PM5315

SPECTRA-2488™

DRIVER MANUAL

PROPRIETARY AND CONFIDENTIAL

RELEASED

ISSUE 2: NOVEMBER, 01

ABOUT THIS MANUAL AND SPECTRA-2488

This manual describes the Spectra-2488 device driver. It describes the driver's functions, data structures, and architecture. This manual focuses on the driver's interfaces to your application, real-time operating system, and to the device. It also describes in general terms how to modify and port the driver to your software and hardware platform.

Audience

This manual was written for people who need to:

- Evaluate and test the Spectra-2488 devices
- Modify and add to the Spectra-2488 driver's functions
- Port the Spectra-2488 driver to a particular platform.

References

For more information about the Spectra-2488 driver, see the driver's release Notes. For more information about the Spectra-2488 device, see the documents listed in Table 1 and any related errata documents.

Table 1: Related Documents

Document Name	Document Number
Spectra-2488 Telecom Standard Product Data Sheet	PMC-1990821

Note: Ensure that you use the document that PMC-Sierra issued for your version of the device and driver.

Revision History

Issue No.	Issue Date	Details of Change
Issue 1	November 2000	Document created
Issue 2	November 2001	Updated for driver production release: 1) Added side effect to spe2488GetStatus and spe2488GetCnt APIs: added latency of the APIs when used in QUAD mode with unused ports. 2) Added extra APIs to retrieve all stats for a given port. 3) Added to the function descriptions of Section, line, path and path-TU3 Termination APIs, to clarify the feature covered by those APIs. 4) Removed API spe2488DiagSysSideSysLoop

Legal Issues

PMC-2001285 (R2), ref PMC-2000048 (R4)

Trademarks

SPECTRA-2488 is a trademark of PMC-Sierra, Inc.

Contacting PMC-Sierra

PMC-Sierra, Inc.
8555 Baxter Place Burnaby, BC
Canada V5A 4V7

Tel: +1-604- 415-6000
Fax: +1-604-415-6200

Document Information: document@pmc-sierra.com
Corporate Information: info@pmc-sierra.com
Technical Support: apps@pmc-sierra.com
Web Site: http://www.pmc-sierra.com

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The following sections of the SPECTRA-2488 Driver Manual describe the SPECTRA-2488 device driver. The code provided throughout this document is written in C language. This has been done to promote greater driver portability to other embedded hardware (Section 6) and Real-Time Operating System environments (Section 7).

Section 3 of this document, Software Architecture, defines the software architecture of the SPECTRA-2488 device driver by including a discussion of the driver's external interfaces and its main components. The Data Structure information in Section 3.6 describes the elements of the driver that configure or control its behavior. Included here are the constants, variables and structures that the SPECTRA-2488 device driver uses to store initialization, configuration and status information. Section 5 provides a detailed description of each function that is a member of the SPECTRA-2488 driver Application Programming Interface (API). The section outlines function calls that hide device-specific details and application callbacks that notify the user of significant device events.

For your convenience, section 8 of this manual provides a brief guide for porting the SPECTRA-2488 device driver to your hardware and RTOS platform. In addition, an extensive Appendix (page 151) and Index (page 162) provides you with useful reference information.

2 DRIVER FUNCTIONS AND FEATURES

This section describes the main functions and features supported by the SPECTRA-2488 driver.

2.1 General Driver Functions

Open/Close Driver Module

Opening the driver module allocates all the memory needed by the driver and initializes all module level data structures.

Closing the driver module shuts down the driver module gracefully after deleting all devices that are currently registered with the driver, and releases all the memory allocated by the driver.

Start/Stop Driver Module

Starting the driver module involves allocating all RTOS resources needed by the driver such as timers and semaphores (except for memory, which is allocated during the Open call).

Stopping the driver module involves de-allocating all RTOS resources allocated by the driver without changing the amount of memory allocated to it.

Add/Delete Device

Adding a device involves verifying that the device exists, associating a device Handle to the device, and storing context information about it. The driver uses this context information to control and monitor the device.

Deleting a device involves shutting down the device and clearing the memory used for storing context information about this device.

Device Initialization

The initialization function resets then initializes the device and any associated context information about it. The driver uses this context information to control and monitor the SPECTRA-2488 device.

Activate/De-Activate Device

Activating a device puts it into its normal mode of operation by enabling interrupts and other global registers. A successful device activation also enables other API invocations.

On the contrary, de-activating a device removes it from its operating state, and disables interrupts and other global registers.

Read/Write Device Registers

These functions provide a 'raw' interface to the device. Device registers that are both directly and indirectly accessible are available for both inspection and modification via these functions. If applicable, block reads and writes are also available.

Interrupt Servicing/Polling

Interrupt Servicing is an optional feature. The user can disable device interrupts and instead poll the device periodically to monitor status and check for alarm/error conditions.

Both polling and interrupt driven approaches detect a change in device status and report the status to a Deferred-Processing Routine (DPR). The DPR then invokes application callback functions based on the status information retrieved. This allows the driver to report significant events that occur within the device to the application.

Statistics Collection

Functions are provided to retrieve a snapshot of the various counts that are accumulated by the SPECTRA-2488 device. Routines should be invoked often enough to avoid letting the counters to saturate.

2.2 SPECTRA-2488 Specific Driver Functions

These functions provide control and monitoring of the various sections of the SPECTRA-2488 device. These sections are generally enabled or disabled and configured by the MODE specified during device initialization. Changes to these registers that would violate the characteristics of the initialized mode should be disallowed.

Input/Output

Functions are provided to configure the line side and system side interface of the SPECTRA-2488.

- Configures the Time Slot Interchange (TSI)
- Enables / Disables insertion of IDLE characters.

Section Overhead

Functions are provided to control the Section Overhead processing.

- Enable/Disable Section Overhead termination
- Read / write the section trace message (J0).
- Read B1 error counter.
- Detects and reports TIM, TIU, SF, SD, LOS, OOF, LOF.

Line Overhead

Functions are provided to control the line overhead processing.

- Enable/Disable line overhead termination
- Read / write the APS bytes (K1, K2)
- Read / write the synchronization byte (S1)
- Read the B2 error counter and REI counter
- Force Line AIS
- Configure to automatically insert line RDI, line REI or line AIS
- Detects and reports LRDI, LREI, LAIS and APS byte failure

Path Overhead

Functions are provided to control the path and path-TU3 overhead processing.

- Enable/Disable Path Overhead termination
- Read/write the path trace messages (J1).
- Read/write the path signal labels (C2).
- Read the B3 error counter and REI counter.
- Configure to automatically insert path RDI, path enhanced RDI, path REI and path AIS.
- Detect and report TIM, TIU, PSLM, PSLU, LOP, path RDI, path enhanced RDI, path REI, path AIS and PDI.

Device Update

A function is provided to update the device's configuration without forcing a hardware reset.

Device Diagnostics

- Device register read/write test
- Line-side line loopback from the line side receive stream to the transmit stream
- Line-side system loopback from the system side transmit stream to the receive stream
- System-side system loopback from the system side transmit stream to the system side receive stream (on a per STS-1/STM-0 basis).
- System-side line loopback from the ADD Telecombus to the DROP Telecombus (on a per STS-1/STM-0 basis).

Statistics and Alarm Monitoring

Functions to gather statistics and do alarm monitoring.

- Retrieve a snapshot of the clock activity bits.
- Retrieve a snapshot of the current alarm status for the various Sonet/SDH alarms (LOP, OOF, LOF, LOS, xAIS, xRDI, xTIU, xTIM…).
- Read the current B1, B2, B3, B3-TU3, REI, REI-TU3 counts.

Specific Callback Functions

Callback functions are available to the application for event notification from the device driver. Applications will be notified via the callback functions for selected events of interest such as:

- Detection of an IO alarm condition (such as a clock error condition)
- Detection of a Section Overhead alarm condition
- Detection of a Line overhead alarm condition
- Detection of a Path Overhead alarm condition
- Detection of a Path-TU3 overhead alarm condition
- Detection of a Drop bus PRBS generator and monitor (DPGM) alarm condition
- Detection of an Add bus PRBS generator and monitor (APGM) alarm condition

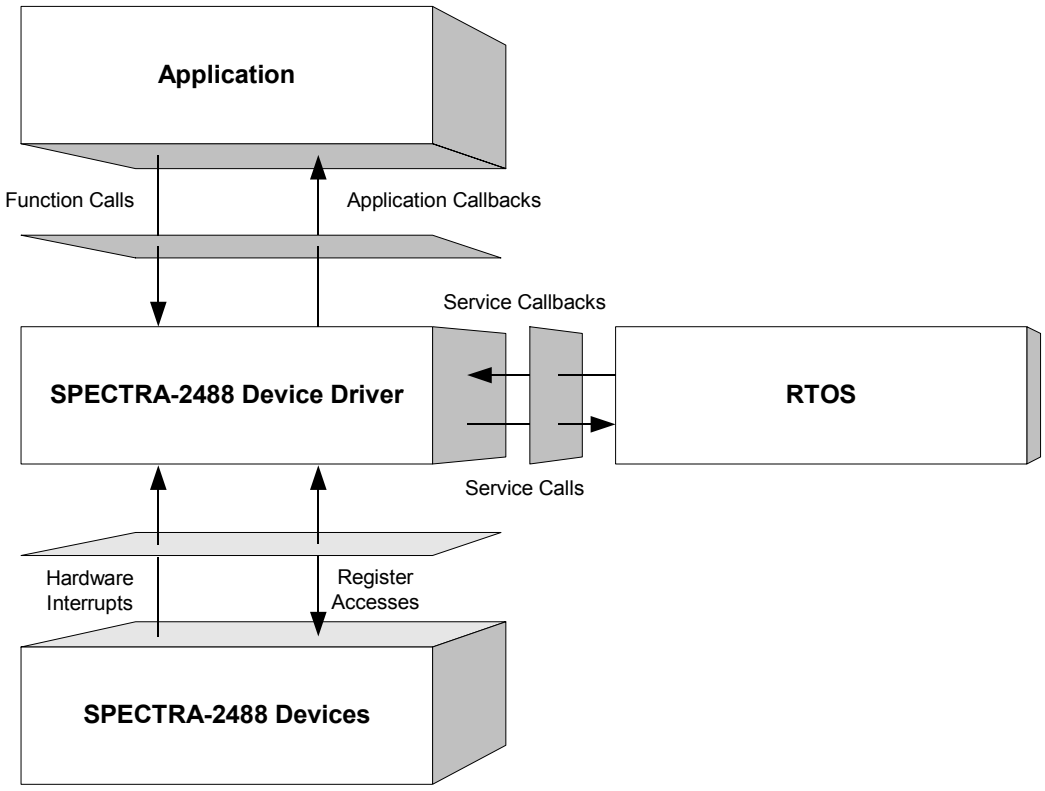
3 SOFTWARE ARCHITECTURE

This section describes the software architecture of the SPECTRA-2488 device driver. This includes a discussion of the driver's external interfaces and its main components.

3.1 Driver External Interfaces

Figure 1 illustrates the external interfaces defined for the SPECTRA-2488 device driver.

Figure 1: Driver External Interfaces



Application Programming Interface

The driver Application Programming Interface (API) is a list of high-level functions that can be invoked by application programmers to configure, control, and monitor the SPECTRA-2488 devices. The API functions perform operations that are more meaningful from a system's perspective. The API includes functions such as:

- Initialize the device(s)
- Perform diagnostic tests
- Validate configuration information
- Retrieve status and statistics information

The driver API functions use the services of the other driver components to provide this system-level functionality to the application programmer.

The driver API also consists of callback routines that are used to notify the application of significant events that take place within the device(s) and module.

Real-Time Operating System (RTOS) Interface

The driver's RTOS interface provides functions that let the driver use RTOS services. The driver requires the memory, interrupt, and preemption services from the RTOS. The RTOS interface functions perform the following tasks for the driver:

- Allocate and de-allocate memory
- Manage buffers for the ISR and the DPR
- Enable and disable preemption

The RTOS interface also includes service callbacks. These are functions installed by the driver using RTOS service calls such as installing interrupts. These service callbacks are invoked when an interrupt occurs.

Note: You must modify the RTOS interface code to suit your RTOS.

Hardware Interface

The hardware interface provides functions that read from and write to the device registers. The hardware interface also provides a template for the ISR that the driver calls when the device raises a hardware interrupt. You must modify this function based on the interrupt configuration of your system.

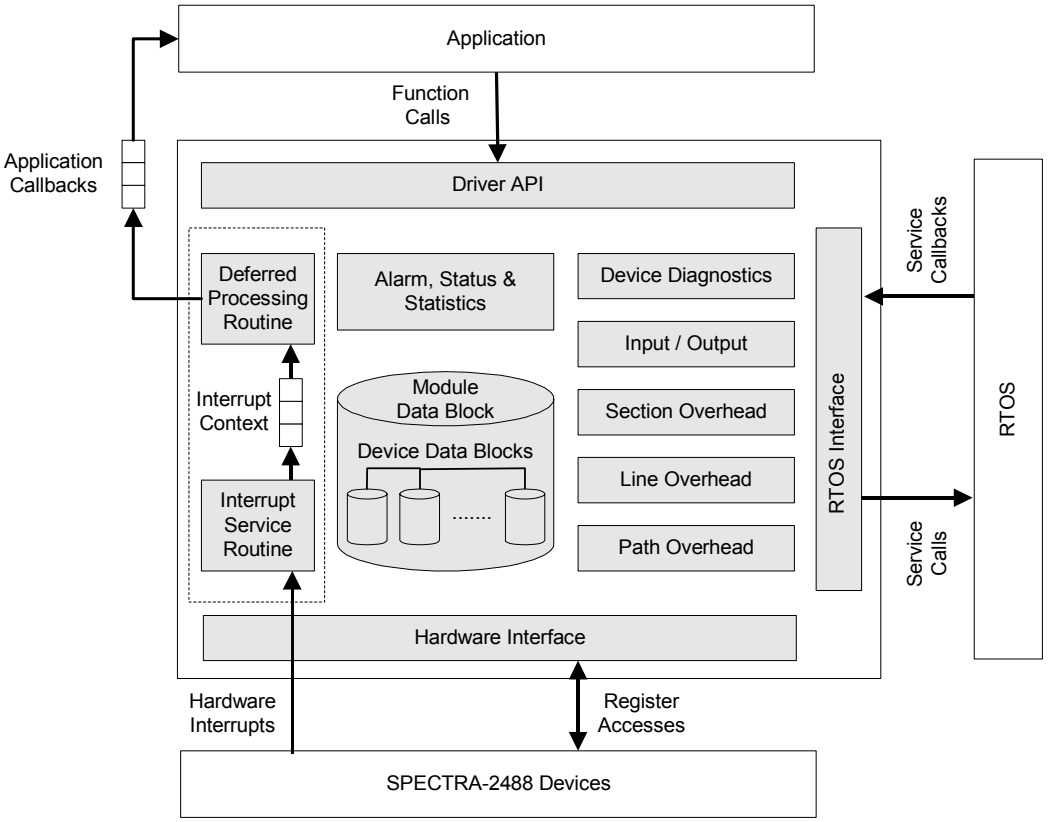
3.2 Main Components

Figure 2 illustrates the top-level architectural components of the SPECTRA-2488 device driver.

The driver includes the following main components:

- Module and device(s) data-blocks
- Interrupt-Service Routine
- Deferred-Processing Routine
- Alarm, status and statistics
- Input/Output
- Section Overhead
- Line overhead
- Path Overhead
- Device diagnostics

Figure 2: Driver Architecture



Module Data-Block and Device(s) Data-Blocks

The Module Data-Block (MDB) is the top layer data structure; it is created by the SPECTRA-2488 driver to store context information about the driver module, such as:

- Module state
- Maximum number of devices
- The DDB(s)

The Device Data-Block (DDB) is contained in the MDB and is initialized by the driver module for each SPECTRA-2488 device that is registered. There is one DDB per device and there is a limit on the number of DDBs; that limit is set by the user when the module is initialized. The DDB is used to store context information about one device, such as:

- Device state
- Control information
- Initialization parameters
- Callback function pointers

Interrupt-Service Routine

The SPECTRA-2488 driver provides an ISR called `spe2488ISR` that checks if there is any valid interrupt condition present for the device. This function can be used by a system-specific interrupt-handler function to service interrupts raised by the device.

The low-level interrupt-handler function that traps the hardware interrupt and calls `spe2488ISR` is system and RTOS dependent. Therefore, it is outside the scope of the driver. Example implementations of an interrupt handler and functions that install and remove it are provided as a reference in section 6.2. You can customize these example implementations to suit your specific needs.

See section 3.5 for a detailed explanation of the ISR and interrupt-servicing model.

Deferred-Processing Routine

The SPECTRA-2488 driver provides a DPR called `spe2488DPR` that processes any interrupt condition gathered by the ISR for that device. Typically, a system specific function, which runs as a separate task within the RTOS, will call `spe2488DPR`.

Example implementations of a DPR task and functions that install and remove it are provided as a reference in section 6.2. You can customize these example implementations to suit your specific needs.

See section 3.5 for a detailed explanation of the DPR and the interrupt-servicing model.

Alarms, Status and Statistics

The alarm, status and statistics section is responsible for monitoring alarms, tracking a device's status information, and retrieving statistical counts for each device registered with (added to) the driver.

Input/Output

The input / output block is responsible for configuring the line side and system side device interfaces. On the line-side, functions are provided to control the 2488 Mbps clock/data interface and the ring control port. On the system-side, functions are provided to control the Add/Drop Telecom Bus data interfaces and the Time-Slot Interchange (TSI).

Section Overhead

The Section Overhead block provides functions to control and monitor the Section Overhead processing. Read / Write access is given to the section trace message (J0). This message is compared with a configurable reference and mismatches are reported. Access is provided to the Section BIP-8 (B1) errors counter. Section Overhead alarms are detected and reported. For diagnostic purposes, errors can be introduced in the Section Overhead bytes.

Line Overhead

The Line Overhead block is responsible for configuring and monitoring the line overhead on both the receive and transmit sides. Read / Write access is given to the APS bytes (K1 and K2) and most of the other overhead bytes. Access is provided to the Line BIP-8 (B2) errors and REI. Line overhead alarms are detected and reported. For diagnostic purposes, errors can be introduced in the line overhead bytes. Additional functions are provided to configure the device to automatically insert line RDI, line REI and line AIS.

Path Overhead

The Path Overhead block configures and monitors the Path Overhead (and Path Overhead TU3 if available) on both the receive and the transmit sides. Read / Write access is given to the path trace message (J1) and the path signal label (C2). Both are compared with a configurable reference and mismatches are reported. Access is provided to the Path BIP-8 (B3) errors and REI counters. Path Overhead alarms are detected and reported. For diagnostic purposes, errors can be introduced in the Path Overhead bytes. Additional functions are provided to configure the device to automatically insert path RDI, path enhanced RDI, path REI and path AIS.

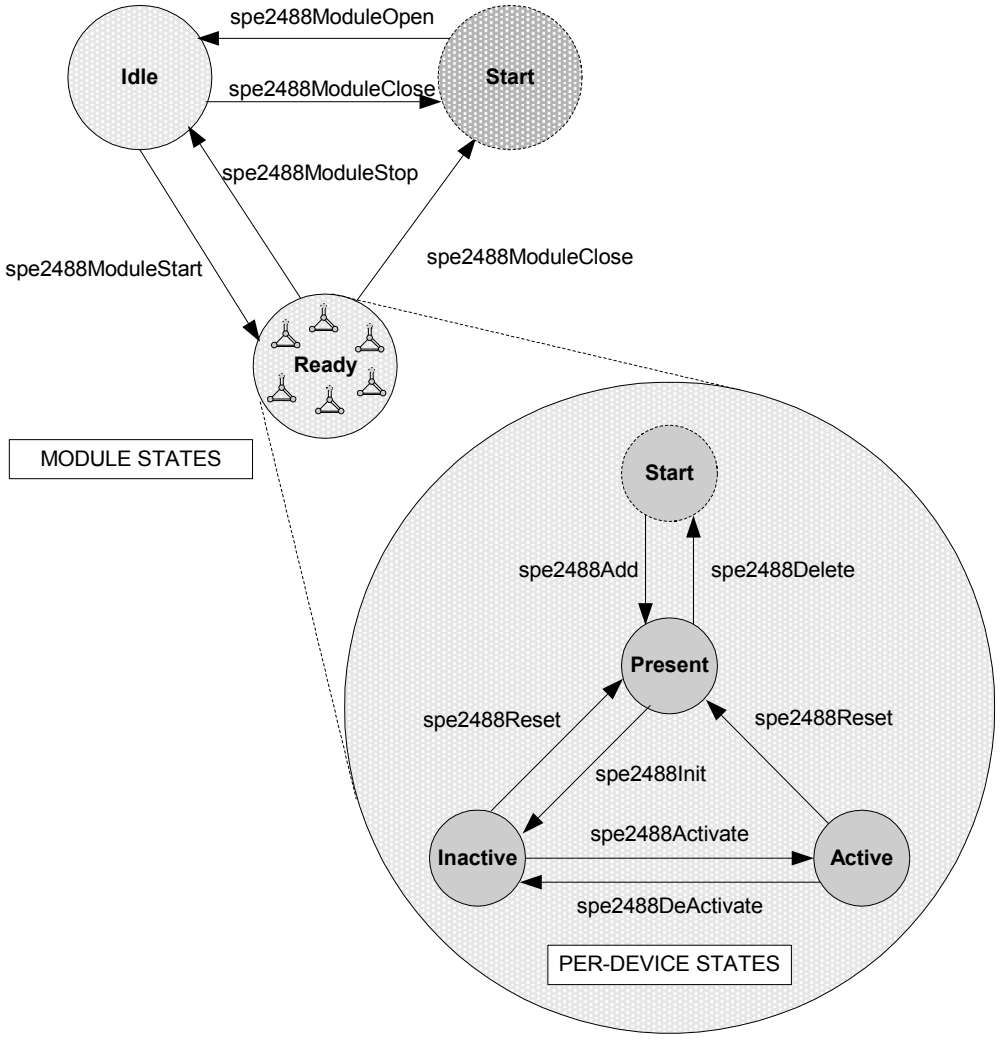
Device Diagnostics

The device diagnostics API can be used to isolate/identify problems within the device and its interfaces.

3.3 Software States

Figure 3 shows the software state diagram for the SPECTRA-2488 driver. State transitions occur on the successful execution of the corresponding transition functions shown. State information helps maintain the integrity of the MDB and DDB(s) by controlling the set of operations allowed in each state.

Figure 3: Driver Software States



Module States

The following is a description of the SPECTRA-2488 module states. See section 5.1 for a detailed description of the API functions that are used to change the module state.

Start

The driver module has not been initialized. In this state the driver does not hold any RTOS resources (e.g., memory and timers), has no running tasks, and performs no actions.

Idle

The driver module has been initialized successfully. The Module Initialization Vector (MIV) has been validated, the Module Data Block (MDB) has been allocated and loaded with current data, the per-device data structures have been allocated, and the RTOS has responded without error to all the requests sent to it by the driver.

Ready

This is the normal operating state for the driver module. This means that all RTOS resources have been allocated and the driver is ready for devices to be added. The driver module remains in this state while devices are in operation.

Device States

The following is a description of the SPECTRA-2488 per-device states. The state that is mentioned here is the software state as maintained by the driver, and not as maintained inside the device itself. See section 5.3 for a detailed description of the API functions that are used to change the per-device state.

Start

The device has not been initialized. In this state the device is unknown to the driver and performs no actions. There is a separate flow for each device that can be added, and they all start here.

Present

The device has been successfully added. A Device Data Block (DDB) has been associated to the device and updated with the user context; and a device handle has been given to the user. In this state, the device performs no actions.

Inactive

In this state the device is configured, but all data functions have been de-activated including interrupts and alarms, and status and statistics functions.

Active

This is the normal operating state for the device. In this state, either interrupt servicing or polling is enabled.

3.4 Processing Flows

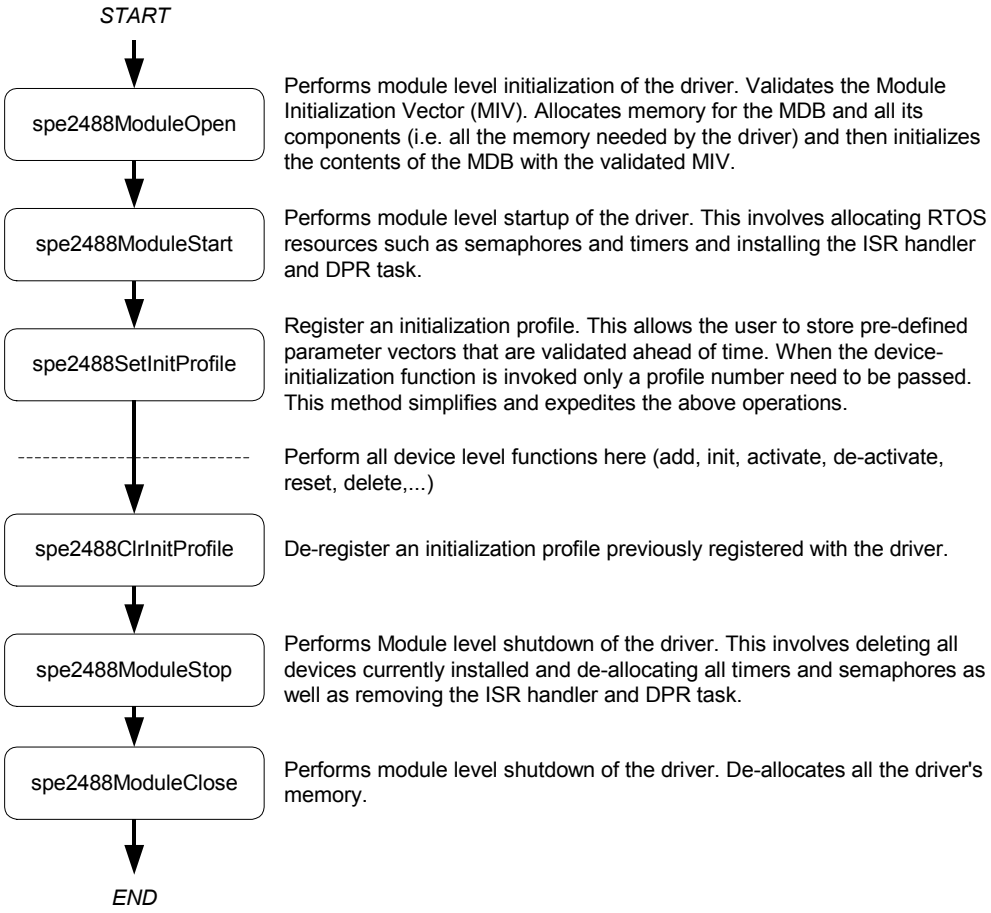
This section describes the main processing flows of the SPECTRA-2488 driver components.

The flow diagrams presented here illustrate the sequence of operations that take place for different driver functions. The diagrams also serve as a guide to the application programmer by illustrating the sequence in which the application must invoke the driver API.

Module Management

The following diagram illustrates the typical function call sequences that occur when initializing or shutting down the SPECTRA-2488 driver module.

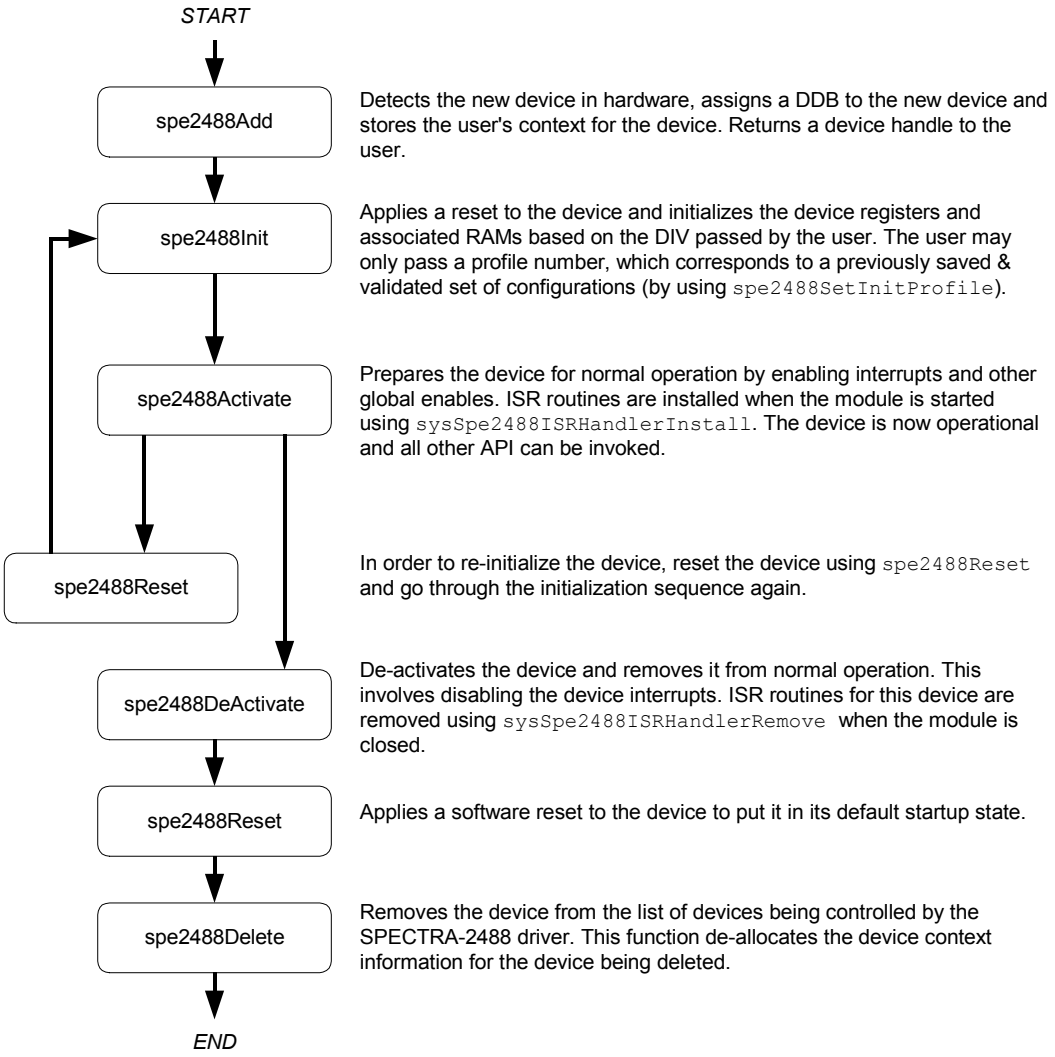
Figure 4: Module Management Flow Diagram



Device Management

The following figure shows the typical function call sequences that the driver uses to add, initialize, re-initialize, and delete the SPECTRA-2488 device.

Figure 5: Device Management Flow Diagram



3.5 Interrupt Servicing

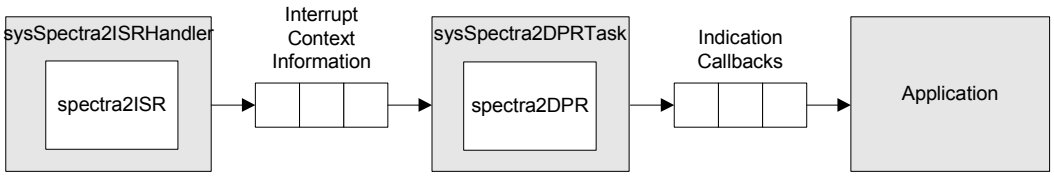
The SPECTRA-2488 driver services device interrupts using an Interrupt-Service Routine (ISR) that traps interrupts and a Deferred-Processing Routine (DPR) that actually processes the interrupt conditions and clears them. This lets the ISR execute quickly and exit. Most of the time-consuming processing of the interrupt conditions is deferred to the DPR by queuing the necessary interrupt-context information to the DPR task. The DPR function runs in the context of a separate task within the RTOS.

Note: Since the DPR task processes potentially serious interrupt conditions, you should set the DPR task's priority higher than that of the application task interacting with the SPECTRA-2488 driver.

The driver provides system-independent functions, `spe2488ISR` and `spe2488DPR`. You must fill in the corresponding system-specific functions, `sysSpe2488ISRHandler` and `sysSpe2488DPRTask`. The system-specific functions isolate the system-specific communication mechanism (between the ISR and DPR) from the system-independent functions, `spe2488ISR` and `spe2488DPR`.

Figure 6 illustrates the interrupt service model used in the SPECTRA-2488 driver design.

Figure 6: Interrupt Service Model



Note: Instead of using an interrupt service model, you can use a polling service model in the SPECTRA-2488 driver to process the device's event-indication registers (see page 26).

Calling spe2488ISR

An interrupt handler function, which is system dependent, must call `spe2488ISR`. But first, the low-level interrupt-handler function must trap the device interrupts. You must implement this function to fit your own system. As a reference, an example implementation of the interrupt handler (`sysSpe2488ISRHandler`) appears on page 137. You can customize this example implementation to suit your needs.

The interrupt handler that you implement (`sysSpe2488ISRHandler`) is installed in the interrupt vector table of the system processor. It is called when one or more SPECTRA-2488 devices interrupt the processor. The interrupt handler then calls `spe2488ISR` for each device in the active state that has interrupt processing enabled.

The spe2488ISR function reads from the master interrupt-status registers and the miscellaneous interrupt-status registers of the SPECTRA-2488. If at least one valid interrupt condition is found, then spe2488ISR fills an Interrupt-Service Vector (ISV) with this status information as well as the current device Handle. The spe2488ISR function also clears and disables all of the device interrupts that have been detected. The sysSpe2488ISRHandler function is then responsible for sending this ISV buffer to the DPR task.

Note: Normally you should save the status information for deferred processing by implementing a message queue. The interrupt handler sends the status information to the queue by the sysSpe2488ISRHandler.

Calling spe2488DPR

The sysSpe2488DPRTask function is a system-specific function that runs as a separate task within the RTOS. You should set the DPR task's priority higher than that of the application task(s) interacting with the SPECTRA-2488 driver. In the message-queue implementation model, this task has an associated message queue. The task waits for messages from the ISR on this message queue. When a message arrives, sysSpe2488DPRTask calls the DPR (spe2488DPR) with the received ISV.

After that, spe2488DPR processes the status information and takes appropriate action based on the specific interrupt condition detected. The nature of this processing can differ from system to system. Therefore, spe2488DPR calls different indication callbacks for different interrupt conditions.

Typically, you should implement these callback functions as simple message posting functions that post messages to an application task. However, you can implement the indication callback to perform processing within the DPR task context and return without sending any messages. In this case, ensure that this callback function does not call any API functions that would change the driver's state, such as spe2488Delete. Also, ensure that the callback function is non-blocking because the DPR task executes while SPECTRA-2488 interrupts are disabled. You can customize these callbacks to suit your system. See page 129 for example implementations of the callback functions.

Note: Since the spe2488ISR and spe2488DPR routines themselves do not specify a communication mechanism, you have full flexibility in choosing a communication mechanism between the two. A convenient way to implement this communication mechanism is to use a message queue, which is a service that most RTOSs provide.

You must implement the two system specific functions, sysSpe2488ISRHandler and sysSpe2488DPRTask. When the driver calls sysSpe2488ISRHandlerInstall, the application installs sysSpe2488ISRHandler in the interrupt vector table of the processor, and the sysSpe2488DPRTask function is spawned as a task by the application. The sysSpe2488ISRHandlerInstall function also creates the communication channel between sysSpe2488ISRHandler and sysSpe2488DPRTask. This communication channel is most commonly a message queue associated with the sysSpe2488DPRTask.

Similarly, during the removal of interrupts, the driver removes `sysSpe2488ISRHandler` from the microprocessor's interrupt vector table and deletes the task associated with `sysSpe2488DPRTask`.

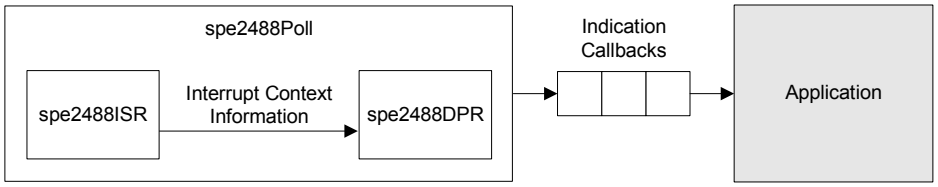
As a reference, this manual provides example implementations of the interrupt installation and removal functions in Section 6.2. You can modify these functions to suit your specific needs.

Calling spe2488Poll

Instead of using an interrupt service model, you can use the polling service model in the SPECTRA-2488 driver to process the device's event-indication registers.

Figure 7 illustrates the polling service model used in the SPECTRA-2488 driver design.

Figure 7: Polling Service Model



In polling mode, the application is responsible for calling `spe2488Poll` often enough to service any pending error or alarm conditions. When `spe2488Poll` is called, the `spe2488ISR` function is called internally.

The `spe2488ISR` function reads from the master interrupt-status registers and the miscellaneous interrupt-status registers of the SPECTRA-2488. If at least one valid interrupt condition is found, then `spe2488ISR` fills an Interrupt-Service Vector (ISV) with this status information as well as the current device Handle. The `spe2488ISR` function also clears and disables all the device interrupts that have been detected. In polling mode, this ISV buffer is passed to the DPR task by calling `spe2488DPR` internally.

3.6 Initialization of the Device for Various Payload Mappings

The SPECTRA-2488 driver allows you to configure the device for all legal payload mappings. Whether the device is used in single- or quad-mode, the Device Initialization Vector (DIV) provides 4 variables that fully describe the payload mapping: `sts12csl`, `sts12c`, `tug3` and `sts3c`. Table 2 describes each of these variables in more details.

The `sts12csl` variable introduces the concept of master/slave STS-12 slices. By specifying that a slice is a slave, you specify that it is part of a bigger payload, such as STS-24c, STS-36c or STS-48c. The first STS-12 slice of a STS-24c, STS-36c or STS-48c payload is the master slice while the remaining slices are the slaves. Note that a STS-12c slice is a master STS-12 slice with no slave.

Table 2: Payload Mapping Parameters

Field Name	Field Type	Field Description
sts12csl[4]	UINT1	This array of 4 elements allows you to configure each STM-4 slice as being part of a STS-24c, a STS-36c or a STS-48c payload. 1 = this STM-4 (STS-12) is a slave concatenated payload 0 = this is a master slice
sts12c[4]	UINT1	This array of 4 elements allows you to configure each STM-4 slice as being an STS-12c or part of a bigger concatenated payload (STS-24c, STS-36c or STS-48c). 1 = this STM-4 (STS-12) is a concatenated payload 0 = this STM-4 (STS-12) is not a concatenated payload
tug3[4][4]	UINT1	This array of 16 elements allows you to identify where the TUG-3s are located in the payload, if there are any. 1 = this STM-1 (STS-3) payload is a TUG3 0 = this STM-1 (STS-3) payload is not a TUG3
sts3c[4][4]	UINT1	This array of 16 elements allows you to identify where the STS-3c are located in the payload, if there are any. 1 = this STS-3 payload is concatenated. 0 = this STS-3 payload is not concatenated.

The following tables provide some example configurations that should help you understand how to configure the driver for any legal payload mapping. Table 3 shows how to configure an all STS-1 payload. Table 4 and Table 5 provide examples of master/slave concatenation. Table 6 presents a mixed payload configuration.

Note that an ‘x’ inside a table indicates that the corresponding value is ignored.

Table 3 provides the configuration that you should use for a payload that contains all STS-1s. In this example, all variables are set to 0 to indicate that there is no STS-12c, no TUG-3, and no STS-3c. This is the default configuration: all STS-1.

Table 3: 48xSTS-1 Configuration

Field Name	i	1	2	3	4

	j	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
sts12csl[i]		0				0				0				0			
sts12c[i]		0				0				0				0			
tug3[i][j]		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
sts3c[i][j]		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		4 x 3xSTS-1				4 x 3 x STS-1				4 x 3 x STS-1				4 x 3 x STS-1			

Table 4 illustrates how to configure a STS-48c payload. The first slice has `sts12c` set to 1 and `sts12csl` set to 0. This indicates that it is the master slice of either a STS-12c, a STS-24c, a STS-36c or a STS-48c. The last three slices have both `sts12c` and `sts12csl` set to 1, which indicates that they are all slave slices. This table shows one master with three slaves: this is a STS-48c.

Table 4: STS-48c Configuration

Field Name	i	1				2				3				4			
	j	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
sts12csl[i]		0				1				1				1			
sts12c[i]		1				1				1				1			
tug3[i][j]		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
sts3c[i][j]		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
		STS-48c															

Table 5 illustrates how to configure a STS-24c with two adjacent STS-12c. Slices 1, 2 and 4 have `sts12c` set to 1 and `sts12csl` set to 0. This indicates that they are master concatenated slices. Slice 3 has both `sts12c` and `sts12csl` set to 1, which indicates that it is a slave. There is a master, a master and a slave, and then another master: slice 1 is a STS-12c, slices 2 and 3 form a STS-24c, and slice 4 is a STS-12c.

Table 5: STS-12c / STS-24c / STS-12c Configuration

Field Name	i	1				2				3				4			
	j	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4

Field Name	i	1				2				3				4			
	j	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
sts12csl[i]		0				0				1				0			
sts12c[i]		1				1				1				1			
tug3[i][j]		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
sts3c[i][j]		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
		STS-12c				STS-24c								STS-12c			

Table 6 illustrates a mixed payload with a combination of STS-3c, STS-12c, TUG3 and STS-1. In the first slice, all variables are set to 0 except for `sts3c`. This is a slice that contains all STS-3c. Slice 2 has a 1 in `sts12c` but a 0 in `sts12csl`, which indicates a master concatenated slice. Slice 3 has both `sts12c` and `sts12csl` set to 0, and therefore slice 2 can only be a STS-12c. Slice 3 has `tug3` and `sts3c` set to 1, this slice is all TUG-3s. Slice 4 has all its variables set to 0, which indicates that it is an all STS-1 slice.

Table 6: 4xSTS-3c/STS-12c/4xTUG-3/12xSTS-1 Configuration

Field Name	I	1				2				3				4			
	j	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
sts12csl[i]		0				0				0				0			
sts12c[i]		0				1				0				0			
tug3[i][j]		0	0	0	0	x	x	x	x	1	1	1	1	0	0	0	0
sts3c[i][j]		1	1	1	1	x	x	x	x	x	x	x	x	0	0	0	0
		4 x STS-3c				STS-12c				4 x TUG3				4 x 3 x STS-1			

4 DATA STRUCTURES

This section describes the elements of the driver that configure or control its behavior and therefore should be of interest to the application programmer. Included here are the constants, variables, and structures that the SPECTRA-2488 device driver uses to store initialization, configuration, and statistics information. For more information on our naming conventions, the reader is referred to Appendix C.

Note: All (STM-4, AU-3) arrays that are used to pass 4 STM-4 and 12 AU-3 information are shown as arrays of size `[4][12]` in this document. However, in the driver source code, these arrays are declared to be of size `[4+1][12+1]` with the zeroeth element left unused. For example, an `sts12c[4]` array shown in this document is declared in the source code as `sts12c[4+1]`. The zeroeth element, `sts12c[0]`, is left unused.

4.1 Constants

The following constants are used throughout the driver code:

- `<SPECTRA-2488 ERROR CODES>`: error codes used throughout the driver code, returned by the API functions, and used in the global error number field of the MDB and DDB. For a description of the return codes refer to Appendix A (page 151)
- `SPE2488_MAX_DEVS`: defines the maximum number of devices that can be supported by this driver. This constant must not be changed without a thorough analysis of the consequences to the driver code.
- `SPE2488_MAX_INIT_PROFS`: defines the maximum number of initialization profiles that can be added to this driver. This constant must not be changed without a thorough analysis of the consequences to the driver code.
- `SPE2488_MOD_START`, `SPE2488_MOD_IDLE`, `SPE2488_MOD_READY` – are the three possible Module states (stored in `stateModule`).
- `SPE2488_START`, `SPE2488_PRESENT`, `SPE2488_ACTIVE`, `SPE2488_INACTIVE` – are the four possible device states (stored in `stateDevice`).

4.2 Structures Passed by the Application

These structures are defined for use by the application and are passed as arguments to functions within the driver. These structures are the Module Initialization Vector (MIV), the Device Initialization Vector (DIV) and the ISR mask.

Module Initialization Vector: MIV

Passed via the `spe2488ModuleOpen` call, this structure contains all of the information needed by the driver to initialize and connect to the RTOS. It contains the following fields:

- `maxDevs` is used to inform the driver how many devices will be operating concurrently during this session. The number is used to calculate the amount of memory that will be allocated by the driver. The maximum value that can be passed is `SPE2488_MAX_DEVS` (see section 4.1).
- `maxInitProfs` is used to inform the driver of how many Initialization profiles can be saved during this session. The number is used to calculate the amount of memory that will be allocated by the driver. The maximum value that can be passed is `SPE2488_MAX_INIT_PROFS` (see section 4.1).

Table 7: SPECTRA-2488 Module Initialization Vector: sSPE2488_MIV

Field Name	Field Type	Field Description
`perrModule`	`INT4 *`	(pointer to) `errModule` (see description in the MDB)
`maxDevs`	`UINT2`	Maximum number of devices supported during this session
`maxInitProfs`	`UINT2`	Maximum number of initialization profiles

Device Initialization Vector: DIV

Passed via the `spe2488Init` call, this structure contains all of the information needed by the driver to initialize a SPECTRA-2488 device. This structure is also passed via the `spe2488SetInitProfile` call when used as an initialization profile. It contains the following fields:

- `valid` indicates that this initialization profile has been properly initialized and may be used by the user. This field should be ignored when the DIV is passed directly.
- `pollISR` is a flag that indicates the type of interrupt servicing the driver is to use. The choices are 'polling' (`SPE2488_POLL_MODE`), and 'interrupt driven' (`SPE2488_ISR_MODE`). When configured in polling, the Interrupt capability of the device is NOT used, and the user is responsible for calling `spe2488Poll` periodically. The actual processing of the event information is the same for both modes.

- cbackIO, cbackSOH, cbackLOH, cbackRPOH, cbackRPOHTU3, cbackTPOH, cbackDPGM and cbackAPGM are used to pass the address of application functions that will be used by the DPR to inform the application code of pending events. If these fields are set as NULL, then any events that might cause the DPR to 'call back' the application will be processed during ISR processing but will be ignored by the DPR.

Table 8: SPECTRA-2488 Device Initialization Vector: sSPE2488_DIV

Field Name	Field Type	Field Description
valid	UINT2	Indicates that this structure is valid
pollISR	eSPE2488_POLL	Indicates the type of ISR / polling to do
cbackIO	void *	Address for the callback function for Input / Output (IO) Events
cbackSOH	void *	Address for the callback function for Section Overhead (SOH) Events
cbackLOH	void *	Address for the callback function for Line Overhead (LOH) Events
cbackRPOH	void *	Address for the callback function for Receive Path Overhead (RPOH) Events
cbackRPOHTU3	void *	Address for the callback function for Receive Path Overhead TU3 (RPOH-TU3) Events
cbackTPOH	void *	Address for the callback function for Transmit Path Overhead (TPOH) Events
cbackDPGM	void *	Address for the callback function for DROP Bus PRBS Generator/Monitor (DPGM) Events
cbackAPGM	void *	Address for the callback function for ADD Bus PRBS Generator/Monitor (APGM) Events
sts12csl[4]	UINT1	When non-zero, this STM-4 (STS-12) is a slave concatenated payload
sts12c[4]	UINT1	When non-zero, this STM-4 (STS-12) is a concatenated payload.
tug3[4][4]	UINT1	When non-zero, this STM-1 (STS-3) payload is a TUG3.

Field Name	Field Type	Field Description
sts3c[4][4]	UINT1	When non-zero, this STS-3 payload is concatenated.
cfgSF[4]	sSPE2488_CFG_SFSD	Signal failure configuration.
cfgSD[4]	sSPE2488_CFG_SFSD	Signal degrade configuration.
cfgCnt	sSPE2488_CFG_CNT	Counter configuration.

PRBS Generator and Monitor Configuration (CFG_PRBS)

This structure contains all the fields needed to configure the PRBS generators and monitors. The same structure is used to configure a monitor or a generator; however, some fields are used to configure a generator only. They should be ignored when configuring a monitor.

Table 9: PRBS Generator and Monitor Configuration: sSPE2488_CFG_PRBS

Field Name	Field Type	Field Description
enable	UINT1	When non-zero, enables the generator/monitor
amode	UINT1	Configures for telecombus when zero, autonomous mode otherwise
invPrbs	UINT1	When non-zero, PRBS data is inverted
b1e1Ena	UINT1	When non-zero, indicates to monitor/generate B1 and E1.
b1	UINT1	Expected B1 byte / B1 byte to transmit (E1 is the complement of this.
seqPrbs	UINT1	Pattern type 0 for PRBS, 1 for sequential
s	UINT1	S bits to be inserted in bits 2 and 3 of H1. Only used in autonomous mode and when processing concatenated payloads. (Generator Only)
lfsrSeed	UINT4	LFSR seed (23 bits)

Signal Failure/Signal Degrade Configuration (CFG_SFSD)

This structure contains all the fields needed to configure the signal failure (SF) and signal degrade (SD) accumulation period and thresholds.

Table 10: Counters Config: sSPE2488_CFG_SFSD

Field Name	Field Type	Field Description
enable	UINT1	Enables SF/SD bit error rate monitoring
accumPeriod	UINT4	The number of 8 KHz frames used to accumulate the B2 error subtotal.
saturation	UINT4	Allowable number of B2 errors that can be accumulated during an evaluation window before a SF or SD threshold event is declared.
declaring	UINT4	The threshold for the declaration of the SF or SD alarm.
clearing	UINT4	The threshold of maximum B2 allowed errors for clearing the SF or SD alarm.
clearMode	UINT1	When non-zero, the clearing window size is 8x that of the declaring window. Otherwise they are both of the same size.
satuMode	UINT1	When non-zero, the number of accumulated B2 errors is limited to the saturation threshold.

Config Counters (CFG_CNT)

The user passes this structure when invoking `spe2488CfgStats`. It allows the user to configure the behaviour of the device counts.

Table 11: SPECTRA-2488 Statistic Counters: sSPE2488_CFG_CNT

Field Name	Field Type	Field Description
`sohBipAccBlk[4]`	`UINT1`	Controls the accumulation of Section BIP errors. When non-zero, Section BIP errors are counted by block
`lohReiAccBlk[4]`	`UINT1`	Controls the extraction and accumulation of Line REI errors from the M1 byte. When non-zero, Line REI are counted by block
`lohReiCntBlk[4]`	`UINT1`	Controls the indication of Line REI errors. When non-zero, Line REI errors are counted by block
`lohBipCntBlk[4]`	`UINT1`	Controls the indication of Line BIP errors. When non-zero, Line BIP errors are counted by block
`lohBipAccBlk[4]`	`UINT1`	Controls the accumulation of Line BIP errors. When non-zero, Line BIP errors are counted by block
`rpohReiAccBlk[4][12]`	`UINT1`	Controls the extraction and accumulation of Path REI errors from the G1 byte. When non-zero, Path REI are counted by block
`rpohReiCntBlk[4][12]`	`UINT1`	Controls the indication of Path REI errors. When non-zero, Path REI errors are counted by block
`rpohBipCntBlk[4][12]`	`UINT1`	Controls the indication of Path BIP errors. When non-zero, Path BIP errors are counted by block
`rpohBipAccBlk[4][12]`	`UINT1`	Controls the accumulation of Path BIP errors. When non-zero, Path BIP errors are counted by block
`rpohReiAccBlkTu3[4][12]`	`UINT1`	Controls the extraction and accumulation of Path REI errors from the G1 byte. When non-zero, Path REI are counted by block

Field Name	Field Type	Field Description
rpohReiCntBlkTu3[4][12]	UINT1	Controls the indication of Path REI errors. When non-zero, Path REI errors are counted by block
rpohBipCntBlkTu3[4][12]	UINT1	Controls the indication of Path BIP errors. When non-zero, Path BIP errors are counted by block
rpohBipAccBlkTu3[4][12]	UINT1	Controls the accumulation of Path BIP errors. When non-zero, Path BIP errors are counted by block
tpohReiAccBlk[4][12]	UINT1	Controls the extraction and accumulation of Path REI errors from the G1 byte. When non-zero, Path REI are counted by block
tpohReiCntBlk[4][12]	UINT1	Controls the indication of Path REI errors. When non-zero, Path REI errors are counted by block
tpohBipCntBlk[4][12]	UINT1	Controls the indication of Path BIP errors. When non-zero, Path BIP errors are counted by block
tpohBipAccBlk[4][12]	UINT1	Controls the accumulation of Path BIP errors. When non-zero, Path BIP errors are counted by block

ISR Enable/Disable Mask

Passed via the `spe2488SetMask`, the `spe2488GetMask`, and the `spe2488ClearMask` calls, this structure contains all the information needed by the driver to enable and disable any of the interrupts in the SPECTRA-2488

Table 12: SPECTRA-2488 ISR Mask: sSPE2488_MASK

Field Name	Field Type	Field Description
`master[4]`	`UINT2`	Master interrupt enable
`aparrerr[4]`	`UINT2`	Add parity error
`dropCoap`	`UINT2`	DROP Change of active page
`addCoap`	`UINT2`	ADD Change of active page
`sbipe[4]`	`UINT2`	Section BIP error
`los[4]`	`UINT2`	Loss of Signal
`lof[4]`	`UINT2`	Loss of Frame
`oof[4]`	`UINT2`	Out of frame
`tiu[4]`	`UINT2`	Section trace identifier unstable
`tim[4]`	`UINT2`	Section trace identifier mismatch
`sfber[4]`	`UINT2`	Signal failure
`sdber[4]`	`UINT2`	Signal degrade
`lreie[4]`	`UINT2`	Line remote error indication error
`lbipe[4]`	`UINT2`	Line BIP error
`cossm[4]`	`UINT2`	Change of SSM message
`coaps[4]`	`UINT2`	Change of APS bytes
`apsbf[4]`	`UINT2`	APS byte failure
`lrdi[4]`	`UINT2`	Line remote defect indication
`lais[4]`	`UINT2`	Line alarm indication signal

Field Name	Field Type	Field Description
ptiu[4][12]	UINT2	Path trace identifier unstable
ptim[4][12]	UINT2	Path trace identifier mismatch
ptiuTu3[4][12]	UINT2	Path TU3 trace identifier unstable
ptimTu3[4][12]	UINT2	Path TU3 trace identifier mismatch
dropPaisc[4][12]	UINT2	DROP Path AIS concatenated
dropPlopc[4][12]	UINT2	DROP Path loss of pointer concatenated
dropPais[4][12]	UINT2	DROP Path AIS
dropPlop[4][12]	UINT2	DROP Path Loss of pointer
dropPje[4][12]	UINT2	DROP Positive justification event
dropNje[4][12]	UINT2	DROP Negative justification event
dropPreie[4][12]	UINT2	DROP Path remote error indication error
dropPbipe[4][12]	UINT2	DROP Path BIP error
dropCoperdi[4][12]	UINT2	DROP Change of path enhanced remote defect indication
dropPerdi[4][12]	UINT2	DROP Path enhanced remote defect indication
dropPrdi[4][12]	UINT2	DROP Path remote defect indication
dropPpdi[4][12]	UINT2	DROP Path payload defect indication
dropPuneq[4][12]	UINT2	DROP Path unequipped
dropPslm[4][12]	UINT2	DROP Path signal label mismatch
dropPslu[4][12]	UINT2	DROP Path signal label unstable
dropCopsl[4][12]	UINT2	DROP Change of path signal label
dropPpj[4][12]	UINT2	DROP positive pointer justification
dropNpj[4][12]	UINT2	DROP negative pointer justification
dropFovr[4][12]	UINT2	DROP FIFO overflow

Field Name	Field Type	Field Description
dropFudr[4][12]	UINT2	DROP FIFO underflow
dropMonErr[4][12]	UINT2	DROP monitor byte error
dropMonE1B1[4][12]	UINT2	DROP monitor change of B1/E1
dropMonSync[4][12]	UINT2	DROP monitor change of synchronization state
dropPloptr[4][12]	UINT2	DROP Path loss of pointer
dropPaisptr[4][12]	UINT2	DROP Path AIS
dropPaiscTu3[4][12]	UINT2	DROP Path AIS concatenated
dropPlopcTu3[4][12]	UINT2	DROP Path loss of pointer concatenated
dropPaisTu3[4][12]	UINT2	DROP Path AIS
dropPlopTu3[4][12]	UINT2	DROP Path Loss of pointer
dropPjeTu3[4][12]	UINT2	DROP Positive justification event
dropNjeTu3[4][12]	UINT2	DROP Negative justification event
dropPreieTu3[4][12]	UINT2	DROP Path remote error indication error
dropPbipeTu3[4][12]	UINT2	DROP Path BIP error
dropCoperdiTu3[4][12]	UINT2	DROP Change of path enhanced remote defect indication
dropPerdiTu3[4][12]	UINT2	DROP Path enhanced remote defect indication
dropPrdiTu3[4][12]	UINT2	DROP Path remote defect indication
dropPpdiTu3[4][12]	UINT2	DROP Path payload defect indication
dropPuneqTu3[4][12]	UINT2	DROP Path unequipped
dropPslmTu3[4][12]	UINT2	DROP Path signal label mismatch
dropPsluTu3[4][12]	UINT2	DROP Path signal label unstable
dropCopslTu3[4][12]	UINT2	DROP Change of path signal label

Field Name	Field Type	Field Description
dropPloptrTu3[4][12]	UINT2	DROP Path loss of pointer
dropPaisptrTu3[4][12]	UINT2	DROP Path AIS
addPpj[4][12]	UINT2	ADD positive pointer justification
addNpj[4][12]	UINT2	ADD negative pointer justification
addFovr[4][12]	UINT2	ADD FIFO overflow
addFudr[4][12]	UINT2	ADD FIFO underflow
addMonErr[4][12]	UINT2	ADD monitor byte error
addMonE1B1[4][12]	UINT2	ADD monitor change of B1/E1
addMonSync[4][12]	UINT2	ADD monitor change of synchronization state
addPaisc [4][12]	UINT2	ADD Path AIS concatenated
addPlopc [4][12]	UINT2	ADD Path loss of pointer concatenated
addPais [4][12]	UINT2	ADD Path AIS
addPlop [4][12]	UINT2	ADD Path Loss of pointer
addPje[4][12]	UINT2	ADD Positive justification event
addNje[4][12]	UINT2	ADD Negative justification event
addPreie[4][12]	UINT2	ADD Path remote error indication error
addPbipe[4][12]	UINT2	ADD Path BIP error
addCoperdi[4][12]	UINT2	ADD Change of path enhanced remote defect indication
addPerdi[4][12]	UINT2	ADD Path enhanced remote defect indication
addPrdi[4][12]	UINT2	ADD Path remote defect indication
addPpdi[4][12]	UINT2	ADD Path payload defect indication
addPuneq[4][12]	UINT2	ADD Path unequipped
addPslm[4][12]	UINT2	ADD Path signal label mismatch

Field Name	Field Type	Field Description
addPslu[4][12]	UINT2	ADD Path signal label unstable
addCopsl[4][12]	UINT2	ADD Change of path signal label

Statistics Counters (CNT)

This structure as well as its component structures is used by the statistics collection APIs to retrieve the device counts. The user can either collect all statistics at once by using spe2488GetCnt or collect statistics from individual blocks using spe2488GetCntSOHLOH, spe2488GetCntRPOH, and/or spe2488GetCntTPOH.

Table 13: SPECTRA-2488 Statistics Counters: sSPE2488_STAT_CNT

Field Name	Field Type	Field Description
cntSOH[4]	sSPE2488_STAT_CNT_SOH	Statistics counters of the Section Overhead (SOH)
cntLOH[4]	sSPE2488_STAT_CNT_LOH	Statistics counters of the Line Overhead (LOH)
cntRPOH[4]	sSPE2488_STAT_CNT_RPOH	Statistics counters of the Receive Path Overhead (RPOH)
cntTPOH[4]	sSPE2488_STAT_CNT_TPOH	Statistics counters for the Transmit Path Overhead (TPOH)

Section Overhead (SOH) Statistics Counters

Table 14: SPECTRA-2488 Section Overhead Statistics Counters: sSPE2488_STAT_CNT_SOH

Field Name	Field Type	Field Description
sohBip	UINT4	Section BIP errors counter

Line Overhead (LOH) Statistics Counters

Table 15: SPECTRA-2488 Line Overhead Status: sSPE2488_STAT_CNT_LOH

Field Name	Field Type	Field Description
lohBip	UINT4	Line BIP errors counter
lohRei	UINT4	Line REI error counter

Receive Path Overhead (RPOH) Statistics Counters

Table 16: SPECTRA-2488 Receive Path Processing Statistics Counters: sSPE2488_STAT_CNT_RPOH

Field Name	Field Type	Field Description
rpohBip[12]	UINT4	Path BIP error counter
rpohRei[12]	UINT4	Path REI error counter
rpohPosJustIn[12]	UINT4	Incoming positive pointer justification event counter
rpohNegJustIn[12]	UINT4	Incoming negative pointer justification event counter
rpohPosJustOut[12]	UINT4	Outgoing positive pointer justification event counter
rpohNegJustOut[12]	UINT4	Outgoing negative pointer justification event counter
rpohBipTu3[12]	UINT4	Path TU3 BIP error counter
rpohReiTu3[12]	UINT4	Path TU3 REI error counter
rpohPosJustInTu3[12]	UINT4	Incoming TU3 positive pointer justification event counter
rpohNegJustInTu3[12]	UINT4	Incoming TU3 negative pointer justification event counter

Field Name	Field Type	Field Description
rpohDPGMPrse[12]	UINT4	Number of PRBS byte errors detected since the last accumulation interval. Errors are only accumulated in the synchronized state and each PRBS data byte can have a maximum of 1 errors.

Transmit Path Overhead (TPOH) Statistics Counters

Table 17: SPECTRA-2488 Receive Path Processing Statistics Counters: sSPE2488_STAT_CNT_TPOH

Field Name	Field Type	Field Description
tpohBip[12]	UINT4	Path BIP error counter
tpohRei[12]	UINT4	Path REI error counter
tpohPosJustIn[12]	UINT4	Incoming positive pointer justification event counter – transmit side
tpohNegJustIn[12]	UINT4	Incoming negative pointer justification event counter – transmit side
tpohPosJustOut[12]	UINT4	Outgoing positive pointer justification event counter – transmit side
tpohNegJustOut[12]	UINT4	Outgoing negative pointer justification event counter – transmit side
tpohAPGMPrse[12]	UINT4	Number of PRBS byte errors detected since the last accumulation interval. Errors are only accumulated in the synchronized state and each PRBS data byte can have a maximum of 1 errors.

Alarm Status (STATUS)

These structures as well as their component structures are used by the statistics collection APIs to retrieve the current alarm status of the device. The user can either collect all statistics at once by using `spe2488GetStatus` or collect statistics from individual blocks using `spe2488GetStatusIO`, `spe2488GetStatusSOHLOH`, `spe2488GetStatusRPOH`, and/or `spe2488GetStatusTPOH`.

Table 18: SPECTRA-2488 Alarm Status: sSPE2488_STATUS

Field Name	Field Type	Field Description
`statIO`	`sSPE2488_STATUS_IO`	Alarm status of the Input / Output (IO)
`statSOH[4]`	`sSPE2488_STATUS_SOH`	Alarm status from the Section Overhead (SOH)
`statLOH[4]`	`sSPE2488_STATUS_LOH`	Alarm status from the Line Overhead (LOH)
`statRPOH[4]`	`sSPE2488_STATUS_RPOH`	Alarm status from the Receive Path Overhead (RPOH)
`statTPOH[4]`	`sSPE2488_STATUS_TPOH`	Alarm status from the Transmit Path Overhead (TPOH)

Input/Output Alarm Status (STATUS_IO)

Table 19: SPECTRA-2488 Input / Output Alarm Status: sSPE2488_STATUS_IO

Field Name	Field Type	Field Description
`rclkAct[4]`	`UINT2`	Monitors for low to high transitions on the internal receive clock of a given slice
`tclkAct[4]`	`UINT2`	Monitors for low to high transitions on the internal transmit clock of a given slice
`dropClkAct`	`UINT2`	Monitors for low to high transitions on the DROP system clock
`addClkAct`	`UINT2`	Monitors for low to high transitions on the ADD system clock
`addJ0J1Act[4]`	`UINT2`	Monitors for low to high transitions on the add J0J1 indication of a given slice

Field Name	Field Type	Field Description
addPldAct[4]	UINT2	Monitors for low to high transitions on the add payload indication of a given slice
addDataAct[4]	UINT2	Monitors for low to high transitions on the add data bus of a given slice

Section Overhead Alarm Status (STATUS_SOH)

Table 20: SPECTRA-2488 Section Overhead Alarm Status: sSPE2488_STATUS_SOH

Field Name	Field Type	Field Description
los	UINT2	LOS
lof	UINT2	LOF
oof	UINT2	OOF
tiu	UINT2	Section trace identifier unstable
tim	UINT2	Section trace identifier mismatch

Line Overhead Alarm Status (STATUS_LOH)

Table 21: SPECTRA-2488 Line Overhead Alarm Status: sSPE2488_STATUS_LOH

Field Name	Field Type	Field Description
sfber	UINT2	SF
sdber	UINT2	SD
apsbf	UINT2	APS byte failure
lrdi	UINT2	Line RDI
lais	UINT2	Line AIS

Receive Path Overhead Alarm Status (STATUS_RPOH)

Table 22: SPECTRA-2488 Receive Path Overhead Alarm Status: sSPE2488_STATUS_RPOH

Field Name	Field Type	Field Description

Field Name	Field Type	Field Description
ptiu[12]	UINT2	Path trace identifier unstable
ptim[12]	UINT2	Path trace identifier mismatch
dropPerdi[12]	UINT2	Path enhanced RDI
dropPrdi[12]	UINT2	Path RDI
dropPpdi[12]	UINT2	Path payload defect indication
dropPuneq[12]	UINT2	Path unequipped
dropPslm[12]	UINT2	Path signal label mismatch
dropPslu[12]	UINT2	Path signal label unstable
dropMonE1B1[12]	UINT2	Monitor B1/E1 error
dropMonSync[12]	UINT2	Monitor synchronization state change
dropPlopptr[12]	UINT2	Path loss of pointer
dropPaisptr[12]	UINT2	Path alarm indication signal
dropPlopc[12]	UINT2	Path loss of pointer concatenated
dropPaisc[12]	UINT2	Path alarm indication signal concatenated
dropPlop[12]	UINT2	Path loss of pointer
dropPais[12]	UINT2	Path alarm indication signal
ptiuTu3[12]	UINT2	Path trace identifier unstable
ptimTu3[12]	UINT2	Path trace identifier mismatch
dropPerdiTu3[12]	UINT2	Path enhanced RDI
dropPrdiTu3[12]	UINT2	Path RDI
dropPpdiTu3[12]	UINT2	Path payload defect indication
dropPuneqTu3[12]	UINT2	Path unequipped
dropPslmTu3[12]	UINT2	Path signal label mismatch

Field Name	Field Type	Field Description
dropPsluTu3[12]	UINT2	Path signal label unstable
dropPlopptrTu3[12]	UINT2	Path loss of pointer
dropPaisptrTu3[12]	UINT2	Path alarm indication signal
dropPlopcTu3[12]	UINT2	Path loss of pointer concatenated
dropPaiscTu3[12]	UINT2	Path alarm indication signal concatenated
dropPlopTu3[12]	UINT2	Path loss of pointer
dropPaisTu3[12]	UINT2	Path alarm indication signal

Transmit Path Overhead Alarm Status (STATUS_TPOH)

Table 23: SPECTRA-2488 Transmit Path Overhead Alarm Status: sSPE2488_STATUS_TPOH

Field Name	Field Type	Field Description
addPerdi[12]	UINT2	Path enhanced RDI
addPrdi[12]	UINT2	Path RDI
addPpdi[12]	UINT2	Path payload defect indication
addPuneq[12]	UINT2	Path unequipped
addPslm[12]	UINT2	Path signal label mismatch
addPslu[12]	UINT2	Path signal label unstable
addMonE1B1[12]	UINT2	Monitor B1/E1 error
addMonSync[12]	UINT2	Monitor synchronization state change
addPlopc[12]	UINT2	Path loss of pointer concatenated
addPaisc[12]	UINT2	Path alarm indication signal concatenated
addPlop[12]	UINT2	Path loss of pointer
addPais[12]	UINT2	Path alarm indication signal

Time-Slot Structure: TSLOT

Passed inside the Time-Slot Interchange (TSI) calls, this structure fully describes a single STS-1 timeslot.

Table 24: SPECTRA-2488 Time-Slot: sSPE2488_TSLOT

Field Name	Field Type	Field Description
stm4	UINT2	STM-4 index
au3	UINT2	AU3 index

4.3 Structures in the Driver's Allocated Memory

These structures are defined and used by the driver and are part of the context memory allocated when the driver is opened. These structures are the Module Data Block (MDB) and the Device Data Block (DDB).

Module Data Block: MDB

The MDB is the top-level structure for the Module. It contains configuration data about the Module level code and pointers to configuration data about the device level codes.

- errModule most of the Module API functions return a specific error code directly. When the returned code is SPE2488_FAILURE, this indicates that the top-level function was not able to carry the specified error code back to the application. Under those circumstances, the proper error code is recorded in this element. The element is the first in the structure so that the user can cast the MDB pointer into a INT4 pointer and retrieve the local error (this eliminates the need to include the MDB template into the application code).
- valid indicates that this structure has been properly initialized and may be read by the user.
- stateModule contains the current state of the Module and could be set to: SPE2488_MOD_START, SPE2488_MOD_IDLE or SPE2488_MOD_READY.

Table 25: SPECTRA-2488 Module Data Block: sSPE2488_MDB

Field Name	Field Type	Field Description
errModule	INT4	Global error indicator for module calls
valid	UINT2	Indicates that this structure has been initialized
stateModule	eSPE2488_MOD_STATE	Module state; can be one of the following IDLE or READY
maxDevs	UINT2	Maximum number of devices supported
numDevs	UINT2	Number of devices currently registered
maxInitProfs	UINT2	Maximum number of initialization profiles
pddb	sSPE2488_DDB *	(array of) Device Data Blocks (DDB) in context memory
pinitProfs	sSPE2488_DIV *	(array of) Initialization profiles in context memory

Device Data Block: DDB

The DDB is the top-level structure for each device. It contains configuration data about the device level code and pointers to configuration data about device level sub-blocks.

- errDevice most of the device API functions return a specific error code directly. When the returned code is SPE2488_FAILURE, this indicates that the top-level function was not able to carry the specific error code back top the application. In addition, some device functions do not return an error code. Under those circumstances, the proper error code is recorded in this element. The element is the first in the structure so that the user can cast the DDB pointer to a INT4 pointer and retrieve the local error (this eliminates the need to include the DDB template in the application code).
- valid indicates that this structure has been properly initialized and may be read by the user.
- stateDevice contains the current state of the device and could be set to: SPE2488_START, SPE2488_PRESENT, SPE2488_ACTIVE or SPE2488_INACTIVE.
- usrCtxt is a value that can be used by the user to identify the device during the execution of the callback functions. It is passed to the driver when spe2488Add is called and returned to the user in the DPV when a callback function is invoked. The element is unused by the driver itself and may contain any value.

Table 26: SPECTRA-2488 Device Data Block: sSPE2488_DDB

Field Name	Field Type	Field Description
errDevice	INT4	Global error indicator for device calls
valid	UINT2	Indicates that this structure has been initialized
stateDevice	eSPE2488_DEV_STATE	Device State; can be one of the following SPE2488_PRESENT, SPE2488_ACTIVE or SPE2488_INACTIVE
baseAddr	UINT2 *	Base address of the device
usrCtxt	void *	Stores the user's context for the device. It is passed as an input parameter when the driver invokes an application callback
profileNum	UINT2	Profile number used at initialization
pollISR	eSPE2488_POLL	Indicates the current type of ISR / polling
cbackIO	void *	Address for the callback function for Input / Output (IO) Events
cbackSOH	void *	Address for the callback function for Section Overhead (SOH) Events
cbackLOH	void *	Address for the callback function for Line Overhead (LOH) Events
cbackRPOH	void *	Address for the callback function for Receive Path Overhead (RPOH) Events
cbackRPOHTU3	void *	Address for the callback function for Receive Path Overhead TU3 (RPOH-TU3) Events
cbackTPOH	void *	Address for the callback function for Transmit Path Overhead (TPOH) Events
cbackDPGM	void *	Address for the callback function for DROP Bus PRBS Generator/Monitor (DPGM) Events

Field Name	Field Type	Field Description
cbackAPGM	void *	Address for the callback function for ADD Bus PRBS Generator/Monitor (APGM) Events
sts12csl[4]	UINT1	When non-zero, this STM-4 (STS-12) is a slave concatenated payload
sts12c[4]	UINT1	When non-zero, this STM-4 (STS-12) is a concatenated payload
tug3[4][4]	UINT1	When non-zero, this STM-1 (STS-3) payload is a TUG3
sts3c[4][4]	UINT1	When non-zero, this STS-3 payload is concatenated
cfgDropGen[4][12]	sSPE2488_CFG_PRBS	DROP Bus PRBS Generator configuration block
cfgDropMon[4][12]	sSPE2488_CFG_PRBS	DROP Bus PRBS Monitor configuration block
cfgAddGen[4][12]	sSPE2488_CFG_PRBS	ADD Bus PRBS Generator configuration block
cfgAddMon[4][12]	sSPE2488_CFG_PRBS	ADD Bus PRBS Monitor configuration block
cfgSF[4]	sSPE2488_CFG_SFSD	Signal fail accumulation period, saturation threshold, declaring threshold, and clearing threshold parameters.
cfgSD[4]	sSPE2488_CFG_SFSD	Signal degrade accumulation period, saturation threshold, declaring threshold, and clearing threshold parameters.
cfgCnt	sSPE2488_CFG_CNT	Counter configuration
ver	UINT2	Version as it appears in JTAG ID
manuID	UINT2	Manufacter ID as it appears in JTAG ID
partNum	UINT2	part number as it appears in JTAG ID

Field Name	Field Type	Field Description
quad	UINT2	Indicates the current state of the QUAD pin
threshold	sSPE2488_MASK	Application callback event thresholds
threshCntr	sSPE2488_MASK	Number of events that have occurred since the last time the thresholds have been reached.
mask	sSPE2488_MASK	Interrupt Enable Mask

4.4 Structures Passed through RTOS Buffers

Interrupt-Service Vector: ISV

This buffer structure is used to capture the status of the device (during a poll or ISR processing) for use by the Deferred-Processing Routine (DPR). It is the template for all device registers that are involved in exception processing. It is the application's responsibility to create a pool of ISV buffers (using this template to determine the buffer's size) when the driver calls the user-supplied sysSpe2488BufferStart function. An individual ISV buffer is then obtained by the driver via sysSpe2488ISVBufferGet and returned to the 'pool' via sysSpe2488ISVBufferRtn.

Table 27: SPECTRA-2488 Interrupt-Service Vector: sSPE2488_ISV

Field Name	Field Type	Field Description
deviceHandle	sSPE2488_HNDL	Handle to the device in cause
mask	sSPE2488_MASK	ISR mask filled with interrupt status

Deferred-Processing Vector: DPV

This block is used in two ways. First it is used to determine the size of the buffer that is required by the RTOS for use in the driver. Second it is the template for the data that is assembled by the DPR and then sent to the application code. Note: the application code is responsible for returning this buffer to the RTOS buffer pool.

The DPR divides the SPECTRA-2488 into 7 sections: IO, SOH, LOH, RPOH, TPOH, DPGM, and APG.M. 7 User-supplied callback routines (one per section) are used to inform the application which section of the device has caused the event being reported.

Table 28: SPECTRA-2488 Deferred-Processing Vector: sSPE2488_DPV

Field Name	Field Type	Field Description
event	SPE2488_DPR_EVENT	Event being reported
cause	UINT2	Reason for the Event

4.5 Global Variables

Although most of the variables within the driver are not meant to be used by the application code, there is one global variable that can be of great use to the application code.

spe2488Mdb: A global pointer to the Module Data Block (MDB). The content of this global variable should be considered read-only by the application.

- errModule: This structure element is used to store an error code that specifies the reason for an API function's failure. The field is only valid for functions that do not return an error code or when a value of SPE2488_FAILURE is returned.
- stateModule: This structure element is used to store the Module state (as shown in Figure 3).
- pddb[ ]: This is an array of pointers to the individual Device Data Blocks. The user is cautioned that a DDB is only valid if the valid flag is set. Note that the array of DDBs is in no particular order.
    - errDevice: This structure element is used to store an error code that specifies the reason for an API function's failure. The field is only valid for functions that do not return an error code or when a value of SPE2488_FAILURE is returned.
    - stateDevice: This structure element is used to store the device state (as shown in Figure 3).

5 APPLICATION PROGRAMMING INTERFACE

This section provides a detailed description of each function that is a member of the SPECTRA-2488 driver Application Programming Interface (API). API functions typically execute in the context of an application task.

Note: These functions are not re-entrant. This means that two application tasks can not invoke the same API at the same time. However the driver protects its data structures from concurrent accesses by the application and the DPR task.

5.1 Module Management

The module management is a set of API functions that are used by the Application to open, start, stop and close the driver module. These functions will take care of initializing the driver, allocating memory and all other RTOS resources needed by the driver. They are also used to change the module state. For more information on the module states, see the state diagram on page 26. For a typical module management flow diagram see page 29.

Opening the Driver Module: spe2488ModuleOpen

Performs module level initialization of the device driver. This involves allocating all of the memory needed by the driver and initializing the internal structures.

Prototype `INT4 spe2488ModuleOpen(sSPE2488_MIV *pmiv)`

Inputs `pmiv` : (pointer to) Module Initialization Vector

Outputs Places a pointer to `errModule` into the MIV passed by the Application.

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_MODULE_STATE`
`SPE2488_ERR_INVALID_MIV`
`SPE2488_ERR_MEM_ALLOC`

Valid States `SPE2488_MOD_START`

Side Effects Changes the MODULE state to `SPE2488_MOD_IDLE`

Closing the Driver Module: spe2488ModuleClose

Performs module level shutdown of the driver. This involves deleting all of the devices being controlled by the driver (by calling `spe2488Delete` for each device) and de-allocating all the memory allocated by the driver.

Prototype `INT4 spe2488ModuleClose(void)`

Inputs None

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_MODULE_STATE`

Valid States `SPE2488_MOD_IDLE`, `SPE2488_MOD_READY`

Side Effects Changes the MODULE state to `SPE2488_MOD_START`

Starting the Driver Module: spe2488ModuleStart

Connects the RTOS resources to the driver. This involves allocating semaphores and timers, initializing buffers, and installing the ISR handler and DPR task. Upon successful return from this function, the driver is ready to add devices.

Prototype `INT4 spe2488ModuleStart(void)`

Inputs None

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_MODULE_STATE`

Valid States `SPE2488_MOD_IDLE`

Side Effects Changes the MODULE state to `SPE2488_MOD_READY`

Stopping the Driver Module: spe2488ModuleStop

Disconnects the RTOS resources from the driver. This involves de-allocating semaphores and timers, freeing-up buffers, and uninstalling the ISR handler and the DPR task. If there are any registered devices, spe2488`Delete` is called for each.

Prototype `INT4 spe2488ModuleStop(void)`

Inputs None

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_MODULE_STATE`

Valid States `SPE2488_MOD_READY`

Side Effects Changes the MODULE state to `SPE2488_MOD_IDLE`

5.2 Profile Management

This section describes the functions that set, get and clear initialization profiles.

Initialization profiles allow the user to store pre-defined Device Initialization Vectors (DIV) that are validated ahead of time. When the device initialization function is invoked, only a profile number need to be passed. This method simplifies and expedites the initialization process.

Setting an Initialization Profile: spe2488SetInitProfile

Creates an initialization profile that is stored by the driver. A device can now be initialized by simply passing the initialization profile number.

Prototype `INT4 spe2488SetInitProfile(sSPE2488_DIV *pProfile, UINT2 *pProfileNum)`

Inputs
`pProfile` : (pointer to) initialization profile being added
`pProfileNum` : (pointer to) allocated memory

Outputs `pProfileNum` : profile number assigned by the driver

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_MODULE_STATE`
`SPE2488_ERR_INVALID_ARG`
`SPE2488_ERR_INVALID_PROFILE`

Valid States `SPE2488_MOD_IDLE, SPE2488_MOD_READY`

Side Effects None

Getting an Initialization Profile: spe2488GetInitProfile

Gets the contents of an initialization profile, given its profile number.

Prototype `INT4 spe2488GetInitProfile(UINT2 profileNum, sSPE2488_DIV *pProfile)`

Inputs
`profileNum` : initialization profile number
`pProfile` : (pointer to) allocated memory

Outputs `pProfile` : contents of the corresponding profile

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_MODULE_STATE`
`SPE2488_ERR_INVALID_ARG`
`SPE2488_ERR_INVALID_PROFILE_NUM`

Valid States `SPE2488_MOD_IDLE, SPE2488_MOD_READY`

Side Effects None

Clearing an Initialization Profile: spe2488ClrInitProfile

Deletes an initialization profile, given its profile number.

Prototype `INT4 spe2488ClrInitProfile(UINT2 profileNum)`

Inputs `profileNum` : initialization profile number

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_MODULE_STATE`
`SPE2488_ERR_INVALID_PROFILE_NUM`

Valid States `SPE2488_MOD_IDLE, SPE2488_MOD_READY`

Side Effects None

5.3 Device Management

The device management is a set of API functions that are used by the Application to control the device. These functions take care of initializing a device in a specific configuration, enabling the device's general activity, as well as enabling interrupt processing for that device. They are also used to change the software state for that device. For more information on the device states, see the state diagram on page 26. For a typical device management flow diagram see page 30.

Adding a Device: spe2488Add

This verifies the presence of a new device in the hardware and then returns a handle back to the user. The device handle is passed as a parameter of most of the device API Functions. It is used by the driver to identify the device on which the operation is to be performed.

Prototype
```
sSPE2488_HNDL spe2488Add(void *usrCtxt, UINT2
*baseAddr, INT4 **pperrDevice)
```

Inputs
`usrCtxt` : user context for this device
`baseAddr` : base address of the device
`pperrDevice` : (pointer to) allocated memory

Outputs ERROR code written to the MDB on failure:
`SPE2488_ERR_INVALID_ARG`
`SPE2488_ERR_INVALID_MODULE_STATE`
`SPE2488_ERR_DEVS_FULL`
`SPE2488_ERR_DEV_ALREADY_ADDED`
`SPE2488_ERR_INVALID_DEV`

`pperrDevice` : (pointer to) `errDevice` (inside the DDB)

Returns Success = Device Handle (to be used bas an argument to most of the SPECTRA-2488 APIs)
Failure = NULL (pointer)

Valid States `SPE2488_MOD_READY`

Side Effects Changes the DEVICE state to `SPE2488_PRESENT`

Deleting a Device: spe2488Delete

This function is used to remove the specified device from the list of devices being controlled by the SPECTRA-2488 driver. Deleting a device involves invalidating the DDB for that device and releasing its associated device handle.

Prototype
```
INT4 spe2488Delete(sSPE2488_HNDL deviceHandle)
```

Inputs `deviceHandle` : device Handle (from `spe2488Add`)

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`

Valid States `SPE2488_PRESENT, SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects Changes the DEVICE state to `SPE2488_PRESENT`

Initializing a Device: spe2488Init

This initializes the Device Data Block (DDB) associated with that device during `spe2488Add`; it also applies a soft reset to the device and configures it according to the DIV passed by the Application. If the DIV is passed as a NULL, the profile number is used. A profile number of zero indicates that all the register bits are to be left in their default state (after a soft reset). Note that the profile number is used only if the passed DIV is NULL.

Prototype
```
INT4 spe2488Init(sSPE2488_HNDL deviceHandle,
sSPE2488_DIV *pdiv, UINT2 profileNum)
```

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`pdiv` : (pointer to) Device Initialization Vector
`profileNum` : profile number (used only if `pdiv` is NULL)

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`

```
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_DIV
SPE2488_ERR_INVALID_PROFILE_NUM
```

Valid States `SPE2488_PRESENT`

Side Effects Changes the DEVICE state to `SPE2488_INACTIVE`

Updating the Configuration of a Device: spe2488Update

This updates the configuration of the device as well as the Device Data Block (DDB) associated with that device according to the DIV passed by the Application. The only difference between `spe2488Update` and `spe2488Init` is that no soft reset will be applied to the device. A profile number of zero is illegal.

Prototype
```
INT4 spe2488Update(sSPE2488_HNDL deviceHandle,
sSPE2488_DIV *pdiv, UINT2 profileNum)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`pdiv`	: (pointer to) Device Initialization Vector
`profileNum`	: profile number (used only if `pdiv` is NULL)

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_DIV`
`SPE2488_ERR_INVALID_PROFILE_NUM`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Resetting a Device: spe2488Reset

This applies a software reset to the SPECTRA-2488 device. It also resets all the DDB contents (except for the user context). This function is typically called before re-initializing the device (via `spe2488Init`).

Prototype
```
INT4 spe2488Reset(sSPE2488_HNDL deviceHandle)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`

Valid States SPE2488_PRESENT, SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects Changes the DEVICE state to SPE2488_PRESENT

Activating a Device: spe2488Activate

Restores the state of a device after a de-activate. Interrupts may be re-enabled.

Prototype INT4 spe2488Activate(sSPE2488_HNDL deviceHandle)

Inputs deviceHandle : device Handle (from spe2488Add)

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE

Valid States SPE2488_INACTIVE

Side Effects Changes the DEVICE state to SPE2488_ACTIVE

De-Activating a Device: spe2488DeActivate

De-activates the device from operation. Interrupts are masked and the device is put into a quiet state via enable bits.

Prototype INT4 spe2488DeActivate(sSPE2488_HNDL deviceHandle)

Inputs deviceHandle : device Handle (from spe2488Add)

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE

Valid States SPE2488_ACTIVE

Side Effects Changes the DEVICE state to SPE2488_INACTIVE

5.4 Device Read and Write

Reading from Device Registers: spe2488Read

This function is used to read a register of a specific SPECTRA-2488 device by providing the register number. It derives the actual address location based on the device handle and register number inputs. It then reads the contents of this address location using the system specific macro, `sysSpe2488Read`. Note that a failure to read returns a zero and any error indication is written to the associated DDB.

Prototype `UINT2 spe2488Read(sSPE2488_HNDL deviceHandle, UINT2 regNum)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`regNum` : register number

Outputs ERROR code written to the MDB:
`SPE2488_ERR_INVALID_DEV`
ERROR code written to the DDB:
`SPE2488_ERR_INVALID_REG`

Returns Success = value read
Failure = 0

Valid States SPE2488_PRESENT, SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects May affect registers that change after a read operation

Writing to Device Registers: spe2488Write

This function can be used to write to a register of a specific SPECTRA-2488 device by providing the register number. It derives the actual address location based on the device handle and register number inputs. It then writes the contents of this address location using the system specific macro, `sysSpe2488Write`. Note that a failure to write returns a zero and any error indication is written to the DDB.

Prototype `UINT2 spe2488Write(sSPE2488_HNDL deviceHandle, UINT2 regNum, UINT2 value)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`regNum` : register number
`value` : value to be written

Outputs ERROR code written to the MDB:
`SPE2488_ERR_INVALID_DEV`
ERROR code written to the DDB:
`SPE2488_ERR_INVALID_REG`

Returns Success = value written
Failure = 0

Valid States `SPE2488_PRESENT, SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects May change the configuration of the device

Reading from a block of Device Registers: spe2488ReadBlock

This function can be used to read a register block of a specific SPECTRA-2488 device by providing the starting register number and the size to read. It derives the actual start address location based on the device handle and starting register number inputs. It then reads the contents of this data block using multiple calls to the system specific macro, `sysSpe2488Read`. Note that a failure to read returns a zero and any error indication is written to the DDB. It is the user's responsibility to allocate enough memory for the block read.

Prototype
```
void spe2488ReadBlock(sSPE2488_HNDL deviceHandle,
UINT2 startRegNum, UINT2 size, UINT2 *pblock)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`startRegNum`	: starting register number
`size`	: size of the block to read
`pblock`	: (pointer to) the block to read

Outputs ERROR code written to the MDB:
`SPE2488_ERR_INVALID_DEV`
ERROR code written to the DDB:
`SPE2488_ERR_INVALID_REG`
`SPE2488_ERR_INVALID_ARG`

`pblock` : (pointer to) the block read

Returns None

Valid States `SPE2488_PRESENT, SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects May affect registers that change after a read operation

Writing to a Block of Device Registers: spe2488WriteBlock

This function can be used to write to a register block of a specific SPECTRA-2488 device by providing the starting register number and the block size. It derives the actual starting address location based on the device handle and starting register number inputs. It then writes the contents of this data block using multiple calls to the system specific macro, `sysSpe2488Write`. A bit from the passed block is only modified in the device's registers if the corresponding bit is set in the passed mask. Note that any error indication is written to the DDB

Prototype

```
void spe2488WriteBlock(sSPE2488_HNDL deviceHandle,
UINT2 startRegNum, UINT2 size, UINT2 *pblock, UINT2
*pmask)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`startRegNum`	: starting register number
`size`	: size of block to read
`pblock`	: (pointer to) block to write
`pmask`	: (pointer to) mask

Outputs

ERROR code written to the MDB:
`SPE2488_ERR_INVALID_DEV`
ERROR code written to the DDB:
`SPE2488_ERR_INVALID_REG`
`SPE2488_ERR_INVALID_ARG`

Returns

None

Valid States

`SPE2488_PRESENT, SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects

May change the configuration of the device

5.5 Input/Output

The Input/Output section provides functions to configure and control the Time-Slot Interchange (TSI).

Setting Timeslot Mapping Mode: spe2488IOSetMapMode

Sets the global mapping mode of all TSIs in the specified device.

Prototype `INT4 spe2488IOSetMapMode(sSPE2488_HNDL deviceHandle, UINT1 direction, UINT1 tsiMode)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`direction`	: direction (symmetry) 0 = Drop 1 = Add
`tsiMode`	: TSI mapping mode 0 = dynamic mode 1 = bypass mode 2 = 12-48c mode 3 = 48c-12 mode

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Getting Timeslot Mapping Mode: spe2488IOGetMapMode

Retrieves the global mapping mode of all TSIs in the specified device.

Prototype `INT4 spe2488IOGetMapMode(sSPE2488_HNDL deviceHandle, UINT1 direction, UINT1 *ptsiMode)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`direction`	: direction (symmetry) 0 = Drop 1 = Add
`ptsiMode`	: (pointer to) allocated memory

Outputs `ptsiMode` : TSI mapping mode
0 = dynamic mode
1 = bypass mode
2 = 12-48c mode
3 = 48c-12 mode

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Creating a Timeslot Mapping: spe2488IOMapSlot

Maps a source timeslot to one or more destination timeslots. This function supports a unicast or a multicast mapping.

Prototype
```
INT4 spe2488IOMapSlot(sSPE2488_HNDL deviceHandle,
UINT1 direction, UINT1 page, sSPE2488_TSLOT
*psrcSlot, sSPE2488_TSLOT *pdestSlot, UINT1
numDestSlots)
```

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`direction` : direction (symmetry)
0 = Drop
1 = Add
`page` : configuration page number
`psrcSlot` : (pointer to) source timeslot
`pdestSlot` : (pointer to) destination timeslot(s)
`numDestSlots` : number of destination timeslots

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Determining if a Timeslot is being Multicasted: spe2488IOIsMulticast

Informs the user of whether or not a source timeslot is being multicasted.

Prototype

```
INT4 spe2488IOIsMulticast(sSPE2488_HNDL
deviceHandle, UINT1 direction, UINT1 page,
sSPE2488_TSLOT *psrcSlot, UINT1 *pnumDestSlots)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`direction`	: direction (symmetry) 0 = Drop 1 = Add
`page`	: configuration page number
`psrcSlot`	: (pointer to) source timeslot
`pnumDestSlot`	: (pointer to) allocated memory

Outputs

`pnumDestSlot`	: number of destination timeslot(s)

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Getting Destination Timeslot(s): spe2488IOGetDestSlot

Retrieves a list of destination timeslots mapped to a specified source timeslot.

Prototype

```
INT4 spe2488IOGetDestSlot(sSPE2488_HNDL
deviceHandle, UINT1 direction, UINT1 page,
sSPE2488_TSLOT *psrcSlot, sSPE2488_TSLOT
*pdestSlot, UINT1 *pnumDestSlots)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`direction`	: direction (symmetry) 0 = Drop 1 = Add
`page`	: configuration page number
`psrcSlot`	: (pointer to) source timeslot
`pdestSlot`	: (pointer to) allocated memory
`pnumDestSlot`	: (pointer to) allocated memory

Outputs

`pdestSlot`	: array of destination timeslot(s)
`pnumDestSlot`	: number of destination timeslot(s)

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Getting Source Timeslot: spe2488IOGetSrcSlot

Retrieves the source timeslot mapped to a specified destination timeslot.

Prototype
```
INT4 spe2488IOGetSrcSlot(sSPE2488_HNDL
deviceHandle, UINT1 direction, UINT1 page,
sSPE2488_TSLOT *pdestSlot, sSPE2488_TSLOT
*psrcSlot)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`direction`	: direction (symmetry) 0 = Drop 1 = Add
`page`	: configuration page number
`pdestSlot`	: (pointer to) destination timeslot
`psrcSlot`	: (pointer to) allocated memory

Outputs `psrcSlot` : source timeslot

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Getting Active Configuration Page: spe2488IOGetPage

Retrieves the currently active timeslot interchange configuration page (0 or 1).

Prototype
```
INT4 spe2488IOGetPage(sSPE2488_HNDL deviceHandle,
UINT1 direction, UINT1 *ppage)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`direction`	: direction (symmetry) 0 = Drop 1 = Add
`ppage`	: (pointer to) allocated memory

Outputs

`ppage`	: active page number

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Setting Active Configuration Page: spe2488IOSetPage

Sets the currently active timeslot interchange configuration page (0 or 1).

Prototype
```
INT4 spe2488IOSetPage(sSPE2488_HNDL deviceHandle,
UINT1 direction, UINT1 page)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`direction`	: direction (symmetry) 0 = Drop 1 = Add
`page`	: active page number

Outputs None

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Inserting IDLE Data Pattern: spe2488IOInsertIdle

Insert arbitrary idle data pattern into a given destination timeslot.

Prototype `INT4 spe2488IOSetPage(sSPE2488_HNDL deviceHandle, UINT1 direction, UINT1 page)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`direction`	: direction (symmetry) 0 = Drop 1 = Add
`page`	: page number
`pdestSlot`	: (pointer to) destination timeslot
`idle`	: idle data pattern (valid range: 0-2047)

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

5.6 Section Overhead

The Section Overhead section provides functions to control and monitor the Section Overhead processing. Read / Write access is given to the section trace message (J0). This message is compared with a configurable reference and mismatches are reported. In addition, section BIP-8 (B1) errors are accumulated in a counter that can be read and cleared, and Section Overhead alarms are detected and reported. For diagnostic purposes, errors can be introduced in the Section Overhead bytes.

Selecting the Section Termination Mode: spe2488SOHTermination

This function enables or disables the section termination of a particular STM-4. There are two possible modes: no section termination (`SPE2488_NO_TERM`) and section termination (`SPE2488_TERM`). When configured for section termination, the internal registers are used for section overhead insertion.

Prototype `INT4 spe2488SOHTermination(sSPE2488_HNDL deviceHandle, UINT2 stm4, eSPE2488_TERM mode)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index

	mode	: section termination mode, one of the following: SPE2488_NO_TERM or SPE2488_TERM

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Reading and Setting the Section Trace Message (J0): spe2488SectionTraceMsg

This function retrieves and sets the section trace message (J0) in the Sonet/SDH Section Trace Buffer. Configuring the transmit trace message will result in modifying the trace message transmitted even if this STM-4 is not configured for section termination.

Note: It is the user's responsibility to ensure that the message pointer points to an area of memory large enough to hold the returned data.

Prototype

```
INT4 spe2488SectionTraceMsg(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 rw, UINT2 type,
UINT2 length, UINT1* pJ0)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
rw	: read/write flag, write if zero
type	: type of access 0 = tx section trace 1 = rx accepted section trace 2 = rx captured section trace 3 = rx expected section trace
length	: length of the message, 16 bytes long if zero 64 bytes long otherwise
pJ0	: (pointer to) the section trace message

Outputs

pJ0	: section trace message

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Writing the J0 Byte: spe2488SOHWriteJ0

This function writes the J0 byte into the transmit Section Overhead. This will overwrite the J0 byte with the passed value, and the section trace message will be ignored. If this STM-4 is not configured for section termination, this function will have no effect on the Section Overhead.

Prototype
```
INT4 spe2488SOHWriteJ0(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT1 J0)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
J0	: J0 byte to write

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Writing the Z0 Byte: spe2488SOHWriteZ0

This function writes the Z0 byte into the transmit Section Overhead. If this STM-4 is not configured for section termination, this function will have no effect on the Section Overhead.

Prototype
```
INT4 spe2488SOHWriteZ0(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT1 Z0)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
Z0	: Z0 byte to write

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Writing the D1-D3 Byte: spe2488SOHWriteD1D3

This function writes the D1-D3 byte into the transmit Section Overhead. If this STM-4 is not configured for section termination, this function will have no effect on the Section Overhead.

Prototype
```
INT4 spe2488SOHWriteD1D3(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT1 D1D3)
```

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`stm4` : STM-4 index
`D1D3` : D1D3 byte to write

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the E1 Byte: spe2488SOHWriteE1

This function writes the E1 byte into the transmit Section Overhead. If this STM-4 is not configured for section termination, this function will have no effect on the Section Overhead.

Prototype
```
INT4 spe2488SOHWriteE1(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT1 valE1)
```

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`stm4` : STM-4 index
`valE1` : E1 byte to write

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the F1 Byte: spe2488SOHWriteF1

This function writes the F1 byte into the transmit Section Overhead. If this STM-4 is not configured for section termination, this function will have no effect on the Section Overhead.

Prototype `INT4 spe2488SOHWriteF1(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT1 F1)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`F1`	: F1 byte to write

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing Out-of-Frame: spe2488SOHForceOOF

This function forces the receive Section Overhead out-of-frame.

Prototype `INT4 spe2488SOHForceOOF(sSPE2488_HNDL deviceHandle, UINT2 stm4)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`
`SPE2488_ERR_INVALID_DEVICE_STATE`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing Errors in the Framing Bytes: spe2488SOHDiagFB

This function enables the insertion of a single bit error continuously in the most significant bit (bit1) of the A1 Section Overhead framing byte. A1 bytes are set to 76H instead of F6H.

Prototype
```
INT4 spe2488SOHDiagFB(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT2 enable)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`enable`	: flag to start/stop error insertion

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing Section BIP-8 Errors: spe2488SOHDiagB1

This function enables the insertion of bit errors continuously in the B1 Section Overhead byte. Only those bits that are set in the passed mask are inverted.

Prototype
```
INT4 spe2488SOHDiagB1(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT1 maskB1, UINT2 enable)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`maskB1`	: mask identifying the bits to toggle in B1
`enable`	: flag to start / stop error insertion

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing Loss-of-Signal: spe2488SOHDiagLOS

This function enables insertion of zeros in the transmit outgoing stream.

Prototype `INT4 spe2488SOHDiagLOS(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT2 enable)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`enable`	: flag to start/stop error insertion

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

5.7 Line Overhead

The Line Overhead section is responsible for configuring and monitoring the line overhead on both receive and transmit sides. Read / Write access is given to the APS bytes (K1 and K2) and most of the other overhead bytes. Line BIP-8 (B2) errors and REI are accumulated in counters that can be read and cleared. Line overhead alarms are detected and reported. For diagnostic purposes, errors can be introduced in the line overhead bytes. Additional functions are provided to configure the device to automatically insert line RDI and AIS.

Selecting the Line Termination Mode: spe2488LOHTermination

This function enables or disables the line termination of a particular STM-4. There are three possible modes: no line termination (`SPE2488_NO_TERM`), line termination (`SPE2488_TERM`) and line termination with automatic alarm insertion (`SPE2488_TERM_WITH_AUTO_INSERT`). If selected, automatic error insertion in K1/K2 is performed. When configured for line termination, the internal registers are used for line overhead insertion.

Prototype `INT4 spe2488LOHTermination(sSPE2488_HNDL deviceHandle, UINT2 stm4, eSPE2488_TERM mode)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`mode`	: line termination mode, one of the following: `SPE2488_NO_TERM`, `SPE2488_TERM` or

SPE2488_TERM_WITH_AUTO_INSERT

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Configuring Signal Fail (SF) and Signal Degrade (SD): spe2488LOHCfgSFSD

This function configures the Signal Fail and Signal Degrade bit error rate monitoring capabilities of the SPECTRA-2488.

Prototype
```
INT4 spe2488LOHCfgSFSD(sSPE2488_HNDL deviceHandle,
UINT2 stm4, sSPE2488_CFG_SFSD *pcfgSF,
sSPE2488_CFG_SFSD *pcfgSD)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
pcfgSF	: (pointer to) SF configuration
pcfgSD	: (pointer to) SD configuration

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Reading the Received K1 and K2 Bytes: spe2488LOHReadK1K2

This function reads the K1 and K2 bytes from the received line overhead.

Prototype

```
INT4 spe2488LOHReadK1K2(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT1 *pK1, UINT1 *pK2)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`pK1`	: (pointer to) allocated memory
`pK2`	: (pointer to) allocated memory

Outputs

`pK1`	: K1 byte read
`pK2`	: K2 byte read

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States

`SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects

None

Writing the Transmitted K1 and K2 Bytes: spe2488LOHWriteK1K2

This function writes the K1 and K2 bytes into the transmit line overhead. If this STM-4 is not configured for line termination, this function will have no effect on the line overhead. Also, it this STM-4 is configured for automatic error insertion, the K1/K2 bytes are automatically generated by the device and this function has no effect.

Prototype

```
INT4 spe2488LOHWriteK1K2(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT1 K1, UINT1 K2)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`K1`	: K1 byte to write
`K2`	: K2 byte to write

Outputs None

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the D4-D12 Byte: spe2488LOHWriteD4D12

This function writes the D4-D12 bytes into the transmit line overhead. If this STM-4 is not configured for line termination, this function will have no effect on the line overhead.

Prototype
```
INT4 spe2488LOHWriteD4D12(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT1 D4D12)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`D4D12`	: D4D12 byte to write

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Reading the S1 Byte: spe2488LOHReadS1

This function reads the S1 byte from the receive line overhead.

Prototype
```
INT4 spe2488LOHReadS1(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT1 *pS1)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`pS1`	: (pointer to) allocated memory

Outputs

`pS1`	: S1 byte read

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the S1 Byte: spe2488LOHWriteS1

This function writes the S1 byte into the transmit line overhead. If this STM-4 is not configured for line termination, this function will have no effect on the line overhead.

Prototype `INT4 spe2488LOHWriteS1(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT1 S1)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`stm4` : STM-4 index
`S1` : S1 byte to write

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the Z1 Byte: spe2488LOHWriteZ1

This function writes the Z1 byte into the transmit line overhead. If this STM-4 is not configured for line termination, this function will have no effect on the line overhead.

Prototype `INT4 spe2488LOHWriteZ1(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT1 Z1)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`stm4` : STM-4 index
`Z1` : Z1 byte to write

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the Z2 Byte: spe2488LOHWriteZ2

This function writes the Z2 byte into the transmit line overhead. If this STM-4 is not configured for line termination, this function will have no effect on the line overhead.

Prototype `INT4 spe2488LOHWriteZ2(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT1 Z2)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`stm4` : STM-4 index
`Z2` : Z2 byte to write

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the E2 Byte: spe2488LOHWriteE2

This function writes the E2 byte into the transmit line overhead. If this STM-4 is not configured for line termination, this function will have no effect on the line overhead.

Prototype `INT4 spe2488LOHWriteE2(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT1 E2)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`stm4` : STM-4 index
`E2` : E2 byte to write

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Inserting Line AIS: spe2488LOHInsertLineAIS

When the enable flag is set, this function forces a Line-AIS insertion. When the enable flag is not set, this function resumes normal processing.

Prototype `INT4 spe2488LOHInsertLineAIS(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT2 enable)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`stm4` : STM-4 index
`enable` : flag to start / stop Line-AIS insertion

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Inserting Line Remote Defect Indication: spe2488LOHInsertLineRDI

This function enables the insertion of a transmit line remote defect indication (RDI). The Line RDI is inserted by transmitting the code 110 in bit positions 6, 7, and 8 of the K2 byte.

Prototype `INT4 spe2488LOHInsertLineRDI(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT2 enable)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`stm4` : STM-4 index
`enable` : flag to start / stop Line RDI insertion

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing Line BIP-8 Errors: spe2488LOHDiagB2

This function enables the insertion of bit errors continuously in each of the line BIP-8 bytes (B2 bytes). Only those bits that are set in the passed mask are inverted.

Prototype

```
INT4 spe2488LOHDiagB2(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT1 maskB2, UINT2 enable)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`maskB2`	: mask identifying the bits to toggle in B2
`enable`	: flag to start / stop B2 error insertion

Outputs None

Returns

Success = `SPE2488_SUCCESS`

Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing Errors in the Payload Pointer: spe2488LOHDiagH1H2

This function enables the insertion of errors continuously in the payload pointer bytes (H1 and H2 bytes). Only those bits that are set in the passed mask are inverted.

Prototype

```
INT4 spe2488LOHDiagH1H2(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT1 maskH1, UINT1
maskH2, UINT2 enable)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`maskH1`	: mask identifying the bits to toggle in H1
`maskH2`	: mask identifying the bits to toggle in H2
`enable`	: flag to start / stop error insertion

Outputs None

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

5.8 Path Overhead

The Path Overhead section configures and monitors the Path Overhead on both receive and transmit sides. Read / Write access is given to the path trace message (J1) and the path signal label (C2). Both are compared with a configurable reference and mismatches are reported. Path BIP-8 (B3) errors and REI are accumulated in a counter that can be read and cleared. Path Overhead alarms are detected and reported. For diagnostic purposes, errors can be introduced in the Path Overhead bytes. Additional functions are provided to configure the device to automatically insert path RDI, path enhanced RSI, and path AIS.

Selecting the Path Termination Mode: spe2488POHTermination

This function enables or disables the path termination of a particular (STM-4, AU3) slice. There are three possible modes: no path termination (SPE2488_NO_TERM), path termination (SPE2488_TERM) and path termination with automatic alarm insertion (SPE2488_TERM_WITH_AUTO_INSERT). If selected, automatic error insertion in G1 is performed. When configured for path termination, the internal registers are used for path overhead insertion.

Prototype

```
INT4 spe2488POHTermination(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, eSPE2488_TERM
mode)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
au3	: AU-3 index
mode	: path termination mode, one of the following: SPE2488_NO_TERM, SPE2488_TERM or SPE2488_TERM_WITH_AUTO_INSERT

Outputs None

Returns

Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Selecting the TU3 Path Termination Mode: spe2488POHTerminationTU3

This function enables or disables the TU3 path termination of a particular (STM-4, AU3) slice. There are three possible modes: no TU3 path termination (SPE2488_NO_TERM), TU3 path termination (SPE2488_TERM) and TU3 path termination with automatic alarm insertion (SPE2488_TERM_WITH_AUTO_INSERT). If selected, automatic error insertion in G1-TU3 is performed. When configured for TU3 path termination, the internal registers are used for TU3 path overhead insertion.

Prototype

```
INT4 spe2488POHTerminationTU3(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, eSPE2488_TERM
mode)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
au3	: AU-3 index
mode	: path-TU3 termination mode, one of the following: SPE2488_NO_TERM, SPE2488_TERM or SPE2488_TERM_WITH_AUTO_INSERT

Outputs None

Returns

Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Retrieving and Setting the Path Trace Messages: spe2488PathTraceMsg

This function retrieves and sets the current path trace message (J1) in the Sonet/SDH Path Trace Buffer. Note: It is the user's responsibility to make sure that the message pointer points to an area of memory large enough to hold the returned data.

Prototype

```
INT4 spe2488PathTraceMsg(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT2 rw,
UINT2 type, UINT2 length, UINT1* pJ1)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
au3	: AU-3 index
rw	: read/write flag, write if zero
type	: type of access 0 = tx path trace

		1 = rx accepted path trace 2 = rx captured path trace 3 = rx expected path trace
	`length`	: length of the message, 16 bytes long if zero 64 bytes long otherwise
	`pJ1`	: (pointer to) the path trace message
Outputs	`pJ1`	: updated path trace message

Returns	Success = `SPE2488_SUCCESS`
	Failure = `SPE2488_ERR_INVALID_DEV`
	`SPE2488_ERR_INVALID_DEVICE_STATE`
	`SPE2488_ERR_INVALID_ARG`	

Valid States	`SPE2488_ACTIVE, SPE2488_INACTIVE`	
Side Effects	None	

Retrieving and Setting the TU3 Path Trace Messages: spe2488PathTraceMsgTU3

This function retrieves and sets the current TU3 path trace message (J1-TU3) in the Sonet/SDH Path Trace Buffer. Note: It is the user's responsibility to make sure that the message pointer points to an area of memory large enough to hold the returned data. This function can only be called if this slice was configured as a TUG3.

Prototype

```
INT4 spe2488PathTraceMsgTU3(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT2 rw,
UINT2 type, UINT2 length, UINT1* pJ1)
```

Inputs	`deviceHandle`	: device Handle (from `spe2488Add`)
	`stm4`	: STM-4 index
	`au3`	: AU-3 index
	`rw`	: read/write flag, write if zero
	`type`	: type of access 0 = tx TU3 path trace 1 = rx accepted TU3 path trace 2 = rx captured TU3 path trace 3 = rx expected TU3 path trace
	`length`	: length of the TU3 message, 16 bytes long if zero 64 bytes long otherwise
	`pJ1`	: (pointer to) the TU3 path trace message
Outputs	`pJ1`	: updated TU3 path trace message
Returns	Success = `SPE2488_SUCCESS` Failure = `SPE2488_ERR_INVALID_DEV`	

```
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG
```

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Receive Path Overhead (RPOH)

Retrieving and Setting the Received Path Signal Label: spe2488RPOHPathSignalLabel

This function retrieves the captured and accepted path signal label (C2) and sets the expected path signal label on the receive link.

Prototype

```
INT4 spe2488RPOHPathSignalLabel(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT2 rw,
UINT2 type, UINT1* pPSL)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`rw`	: read/write flag, write if zero
`type`	: type of access 1 = rx accepted path signal label 2 = rx captured path signal label 3 = rx expected path signal label
`pPSL`	: (pointer to) path signal label

Outputs

`pPSL`	: updated path signal label

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Retrieving and Setting the TU3 Path Signal Label: spe2488PathSignalLabelTU3

This function retrieves the captured and accepted path signal label (C2-TU3) and sets the expected TU3 path signal label on the receive link.

Prototype

```
INT4 spe2488PathSignalLabelTU3(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT2 rw,
UINT2 type, UINT1* pPSL)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`rw`	: read/write flag, write if zero
`type`	: type of access 1 = rx accepted path signal label 2 = rx captured path signal label 3 = rx expected path signal label
`pPSL`	: (pointer to) path signal label

Outputs

`pPSL`	: updated path signal label

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing Loss-of-Pointer: spe2488RPOHDiagLOP

This function forces the downstream pointer processing to enter the Loss of Pointer (LOP) state. It does so by inverting the new data flag (NDF) field of the payload pointer that is inserted in the DROP bus.

Prototype

```
INT4 spe2488RPOHDiagLOP(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT2 au3, UINT2 enable)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`enable`	: flag to start/stop NDF inversion

Outputs None

Returns

Success = `SPE2488_SUCCESS`

Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing Pointer Justification: spe2488RPOHDiagPJ

This function diagnoses the downstream pointer processing elements so that there are correct reactions to pointer justification events. In addition, the function either forces positive or negative justification events or disables the generation of any pointer justification element.

Prototype
```
INT4 spe2488RPOHDiagPJ(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT2 au3, UINT2 enable, INT1 type)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`enable`	: flag to start/stop diagnostic
`type`	: type of pointer justification event: 0 = no pointer justification –1 = negative pointer justification +1 = positive pointer justification

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Retrieving the Received Pointer Value: spe2488RPOHReadPtr

Retrieves the pointer value received in the H1 and H2 bytes of the received stream.

Prototype
```
INT4 spe2488RPOHReadPtr(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT2 *pcptr)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`pcptr`	: (pointer to) allocated memory

Outputs `pcptr` : current pointer value received in H1-H2

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Retrieving the Received TU3 Pointer Value: spe2488RPOHReadPtrTU3

Retrieves the pointer value received in the H1-TU3 and H2-TU3 bytes of the received stream. This function can only be called if this slice was configured as a TUG3.

Prototype
```
INT4 spe2488RPOHReadPtrTU3(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT2 *pcptr)
```

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`stm4` : STM-4 index
`au3` : AU-3 index
`pcptr` : (pointer to) allocated memory

Outputs `pcptr` : current pointer value received in H1-H2

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Transmit Path Overhead (TPOH)

Retrieving and Setting the Transmit Path Signal Label: spe2488TPOHPathSignalLabel

This function retrieves the captured and accepted path signal label (C2) and sets the expected path signal label on the transmit link.

Prototype

```
INT4 spe2488TPOHPathSignalLabel(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT2 rw,
UINT2 type, UINT1* pPSL)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`rw`	: read/write flag, write if zero
`type`	: type of access 1 = rx accepted path signal label 2 = rx captured path signal label 3 = rx expected path signal label
`pPSL`	: (pointer to) path signal label

Outputs

`pPSL`	: updated path signal label

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States

`SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects

None

Forcing Path AIS: spe2488TPOHInsertPAIS

This function enables the insertion of the path alarm indication signal (PAIS) in the transmit stream. The synchronous payload envelope and the pointer bytes (H1 – H3) are set to all ones.

Prototype

```
INT4 spe2488TPOHInsertPAIS(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT2 enable)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`enable`	: flag to start/stop PAIS insertion

Outputs

None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing Path BIP-8 Errors: spe2488TPOHDiagB3

This function enables the inversion of the path BIP-8 byte (B3) in the transmit stream. Only those bits that are set in the passed mask are inverted.

Prototype
```
INT4 spe2488TPOHDiagB3(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT2 au3, UINT1 maskB3, UINT2 enable)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`maskB3`	: mask identifying the bits to toggle in B3
`enable`	: flag to start/stop B3 inversion

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing TU3 Path BIP-8 Errors: spe2488TPOHDiagB3TU3

This function enables the inversion of the TU3 path BIP-8 byte (B3) in the transmit stream. Only those bits that are set in the passed mask are inverted. This function can only be called if this slice was configured as a TUG3.

Prototype
```
INT4 spe2488TPOHDiagB3TU3(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT2 au3, UINT1 maskB3, UINT2 enable)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`maskB3`	: mask identifying the bits to toggle in B3
`enable`	: flag to start/stop B3 inversion

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the Path Remote Error Indication Count: spe2488TPOHInsertPREI

This function inserts the path remote error indication count passed in argument inside the path status byte. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype
```
INT4 spe2488TPOHInsertPREI(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT1 PREI)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`prei`	: PREI value

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the TU3 Path Remote Error Indication Count: spe2488TPOHInsertPREITU3

This function inserts the TU3 path remote error indication count passed in argument inside the path status byte. This function can only be called if this slice was configured as a TUG3. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype
```
INT4 spe2488TPOHInsertPREITU3(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT1 PREI)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`prei`	: PREI value

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Controlling Pointer Justification: spe2488TPOHDiagPJ

This function diagnoses the downstream pointer processing elements for correct reaction to pointer justification events. It can either force positive or negative stuff justification events or disable the generation of any pointer justification element.

Prototype
```
INT4 spe2488TPOHDiagPJ(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT2 au3, UINT2 enable, INT1 type)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`enable`	: flag to start/stop diagnostic
`type`	: type of pointer justification event: 0 for no pointer justification -1 for negative +1 for positive

Outputs None

Returns Success = `SPE2488_SUCCESS`
Fail

Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Writing the J1 Byte: spe2488TPOHWriteJ1

This function writes the J1 byte into the Transmit Path Overhead. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype
```
INT4 spe2488TPOHWriteJ1(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT1 J1)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
au3	: AU-3 index
J1	: J1 byte to write

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Writing the J1-TU3 Byte: spe2488TPOHWriteJ1TU3

This function writes the J1-TU3 byte into the transmit TU3 Path Overhead. This function can only be called if this slice was configured as a TUG3. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype
```
INT4 spe2488TPOHWriteJ1TU3(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT1 J1)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
au3	: AU-3 index
J1	: TU3 byte to write

Outputs None

Returns Success = SPE2488_SUCCESS

Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Writing the C2 Byte: spe2488TPOHWriteC2

This function writes the C2 byte into the Transmit Path Overhead. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype
```
INT4 spe2488TPOHWriteC2(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT2 au3, UINT1 C2)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
au3	: AU-3 index
C2	: C2 byte to write

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Writing the C2-TU3 Byte: spe2488TPOHWriteC2TU3

This function writes the C2-TU3 byte into the transmit TU3 Path Overhead. This function can only be called if this slice was configured as a TUG3. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype
```
INT4 spe2488TPOHWriteC2TU3(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT1 C2)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
au3	: AU-3 index
C2	: C2 byte to write

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV

Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Writing the F2 Byte: spe2488TPOHWriteF2

This function writes the F2 byte into the Transmit Path Overhead. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype
```
INT4 spe2488TPOHWriteF2(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT2 au3, UINT1 F2)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
au3	: AU-3 index
F2	: F2 byte to write

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Writing the F2-TU3 Byte: spe2488TPOHWriteF2TU3

This function writes the F2-TU3 byte into the transmit TU3 Path Overhead. This function can only be called if this slice was configured as a TUG3. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype `INT4 spe2488TPOHWriteF2TU3(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT2 au3, UINT1 F2)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`F2`	: F2 byte to write

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the Z3 Byte: spe2488TPOHWriteZ3

This function writes the Z3 byte into the Transmit Path Overhead. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype `INT4 spe2488TPOHWriteZ3(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT2 au3, UINT1 Z3)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`Z3`	: Z3 byte to write

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the Z3-TU3 Byte: spe2488TPOHWriteZ3TU3

This function writes the Z3-TU3 byte into the transmit TU3 Path Overhead. This function can only be called if this slice was configured as a TUG3. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype
```
INT4 spe2488TPOHWriteZ3TU3(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT1 Z3)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`Z3`	: Z3 byte to write

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the Z4 Byte: spe2488TPOHWriteZ4

This function writes the Z4 byte into the Transmit Path Overhead. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype
```
INT4 spe2488TPOHWriteZ4(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT2 au3, UINT1 Z4)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`Z4`	: Z4 byte to write

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the Z4-TU3 Byte: spe2488TPOHWriteZ4TU3

This function writes the Z4-TU3 byte into the transmit TU3 Path Overhead. This function can only be called if this slice was configured as a TUG3. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype
```
INT4 spe2488TPOHWriteZ4TU3(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT1 Z4)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`Z4`	: Z4 byte to write

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the Z5 Byte: spe2488TPOHWriteZ5

This function writes the Z5 byte into the Transmit Path Overhead. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype
```
INT4 spe2488TPOHWriteZ5(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT2 au3, UINT1 Z5)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`Z5`	: Z5 byte to write

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Writing the Z5-TU3 Byte: spe2488TPOHWriteZ5TU3

This function writes the Z5-TU3 byte into the transmit TU3 Path Overhead. This function can only be called if this slice was configured as a TUG3. If this (STM-4, AU3) is not configured for path termination, this function will have no effect on the Path Overhead.

Prototype
```
INT4 spe2488TPOHWriteZ5TU3(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT1 Z5)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`Z5`	: Z5 byte to write

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Retrieving Current Transmit Pointer Value: spe2488TPOHReadPtr

This function retrieves the pointer value being inserted in H1 and H2 bytes of the reansmit stream.

Prototype
```
INT4 spe2488TPOHReadPtr(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT2 au3, UINT2 *pcptr)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`pcptr`	: (pointer to) allocated memory

Outputs

`pcptr`	: current pointer value received in H1-H2

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing AU Pointer Value: spe2488TPOHForceTxPtr

This function enables insertion of the AU pointer value passed in argument into the H1 and H2 bytes of the transmit stream. It can only be called if this slice is in TU3 mode.

Prototype

```
INT4 spe2488TPOHForceTxPtr(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT2 enable,
UINT2 aptr)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`enable`	: flag to start/stop generation
`aptr`	: pointer value to insert in H1-H2

Outputs None

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

5.9 DROP/ADD Bus PRBS Generator and Monitor (DPGM / APGM)

DROP Bus PRBS Generator and Monitor (DPGM)

Configuring the PRBS Generator: spe2488DPGMGenCfg

This function fully configures the Drop bus PRBS generator.

Prototype `INT4 spe2488DPGMGenCfg(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT2 au3, UINT2 type, sSPE2488_CFG_PRBS *pcfgPrbs)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`type`	: type of payload 0 = STS-12 / STM-4 1 = STS-24 / STM-8 2 = STS-36 / STM-12 3 = STS-48 / STM-16
`pcfgPrbs`	: PRBS configuration structure

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Configuring the PRBS Monitor: spe2488DPGMMonCfg

This function fully configures the Drop bus PRBS monitor.

Prototype `INT4 spe2488DPGMMonCfg(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT2 au3, UINT2 type, sSPE2488_CFG_PRBS *pcfgPrbs)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`type`	: type of payload 0 = STS-12 / STM-4 1 = STS-24 / STM-8 2 = STS-36 / STM-12 3 = STS-48 / STM-16
`pcfgPrbs`	: PRBS configuration structure

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing Bit Errors: spe2488DPGMGenForceErr

This function forces bit errors in the inserted pseudo random bit sequence (PRBS). Thereafter, the MSB of the PRBS is inverted, inducing a single bit error.

Prototype `INT4 spe2488DPGMGenForceErr(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT2 au3)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Forcing a Resynchronization: spe2488DPGMMonResync

This function forces the resynchronization of the monitor to the incoming pseudo random bit sequence (PRBS). The monitor will go out of synchronization and begin re-synchronizing the incoming PRBS payload. This will automatically force all slaves to resynchronize at the same time.

Prototype
```
INT4 spe2488DPGMMonResync(sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
au3	: AU-3 index

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

ADD Bus PRBS Generator and Monitor (APGM)

Configuring the PRBS Generator: spe2488APGMGenCfg

This function fully configures the Add bus PRBS generator.

Prototype
```
INT4 spe2488APGMGenCfg(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT2 au3, UINT2 type, sSPE2488_CFG_PRBS
*pcfgPrbs)
```

Inputs

deviceHandle	: device Handle (from spe2488Add)
stm4	: STM-4 index
au3	: AU-3 index
type	: type of payload 0 = STS-12 / STM-4 1 = STS-24 / STM-8 2 = STS-36 / STM-12 3 = STS-48 / STM-16

```
pcfgPrbs        : PRBS configuration structure
```

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Configuring the PRBS Monitor: spe2488APGMMonCfg

This function fully configures the Add bus PRBS monitor.

Prototype

```
INT4 spe2488APGMMonCfg(sSPE2488_HNDL deviceHandle,
UINT2 stm4, UINT2 au3, UINT2 type, sSPE2488_CFG_PRBS
*pcfgPrbs)
```

Inputs

```
deviceHandle    : device Handle (from spe2488Add)
stm4            : STM-4 index
au3             : AU-3 index
type            : type of payload
                      0 = STS-12 / STM-4
                      1 = STS-24 / STM-8
                      2 = STS-36 / STM-12
                      3 = STS-48 / STM-16
pcfgPrbs        : PRBS configuration structure
```

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing Bit Errors: spe2488APGMGenForceErr

This function forces bit errors in the inserted pseudo random bit sequence (PRBS). Thereafter, the MSB of the PRBS is inverted, inducing a single bit error.

Prototype `INT4 spe2488APGMGenForceErr(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT2 au3)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Forcing a Resynchronization: spe2488APGMMonResync

This function forces the resynchronization of the monitor to the incoming pseudo random bit sequence (PRBS). The monitor will go out of synchronization and begin re-synchronizing the incoming PRBS payload. This will automatically force all slaves to resynchronize at the same time.

Prototype `INT4 spe2488APGMMonResync(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT2 au3)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

5.10 Interrupt Service Functions

This section describes interrupt-service functions that perform the following tasks:

- Set, get and clear the interrupt enable mask
- Read and process the interrupt-status registers
- Poll and process the interrupt-status registers

See page 31 for an explanation of our interrupt servicing architecture.

Configuring ISR Processing: spe2488ISRConfig

This function allows the user to configure how ISR processing is to be handled: polling (`SPE2488_POLL_MODE`) or interrupt driven (`SPE2488_ISR_MODE`). If polling is selected, the user is responsible for calling periodically `spe2488Poll` to collect exception data form the device.

Prototype
```
INT4 spe2488ISRConfig(sSPE2488_HNDL deviceHandle,
UINT2 mode)
```

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`mode` : mode of operation

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_MODE`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Getting the Interrupt Status Mask: spe2488GetMask

This function returns the contents of the interrupt mask registers of the SPECTRA-2488 device.

Prototype
```
INT4 spe2488GetMask(sSPE2488_HNDL deviceHandle,
sSPE2488_MASK *pmask)
```

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`pmask` : (pointer to) mask structure

Outputs
`pmask` : (pointer to) updated mask structure

Returns
Success = `SPE2488_SUCCESS`

Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects None

Setting the Interrupt Enable Mask: spe2488SetMask

This function sets the contents of the interrupt mask registers of the SPECTRA-2488 device. Any bits that are set in the passed structure are set in the associated SPECTRA-2488 registers.

Prototype
```
INT4 spe2488SetMask(sSPE2488_HNDL deviceHandle,
sSPE2488_MASK *pmask)
```

Inputs
deviceHandle : device Handle (from spe2488Add)
pmask : (pointer to) mask structure

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects May change the operation of the ISR / DPR

Clearing the Interrupt Enable Mask: spe2488ClearMask

This function clears individual interrupt bits and registers in the SPECTRA-2488 device. Any bits that are set in the passed structure are cleared in the associated SPECTRA-2488 registers.

Prototype
```
INT4 spe2488ClearMask(sSPE2488_HNDL deviceHandle,
sSPE2488_MASK *pmask)
```

Inputs
deviceHandle : device Handle (from spe2488Add)
pmask : (pointer to) mask structure

Outputs None

Returns Success = SPE2488_SUCCESS
Failure = SPE2488_ERR_INVALID_DEV
SPE2488_ERR_INVALID_DEVICE_STATE
SPE2488_ERR_INVALID_ARG

Valid States SPE2488_ACTIVE, SPE2488_INACTIVE

Side Effects May change the operation of the ISR / DPR

Polling the Interrupt Status Registers: spe2488Poll

This function commands the driver to poll the interrupt registers in the device. The call will fail unless the device was initialized (via `spe2488Init`) or configured (via `spe2488ISRConfig`) into polling mode.

Prototype `INT4 spe2488Poll(sSPE2488_HNDL deviceHandle)`

Inputs `deviceHandle` : device Handle (from `spe2488Add`)

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_MODE`

Valid States `SPE2488_ACTIVE`

Side Effects None

Interrupt-Service Routine: spe2488ISR

This function reads the state of the interrupt registers in the SPECTRA-2488 and stores them in an ISV. Performs whatever functions are needed to clear the interrupt, from simply clearing bits to complex functions. This routine is called by the application code, from within `sysSpe2488ISRHandler`. If ISR mode is configured all interrupts that were detected are disabled and the ISV is returned to the Application. Note that the Application is then responsible for sending this buffer to the DPR task. If polling mode is selected, no ISV is returned to the Application and the DPR is called directly with the ISV.

Prototype `void * spe2488ISR(sSPE2488_HNDL deviceHandle)`

Inputs `deviceHandle` : device Handle (from `spe2488Add`)

Outputs None

Returns (pointer to) ISV buffer (to send to the DPR) or NULL (pointer)

Valid States `SPE2488_ACTIVE`

Side Effects None

Deferred-Processing Routine: spe2488DPR

This function acts on data contained in the passed ISV, allocates one or more DPV buffers (via `sysSpe2488DPVBufferGet`) and invokes one or more callbacks (if defined and enabled). This routine is called by the application code, within `sysSpe2488DPRTask`. Note that the callbacks are responsible for releasing the passed DPV. It is recommended that it be done as soon as possible to avoid running out of DPV buffers.

Prototype `void spe2488DPR(sSPE2488_ISV *pisv)`

Inputs `pisv` : (pointer to) ISV buffer

Outputs None

Returns None

Valid States `SPE2488_ACTIVE`

Side Effects None

Setting the Thresholds for Callbacks: spe2488SetThresh

This function sets the number of events that must be received (thresholds) before those events are reported via the callbacks.

Prototype
```
INT4 spe2488SetThresh(sSPE2488_HNDL deviceHandle,
sSPE2488_MASK *pthreshold)
```

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`pthreshold` : (pointer to) threshold structure

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Getting the Thresholds for Callbacks: spe2488GetThresh

This function sets the number of events that must be received (thresholds) before those events are reported via the callbacks.

Prototype `INT4 spe2488GetThresh(sSPE2488_HNDL deviceHandle, sSPE2488_MASK *pthreshold)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`pthreshold` : (pointer to) allocated memory

Outputs `pthreshold` : threshold structure

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Getting the Event Counters: spe2488GetThreshCntr

This function returns the number of events that have occurred while the thresholds have not been reached.

Prototype `INT4 spe2488SetThreshCntr(sSPE2488_HNDL deviceHandle, sSPE2488_MASK *pthreshCntr)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`pthreshold` : (pointer to) allocated memory

Outputs `pthreshold` : event counters

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

5.11 Alarm, Status and Statistics Functions

Configuring the Device Statistics: spe2488CfgStats

This function configures the behaviour of the device counts.

Prototype `INT4 spe2488CfgStats(sSPE2488_HNDL deviceHandle, sSPE2488_CFG_CNT cfgCnt)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`cfgCnt` : counters configuration block

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`

Valid States `SPE2488_ACTIVE, SPE2488_INACTIVE`

Side Effects None

Statistics Collection Routine: spe2488GetCnt

This function retrieves all the device counts; it should be called by the application code, in the context of a separate periodic task. It is the user's responsibility to ensure that this function is called often enough to prevent the device counts from saturating.

Prototype `INT4 spe2488GetCnt(sSPE2488_HNDL deviceHandle, sSPE2488_STAT_CNT *pcnt)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`pcnt` : (pointer to) allocated memory

Outputs
`pcnt` : current device counts

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE`

Side Effects When a port is not in use (and the device is in QUAD mode), indirect accesses to this port's registers will time out. These timeouts can substantially increase the execution time of this function. So, the USER should make use of spe2488GetCntStm4 whenever one of the 4 slices is not used, in order to selectively retrieve the per-slice device counts.

Statistics Collection Routine: spe2488GetCntStm4

This function retrieves the per-slot device counts; it should be called by the application code, in the context of a task. It is the USER's responsibility to ensure that this function is called often enough to prevent the device counts from saturating.

Prototype
```
INT4 spe2488GetCntStm4(sSPE2488_HNDL deviceHandle,
UINT2 stm4, sSPE2488_STAT_CNT *pcnt)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`pcnt`	: (pointer to) allocated memory

Outputs

`pcnt`	: current device counts

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE`

Side Effects None

Getting Counter for SOH and LOH Block: spe2488GetCntSOHLOH

This function retrieves the specified device counts block. It is the user's responsibility to poll this function often enough to prevent the device counts from saturating.

Prototype
```
INT4 spe2488GetCntSOHLOH(sSPE2488_HNDL deviceHandle,
UINT2 stm4, sSPE2488_STAT_CNT_SOH *pcntSOH,
sSPE2488_STAT_CNT_LOH *pcntLOH)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`pcntSOH`	: (pointer to) allocated memory
`pcntLOH`	: (pointer to) allocated memory

Outputs

`pcntSOH`	: current device counts
`pcntLOH`	: current device counts

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE`

Side Effects None

Getting Counter for RPOH Block: spe2488GetCntRPOH

This function retrieves the specified device counts block. It is the user's responsibility to poll this function often enough to prevent the device counts from saturating.

Prototype
```
INT4 spe2488GetCntRPOH(sSPE2488_HNDL deviceHandle,
UINT2 stm4, sSPE2488_STAT_CNT_RPOH *pcntRPOH)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`pcntRPOH`	: (pointer to) allocated memory

Outputs

`pcntRPOH`	: current device counts

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE`

Side Effects None

Getting Counter for TPOH Block: spe2488GetCntTPOH

This function retrieves the specified device counts block. It is the user's responsibility to poll this function often enough to prevent the device counts from saturating.

Prototype
```
INT4 spe2488GetCntTPOH(sSPE2488_HNDL deviceHandle,
UINT2 stm4, sSPE2488_STAT_CNT_TPOH *pcntTPOH)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`pcntTPOH`	: (pointer to) allocated memory

Outputs

`pcntTPOH`	: current device counts

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE`

Side Effects None

Getting Current Alarm Status: spe2488GetStatus

This function retrieves the current alarm status by reading all the alarm status registers. It is the user's responsibility to ensure that the passed structure points to an area of memory large enough to hold a copy of the counter structure.

Prototype

```
INT4 spe2488GetStatus(sSPE2488_HNDL deviceHandle,
sSPE2488_STATUS *palm)
```

Inputs

`deviceHandle` : device Handle (from `spe2488Add`)
`palm` : (pointer to) allocated memory

Outputs

`palm` : current alarm status

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States

`SPE2488_ACTIVE`

Side Effects

When a port is not in use (and the device is in QUAD mode), indirect accesses to this port's registers will time out. These timeouts can substantially increase the execution time of this function. So, the USER should make use of spe2488GetStatusStm4 whenever one of the 4 slices is not used, in order to selectively retrieve the per-slice device status.

Getting Current Alarm Status: spe2488GetStatusStm4

This function retrieves a per-port snapshot of the current status form the status registers of the SPECTRA-2488. It is the USER's responsibility to ensure that the passed structure points to an area of memory large enough to hold a copy of the status structure.

Prototype
```
INT4 spe2488GetStatusStm4(sSPE2488_HNDL
deviceHandle, UINT2 stm4, sSPE2488_STATUS *palm)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`palm`	: (pointer to) allocated memory

Outputs

`palm`	: current alarm status

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE`

Side Effects None

Getting Current Alarm Status for IO block: spe2488GetStatusIO

This function returns the current status of the IO block.

Prototype
```
INT4 spe2488GetStatusIO(sSPE2488_HNDL deviceHandle,
sSPE2488_STATUS_IO *palmIO)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`palmIO`	: (pointer to) allocated memory

Outputs

`palmIO`	: current alarm status

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE`

Side Effects None

Getting Current Alarm Status for SOH and LOH block: spe2488GetStatusSOHLOH

This function returns the current status of the SOH and LOH blocks for a given STM-4 slice.

Prototype
```
INT4 spe2488GetStatusSOHLOH(sSPE2488_HNDL
deviceHandle, UINT2 stm4, sSPE2488_STATUS_SOH
*palmSOH, sSPE2488_STATUS_LOH *palmLOH)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`palmSOH`	: (pointer to) allocated memory
`palmLOH`	: (pointer to) allocated memory

Outputs

`palmSOH`	: current alarm status
`palmLOH`	: current alarm status

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE`

Side Effects None

Getting Current Alarm Status for RPOH block: spe2488GetStatusRPOH

This function returns the current status of the RPOH block for a given STM-4 slice.

Prototype
```
INT4 spe2488GetStatusRPOH(sSPE2488_HNDL
deviceHandle, UINT2 stm4, sSPE2488_STATUS_RPOH
*palmRPOH)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`palmRPOH`	: (pointer to) allocated memory

Outputs

`palmRPOH`	: current alarm status

Returns

Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE`

Side Effects None

Getting Current Alarm Status for TPOH block: spe2488GetStatusTPOH

This function returns the current status of the TPOH block for a given STM-4 slice.

Prototype `INT4 spe2488GetStatusTPOH(sSPE2488_HNDL deviceHandle, UINT2 stm4, sSPE2488_STATUS_TPOH *palmTPOH)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)
`stm4` : STM-4 index
`palmTPOH` : (pointer to) allocated memory

Outputs `palmTPOH` : current alarm status

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE`

Side Effects None

5.12 Device Diagnostics

Testing Register Accesses: spe2488DiagTestReg

This function verifies the hardware access to the device registers by writing and reading back values.

Prototype `INT4 spe2488DiagTestReg(sSPE2488_HNDL deviceHandle)`

Inputs
`deviceHandle` : device Handle (from `spe2488Add`)

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_FAILURE`

Valid States `SPE2488_PRESENT`

Side Effects None

Enabling Line-Side Line Loopbacks: spe2488DiagLineSideLineLoop

This function Clears/Sets a Line-Side Line Loopback. It is up to the user to perform any tests on the looped data.

Prototype `INT4 spe2488DiagLineSideLineLoop(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT2 enable)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`enable`	: sets loop if non-zero, else clears loop

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE`

Side Effects Will inhibit the flow of active data

Enabling Line-Side System Loopbacks: spe2488DiagLineSideSysLoop

This function Clears/Sets a Line-Side System Loopback. It is up to the user to perform any tests on the looped data.

Prototype `INT4 spe2488DiagLineSideSysLoop(sSPE2488_HNDL deviceHandle, UINT2 stm4, UINT2 enable)`

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`enable`	: sets loop if non-zero, else clears loop

Outputs None

Returns Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE`

Side Effects Will inhibit the flow of active data

Enabling System-Side Line Loopbacks: spe2488DiagSysSideLineLoop

This function Clears/Sets a System-Side Line Loopback. It is up to the user to perform any tests on the looped data.

Prototype
```
INT4 spe2488DiagSysSideLineLoop (sSPE2488_HNDL
deviceHandle, UINT2 stm4, UINT2 au3, UINT2 enable)
```

Inputs

`deviceHandle`	: device Handle (from `spe2488Add`)
`stm4`	: STM-4 index
`au3`	: AU-3 index
`enable`	: sets loop if non-zero, else clears loop

Outputs None

Returns
Success = `SPE2488_SUCCESS`
Failure = `SPE2488_ERR_INVALID_DEV`
`SPE2488_ERR_INVALID_DEVICE_STATE`
`SPE2488_ERR_INVALID_ARG`

Valid States `SPE2488_ACTIVE`

Side Effects Will inhibit the flow of active data

5.13 Callback Functions

The SPECTRA-2488 driver has the capability to callback to functions within the user code when certain events occur. These events and their associated callback routine declarations are detailed below. There is no user code action that is required by the driver for these callbacks; this means that the user is free to implement these callbacks in any manner or else to delete them from the driver.

The names given to the callback functions are given as examples only. The addresses of the callback functions invoked by the `spe2488DPR` function are passed during the `spe2488Init` call (inside a DIV). However, the user shall use the exact same prototype. The Application is left responsible for releasing the passed DPV as soon as possible (to avoid running out of DPV buffers) by calling `sysSpe2488DPVBufferRtn` either within the callback function or later inside the Application code.

Notifying the Application of IO Events: cbackSpe2488IO

This callback function is provided by the user and is used by the DPR to report significant IO section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `spe2488Init` call. If the address of the callback function was passed as a NULL at initialization, no callback will be made.

Prototype `void cbackSpe2488IO(sSPE2488_USR_CTXT usrCtxt, sSPE2488_DPV *pdpv)`

Inputs

`usrCtxt`	: user context (from `spe2488Add`)
`pdpv`	: (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `SPE2488_ACTIVE`

Side Effects None

Notifying the Application of SOH Events: cbackSpe2488SOH

This callback function is provided by the user and is used by the DPR to report significant SOH section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `spe2488Init` call. If the address of the callback function was passed as a NULL at initialization, no callback will be made.

Prototype `void cbackSpe2488SOH(sSPE2488_USR_CTXT usrCtxt, sSPE2488_DPV *pdpv)`

Inputs

`usrCtxt`	: user context (from `spe2488Add`)
`pdpv`	: (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `SPE2488_ACTIVE`

Side Effects None

Notifying the Application of LOH Events: cbackSpe2488LOH

This callback function is provided by the user and is used by the DPR to report significant LOH section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `spe2488Init` call. If the address of the callback function was passed as a NULL at initialization, no callback will be made.

Prototype `void cbackSpe2488LOH(sSPE2488_USR_CTXT usrCtxt, sSPE2488_DPV *pdpv)`

Inputs

`usrCtxt`	: user context (from `spe2488Add`)
`pdpv`	: (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `SPE2488_ACTIVE`

Side Effects None

Notifying the Application of RPOH Events: cbackSpe2488RPOH

This callback function is provided by the user and is used by the DPR to report significant RPOH section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `spe2488Init` call. If the address of the callback function was passed as a NULL at initialization, no callback will be made.

Prototype `void cbackSpe2488RPOH(sSPE2488_USR_CTXT usrCtxt, sSPE2488_DPV *pdpv)`

Inputs

`usrCtxt`	: user context (from `spe2488Add`)
`pdpv`	: (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `SPE2488_ACTIVE`

Side Effects None

Notifying the Application of RPOH-TU3 Events: cbackSpe2488RPOHTU3

This callback function is provided by the user and is used by the DPR to report significant RPOH-TU3 section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `spe2488Init` call. If the address of the callback function was passed as a NULL at initialization, no callback will be made.

Prototype `void cbackSpe2488RPOHTU3(sSPE2488_USR_CTXT usrCtxt, sSPE2488_DPV *pdpv)`

Inputs

`usrCtxt`	: user context (from `spe2488Add`)
`pdpv`	: (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `SPE2488_ACTIVE`

Side Effects None

Notifying the Application of TPOH Events: cbackSpe2488TPOH

This callback function is provided by the user and is used by the DPR to report significant TPOH section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `spe2488Init` call. If the address of the callback function was passed as a NULL at initialization, no callback will be made.

Prototype `void cbackSpe2488TPOH(sSPE2488_USR_CTXT usrCtxt, sSPE2488_DPV *pdpv)`

Inputs

`usrCtxt`	: user context (from `spe2488Add`)
`pdpv`	: (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `SPE2488_ACTIVE`

Side Effects None

Notifying the Application of DPGM Events: cbackSpe2488DPGM

This callback function is provided by the user and is used by the DPR to report significant DPGM section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `spe2488Init` call. If the address of the callback function was passed as a NULL at initialization, no callback will be made.

Prototype `void cbackSpe2488DPGM(sSPE2488_USR_CTXT usrCtxt, sSPE2488_DPV *pdpv)`

Inputs

`usrCtxt`	: user context (from `spe2488Add`)
`pdpv`	: (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `SPE2488_ACTIVE`

Side Effects None

Notifying the Application of APGM Events: cbackSpe2488APGM

This callback function is provided by the user and is used by the DPR to report significant APGM section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `spe2488Init` call. If the address of the callback function was passed as a NULL at initialization, no callback will be made.

Prototype `void cbackSpe2488APGM(sSPE2488_USR_CTXT usrCtxt, sSPE2488_DPV *pdpv)`

Inputs

`usrCtxt`	: user context (from `spe2488Add`)
`pdpv`	: (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `SPE2488_ACTIVE`

Side Effects None

6 HARDWARE INTERFACE

The SPECTRA-2488 driver interfaces directly with the user's hardware. In this section, a listing of each point of interface is shown, along with a declaration and any specific porting instructions. It is the responsibility of the user to connect these requirements into the hardware, either by defining a macro or by writing a function for each item listed. Care should be taken when matching parameters and return values.

6.1 Device I/O

Reading from a Device Register: sysSpe2488Read

The most basic hardware connection, sysSpe2488Read, reads the contents of a specific register location. This Macro should be defined by the user to reflect the target system's addressing logic. There is no need for error recovery in this function.

Format `#define sysSpe2488Read(base, offset)`

Prototype `UINT2 sysSpe2488Read(void *base, UINT2 offset)`

Inputs

`base`	: base address of device being accessed
`offset`	: offset to the memory location as it appears in the hardware data-sheet.

Outputs None

Returns value read from the addressed register location

Writing to a Device Register: sysSpe2488Write

The most basic hardware connection, sysSpe2488Write, writes the supplied value to the specific register location. This macro should be defined by the user to reflect the target system's addressing logic. There is no need for error recovery in this function.

Format `#define sysSpe2488Write(base, offset, data)`

Prototype
```
UINT2 sysSpe2488Write(void *base, UINT2 offset, UINT2
data)
```

Inputs

`base`	: base address of device being accessed
`offset`	: offset to the memory location as it appears in the hardware data-sheet.
`data`	: data to be written

Outputs None

Returns Value written to the addressed register location

Polling a Bit: sysSpe2488PollBit

This function simply polls a register masked data until it is zero or times out.

Format `#define sysSpe2488PollBit(base, offset, mask)`

Prototype `UINT2 sysSpe2488PollBit(void *base, UINT2 offset, UINT2 mask)`

Inputs

`base`	: base address of device being accessed
`offset`	: offset to the memory location as it appears in the hardware data-sheet.
`mask`	: mask to apply to byte read

Outputs None

Returns Success = 0
Failure = <any other value>

6.2 System-Specific Interrupt Servicing

The porting of an ISR routine between platforms is a rather difficult task. There are many different implementations of these hardware level routines. In this driver, the user is responsible for installing an interrupt handler (`sysSpe2488ISRHandler`) in the interrupt vector table of the system processor. This handler shall call `spe2488ISR` for each device that has interrupt servicing enabled, to perform the ISR related housekeeping required by each device.

During execution of the API function `spe2488ModuleStart` / `spe2488ModuleStop` the driver informs the application that it is time to install / uninstall this shell via `sysSpe2488ISRHandlerInstall` / `sysSpe2488ISRHandlerRemove`, that needs to be supplied by the user.

Note: A device can be initialized with ISR disabled. In that mode, the user should periodically invoke a provided 'polling' routine (`spe2488Poll`) that in turn calls `spe2488ISR`.

Installing the ISR Handler: sysSpe2488ISRHandlerInstall

This function installs the user-supplied Interrupt-Service Routine (ISR), `sysSpe2488ISRHandler`, into the processor's interrupt vector table. Informs the application that it is time to install the user-supplied function `sysSpe2488DPRTask` into the RTOS as a task.

Format `#define sysSpe2488ISRHandlerInstall()`

Prototype `INT4 sysSpe2488ISRHandlerInstall(void)`

Inputs None

Outputs None

Returns Success = 0
Failure = <any other value>

ISR Handler: sysSpe2488ISRHandler

This routine is invoked when one or more SPECTRA-2488 devices raise the interrupt line to the microprocessor. This routine invokes the driver-provided routine, `spe2488ISR`, for each device registered with the driver.

Format `#define sysSpe2488ISRHandler()`

Prototype `void sysSpe2488ISRHandler(void)`

Inputs None

Outputs None

Returns None

DPR Task: sysSpe2488DPRTask

This routine is installed as a separate task within the RTOS. It runs periodically and retrieves the interrupt status information sent to it by `spe2488ISR` and then invokes `spe2488DPR` for the appropriate device.

Format `#define sysSpe2488DPRTask()`

Prototype `void sysSpe2488DPRTask(void)`

Inputs None

Outputs None

Returns None

Removing the ISR Handler: sysSpe2488ISRHandlerRemove

This function disables interrupt processing for this device. Informs the application that it is time to remove (suspend) the user supplied task, `sysSpe2488DPRTask`. Removes the user-supplied Interrupt-Service Routine (ISR), `sysSpe2488ISRHandler`, from the processor's interrupt vector table.

Format `#define sysSpe2488ISRHandlerRemove()`

Prototype `void sysSpe2488ISRHandlerRemove(void)`

Inputs None

Outputs None

Returns None

7 RTOS INTERFACE

The SPECTRA-2488 driver requires the use of some RTOS resources. In this section, a listing of each required resource is shown, along with a declaration and any specific porting instructions. It is the responsibility of the user to connect these requirements into the RTOS, either by defining a macro or writing a function for each item listed. Care should be taken when matching parameters and return values.

7.1 Memory Allocation/De-Allocation

Allocating Memory: sysSpe2488MemAlloc

This function allocates specified number of bytes of memory.

Format `#define sysSpe2488MemAlloc(numBytes)`

Prototype `UINT1 *sysSpe2488MemAlloc(UINT4 numBytes)`

Inputs `numBytes` : number of bytes to be allocated

Outputs None

Returns Success = Pointer to first byte of allocated memory
Failure = NULL pointer (memory allocation failed)

Freeing Memory: sysSpe2488MemFree

This function frees memory allocated using `sysSpe2488MemAlloc`.

Format `#define sysSpe2488MemFree(pfirstByte)`

Prototype `void sysSpe2488MemFree(UINT1 *pfirstByte)`

Inputs `pfirstByte` : pointer to first byte of the memory region being de-allocated

Outputs None

Returns None

Initializing Memory: sysSpe2488MemSet

This function initializes a block of memory with a given value.

Format `#define sysSpe2488MemSet(pmem, val, sz)`

Prototype `UINT1 *sysSpe2488MemSet(UINT1 *pmem, UINT1 val, UINT4 sz)`

Inputs

`pmem`	: (pointer to) first byte of the memory region to to initialize
`val`	: value to use
`sz`	: size of the block to initialize

Outputs

`pmem`	: updated memory block

Returns None

Copying Memory: sysSpe2488MemCpy

This function copy a block of memory.

Format `#define sysSpe2488MemCpy(pdst, psrc, sz)`

Prototype `UINT1 *sysSpe2488MemCpy(UINT1 *pdst, UINT1 *psrc, UINT2 sz)`

Inputs

`pdst`	: (pointer to) first byte of destination block
`psrc`	: (pointer to) first byte of source block
`sz`	: size of the block to initialize

Outputs

`pdst`	: updated memory block

Returns None

7.2 Buffer Management

All operating systems provide some sort of buffer system, particularly for use in sending and receiving messages. The following calls, provided by the user, allow the driver to Get and Return buffers from the RTOS. It is the user's responsibility to create any special resources or pools to handle buffers of these sizes during the `sysSpe2488BufferStart` call.

Starting Buffer Management: sysSpe2488BufferStart

This function alerts the RTOS that the time has come to make sure that the ISV buffers and DPV buffers are available and sized correctly. This may involve the creation of new buffer pools and it may involve nothing, depending on the RTOS.

Format `#define sysSpe2488BufferStart()`

Prototype `INT4 sysSpe2488BufferStart(void)`

Inputs None

Outputs None

Returns Success = 0
Failure = <any other value>

Getting an ISV Buffer: sysSpe2488ISVBufferGet

This function gets a buffer from the RTOS that will be used by the ISR code to create an Interrupt-Service Vector (ISV) . The ISV consists of data transferred from the devices interrupt status registers.

Format `#define sysSpe2488ISVBufferGet()`

Prototype `sSPE2488_ISV *sysSpe2488ISVBufferGet(void)`

Inputs None

Outputs None

Returns Success = (pointer to) a ISV buffer
Failure = NULL (pointer)

Returning an ISV Buffer: sysSpe2488ISVBufferRtn

This function returns an ISV buffer to the RTOS when the information in the block is no longer needed by the DPR.

Format `#define sysSpe2488ISVBufferRtn(pisv)`

Prototype `void sysSpe2488ISVBufferRtn(sSPE2488_ISV *pisv)`

Inputs `pisv` : (pointer to) a ISV buffer

Outputs None

Returns None

Getting a DPV Buffer: sysSpe2488DPVBufferGet

This function gets a buffer from the RTOS that will be used by the DPR code to create a Deferred-Processing Vector (DPV) . The DPV consists of information about the state of the device that is to be passed to the user via a callback function.

Format `#define sysSpe2488DPVBufferGet()`

Prototype `sSPE2488_DPV *sysSpe2488DPVBufferGet(void)`

Inputs None

Outputs None

Returns Success = (pointer to) a DPV buffer
Failure = NULL (pointer)

Returning a DPV Buffer: sysSpe2488DPVBufferRtn

This function returns a DPV buffer to the RTOS when the information in the block is no longer needed by the DPR.

Format `#define sysSpe2488DPVBufferRtn(pdpv)`

Prototype `void sysSpe2488DPVBufferRtn(sSPE2488_DPV *pdpv)`

Inputs `pdpv` : (pointer to) a DPV buffer

Outputs None

Returns None

Stopping Buffer Management: sysSpe2488BufferStop

This function alerts the RTOS that the driver no longer needs any of the ISV buffers or DPV buffers, and that if any special resources were created to handle these buffers, they can be deleted now.

Format `#define sysSpe2488BufferStop()`

Prototype `void sysSpe2488BufferStop(void)`

Inputs None

Outputs None

Returns None

7.3 Timers

Sleeping a Task: sysSpe2488TimerSleep

This function suspends execution of a driver task for a specified number of milliseconds.

Format `#define sysSpe2488TimerSleep(msec)`

Prototype `void sysSpe2488TimerSleep(UINT4 msec)`

Inputs `msec` : sleep time in milliseconds

Outputs None

Returns Success = 0
Failure = <any other value>

7.4 Preemption

Disabling Preemption: sysSpe2488PreemptDisable

This routine prevents the calling task from being preempted. If the driver is in interrupt mode, this routine locks out all interrupts, as well as other tasks in the system. If the driver is in polling mode, this routine locks out other tasks only.

Format `#define sysSpe2488PreemptDisable()`

Prototype `INT4 sysSpe2488PreemptDisable(void)`

Inputs None

Outputs None

Returns Preemption key (passed back as an argument in `sysSpe2488PreemptEnable`)

Re-Enabling Preemption: sysSpe2488PreemptEnable

This routine allows the calling task to be preempted. If the driver is in interrupt mode, this routine unlocks all interrupts and other tasks in the system. If the driver is in polling mode, this routine unlocks other tasks only.

Format `#define sysSpe2488PreemptEnable(key)`

Prototype `void sysSpe2488PreemptEnable(INT4 key)`

Inputs `key` : preemption key (returned by `sysSpe2488PreemptDisable`)

Outputs None

Returns None

8 PORTING THE SPECTRA-2488 DRIVER

This section outlines how to port the SPECTRA-2488 device driver to your hardware and RTOS platform. However, this manual can offer only guidelines for porting the SPECTRA-2488 driver as each platform and application is unique.

8.1 Driver Source Files

The C source files listed in the following table contain the code for the SPECTRA-2488 driver. You may need to modify the code or develop additional code. The code is in the form of constants, macros, and functions. For the ease of porting, the code is grouped into source files (`src`) and header files (`inc`). The source files contain the functions and the header files contain the structures, constants and macros.

Directory	File	Description
`src`	`spe2488_api1.c`	All the API functions that take care of module, device and profile management
	`spe2488_api2.c`	All the SPECTRA-2488 specific API functions.
	`spe2488_hw.c`	Hardware interface functions
	`spe2488_isr.c`	Internal functions for interrupt servicing
	`spe2488_prof.c`	Internal functions for profiles
	`spe2488_rtos.c`	RTOS interface functions
	`spe2488_stat.c`	Internal functions for statistics
	`spe2488_util.c`	All the remaining internal functions
`inc`	`spe2488_api.h`	All API headers
	`spe2488_defs.h`	Driver macros, constants and definitions (such as register mapping and bit masks)
	`spe2488_err.h`	SPECTRA-2488 error codes
	`spe2488_fns.h`	Prototype of non-API functions
	`spe2488_hw.h`	HW interface macros and prototype
	`spe2488_rtos.h`	RTOS interface macros and prototypes

Directory	File	Description
	spe2488_strs.h	driver structures
	spe2488_typs.h	types definitions
example	spe2488_app.c	Example callback functions and example code
	spe2488_app.h	Prototype and definitions for the example code
	spe2488_debug.c	Example debug code for reporting register accesses to the device
	spe2488_debug.h	Prototype and definitions for the debug code

8.2 Driver Porting Procedures

The following procedures summarize how to port the SPECTRA-2488 driver to your platform. The subsequent sections describe these procedures in more detail.

To port the SPECTRA-2488 driver to your platform:

Step 1: Port the driver's RTOS extensions (page 146):

Step 2: Port the driver to your hardware platform (page 148):

Step 3: Port the driver's application-specific elements (page 149):

Step 4: Build the driver (page 150).

Step 1: Porting the RTOS Extensions

The RTOS extensions encapsulate all RTOS specific services and data types used by the driver. These RTOS extensions include:

- Memory Management
- Task management
- Message queues, semaphores and timers

The compiler-specific data type definitions are located in spe2488_typs.h. The files spe2488_rtos.h and spe2488_rtos.c contain macros and functions for RTOS specific services.

To port the driver's RTOS extensions:

1. Modify the data types in `spe2488_typs.h`. The number after the type identifies the data-type size. For example, UINT4 defines a 4-byte (32-bit) unsigned integer. Substitute the compiler types that yield the desired types as defined in this file.

2. Modify the RTOS specific macros in `spe2488_rtos.h`:

Service Type	Macro Name	Description
Memory	`sysSpe2488MemAlloc`	Allocates a memory block
	`sysSpe2488MemFree`	Frees a memory block
	`sysSpe2488MemCpy`	Copies the contents of one memory block to another
	`sysSpe2488MemSet`	Fills a memory block with a specified value

3. Modify the RTOS specific functions in `spe2488_rtos.c`:

Service Type	Function Name	Description
Timer	`sysSpe2488TimerSleep`	Delays the task execution for a given number of milliseconds
Pre-emption Lock/Unlock	`sysSpe2488PreemptDisable`	Disables pre-emption of the currently executing task by any other task or interrupt
	`sysSpe2488PreemptEnable`	Re-enables pre-emption of a task by other tasks and/or interrupts
Buffer	`sysSpe2488BufferStart`	Starts buffer management
	`sysSpe2488BufferStop`	Stops buffer management
	`sysSpe2488ISVBufferGet`	Gets an ISV buffer from the ISV buffer queue
	`sysSpe2488ISVBufferRtn`	Returns an ISV buffer to the ISV buffer queue
	`sysSpe2488DPVBufferGet`	Gets a DPV buffer from the DPV buffer queue
	`sysSpe2488DPVBufferRtn`	Returns a DPV buffer to the DPV buffer queue

Step 2: Porting the Driver to Your Hardware Platform

This section describes how to modify the SPECTRA-2488 driver for your hardware platform.

To port the driver to your hardware platform:

1. Modify the hardware specific macros in `spe2488_hw.h`:

Service Type	Function Name	Description
Register Access	`sysSpe2488Read`	Reads a device register given its real address in memory
	`sysSpe2488Write`	Writes to a device register given its real address in memory

2. Modify the hardware specific functions in `spe2488_hw.c`:

Service Type	Function Name	Description
Register Access	`sysSpe2488PollBit`	Polls a device register given its real address in memory
Interrupt	`sysSpe2488ISRHandlerInstall`	Installs the interrupt handler in the interrupt vector table and spawn the DPR task
	`sysSpe2488ISRHandlerRemove`	Removes the interrupt handler from the vector table and suspend the DPR task
	`sysSpe2488ISRHandler`	Interrupt handler for the SPECTRA-2488 device
	`sysSpe2488DPRTask`	Task that calls the SPECTRA-2488 DPR

Step 3: Porting the Application-Specific Elements

Porting the application-specific elements includes coding the application callback and defining all the constants used by the API.

To port the driver's application-specific elements:

1. Modify the base value of `SPE2488_ERR_BASE` (default = -500) in `spe2488_err.h`.

2. Modify the application specific constants in `spe2488_defs.h`:

Task Constant	Description	Default
`SPE2488_MAX_DEVS`	The maximum number of SPECTRA-2488 devices that can be supported by the driver	5
`SPE2488_MAX_INIT_PROFS`	The maximum number of initialization profiles that can be added to the driver	100

3. Define the following application specific constants for your RTOS-specific services in `spe2488_rtos.h`:

Task Constant	Description	Default
`SPE2488_MAX_ISV_BUF`	The queue message depth of the queue used for passing interrupt context between the ISR handler and the DPR task	50
`SPE2488_MAX_DPV_BUF`	The queue message depth of the queue used for pass interrupt context between the DPR task and the callback functions	950

4. Define the following constants for your OS-specific services in `spe2488_hw.h`:

Task Constant	Description	Default
`SPE2488_DPR_TASK_PRIORITY`	Deferred Task (DPR) task priority	85
`SPE2488_DPR_TASK_STACK_SZ`	DPR task stack size, in bytes	8192
`SPE2488_MAX_DELAY`	Delay between two consecutive polls of a busy bit	100us

Task Constant	Description	Default
SPE2488_MAX_POLL	Maximum number of times a busy bit will be polled before the operation times out	100
SPE2488_MAX_MSGS	The maximum number of messages in the message queue.	1000

4. Code the callback functions according to your application. There are sample callback functions in `spe2488_app.c`. You can use these callback functions or you can customize them before using the driver. The driver will call these callback functions when an event occurs on the device. These functions must conform to the following prototype (`cback` should be replaced with your callback function name):

```
void cback(sSPE2488_USR_CTXT usrCtxt, sSPE2488_DPV *pdpv)
```

Step 4: Building the Driver

This section describes how to build the SPECTRA-2488 driver.

To build the driver:

1. Modify the `Makefile` to reflect the absolute path of your code, your compiler and compiler options.

2. Choose from among the different compile options supported by the driver as per your requirements.

3. Compile the source files and build the SPECTRA-2488 API driver library using your make utility.

4. Link the SPECTRA-2488 API driver library to your application code.

APPENDIX A: DRIVER RETURN CODES

Table 29 describes the driver's return codes.

Table 29: Return Codes

Return Type	Description
SPE2488_ERR_MEM_ALLOC	Memory allocation failure
SPE2488_ERR_INVALID_ARG	Invalid argument
SPE2488_ERR_INVALID_MODULE_STATE	Invalid module state
SPE2488_ERR_INVALID_MIV	Invalid Module Initialization Vector (MIV)
SPE2488_ERR_PROFILES_FULL	Maximum number of profiles already added
SPE2488_ERR_INVALID_PROFILE	Invalid profile
SPE2488_ERR_INVALID_PROFILE_NUM	Invalid profile number
SPE2488_ERR_INVALID_DEVICE_STATE	Invalid device state
SPE2488_ERR_DEVS_FULL	Maximum number of devices already added
SPE2488_ERR_DEV_ALREADY_ADDED	Device already added
SPE2488_ERR_INVALID_DEV	Invalid device handle
SPE2488_ERR_INVALID_DIV	Invalid Device Initialization Vector (DIV)
SPE2488_ERR_INT_INSTALL	Error while installing interrupts
SPE2488_ERR_INVALID_MODE	Invalid ISR/polling mode
SPE2488_ERR_INVALID_REG	Invalid register number
SPE2488_ERR_POLL_TIMEOUT	Time-out while polling

APPENDIX B: SPECTRA-2488 EVENTS

This section of the manual describes the events used in the SPECTRA-2488 device driver.

Table 30: SPECTRA-2488 Events for IO callbacks

Event Name	Description
SPE2488_EVENT_IO_APARRER	Add parity error

Table 31: SPECTRA-2488 Events for SOH callbacks

Event Name	Description
SPE2488_EVENT_SOH_BIPE	Section BIP error
SPE2488_EVENT_SOH_LOS	Loss of signal
SPE2488_EVENT_SOH_LOF	Loss of frame
SPE2488_EVENT_SOH_OOF	Out of frame
SPE2488_EVENT_SOH_TIU	Trace identifier unstable
SPE2488_EVENT_SOH_TIM	Trace identifier mismatch

Table 32: SPECTRA-2488 Events for LOH callbacks

Event Name	Description
SPE2488_EVENT_LOH_SF	Signal failure
SPE2488_EVENT_LOH_SD	Signal degrade
SPE2488_EVENT_LOH_REI	Line remote error indication
SPE2488_EVENT_LOH_BIPE	Line BIP error
SPE2488_EVENT_LOH_COSSM	Change of SSM message
SPE2488_EVENT_LOH_COAPS	Change of APS bytes
SPE2488_EVENT_LOH_APSBF	APS byte failure
SPE2488_EVENT_LOH_RDI	Line remote defect indication

Event Name	Description
SPE2488_EVENT_LOH_AIS	Line alarm indication signal

Table 33: SPECTRA-2488 Events for RPOH, RPOH-TU3 and TPOH callbacks

Event Name	Description
SPE2488_EVENT_POH_TIU	Path trace identifier unstable
SPE2488_EVENT_POH_TIM	Path trace identifier mismatch
SPE2488_EVENT_POH_PJE	Positive justification event
SPE2488_EVENT_POH_NJE	Negative justification event
SPE2488_EVENT_POH_PREI	Path remote error indication
SPE2488_EVENT_POH_BIPE	Path BIP error
SPE2488_EVENT_POH_COPERDI	Change of path enhanced remote defect indication
SPE2488_EVENT_POH_PERDI	Path enhanced remote defect indication
SPE2488_EVENT_POH_PRDI	Path remote defect indication
SPE2488_EVENT_POH_PPDI	Path payload defect indication
SPE2488_EVENT_POH_UNEQ	Path unequipped
SPE2488_EVENT_POH_PSLM	Path signal label mismatch
SPE2488_EVENT_POH_PSLU	Path signal label unstable
SPE2488_EVENT_POH_COPSL	Change of path signal label
SPE2488_EVENT_POH_PPJ	Positive pointer justification
SPE2488_EVENT_POH_NPJ	Negative pointer justification
SPE2488_EVENT_POH_FOVR	FIFO overflow
SPE2488_EVENT_POH_FUDR	FIFO underflow
SPE2488_EVENT_POH_LOP	Loss of pointer
SPE2488_EVENT_POH_AIS	alarm indication signal

Event Name	Description
SPE2488_EVENT_POH_PAISC	Path alarm indication signal concatenated
SPE2488_EVENT_POH_PLOPC	Path alarm indication signal concatenated
SPE2488_EVENT_POH_PAIS	Path alarm indication signal
SPE2488_EVENT_POH_PLOP	Path loss of pointer

Table 34: SPECTRA-2488 Events for DPGM and APGM callbacks

Event Name	Description
SPE2488_EVENT_PGM_MON_ERR	Monitor byte error
SPE2488_EVENT_PGM_MON_E1B1	Monitor change of B1/E1
SPE2488_EVENT_PGM_MON_SYNC	Monitor change of synchronization state

APPENDIX C: CODING CONVENTIONS

This section describes the coding conventions used in the implementation of all PMC driver software.

Variable Type Definitions

Table 35: Variable Type Definitions

Type	Description
UINT1	unsigned integer – 1 byte
UINT2	unsigned integer – 2 bytes
UINT4	unsigned integer – 4 bytes
INT1	signed integer – 1 byte
INT2	signed integer – 2 bytes
INT4	signed integer – 4 bytes

Naming Conventions

Table 36 presents a summary of the naming conventions followed by all PMC driver software. A detailed description is then given in the following sub-sections.

The names used in the drivers are verbose enough to make their purpose fairly clear. This makes the code more readable. Generally, the device's name or abbreviation appears in the prefix.

Table 36: Naming Conventions

Type	Case	Naming convention	Examples
Macros	Uppercase	prefix with "m" and device abbreviation	mSPE2488_WRITE
Constants	Uppercase	prefix with device abbreviation	SPE2488_MAX_DEVS
Structures	Hungarian Notation	prefix with "s" and device abbreviation	sSPE2488_DDB

Type	Case	Naming convention	Examples
API Functions	Hungarian Notation	prefix with device name	`spe2488Add()`
Porting Functions	Hungarian Notation	prefix with "`sys`" and device name	`sysSpe2488Read()`
Other Functions	Hungarian Notation		`myOwnFunction()`
Variables	Hungarian Notation		`maxDevs`
Pointers to variables	Hungarian Notation	prefix variable name with "`p`"	`pmaxDevs`
Global variables	Hungarian Notation	prefix with device name	`spe2488Mdb`

Macros

- Macro names must be all uppercase.
- Words shall be separated by an underscore.
- The letter '`m`' in lowercase is used as a prefix to specify that it is a macro, then the device abbreviation must appear.
- Example: `mSPE2488_WRITE` is a valid name for a macro.

Constants

- Constant names must be all uppercase.
- Words shall be separated by an underscore.
- The device abbreviation must appear as a prefix.
- Example: `SPE2488_REG` is a valid name for a constant.

Structures

- Structure names must be all uppercase.
- Words shall be separated by an underscore.
- The letter '`s`' in lowercase must be used as a prefix to specify that it is a structure, then the device abbreviation must appear.
- Example: `sSPE2488_DDB` is a valid name for a structure.

Functions

API Functions

- Naming of the API functions must follow the hungarian notation.
- The device's full name in all lowercase shall be used as a prefix.
- Example: `spe2488Add()` is a valid name for an API function.

Porting Functions

Porting functions correspond to all functions that are HW and/or RTOS dependent.

- Naming of the porting functions must follow the hungarian notation.
- The `'sys'` prefix shall be used to indicate a porting function.
- The device's name starting with an uppercase must follow the prefix.
- Example: `sysSpe2488Read()` is a hardware / RTOS specific.

Other Functions

- Other Functions are all the remaining functions that are part of the driver and have no special naming convention. However, they must follow the hungarian notation.
- Example: `myOwnFunction()` is a valid name for such a function.

Variables

- Naming of variables must follow the hungarian notation.
- A pointer to a variable shall use `'p'` as a prefix followed by the variable name unchanged. If the variable name already starts with a `'p'`, the first letter of the variable name may be capitalized, but this is not a requirement. Double pointers might be prefixed with `'pp'`, but this is not required.
- Global variables must be identified with the device's name in all lowercase as a prefix.
- Examples: `maxDevs` is a valid name for a variable, `pmaxDevs` is a valid name for a pointer to `maxDevs`, and `spe2488Mdb` is a valid name for a global variable. Note that both `pprevBuf` and `pPrevBuf` are accepted names for a pointer to the `prevBuf` variable, and that both `pmatrix` and `ppmatrix` are accepted names for a double pointer to the variable `matrix`.

File Organization

Table 37 presents a summary of the file naming conventions. All file names must start with the device abbreviation, followed by an underscore and the actual file name. File names should convey their purpose with a minimum amount of characters. If a file size is getting too big one might separate it into two or more files, providing that a number is added at the end of the file name (e.g. `spe2488_api1.c` or `spe2488_api2.c`).

There are 4 different types of files:

- The API file containing all the API functions
- The hardware file containing the hardware dependent functions
- The RTOS file containing the RTOS dependent functions
- The other files containing all the remaining functions of the driver

Table 37: File Naming Conventions

File Type	File Name
API	`spe2488_api1.c, spe2488_api.h`
Hardware Dependent	`spe2488_hw.c, spe2488_hw.h`
RTOS Dependent	`spe2488_rtos.c, spe2488_rtos.h`
Other	`spe2488_init.c, spe2488_init.h`

API Files

- The name of the API files must start with the device abbreviation followed by an underscore and `'api'`. If needed, a number can be added at the end of the name.
- Examples: `spe2488_api1.c` is the only valid name for the file that contains the first part of the API functions, `spe2488_api.h` is the only valid name for the file that contains all of the API functions headers.

Hardware Dependent Files

- The name of the hardware dependent files must start with the device abbreviation followed by an underscore and `'hw'`. If needed, a number might be added at the end of the file name.
- Examples: `spe2488_hw.c` is the only valid name for the file that contains all of the hardware dependent functions, and `spe2488_hw.h` is the only valid name for the file that contains all of the hardware dependent functions headers.

RTOS Dependent Files

- The name of the RTOS dependent files must start with the device abbreviation followed by an underscore and `'rtos'`. If needed, a number might be added at the end of the file name.
- Examples: `spe2488_rtos.c` is the only valid name for the file that contains all of the RTOS dependent functions, and `spe2488_rtos.h` is the only valid name for the file that contains all of the RTOS dependent functions headers.

Other Driver Files

- The names of the remaining driver files must start with the device abbreviation followed by an underscore and the file name itself, which should convey the purpose of the functions within that file with a minimum amount of characters.
- Examples: `spe2488_init.c` is a valid name for a file that would deal with initialization of the device, and `spe2488_init.h` is a valid name for the corresponding header file.

LIST OF TERMS

APPLICATION: Refers to protocol software used in a real system as well as validation software written to validate the SPECTRA-2488 driver on a validation platform.

API (Application Programming Interface): Describes the connection between this module and the user's Application code.

ISR (Interrupt-Service Routine): A common function for intercepting and servicing device events. This function is kept as short as possible because an Interrupt preempts every other function starting the moment it occurs and gives the service function the highest priority while running. Data is collected, Interrupt indicators are cleared and the function ended.

DPR (Deferred-Processing Routine): This function is installed as a task, at a user configurable priority, that serves as the next logical step in Interrupt processing. Data that was collected by the ISR is analyzed and then calls are made into the Application that inform it of the events that caused the ISR in the first place. Because this function is operating at the task level, the user can decide on its importance in the system, relative to other functions.

DEVICE : One SPECTRA-2488 Integrated Circuit. There can be many devices, all served by this one driver module

- DIV (Device Initialization Vector): Structure passed from the API to the device during initialization; it contains parameters that identify the specific modes and arrangements of the physical device being initialized.
- DDB (Device Data Block): Structure that holds the Configuration Data for each device.

MODULE: All of the code that is part of this driver, there is only one instance of this module connected to one or more SPECTRA-2488 chips.

- MIV (Module Initialization Vector): Structure passed from the API to the module during initialization, it contains parameters that identify the specific characteristics of the driver module being initialized.
- MDB (Module Data Block): Structure that holds the Configuration Data for this module.
- RTOS (Real-Time Operating System): The host for this driver

ACRONYMS

APGM: Add bus PRBS Generator and Monitor

API: Application Programming Interface

DDB: Device Data Block

DIV: Device Initialization Vector

DPGM: Drop bus PRBS Generator and Monitor

DPR: Deferred-Processing Routine

DPV: Deferred-Processing (routine) Vector

FIFO: First In, First Out

IO: Input / Output

ISR: Interrupt-Service Routine

ISV: Interrupt-Service (routine) Vector

LOH: Line overhead

MDB: Module Data Block

MIV: Module Initialization Vector

POH: Path Overhead

RPOH: Receive Path Overhead

RTOS: Real-Time Operating System

SOH: Section Overhead

TPOH: Transmit Path Overhead

INDEX