ptp_cmlds_link_port_ds_t cmlds_p;

    //PTP Parent node structure
    TSN_ptp_parent_ds_t ds_p;

    //To print CMLDS parameters, assign Input as 2
    int32_t input = 1;


    char pState[10][20] = { " ", "initializing", "Faulty", "Disabled", "Listening", "Pre_timeTransmit▶
ter", "timeTransmitter", "Passive", "Uncalibrated", "timeReceiver" };

    switch (input) {

    case 1:
        SES_PtpGetDefaultDs(0, &default_ds);
        int ports = default_ds.number_ports;
        printf("Number of ports :: %d\n", ports);

        for (int i = 0; i < ports; i++) {
            printf("Port :: %d\t", i);
            printf("SES_PtpGetPortDs :: %d\t", SES_PtpGetPortDs(0, i, &port_ds));
            printf("AS Capable :: %s\t", port_ds.as_capable ? "true" : "false");
            printf("Port State :: %s\t", pState[port_ds.port_state]);
            printf("Synch Status :: %s\t", port_ds.sync_locked ? "true" : "false");
            printf("Mean Link Delay :: %d ns\t", port_ds.mean_link_delay / 65535);
            SES_PtpGetParentDs((uint32_t)0, &ds_p);
            printf("MAC Address:: ");
            for (int i = 0; i < 6; i++) {
                printf("%02X", ds_p.grandmaster_identity[i]);
```

```
                if (i < 5) {
                    printf(":");
                }
            }printf("\n");

        }
        break;

    case 2:

        //Check port wise CMLDS active or not, retrieve Port wise CMLDS parameters
        for (uint16_t i = 0; i < 3; i++) {
            SES_PtpGetCmldsLinkPortDs(i, &cmlds_p);
            printf("Cmlds Link Port %d :: %s \t ", i, cmlds_p.cmlds_link_port_enabled ? "true" :
"false");
            if (cmlds_p.cmlds_link_port_enabled) {
                printf("Mean Link Delay : %d ns\n", cmlds_p.mean_link_delay / 65535);
            }
            else {
                printf("CMLDS False\n");
            }
        }

        break;

    default:
        printf("Invalid input !!\n");
        break;
    }

}
```

IEEE 1588 2019 PROFILE

IEEE 1588-2019 is the Precision Time Protocol (PTP) standard for precision clock synchronization protocol in networked measurement and control systems. This profile is used in industrial, instrumentation and power grid applications.

Figure 17 illustrates a typical network setup using IEEE 1588 PTP for synchronizing clocks across networked devices. This configuration is commonly used in complex network environments. The switch supports configuration as a boundary clock, an ordinary clock, an End-to-End (E2E) transparent clock, or a Peer-to-Peer (P2P) transparent clock. The switch also supports propagation delay measurement using either the Delay Request-Response or Peer-to-Peer mechanism.

Key differences between the 1588 Profiles:

- E2E: Only the Time Receiver calculates the path delay with its upstream Time Transmitter
- P2P: Every node continuously measures its link delay with both neighbors, distributing the calculation across the entire chain.
- Transparent Clock (TC): A TC-P2P adds residence time, and performs peer delay measurements on its port.

The P2P delay mechanism differs from the E2E mechanism. In P2P mode, each device in the chain, Ordinary, Boundary, and Transparent Clocks, measures the delay of its own links with directly connected peers. This measurement is performed using Peer Delay messages (Pdelay_Req, Pdelay_Resp, and Pdelay_Resp_Follow_Up), which are exchanged in both the upstream and downstream directions.

TIME SYNCHRONIZATION

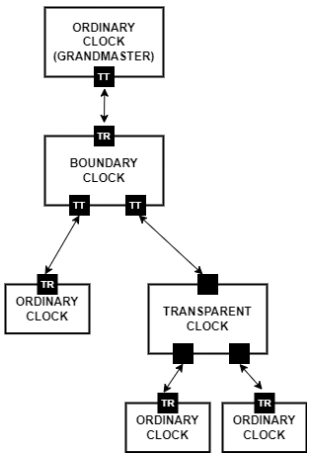

Figure 17. Overview of 1588 devices showing Grandmaster, Boundary, Ordinary and Transparent Clocks

To enable a PTP instance for the IEEE 1588 profile, the following is a summary of the API calls required:

- SES_PtpStart(): Starts the PTP stack.
- SES_PtpCreatePtpInstance(): Creates a PTP instance.
- SES_PtpSetDefaultDs(): Configures the default data set.
- SES_PtpSetPortDs(): Configures the port data set. This API must be called once per port.
- SES_PtpSetTimestampCorrectionPortDs(): Configures the ingress and egress latency (PHY delay values). This API must also be called once per port. When using ADI PHYs, the delay values can be obtained by calling SES_GetPhyDelays() API and then passed to SES_PtpSetTimestampCorrectionPortDs().

The following examples demonstrate how to configure the switch PTP stack for 1588 profile (E2E) as a boundary clock, an ordinary clock and a transparent clock.

```
void Ses_BoundaryClock_1588_Example() {
    int32_t rv = 0;
    uint32_t instanceIndex;
    TSN_ptp_default_ds_t default_ds;
    uint16_t numberPtpPorts = 6;
    uint16_t linkPortNumber[6] = { 1,2,3,4,5,6 };
    TSN_ptp_timestamp_correction_port_ds_t ds_p;
    int16_t rxDelay_p;
    int16_t txDelay_p;

    /* Initialize PTP Stack */
    if (SES_OK != SES_PtpStart()) {
        printf("PTP initialization failed !!\n");
        return;
    }

    /* Initialize Instance and its parameters */
    TSN_ptp_init_instance_ds_t init_s = {
    .clock_identity = { 0x7a, 0xc6, 0xbb, 0x22, 0x22, 0x22, 0x00, 0x00 },
    .clock_number = 0,
    .domain_number = 0
    };
    rv = SES_PtpCreatePtpInstance(TSN_ptp_instance_type_bc, numberPtpPorts, linkPortNumber, &init_s,
&instanceIndex);
```

TIME SYNCHRONIZATION

```
    /* Initialize Default Data set */
    rv = SES_PtpGetDefaultDs(instanceIndex, &default_ds);
    default_ds.instance_type = TSN_ptp_instance_type_bc;
    default_ds.instance_enable = 1;
    default_ds.priority1 = 248;
    default_ds.priority2 = 248;
    rv = SES_PtpSetDefaultDs(instanceIndex, &default_ds);
    printf("SES_PtpSetDefaultDs - %d\n", rv);


    /* PTP port data set Configuration */
    TSN_ptp_port_ds_t port_ds;
    rv = SES_PtpGetPortDs(instanceIndex, 0, &port_ds);
    port_ds.delay_mechanism = TSN_ptp_delay_mechanism_e2e,
        port_ds.port_enable = 1;

    /* Per port PTP instance configuration */
    for (int port = 0; port < 6; port++) {
        printf("SES_PtpSetPortDs for Port :: %d and rv :: %d\n", port, SES_PtpSetPortDs(instanceIndex,
port, &port_ds));

        /*To configure the ingress and egress latency(PHY delay values). This must also be called once
per port.*/
        printf("Reading Phy Delays:: %d\n", SES_GetPhyDelays(port + 1, &rxDelay_p, &txDelay_p));

        ds_p.egress_latency = txDelay_p;
        ds_p.ingress_latency = rxDelay_p;

        printf("Set PTP time stamp correction :: %d\n", SES_PtpSetTimestampCorrectionPortDs(instanceIn▶
dex, port, &ds_p));

        rxDelay_p = 0;
        txDelay_p = 0;
    }
}
```

```
void Ses_OrdinaryClock_1588_GM() {

    int32_t rv = 0;
    uint32_t instanceIndex;
    TSN_ptp_default_ds_t default_ds;
    uint16_t numberPtpPorts = 1;
    uint16_t linkPortNumber[1] = { 1 };

    if (SES_OK != SES_PtpStart()) {
        printf("PTP initialization failed !!\n");
        return;
    }


    //Initialize Instance and its parameters
    TSN_ptp_init_instance_ds_t init_s = {
    .clock_identity = { 0x7a, 0xc6, 0xbb, 0x11, 0x11, 0x11, 0x00, 0x00 },
```

TIME SYNCHRONIZATION

```
    .clock_number = 0,
    .domain_number = 0
    };
    rv = SES_PtpCreatePtpInstance(TSN_ptp_instance_type_oc, numberPtpPorts, linkPortNumber, &init_s,
&instanceIndex);


    //Initialize Default Data set
    rv = SES_PtpGetDefaultDs(instanceIndex, &default_ds);
    default_ds.instance_type = TSN_ptp_instance_type_oc;
    default_ds.instance_enable = 1;
    //Utilize Priority to to make this OC as GM
    default_ds.priority1 = 128;
    default_ds.priority2 = 248;
    rv = SES_PtpSetDefaultDs(instanceIndex, &default_ds);
    printf("SES_PtpSetDefaultDs - %d\n", rv);


    // PTP port Configuration
    TSN_ptp_port_ds_t port_ds;
    rv = SES_PtpGetPortDs(instanceIndex, 0, &port_ds);
    port_ds.delay_mechanism = TSN_ptp_delay_mechanism_e2e,
        port_ds.port_enable = 1;

    printf("SES_PtpSetPortDs :: %d\n", SES_PtpSetPortDs(instanceIndex, 0, &port_ds));
}
```

```
void Ses_TransparentClock_1588_E2E(void) {

    int32_t rv = 0;
    uint32_t instanceIndex;
    TSN_ptp_default_ds_t default_ds;
    uint16_t numberPtpPorts = 6;
    uint16_t linkPortNumber[6] = { 1,2,3,4,5,6 };
    TSN_ptp_timestamp_correction_port_ds_t ds_p;
    int16_t rxDelay_p;
    int16_t txDelay_p;

    /* Initialize PTP Stack */
    if (SES_OK != SES_PtpStart()) {
        printf("PTP initialization failed !!\n");
        return;
    }


    /* Initialize PTP Instance and its parameters */
    TSN_ptp_init_instance_ds_t init_s = {
    .clock_identity = { 0x7a, 0xc6, 0xbb, 0x33, 0x33, 0x33, 0x00, 0x00 },
    .clock_number = 0,
    .domain_number = 0
    };
    rv = SES_PtpCreatePtpInstance(TSN_ptp_instance_type_e2e_tc, numberPtpPorts, linkPortNumber, &in▶
it_s, &instanceIndex);
```

TIME SYNCHRONIZATION

```
    /* Initialize Deafault Data set */
    rv = SES_PtpGetDefaultDs(instanceIndex, &default_ds);
    default_ds.instance_type = TSN_ptp_instance_type_e2e_tc;
    default_ds.instance_enable = 1;
    rv = SES_PtpSetDefaultDs(instanceIndex, &default_ds);
    printf("SES_PtpSetDefaultDs - %d\n", rv);


    /*PTP port Data set Configuration */
    TSN_ptp_port_ds_t port_ds;
    rv = SES_PtpGetPortDs(instanceIndex, 0, &port_ds);
    port_ds.delay_mechanism = TSN_ptp_delay_mechanism_e2e,
        port_ds.port_enable = 1;

    /* Per port PTP instance configuration */
    for (int port = 0; port <6; port++) {
        printf("SES_PtpSetPortDs for Port :: %d and rv :: %d\n", port, SES_PtpSetPortDs(instanceIndex,
port, &port_ds));

        /*To configure the ingress and egress latency(PHY delay values). This must also be called per
port.*/
        printf("Reading Phy Delays:: %d\n", SES_GetPhyDelays(port + 1, &rxDelay_p, &txDelay_p));

        ds_p.egress_latency = txDelay_p;
        ds_p.ingress_latency = rxDelay_p;

        printf("Set PTP time stamp correction :: %d\n", SES_PtpSetTimestampCorrectionPortDs(instanceIn▶
dex, port, &ds_p));
        rxDelay_p = 0;
        txDelay_p = 0;
    }
}
```

The following examples demonstrate how to configure the switch PTP stack for 1588 profile (P2P) as a boundary clock, an ordinary clock and a transparent clock.

```
/* Configure Boundary clock */
void Ses_BoundaryClock_1588_Example(void) {

  int32_t rv = 0;
  uint32_t instanceIndex;
  TSN_ptp_default_ds_t default_ds;
  uint16_t numberPtpPorts = 6;
  uint16_t linkPortNumber[6] = { 1,2,3,4,5,6 };
  TSN_ptp_timestamp_correction_port_ds_t ds_p;
  int16_t rxDelay_p;
  int16_t txDelay_p;

  /* Initialize PTP Stack */
  if (SES_OK != SES_PtpStart()) {
      printf("PTP initialization failed !!\n");
      return;
```

TIME SYNCHRONIZATION

```
  }

  /* Initialize Instance and its parameters */
  TSN_ptp_init_instance_ds_t init_s = {
      .clock_identity = { 0x7A, 0xC6, 0xBB, 0x22, 0x22, 0x22, 0x00, 0x00 },
      .clock_number = 0,
      .domain_number = 0
  };
  rv = SES_PtpCreatePtpInstance(TSN_ptp_instance_type_bc, numberPtpPorts, linkPortNumber,
              &init_s, &instanceIndex);

  /* Initialize Deafault Data set */
  rv = SES_PtpGetDefaultDs(instanceIndex, &default_ds);
  default_ds.instance_type = TSN_ptp_instance_type_bc;
  default_ds.instance_enable = 1;
  default_ds.priority1 = 200;
  default_ds.priority2 = 128;
  rv = SES_PtpSetDefaultDs(instanceIndex, &default_ds);
  printf("SES_PtpSetDefaultDs - %d\n", rv);

  /* PTP port data set Configuration */
  TSN_ptp_port_ds_t port_ds;
  rv = SES_PtpGetPortDs(instanceIndex, 0, &port_ds);
  port_ds.delay_mechanism = TSN_ptp_delay_mechanism_p2p,
  port_ds.port_enable = 1;

  /* Per port PTP instance configuration */
  for (int port = 0; port < 6; port++) {
    printf("SES_PtpSetPortDs for Port :: %d and rv :: %d\n", port,
              SES_PtpSetPortDs(instanceIndex, port, &port_ds));

    /* To configure the ingress and egress latency (PHY delay values). This must also be called once
per port. */
    printf("Reading Phy Delays:: %d\n", SES_GetPhyDelays(port + 1, &rxDelay_p, &txDelay_p));

    ds_p.egress_latency = txDelay_p;
    ds_p.ingress_latency = rxDelay_p;

    printf("Set PTP time stamp correction :: %d\n",
              SES_PtpSetTimestampCorrectionPortDs(instanceIndex, port, &ds_p));

    rxDelay_p = 0;
    txDelay_p = 0;
  }

}
```

```
/* Configure Ordinary clock */
void Ses_OrdinaryClock_1588_GM(void) {

  int32_t rv = 0;
  uint32_t instanceIndex;
  TSN_ptp_default_ds_t default_ds;
  uint16_t numberPtpPorts = 1;
```

TIME SYNCHRONIZATION

```
  uint16_t linkPortNumber[1] = { 1 };
  TSN_ptp_timestamp_correction_port_ds_t ds_p;
  int16_t rxDelay_p;
  int16_t txDelay_p;

  /* Initialize PTP Stack */
  if (SES_OK != SES_PtpStart()) {
    printf("PTP initialization failed !!\n");
    return;
  }

  /* Initialize Instance and its parameters */
  TSN_ptp_init_instance_ds_t init_s = {
    .clock_identity = { 0x7A, 0xC6, 0xBB, 0x11, 0x11, 0x11, 0x00, 0x00 },
    .clock_number = 0,
    .domain_number = 0
  };
  rv = SES_PtpCreatePtpInstance(TSN_ptp_instance_type_oc, numberPtpPorts, linkPortNumber,
              &init_s, &instanceIndex);

  /* Initialize Deafault Data set */
  rv = SES_PtpGetDefaultDs(instanceIndex, &default_ds);
  default_ds.instance_type = TSN_ptp_instance_type_oc;
  default_ds.instance_enable = 1;

  /* Utilize Priority to make SES as GM, Lower value, higher priority */
  default_ds.priority1 = 128;
  default_ds.priority2 = 128;
  rv = SES_PtpSetDefaultDs(instanceIndex, &default_ds);
  printf("SES_PtpSetDefaultDs - %d\n", rv);

  /* PTP port data set Configuration */
  TSN_ptp_port_ds_t port_ds;
  rv = SES_PtpGetPortDs(instanceIndex, 0, &port_ds);
  port_ds.delay_mechanism = TSN_ptp_delay_mechanism_p2p,
  port_ds.port_enable = 1;

  printf("SES_PtpSetPortDs :: %d\n",
              SES_PtpSetPortDs(instanceIndex, 0, &port_ds));

  printf("Reading Phy Delays:: %d\n",
              SES_GetPhyDelays(SES_macPort0, &rxDelay_p, &txDelay_p));

  ds_p.egress_latency = txDelay_p;
  ds_p.ingress_latency = rxDelay_p;
  printf("Set PTP time stamp correction :: %d\n",
              SES_PtpSetTimestampCorrectionPortDs(instanceIndex, 0, &ds_p));
}
```

```
/* Configure P2P Transparent clock */
void Ses_TransparentClock_P2P(void) {

  int32_t rv = 0;
```

TIME SYNCHRONIZATION

```
  uint32_t instanceIndex;
  TSN_ptp_default_ds_t default_ds;
  uint16_t numberPtpPorts = 6;
  uint16_t linkPortNumber[6] = { 1,2,3,4,5,6 };
  TSN_ptp_timestamp_correction_port_ds_t ds_p;
  int16_t rxDelay_p;
  int16_t txDelay_p;

  /* Initialize PTP Stack */
  if (SES_OK != SES_PtpStart()) {
    printf("PTP initialization failed !!\n");
    return;
  }


  /* Initialize PTP Instance and its parameters */
  TSN_ptp_init_instance_ds_t init_s = {
    .clock_identity = { 0x7A, 0xC6, 0xBB, 0x33, 0x33, 0x33, 0x00, 0x00 },
    .clock_number = 0,
    .domain_number = 0
  };
  rv = SES_PtpCreatePtpInstance(TSN_ptp_instance_type_p2p_tc, numberPtpPorts, linkPortNumber,
              &init_s, &instanceIndex);


  /* Initialize Deafault Data set */
  rv = SES_PtpGetDefaultDs(instanceIndex, &default_ds);
  default_ds.instance_type = TSN_ptp_instance_type_p2p_tc;
  default_ds.instance_enable = 1;
  rv = SES_PtpSetDefaultDs(instanceIndex, &default_ds);
  printf("SES_PtpSetDefaultDs - %d\n", rv);

  /*PTP port Data set Configuration */
  TSN_ptp_port_ds_t port_ds;
  rv = SES_PtpGetPortDs(instanceIndex, 0, &port_ds);
  port_ds.delay_mechanism = TSN_ptp_delay_mechanism_p2p,
  port_ds.port_enable = 1;

  /* Per port PTP instance configuration */
  for (int port = 0; port < 6; port++) {
    printf("SES_PtpSetPortDs for Port :: %d and rv :: %d\n", port,
              SES_PtpSetPortDs(instanceIndex, port, &port_ds));

    /*To configure the ingress and egress latency(PHY delay values). This must also be called per
port.*/
    printf("Reading Phy Delays:: %d\n",
              SES_GetPhyDelays(port + 1, &rxDelay_p, &txDelay_p));
    ds_p.egress_latency = txDelay_p;
    ds_p.ingress_latency = rxDelay_p;
    printf("Set PTP time stamp correction :: %d\n",
              SES_PtpSetTimestampCorrectionPortDs(instanceIndex, port, &ds_p));
    rxDelay_p = 0;
    txDelay_p = 0;
  }
```

```
}
```

IEEE C37.238-2017 PROFILE

Support for this profile will be added in future software updates.

FRAME PREEMPTION

Frame preemption is a mechanism that allows higher priority traffic to interrupt or preempt lower priority traffic during transmission. This feature is crucial for ensuring that time-sensitive data can be delivered with minimal delay, even in congested networks. Frame preemption is defined in IEEE 802.1Qbu/802.3Qbr.

Frame preemption is designed to be used as part of the TSN suite of features but not with redundancy features such as PRP or HSR. Frame preemption relies on LLDP being enabled first as LLDP messages are used to advertise preemption capability to the link partner through the additional Ethernet capabilities TLVs of the LLDP frames per port.

Frame Preemption APIs are included in the 'SES_preemption.h' file.

EXAMPLE FRAME PREEMPTION CONFIGURATION

Frame preemption configured per port using one of the following two methods:

1. By calling individual frame preemption configuration APIs and then applying the settings by calling SES_PREEMPT_ProcessSettings().
2. By populating a TSN_ieee802_dot1q_frame_preemption_config_t' structure with the desired configuration parameters and passing it as a reference to SES_PREEMPT_SetPreemptionConfig().

```
/* Method 1: Set preemption configuration using multiple preemption APIs */
void sesSetPreemptionConfigurationApi_Example(void) {
  int32_t rv = SES_OK;

  /* Port 5 */
  SES_mac_t mac = SES_macPort5;

  /* 1010 1010: Express queues: 7, 5, 3 and 1 */
  uint8_t express_queue_mask = 0xAA;

  /* Enable preemption support */
  rv |= SES_PREEMPT_SetSupport(mac, true);

  /* Do not ignore peer state */
  rv |= SES_PREEMPT_SetIgnorePeer(mac, false);

  /* Set min frag size to 64 bytes */
  rv |= SES_PREEMPT_SetMinFragmentSize(mac, 0x00);

  /* Set verify period to 20ms */
  rv |= SES_PREEMPT_SetVerifyPeriod(mac, 20);

  /* Send verify messages */
  rv |= SES_PREEMPT_SetVerifyDisable(mac, false);

  /* Set the preemptible and expressqueues */
  rv |= SES_PREEMPT_SetExpressQueueMask(mac, express_queue_mask);

  /* Commit the changes into the switch */
  rv |= SES_PREEMPT_ProcessSettings();

  printf("sesSetPreemptionConfigurationApi_Example :: %d\n", rv);
}

/* Method 2: Set preemption configuration using preemption configuration structure */
void sesSetPreemptionConfigurationStructure_Example(void) {
  int32_t rv = SES_OK;
  SES_mac_t mac = SES_macPort5;
```

FRAME PREEMPTION

```
  TSN_ieee802_dot1q_frame_preemption_config_t config_p = {
    .preempt_enabled = true, /* preempt_enabled */
    .ignore_peer = false, /* ignore_peer */
    .verify_disable = false, /* verify disable */
    .fragment_size = 0x00, /* fragment_size */
    .verify_period = 20, /* verify_period */
    .preemption_table = {
      [0].traffic_class = 0,
      [0].queue_status = TSN_ieee802_dot1q_frame_preemption_express,

      [1].traffic_class = 0,
      [1].queue_status = TSN_ieee802_dot1q_frame_preemption_preemptible,

      [2].traffic_class = 0,
      [2].queue_status = TSN_ieee802_dot1q_frame_preemption_express,

      [3].traffic_class = 0,
      [3].queue_status = TSN_ieee802_dot1q_frame_preemption_preemptible,

      [4].traffic_class = 0,
      [4].queue_status = TSN_ieee802_dot1q_frame_preemption_express,

      [5].traffic_class = 0,
      [5].queue_status = TSN_ieee802_dot1q_frame_preemption_preemptible,

      [6].traffic_class = 0,
      [6].queue_status = TSN_ieee802_dot1q_frame_preemption_express,

      [7].traffic_class = 0,
      [7].queue_status = TSN_ieee802_dot1q_frame_preemption_preemptible,
    }
  };
  rv = SES_PREEMPT_SetPreemptionConfig(mac, &config_p);
  printf("sesSetPreemptionConfigurationStructure_Example :: %d\n", rv);
}
```

It is possible to retrieve the current frame preemption configuration for a specific port using the SES_PREEMPT_GetPreemptionConfig() API. This API allows the user to query whether frame preemption is enabled on a given MAC port and to inspect how each traffic queue is configured, either as express or preemptible.

```
/* Example to retrieve Preemption configuration */
void sesGetPreemptionConfiguration_Example() {
  SES_mac_t mac = SES_macPort5;
  TSN_ieee802_dot1q_frame_preemption_config_t config_p;

  printf("--------------Reading Preemption Confiruration----------\n");
  if (SES_OK == SES_PREEMPT_GetPreemptionConfig(mac, &config_p)) {
    printf("Preemption Enabled :: %s\n", config_p.preempt_enabled ? "true" : "false");
    for (int queue = 0; queue < 8; queue++) {
      switch (config_p.preemption_table[queue].queue_status) {
      case TSN_ieee802_dot1q_frame_preemption_express:
        printf("Queue Configuration :: %d --> Express\n", queue);
        break;
      case TSN_ieee802_dot1q_frame_preemption_preemptible:
```

FRAME PREEMPTION

```
        printf("Queue Configuration :: %d --> Preemptible\n", queue);
        break;
      default:
        printf("Queue Configuration :: %d --> Unknown\n", queue);
        break;
      }
    }
  }
  else {
    printf("Failed to get preemption configuration\n");
  }
}
```

In order to disable preemption for a specific port, simply call SES_PREEMPT_SetSupport() with 'false'. For example, the following call would disable preemption on Port number 5:

```
/* Example to disable preemption */
void sesDisablePreemption_Example(void) {
  int32_t rv = SES_OK;

  /* Disable preemption support on physical port5 */
  rv |= SES_PREEMPT_SetSupport(SES_macPort5, false);
  rv |= SES_PREEMPT_ProcessSettings();

  printf("sesDisablePreemption_Example :: %d\n", rv);
}
```

PREEMPTION STATUS AND STATISTICS

The SES_PREEMPT_CollectStats() API allows the user to retrieve status and statistics information related to frame preemption on a specific port. The returned statistics provide insight into whether preemption is enabled and active, reflect the current configuration, and include counters for activity on both the receive and transmission side.

```
/* Example to get preemption statistics */
void sesGetPreemptionStatistics_Example(void) {

  int32_t rv = SES_OK;
  SES_mac_t mac = SES_macPort5;

  /* pointer to where to store the information */
  SES_preemptionStats_t stats_p;
  rv = SES_PREEMPT_CollectStats(mac, &stats_p);
  printf("sesGetPreemptionStatistics_Example :: %d\n", rv);
  printf("Preemption Supported :: %d for Port :: %d\n",stats_p.supported, mac -1);
  printf("Preemption Verify Status :: %d\n", stats_p.verifyStatus);
  printf("Preemption Enable Tx :: %d\n", stats_p.enableTx);
  printf("Preemption Verify Disable Tx :: %d\n", stats_p.verifyDisableTx);
  printf("Preemption Status Tx :: %d\n", stats_p.statusTx);
  printf("Preemption Verify Time Ms :: %d\n", stats_p.verifyTimeMs);
  printf("Preemption Add Frag Size :: %d\n", stats_p.addFragSize);
  printf("Preemption Frame Assembly Error Count :: %d\n", stats_p.frameAssemblyErrorCount);
  printf("Preemption Frame Smd Error Count :: %d\n", stats_p.frameSmdErrorCount);
  printf("Preemption Frame Assembly Ok Count :: %d\n", stats_p.frameAssemblyOkCount);
  printf("Preemption Frag Count Rx :: %d\n", stats_p.fragCountRx);
  printf("Preemption Frag Count Tx :: %d\n", stats_p.fragCountTx);
```

FRAME PREEMPTION

```
  printf("Preemption Hold Count :: %d\n", stats_p.holdCount);
}
```

ENABLING AND CONFIGURING THE LLDP STACK

LLDP APIs are included in the 'SES_lldp.h' file.

ENABLING LLDP FUNCTION

The Switch supports an LLDP stack running on the packet assist engine. The stack is compatible with the IEEE std 802.1AB – 2016. Note the LLDP stack does not start automatically and needs to be enabled with two API calls, firstly the SES_LLDP_Init() API, followed by the SES_LLDP_Start() API. There is no disable function, therefore, decide whether using the LLDP stack on the switch initially and enable during the initial configuration or alternatively keep it disabled if not required.

The host can subscribe to notifications when there are LLDP events, see the SES_event.h for details on available events related to LLDP function.

The default configuration for LLDP messages is enabled for Rx and Tx, LLDP frames will be transmitted at a 30 second interval.

```
//Example to enable LLDP stack on the switch
rv = SES_LLDP_Init();// Initialize LLDP service and stack. Must be called prior to SES_LLDP_Start()
if (rv == SES_PORT_OK) {
rv = SES_LLDP_Start();// Starts the LLDP service and stack. Enabled LLDP tx/rx on all ports by
default.
}
```

APIs are provided to control LLDP transmission, reception, and addressing. SES_LLDP_SetAdminConfig () can configure a port for transmission only, receive only, or transmit and receive operating modes. The transmission timing of both fast and normal LLDP transmission can also be configured through this API.

NOTE: The maximum number of peers to be supported per port is 4.

LLDP TX APIS

The example shows the LLDP transmission timing, operating mode, destination address, being configured.

```
SES_mac_t mac = SES_macPort0;
SES_LLDP_adminConfig_t adminConfig_p = {
    .lldpAdminConfig = {
      .adminStatus = SES_LLDP_enabledRxTx, // Enable/Disable Rx and/or Tx
      .msgFastTx = 1, // Time interval (in ticks) between fast transmission
      .msgTxInterval = 1, // Time interval (in ticks) between transmission for normal transmission
      .tlvsTxEnable = 0x0F, // Bit map to enable Tx for optional TLVs
      .msgTxHold = 4, // Multiplier for msgTxInterval to determine the value of txTTL
      .notificationEnable = true, // Enables remote statistics change notification
      .reInitDelay = 2, // Delay until reinitialization is attempted when adminStatus becomes 'disa▶
bled'
      .txCreditMax = 5, // Maximum value of txCredit
      .txFastInit = 4, // Initial value for the txFast variable
      .peerSupPerPort = 4, // Number of peers to be supported per port
      .enableEndLldpduTlv = true // Enable or Disable End of LLDPDU TLV
},
     .destMacAddr = { 0x01, 0x80, 0xC2, 0x00, 0x00, 0x0E} // Destination address of LLDP frame
};
rv = SES_LLDP_SetAdminConfig(mac, &adminConfig_p);
```

SES_LLDP_UpdateTxTlv () should only be used to add or update optional TLVs to the LLDP frames transmitted at a port.

ENABLING AND CONFIGURING THE LLDP STACK

The following example shows the transmission TLV for port 0, TLV type 4 (Port Description TLV type), being updated. The SES_LLDP_QueryTxTlv () API can be used to check the TLV at the transmission buffer. The API returns the TLV data and TLV data length.

```
//If device has just powered up & SES_Init() and port configuration is complete, allow some 2-3seconds
for Gigabit speed links to establish
SES_LLDP_Init();
SES_LLDP_Start();
SES_mac_t mac = SES_macPort0;
SES_LLDP_tlvQueryIn_t txTlvQueryIn_p = {
     .uPeerId = 0, // Unique peer identifier for peer entry stored in MIB
     .tlvId.tlvType = 4, // TLV type 4 (Port Description)
     .tlvId.infoLen = 0 // Length of tlvInfo
};
SES_LLDP_tlvQueryOut_t txTlvQueryOut_p;
// Description in TLV data: "ADIN6310, Port 0, SES_LLDP_enabledRxTx"
uint8_t tlvData_p[] = { 0x41, 0x44, 0x49, 0x4E, 0x36, 0x33, 0x31, 0x30, 0x2C, 0x20,
        0x50, 0x6F, 0x72, 0x74, 0x20, 0x30, 0x2C, 0x20, 0x53, 0x45,
        0x53, 0x5F, 0x4C, 0x4C, 0x44, 0x50, 0x5F, 0x65, 0x6E, 0x61,
        0x62, 0x6C, 0x65, 0x64, 0x52, 0x78, 0x54, 0x78 };
uint16_t tlvDataLen = sizeof(tlvData_p);
rv = SES_LLDP_UpdateTxTlv(mac, &txTlvQueryIn_p.tlvId, &tlvData_p, tlvDataLen);
rv = SES_LLDP_QueryTxTlv(mac, &txTlvQueryIn_p, &txTlvQueryOut_p);
```

The following example shows configuration of the System Capabilities in the LLDP frames.

```
SES_LLDP_tlvQueryIn_t txTlvQueryIn_p = {
     .uPeerId = 0,
     .tlvId.tlvType = 7, // TLV type 7 (System Capabilities)
     .tlvId.infoLen = 0
};
SES_LLDP_tlvQueryOut_t txTlvQueryOut_p;
// First two bytes are the system capabilities, the next two bytes are the enabled capabilities
uint8_t tlvData_p[] = = { 0x00, 0x04, 0x00, 0x04 };
uint16_t tlvDataLen = sizeof(tlvData_p);
rv = SES_LLDP_UpdateTxTlv(mac, &txTlvQueryIn_p.tlvId, &tlvData_p, tlvDataLen);
```

LLDP RX APIS

For incoming LLDP frames, SES_LLDP_QueryRxTlv () API can check the TLV data and TLV data length of a particular port and type.

The example below shows querying the receive TLVs. In this scenario, two SES are daisy-chained together for purpose of one device transmitting a TLV and the other device receiving the TLV.

The SES (DEVICE_A) transmits the updated LLDP frames, while the SES (DEVICE_B) queries the port for a particular type of TLV.

```
//This configuration assumes a Multi-SES configuration, where one host is configuring two SES devices.
//The APIs used here are the SES_Mx_... APIs with the device ID passed as a parameter.
//example assumes each device has been initialized and configured previously.

rv = SES_MX_LLDP_Init(DEVICE_A);
rv = SES_MX_LLDP_Init(DEVICE_B);
```

ENABLING AND CONFIGURING THE LLDP STACK

```
rv = SES_MX_LLDP_Start(DEVICE_A);
rv = SES_MX_LLDP_Start(DEVICE_B);
SES_mac_t mac = SES_macPort0;
SES_LLDP_tlvQueryIn_t txTlvQueryIn_p = {
     .uPeerId = 0,
     .tlvId.tlvType = 4,
     .tlvId.infoLen = 0
};
SES_LLDP_tlvQueryOut_t txTlvQueryOut_p;
uint8_t tlvData_p[] = { 0x52, 0x65, 0x20, 0x49, 0x6E, 0x69, 0x74, 0x20, 0x64, 0x65,
            0x6C, 0x61, 0x79, 0x20, 0x69, 0x73, 0x20, 0x32 };
uint16_t tlvDataLen = sizeof(tlvData_p);
rv = SES_MX_LLDP_UpdateTxTlv(DEVICE_B, mac, &txTlvQueryIn_p.tlvId, &tlvData_p, tlvDataLen);
SES_LLDP_adminConfig_t adminConfig_p = {
     .lldpAdminConfig = {
          .adminStatus = SES_LLDP_enabledRxTx,
          .msgFastTx = 1,
          .msgTxInterval = 1,
          .tlvsTxEnable = 0x0F,
          .msgTxHold = 4,
          .notificationEnable = true,
          .reInitDelay = 2,
          .txCreditMax = 5,
          .txFastInit = 4,
          .peerSupPerPort = 4,
          .enableEndLldpduTlv = true
     },
     .destMacAddr = { 0x01, 0x80, 0xC2, 0x00, 0x00, 0x0E}
};
rv = SES_MX_LLDP_SetAdminConfig(DEVICE_B, mac, &adminConfig_p);
Sleep(5000); // Delay to wait for LLDP messages
rv = SES_MX_LLDP_QueryRxTlv(DEVICE_A, mac, &txTlvQueryIn_p, &txTlvQueryOut_p);
```

LLDP STATISTICS

SES_LLDP_GetTxStatistics () and SES_LLDP_GetRxStatistics () are APIs to check the transmit and receive LLDP statistics of a port.

Transmit statistics includes total number of transmitted LLDP frames and total number of TLV length errors found while constructing the transmit buffer.

Receive statistics includes ageouts, discarded frames, error frames, frames received, discarded TLVs, and unrecognized TLVs.

```
SES_mac_t mac = SES_macPort5;
SES_LLDP_txStatistics_t txStats_p;
SES_LLDP_rxStatistics_t rxStats_p;
rv = SES_LLDP_GetTxStatistics(mac, &txStats_p);
rv = SES_LLDP_GetRxStatistics(mac, &rxStats_p);
```

SCHEDULED TRAFFIC EXAMPLES

Scheduled traffic APIs are included in the 'SES_scheduled_traffic.h' file. The following section covers typical configuration of scheduled traffic and any other features relevant to scheduled traffic operations. As scheduled traffic relies on time synchronization, ports running schedules should be time aware for accurate scheduling.

BASIC SCHEDULED TRAFFIC CONFIGURATION

The following is an example to set Scheduled traffic Qbv gate parameters. Review the CVLAN configuration to allow traffic to pass through the switch. By default only untagged and VID 0 traffic will egress, therefore user should enable the VLANs of interest prior to using Scheduled traffic. Scheduled traffic configuration are performed on a per-port basis. In the example below, the gate parameters are set for Port-4 only. To have more and/or different settings for other ports, the 'SES_QbvSetGateParameters()' API must be called again for each port with the required configuration. For each port, the gate control list supports 32 entries.

When configuring the gate control list, the device does allow the sum of the time intervals to be programmed to excess of the cycle time. Any duration that exceeds the cycle time will be ignored.

When enabling Guardbands, ensure that the port has a link established, otherwise, the API will return a -304 error code (SES_GUARD_BAND_INSERTION_UNSUCCESSFUL). The guardband duration depends on link speed. The firmware needs to know the link speed in order to calculate the guardband duration. When the link establishes on a port, the switch checks the speed of the link and uses that information in the guardband calculation. If the link is down, it is not be possible to calculate the correct duration. Thus, the guard band insertion would fail.

```
int32_t rv = SES_OK;
TSN_ieee802_dot1q_types_port_number_t portNumber = 4;
TSN_ieee802_dot1q_sched_gate_parameters_t gateParam;
gateParam.gate_enabled = true;
gateParam.config_change = true;
gateParam.guard_band_gate_event = false;
gateParam.guard_band_hold_event = false;
gateParam.admin_gate_states = 255;
gateParam.admin_control_list[0].operation_name = TSN_ieee802_dot1q_sched_set_and_release_mac;
gateParam.admin_control_list[0].time_interval_value = 300000;
gateParam.admin_control_list[0].gate_state_value = 0xAA;
gateParam.admin_control_list[1].operation_name = TSN_ieee802_dot1q_sched_set_and_release_mac;
gateParam.admin_control_list[1].time_interval_value = 600000;
gateParam.admin_control_list[1].gate_state_value = 0x55;
gateParam.admin_control_list[2].operation_name = TSN_ieee802_dot1q_sched_set_and_release_mac;
gateParam.admin_control_list[2].time_interval_value = 100000;
gateParam.admin_control_list[2].gate_state_value = 0x80;
gateParam.admin_control_list[3].operation_name = TSN_ieee802_dot1q_sched_set_and_release_mac;
gateParam.admin_control_list[3].time_interval_value = 0x00;
gateParam.admin_control_list[3].gate_state_value = 0;
gateParam.admin_cycle_time.numerator = 1;
gateParam.admin_cycle_time.denominator = 1000;
gateParam.admin_cycle_time_extension = 0;
gateParam.admin_base_time.seconds = 0;
gateParam.admin_base_time.nanoseconds = 0;

rv = SES_QbvSetGateParameters(portNumber, &gateParam);
```

SCHEDULED TRAFFIC ERROR CODES

The following error codes are defined in SES_codes.h. They are related to scheduled traffic APIs and provide information in event the applied schedule was not successful.

- SES_SCHEDULE_TOO_SHORT = -300

SCHEDULED TRAFFIC EXAMPLES

- SES_SCHEDULE_TOO_LONG = -301
- SES_CYCLE_EXTENSION_TOO_LONG = -302
- SES_CYCLE_TIME_NOT_PRODUCIBLE = -303
- SES_GUARD_BAND_INSERTION_UNSUCCESSFUL = -304
- SES_INVALID_BASETIME = -305
- SES_INVALID_CYCLETIME = -306

RUNNING A SCHEDULE ON THE HARDWARE TIMER PINS (TIMER 0-3)

It is possible to run a schedule on all four hardware Timer pins of the device by calling the above API ("QbvSetGateParameters") with port index as 6. For the Timer pins, the gate scheduling operation used must be "TSN_ieee802_dot1q_sched_set_gate_states" as the other operations are related to the Preemption feature and do not apply to the Timer pins. The timer pin configuration state must be enabled and the timer mode configuration set to "SES_timerModeTsnOut" using the "SES_SetGpioTimerConfig".

When using SPI interface as host communication port between the host and Switch, Timer0 pin is used as the INT (interrupt) pin to the host and not available for timer functions.

The below example will set a schedule on timer pins:

```
int32_t rv = SES_OK;
SES_gpioTimerConfig_t config;
config.gpioConfig.state = 1;
config.timerMode = SES_timerModeTsnOut;

rv = SES_SetGpioTimerConfig(SES_gpio4, &config);
rv = SES_SetGpioTimerConfig(SES_gpio5, &config);
rv = SES_SetGpioTimerConfig(SES_gpio7, &config);

TSN_ieee802_dot1q_types_port_number_t portNumber = 6; // Timer pins are configured through Port 6
TSN_ieee802_dot1q_sched_gate_parameters_t gateParam;
gateParam.gate_enabled = true;
gateParam.config_change = true;
gateParam.guard_band_gate_event = false;
gateParam.guard_band_hold_event = false;
gateParam.admin_gate_states = 255;
gateParam.admin_control_list[0].operation_name = TSN_ieee802_dot1q_sched_set_gate_states;
gateParam.admin_control_list[0].time_interval_value = 100000;
gateParam.admin_control_list[0].gate_state_value = 0x01;
gateParam.admin_control_list[1].operation_name = TSN_ieee802_dot1q_sched_set_gate_states;
gateParam.admin_control_list[1].time_interval_value = 100000;
gateParam.admin_control_list[1].gate_state_value = 0x02;
gateParam.admin_control_list[2].operation_name = TSN_ieee802_dot1q_sched_set_gate_states;
gateParam.admin_control_list[2].time_interval_value = 100000;
gateParam.admin_control_list[2].gate_state_value = 0x03;
gateParam.admin_control_list[3].operation_name = TSN_ieee802_dot1q_sched_set_gate_states;
gateParam.admin_control_list[3].time_interval_value = 100000;
gateParam.admin_control_list[3].gate_state_value = 0x08;
gateParam.admin_control_list[4].operation_name = TSN_ieee802_dot1q_sched_set_gate_states;
gateParam.admin_control_list[4].time_interval_value = 100000;
gateParam.admin_control_list[4].gate_state_value = 0x09;
gateParam.admin_control_list[5].operation_name = TSN_ieee802_dot1q_sched_set_gate_states;
gateParam.admin_control_list[5].time_interval_value = 100000;
gateParam.admin_control_list[5].gate_state_value = 0x0A;
gateParam.admin_control_list[6].operation_name = TSN_ieee802_dot1q_sched_set_gate_states;
gateParam.admin_control_list[6].time_interval_value = 100000;
```

SCHEDULED TRAFFIC EXAMPLES

```
gateParam.admin_control_list[6].gate_state_value = 0x0B;
gateParam.admin_control_list[7].operation_name = TSN_ieee802_dot1q_sched_set_gate_states;
gateParam.admin_control_list[7].time_interval_value = 0;
gateParam.admin_control_list[7].gate_state_value = 0x00;
gateParam.admin_cycle_time.numerator = 1;
gateParam.admin_cycle_time.denominator = 1000;
gateParam.admin_cycle_time_extension = 0;
gateParam.admin_base_time.seconds = 0;
gateParam.admin_base_time.nanoseconds = 0;

rv = SES_QbvSetGateParameters(portNumber, &gateParam);
```

A schedule can also be passed as an array as shown in the example below:

```
int32_t rv = SES_OK;
TSN_ieee802_dot1q_types_port_number_t portNumber = 6; // Timer pins are configured through Port 6
TSN_ieee802_dot1q_sched_gate_parameters_t gateParam = {
[0] = {
    .gate_enabled =    true, .config_change = true,
        .admin_gate_states = 0xFF,
    .admin_control_list = {
        [0]  = {.operation_name = TSN_ieee802_dot1q_sched_set_gate_states,
 .time_interval_value = 100000, .gate_state_value = 0x01 }, // Timer0 activated for 100us
        [1]  = {.operation_name = TSN_ieee802_dot1q_sched_set_gate_states,
 .time_interval_value = 100000, .gate_state_value = 0x02 }, // Timer1 activated for 100us
        [2]  = {.operation_name = TSN_ieee802_dot1q_sched_set_gate_states,
 .time_interval_value = 100000, .gate_state_value = 0x03 }, // Timer0 & 1 activated for 100us
        [3]  = {.operation_name = TSN_ieee802_dot1q_sched_set_gate_states,
 .time_interval_value = 100000, .gate_state_value = 0x08 },// Timer3 activated for 100us
        [4]  = {.operation_name =  TSN_ieee802_dot1q_sched_set_gate_states,
 .time_interval_value = 100000, .gate_state_value = 0x09 },// Timer0 & 3 activated for 100us
.
.
.
        [31] = {.operation_name = TSN_ieee802_dot1q_sched_set_gate_states,
 .time_interval_value = 0, .gate_state_value = 0x00 }
        },
        .admin_cycle_time = { .numerator = 1, .denominator = 1000 },
        .admin_cycle_time_extension = 0,
        .admin_base_time = { .seconds = 0, .nanoseconds = 0 }
    }
};
rv = SES_QbvSetGateParameters(portNumber, &gateParam_p);
```

SEAMLESS SCHEDULE SWITCHOVER USING BASE TIME & CYCLE TIME EXTENSION

Seamless switchover from an existing schedule to a new schedule is handled by using Base time and Cycle time extension capabilities. The programmed base time value is the absolute time at which a new schedule is required to take effect. Cycle Time Extension value defines the maximum amount of time by which the old cycle for the port is permitted to be extended when switching to a new schedule. When changing from an old schedule to a new schedule, without cycle time extension, the new cycle could result in a partial or runt cycle of the old schedule directly before the transition to the new cycle. Using the Cycle Time Extension ensures a more seamless transition between schedules. Now instead of a partial old schedule, the last valid cycle is extended with the old schedule gate states being retained until the new schedule is implemented at the programmed base time, thereby bridging the switchover between the two schedules. The following example shows a

SCHEDULED TRAFFIC EXAMPLES

schedule change at a time of 50s with a 500µs cycle time extension. Consideration needs to be given to the old schedule parameters (cycle time and base time) and the new schedule parameters to ensure sufficient cycle time extension.

```
TSN_ieee802_dot1q_types_port_number_t portNumber = 4;
TSN_ieee802_dot1q_sched_gate_parameters_t gateParam;
gateParam.gate_enabled = true;
gateParam.config_change = true;
gateParam.guard_band_gate_event = false;
gateParam.guard_band_hold_event = false;
gateParam.admin_gate_states = 255;
gateParam.admin_control_list[0].operation_name = TSN_ieee802_dot1q_sched_set_and_release_mac;
gateParam.admin_control_list[0].time_interval_value = 300000;
gateParam.admin_control_list[0].gate_state_value = 0xAA;
gateParam.admin_control_list[1].operation_name = TSN_ieee802_dot1q_sched_set_and_release_mac;
gateParam.admin_control_list[1].time_interval_value = 600000;
gateParam.admin_control_list[1].gate_state_value = 0x55;
gateParam.admin_control_list[2].operation_name = TSN_ieee802_dot1q_sched_set_and_release_mac;
gateParam.admin_control_list[2].time_interval_value = 100000;
gateParam.admin_control_list[2].gate_state_value = 0x80;
gateParam.admin_control_list[3].operation_name = TSN_ieee802_dot1q_sched_set_and_release_mac;
gateParam.admin_control_list[3].time_interval_value = 0;
gateParam.admin_control_list[3].gate_state_value = 0x00;
gateParam.admin_cycle_time.numerator = 1;
gateParam.admin_cycle_time.denominator = 1000;
gateParam.admin_cycle_time_extension = 500000;
gateParam.admin_base_time.seconds = 50;
gateParam.admin_base_time.nanoseconds = 0;

rv = SES_QbvSetGateParameters(portNumber, &gateParam);
```

GUARD BAND EXAMPLE

Guard bands are used with scheduled traffic to protect transmission of the schedule gate open times. An Ethernet port that has started transmission of a frame must complete transmitting that frame before another transmission takes place. In the event a new frame transmission starts just at the end of the first cycle, but the size of the frame is too large to complete before the second cycle is due to begin, this will result in a delayed start of the second cycle. This may result in the lower priority traffic infringing on the start of time critical time slice. As a consequence real-time frames could get delayed impacting the application requirements. Scheduled traffic can use guard bands in front of every time slice that carries time critical traffic. During the guardband duration, only in-flight transmissions can complete, no new Ethernet transmissions can be started. The duration of the guard band equates to the time taken for the maximum frame size to safely transmit. When the “Guard band Gate Event” is enabled, the switch automatically inserts a guard band between the step that has the gate open for a traffic class and the step that has the gate closed. The length of the guard band is the product of the Maximum Service Data Unit (Max SDU) value of the queue associated with the gate and the current link speed. The guard band time value is subtracted from the gate close time. This ensures that the start of the time slots do not get delayed. As the different queues can have different MaxSDU values, the guard bands for the different queues will be calculated. Different MaxSDU values do consume entries in the internal Gate control list. In the event the automatic guard band insertion fails, the driver package reports a return error, see Scheduled Traffic Error Codes. Exotic schedules with many different time slots and different MaxSDU values could result in failure of the insertion of guard bands. However, the driver will prompt in this event and user can review their schedule and revise accordingly.

The guard_band_hold_event is relevant where Scheduled traffic and Frame pre-emption co-exist and the operation selected is “TSN_ieee802_dot1q_sched_set_and_hold_mac” in the gate control list for a particular time interval is enabled. When “Guard Band Hold Event” is enabled, a guard band of holdAdvance length will be inserted between the step that has the Hold_en signal asserted and the previous step and the releaseAdvance length will be inserted between the steps where the hold_EN signal transitions from asserted to deasserted.

The following example shows a schedule on Port 4 with guard bands for gate events and hold events enabled.

SCHEDULED TRAFFIC EXAMPLES

The switch MaxSDU setting defaults to 10,000 bytes, therefore, when interfacing directly to the driver from own host, ensure to configure the MaxSDU values required. This avoids having excessive guardbands. See Queue MaxSDU Example example for details on programming nominal MaxSDU values per port and per queue.

```
TSN_ieee802_dot1q_types_port_number_t portNumber = 4;
TSN_ieee802_dot1q_sched_gate_parameters_t gateParam;
gateParam.gate_enabled = true;
gateParam.config_change = true;
gateParam.guard_band_gate_event = true;
gateParam.guard_band_hold_event = true;
gateParam.admin_gate_states = 255;
gateParam.admin_control_list[0].operation_name = TSN_ieee802_dot1q_sched_set_and_release_mac;
gateParam.admin_control_list[0].time_interval_value = 300000;
gateParam.admin_control_list[0].gate_state_value = 0xAA;
gateParam.admin_control_list[1].operation_name = TSN_ieee802_dot1q_sched_set_and_release_mac;
gateParam.admin_control_list[1].time_interval_value = 600000;
gateParam.admin_control_list[1].gate_state_value = 0x55;
gateParam.admin_control_list[2].operation_name = TSN_ieee802_dot1q_sched_set_and_release_mac;
gateParam.admin_control_list[2].time_interval_value = 100000;
gateParam.admin_control_list[2].gate_state_value = 0x80;
gateParam.admin_control_list[3].operation_name = TSN_ieee802_dot1q_sched_set_and_release_mac;
gateParam.admin_control_list[3].time_interval_value = 0;
gateParam.admin_control_list[3].gate_state_value = 0x00;

gateParam.admin_cycle_time.numerator = 1;
gateParam.admin_cycle_time.denominator = 1000;
gateParam.admin_cycle_time_extension = 0;
gateParam.admin_base_time.seconds = 0;
gateParam.admin_base_time.nanoseconds = 0;

rv = SES_QbvSetGateParameters(portNumber, &gateParam);
```

QUEUE MAXSDU EXAMPLE

The SES_QbvSetQueueMaxSduTable() API provides user ability to adjust the SDU size of the frames that are allowed to egress per queue per port. The hardware default is 10,000 bytes, therefore, it's likely user will want to load smaller byte size during configuration. QueueMaxSDU definition does not include the MAC addresses and FCS, therefore equals (Frame Size – 16 bytes). Adjusting the SDU size allows the user to fine tune the timing of the scheduled traffic as no new traffic will be allowed to start to egress during the guard band time.

A value of 1520 would provide at 12.3 µs guardband at Gbps link speed. The example below configures different MaxSDU values for each queue for Port 1. When the switch is cutting through frames, it will not have visibility into the frame size, therefore the Queue MaxSDU values only apply when switch is operating in Store and Forward mode. See SES_SetStoreAndForwardMask() API to configure store and forward operation.

```
//Example of configuring The Max SDU settings on Port 1 for all queues

int32_t rv = SES_OK;
TSN_ieee802_dot1q_types_port_number_t portNumber = 1;
uint32_t numEntries = 8;
TSN_ieee802_dot1q_sched_queue_max_sdu_table_t queueMaxSduTable = {1520, 1520, 1520, 1520, 1520, 1520,
1520, 1520};

rv = SES_QbvSetQueueMaxSduTable(portNumber, &queueMaxSduTable);
```

SCHEDULED TRAFFIC EXAMPLES

TRANSMISSION OVERRUN/CONFIGURATION CHANGE ERROR EXAMPLE

Scheduled traffic provides statistics around whether there has been a configuration change error or if frames have overrun into the next timeslot. This information can be read back using the SES_QbvGetStats() API.

```
int32_t rv = SES_OK;
TSN_ieee802_dot1q_types_port_number_t portNumber = 1;
SES_qbvStats_t qbvStats_p;
rv = SES_QbvGetStats(portNumber, &qbvStats_p);
```

EVENT API EXAMPLE

The APIs related to Events are available to view in SES_event.h. Events generated on the switch can be propagated to the host. The host subscribes to events of interest using the SES_SubscribeEvent() API. The following example shows how to subscribe to link up and down events. When such events occur, the callback provides information on the event type and user can process and handle accordingly.

There are currently a number of events available detailed in the parameter SES_eventId_t (e.g. LLDP changes, port link state change, network synchronization change).

When the switch is configured for PRP or HSR mode, the normal learning of the forwarding table is disabled, instead the table stores entries based on the HSR/PRP devices/traffic , the event SES_dynTblLimitExceedEvent indicates that the number of entries exceeds 1792 or 87.5% of the table entries.

```
//Example of subscribing to events
static int32_t test_SES_SubscribeEvents(void) {
int32_t rv = SES_OK;
SES_eventId_t  eventId;
// Subscribe to Link Up Event
eventId = SES_linkUpEvent;
rv = SES_SubscribeEvent(eventId, SES_eventCb);

// Subscribe to Link Down Event
eventId = SES_linkDownEvent;
rv = SES_SubscribeEvent(eventId, SES_eventCb);
//…etc
//Note the rv values returned have different values for the different events
}
//Process call back
static int32_t SES_eventCb(int32_t eventId, void* param_p)
{
    switch (eventId)
    {
    case 0:
        //Handle Config Complete event
        break;
    case 1:
        //Handle Link Up  event
        break;
    case 2:
        //Handle Link Down event
        break;
    case 3:
        //Handle MAC Address updated event
        break;
    case 4:
        //Handle Port Config event
        break;
    case 5:
        //Handle Network Sync event
        break;
    case 6:
        //Handle Network Sync Ready event
                break;
    case 7:
        //Handle LLDP New Neighbour Event
                break;
    case 8:
```

EVENT API EXAMPLE

```
        //Handle LLDP Neighbour shutdown Event
                break;
    case 9:
        //Handle LLDP Something changed remotely
                break;
    }
}
```

USING TIMER3 TO TRIGGER CAPTURE OF TIMESTAMPS

Timer2 and Timer3 can be configured as Capture inputs. The example below shows how to configure Timer3 to be a capture input, so either a rising or falling edge can trigger the switch hardware to capture a timestamp. When triggered, there are three timestamps captured and returned to the callback function which are available to read by the host.

```
//Timer Configuration

void SES_SetGpioTimerConfigTest() {

    SES_gpioTimerSignal_t signalTimer3 = SES_gpio7; //gpio7 is Timer3
    SES_gpioTimerConfig_t config_p;

    if(SES_PORT_OK == SES_GetGpioTimerConfig(signalTimer3, &config_p, sizeof(config_p))) {
        config_p.gpioConfig.state = SES_gpioTimerEnabled;
        config_p.gpioConfig.direction = 1;
        config_p.timerMode = SES_timerModeCaptureIn;
        config_p.captureEdge = 1; //capture edge is rising edge when set to 1
    }
    printf("SetGpioTimerConfigTimer3 :: %d\n", SES_SetGpioTimerConfig(signalTimer3, &config_p));

}

//Subscribe to Input Capture Event

void SES_timer_Event_test() {
    SES_eventId_t eventId = SES_inputCaptureEvent;

    if (0 <= SES_SubscribeEvent(eventId, SES_Timer_eventCb)) {
        printf("Timer Event Subscribed success!!");
    }

    while (1) {

    }

}
//Event Callback

int32_t SES_Timer_eventCb(sesID_t sesId, int32_t eventID, void* param_p) {
    printf("Event ID :: %d\n", eventID);

    SES_inputCaptureEventParam_t* captureParam = (SES_inputCaptureEventParam_t*)param_p;
    printf("interfaceIdx - %d\n", captureParam->interfaceIdx);
    printf("capture - %d\n", captureParam->capture);
    printf("freeTimestamp - %llu\n", captureParam->freeTimestamp);
```

EVENT API EXAMPLE

```
    printf("synt0Timestamp - %llu\n", captureParam->synt0Timestamp);
    printf("synt1Timestamp - %llu\n", captureParam->synt1Timestamp);
}
```

LOGICAL MAC EXAMPLE

Logical MAC APIs are included in the 'SES_logical_mac.h' file The logical MAC feature provides ability to logically partition the physical MAC ports into different logical MAC groups with a specific link port and one or more network ports. The feature can add tag information at the end of the frames sent to the link port thereby allowing the stack processor to distinguish which network port the frames originated from. From the stack processor side, it can control which port or ports a frame that it originates is transmitted out of. The switch can support changing the source MAC address of the egressing packets from the switch network ports. The host can register callbacks for frames received on the desired logical MAC with the tag information indicating which port it originated from Table 2.

The following example details a configuration where there is one logical MAC group, with 3 ports. The group contains Ports 0, 1, 2, where the link port is Port 0 and Ports 1 & 2 are network ports.

After configuring this logical MAC group, traffic from the network ports targeted at the Logical MAC address (to the host/Link Port) will have a 6-byte "tail tag" added at the end of the frame right before the FCS. To tag other traffic intended for the host port, add a static table entry as discussed in Tagging All Traffic up to the Host, Adding Static Entry With TX Transform. The values of interest in this tag are the upper three bits (bits 13-15) which indicate the SES port the traffic originated from. The tail tag provided allows user to identify any of the six ports as shown in Table 2. An example of a tag added into the frame traffic originating from Port 2 aimed at the logical MAC address would have "6x xx 00 30 88 fb" added at end of the frame. The "6" indicating from Port 2. As this feature repurposes the PRP RCT trailer, the 0x88fb at the end is the PRP suffix. Other bit fields within the tag will vary depending on frame size and sequence counter, the sequence counter gets reset every 5ms to ensure it doesn't overwrite the port ID information.

NOTE: The Logical MAC tail tag uses the PRP RCT trailer. It is not intended to be used with PRP.

Table 2. Tail Tag Port Identification

Port Number	Tail Tag Value	Top 3 bits (Bits 15-13)
Port0	0x2xxx/0x3xxx	001
Port1	0x4xxx/0x5xxx	010
Port2	0x6xxx/0x7xxx	011
port3	0x8xxx/0x9xxx	100
port4	0xAxxx/0xBxxx	101
Port5	0xCxxx/0xDxxx	110

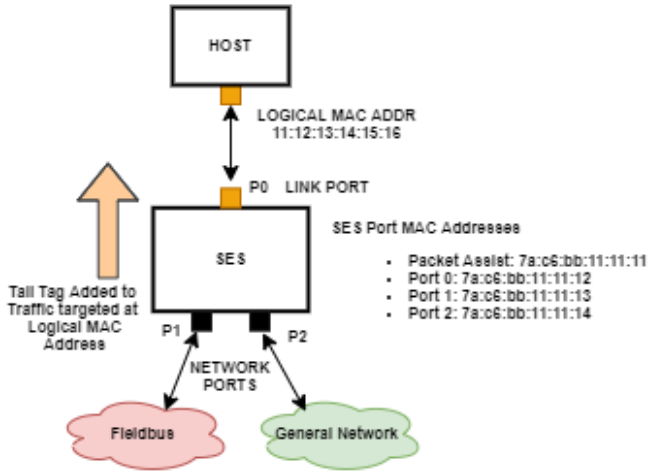

Figure 18. Example of a Logical MAC Group with 3 ports

Creating a logical mac instance automatically installs the required entries in the lookup table to route traffic destined to the logical MAC address. In addition, when a frame is transmitted with a source MAC address that matches a port MAC address, the switch automatically forwards the frame out of the corresponding port. This behavior is intended for use cases where the host is running a protocol stack such as PTP or LLDP and is transmitting the port-specific messages.

Other traffic will not be forwarded automatically to the host/link port, therefore, for other traffic of interest to the host, static entries must be installed in the forwarding table using the SES_AddStaticTableEntry() API. When creating a logical mac grouping, the API takes care of creating a port forwarding mask between the logical mac ports to ensure traffic is constrained to the specific ports of the group. The user doesn't need to do any additional configuration of the port forwarding.

LOGICAL MAC EXAMPLE

Network ports cannot be part of two different groups, but link ports can belong to different groups.

```
/* Example of using the Logical MAC feature for group of three ports, 0, 1, 2 with 0 as Link Port/Host
*/

int32_t rv = SES_OK;
uint8_t portMap;
int8_t LinkPort;
uint8_t macAddress_g[6] = { 0x11,0x12,0x13,0x14,0x15,0x16 }; //Logical MAC Address
SES_logicalMacConfig_t logicalMacConfig;
SES_logicalMacReturns_t logicalMacReturns;
memcpy(&logicalMacConfig.logicalMac, macAddress_g, 6);
logicalMacConfig.portMap = 0x7; //Port 0, 1 and 2 are part of this logical MAC group
logicalMacConfig.deviceLinkPort = 0; //Port 0 is the Link Port
logicalMacConfig.rxLogicalMacCallback = SES_rxlogicalmaccb;
rv = SES_AddLogicalMac(&logicalMacConfig, &logicalMacReturns);
```

The following example details a configuration where there are two logical MAC groups, each with 2 ports. In this scenario, the Host has two MAC addresses associated with it. The first group is Ports 0, 1 where the link port is Port 0 and Port 1 is the network port, while the second group has Link port 0 and network port of 2. As discussed above, link ports can belong to more than one group, but network ports can only belong to one group instance.

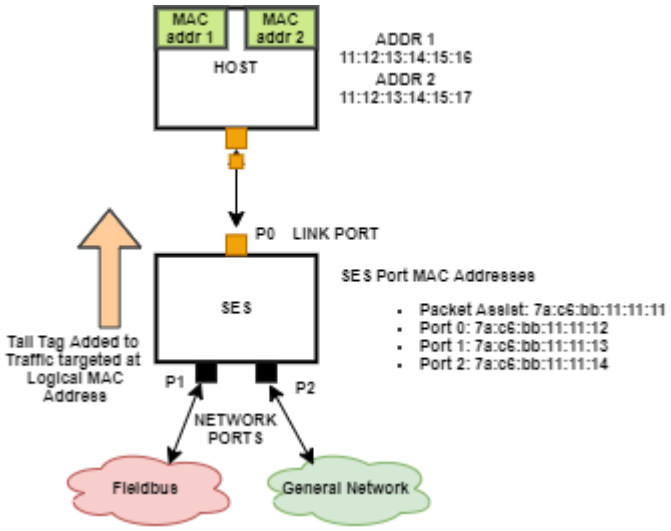

Figure 19. Example of a Logical MAC Group where the Host MAC has two MAC Addresses, so two Logical MAC groups are required.

```
//Example of using the Logical MAC feature for a host with two MAC addresses, so two logical MAC groups

int32_t rv = SES_OK;
uint8_t portMap;
int8_t LinkPort;
uint8_t macAddress_g[6] = { 0x11,0x12,0x13,0x14,0x15,0x16 }; //logical MAC address for first group
uint8_t macAddress_g2[6] = { 0x11,0x12,0x13,0x14,0x15,0x17 };//logical MAC address for second group
SES_logicalMacConfig_t logicalMacConfig, logicalMacConfig2;
SES_logicalMacReturns_t logicalMacReturns, logicalMacReturns2;

//Configure first logical mac group
memcpy(&logicalMacConfig.logicalMac, macAddress_g, 6);
```

LOGICAL MAC EXAMPLE

```
logicalMacConfig.portMap = 0x3; //Port 0 and 1 are part of this logical MAC group
logicalMacConfig.deviceLinkPort = 0; //Port 0 is device link port
logicalMacConfig.rxLogicalMacCallback = SES_rxlogicalmaccb; //Call back function
rv = SES_AddLogicalMac(&logicalMacConfig, &logicalMacReturns); //Logical MAC instance index returned

//Configure second logical mac group
memcpy(&logicalMacConfig2.logicalMac, macAddress_g2, 6);
logicalMacConfig2.portMap = 0x5; //Port 0 and 2 are part of this logical MAC group
logicalMacConfig2.deviceLinkPort = 0; //Port 0 is device link port
logicalMacConfig2.rxLogicalMacCallback = func1; //Call back function
rv = SES_AddLogicalMac(&logicalMacConfig2, &logicalMacReturns2); //Logical MAC instance index returned
```

The following example details a configuration where there are two logical MAC groups, each with 3 ports. The first group is Ports 0, 1, 2, where the link port is Port 0 and Ports 1 & 2 are network ports, while the second group has Link port 3 and network ports 4 & 5. There's only one host that configures/controls the SES device in this scenario.

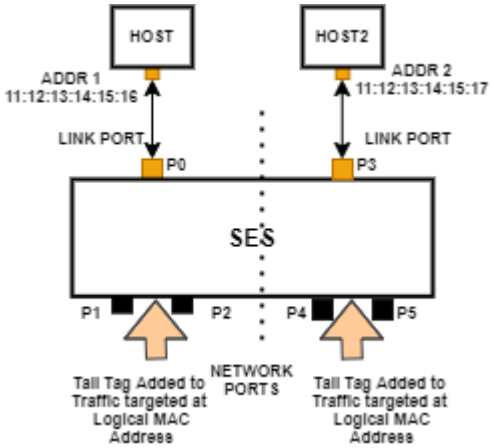

Figure 20. Example of a Logical MAC Group where the switch is partitioned into two Logical MAC groups

```
//Example of using the Logical MAC feature for 2 x 3 port logical groups
int32_t rv = SES_OK;
uint8_t portMap;
int8_t LinkPort;
uint8_t macAddress_g[6] = { 0x11,0x12,0x13,0x14,0x15,0x16 }; //logical MAC address for first group
uint8_t macAddress_g2[6] = { 0x11,0x12,0x13,0x14,0x15,0x17 };//logical MAC address for second group
SES_logicalMacConfig_t logicalMacConfig, logicalMacConfig2;
SES_logicalMacReturns_t logicalMacReturns, logicalMacReturns2;

//Configure first logical mac group
memcpy(&logicalMacConfig.logicalMac, macAddress_g, 6);
logicalMacConfig.portMap = 0x7; //Port 0,  1 and 2 are part of this logical MAC group
logicalMacConfig.deviceLinkPort = 0; //Port 0 is device link port
logicalMacConfig.rxLogicalMacCallback = SES_rxlogicalmaccb;
rv = SES_AddLogicalMac(&logicalMacConfig, &logicalMacReturns); //Logical MAC instance index returned

//Configure second logical mac group
memcpy(&logicalMacConfig2.logicalMac, macAddress_g2, 6);
logicalMacConfig2.portMap = 0x38; //Port 3,4 and 5 are part of this logical MAC group
logicalMacConfig2.deviceLinkPort = 3; //Port 3 is device link port
```

LOGICAL MAC EXAMPLE

```
logicalMacConfig2.rxLogicalMacCallback = func1;
rv = SES_AddLogicalMac(&logicalMacConfig2, &logicalMacReturns2); //Logical MAC instance index returned
```

TAGGING ALL TRAFFIC UP TO THE HOST, ADDING STATIC ENTRY WITH TX TRANSFORM

When a Logical MAC group is configured, the switch automatically installs a static entry for the Logical MAC address with a transmit transform that appends a tail tag.

To apply tail tagging to additional traffic (for example, traffic destined for the host), an additional static table entry must be added using the logical MAC returns (SES_logicalMacReturns_t logicalMacReturns).

Configure the entry as follows:

- Set rxSequenceSet to logicalMacReturns.sgaIndex
- Set transmitFilter to logicalMacReturns.txPrpXformIndex
- Set sequenceMgmt to dynamicTblSequenceGeneration'

```
int32_t seslogicalMAC_applying_tail_tag_Example(void)
{
  int32_t rv = SES_OK;
  SES_logicalMacConfig_t logicalMacConfig;
  SES_logicalMacReturns_t logicalMacReturns;

  uint8_t macAddress_g[6] = { 0x00, 0x00, 0x00, 0x11, 0x11, 0x11 };

  memcpy(&logicalMacConfig.logicalMac, macAddress_g, 6);
  logicalMacConfig.portMap = 0x01 | 0x02; // Port 0 and Port 1
  logicalMacConfig.deviceLinkPort = 0; // Link Port - Port 0
  logicalMacConfig.rxLogicalMacCallback = NULL;
  rv = SES_AddLogicalMac(&logicalMacConfig, &logicalMacReturns);

  uint8_t macAddress[6] = { 0x00, 0x00, 0x00, 0x22, 0x22, 0x22 };
  int32_t index;

  /* Static entry to add tail tag to other traffic to host port */
  rv = SES_AddStaticTableEntryEx(
    0, // priority
    macAddress, // MACaddr_p
    NULL, // MACmask_p
    0x0FFF, // vlanID
    0, // vlanMask
    0, // sourceEntry
    0, // override
    0, // sourceOverride
    0, // hsrOrPrpSupervisory
    SES_dynamicTblLookupBasic, // lookupType
    0x01, // portMap
    0, // sendToAE
    0, // danp
    0, // cutThrough
    0, // syntTimestamp
    0, // localTimestamp
    dynamicTblSequenceGeneration, // sequenceMgmt
    logicalMacReturns.sgaIndex, // rxSequenceSet
    logicalMacReturns.txPrpXformIndex, // transmitFilter
    SES_IGNORE_FIELD, // receiveFilter
    &index);
```

LOGICAL MAC EXAMPLE

```
  return rv;
}
```

CONFIGURING LOGICAL MAC FRAME (REPLACE SOURCE MAC)

The SES_ConfigureLogicalMacFrame() API allows user to enable the automatic source MAC address replacement for specified frames transmitted out a network port. User can configure the source MAC addresses that they want replaced with a choice of replacing with the logical mac address or with the egressing port MAC address.

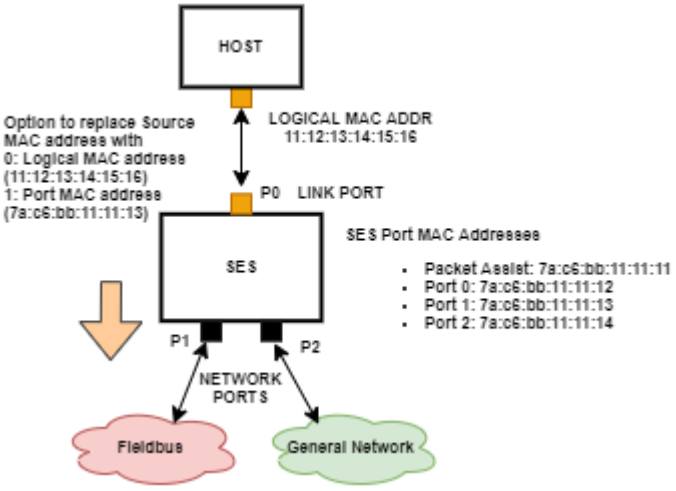

Figure 21. Example where Source MAC address gets replaced on Port 1.

In the example below, any frames ingressing Port 0 with source MAC address 11:11:11:11:11:55 destined to egress on Port 1 will have the source MAC updated to the port MAC address 7a:c6:bb:11:11:13.

If useDeviceMac was set to 1, then the frames would egress with logical MAC address which was set when the logical MAC configuration was initially set up, e.g. based on previous example, this would be 11:12:13:14:15:16

```
/* Example of enabling automatic source MAC address replacement for frames transmitted out Port 1 */

int32_t rv = SES_OK;
uint8_t portNum = 1; // Port number 1
uint8_t replaceaddr[6] = { 0x11, 0x11, 0x11, 0x11, 0x11, 0x55 }; // MAC address for frames which
should have Source MAC replaced
uint8_t useDeviceMac = 0;  // 1 :frames will send out with logical mac address
                           // 0 : frames will send out with port mac address

rv = SES_ConfigureLogicalMacFrame(replaceaddr, portNum, useDeviceMac);
```

REMOVE A LOGICAL MAC GROUP

Removing a logical MAC group is done by calling the SES_DeleteLogicalMac() API with the index and mac Address to remove. When a logical MAC group is removed, the port tail tagging will no longer be applied.

```
//Example of removing a Logical MAC

int32_t rv = SES_OK;
uint16_t index = 0; //Index of group
```

LOGICAL MAC EXAMPLE

```
uint8_t macAddress_g[6] = { 0x11,0x12,0x13,0x14,0x15,0x16 }; //logical MAC address for group

rv = SES_DeleteLogicalMac(index, macAddress_g);
```

PORT FORWARDING MASK

The port masking feature can be used independently of the Logical MAC function. The switch supports applying a forwarding mask per receive port, which can be used to restrict or control the set of ports to which traffic may be forwarded.

When configured, the forwarding mask limits traffic received on a given port to be forwarded only to the ports specified in the mask.

In the example below, frames received on Port 1 are forwarded only to Ports 2 and 3. Additional forwarding masks can be applied to other receive ports by making repeated calls to the SES_ApplyForwardMask() API, as required.

```
int32_t sesTestApplyForwardMask_Example(void) {
    int32_t rv = 0;
    SES_mac_t portNum = SES_macPort1;
    uint8_t portMap = 0x0C;
    rv = SES_ApplyForwardMask(portNum, portMap);
    return rv;
}
```

LAYER 2 TX/RX EXAMPLE

These APIs are included in the 'SES_frame.h' file. The purpose of these APIs is to support layer 2 stacks (LLDP and gPTP) to be run on the host/stack processor instead of on the assist engine. This functionality is supported for both Ethernet and SPI interfaced host.

RECEIVE EXAMPLES

The Layer 2 receive API returns the requested frame and associated data through a callback function registered by the user. Requested frames are identified based on the destination MAC address (primary priority) or Ethertype (secondary priority). Frames can be received either locally by the stack processor or remotely by SES.

The user can request that remotely received frames, those received at an SES port, include ingress timestamps or port data. In addition, a static table entry must also be installed with sendToAE=1 for the MAC address of interest.

```
/* Example of Static Entry to be Installed for layer 2 receive API */
void sesL2ReceiveFrameStaticEntry_Example() {
    int32_t rv = SES_OK;
        /*Attribute of interest*/
        uint32_t portAttribute = SES_REQUEST_PORT_ATTRIBUTE;
    /*Frames requested with this MAC address*/
    uint8_t macAddr[6] = { 0x00, 0x00, 0x00, 0x11, 0x11, 0x11 };
    /*low priority = 0, entry will be placed towards bottom of table*/
    uint8_t lookupPriority = 0;
    uint8_t macMask[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
    int16_t vlanId = 0xFFF;
    uint16_t vlanMask = 0;
    uint8_t sourceEntry = 0;
    uint8_t override = 0;
    uint8_t sourceOverride = 0;
    uint8_t hsrOrPrpSupervisory = 0;
    SES_dynTblLookup_t lookupType = SES_dynamicTblLookupBasic;
    uint8_t portMap = 0x00;
    /*Set to 1 to send to Assist Engine*/
    uint8_t sendToAE = 1;
    uint8_t danp = 0;
    /*Store and Forward mode*/
    uint8_t cutThrough = 0;
    /*ADI use only, set to 0*/
    uint8_t syntTimestamp = 0;
    /*ADI use only, set to 0*/
    uint8_t localTimestamp = 0;
    /*SET TO dynamicTblNoSequenceOp*/
    SES_dynTblSeqMgmt_t sequenceMgmt = dynamicTblNoSequenceOp;
    /*NOT CURRENTLY USED, SET TO 0*/
    int rxSequenceSet = 0;
    /* SET TO -1*/
    int transmitFilter = SES_IGNORE_FIELD;
    /* SET TO -1*/
    int receiveFilter = SES_IGNORE_FIELD;
    int index;
    printf( "Adding Static Entry :: %d\n",SES_AddStaticTableEntryEx(lookupPriority,
      &macAddr, &macMask,
      vlanId,
      vlanMask,
      sourceEntry,
      override,
      sourceOverride,
      hsrOrPrpSupervisory,
      lookupType,
```

LAYER 2 TX/RX EXAMPLE

```
      portMap,
      sendToAE,
      danp,
      cutThrough,
      syntTimestamp,
      localTimestamp,
      sequenceMgmt,
      rxSequenceSet,
      transmitFilter,
      receiveFilter,
      &index));
}
```

NOTE: Other API functions must not be called within the callback routine. Callbacks are executed in the received thread, and invoking an API function from a callback that sends a request message and waits for a response may cause the response to be lost. This occurs because the receive thread is already active handling the callback and would be blocked while waiting for an additional response.

```
//Example of requesting frames by DMAC received remotely at SES and requesting port information
//User must also install static table entry with sendToAE=1 and configure ports for VLAN ID of interest

int32_t SES_ExampleRxSesFramesByMac() {
  int32_t rv = SES_OK;
  uint8_t macAddress_g[6] = { 0x00, 0x00, 0x00, 0x11, 0x11, 0x11 };
  uint32_t attributeRequest = SES_REQUEST_FREERUN_TIMESTAMP_ATTRIBUTE;

  /*Static entry is required to send the packets to AE */
  sesL2ReceiveFrameStaticEntry_Example();

  rv = SES_RxSesFramesByMac(macAddress_g, attributeRequest, sesL2RxCallback_Example);
  printf("Requesting L2 Packet Receive by MAC Address :: %d\n", rv);

  return rv;
}

/* Example of callback printing received frames */
void sesL2RxCallback_Example(int32_t frameLength, uint8_t* frame_p, SES_frameAttributes_t* frameAttri▶
butes_p) {
    printf("Packet Received!!! %d\n", frameLength);
    printf("Received Packet :: \n");
    for (int i = 0; i < frameLength; i++) {
        /*Print each element in hexadecimal format*/
        printf("%02X ", frame_p[i]);
    }
    printf("\n");
}
```

When frames with this MAC address and VLAN ID are received, the switch will return the requested frame and data to the host. The frame will be returned as an SMP message from the switch, with Source MAC address of the switch packet assist engine (e.g. 7a c6 bb 11 11 11) and destination MAC the HOST MAC address.

Example (ingressing frame with requested MAC address):

Returned frame and attribute information encapsulated in SMP message from Switch:

00 0a cd 3e 15 d3 **7a c6 bb 11 11 11** aa c5 f0 00 00 00 c3 00 62 80 be 00 00 00 00 00 00 00 00 00

00 00 00 00 06 00 00 00 aa 00 00 00 96 00 00 00 **00 00 00 11 11 11** 00 00 00 00 00 22 **81 00 00 01**

LAYER 2 TX/RX EXAMPLE

12 34 e3 6d 2c 5d 5a be 2b c9 49 78 69 60 a0 00 00 00 10 11 12 13 c2 3c 55 ee 00 64 1a 1b 1c 1d

1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d

3e 3f 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d

5e 5f 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d

2b 7a 53 03 2c 19 01 00 00 00 03 00 00 00 00 00 00 00 00 00 00 00

Where bytes shown in bold above correspond to:

- Source MAC: Switch primary MAC "7a c6 bb 11 11 11"
- MAC of interest: "00 00 00 11 11 11"
- VLAN tag: VID = "1"
- Port information

The second example requests that frames with the specified Ethertype and data received at SES port be returned to the call back function with port attribute information.

```
/* Retrieve the Frames with the specified ethertype and data received at a SES port
* will be returned to the callback function with the requested attribute data.
* NOTE : Maximum of 3 distinct matches can be performed apart from the ethertype
* using the SES_matchData_t
*/
/*In the below example, the ethertype will be matched along with the data1, data2 and data3 post
ethertype data */
int32_t sesRxSesFramesByEtype_Example(void) {
  int32_t rv = SES_OK;

  const uint8_t data1[] = { 0xaa };
  const uint8_t data2[] = { 0xbb };
  const uint8_t data3[] = { 0xcc };

  const uint8_t mask[] = { 0xff };

  uint16_t ethertype = 0x1234;
  uint16_t ethertypeMask = 0xffff;
  uint8_t matchCount = 3;
  SES_matchData_t matchData_p[] =
    { {1, data1, mask, 0}, {1, data2, mask, 1}, {1, data3, mask, 2} };
  uint32_t attributeRequest = SES_REQUEST_PORT_ATTRIBUTE;

  /*Static entry is required to send the packets to AE*/
  sesL2ReceiveFrameStaticEntry_Example();

  rv = SES_RxSesFramesByEtype(ethertype, ethertypeMask, matchCount, matchData_p,
    attributeRequest, sesL2RxCallback_Example);
  printf("L2 Packet Receive by Ethertype along with other matches :: %d\n", rv);

  return rv;
}
```

Returned frame and attribute information encapsulated in SMP message from Switch :

0000 00 0a cd 3e 15 d2 **7a c6 bb 11 11 11** aa c5 f0 000010 00 00 7d 00 62 80 78 00 00 00 00 00 00 00 00 000020 00 00 00 00 04 00 00 00 64 00 00 00 50 00 00 000030 01 00 00 00 03 00 00 00 00 00 00 00 00 00 00 000040 **00 00 00 11 11 11** b4 96 91 8c 11 11 **81 00 00 01**0050 **12 34**

LAYER 2 TX/RX EXAMPLE

00 00**aa aa** aa aa aa aa aa aa 00 00 00 000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 000080 00 00 00 00 00 00 00 00 00 00 00 00 **ba dd f1 200090** 00

Where bytes shown in bold above correspond to:

- Source MAC: Switch primary MAC "7a c6 bb 11 11 11"
- MAC of interest: "00 00 00 11 11 11"
- VLAN tag: VID = "1"
- Ethertype: "12 34"
- Data: "aa aa"
- Timestamp information

Frames with the specified destination MAC address received at the stack processor can also be returned to the callback function. No attribute data is available for frames received at the stack processor.

```
/*Example of requesting frames with the specified DMAC received at the stack processor*/
int32_t sesRxStackProcessorFramesByMac_Example() {
  int32_t rv = SES_OK;
  uint8_t macAddress_g[6] = { 0x00, 0x00, 0x00, 0x22, 0x22, 0x22 };

  rv = SES_RxStackProcessorFramesByMac(macAddress_g, sesL2RxCallback_Example);
  printf("Requesting L2 Packet Receive by MAC :: %d\n", rv);

  return rv;
}
```

Frames with the specified destination MAC address and Ethertype received at the stack processor can also be returned to the callback function. No attribute data is available for frames received at the stack processor.

```
/*Example of requesting frames with the specified ethertype received at the stack processor*/
int32_t sesRxStackProcessorFramesByEtype_Example() {
  int32_t rv = SES_OK;
  /*Frames requested with this Ethertype*/
  uint16_t ethertype = 0x1234;

  rv = SES_RxStackProcessorFramesByEtype(ethertype, sesL2RxCallback_Example);
  printf("SES_RxStackProcessorFramesByEtype rv :: %d\n", rv);

  return rv;
}
```

TRANSMIT EXAMPLE

The L2 transmit API provides a mechanism to send layer 2 frames through SES. The user will be able to request frames return egress timestamps to the user via callback. Additionally, the transmit API supports SES port features for automatically generating the FCS and priority. Finally, the transmit API will support use or override of SES standard frame forwarding.

```
// Example of transmitting a frame out on all ports
int32_t rv = SES_OK;
uint8_t pkt_p[] = { 0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC, 0x22, 0x22, 0x22,
    0x22, 0x22, 0x22, 0xaa, 0xc5, 0x40, 0x64, 0x81, 0x00, 0x8F,
    0xA7, 0x57, 0xF9, 0xFC, 0xF6, 0xF1, 0x1F, 0xFF, 0xD1, 0xA2,
    0xDA, 0x4E, 0xDE, 0xC8, 0x84, 0xC1, 0xF6, 0x9E, 0x1F, 0xD8,
```

LAYER 2 TX/RX EXAMPLE

```
    0xC9, 0x19, 0x53, 0x8E, 0x46, 0x44, 0x7A, 0xE3, 0xFC, 0x62,
    0x08, 0xC9, 0x52, 0xA6, 0x02, 0x2E, 0x89, 0x21, 0xDC, 0x85,
    0x4B, 0x16, 0x70, 0x22, 0xF5, 0x24, 0x7B, 0x4A, 0x29, 0x82,
    0xB7, 0x0F, 0x79, 0x2B, 0xD6, 0x26, 0x1B, 0x9B, 0x92, 0x56,
    0x75, 0xD6, 0x26, 0x24, 0x82, 0x86, 0x06, 0x22, 0x80, 0x25,
    0x53, 0xA5, 0x98, 0x9F, 0xFF, 0x6D, 0x6C};

    SES_transmitFrameData_t txData_p = {
    .frameType = SES_standardFrame,
    .data_p = &pkt_p,
    .byteCount = sizeof(pkt_p),
    .cb_p = 0,
    .cbParam_p = 0,
    .ses = {
        .generateFcs = 1,
        .xmitPriority = 2,
        // .egressPortMap = SES_USE_FORWARDING_TABLE,
        .egressPortMap = 0xFF,
        .transformId = SES_NO_TRANSFORM,
        .attributeRequest = 0
    }
};

rv = SES_XmitFrame( &txData_p );
```

Figure 22 shows a wireshark capture of the transmitted frame as seen on Port 0/Host port.

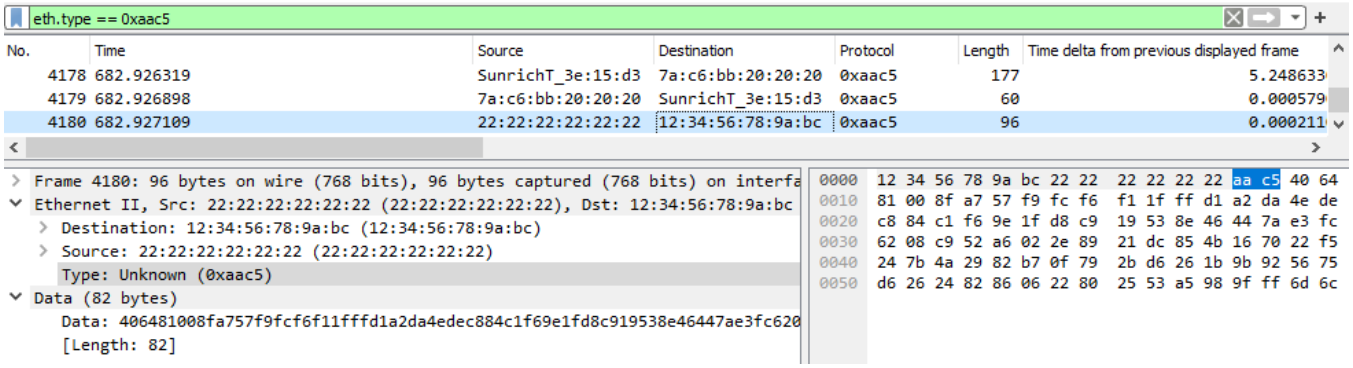

Figure 22. Wireshark capture of trasnmitted frame from SES_XmitFrame() API call in example above.

TIME TUNING

The time tuning APIs directly influence the time synchronization in the switch. They are exposed to customer for use when the gPTP stack is not running on the switch itself, instead gPTP is running on the host processor. The APIs associated with Time tune provide a means for the host to synchronize when using a different mechanism other than the internal GPTP stack. The detail for the APIs associated with this feature are contained in the SES_switch_time.h header.

After the switch is initialized and gPTP is enabled on the host side, the SES_TimeSetTuning() API should be called once to configure how strongly or weakly the clock tuning algorithm on the switch adjusts the system clock in response to calls to SES_TimeTune() API.

The host captures timing information (got from gPTP) and provides the time information to the switch via SES_TimeTune() API, the switch then tunes its clock according to that information. Calls to SES_TimeTune() API are issued on a regular basis, likely after each Sync frame, so typically every 125 ms.

The SES_TimeGetState() API can be used to check if the device is synchronized, returning True if it is synchronized and false otherwise.

```
// Example of synchronizing SES using Time Tune
…
// Complete SES initialization sequence
// Enable Host gPTP stack
int32_t rv = SES_OK;
uint32_t errWindow = 40; // 40ns error window
uint32_t errCount = 750; // Number of times (in a row) that supplied errWindow can be exceeded before
system takes //action and forces synchronisation with SES synthonized timer.
SES_timer_t timer = SES_syntTimerA;
SES_timeSpec_t localTime;
SES_timeSpec_t referenceTime;bool syncd;

localTime.tv_sec = 100;
localTime.tv_nsec = 100;

referenceTime.tv_sec = 100;
referenceTime.tv_nsec = 120;

rv = SES_TimeSetTuning(errWindow, errCount, timer); // Only called once after initialization

// This function should be called repeatedly in order to completely tune the local clock into synchro▶
nization.
rv = SES_TimeTune(&localTime, &referenceTime, timer); // Note the timespec values are 32-bit

rv = SES_TimeGetState(syncd);
```

TIMER, 1PPS SIGNAL

The SES_TimeDisableDiagPulse() and SES_TimeEnableDiagPulse() APIs in the SES_switch_time.h header allow user to enable or disable the 1PPS signal. By default, the Timer2 hardware pin outputs a 1PPS (1 pulse per second) signal.

PRP (PARALLEL REDUNDANCY PROTOCOL)

PRP APIs are included in the 'SES_prp.h' file. The Switch supports PRP, where the initial capability is to support one instance of a DANP function configured over Ethernet connected host. The host configures the Switch for the PRP function, defining which ports are A/B/C, sets the LRE MAC address and enables the PRP function.

The Switch hardware takes care of duplicating the outgoing traffic onto LAN A/B, inserting the RCT tag to the end of the frame, consumes the first frame, removes the tag on reception (if required), discards duplicates, in addition to generating supervisory frames and maintain a nodes table of other entities in the network. The PRP supervisory frames are generated periodically with or without VLAN tag every 2 seconds by default, but the interval can be adjusted (LifeCheckInterval parameter). The hardware records the last time a frame was received from a node, refreshes the software-based nodes table. Node entries are removed from the table based on the NodeForgetTime default of 1 minute.

The node table is currently capable of supporting up to 1024 entries. The proxy node table supports 8 entries.

The current firmware/diver supports the following:

- PRP as a DANP/Redbox
- PRP configured over SPI host
- PRP with LLDP
- PRP with VLAN

And does not support the following:

- PRP with PTP.
- PRP with scheduled traffic
- PRP with frame preemption
- Multiple instances of PRP running on a 6-port switch.

Three Ethernet ports are involved in a PRP DANP device, Port A, Port B are LAN ports connecting to the network. Port C is connected to the Host/End node over Ethernet and is used for control plane configuration of the Switch device and PRP data plane traffic. When configured as a PRP Redbox, there can also be a number of interlink ports that contribute to the PRP network.

NOTE: All switch ports must be enabled during port initialization before PRP initialization.

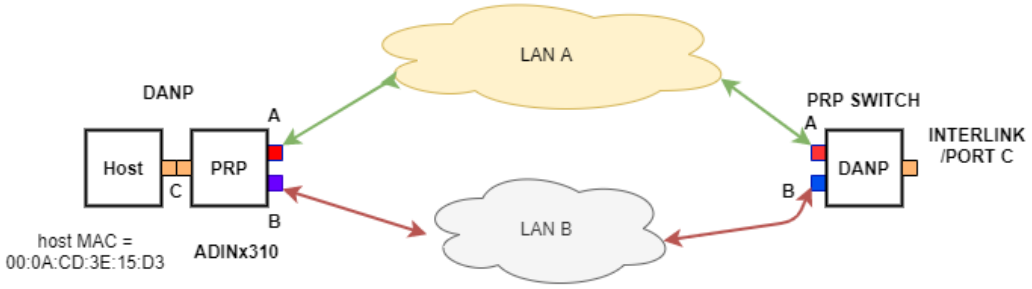

Figure 23. Example of Switch configuration as PRP-DANP (Host connected over Ethernet)

ENABLING PRP AS A DANP EXAMPLE

When PRP is enabled (SES_PrpStart() API) the default mode is the following:

- lrePortAdminStateA - active
- lrePortAdminStateB - active
- lreDuplicateDiscard - duplicate discard
- lreTransparentReception - remove rct
- lreEvaluateSupervision - true
- lreSupervisionVid - No VLAN tag

```
// Example of enabling PRP function with ports 1 & 2 as Ports A & B and Port 0 as host/Port C
```

PRP (PARALLEL REDUNDANCY PROTOCOL)

```
(ethernet host).

int32_t ses_test_prp_start() {
    int32_t result = 0;

    /*SES_PrpStart(bool danpDevice,
        MIB_PRP_HSR_lrePorts_t * lrePorts_p,
        MIB_PRP_HSR_rbInterlinkPorts_t * rbInterlinkPorts_p,
        uint8_t * lreMacAddress_p,
        int32_t lreDupListResideMaxTime,
        MIB_PRP_HSR_statisticOptions_t statisticOptions);*/


    MIB_PRP_HSR_redundancyType_t redundancyType = MIB_PRP_HSR_prpMode1;
    bool danpDevice = true;
    MIB_PRP_HSR_lrePorts_t lrePorts_p = {1, 2, 0};
    MIB_PRP_HSR_rbInterlinkPorts_t rbInterlinkPorts_p = { -1,-1,-1,-1 };
    uint8_t macAddress_g[6] = { 0x00, 0x0A, 0xCD, 0x3E, 0x15, 0xD3 };
    int32_t lreDupListResideMaxTime = 161;

    /* NOTE: statisticOptions is used for HSR */
    result = SES_PrpStart(danpDevice, &lrePorts_p, &rbInterlinkPorts_p,
        &macAddress_g, lreDupListResideMaxTime, MIB_PRP_HSR_fullStatistics);

    return result;
}
```

The other PRP get/set APIs can be called to change PRP configuration only after PRP has been started.

CONFIGURING PRP PORTS AS ACCESS PORTS AND INTERLINK PORTS AS TRUNK

The example demonstrates the use of VLAN configuration along with a PRP Redbox. The examples configures Port A and Port B as an access ports with VLAN ID 15, and the interlink port (Port 5) as a trunk port supporting VLAN ID 10 to 20.

```
int32_t prp_redBoxstart_Vlan_Example() {
    int32_t result = 0;

    bool danpDevice = false;
    MIB_PRP_HSR_lrePorts_t lrePorts_p = { 1, 2, 0 };
    MIB_PRP_HSR_rbInterlinkPorts_t rbInterlinkPorts_p = { 5,-1,-1,-1 };
    uint8_t macAddress_g[6] = { 0xAC, 0x91, 0xA1, 0x91, 0xAC, 0x97 };
    int32_t lreDupListResideMaxTime = 161;

    printf("SES_SetVlanPortType 1 :: %d\n", SES_SetVlanPortType(SES_macPort1, 15, 15, 4, SES_vlanAc▶
cess));
    printf("SES_SetVlanPortType 2 :: %d\n", SES_SetVlanPortType(SES_macPort2, 15, 15, 4, SES_vlanAc▶
cess));
    printf("SES_SetVlanPortType 5 :: %d\n", SES_SetVlanPortType(SES_macPort5, 10, 20, 4, SES_vlan▶
Trunk));

    result = SES_PrpStart(danpDevice, &lrePorts_p, &rbInterlinkPorts_p,
          &macAddress_g, lreDupListResideMaxTime, MIB_PRP_HSR_fullStatistics);

    return result;
}
```

PRP (PARALLEL REDUNDANCY PROTOCOL)

DUPLICATE LIST RESIDE TIME

Consideration needs to be given to the programmed lreDupListResideMaxTime. The switch maintains a list in the forwarding table (max 2k entries) for the duplicate discard algorithm. It stores the sequence number and source MAC for the duration of the programmed duplicate list reside time (lreDupListResideMaxTime).The value programmed indicates the maximum time an entry can reside in the LRE duplicate list, expressed as a number of seconds divided by 65536. The standard defines a default time of 400ms (corresponding to 26214), the programmable range is 1 to 26214 (15 µs to 400 ms). The age out time is based on a calculation using the programmed reside time. The minimum ageout time is 1ms (lreDupListResideMaxTime = 33 to 98).

```
SECOND_FRACTION_NS = 15259;
ONE_HALF_MS = 500000;
ONE_MS = 1000000;
ageOutMs = (uint32_t)(((ResideMaxTime * SECOND_FRACTION_NS) +ONE_HALF_MS) / ONE_MS);
```

When operating at Gigabit speed, with minimum frame size (64 bytes + preamble + IFG = 90 bytes or 720ns per frame, the table would fill in 2000 × 720 ns = 1.44 ms. For medium sized frames, e.g. 326 bytes (including Ethernet & PRP overhead), would be 326 × 8 ns = 2,608 ns per frame and 2000 frames in 5.2 ms.

Larger frame sizes of 1532 would fill the table in approx. 24 ms.

LRE STATISTICS

The SES_GetLreStatistics () API provides the statistics of the LRE. It contains information such as the number of frames transmitted or received on a port that has an HSR tagged or contains PRP RCT, the number of frames with the wrong LAN identifier, the number of frames with error received, the number of nodes in the nodes table, and the number of entries in the duplicate detection mechanism.

```
int32_t rv = SES_OK;
MIB_PRP_HSR_lreStatistics_t lreStatistics_p;
rv = SES_GetLreStatistics(&lreStatistics_p);
```

NODES TABLE

The switch maintains a nodes table to track DANPs and SANs in the PRP network. The node table is populated based on incoming traffic and PRP supervision frames on PRP ports. The assist engine automatically inserts or deletes entries based on the node table logic. The nodes table records duplicate discard or duplicate accept based upon received PRP supervision frames for a node. It records the last time a frame was received from a particular node and refreshes the nodes table every 60 seconds. If no frames have been received after this time, the node entry is removed. The user can interrogate the nodes table through driver APIs. The nodes table can accommodate 1024 entries max. The SES_GetLreNodesEntry () API provides information on a particular LRE nodes entry. The API will return information such as the index of the entry, the time the frame was received on a port, the MAC address, and node type.

```
int32_t SES_read_GetNodeEntry_Example() {
    int32_t result;
    int32_t nodeCount = 0;

    MIB_PRP_HSR_lreNodesEntry_t lreNodesEntry_p;
    MIB_PRP_HSR_lreStatistics_t lreStatistics_p;

    result = SES_GetLreStatistics(&lreStatistics_p);
    nodeCount = lreStatistics_p.lreCntNodes;
    printf("****************************************\n");
    printf("NodeCount =   %d\n", nodeCount);

    if (nodeCount > 0) {
        for (int i = 1; i <= nodeCount; i++)
```

PRP (PARALLEL REDUNDANCY PROTOCOL)

```
        {
            result = SES_GetLreNodesEntry(i, &lreNodesEntry_p);
            /*
             * Handle to avoid reading aged-out entries while reading the entire table.
             * If an entry has aged out during the read operation, the SES_GetLreNodesEntry
             * will return -1. This situation must be handled by the user.
             */
            if (result < 0) {
                continue;
            }
            else {
                printf("---------SES_GetLreNodesEntry---------::  %d \n", result);
                printf("lreNodesIndex   ::  %d\n", lreNodesEntry_p0.lreNodesIndex);
                printf("lreTimeLastSeenA :: %d\n", lreNodesEntry_p0.lreTimeLastSeenA);
                printf("lreTimeLastSeenB :: %d\n", lreNodesEntry_p0.lreTimeLastSeenB);
                printf("lre Node Type    :: %d\n", lreNodesEntry_p0.lreRemNodeType);

                /*
                 * The lreNodeReadTimeStamp parameter provides the timestamp when executing
                 * SES_GetLreNodesEntry for index 'i'. This helps the user verify the entry's
                 * aging and also track the corresponding API call.
                 */
                printf("lreNodeReadTimeStamp :: %d\n", lreNodesEntry_p0.lreNodeReadTimeStamp);
                printf("MAC Address:: ");
                for (int i = 0; i < 6; i++) {
                    printf("%02X", lreNodesEntry_p0.lreNodesMacAddress[i]);
                    if (i < 5) {
                        printf(":");
                    }
                }printf("\n");
            }
        }
    }
    printf("****************************************\n");

    return result;
}
```

The switch can also be configured as a PRP REDBOX, where other ports can be configured as interlink ports and contribute to the PRP network. The switch sends supervision frames into the PRP network, they are generated periodically with or without VLAN tag every LifeCheckInterval of 2 seconds. When configured as a Redbox, the switch will send corresponding supervision frames for Redbox and also transmit supervision frames for any Virtual DANs/SANs connected to the Redbox at the same interval.

PRP (PARALLEL REDUNDANCY PROTOCOL)

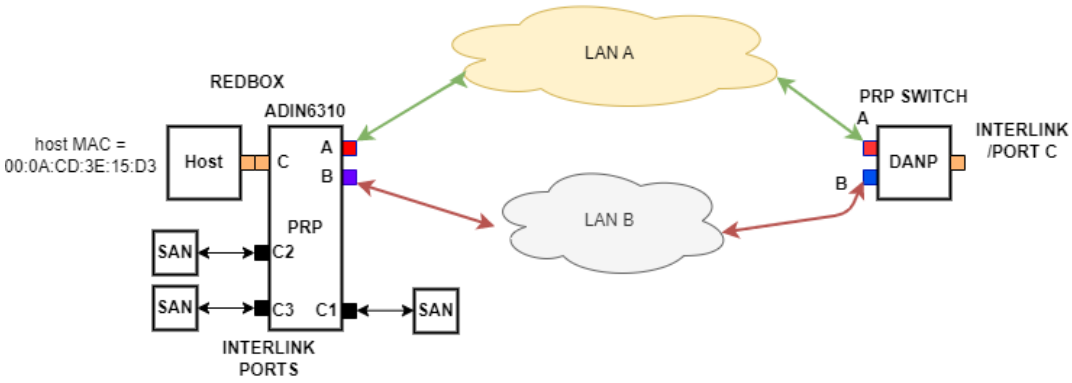

Figure 24. Example of Switch configuration as PRP-REDBOX (Host connected over Ethernet)

```
int32_t ses_test_prp_redBoxstart() {
    int32_t result = 0;

    /*SES_PrpStart(bool danpDevice,
        MIB_PRP_HSR_lrePorts_t * lrePorts_p,
        MIB_PRP_HSR_rbInterlinkPorts_t * rbInterlinkPorts_p,
        uint8_t * lreMacAddress_p,
        int32_t lreDupListResideMaxTime);*/

    MIB_PRP_HSR_redundancyType_t redundancyType = MIB_PRP_HSR_prpMode1;
    bool danpDevice = false; /*PRP REDBOX Configuration*/
    MIB_PRP_HSR_lrePorts_t lrePorts_p = { 2, 1, 0};  /*Port 2 = PORT A, Port 1 = PORT B, Port 0 = PORT
C*/
/*Ports 3, 4 and 5 configured as interlink ports */
    MIB_PRP_HSR_rbInterlinkPorts_t rbInterlinkPorts_p = { 3, 4, 5,-1 };
    uint8_t macAddress_g[6] = { 0x00, 0x0A, 0xCD, 0x3E, 0x15, 0xD3 };
    int32_t lreDupListResideMaxTime = 161;

    result = SES_PrpStart(danpDevice, &lrePorts_p, &rbInterlinkPorts_p, &macAddress_g, lreDupListResi▶
deMaxTime, MIB_PRP_HSR_fullStatistics);

    return result;
}
```

The switch can also be configured with PRP through an SPI host interface as shown in the example below.

```
int32_t ses_test_prp_redBoxstart() {
    int32_t result = 0;

    /*SES_PrpStart(bool danpDevice,
        MIB_PRP_HSR_lrePorts_t * lrePorts_p,
        MIB_PRP_HSR_rbInterlinkPorts_t * rbInterlinkPorts_p,
        uint8_t * lreMacAddress_p,
        int32_t lreDupListResideMaxTime);*/

    MIB_PRP_HSR_redundancyType_t redundancyType = MIB_PRP_HSR_prpMode1;
    bool danpDevice = false; /*PRP REDBOX Configuration*/
    MIB_PRP_HSR_lrePorts_t lrePorts_p = { 2, 1, -1};  /*Port 2 = PORT A, Port 1 = PORT B, SPI HOST =
```

PRP (PARALLEL REDUNDANCY PROTOCOL)

```
PORT C*/
/*Ports 3, 4 and 5 configured as interlink ports */
    MIB_PRP_HSR_rbInterlinkPorts_t rbInterlinkPorts_p = { 3, 4, 5,-1 };
    uint8_t macAddress_g[6] = { 0x00, 0x0A, 0xCD, 0x3E, 0x15, 0xD3 };
    int32_t lreDupListResideMaxTime = 161;

    result = SES_PrpStart(danpDevice, &lrePorts_p, &rbInterlinkPorts_p, &macAddress_g, lreDupListResi▶
deMaxTime, MIB_PRP_HSR_fullStatistics);

    return result;
}
```

PROXY NODES TABLE

The proxy nodes table is a list of the detected SANs that are connected to the Redbox and the last time they were seen. The proxy nodes table learns the SAN/VDAN MAC based on ingressing traffic on an interlink port. The user can interrogate the proxy nodes table. Like the nodes table, the Proxy node table keeps its table refreshed based on incoming frames and ages out entries after 60 seconds. The maximum size of the proxy node table for PRP RedBox is 8.

The proxy node table entries/indices do not map directly to MIB lreCntProxyNode statistics, therefore all node entries should be read back.

```
int32_t SES_read_GetProxyNodeEntry_Example() {
    int32_t result;
    int32_t pNodeCount = 0;

    MIB_PRP_HSR_lreNodesEntry_t lreNodesEntry_p0;
    MIB_PRP_HSR_lreProxyNodeEntry_t lreProxyNodeEntry_p;

    MIB_PRP_HSR_lreStatistics_t lreStatistics_p;
    result = SES_GetLreStatistics(&lreStatistics_p);
    pNodeCount = lreStatistics_p.lreCntProxyNodes;

    printf("*****************************************\n");
    printf("ProxyNodeCount =    %d\n", pNodeCount);

    if (pNodeCount > 0) {
        for (int32_t i = 0; i <= pNodeCount; i++)
        {
            result = SES_GetLreProxyNodeEntry(i, &lreProxyNodeEntry_p);

            /*
             * Handle to avoid reading aged-out entries while reading the entire table.
             * If an entry has aged out during the read operation, the SES_GetLreProxyNodeEntry API
             * will return -1. This situation must be handled by the user.
             */

            if (result < 0) {
                continue;
            }
            else {
                printf("SES_GetLrProxyeNodesEntry :: %d,  lreProxyNodesIndex :: %d \n", result, lre▶
ProxyNodeEntry_p.lreProxyNodeIndex);
                printf("MAC Address:: ");
                for (int i = 0; i < 6; i++) {
                    printf("%02X", lreProxyNodeEntry_p.lreProxyNodeMacAddress[i]);
```

PRP (PARALLEL REDUNDANCY PROTOCOL)

```
                    if (i < 5) {
                        printf(":");
                    }
                }printf("\n");
            }
        }
    }
    printf("****************************************\n");

    return result;
}
```

User can check if the PRP function is executing by calling the SES_VerifyReduExecuting() API.

INSTALLING STATIC ENTRY TABLE FOR BROADCAST TRAFFIC

By default, broadcast entries are not forwarded in a PRP device. Therefore, users must install a broadcast entry in the switching table to support pinging across a PRP devices. Without a static entry, broadcast frames will egress on LAN ports(A/B), but on the receiving side, the same frame will not be forwarded to PortC because there is no destination entry for the broadcast MAC. The below example to install static entry with source lookup and SendToAE enabled. In case of PRP Redbox, the Port map must be configured according to the interlink port configuration.

```
int32_t addStaticEntryBroadcastTrafficPRP_Example() {
  int32_t rv;
  uint8_t lookupPriority = 0;
  uint8_t macAddr[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
  uint8_t macMask[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
  int16_t vlanId = 0xFFF;
  uint16_t vlanMask = 0xFFF;
  uint8_t sourceEntry = 0;
  uint8_t override = 0;
  uint8_t sourceOverride = 0;
  uint8_t hsrOrPrpSupervisory = 0;
  SES_dynTblLookup_t lookupType = SES_dynamicTblLookupBasic;

  /*Port Map to forward Broadcast Packets*/
  uint8_t portMap = 0x3F;
  uint8_t sendToAE = 1;
  uint8_t danp = 0;
  uint8_t cutThrough = 0;
  uint8_t syntTimestamp = 0;
  uint8_t localTimestamp = 0;
  SES_dynTblSeqMgmt_t sequenceMgmt = dynamicTblSequenceGeneration;
  int rxSequenceSet = 0;
  int receiveFilter = SES_IGNORE_FIELD;
  int entryIndex;
  int16_t addRedundancyTagAndVlanIdx_p;
  int16_t removeRedundancyTagAndVlanIdx_p;
  int16_t vlanOnlyIdx_p;

  /* Retrieve the VLAN TX filter indices and use addRedundancyTagAndVlanIdx_p
  * to insert the PRP RCT tag when egressing broadcast packets on PRP redundant ports.
  */
  printf("SES_GetVlanTxFilterIndex :: %d\n",
          SES_GetVlanTxFilterIndex(&addRedundancyTagAndVlanIdx_p,
```

PRP (PARALLEL REDUNDANCY PROTOCOL)

```
            &removeRedundancyTagAndVlanIdx_p, &vlanOnlyIdx_p));


  rv = SES_AddStaticTableEntryEx(lookupPriority, macAddr, macMask, vlanId, vlanMask,
           sourceEntry, override, sourceOverride, hsrOrPrpSupervisory, lookupType,
           portMap, sendToAE, danp, cutThrough, syntTimestamp, localTimestamp,
           sequenceMgmt, rxSequenceSet, addRedundancyTagAndVlanIdx_p, receiveFilter,
           &entryIndex);

  return rv;
}
```

HSR (HIGH AVAILABILITY SEAMLESS REDUNDANCY)

Details of the HSR APIs are included in the 'SES_prp_hsr.h' header file. The Switch device supports being configured as a DANH or HSR REDBOX. Following initial device configuration, the host configures the Switch for the desired HSR function, defining which ports are A/B/C in the case of a DANH and also any other interlink ports in the case of a Redbox, the host sets the LRE MAC address (same as host MAC address) and enables the HSR function.

Once configured for the HSR mode, the Switch hardware takes care of HSR functionality, duplicating the outgoing traffic onto each of its ring ports with the HSR tag inserted into the frame. On receipt of HSR frames from the ring, the receiving switch consumes the first frame, removes the tag on reception and discards duplicates. The switch generates supervisory frames and maintains a nodes table of other entities in the network based on the traffic the switch received from the ring ports. The HSR supervisory frames are generated periodically with or without VLAN tag every 2 seconds. The hardware records the last time a frame was received from a node, refreshing the nodes table. Each device in the HSR ring maintains its own nodes table. Node entries are removed from the table based on the NodeForgetTime default of 1 minute. The node table is currently capable of supporting up to 1024 entries. The nodes table records entries for DANH, Redbox and VDAN devices connected to the ring, based on the supervision frames circulating the ring.

When operating as a RedBox, the switch also maintains a Proxy node table in addition to the nodes table. The proxy nodes table is a list of the detected SANs that are connected to the Redbox and the last time they were seen. The proxy nodes table learns the SAN/VDAN MAC based on ingressing traffic on ports configured as interlink ports. Like the nodes table, the Proxy node table keeps its table refreshed based on incoming frames and ages out entries after 60 seconds. The maximum size of the proxy node table for HSR RedBox is 8.

LLDP and VLAN table may be used with HSR functionality.

NOTE: Internally, the switch uses a VLAN ID to route HSR tagged frames. Users must not use this VLAN ID for general traffic switching. By default, the VLAN ID used by the HSR protocol is 4094 (0xFFE). A different VLAN ID can be configured for the HSR stack using the SES_SetHsrVlanId.

When the switch is configured for PRP or HSR mode, the normal learning of the forwarding table is disabled, instead the table stores entries based on the HSR/PRP devices/traffic. The host can subscribe to an event SES_dynTblLimitExceedEvent which indicates that the number of entries exceeds 1792 or 87.5% of the table entries.

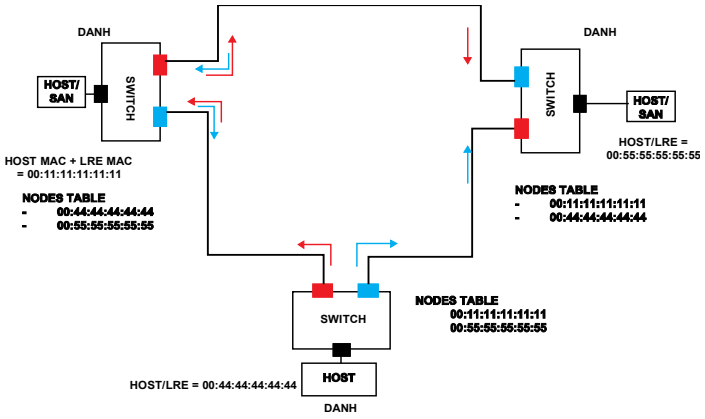

Figure 25. HSR configuration with 3 DANH devices

HSR (HIGH AVAILABILITY SEAMLESS REDUNDANCY)

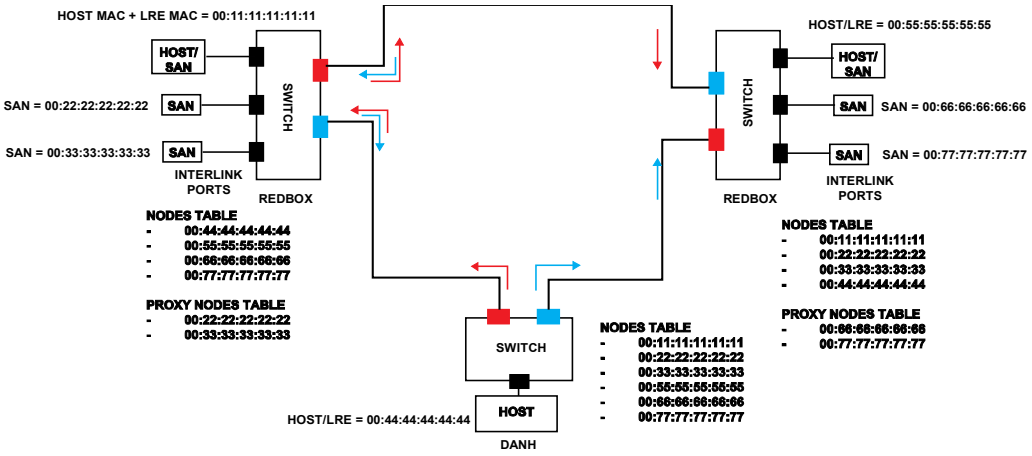

Figure 26. HSR configuration with 2 HSR REBOX devices and 1 DANH

Supported features with latest release:

- HSR as DANH or REDBOXSAN
- HSR configuration via SPI Host interface
- HSR with LLDP
- HSR with VLAN functionality
- Cut through operation with HSR (Possible across the ring only)

Currently not supported:

- HSR with PTP.
- HSR to HSR / HSR Quadbox
- HSR to PRP Redbox

Not supported:

- Multiple instances of HSR running on a 6-port switch.
- HSR with any TSN functionality

PREVENTING TRAFFIC LOOPS DURING HSR INITIALIZATION

In some HSR deployment scenarios, the physical ring topology may be established before HSR is enabled on the switch. During this period, packet looping or flooding may occur due to the switch's default forwarding behavior, in which unknown packets are forwarded to all ports. This condition can result in excessive traffic and network instability.

To prevent this behavior, the host application can explicitly configure the switch to prevent routing unknown unicast and multicast traffic on the ring ports until HSR initialization is complete. This can be achieved using the following APIs:

- SES_SetUnicastMissReturn
- SES_SetMulticastMissReturn

These APIs allow the application to drop packets received on the ring ports instead of forwarding them to other ports, thereby preventing traffic from looping within the ring. Once HSR is initialized, these temporary forwarding rules are automatically overridden by HSR-specific forwarding rules, and normal HSR operation begins.

HSR (HIGH AVAILABILITY SEAMLESS REDUNDANCY)

For example, if the ring topology is connected through Port 1 and Port 2, the application can configure both ports to drop unicast and multicast miss traffic prior to HSR initialization.

```
//Unicast miss return for the ring ports

SES_SetUnicastMissReturn(SES_macPort1, 0x00U, false, false, 0x11, 0U, false, 0U, false);
SES_SetUnicastMissReturn(SES_macPort2, 0x00U, false, false, 0x11, 0U, false, 0U, false);

//Multicast miss return for the ring ports
SES_SetMulticastMissReturn(SES_macPort1, 0x00U, false, false, 0x11, 0U, false, 0U, false);
SES_SetMulticastMissReturn(SES_macPort2, 0x00U, false, false, 0x11, 0U, false, 0U, false);
```

HSR OPERATING MODES

The switch supports the various HSR Modes. Mode H is the default operating mode.

- MIB_PRP_HSR_modeH: Default mode - the DANH shall insert the HSR tag on behalf of its host and shall forward the ring traffic, except for frames sent by the node itself, duplicate frames and frames for which the node is the unique destination
- MIB_PRP_HSR_modeN: No forwarding - Node shall behave like Mode H with the exception that it shall not forward ring traffic from port to port
- MIB_PRP_HSR_modeT: Transparent forwarding - shall remove the HSR tag before forwarding the frame to the other port and shall send a frame from the host to both ports, untagged and without discarding duplicates
- MIB_PRP_HSR_modeU: Unicast forwarding - the node shall behave as in Mode H, except that it shall also forward traffic for which it is the unique destination
- MIB_PRP_HSR_modeM: Mixed forwarding - the DANH shall insert the HSR tag depending on local criteria when injecting frames into the ring
- MIB_PRP_HSR_modeX: No sending on counter-duplicate - Node shall behave as in Mode H, except that a port shall not send a frame that is a duplicate of a frame that it received completely and correctly from the opposite direction

HSR EXAMPLES

The following examples show how to configure the switch for HSR modes of operation.

NOTE: All switch ports must be enabled during port initialization before HSR initialization.

Enabling HSR as DANH Example (Host Interface as Ethernet Port)

```
//Call after initialization and port configuration is complete

int32_t ses_test_hsr_start() {
    int32_t result = 0;

   /* int32_t SES_HsrStart(MIB_PRP_HSR_switchEndNode_t lreSwitchingEndNode,
        MIB_PRP_HSR_lrePorts_t * lrePorts_p,
        MIB_PRP_HSR_rbInterlinkPorts_t * rbInterlinkPorts_p,
        uint8_t * lreMacAddress_p,
        int32_t lreDupListResideMaxTime);*/

    MIB_PRP_HSR_switchEndNode_t lreSwitchingEndNode = MIB_PRP_HSR_hsrNode;
    MIB_PRP_HSR_lrePorts_t lrePorts_p = { 2, 1, 0 };
    MIB_PRP_HSR_rbInterlinkPorts_t rbInterlinkPorts_p = { -1,-1,-1,-1 };
    uint8_t macAddress_g[6] = { 0xAC, 0x91, 0xA1, 0x99, 0x99, 0x99 };
    int32_t lreDupListResideMaxTime = 161;

    result = SES_HsrStart(redundancyType, lreSwitchingEndNode, &lrePorts_p, &rbInterlinkPorts_p, &mac▶
```

HSR (HIGH AVAILABILITY SEAMLESS REDUNDANCY)

```
Address_g, lreDupListResideMaxTime, MIB_PRP_HSR_fullStatistics);

    return result;
}
```

The switch supports configuration of HSR when the Host interface is SPI interface. The SPI interface is seen as Port C in this case and traffic originating from the host, will be forwarded into the ring as HSR traffic, traffic coming from the ring targeted at the host will have the duplicates and HSR tags removed.

Enabling HSR as DANH Example (Host Interface as SPI Interface)

```
//Call after initialization and port configuration is complete

int32_t ses_test_hsr_start() {
    int32_t result = 0;

   /* int32_t SES_HsrStart(MIB_PRP_HSR_switchEndNode_t lreSwitchingEndNode,
        MIB_PRP_HSR_lrePorts_t * lrePorts_p,
        MIB_PRP_HSR_rbInterlinkPorts_t * rbInterlinkPorts_p,
        uint8_t * lreMacAddress_p,
        int32_t lreDupListResideMaxTime);*/

    MIB_PRP_HSR_switchEndNode_t lreSwitchingEndNode = MIB_PRP_HSR_hsrNode;
    //For SPI interface, lreDanPortC must be -1
    MIB_PRP_HSR_lrePorts_t lrePorts_p = { 2, 1, -1 };
    MIB_PRP_HSR_rbInterlinkPorts_t rbInterlinkPorts_p = { -1,-1,-1,-1 };
    uint8_t macAddress_g[6] = {0x00, 0x00, 0x00, 0x11, 0x11, 0x11};
    int32_t lreDupListResideMaxTime = 161;

    result = SES_HsrStart(lreSwitchingEndNode, &lrePorts_p, &rbInterlinkPorts_p, &macAddress_g, lreDu▶
pListResideMaxTime, MIB_PRP_HSR_fullStatistics);

    return result;
}
```

Enabling HSR as HSR REDBOX SAN, Host Interface as Ethernet Port

```
//Call after initialization and port configuration is complete

int32_t ses_test_hsr_redBoxstart() {
    int32_t rv = 0;

    /*int32_t SES_HsrStart( MIB_PRP_HSR_switchEndNode_t lreSwitchingEndNode,
                          MIB_PRP_HSR_lrePorts_t * lrePorts_p,
                          MIB_PRP_HSR_rbInterlinkPorts_t * rbInterlinkPorts_p,
                          uint8_t * lreMacAddress_p,
                          int32_t lreDupListResideMaxTime)*/

    // HSR configurations: MIB_PRP_HSR_hsrRedboxSan, MIB_PRP_HSR_hsrNode

    MIB_PRP_HSR_switchEndNode_t lreSwitchingEndNode = MIB_PRP_HSR_hsrRedboxSan;
    MIB_PRP_HSR_lrePorts_t lrePorts_p = { 1, 2, 0 };
```

HSR (HIGH AVAILABILITY SEAMLESS REDUNDANCY)

```
    //When configured as a HSRREDBOX, interlink ports can be used,here Ports 3 and 4 are interlink
ports
    MIB_PRP_HSR_rbInterlinkPorts_t rbInterlinkPorts_p = { 3,4,-1,-1 };

    //LRE MAC address must be same as Host MAC address
    uint8_t macAddress_g[6] = { 0xAC, 0x91, 0xA1, 0x88, 0x88, 0x88 };
    int32_t lreDupListResideMaxTime = 161;

    rv = SES_HsrStart(lreSwitchingEndNode, &lrePorts_p, &rbInterlinkPorts_p, &macAddress_g, lreDupLis▶
tResideMaxTime, MIB_PRP_HSR_fullStatistics);

    return rv;
}
```

The switch captures detailed information in the form of statistics showing what is happening in the network.

Read Back Statistics

When the switch operates in HSR mode, LRE statistics are maintained in software by the Packet Assist Engine (PAE). To support statistics collection, traffic is routed to the PAE for processing. This introduces additional overhead on the PAE, and under high traffic conditions the PAE may not be able to update statistics accurately, potentially resulting in inconsistent LRE counters.

During HSR initialization, users must select a statistics mode using the **statisticOptions** parameter. Two types of statistics are supported partial statistics or full statistics.

Selecting partial statistics reduces the amount of traffic sent to the PAE, thereby lowering processing overhead and improving performance.

The **statisticOptions** parameter applies only to HSR mode when the host interface is Ethernet. When the host interface is SPI, the switch always operates in full statistics mode, and any attempt to select partial statistics is ignored.

When partial statistics are enabled (MIB_PRP_HSR_partialStatistics):

- Only unicast statistics for traffic to and from the Port C or Interlink ports are recorded by switch.
- Unicast, multicast, and broadcast miss traffic is not captured.
- To maintain accurate statistics, unicast traffic must be limited to less than 2000 frames per second, higher traffic rates may result in inaccurate statistics.

In this mode, PAE performance improves by avoiding the transfer of unnecessary frames to the PAE.

When full statistics are enabled (MIB_PRP_HSR_fullStatistics):

- Unicast and multicast statistics are maintained for all frames received or transmitted at the HSR ring ports and interlink ports.
- For Ethernet host interfaces, performance improved by sending frame metadata-only to the PAE instead of full frames, enabling accurate statistics up to 2000 frames per second.
- For SPI host interface, the metadata-only optimization is not supported because the host may have subscribed to callbacks based on MAC address, port, or Ethertype. As a result, performance improvement for full statistics (metadata-only) does not apply to SPI, and SPI full-statistics mode may experience higher overhead.
- Traffic rate above this limit may result inaccurate statistics.

The example below demonstrates how to configure the statistics option to use partial statistics.

```
int32_t ses_hsr_init() {
  int32_t rv = 0;
  MIB_PRP_HSR_switchEndNode_t lreSwitchingEndNode = MIB_PRP_HSR_hsrNode;
  MIB_PRP_HSR_lrePorts_t lrePorts_p = {1, 2, 0};
  MIB_PRP_HSR_rbInterlinkPorts_t rbInterlinkPorts_p = {-1, -1, -1, -1};
  uint8_t macAddress_g[6] = {0x4C, 0xD7, 0x17, 0xA1, 0xC5, 0xA1};
```

HSR (HIGH AVAILABILITY SEAMLESS REDUNDANCY)

```
  int32_t lreDupListResideMaxTime = 161;

  /* Configured MIB_PRP_HSR_statisticOptions_t for partial statistics */
  rv = SES_HsrStart(lreSwitchingEndNode, &lrePorts_p, &rbInterlinkPorts_p,
          &macAddress_g, lreDupListResideMaxTime, MIB_PRP_HSR_partialStatistics);

  return rv;
}
```

The example below demonstrates how to retrieve LRE statistics.

```
//HSR must be configured and running
int32_t SES_test_GetLreStatistics() {

    int32_t  rv;
    MIB_PRP_HSR_lreStatistics_t lreStatistics_p;
    rv = SES_GetLreStatistics( &lreStatistics_p);

    printf("-------------HSR LreStatistics---------\n");

    printf("lreCntTxA :: %d\n", lreStatistics_p.lreCntTxA);
    printf("lreCntTxB :: %d\n", lreStatistics_p.lreCntTxB);

    printf("lreCntRxA :: %d\n", lreStatistics_p.lreCntRxA);
    printf("lreCntRxB :: %d\n", lreStatistics_p.lreCntRxB);
    printf("lreCntRxC :: %d\n", lreStatistics_p.lreCntRxC);

//Shows the number of nodes in the Nodes table and Proxy nodes table when operating as Redbox

    printf("NodeCount ::  %d\n", lreStatistics_p.lreCntNodes);
    printf("ProxyNodeCount ::    %d\n", lreStatistics_p.lreCntProxyNodes);

//Other available statistics include Unique count, duplicate count, multiple counts and HSR //frames
received on ring ports that originated from this device.

    return rv;
}
```

Both the nodes and proxy nodes table provide insight into the devices connected in the ring. The Proxy nodes table records the interlink nodes connected to the Redbox, while the nodes table records the MAC address of the nodes in the network and the last time seen on each ring port.

Reading Nodes Table and Proxy Nodes Table

Proxy Node Table

```
//HSR must be configured and running
int32_t SES_read_GetNodeEntry_Example() {
    int32_t result;
    int32_t nodeCount = 0;

    MIB_PRP_HSR_lreNodesEntry_t lreNodesEntry_p;
    MIB_PRP_HSR_lreStatistics_t lreStatistics_p;
```

HSR (HIGH AVAILABILITY SEAMLESS REDUNDANCY)

```
    result = SES_GetLreStatistics(&lreStatistics_p);
    nodeCount = lreStatistics_p.lreCntNodes;
    printf("****************************************\n");
    printf("NodeCount = %d\n", nodeCount);

    if (nodeCount > 0) {
        for (int i = 1; i <= nodeCount; i++)
        {
            result = SES_GetLreNodesEntry(i, &lreNodesEntry_p);
            /*
            * Handle to avoid reading aged-out entries while reading the entire table.
            * If an entry has aged out during the read operation, the SES_GetLreNodesEntry API
            * will return -1. This situation must be handled by the user.
            */
            if (result < 0) {
                continue;
            }
            else {
                printf("---------SES_GetLreNodesEntry---------:: %d \n", result);
                printf("lreNodesIndex :: %d\n", lreNodesEntry_p0.lreNodesIndex);
                printf("lreTimeLastSeenA :: %d\n", lreNodesEntry_p0.lreTimeLastSeenA);
                printf("lreTimeLastSeenB :: %d\n", lreNodesEntry_p0.lreTimeLastSeenB);
                printf("lre Node Type :: %d\n", lreNodesEntry_p0.lreRemNodeType);

                /*
                * The lreNodeReadTimeStamp parameter provides the timestamp when executing
                * SES_GetLreNodesEntry for index 'i'. This helps the user verify the entry's
                * aging and also track the corresponding API call.
                */
                printf("lreNodeReadTimeStamp :: %d\n", lreNodesEntry_p0.lreNodeReadTimeStamp);
                printf("MAC Address:: ");
                for (int i = 0; i < 6; i++) {
                    printf("%02X", lreNodesEntry_p0.lreNodesMacAddress[i]);
                    if (i < 5) {
                        printf(":");
                    }
                }printf("\n");

            }
        }
    }
    printf("****************************************\n");

    return result;
}
```

Proxy Nodes Table

```
int32_t SES_read_GetProxyNodeEntry_Example() {
    int32_t result;
    int32_t pNodeCount = 0;

    MIB_PRP_HSR_lreNodesEntry_t lreNodesEntry_p0;
```

HSR (HIGH AVAILABILITY SEAMLESS REDUNDANCY)

```
    MIB_PRP_HSR_lreProxyNodeEntry_t lreProxyNodeEntry_p;

    MIB_PRP_HSR_lreStatistics_t lreStatistics_p;
    result = SES_GetLreStatistics(&lreStatistics_p);
    pNodeCount = lreStatistics_p.lreCntProxyNodes;

    printf("*****************************************\n");
    printf("ProxyNodeCount =    %d\n", pNodeCount);

    if (pNodeCount > 0) {
        for (int32_t i = 0; i <= pNodeCount; i++)
        {
            result = SES_GetLreProxyNodeEntry(i, &lreProxyNodeEntry_p);

            /*
             * Handle to avoid reading aged-out entries while reading the entire table.
             * If an entry has aged out during the read operation, the SES_GetLreProxyNodeEntry API
             * will return -1. This situation must be handled by the user.
             */

            if (result < 0) {
                continue;
            }
            else {
                printf("SES_GetLrProxyeNodesEntry :: %d,  lreProxyNodesIndex :: %d \n", result, lre▶
ProxyNodeEntry_p.lreProxyNodeIndex);
                printf("MAC Address:: ");
                for (int i = 0; i < 6; i++) {
                    printf("%02X", lreProxyNodeEntry_p.lreProxyNodeMacAddress[i]);
                    if (i < 5) {
                        printf(":");
                    }
                }printf("\n");
            }
        }
    }
    printf("*****************************************\n");

    return result;
}
```

Change HSR Supervision Frame MAC Address

HSR supervision frames are sent with a reserved multicast address 01:15:4e:00:01:XX. By default, the switch sends frames with an address of 01:15:4e:00:01:00. The last byte value can be changed using the SES_SetSupMacAddress() API.

```
//HSR must be configured and running

int32_t SES_test_SetSupMacAddress() {
    int32_t rv;
    //MAC Address must be in range 01-15-4E-00-01-00 to 01-15-4E-00-01-ff
    uint8_t macAddress_g[6] = { 0x01, 0x15,  0x4E, 0x00,  0x01,  0xFF };
    rv = SES_SetSupMacAddress(&macAddress_g);
```

HSR (HIGH AVAILABILITY SEAMLESS REDUNDANCY)

```
    return rv;
}
```

HSR mode supports both modes of forwarding, cut-through or Store and forward. By default in HSR mode, the switch is configured for Store and forward mode. This can be changed with a call to SES_SetCutThroughForward() API.

Changing From Store and Forward Mode to Cut-through

HSR Redbox mode cut-through operation only applies to Port A and B. It does not apply to Port C or interlink ports as the HSR tag must have frame size information to calculate the LDSU to insert in the HSR tag.

```
//HSR must be configured and running
//Configure device to be in cut-through mode.

void SES_test_SetCutThroughForward() {

    MIB_PRP_HSR_cutThrough_t cutThrough = MIB_HSR_cutThrough;
    printf("SES_SetCutThroughForward status :: %d\n", SES_SetCutThroughForward(cutThrough));

}//Readback what mode switch is in

int32_t SES_test_GetCutThroughForward() {

    int32_t rv;
    MIB_PRP_HSR_cutThrough_t cutThrough_p;
    result = SES_GetCutThroughForward(&cutThrough_p);
    printf("MIB_PRP_HSR_cutThrough_t :: %d\n", cutThrough_p);
    return rv;
}
```

MEDIA REDUNDANCY PROTOCOL, MRP

MRP is another redundancy protocol used to avoid single points of failure in industrial communications networks. MRP is a recovery protocol based on a ring topology and is aligned with IEC 62439-2 standard. MRP can be used in ring networks. For full details on MRP protocol review the detailed standard. The following descriptions provide an overview of MRP to help describe the functionality provided by the Switch devices and does not intend to be a full overview of MRP function.

MRP STACK ON THE SWITCH

All MRP related APIs are detailed in SES_mrp.h in ses-tsn-api-srv. MRP can be configured on startup of the switch. The MRP stack is running on the Packet Assist Engine, thereby offloading MRP overhead from the host. The Switch supports operation as a Media Redundancy Client (MRC), a Media Redundancy Manager (MRM) or a Media Redundancy Automanager (MRA).

The switch does not support interconnected rings.

One instance of MRA/MRC/MRM is supported on a 6-port device. All TSN functionality is supported with MRP.

RECOVERY PROFILES

When the Switch device is configured for MRP operation, it supports recovery profiles of 500 ms, 200 ms or 30 ms. Currently 10ms profile is not supported.

MRP TRANSMIT PRIORITY

The default transmit priority for MRP, PTP and LLDP is Q7. MRP should have higher transmit priority than gPTP and LLDP, particularly if using MRP with the faster recovery profiles. The SES_MrpSetXmitPriority() API is used to configure the priority for MRP while the SES_MrpGetXmitPriority() API provides the current priority (default 7).

ENABLING MRP

MRP function can be started using the SES_MrpStart() API with a default configuration of MRC, recovery rate of 500 ms and ring ports configured for Ports 1 & 2. The configuration can be changed after starting the MRP stack. Alternatively, the MRP stack supports configuring the function first followed by calling SES_MrpStart() API.

Once MRP function is enabled, user can verify using the SES_MrpVerifyEnabled() API.

Note: The Domain VLAN ID should be set after VLAN configuration is applied on the ports. For example, if the Domain VLAN ID is supposed to be 100, the corresponding ports should be configured as Access/Trunk members of VLAN 100 first and then the Domain VLAN ID should be set to 100.

```
//MRP default configuration

void SES_test_StartMRP() {

        /* Initialize MRP with default configuration
            Ring Role: Client
            Recovery Rate: 500 ms
            Ring Ports: Port 1 & 2
            VLAN: No Vlan(0x0FFF)
            Priority: Default Manager / AutoManager priorities
            React On Link Change: Disabled*/
    printf("MRP Initialization :: %d\n", SES_MrpStart());
}

/*MRP configuration and Initialization, configure device as MRM, then start/initialize MRP protocol */

void SES_MRPConfigAndInit() {
    // Port 2 and 3 will be configured as Ring ports
    SES_mac_t rPort1Mac = SES_macPort2;
    SES_mac_t rPort2Mac = SES_macPort3;
    //Ring Port Status
```

MEDIA REDUNDANCY PROTOCOL, MRP

```
    SES_mrpPortInfo_t port1;
    SES_mrpPortInfo_t port2;

    SES_mrpInstanceInfo_t config_p = {
        .oui = {0xAA,0xBB,0xCC},
        .domainUUID = {0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
                       0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff},
        .reactOnLinkChange = 0,
        .isBlockedSupported = 1,
        .priority = 0x8000,
              .domainVlanID = 0x0FFF,
        .ringRole = SES_ringRoleManager,
        .autoMgrActiveRole = SES_ringRoleInactive,
        .recoveryRate = SES_recoveryRate200

    };
    // Configure Ring ports and initialize MRP
    if (SES_OK == SES_MrpSetRingPorts(rPort1Mac, rPort2Mac)) {
        SES_MrpGetRingPortInfo(&port1, &port2);
        printf("Ring Port1 :: %d, Port 2 :: %d\n", port1.mac, port2.mac);

        if (SES_OK == SES_MrpSetInstanceConfig(&config_p)) {
            printf("MRP Initialization :: %d\n", SES_MrpStart());
        }
    }
}
```

READ THE OPERATIONAL MRP INSTANCE CONFIGURATION

SES_MrpGetInstanceConfig() API provides insight into the current operational state of the ring.

```
void SES_test_GetInstanceConfig() {

    // Get the operational MRP instance configuration
    SES_mrpInstanceInfo_t config_p;
    if (SES_OK == SES_MrpGetInstanceConfig(&config_p)) {
        printf("MRP Ring Role :: %d\n", config_p.ringRole);
        printf("AutoManager Active Role :: %d\n", config_p.autoMgrActiveRole);
        printf("Ring State :: %d\n", config_p.ringState);
        printf("React on link change :: %d \n", config_p.reactOnLinkChange);
        printf("priority ::  %d\n", config_p.priority);
        printf("DomainVLANid ::  %d\n", config_p.domainVlanID);
        printf("Recovery Rate  ::  %d\n", config_p.recoveryRate);
        printf("isBlockedSupported ::  %d\n", config_p.isBlockedSupported);
    }
}
```

READ DOMAIN STATISTICS

SES_MrpGetDomainStatistics() API provides insight into statistics associated with MRP such as a count of how many times the ring has been observed to be open, a timestamp of the last change of ring state to open in addition to the round trip delays measured since startup and expressed in ms.

```
void SES_test_GetDomainStatistics() {
```

MEDIA REDUNDANCY PROTOCOL, MRP

```
    SES_mrpStatistics_t stats_p;

    if (SES_OK == SES_MrpGetDomainStatistics(&stats_p)) {
        printf("MRP Stats ::\n");
        printf("ringOpenCount :: %d\n", stats_p.ringOpenCount);
        printf("lastRingOpenTimestamp :: %d\n", stats_p.lastRingOpenTimestamp);
        printf("roundTripDelaysMin :: %d\n", stats_p.roundTripDelaysMin);
        printf("roundTripDelaysMax :: %d\n", stats_p.roundTripDelaysMax);
    }
}
```

UPDATING THE RUNNING MRP INSTANCE CONFIGURATION

MRP configuration can be changed after starting the MRP stack.

```
void SES_test_SetMrpClient() {
    //Update the Role and reactOnLinkChange parameter
    SES_mrpInstanceInfo_t config_p;
    if (SES_OK == SES_MrpGetInstanceConfig(&config_p)) {
        config_p.ringRole = SES_ringRoleClient;
        config_p.reactOnLinkChange = 0;
        printf("SES_SetInstanceConfig Status :: %d \n", SES_MrpSetInstanceConfig(&config_p));
    }
}
```

SENDLIST

The Switch supports ability to develop a generic send list function that can meet the requirements of IEC60802 and the needs of devices operating layer 2 data protocols like PROFINET, CC-LINK and OPC-UA FLC.

The primary concern is to support devices that rely on the SES SPI interface for transmission of frames and are therefore unable to have precise control of the timing. A secondary goal will be to support transmission of a subset of traffic using this facility for devices connected over MAC interface but lacking the timing/queueing/DMA facilities to fully implement a send-list.

A send list is a periodic transmission of a number of sublists. Each sublist will contain a number of ethernet frames to be sent at a specific offset in the send list period. As the Sendlist is periodic, if the frame data is not updated before a sublist begins the next cycle transmission, the last frame data will be sent. The Sendlist function relies on using a dedicated port in the switch, configured into loopback mode internally. In a 6-port device, this port can be any of the 6 ports (just not the host interface port). In the ADIN3310, 3-port device, there is a dedicated port (port number 5) available for this purpose, which does not affect operation on ports 0-2.

Once the port is configured in loopback mode as part of the Sendlist configuration, it is not available for regular ingressing traffic and must be dedicated to the Sendlist function. The loopback configuration connects the transmit MAC with the receive MAC internally. Only one Sendlist instance is supported. All Sendlist related APIs are described in SES_sendlist.h. MSTP needs additional configuration when using with Sendlist. The MSTP stack should not be enabled for loopback port of the Sendlist. SES_MstpEnableSpanningTreePort() can be used to disable MSTP on the loopback port.

When creating a SendList, the following steps outline the intended order of operation:

1. Create a send list with the number of sublists and desired period, SES_SendListCreate()
2. Create sublist 0-n with the desired start offset from the start of the send list period, SES_SublistCreate()
3. Register send list frames that will be added to the sublists, SES_RegisterFrame()
4. Add a static table entry for each registered frame with the desired port map.
5. Add the registered frames to the appropriate sublists, SES_AddFrameToSublist()
6. Configure the scheduled traffic for the loopback port.
7. Send registered frame data with the transmit enable parameter set for each registered frame, SES_UpdateRegisteredFrame()
8. Start the send list with the desired start time, SES_SendListStart()
9. To stop transmitting a specific frame, use the enable/disable parameter of the update registered frame or call the stop registered frame function with the frameID of the frame to stop transmitting, SES_StopRegisteredFrame()
10. To stop transmitting a sendlist, disable or stop all registered frames.
11. To change or update a sendlist program, create the new sendlist and sublists. Then add registered frames to sublists and start the new sendlist.

Normal AE transmits will be allowed between the end of the sublist and the start of the next. The sendlist will halt the AE transmit the max frame time before the start of a sublist. The send list will start transmitting the largest registered frame time before the production time to ensure the frame is queued before the appropriate gate opens. Ethernet host transmits will prioritize based on bridge configuration and priority.

SENDLIST EXAMPLE

The following example shows how to configure the Sendlist feature with 4 sendlists. Sublist0 had three frame entries, while Sublists 1-3 have just two frame entries. The frames in each sublist are shown in Figure 27 and correspond to the example described below. The example shows SES_MstpEnableSpanningTreePort in comments because the MSTP stack is not running on the example.

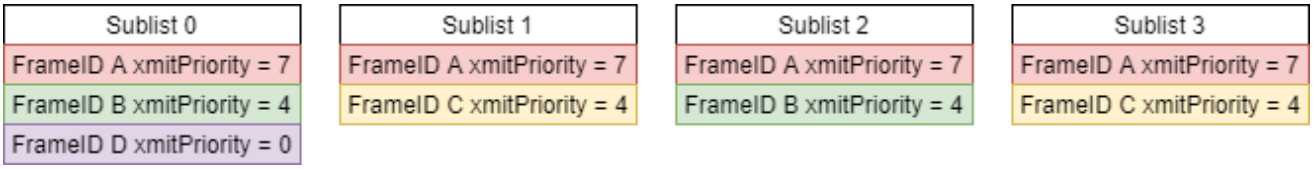

Figure 27. Sendlist Example

```
/* Example of creating a simple Sendlist with 4 sublists & associated frames as shown above */

int32_t rv = 0;
```

SENDLIST

```
int32_t* frameID_A;
int32_t* frameID_B;
int32_t* frameID_C;
int32_t* frameID_D;

TSN_ieee802_dot1q_sched_gate_parameters_t gateParams;
TSN_ieee802_dot1q_sched_gate_oper_parameters_t opergateParams;
TSN_ieee802_dot1q_types_port_number_t portNumber = 5; /* Port number for loopback port */


/* example frame data, same DMAC for all, SMAC has AA, BB, CC, DD addresses to simplify visibility in
wireshark example*/
uint8_t frameData_A[] = { 0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC, 0x11, 0xAA, 0xAA,
        0xAA, 0xAA, 0xAA, 0x00, 0x00, 0xe0, …};
uint8_t frameData_B[] = { 0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC, 0x11, 0xBB, 0xBB,
        0xBB, 0xBB, 0xBB, 0x00, 0x00, 0x80, …};
uint8_t frameData_C[] = { 0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC, 0x11, 0xCC, 0xCC,
        0xCC, 0xCC, 0xCC, 0x00, 0x00, …};
uint8_t frameData_D[] = { 0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC, 0x11, 0xDD, 0xDD,
        0xDD, 0xDD, 0xDD, 0x00, 0x00, 0x00, …};

/*
 * The function below disables MSTP on specific port (loopback port 5)
 */
// rv = SES_MstpEnableSpanningTreePort(6, false);

/* Add a static table entry (or entries) to steer the traffic out intended port. No entry is required
for loopback port */

uint8_t macAddr_p[6] = { 0x12, 0x34, 0x56, 0x78, 0x9a, 0xbc };
int16_t vlanId = 0xFFF;
uint8_t portMap = 0x4; //egress on Port3
rv = SES_AddStaticTableEntry(&macAddr_p, vlanId, portMap);

/* Initialize send list, identifying number of sublists, four in this case, with a repeat period of
100ms & loopback port of 5 */
rv = SES_SendListCreate(4, 100000000, portNumber);

/* Create sublists, identify number of entries and time offset*/
rv = SES_SublistCreate(0, 3, 0);
rv = SES_SublistCreate(1, 2, 800000);
rv = SES_SublistCreate(2, 2, 1600000);
rv = SES_SublistCreate(3, 2, 2400000);


/* Register send list frames with transmit priority */
rv = SES_RegisterFrame(&frameID_A, sizeof(frameData_A), 1, 7);
rv = SES_RegisterFrame(&frameID_B, sizeof(frameData_B), 1, 4);
rv = SES_RegisterFrame(&frameID_C, sizeof(frameData_C), 1, 4);
rv = SES_RegisterFrame(&frameID_D, sizeof(frameData_D), 1, 0);

/* Add frame A to first entry of each sublist */
rv = SES_AddFrameToSublist(frameID_A, 0, 0);
rv = SES_AddFrameToSublist(frameID_A, 1, 0);
rv = SES_AddFrameToSublist(frameID_A, 2, 0);
rv = SES_AddFrameToSublist(frameID_A, 3, 0);
```

SENDLIST

```
/* Add frame B to second entry of sublist 0 and 2 */
rv = SES_AddFrameToSublist(frameID_B, 0, 1);
rv = SES_AddFrameToSublist(frameID_B, 2, 1);

/* Add frame C to second entry of sublist 1 and 3 */
rv = SES_AddFrameToSublist(frameID_C, 1, 1);
rv = SES_AddFrameToSublist(frameID_C, 3, 1);

/* Add frame D to third entry of sublist 0 */
rv = SES_AddFrameToSublist(frameID_D, 0, 2);

/* Set up schedule for loopback port (5), 800us schedule with three intervals */



rv = SES_QbvGetGateParameters(portNumber, &opergateParams);
uint64_t currentTimeSec = opergateParams.current_time.seconds;
uint32_t currentTimeNSec = opergateParams.current_time.nanoseconds;
if (currentTimeNSec  > 500000000) {
    currentTimeSec += 1;
    }

gateParams.gate_enabled = true;
gateParams.config_change = true;
gateParams.guard_band_gate_event = false;
gateParams.guard_band_hold_event = false;
gateParams.admin_gate_states = 255;
gateParams.admin_control_list[0].operation_name = TSN_ieee802_dot1q_sched_set_gate_states;
gateParams.admin_control_list[0].time_interval_value = 200000;
gateParams.admin_control_list[0].gate_state_value = 0x80;
gateParams.admin_control_list[0].tcu_state_value = 0;
gateParams.admin_control_list[1].operation_name = TSN_ieee802_dot1q_sched_set_gate_states;
gateParams.admin_control_list[1].time_interval_value = 200000;
gateParams.admin_control_list[1].gate_state_value = 0x10;
gateParams.admin_control_list[1].tcu_state_value = 0;
gateParams.admin_control_list[2].operation_name = TSN_ieee802_dot1q_sched_set_gate_states;
gateParams.admin_control_list[2].time_interval_value = 200000;
gateParams.admin_control_list[2].gate_state_value = 0x01;
gateParams.admin_control_list[2].tcu_state_value = 0;
gateParams.admin_control_list[3].operation_name = TSN_ieee802_dot1q_sched_set_gate_states;
gateParams.admin_control_list[3].time_interval_value = 0;
gateParams.admin_control_list[3].gate_state_value = 0x00;
gateParams.admin_control_list[3].tcu_state_value = 0;
gateParams.admin_cycle_time.numerator = 8;
gateParams.admin_cycle_time.denominator = 10000;
gateParams.admin_cycle_time_extension = 0;
gateParams.admin_base_time.seconds = currentTimeSec + 1;
gateParams.admin_base_time.nanoseconds = 0;

rv = SES_QbvSetGateParameters(portNumber, &gateParams); // Set schedule for loopback port

/* Update send list frames */
rv = SES_UpdateRegisteredFrame(frameID_A, 1, sizeof(frameData_A), &frameData_A);
rv = SES_UpdateRegisteredFrame(frameID_B, 1, sizeof(frameData_B), &frameData_B);
rv = SES_UpdateRegisteredFrame(frameID_C, 1, sizeof(frameData_C), &frameData_C);
```

SENDLIST

```
rv = SES_UpdateRegisteredFrame(frameID_D, 1, sizeof(frameData_D), &frameData_D);

/* Start send list */
rv = SES_SendListStart(3200000);

/* Example of stopping one registered frame, to stop a sendlist, call this API for each frame */
rv = SES_StopRegisteredFrame(frameID_A);
```

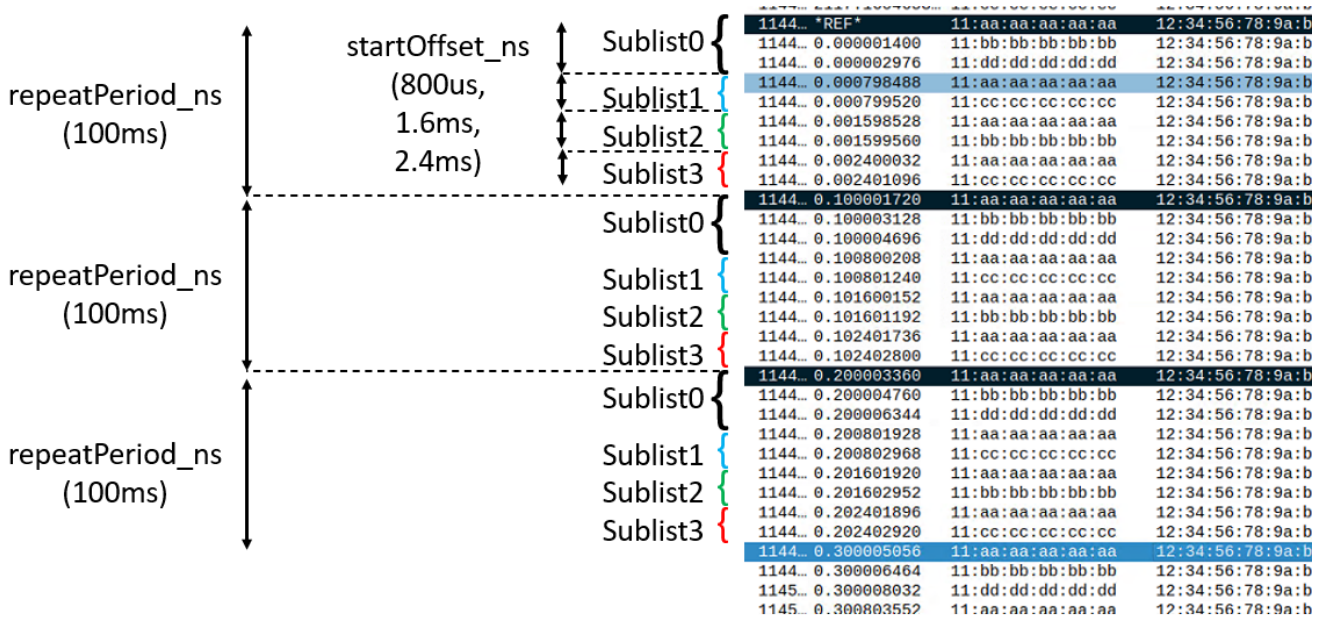

Figure 28. Traffic example showing operation of Sendlist

IGMP SNOOPING

IGMP snooping is a method network switches use to identify multicast groups and forward packets accordingly. Multicast is a one-to-many communication method where data is sent from one source to multiple specific destinations. In a multicast setup, data packets are addressed to a specific group of devices that have expressed interest in receiving the data, making it more efficient than broadcast for targeted communication. As such, multicasting is highly efficient as it reduces unnecessary data transmission and processing with only the hosts that need the data receiving it, thus conserving bandwidth and reducing processing load on uninterested hosts.

IGMP snooping can be used to manage multicast groups. It works by observing Internet Group Management Protocol (IGMP) network traffic and using this information to map the ports of interest in a particular multicast group in order to control traffic flow. The Switch can support IGMP snooping (versions 1, 2 and 3).

IGMP messages are sent by devices informing their intention to join or leave a multicast group. The switch snoops on these messages and maintains an internal map of which ports are members of which IP multicast transmission, ensuring that the multicast traffic is only sent to the hosts that have requested it. The example in Figure 29 shows a scenario where the multicast source out on Port 2 sends IGMP queries, and devices on Port 0 and Port 5 send IGMP reports indicating interest in this multicast group, the switch controls the flow of traffic to ensure only these two ports receive this particular multicast traffic.

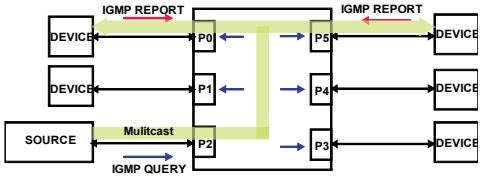

Figure 29. Example of IGMP

IGMP snooping is disabled by default. To use IGMP snooping, enable the function using the APIs included in “SES_igmp.h”.

```
//Enabling IGMP Snooping

void SES_test_IgmpEnable() {

    if (SES_OK != SES_IgmpEnable()) {
        printf("IGMP Snooping Init Error!!\n");
    }else
        printf("IGMP Snooping Start Success!!\n");
}

//Disabling IGMP Snooping
void SES_test_IgmpDisable() {

    if (SES_OK != SES_IgmpDisable()) {
        printf("IGMP Snooping Disable Error!!\n");
    }
    else
        printf("IGMP Snooping Disable Success!!\n");
}
```

ROUTER TIMEOUT

The router timeout is the duration for which a switch considers a multicast router to be present on a particular port. When a switch receives IGMP queries from a router on a port, it marks that port as having an active multicast router.

The purpose of the router timeout is to ensure that the switch doesn't keep forwarding multicast traffic to a port where the multicast router is no longer active. This helps prevent unnecessary flooding of multicast traffic. The default router timeout is 260 seconds and it can be configure between the range of 1 - 300 seconds.

IGMP SNOOPING

GROUP MEMBER TIMEOUT

The group member timeout is the duration for which a switch considers a host to be a member of a particular multicast group. When a switch receives IGMP membership reports from hosts on specific ports, it marks those ports as having active members for the corresponding multicast groups.

The purpose of the group member timeout is to ensure that the switch doesn't keep forwarding multicast traffic to ports where there are no longer interested receivers for a particular multicast group. The default group member timeout is 260 seconds and it can be configured between the range of 30 - 600 seconds.

```
//Updating IGMP Group Timeout
void SES_test_IgmpSetGroupTimeout(uint32_t seconds) {

    if (seconds > 600 || seconds < 30) {
        printf("Wrong timer value :: %d\n", seconds);
    }
    else {
        if (SES_OK != SES_IgmpSetGroupTimeout(seconds)) {
            printf("Set group timeout error !!\n");
        }else
            printf("Set group timeout success !!\n");
    }
}

//Updating Router Timeout
void SES_test_IgmpSetRouterTimeout(uint32_t seconds) {
if (seconds > 300 || seconds < 1)

        printf("Wrong timer value :: %d\n", seconds);
    }
    else {
        if (SES_OK != SES_IgmpSetRouterTimeout(seconds)) {
            printf("Set router timeout error !!\n");
        }
        else
            printf("Set router timeout success !!\n");
    }
```

```
//Check the status of IGMP Snooping
void SES_test_IgmpGetStatus() {
    SES_igmpStatus_t status_p;
    if (SES_IgmpGetStatus(&status_p) == SES_OK) {
        printf("IGMP Status :: %d\n", status_p.enable);
        printf("groupMemberTimeout :: %d\n", status_p.groupMemberTimeout);
        printf("routerTimeout :: %d\n", status_p.routerTimeout);
    }
    else {
        printf("Error in get IGMP status API!!!\n");
    }
}
```

IGMP SNOOPING

IGMP VERSIONS

IGMPv1:

- Basic Operation: Hosts send IGMP Reports to join groups. The switch periodically sends IGMP Queries to verify group memberships.
- Leave Group: Hosts do not explicitly send Leave messages. The switch relies on the absence of Reports to determine if a host has left the group.

IGMPv2:

- Enhanced Operation: Similar to IGMPv1 but includes the ability for hosts to send Leave Group messages, allowing for quicker leave detection.
- Leave Group: Hosts can send IGMP Leave messages, prompting the switch to send a Group-Specific Query to verify if any other hosts are still interested in the group.

IGMPv3:

- Advanced Features: Supports Source-Specific Multicast (SSM), allowing hosts to specify which sources they want to receive traffic from within a multicast group.
- Leave Group: Enhanced leave mechanisms and ability to manage memberships based on specific sources.

IGMP SNOOPING EXAMPLE

With IGMP snooping disabled, multicast traffic will be visible on all ports. IGMP report messages sent from any port to join a multicast group will be ignored and all ports will continue to receive the multicast traffic.

When IGMP snooping is enabled, with router timeout as 40 seconds and group member timeout as 40 seconds, consider the following scenario where an IGMP report is sent from the device connected to port P3 (Ethernet_5) to join a multicast group. At this point, a multicast channel is established between P0 (Ethernet_6) and P3 (Ethernet_5). All multicast traffic entering P0 will be forwarded only to Port 3 and not to the other ports.

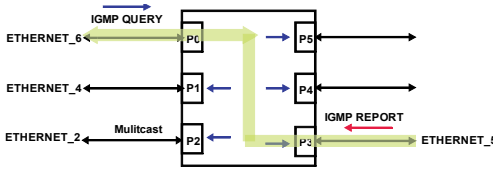

Figure 30. IGMP snooping example

This configuration remains active until the Group Member Timeout is reached, which is 40 seconds in this case, per configuration in Figure 31. If no new IGMP report is received within this period, the switch will delete the multicast group, and any multicast traffic of this group sent into P0 will be treated as broadcast and will be forwarded on all ports.

IGMP SNOOPING

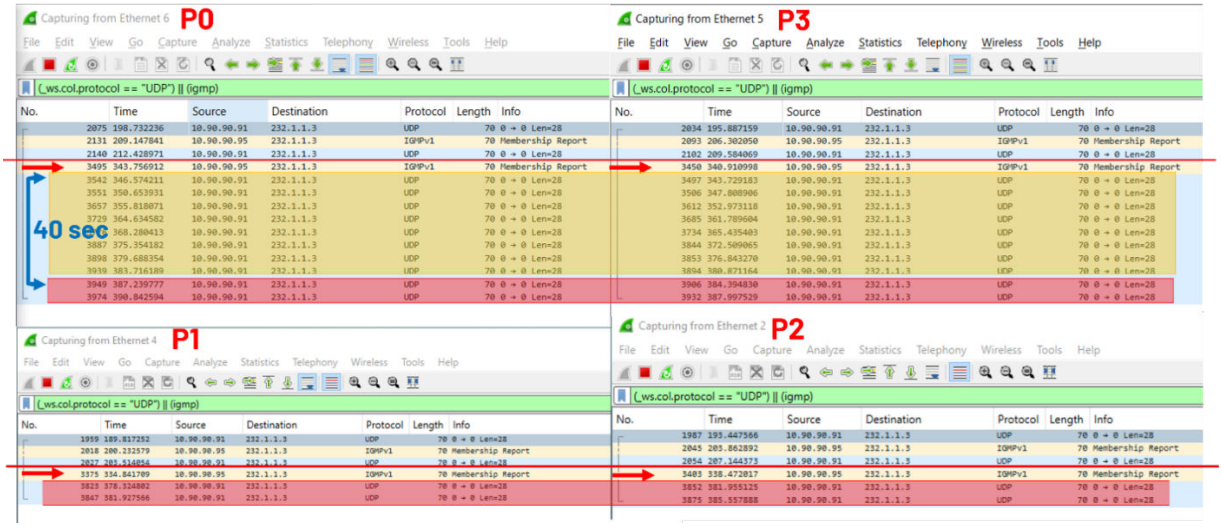

Figure 31. IGMPv1 Multicast Group created & Group Member timeout

The Wireshark screenshots in Figure 31 shows the behavior of the network traffic. The group of messages highlighted in orange occurs within the 40-second timeout period. During this time, multicast traffic sent into P0 is seen only by P3 (Ethernet_5). After the 40-second period, since no further IGMP report was received, the switch deletes the multicast group. The subsequent multicast traffic sent into P0 is treated as broadcast and forwarded on all ports, as highlighted in red.

With IGMPv1, a port will continue receiving multicast traffic until the Group Member Timeout elapses. If a port wishes to stop receiving multicast traffic before the timeout, It is not possible for a device to stop receiving before the timeout with IGMPv1. This limitation is addressed in IGMPv2, which introduces Leave Group messages allowing devices to explicitly signal when they no longer wish to receive multicast traffic.

IGMPv2

The use of IGMP version 2 is demonstrated to show that it is possible for a host to leave a multicast group before the Group Member Timeout. The scenario is shown in Figure 32.

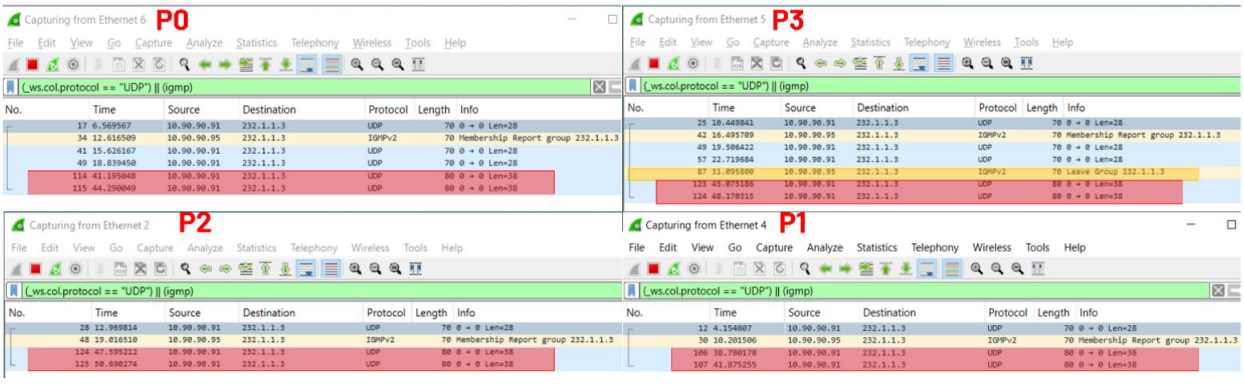

Figure 32. IGMPv2 - Leaving message scenario on port P3

The first UDP packet (length 70) is multicast and sent into port P0 (Ethernet_6) and forwarded on all other ports. A membership report is then sent into port P3 (Ethernet_5), after which two more UDP packets of length 70 are sent into P0. The result is that these multicast UDP packets are seen only on port P3 (Ethernet_5) since it joined the multicast group with an IGMPv2 report. At this point, an IGMPv2 leave message is sent into P3, and another two UDP packets (this time with a length of 80, to distinguish them in Wireshark) are sent again into P0 (Ethernet_6). These packets are visible on all ports because P3 has left the group, and if the switch does not see any members, it treats the packets as broadcast.

IGMP SNOOPING

This behavior shows that IGMPv2 allows a host to dynamically manage its membership in multicast groups, providing greater flexibility and efficiency in network traffic management. By leaving the group, the host ensures it no longer receives unnecessary multicast traffic, thereby optimizing network resources and performance.

IGMPv3

IGMPv3 provides the same functionality as IGMPv2 but also supports Source-Specific Multicast (SSM) which allows a device to join a multicast group and specify from which server it wants to receive traffic. For example, if two servers (10.90.90.1 and 10.90.90.2) are sending multicast traffic to group 239.1.1.1, a device on port P3 of the switch can request traffic only from server 10.90.90.1 when it sends the membership report.

The switch is fully compatible with IGMPv3 queries, report, and leave messages. However, Source-Specific-Multicast selection is not supported.

PER STREAM FILTERING AND POLICING, QCI

Per-stream filtering and policing (PSFP), as defined by IEEE 802.1Qci standard provides filtering policing for a stream.

The purpose of this feature is to prevent traffic overload conditions from affecting the receiving node. This is done by filtering traffic on a per stream basis by providing an input gate for each stream. The input gate serves to enforce a contract between the talker and listener.

PSFP applies to Store and Forward switching as frame size must be known to apply a filter.

The filtering and policing capabilities apply on a stream basis to the receive path.

- Size based filters – 32 per port
- Time based filters – 16 gates per port
- Rate based filters – 8 per port

Any stream can be assigned to any combination of filters and the device can support up to 32 combinations of filters.

PSFP relies on switching table entries to map to a stream filter. Static entries can be installed to support PSFP. Stream table entries with PSFP are not yet supported.

STREAM FILTER

Filtering of frames is based on VLAN Priority (PCP), stream ID and frame size. The filter can be configured based on the incoming frame size. If the frame is over a specified size, it will be discarded. The stream filter can be blocked or unblocked. The stream filter can be associated with a stream gate and flow meter. Frames that fail the filter are discarded. Stream filter statistics are available for frames that pass or fail for size.

The following example shows the approach for applying a stream filter to specific traffic. First, a static table entry is installed with the Stream Filter ID to associate with this particular traffic, the VLAN table is configured for the VLAN ID of the traffic and then the stream filter entry is configured with MaxSDU = 850bytes. All traffic matching the stream filter characteristics and less than MaxSDU will be received by the switch and allowed to forward. Any matching frames in excess of the programmed MaxSDU will be discarded. If the stream block feature was enabled, then any matching ingress frames in excess of the programmed MaxSDU would cause all subsequent frames in this stream to be blocked.

```
int32_t SES_streamFilterExample() {

    int rv;
    int32_t index_p;
    TSN_ieee802_dot1q_psfp_stream_filter_instance_table_config_t streamFilterInstanceTable;

    /* Clear the contents of the structure before configuring */
    memset(&streamFilterInstanceTable, 0, sizeof(TSN_ieee802_dot1q_psfp_stream_filter_instance_ta▶
ble_config_t));

    /* Configure SES to enable stream Filter on Port 2 */
    SES_mac_t mac = SES_macPort2;

    /* priority for ingressing frames (wildcard - 0xFF) */
    uint32_t pcp = 0x04;

    /* Egress port chosen in Port3 of the ADINx310 switch */
    uint8_t portMap = 0x08;

    /* vlan ID of ingressing stream */
    uint16_t VID = 1;

    /* Destination mac address of ingressing stream */
    uint8_t macAddr[6] = { 0x00, 0x00, 0x00, 0x00, 0x22, 0x22 };
    uint8_t macMask[6] = NULL;

    /* Get available stream filter ID for Port2 */
```

PER STREAM FILTERING AND POLICING, QCI

```
    uint8_t rxFilterIndex = SES_PSFP_GetStreamFilterIndex(0x04);

    /* Install entry as Store & Forward required for PSFP function to work */
    uint8_t cutThrough = 0;
    rv = SES_AddStaticTableEntryEx(1, &macAddr, &macMask, VID,
        SES_DYNTBL_VLAN_EXACT_MATCH, 0, 0, 0,
        0, SES_dynamicTblLookupBasic, portMap, 0,
        0, cutThrough, 0, 0,
        dynamicTblNoSequenceOp, 0, SES_IGNORE_FIELD, rxFilterIndex,
        &index_p);

    /* Enable VID 1 for all ports or only for ports of interest) */
    rv = SES_SetVlanMode(VID, 0xFFF);

    streamFilterInstanceTable.stream_filter_instance_id = rxFilterIndex;

    /* pcp, priority of traffic */
    streamFilterInstanceTable.priority_spec_type.priority_spec = pcp;

    streamFilterInstanceTable.stream_handle_spec.stream_handle = rxFilterIndex;

    /* maxSDU = 850 byte */
    streamFilterInstanceTable.max_sdu_size = 850;

    /* blockEnable, disabled */
    streamFilterInstanceTable.stream_blocked_due_to_oversize_frame_enabled = 0;

    /* flowMeterID, no flow meter associated with this stream filter */
    streamFilterInstanceTable.flow_meter_ref = 0xFF;

    /* gateID, no stream gate associated with this stream filter */
    streamFilterInstanceTable.stream_gate_ref = 0xFF;

    rv = SES_PSFP_SetStreamFilterActive(mac, &streamFilterInstanceTable);
    printf("SES_streamFilterExample :: %d\n", rv);

    return rv;
}
```

STREAM GATE

The stream gate is either open or closed. The stream gate monitors the arrival time of frames on that stream and uses the port Timer control unit to control the gate, similar to that of Scheduled traffic on the Transmit side. If a stream arrives when gate is open, accept the frame and perform the required lookups and handle as required. Alternatively, if the stream arrives when the gate is closed, discard the frame. Internal Priority Vector – the stream gate can change the IPV of a frame.

NOTE: The cycle time for the stream gate must match that of Scheduled Traffic for the port.

The following example shows the approach for applying a stream filter and a stream gate to specific traffic. First, a static table entry is installed with the Stream Filter ID to associate with this particular traffic, the VLAN table is configured for the VLAN ID of the traffic. The stream filter entry is configured with a certain MaxSDU size and a stream gate is associated with the stream filter. The parameters of the stream gate in this example open the gate for the first 100 µs of the 1ms cycle time and keep the gate closed for the remaining 900 µs. The IPV value is set to ignore IPV (0xFF), thereby using the existing traffic class for this traffic.

PER STREAM FILTERING AND POLICING, QCI

Using Stream Gate IPV to Reprioritize

Passing a value 0-7 into the IPV field gives user opportunity to reprioritize which traffic class the stream should egress. Pass 0xFF to IPV to leave the stream egress based on the incoming priority. Additional examples for re-prioritization are included in the Using Extended Table to Reprioritize (PSFP Stream Gate) Based on Ethertype section.

```
int32_t SES_streamGateExample() {

    int rv;
    int32_t index_p;
    TSN_ieee802_dot1q_psfp_stream_filter_instance_table_config_t streamFilterInstanceTable;
    TSN_ieee802_dot1q_psfp_stream_gate_instance_table_config_t streamGateInstanceTable;

    /* Clear the contents of the structure before configuring */
    memset(&streamFilterInstanceTable, 0, sizeof(TSN_ieee802_dot1q_psfp_stream_filter_instance_ta▶
ble_config_t));
    memset(&streamGateInstanceTable, 0, sizeof(TSN_ieee802_dot1q_psfp_stream_gate_instance_table_con▶
fig_t));

    /* Configure SES to enable stream Filter on Port 2 */
    SES_mac_t mac = SES_macPort2;

    /* priority for ingressing frames (wildcard - 0xFF) */
    uint32_t pcp = 0x04;

    /* Egress port, Port 3 */
    uint8_t portMap = 0x08;

    /* vlan ID of ingressing stream, VLAN ID 1 */
    uint16_t VID = 1;

    /* Destination mac address of ingressing stream */
    uint8_t macAddr[6] = { 0x00, 0x00, 0x00, 0x00, 0x22, 0x22 };
    uint8_t macMask[6] = NULL;

    /* Get available stream filter ID for Port2 */
    uint8_t rxFilterIndex = SES_PSFP_GetStreamFilterIndex(0x04);

    /* Get available stream gate ID for Port2 */
    uint8_t rxStreamGateIndex = SES_PSFP_GetStreamGateIndex(0x04);

    /* Install entry as Store & Forward required for PSFP function to work */
    uint8_t cutThrough = 0;

    /* 0xFF indicates to use existing traffic class for traffic */
    uint8_t ipv = 0xFF;

    rv = SES_AddStaticTableEntryEx(1, &macAddr, &macMask, VID,
        SES_DYNTBL_VLAN_EXACT_MATCH, 0, 0, 0,
        0, SES_dynamicTblLookupBasic, portMap, 0,
        0, cutThrough, 0, 0,
        dynamicTblNoSequenceOp, 0, SES_IGNORE_FIELD, rxFilterIndex,
        &index_p);

    /* Enable VID 1 for all ports or only for ports of interest */
```

PER STREAM FILTERING AND POLICING, QCI

```
    rv = SES_SetVlanMode(VID, 0xFFF);

    /* Configure the stream filter */

    /* Stream filter Index */
    streamFilterInstanceTable.stream_filter_instance_id = rxFilterIndex;

    /* pcp, traffic class */
    streamFilterInstanceTable.priority_spec_type.priority_spec = pcp;

    /* streamID */
    streamFilterInstanceTable.stream_handle_spec.stream_handle = rxFilterIndex;

    /* MaxSDU = 850 bytes */
    streamFilterInstanceTable.max_sdu_size = 850;

    /* blockEnable */
    streamFilterInstanceTable.stream_blocked_due_to_oversize_frame_enabled = 0;

    /* flowMeterID, 0xFF indicates no flowmeter associated */
    streamFilterInstanceTable.flow_meter_ref = 0xFF;

    /* gateID */
    streamFilterInstanceTable.stream_gate_ref = rxStreamGateIndex;

    /* Close gate if Octets Exceeded, disabled */
    streamGateInstanceTable.gate_closed_due_to_octets_exceeded_enable = 0;

    /* Close gate if frames received while gate is closed, disabled */
    streamGateInstanceTable.gate_closed_due_to_invalid_rx_enable = false;

    /* Open all the gates */
    streamGateInstanceTable.admin_cycle_time.numerator = 1;

    /* 1ms Cycle Time */
    streamGateInstanceTable.admin_cycle_time.denominator = 1000;
    streamGateInstanceTable.config_change = true;

    /* Priority for ingressing frames, pass 0xFF as wildcard */
    streamGateInstanceTable.admin_ipv = 0x4;
    streamGateInstanceTable.gate_enable = true;
    streamGateInstanceTable.admin_gate_states = true;
    streamGateInstanceTable.admin_control_list_length = 2;

    /* octetInterval1Enable */
    streamGateInstanceTable.admin_control_list[0].octet_interval_enable = 0;

    /* ipv set, use to reprioritize traffic class to different gate */
    streamGateInstanceTable.admin_control_list[0].ipv_spec = ipv;

    /* gateOpen */
    streamGateInstanceTable.admin_control_list[0].gate_state_value = 1;

    /* Time interval value, 100us */
    streamGateInstanceTable.admin_control_list[0].time_interval_value = 100000;
```

PER STREAM FILTERING AND POLICING, QCI

```
    /* Interval Octet Max */
    streamGateInstanceTable.admin_control_list[0].interval_octet_max = 0;

    /* octetInterval1Enable */
    streamGateInstanceTable.admin_control_list[1].octet_interval_enable = 0;

    /* ipv set, use to reprioritize traffic class to different gate */
    streamGateInstanceTable.admin_control_list[1].ipv_spec = ipv;

    /* gate closed */
    streamGateInstanceTable.admin_control_list[1].gate_state_value = 0;

    /* Time Interval value, 900us */
    streamGateInstanceTable.admin_control_list[1].time_interval_value = 900000;

    /* Interval Octet Max */
    streamGateInstanceTable.admin_control_list[1].interval_octet_max = 0;
    rv = SES_PSFP_SetStreamFilterActive(mac, &streamFilterInstanceTable);
    rv = SES_PSFP_SetStreamGateActive(mac, &streamGateInstanceTable);
    printf("SES_streamGateExample :: %d\n", rv);

    return rv;
}
```

FLOW METER

Stream data per unit time, allows a certain amount of traffic through the port. This feature uses a token bucket or a bandwidth profile where it compares a frame size to how many tokens in a bucket (commit, extended), if smaller it can proceed.

The following example shows the approach for applying a stream filter and a stream gate to specific traffic. First, a static table entry is installed with the Stream Filter ID to associate with this particular traffic, the VLAN table is configured for the VLAN ID of the traffic. The stream filter entry is configured with a certain MaxSDU size and a flow meter is associated with the stream filter and a flowmeter configured for 100kbps using just the commit bucket.

```
int32_t SES_flowMeterExample() {

    int rv;
    int32_t index_p;
    TSN_ieee802_dot1q_psfp_stream_filter_instance_table_config_t streamFilterInstanceTable;
    TSN_ieee802_dot1q_psfp_flow_meter_instance_table_config_t flowMeterInstanceTable;

    /* Clear the contents of the structures before configuring */
    memset(&streamFilterInstanceTable, 0, sizeof(TSN_ieee802_dot1q_psfp_stream_filter_instance_ta▶
ble_config_t));
    memset(&flowMeterInstanceTable, 0, sizeof(TSN_ieee802_dot1q_psfp_flow_meter_instance_table_con▶
fig_t));

    /* Configure SES to enable stream Filter on Port 2 */
    SES_mac_t mac = SES_macPort2;

    /* priority for ingressing frames (wildcard - 0xFF) */
    uint32_t pcp = 0x04;

    /* Egress port, Port 3 */
    uint8_t portMap = 0x08;
```

PER STREAM FILTERING AND POLICING, QCI

```
    /* vlan ID of ingressing stream */
    uint16_t VID = 1;

    /* Destination mac address of ingressing stream */
    uint8_t macAddr[6] = { 0x00, 0x00, 0x00, 0x00, 0x22, 0x22 };
    uint8_t macMask[6] = NULL;

    /* Get available stream filter, gate and Flowmeter IDs for Port 2 */
    uint8_t rxFilterIndex = SES_PSFP_GetStreamFilterIndex(0x4);
    uint8_t rxFlowMeterIndex = SES_PSFP_GetFlowMeterIndex(0x4);
    uint8_t rxStreamGateIndex = SES_PSFP_GetStreamGateIndex(0x04);

    /* Install entry as Store & Forward required for PSFP function to work */
    uint8_t cutThrough = 0;

    /* Set port flow meter wait time, hardware default 80 (0x50), 10000 or 0x2710 */
    uint16_t waitTime = 0x2710;
    rv = SES_AddStaticTableEntryEx(1, &macAddr, &macMask, VID, SES_DYNTBL_VLAN_EXACT_MATCH, 0, 0,
0, 0, SES_dynamicTblLookupBasic, portMap, 0, 0, cutThrough, 0, 0, dynamicTblNoSequenceOp, 0, SES_IG▶
NORE_FIELD, rxFilterIndex, &index_p);

    /* Enable VID 1 for all ports or only for ports of interest) */
    rv = SES_SetVlanMode(VID, 0xFFF);

    /* Configure the stream filter */
    streamFilterInstanceTable.stream_filter_instance_id = rxFilterIndex;

    /*  pcp, priority of traffic */
    streamFilterInstanceTable.priority_spec_type.priority_spec = pcp;

    streamFilterInstanceTable.stream_handle_spec.stream_handle = rxFilterIndex;

    /* maxSDU = 850 bytes */
    streamFilterInstanceTable.max_sdu_size = 850;

    /* blockEnable, disabled */
    streamFilterInstanceTable.stream_blocked_due_to_oversize_frame_enabled = 0;

    streamFilterInstanceTable.flow_meter_ref = rxFlowMeterIndex;

    /* stream gateID, no gate associated (0xFF) */
    streamFilterInstanceTable.stream_gate_ref = 0xFF;

    /* Configure the flow meter
    * CIR register bytes per flow meter period (flow meter waitTime*8ns),
    * for waitTime = 80 (hardware default), a CIR value = 1 corresponds to 12.5Mbps
    * for waitTime = 10000, a CIR value = 1 corresponds to 100kbps
    */

        /* flowMeterIndex */
        flowMeterInstanceTable.flow_meter_instance_id = rxFlowMeterIndex;

    /* Rate at which tokens are added to commit bucket */
    flowMeterInstanceTable.committed_information_rate = 1;
```

PER STREAM FILTERING AND POLICING, QCI

```
    /* Maximum capacity of commit bucket */
    flowMeterInstanceTable.comitted_burst_size = 1500;
    flowMeterInstanceTable.commit_bucket_tokens = 0;

    /* Rate at which tokens are added to excess bucket */
    flowMeterInstanceTable.excess_information_rate = 0;

    /* Maximum capacity of excess bucket */
    flowMeterInstanceTable.excess_burst_size = 0;

    /* Color mode, 0= Color Aware, 1 = color blind */
    flowMeterInstanceTable.color_mode = 0;

    /* No coupling of commit overflow tokens to excess bucket */
    flowMeterInstanceTable.coupling_flag = 0;

    /* Drop on Yellow, false = forward yellow frames, true = drop yellow frames */
    flowMeterInstanceTable.drop_on_yellow = false;
    rv = SES_PSFP_SetFlowMeterWaitTime(mac, waitTime);
    rv = SES_PSFP_SetFlowMeterActive(mac, &flowMeterInstanceTable);
    rv = SES_PSFP_SetStreamFilterActive(mac, &streamFilterInstanceTable);

    printf("SES_flowMeterExample :: %d\n", rv);

    return rv;
}
```

READING PSFP STATISTICS

PSFP statistics are available to read for the stream filter, stream gate and flowmeter using the API call SES_PSFP_GetStats() for the port of interest. All available statistics information is returned for all supported 32 stream filters, 16 stream gates and 8 flow meters. For the stream filter, the statistics report the number of frames passing and not passing the MaxSDU count and the count of frames which match the stream filter. For the stream gate, the statistics report the number of frames passing and not passing the stream gate. The statistics reported for the flowmeter are the count of frames that do not pass the flow meter (Red frames).

```
int32_t sesReadPsfpStatistics_Example(void) {
  int rv;
  SES_mac_t mac = SES_macPort2;
  SES_psfpStatsFilterSdu_t destFilterSdu;
  SES_psfpStatsGate_t destGate;
  SES_psfpStatsFilterMatchMeter_t destFilterMatchMeter;
  rv = SES_PSFP_GetStats(mac, &destFilterSdu, &destGate, &destFilterMatchMeter);
  printf("SES_readPsfpStatistics :: %d rv);

  printf("=== PSFP Statistics ===\n");

  printf("-- Stream Filter SDU Stats --\n");
  for (int i = 0; i < SES_PSFP_STREAM_NUM_MAX; i++) {
    if (destFilterSdu.filterSduPassCount[i] || destFilterSdu.filterSduNotPassCount[i]) {
      printf(" Stream[%2d]: pass=%u, drop=%u\n",
        i,
        destFilterSdu.filterSduPassCount[i],
      destFilterSdu.filterSduNotPassCount[i]);
    }
  }
```

PER STREAM FILTERING AND POLICING, QCI

```
  printf("-- Stream Gate Stats --\n");
  for (int i = 0; i < SES_PSFP_GATE_NUM_MAX; i++) {
    if (destGate.gatePassCount[i] || destGate.gateNotPassCount[i]) {
    printf(" Gate[%2d]: pass=%u, drop=%u\n",
      i,
      destGate.gatePassCount[i],
      destGate.gateNotPassCount[i]);
    }
  }

  printf("-- Filter Match / Flow Meter Discard Stats --\n");
  for (int i = 0; i < SES_PSFP_STREAM_NUM_MAX; i++) {
    if (destFilterMatchMeter.filterMatchCount[i]) {
      printf(" Stream[%2d]: filterMatch=%u\n",
        i,
        destFilterMatchMeter.filterMatchCount[i]);
    }
  }
  for (int i = 0; i < SES_PSFP_METER_NUM_MAX; i++) {
    if (destFilterMatchMeter.meterDiscardCount[i]) {
      printf(" Meter[%2d]: meterDiscard=%u\n",
      i,
      destFilterMatfchMeter.meterDiscardCount[i]);
    }
  }
  return rv;
}
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

Frame Replication and Elimination for Reliability (FRER) aims to improve network reliability by reducing packet loss due to equipment failures as defined by the IEEE 802.1CB standard. The Switch supports the replication of frames in the talker (source) and elimination of frames in the listener (destination). The talker sends replicated stream along two or more redundant paths with a redundancy tag added to the frame. The point of having these redundant paths is to minimize packet loss due to link failure, device failure, or stream congestion. The listener is then responsible for eliminating the duplicate packet and removing the redundancy tag. The network ensures that no matter which of the paths the stream takes, it arrives where and when it was supposed to. The Switch includes the sequence generation and recovery algorithms for FRER support.

The following descriptions provide an overview of FRER to help described the functionality provided by Switch and does not intend to be a full overview of the FRER protocol. The full overview of FRER is available in the standard. Details of the FRER APIs are included in the 'SES_frer.h' header file.

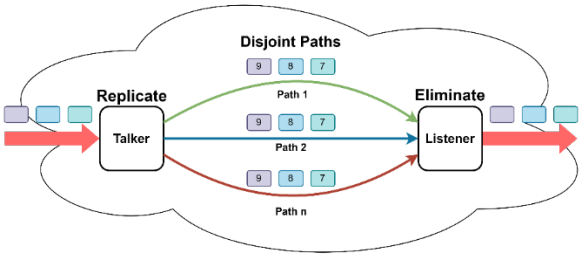

Figure 33. FRER overview

LOOP PROTECTION

The switch requires a loop protection to prevent non-FRER traffic from creating a storm when there are multiple paths to another switch. There are two approach that can be implemented to break traffic from looping, using the MSTP APIs or configuring the miss return APIs.

Approach 1: MSTP

MSTP is disabled by default and can be enabled using the SES_MstpStart API. When MSTP is enabled, it will manage the redundant paths from one switch to another to prevent traffic from looping. MSTP manages all VLANs by default, the user must exclude any VLAN used with FRER. SES_MstpExcludeVlanFromSpanningTree can be used to exclude VLAN from the spanning tree state. Once a VLAN has been excluded from the spanning tree, there is currently no way to include them again to the spanning tree. The MSTP section discusses more details about MSTP.

The example demonstrates VLAN 100 excluded from the spanning tree. The example assumes MSTP is enabled.

```
int32_t rv = 0;

/* Exclude VLAN 100 from Spanning Tree */
rv = SES_MstpExcludeVlanFromSpanningTree(100);
printf("SES_MstpExcludeVlanFromSpanningTree :: %d\n", rv);
```

Approach 2: Miss Return

The default behavior of the switch is to forward unknown unicast/multicast traffic on all ports. The SES_SetUnicastMissReturn and SES_SetMulticastMissReturn can be used to configure these behavior. The example below assumes that Port 1 and Port 2 of the switch acts as the redundant paths. The current unicast/multicast miss return of a redundant path is retrieved and modified to prevent non-FRER traffic from being forwarded to the other redundant path. The user must implement accordingly the miss returns based on there network.

NOTE: Implement loop protection immediately after initializing the ports of the switch.

```
int32_t rv = 0;
uint8_t portMap;
bool cutThrough;
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

```
uint8_t lookupDone;
uint8_t txFilter;
bool txFilterValid;
uint8_t rxFilter;
bool rxFilterValid;
/* Retrieve the current unicast miss return configuration for Port 1 */
rv = SES_GetUnicastMissReturn(SES_macPort1, &portMap, &cutThrough, &lookupDone, &txFilter, &txFilter▶
Valid, &rxFilter, &rxFilterValid);
if (rv == SES_PORT_OK) {
    /* Clear the portMap for port 2 (Bit 2)
    Prevents unicast traffic ingressing Port 1 to be forwarded on Port 2*/
    portMap = portMap & ~(1 << 2);
    rv = SES_SetUnicastMissReturn(SES_macPort1, portMap, cutThrough, lookupDone, txFilter, txFilterVal▶
id, rxFilter, rxFilterValid);
    printf("SES_SetUnicastMissReturn :: %d\n", rv);
}
/* Retrieve the current multicast miss return configuration for Port 1 */
rv = SES_GetMulticastMissReturn(SES_macPort1, &portMap, &cutThrough, &lookupDone, &txFilter, &txFilter▶
Valid, &rxFilter, &rxFilterValid);
if (rv == SES_PORT_OK) {
    /* Clear the portMap for port 2 (Bit 2) -
    Prevents multicast traffic ingressing Port 1 to be forwarded on Port 2*/
    portMap = portMap & ~(1 << 2);
    rv = SES_SetMulticastMissReturn(SES_macPort1, portMap, cutThrough, lookupDone, txFilter, txFilter▶
Valid, rxFilter, rxFilterValid);
    printf("SES_SetMulticastMissReturn :: %d\n", rv);
}

/* Retrieve the current unicast miss return configuration for Port 2 */
rv = SES_GetUnicastMissReturn(SES_macPort2, &portMap, &cutThrough, &lookupDone, &txFilter, &txFilter▶
Valid, &rxFilter, &rxFilterValid);
if (rv == SES_PORT_OK) {
    /* Clear the portMap for port 1 (Bit 1)
    Prevents unicast traffic ingressing Port 2 to be forwarded on Port 1*/
    portMap = portMap & ~(1 << 1);
    rv = SES_SetUnicastMissReturn(SES_macPort2, portMap, cutThrough, lookupDone, txFilter, txFilterVal▶
id, rxFilter, rxFilterValid);
    printf("SES_SetUnicastMissReturn :: %d\n", rv);
}
/* Retrieve the current multicast miss return configuration for Port 2 */
rv = SES_GetMulticastMissReturn(SES_macPort2, &portMap, &cutThrough, &lookupDone, &txFilter, &txFilter▶
Valid, &rxFilter, &rxFilterValid);
if (rv == SES_PORT_OK) {
    /* Clear the portMap for port 1 (Bit 1)
    Prevents multicast traffic ingressing Port 2 to be forwarded on Port 1*/
    portMap = portMap & ~(1 << 1);
    rv = SES_SetMulticastMissReturn(SES_macPort2, portMap, cutThrough, lookupDone, txFilter, txFilter▶
Valid, rxFilter, rxFilterValid);
    printf("SES_SetMulticastMissReturn :: %d\n", rv);
}
```

STREAM IDENTIFICATION

The stream identification function is used to determine which ports the streams would go. There are two types of stream identification active or passive. Passive identification only examines the packets of the stream, while active identification modifies data parameters of the

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

packet to be transmitted. The Switch supports different types of stream identification. Details about stream identification can be found in the 'TSN_ieee802_dot1cb_stream.h' header file.

Note: When configuring multiple streams, it is possible to use the Stream Table by enabling it during configuration. The stream table reduces the number of lookup entries for handling multiple streams. Enabling the stream table groups streams together in a single block based on the base MAC address, which is defined by the first 38 bits of the MAC address. The stream table supports up to 16 stream blocks with 1k streams each.

The following examples for Stream Table and FRER functionality use these definitions:

```
#define SES_PORT_0 0
#define SES_PORT_1 1
#define SES_PORT_2 2
#define SES_PORT_3 3
#define SES_PORT_4 4
#define SES_PORT_5 5
#define SES_PORTMAP_PORT_0 1
#define SES_PORTMAP_PORT_1 2
#define SES_PORTMAP_PORT_2 4
#define SES_PORTMAP_PORT_3 8
#define SES_PORTMAP_PORT_4 16
#define SES_PORTMAP_PORT_5 32
```

Null Stream

```
void null_stream_example(void) {

    TSN_ieee802Dot1cbStream_t streamParameters;

    uint8_t destinationMAC[6] = {0x00, 0x00, 0x00, 0x11, 0x11, 0x11};


        /*Null stream identification*/
    streamParameters.TSN_streamIdentity.identificationType = TSN_nullStream;


        /*Assigning Desination MAC address*/
    for (int i = 0; i < 6; i++) {
streamParameters.TSN_streamIdentity.TSN_parameters.TSN_nullStreamIdentification.destinationMac[i] =
destinationMAC[i];
    }

        /*VLAN Identifier*/
    streamParameters.TSN_streamIdentity.TSN_parameters.TSN_nullStreamIdentification.vlan = 100;


        /*Port MAP for Egress traffic*/
    streamParameters.portMap = SES_PORTMAP_PORT_1 | SES_PORTMAP_PORT_2;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[0] = -1;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[1] = -1;
    streamParameters.TSN_HardwareTableIndices.exTblruleId = -1;
    streamParameters.useStreamtable = false;
    streamParameters.usedCnt = 0;
}
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

Source MAC and VLAN Stream

```
void smac_vlan_example(void) {

    uint8_t destinationMAC[6] = { 0x00, 0x00, 0x00, 0x22, 0x22, 0x22 };
    TSN_ieee802Dot1cbStream_t streamParameters;


        /*Source MAC and VLAN stream identification*/
    streamParameters.TSN_streamIdentity.identificationType = TSN_smacVlan;


        /*Assigning Desination MAC address*/
    for (int i = 0; i< 6; i++) {
        streamParameters.TSN_streamIdentity.TSN_parameters.TSN_nullStreamIdentification.destination▶
Mac[i] = destinationMAC[i];
    }

        /*VLAN Identifier*/    streamParameters.TSN_streamIdentity.TSN_parameters.TSN_smacVlanStreamI▶
dentification.vlan = 100;


        /*Port MAP for Egress traffic*/
    streamParameters.portMap = SES_PORTMAP_PORT_1 | SES_PORTMAP_PORT_2; // bit field mapped to physi▶
cal egress ports for stream
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[0] = -1;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[1] = -1;
    streamParameters.TSN_HardwareTableIndices.exTblruleId = -1;
    streamParameters.useStreamtable = false;
    streamParameters.usedCnt = 0;
}
```

Active Destination MAC and VLAN Stream

```
void dmac_vlan_example(void) {

    TSN_ieee802Dot1cbStream_t streamParameters;
    uint8_t downDestinationMAC[6] = { 0x00, 0x00, 0x00, 0x33, 0x33, 0x33 };
    uint8_t upDestinationMAC[6] = { 0x00, 0x00, 0x44, 0x44, 0x44, 0x44 };


        /* Active Destination MAC and VLAN stream identification */
    streamParameters.TSN_streamIdentity.identificationType = TSN_dmacVlan;


        /* Assigning Down Desination MAC address */
    for (int i = 0; i < 6; i++) {
        streamParameters.TSN_streamIdentity.TSN_parameters.TSN_dmacVlanStreamIdentifica▶
tion.TSN_down.destinationMac[i] = downDestinationMAC[i];
    }
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

```
        /* Assigning UP Desination MAC address */
    for (int i = 0; i < 6; i++) {
        streamParameters.TSN_streamIdentity.TSN_parameters.TSN_dmacVlanStreamIdentification.TSN_up.des▶
tinationMac[i] = upDestinationMAC[i];
    }

/*VLAN Identifier*/
streamParameters.TSN_streamIdentity.TSN_parameters.TSN_dmacVlanStreamIdentification.TSN_down.vlan =
100;
    streamParameters.TSN_streamIdentity.TSN_parameters.TSN_dmacVlanStreamIdentification.TSN_up.vlan =
200;
    streamParameters.TSN_streamIdentity.TSN_parameters.TSN_dmacVlanStreamIdentification.TSN_up.priori▶
ty = 1;


        /* bit field mapped to physical egress ports for stream */
    streamParameters.portMap = SES_PORTMAP_PORT_1 | SES_PORTMAP_PORT_2;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[0] = -1;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[1] = -1;
    streamParameters.TSN_HardwareTableIndices.exTblruleId = -1;
    streamParameters.useStreamtable = false;
    streamParameters.usedCnt = 0;

}
```

IP Stream

```
void ip_example(void) {

    TSN_ieee802Dot1cbStream_t streamParameters;
    uint8_t destinationMAC[6] = { 0x00, 0x11, 0x11, 0x11, 0x11, 0x11 };
    int ipSource[4] = {192,168,1,3};
    int ipDestination[4] = { 192,168,1,15 };


        /* IP stream identification */
    streamParameters.TSN_streamIdentity.identificationType = TSN_ip;


        /* Assigning IP address */
    for (int i = 0; i < 6; i++) {
        streamParameters.TSN_streamIdentity.TSN_parameters.TSN_ipStreamIdentification.destination▶
Mac[i] = destinationMAC[i];
    }

    streamParameters.TSN_streamIdentity.TSN_parameters.TSN_ipStreamIdentification.vlan=100;

    for (int i = 0; i < 4; i++) {
        streamParameters.TSN_streamIdentity.TSN_parameters.TSN_ipStreamIdentification.ip▶
Source.TSN_ipv4Address[i] = ipSource[i];
        streamParameters.TSN_streamIdentity.TSN_parameters.TSN_ipStreamIdentification.ipDestina▶
tion.TSN_ipv4Address[i] = ipDestination[i];
    }

    streamParameters.TSN_streamIdentity.TSN_parameters.TSN_ipStreamIdentification.dscp=0x00;
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

```
streamParameters.TSN_streamIdentity.TSN_parameters.TSN_ipStreamIdentification.nextProtocol =
TSN_udp;    streamParameters.TSN_streamIdentity.TSN_parameters.TSN_ipStreamIdentification.sourcePort =
0x0000;
streamParameters.TSN_streamIdentity.TSN_parameters.TSN_ipStreamIdentification.destinationPort = 0x0000;
streamParameters.TSN_streamIdentity.TSN_parameters.TSN_ipStreamIdentification.ipv4 = true;


        /* bit field mapped to physical egress ports for stream */
    streamParameters.portMap = SES_PORTMAP_PORT_1 | SES_PORTMAP_PORT_2;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[0] = -1;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[1] = -1;
    streamParameters.TSN_HardwareTableIndices.exTblruleId = -1;
    streamParameters.useStreamtable = false;
    streamParameters.usedCnt = 0;
}
```

Mask-and-match Stream

```
void mask_and_match_example(void) {

    TSN_ieee802Dot1cbStream_t streamParameters;
    TSN_ieee802Dot1cbMaskAndMatch_t maskAndMatchParameters;
    uint8_t MaskMAC[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
    uint8_t destinationMACMatch[6] = { 0x00, 0x66, 0x66, 0x66, 0x66, 0x66 };
    uint8_t sourceMACMatch[6] = { 0x00, 0x77, 0x77, 0x77, 0x77, 0x77 };

        /* Mask and match stream identification */
    streamParameters.TSN_streamIdentity.identificationType = TSN_maskAndMatch;
    streamParameters.TSN_streamIdentity.TSN_parameters.TSN_maskAndMatchIdentification.TSN_identifica▶
tionTtype.typeNumber = TSN_maskAndMatch;
        /* Assigning Source and Desination MAC mask */
    for (int i = 0; i < 6; i++) {
        streamParameters.TSN_streamIdentity.TSN_parameters.TSN_maskAndMatchIdentification.destination▶
MacMask[i] = MaskMAC[i];
        streamParameters.TSN_streamIdentity.TSN_parameters.TSN_maskAndMatchIdentification.sourceMac▶
Mask[i] = MaskMAC[i];
    }
        /* Assigning Desination MAC address */
    for (int i = 0; i < 6; i++) {
        streamParameters.TSN_streamIdentity.TSN_parameters.TSN_maskAndMatchIdentification.destination▶
MacMatch[i] = destinationMACMatch[i];
    }

        /* Assigning Source MAC address */
    for (int i = 0; i < 6; i++) {streamParameters.TSN_streamIdentity.TSN_parameters.TSN_maskAndMatchI▶
dentification.sourceMacMatch[i] = sourceMACMatch[i];
    }


        /* msduMatch size (2) + buffer (2) */    streamParameters.TSN_streamIdentity.TSN_parame▶
ters.TSN_maskAndMatchIdentification.msduMaskLength = 2 + 2;


        /* bit field mapped to physical egress ports for stream */
    streamParameters.portMap = SES_PORTMAP_PORT_1 | SES_PORTMAP_PORT_2;
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

```
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[0] = -1;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[1] = -1;
    streamParameters.TSN_HardwareTableIndices.exTblruleId = -1;
    streamParameters.useStreamtable = false;
    streamParameters.usedCnt = 0;

    maskAndMatchParameters.msduMask[0] = 0xFF;
    maskAndMatchParameters.msduMask[1] = 0xFF;
    maskAndMatchParameters.msduMask[2] = 0x00;
    maskAndMatchParameters.msduMask[3] = 0x00;
        /* Frame Ethertype - msduMatch[0:1] */
    maskAndMatchParameters.msduMatch[0] = 0x12;
    maskAndMatchParameters.msduMatch[1] = 0x34;
    maskAndMatchParameters.msduMatch[2] = 0x00;
    maskAndMatchParameters.msduMatch[3] = 0x00;
}
```

TALKER CONFIGURATION

The SES_FrerConfigureTalkerProxyForStream() API can be used to configure the Switch as a talker proxy. This allows certain ports of the Switch to be used for a talker system. In the example, a null stream identification is configured. The port map of the stream is configured to egress the traffic out Port 1 and Port 2. The Sequence generation is set to Port 0. Traffic that ingress this port will be replicated and egress with the redundancy tag added. The SES_FrerRemoveTalkerProxyForStream() API can be used to remove the talker proxy.

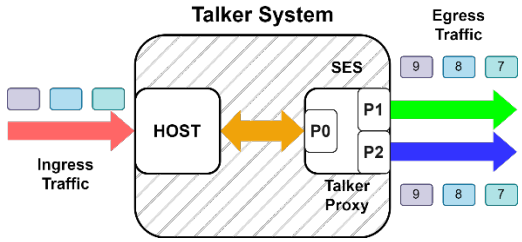

Figure 34. FRER Talker Configuration

```
/* Enable Loop Protection before configuring Talker System */
int32_t ses_frer_talker_example(void) {
    int32_t rv = 0;

    uint8_t destinationMAC[6] = { 0x00, 0x00, 0x00, 0x11, 0x11, 0x11 };

    TSN_ieee802Dot1cbFrer_t frerParameters;
    TSN_ieee802Dot1cbStream_t streamParameters;
    TSN_ieee802Dot1cbMaskAndMatch_t maskAndMatchParameters;

    memset(&frerParameters, 0, sizeof(TSN_ieee802Dot1cbFrer_t));
    memset(&streamParameters, 0, sizeof(TSN_ieee802Dot1cbStream_t));
    memset(&maskAndMatchParameters, 0, sizeof(TSN_ieee802Dot1cbMaskAndMatch_t));

        /* Stream table Parameters - Null Stream Identification */
    streamParameters.TSN_streamIdentity.identificationType = TSN_nullStream; // Null stream identifica▶
tion


        /* Assigning Desination MAC address */
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

```
    for (int i = 0; i < 6; i++) {
        streamParameters.TSN_streamIdentity.TSN_parameters.TSN_nullStreamIdentification.destination▶
Mac[i] = destinationMAC[i];
    }

        /* VLAN ID 100 */
    streamParameters.TSN_streamIdentity.TSN_parameters.TSN_nullStreamIdentification.vlan = 100;
        /* bit field mapped to physical egress ports for stream */
    streamParameters.portMap = SES_PORTMAP_PORT_1 | SES_PORTMAP_PORT_2;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[0] = -1;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[1] = -1;
    streamParameters.TSN_HardwareTableIndices.exTblruleId = -1;
    streamParameters.useStreamtable = false;
    streamParameters.usedCnt = 0;

        /* frer parameters */
    frerParameters.TSN_sequenceGeneration.reset = false;
    frerParameters.TSN_sequenceRecovery.individualRecovery = false;

        /* Port where stream split is applied - Ingress traffic goes to this port */
    frerParameters.TSN_streamSplit.port = SES_PORT_0;

    frerParameters.TSN_HardwareTableIndices.individualRecoveryTblIdx = -1;
    frerParameters.TSN_HardwareTableIndices.sequenceGenerationTblIdx = -1;
    frerParameters.TSN_HardwareTableIndices.sequenceRecoveryTblIdx = -1;
    frerParameters.TSN_HardwareTableIndices.txTransformTblIdx = -1;

        /* Configure VLAN mode */
    rv = SES_SetVlanMode(100, 0xFFF);
    if (rv == SES_PORT_OK) {
        printf("SES_SetVlanMode :: %d\n", rv);
    }

    rv = SES_FrerConfigureTalkerProxyForStream(&frerParameters, &streamParameters, &maskAndMatchParame▶
ters);

        printf("ses_frer_talker_example :: %d\n\r", rv);
    return rv;
}
```

LISTENER CONFIGURATION

The SES_FrerConfigureListenerProxyForStream() API can be used to configure the switch as a listener proxy. This allows certain ports of the switch to be used for a listener system. In the example, a null stream identification is configured. The port map of the stream is configured to egress the traffic out Port 0. The Switch supports both types of Sequence Recovery algorithm: Vector Recovery algorithm and Match Recovery algorithm. The example shows the Sequence Recovery set to the Vector Recovery algorithm. The SES_FrerRemoveListenerProxyForStream() API can be used to remove the listener proxy.

Note: Latent Error Detection is not yet supported in this software release.

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

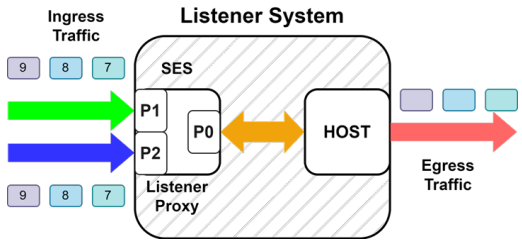

Figure 35. FRER Listener Configuration

```
/* Enable Loop Protection before configuring Listener System */
int32_t ses_frer_listener_example(void) {
    int32_t rv = 0;
    uint8_t destinationMAC[6] = { 0x00, 0x00, 0x00, 0x11, 0x11, 0x11 };
    TSN_ieee802Dot1cbFrer_t frerParameters;
    TSN_ieee802Dot1cbStream_t streamParameters;
    TSN_ieee802Dot1cbMaskAndMatch_t maskAndMatchParameters;

    memset(&frerParameters, 0, sizeof(TSN_ieee802Dot1cbFrer_t));
    memset(&streamParameters, 0, sizeof(TSN_ieee802Dot1cbStream_t));
    memset(&maskAndMatchParameters, 0, sizeof(TSN_ieee802Dot1cbMaskAndMatch_t));

       /* Stream table Parameters - Null Stream Identification */
    streamParameters.TSN_streamIdentity.identificationType = TSN_nullStream;

       /* Assigning Desination MAC address */
    for (int i = 0; i < 6; i++) {
        streamParameters.TSN_streamIdentity.TSN_parameters.TSN_nullStreamIdentification.destination▶
Mac[i] = destinationMAC[i];
    }
       /* VLAN ID 100 */
    streamParameters.TSN_streamIdentity.TSN_parameters.TSN_nullStreamIdentification.vlan = 100;

       /* Egress traffic on port 0 */
    streamParameters.portMap = SES_PORTMAP_PORT_0;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[0] = -1;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[1] = -1;
    streamParameters.TSN_HardwareTableIndices.exTblruleId = -1;
    streamParameters.useStreamtable = false;

       /* Configure frer */
    frerParameters.TSN_sequenceRecovery.algorithm = TSN_vector;
    frerParameters.TSN_sequenceRecovery.historyLength = 5;
    frerParameters.TSN_sequenceRecovery.resetTimeout = 0xffff;
    frerParameters.TSN_sequenceRecovery.takeNoSequence = false;
    frerParameters.TSN_sequenceRecovery.individualRecovery = false;

    frerParameters.TSN_HardwareTableIndices.individualRecoveryTblIdx = -1;
    frerParameters.TSN_HardwareTableIndices.sequenceGenerationTblIdx = -1;
    frerParameters.TSN_HardwareTableIndices.sequenceRecoveryTblIdx = -1;
    frerParameters.TSN_HardwareTableIndices.txTransformTblIdx = -1;

       /* Configure VLAN mode */
    rv = SES_SetVlanMode(100, 0xFFF);
    if (rv != SES_PORT_OK) {
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

```
        printf("SES_SetVlanMode => %d\n", rv);
    }

    rv = SES_FrerConfigureListenerProxyForStream(&frerParameters, &streamParameters, &maskAndMatchPara▶
meters);

       printf("ses_frer_listener_example :: %d\n\r", rv);
    return rv;
}
```

RELAY CONFIGURATION

The SES_FrerConfigureRelaySystemForStream() API can be used to configure the switch as a relay system. As a relay system the Switch can insert tag, remove tag, or no operation to incoming streams. In the example, a null stream identification is configured. The port map of the stream is configured to egress the traffic out Port 0. The example shows the Sequence Recovery set to the Vector Recovery algorithm. The example shows the relay removing the redundancy tag of streams that ingress Port 1 and Port 2, then egress them at Port 0. The TSN_streamSplit.port is set to 0xFF for remove tag and no operation mode. The SES_FrerRemoveRelaySystemForStream() API can be used to remove the listener proxy.

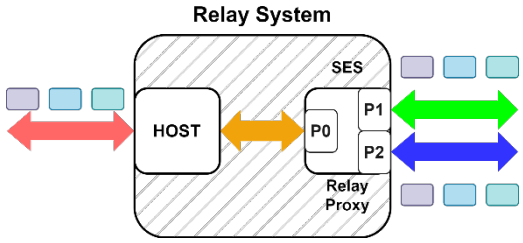

Figure 36. FRER Relay Configuration

```
/* Enable Loop Protection before configuring Relay System */
int32_t ses_frer_relay_example(void) {
    int32_t rv = 0;

        uint8_t destinationMAC[6] = { 0x00, 0x00, 0x00, 0x11, 0x11, 0x11 };

        TSN_ieee802Dot1cbFrer_t frerParameters;
        TSN_ieee802Dot1cbStream_t streamParameters;
        TSN_ieee802Dot1cbMaskAndMatch_t maskAndMatchParameters;

        memset(&frerParameters, 0, sizeof(TSN_ieee802Dot1cbFrer_t));
          memset(&streamParameters, 0, sizeof(TSN_ieee802Dot1cbStream_t));
           memset(&maskAndMatchParameters, 0, sizeof(TSN_ieee802Dot1cbMaskAndMatch_t));

        /* Stream table Parameters - Null Stream Identification */
        streamParameters.TSN_streamIdentity.identificationType = TSN_nullStream;

        /* Assigning Desination MAC address */
        for (int i = 0; i < 6; i++) {
            streamParameters.TSN_streamIdentity.TSN_parameters.TSN_nullStreamIdentification.destina▶
tionMac[i] = destinationMAC[i];
        }

        /* VLAN ID 100 */
        streamParameters.TSN_streamIdentity.TSN_parameters.TSN_nullStreamIdentification.vlan = 100;
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

```
        /* Egress traffic on port 0 */
        streamParameters.portMap = SES_PORTMAP_PORT_0;
        streamParameters.TSN_HardwareTableIndices.dynTblIdx[0] = -1;
        streamParameters.TSN_HardwareTableIndices.dynTblIdx[1] = -1;
        streamParameters.TSN_HardwareTableIndices.exTblruleId = -1;
        streamParameters.useStreamtable = false;
        streamParameters.usedCnt = 0;

        /* Configure frer */
        frerParameters.TSN_sequenceRecovery.algorithm = TSN_vector;
        frerParameters.TSN_sequenceRecovery.historyLength = 5;
        frerParameters.TSN_sequenceRecovery.resetTimeout = 0xffff;
        frerParameters.TSN_sequenceRecovery.takeNoSequence = false;
        frerParameters.TSN_sequenceRecovery.individualRecovery = false;

        /* Set to 0xFF or 255 for listener/relay operation */
        frerParameters.TSN_streamSplit.port = 0xFF; // Set to 0xFF or 255 for listener/relay operation

        /* Listener operation */
        frerParameters.TagOperation = TSN_relayRemoveTag;
        frerParameters.TSN_HardwareTableIndices.individualRecoveryTblIdx = -1;
        frerParameters.TSN_HardwareTableIndices.sequenceGenerationTblIdx = -1;
        frerParameters.TSN_HardwareTableIndices.sequenceRecoveryTblIdx = -1;
        frerParameters.TSN_HardwareTableIndices.txTransformTblIdx = -1;

           /* Configure VLAN mode */
           rv = SES_SetVlanMode(100, 0xFFF);
           if (rv != SES_PORT_OK) {
              printf("SES_SetVlanMode => %d\n", rv);
           }

        rv = SES_FrerConfigureRelaySystemForStream(&frerParameters, &streamParameters, &maskAndMatch▶
Parameters);

        return rv;
}
```

INDIVIDUAL RECOVERY

Individual recovery focuses on handling errors such as a stuck transmitter repeatedly sending the same packet. This is done by removing repeating sequence numbered packets received from a stuck transmitter. Applying the individual recovery function, error streams can be discovered early, and pollution of the merged stream avoided. The individual recovery is implemented on the ingress port of the switch. This prevents repeating sequence from being forward to the rest of the ports. The individual recovery can be applied on its own or with conjunction with a sequence recovery function. If the individual recovery is used on its own, it would just remove the packets with repeating sequence number. The example configures the relay system to handle the individual recovery and configures the listener system to handle the sequence recovery.

```
int32_t ses_frer_individual_recovery_example(void) {
    int32_t rv = 0;
    /* Enable Loop Protection before configuring FRER */

    uint8_t destinationMAC[6] = { 0x00, 0x00, 0x00, 0x11, 0x11, 0x11 };

    TSN_ieee802Dot1cbFrer_t frerParameters;
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

```
    TSN_ieee802Dot1cbStream_t streamParameters;
    TSN_ieee802Dot1cbMaskAndMatch_t maskAndMatchParameters;

    memset(&frerParameters, 0, sizeof(TSN_ieee802Dot1cbFrer_t));
    memset(&streamParameters, 0, sizeof(TSN_ieee802Dot1cbStream_t));
    memset(&maskAndMatchParameters, 0, sizeof(TSN_ieee802Dot1cbMaskAndMatch_t));

    /* Stream table Parameters - Null Stream Identification */
    streamParameters.TSN_streamIdentity.identificationType = TSN_nullStream;

    /* Assigning Desination MAC address */
    for (int i = 0; i < 6; i++) {
        streamParameters.TSN_streamIdentity.TSN_parameters.TSN_nullStreamIdentification.destination▶
Mac[i] = destinationMAC[i];
    }

    /* VLAN ID 100 */
    streamParameters.TSN_streamIdentity.TSN_parameters.TSN_nullStreamIdentification.vlan = 100;

    /* Egress traffic on port 0 */
    streamParameters.portMap = SES_PORTMAP_PORT_0;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[0] = -1;
    streamParameters.TSN_HardwareTableIndices.dynTblIdx[1] = -1;
    streamParameters.TSN_HardwareTableIndices.exTblruleId = -1;
    streamParameters.useStreamtable = false;
    streamParameters.usedCnt = 0;

    /* Configure frer */
    frerParameters.TSN_sequenceRecovery.port = SES_PORTMAP_PORT_1 | SES_PORTMAP_PORT_2;
    frerParameters.TSN_sequenceRecovery.algorithm = TSN_match;
    frerParameters.TSN_sequenceRecovery.resetTimeout = 0xffff;
    frerParameters.TSN_sequenceRecovery.takeNoSequence = false;
    frerParameters.TSN_sequenceRecovery.individualRecovery = true;

    /* Set to 0xFF or 255 for listener/relay operation */
    frerParameters.TSN_streamSplit.port = 0xFF;

    /* Listener operation */
    frerParameters.TagOperation = TSN_relayTagNoOperation;
    frerParameters.TSN_HardwareTableIndices.individualRecoveryTblIdx = -1;
    frerParameters.TSN_HardwareTableIndices.sequenceGenerationTblIdx = -1;
    frerParameters.TSN_HardwareTableIndices.sequenceRecoveryTblIdx = -1;
    frerParameters.TSN_HardwareTableIndices.txTransformTblIdx = -1;

    /* Configure VLAN mode */
    rv = SES_SetVlanMode(100, 0xFFF);
    if (rv < SES_PORT_OK) {
        printf("SES_SetVlanMode :: %d\n", rv);
    }

    rv = SES_FrerConfigureRelaySystemForStream(&frerParameters, &streamParameters, &maskAndMatchParame▶
ters);
    printf("Configure Individual Recovery :: %d\n\r", rv);

    /* Configure frer */
    frerParameters.TSN_sequenceRecovery.algorithm = TSN_match;
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

```
    frerParameters.TSN_sequenceRecovery.resetTimeout = 0xffff;
    frerParameters.TSN_sequenceRecovery.takeNoSequence = false;
    frerParameters.TSN_sequenceRecovery.individualRecovery = false;

    rv = SES_FrerConfigureListenerProxyForStream(&frerParameters, &streamParameters, &maskAndMatchPara▶
meters);
    printf("Configure Sequence Recovery :: %d\n\r", rv);
    return rv;
}
```

SEQUENCE RECOVERY STATISTICS

The SES_FrerGetSequenceRecoveryStatistics() API can be used to get the sequence recovery statistics of a listener or relay system. The API is configured to get the statistics one port at a time. The example shows the switch configured as a listener system, then calling the API to check the sequence recovery statistics.

```
int32_t ses_frer_recovery_statistics_example(void) {
  int32_t rv = 0;

  uint8_t destinationMAC[6] = { 0x00, 0x00, 0x00, 0x11, 0x11, 0x11 };

  TSN_ieee802Dot1cbFrer_t frerParameters;
  TSN_ieee802Dot1cbStream_t streamParameters;
  TSN_ieee802Dot1cbMaskAndMatch_t maskAndMatchParameters;

  memset(&frerParameters, 0, sizeof(TSN_ieee802Dot1cbFrer_t));
  memset(&streamParameters, 0, sizeof(TSN_ieee802Dot1cbStream_t));
  memset(&maskAndMatchParameters, 0, sizeof(TSN_ieee802Dot1cbMaskAndMatch_t));

  /* Stream table Parameters - Null Stream Identification */
  streamParameters.TSN_streamIdentity.identificationType = TSN_nullStream;

  /* Assigning Desination MAC address */
  for (int i = 0; i < 6; i++) {
streamParameters.TSN_streamIdentity.TSN_parameters.TSN_nullStreamIdentification.destinationMac[i] =
destinationMAC[i];
  }

  /* VLAN ID 100 */
  streamParameters.TSN_streamIdentity.TSN_parameters.TSN_nullStreamIdentification.vlan = 100;

  /* Egress traffic on port 0 */
  streamParameters.portMap = SES_PORTMAP_PORT_0;
  streamParameters.TSN_HardwareTableIndices.dynTblIdx[0] = -1;
  streamParameters.TSN_HardwareTableIndices.dynTblIdx[1] = -1;
  streamParameters.TSN_HardwareTableIndices.exTblruleId = -1;
  streamParameters.useStreamtable = false;

  /* Configure frer */
  frerParameters.TSN_sequenceRecovery.algorithm = TSN_vector;
  frerParameters.TSN_sequenceRecovery.historyLength = 2;
  frerParameters.TSN_sequenceRecovery.resetTimeout = 0xffff;
  frerParameters.TSN_sequenceRecovery.takeNoSequence = false;
  frerParameters.TSN_sequenceRecovery.individualRecovery = false;
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

```
  frerParameters.TSN_HardwareTableIndices.individualRecoveryTblIdx = -1;
  frerParameters.TSN_HardwareTableIndices.sequenceGenerationTblIdx = -1;
  frerParameters.TSN_HardwareTableIndices.sequenceRecoveryTblIdx = -1;
  frerParameters.TSN_HardwareTableIndices.txTransformTblIdx = -1;

  SES_FrerConfigureListenerProxyForStream(&frerParameters, &streamParameters,
    &maskAndMatchParameters);
  printf("Listener System :: %d\n\r", rv);

  /* Delay for stream to ingress listener */
  Sleep(10000); ;

  uint8_t requestSrfIndexes[1] = { 0 };
  requestSrfIndexes[0] = frerParameters.TSN_HardwareTableIndices.sequenceRecoveryTblIdx;
  TSN_frerPerPortPerStreamLessStatistics_t statisticsResponse[SES_FRER_NUM_MAX_STATS_PARAM];

  bool extendedStatistics = false;
  printf("ses_frer_recovery_statistics_example :: %d\n\r", rv);

  /* Run loop to check number of packets passed or discarded */
  while (1) {
    Sleep(3000); /* Delay for each iteration for traffic to ingress the switch */
    rv = SES_FrerGetSequenceRecoveryStatistics(SES_macPort0, 1, requestSrfIndexes,        statistics▶
Response, extendedStatistics);
    printf("Sequence Recovery - Port 0 :: %d\n\r", rv);

    printf("Port 0 Statistics :: \n");
    printf("rxPassedPkts :: %llu\n", statisticsResponse[0].rxPassedPkts);
    printf("rxDiscardedPkts :: %llu\n", statisticsResponse[0].rxDiscardedPkts);
  }

  printf("ses_frer_recovery_statistics_example :: %d\n\r", rv);
  return rv;
}
```

LATENT ERROR DETECTION

Latent error detection function is used by listener or relay devices to identify errors in redundant traffic streams that are not caught immediately when the listener or relay system is eliminating redundant frames.

In an FRER network, streams are transmitted along multiple redundant paths to improve reliability. While duplicate frames can be discarded as soon as they arrive at a listener or relay system, certain errors may only manifest over time. These errors are referred to as latent errors.

The purpose of latent error detection is to ensure that streams remain constant across redundant paths. It provides an additional check against failures such as persistent loss of packets on one path, misconfiguration in the system that leads to imbalance between passed and discarded frames, and possible degradation of the performance of a link or node in the network. The latent error detection assumes that if there are *N* number of redundant paths, then *N-1* frames should be discarded.

Latent error detection is only available with sequence recovery and is not used with Individual recovery.

There are four parameters that need to be configured when latent error detection is enabled:

- **difference**: This is the maximum allowed deviation between the number of packets passed and those discarded without triggering an error.
- **period**: This is the interval at which the latent error test function is called.
- **resetPeriod**: This is the interval at which the latent error reset function is called.

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

- **paths**: This is the number of redundant paths for the stream

```
/* FRER Configuration Structure */
typedef struct {
    ……
    struct {
        uint32_t index;
        uint32_t stream;
        uint8_t port; // type if:interface-ref in spec
        TSN_direction_t directionOutFacing;
        bool reset;
        TSN_seqRcvyAlgorithm_t algorithm;
        TSN_sequenceHistoryLength_t historyLength;
        uint32_t resetTimeout;
        uint32_t invalidSequenceValue;
        bool takeNoSequence;
        bool individualRecovery;
        bool latentErrorDetection;
        /* Latent Error Detecion Parameters */
        struct {
            int32_t difference;
            uint32_t period;
            uint16_t paths;
            uint32_t resetPeriod;
        } TSN_latentErrorDetectionParameters;
    } TSN_sequenceRecovery;
    ……
} TSN_ieee802Dot1cbFrer_t;
```

When latent error detection is enabled, the switch captures statistics related to the sequence recovery functions and periodically reports this information to the host. Figure 37 illustrates the interactions between the host and the switch. The host uses these statistics to determine whether a latent error has occurred.

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

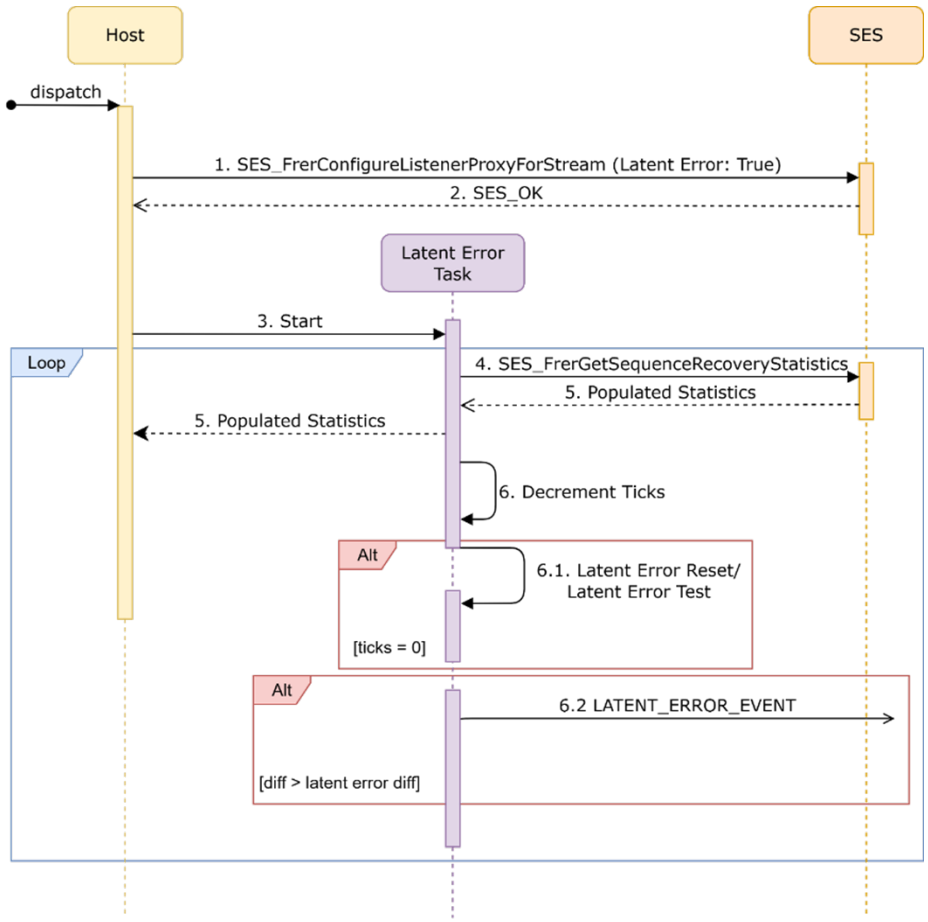

Figure 37. Overview of interaction between host and switch for latent error function with FRER

The switch captures full statistics for the first eight sequence recovery indexes. For any sequence recovery functions beyond eight, the switch captures partial statistics only, specifically the count of passed and discarded packets.

Latent Error Function

The latent error function is composed of two periodic functions, the latent error test function and latent error reset function. The TSN driver library package provides example implementations that demonstrate the use of the latent error detection feature, including corresponding host-side implementations. Examples of the latent error functions are included in the Linux examples.

Latent Error Reset

The latent error reset function recomputes the *CurBaseDifference* variable, which represents the difference between the expected and the actual number of packets discarded. An example of the latent error reset function is shown below, as defined in the IEEE 802.1CB-2017 standard.

```
void LatentErrorReset() {
    CurBaseDifference = (frerCpsSeqRcvyPassedPackets *
        (frerSeqRcvyLatentErrorPaths - 1)) - frerCpsSeqRcvyDiscardedPackets;
    frerCpsSeqRcvyLatentErrorResets = frerCpsSeqRcvyLatentErrorResets + 1;
}
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

The example implementation of the latent error reset function in the driver library is shown below.

```
static int32_t SES_LatentErrorReset(
 TSN_frerPerPortPerStreamLessStatistics_t* seqRecovStatistics_p,
 TSN_frerLeInstanceState_t** latentErrorParams_p)
 {
 if (seqRecovStatistics_p == NULL)
    return SES_PARAM_ERROR;

 /* Update the values for all sub-instances of this SRF - for each port*/
 for (uint8_t port = 0; port < MAX_NUM_DEVICE_PORTS; port++) {
 if (latentErrorParams_p[port] != NULL) {
    /* currBaseDiff = (passedPkts * (paths-1)) - disscardedPkts */
    /* Assign each subinstance the aggregated currBaseDiff */
    latentErrorParams_p[port]->curBaseDifference = (seqRecovStatistics_p->rxPassedPkts *
          ((latentErrorParams_p[port]->latentErrorDetectionParameters.paths) - 1)) -
          seqRecovStatistics_p->rxDiscardedPkts;

    /* Increment resets counter */
    latentErrorParams_p[port]->frerCpsSeqRcvyLatentErrorResets++;
    /* Reset the ticks */
    latentErrorParams_p[port]->resetTicks =
          latentErrorParams_p[port]->latentErrorDetectionParameters.resetPeriod;
    }
 }
    return SES_OK;
}
```

Latent Error Test

The latent error test function counts the number of frames that are passed and discarded. This function raises a latent error flag when the differences between the passed and discarded exceeds the set threshold.

An example of the latent error test function, as defined in the IEEE 802.1CB-2017 standard, is shown below. The *diff* variable is computed based on the *CurBaseDifference*, the number of passed packets, the number of redundant paths, and the number of discarded packets. The function checks whether more than one redundant path is present and whether the period is greater than zero. If both conditions are satisfied, the absolute value of *diff* is compared against the *difference* threshold configured in the FRER instance.

```
void LatentErrorTest() {
    int diff = CurBaseDifference - ((frerCpsSeqRcvyPassedPackets *
        (frerSeqRcvyLatentErrorPaths - 1)) -
        frerCpsSeqRcvyDiscardedPackets);
    if (frerSeqRcvyLatentErrorPaths > 1 && // There are multiple paths
        frerSeqRcvyLatentErrorPeriod > 0) // LE detection is turned on
    {
        if (diff < 0)
            diff = -diff;
        if (diff > frerSeqRcvyLatentErrorDifference) {
            SIGNAL_LATENT_ERROR;
        }
    }
}
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

The example implementation of the latent error test function in the driver library is shown below.

```
static int32_t SES_LatentErrorTest(
 TSN_frerPerPortPerStreamLessStatistics_t* seqRecovStatistics_p,
 TSN_frerLeInstanceState_t** latentErrorParams_p,
 uint32_t streamHandle)
{
 uint8_t port;
 int64_t diff;

 if (seqRecovStatistics_p == NULL)
    return SES_PARAM_ERROR;

 for (port = 0; port < MAX_NUM_DEVICE_PORTS; port++) {
    if (latentErrorParams_p[port] != NULL) {
      /* currBaseDiff - ((passedPkts x (paths-1)) - disacrdedPkts) */
      /* Each instance already has the aggregated currBaseDiff */
      diff = latentErrorParams_p[port]->curBaseDifference -
          ((seqRecovStatistics_p->rxPassedPkts *
          (latentErrorParams_p[port]->latentErrorDetectionParameters.paths - 1)) -
          seqRecovStatistics_p->rxDiscardedPkts);

      /* No LE if there is only a single path */
      if ((latentErrorParams_p[port]->latentErrorDetectionParameters.paths > 1)) {
        if (diff < 0)
          diff = -diff;
        if (diff > latentErrorParams_p[port]->latentErrorDetectionParameters.difference) {
          /* Handle latent error once it occurs */
          if (latentErrorHandlerFunc_p == NULL)
            printf("Latent Error occured but handler is NULL!\n");
          else {
            SES_latentErrorEventParam_t event = {
              .streamHandle = streamHandle,
              .expectedDiff =
              latentErrorParams_p[port]->latentErrorDetectionParameters.difference,
              .latentError = diff
          };
          latentErrorHandlerFunc_p(event);
        }
      }
    }
    /* If an SRF is not attached to a given port this entry is NULl in latentErrorParams_p array for a
port.
    Since statistics are aggregated for all available ports, checking the currentBaseDifference for
the first
    non-null port is sufficent to determine if there is a latent error.*/

    break;
  }
 }

 /* Reset the period for all ports */
 for (port = 0; port < MAX_NUM_DEVICE_PORTS; port++) {
    if (latentErrorParams_p[port] != NULL)
      latentErrorParams_p[port]->testTicks =
          latentErrorParams_p[port]->latentErrorDetectionParameters.period;
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

```
 }
 return SES_OK;
}
```

Host Implementation

This section describes how to use the Linux example for latent error detection. In the examples directory of the driver library, the SES_lat_err_middleware source code provides two functions for configuring a FRER listener or relay system:

- SES_ConfigureListenerProxyForStream
- SES_ConfigureRelaySystemForStream

These functions are used to configure a FRER instance. When the FRER instance is configured successfully, the associated sequence recovery instance is set to the active state.

The source code also includes functions for removing previously configured listener and relay systems.

```
/**
* Configure a proxy for an FRER unaware listener for a stream
*/
int32_t SES_ConfigureListenerProxyForStream(
    TSN_ieee802Dot1cbFrer_t* frerParameters_p,
    TSN_ieee802Dot1cbStream_t* streamParameters_p,
    TSN_ieee802Dot1cbMaskAndMatch_t* maskAndMatchParameters_p)
{
    int retVal = SES_ERROR;
    printf("Configuring....\n");

    retVal = SES_FrerConfigureListenerProxyForStream(frerParameters_p,
        streamParameters_p,
        maskAndMatchParameters_p);

    /* if SRF configuration is successful, then start collecting statistics*/
    if (retVal == SES_OK)
        retVal = SES_SetSrfInstanceActive(frerParameters_p, streamParameters_p);

    return retVal;
}

/**
* Configure a relay system for a stream
*/
int32_t SES_ConfigureRelaySystemForStream(
    TSN_ieee802Dot1cbFrer_t* frerParameters_p,
    TSN_ieee802Dot1cbStream_t* streamParameters_p,
    TSN_ieee802Dot1cbMaskAndMatch_t* maskAndMatchParameters_p)
{
    int retVal = SES_ERROR;

    retVal = SES_FrerConfigureRelaySystemForStream(frerParameters_p,
        streamParameters_p,
        maskAndMatchParameters_p);

    if (retVal == SES_OK)
        retVal = SES_SetSrfInstanceActive(frerParameters_p, streamParameters_p);
```

FRAME REPLICATION AND ELIMINATION FOR RELIABILITY, 802.1CB

```
    return retVal;
}
```

In addition to configuring a listener or relay system, the user must also handle latent error events. In the provided example, the user can register a callback function to handle these events. If no callback function is registered, the example can still detect a latent error. However, the notification is limited to a console print message.

```
int32_t SES_RegisterLEHandler(SES_LatErrHandlerFunc_t latErrHandler_p)
{
    if (latErrHandler_p == NULL) {
        printf("Latent Error Handler is invalid\n");
        return SES_ERROR;
    }
    latentErrorHandlerFunc_p = latErrHandler_p;

    return SES_OK;
}
```

MULTIPLE SPANNING TREE PROTOCOL, MSTP

Spanning tree protocols were developed to prevent broadcast storms by eliminating loops in network topologies. STP is the original spanning tree protocol, Rapid Spanning Tree Protocol (RSTP) was defined after significantly improving convergence speed and Multiple Spanning Tree Protocol (MSTP) is the most recent version defined in IEEE 802.1Q-2022. STP and RSTP networks use a single spanning tree instance for the entire network, whereas MSTP supports grouping and mapping of VLANs into different spanning tree instances. This approach reduces the number of spanning tree calculations, improving resource utilization.

When operating in MSTP mode, the switch is interoperable with other switches running MSTP, STP or RSTP. It supports up to four instances, three MSTIs (multiple spanning tree instances), and the CIST (common and internal spanning tree). Each MSTI operates independently within an MST region and elects its own root bridge to optimize traffic flow for the VLANs assigned to that instance. Any number of VLANs can be mapped to an instance. However, only 64 VLANs can be active. By default, all VLANs (0 to 4095) are mapped to CIST. Mapping doesn't indicate that VLANs are active. It just serves as a lookup table when VLANs are activated. For example, if 1000-2000 VLAN IDs are mapped to MSTI 1, and VLAN ID 1500 gets activated later, it will be subjected to port states of MSTI 1.

The switch runs the MSTP stack on the packet assist engine and supports configuration for MSTP or the earlier profiles, STP or RSTP. If configured for STP or RSTP, only one instance can be created. When operating in MSTP mode, the switch is interoperable with other switches running STP or RSTP. MSTP on the switch is compatible with other network protocols, such as LLDP, time synchronization, scheduled traffic, FRER, PSFP and frame preemption. MSTP is not intended to operate with other redundancy protocols such as MRP, PRP or HSR.

The MSTP stack is disabled by default. To enable the MSTP stack, SES_MstpStart() needs to be called. When the MSTP stack is enabled the CIST/MSTI 0 will be configured with default values recommended in the specification and all VLAN (0 to 4095) are mapped to CIST. It is possible to configure MSTP instances before or after enabling the MSTP stack using SES_MstpCreateMstInstance(). The example below shows multiple MSTP instances created before enabling the MSTP stack. The VLAN mapped to any MSTP instance are not active unless configured through the Trunk/Access configuration or the VLAN table.

```
int32_t ses_multiple_msti_example(void){
    int32_t rv = SES_ERROR;

    uint8_t mstId0;
    uint8_t mstId1;
    uint8_t mstId2;
    char vlanIds1[3][SES_MSTP_VLAN_RANGE_NAME_LEN] = {"10-20", "50", "90-100"};
    char vlanIds2[2][SES_MSTP_VLAN_RANGE_NAME_LEN] = {"30", "40"};

    /* For first instance (CIST/MSTI0), all VLANs (0 to 4095) will be mapped
     * and length and vlanIDs parameters will be ignored
     */

    rv = SES_MstpCreateMstInstance(0x3f, 4096, NULL, NULL, &mstId0);

    /* portMap is applicable only for first instance (CIST) */
    rv = SES_MstpCreateMstInstance(NULL, 8192, 3, &vlanIds1, &mstId1);
    rv = SES_MstpCreateMstInstance(NULL, 12288, 2, &vlanIds2, &mstId2);
    rv = SES_MstpStart();

    return rv;
}
```

The switch is able to configure the MSTP timing parameters such as the Bridge Hello Time, Bridge Max Age, and Bridge Forward Delay. The example shows the default values when starting the MSTP stack and the allowable range of these parameters . The switch enforces the relationship of the timing parameters as specified by the standard. In the example, if the Bridge Max Age is set to 40 before the Bridge Forward Delay, the function SES_MstpSetBridgeMaxAge() will return an error.

```
int32_t ses_bridge_forward_delay_example(void){
    int32_t rv = SES_ERROR;
```

MULTIPLE SPANNING TREE PROTOCOL, MSTP

```
    /* Bridge Hello Time range: 1 to 10. Default value: 2.
     * Bridge Max Age range: 6 to 40. Default value: 20.
     * Bridge Forward Delay range: 4 to 30. Default value: 15.
     * The bridge enforces the following relationship:
     *         2 x ( Bridge Forward Delay - 1 seconds ) <= Bridge Max Age
     *        Bridge Max Age >= 2 x ( Bridge Hello Time + 1 seconds )
     */
    uint16_t bridgeForwardDelay = 21;
    uint16_t bridgeMaxAge = 40;

    /* Start MSTP stack */
    rv = SES_MstpStart();
    if (rv == SES_OK) {
        /* Set Bridge Forward Delay */
        rv = SES_MstpSetBridgeForwardDelay(bridgeForwardDelay);
        if (rv < SES_OK)
            printf("SES_MstpSetBridgeForwardDelay :: %d\n", rv);
        /* Set Bridge Max Age */
        rv = SES_MstpSetBridgeMaxAge(bridgeMaxAge);
        if (rv < SES_OK)
            printf("SES_MstpSetBridgeMaxAge :: %d\n", rv);
    }

    return rv;
}
```

It is possible to exclude a VLAN from being managed by the spanning tree using the function SES_MstpExcludeVlanFromSpanningTree(). The VLAN will share the same links with the spanning tree, but not subjected to the spanning tree state. This is useful in scenarios where VLANs have a dedicated static route. Independence from the spanning tree is managed by using other protocols (e.g., FRER) to handle loops and redundancy in the network.

TROUBLE-SHOOTING

LINUX DRIVER BUILD

- While installing Libyang, the following error can be resolved by ensuring the dependency is installed `sudo apt install -y libpcre2-dev`

```
CMake Error at /usr/share/cmake-3.25/Modules/FindPackageHandleStandardArgs.cmake:230 (message):
Could NOT find PCRE2 (missing: PCRE2_LIBRARY PCRE2_INCLUDE_DIR) (Required is at least version
"10.21")
```

- When building the libraries and project, ensure the time zone of the linux machine is not Dublin time zone.
- To build the cmake project into debug mode, follow the steps below:

```
cd ses-example-app/project/cmake/
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=Debug ..
cmake --build . --config Debug
```

NETCONF AND SYSREPO

- To retrieve the list of installed YANG modules, use the following command: `sysrepoctl -l`
- To check that the startup datastore is correctly configured and the initial configuration has been imported, use the following command to display the content of the startup datastore `sysrepocfg -X -d startup`
- If the content of the startup datastore is incorrect, as shown in the example output below, it can be manually updated without need to clear the Sysrepo repository or restarting the application, simply perform the following
  - navigate to the sysrepo/modules folder
  - run the following command: `sysrepocfg --edit=ses-default-startup.xml -d startup`
  - Incorrect output example:

```
<iec62439 xmlns="urn:ietf:params:xml:ns:yang:iec62439">
<lreInterfaceConfigTable>
<lreInterfaceConfigEntry>
<lreInterfaceConfigIndex>0</lreInterfaceConfigIndex>
<lreRedundancyDevice xmlns="urn:adi:ses">redbox</lreRedundancyDevice>
</lreInterfaceConfigEntry>
</lreInterfaceConfigTable>
</iec62439>
```

YOCTO BUILD

- While installing the respective libraries if the installation returns “unable to locate”, review updating apt repository. Ensure to install correct version of libraries based upon the platform being used and run the following commands:

- sudo apt –fix-broken install
  - sudo apt-get update
  - sudo apt-get upgrade

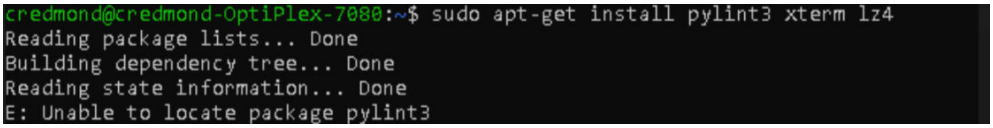

Figure 38. Unable to locate

TROUBLE-SHOOTING

- If netopeer2-server modules are not installed or netopeer2-server does not start correctly, then install the netopeer2 in yocto image using ssh:
  - git clone -b v2.1.16 https://github.com/CESNET/netopeer2.git
  - Copy the folder in some directory in yocto.
  - cd netopeer2 && chmod +x scripts/setup.sh && chmod +x scripts/merge_*.sh && mkdir build && cd build && cmake -DCMAKE_BUILD_TYPE:String=Release .. && make && make install
- If the netopeer2-server is not able to bind at port 830 then change the <local-port> value to greater than 1024 in ietf-netocnf-server running datastore and run netopeer2-server again.
- Firmware updates with yocto build on devices with existing firmware are not currently supported. During the initial configuration, the SES_AddDevice() API will return negative response (-1: SES_AddDevice error). Firmware updates are successful using cmake or alternatively using the evaluation package. This issue is captured as a known issue in the release notes and will be reviewed in future updates.

SOFTWARE RELEASE NOTES

Detailed software release notes are included in the driver package with information about the changes and known issues.