ZLS30390 IEEE 1588-2008 Protocol Engine Software API User's Guide

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip product is strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.

For information regarding Microchip’s Quality Management Systems, please visit www.microchip.com/quality.

Trademarks

ISBN: 978-1-5224-9804-9

Preface

Features

- IEEE 1588-2008 Protocol Engine
  - Version 2.0 operation supported
  - Version 2.1 operation in development
- Profiles
  - Annex J.3 (Delay Request-Response) Default
  - Annex J.4 (Peer-to-Peer) Default
  - ITU-T G.8265.1 Telecom Profile for Frequency
  - ITU-T G.8275.1 Telecom Profile for Phase with Full Timing Support Networks (Ed 1)
  - ITU-T G.8275.1 Telecom Profile for Phase with Full Timing Support Networks (Ed 2)
  - ITU-T G.8275.2 Telecom Profile for Phase with Partial Timing Support Networks
  - CableLabs CM-SP-RDTI Remote DTI Profile
  - IEEE C37.238-2011 Power Profile
  - IEEE C37.238-2017 Power Profile
  - IEC/IEEE 61850-9-3 2016 Power Utility Automation Profile
  - IEC 62439-3 Annex C profile for High-Availability Automation Networks (Ed 3)
  - AES 67 Standard for Audio Applications of Networks – High-Performance Streaming Audio-over-IP interoperability: PTP Profile for Media Applications
  - SMPTE 2059-2 Profile for Use of IEEE-1588 Precision Time Protocol in Professional Broadcast Applications
  - AES R16 Project Report – PTP parameters for AES67 and SMPTE 2059-2 interoperability
  - IEEE 802.1as PTP profile for transport of timing over full-duplex, point-to-point links (alpha)

Software Release Version 5.5.0

ZLS30390

- Device Types
  - Ordinary Clock (Grandmaster, Slave)
  - Boundary Clock
- Path Delay
  - Delay-Request-Response
  - Peer-to-Peer
- Message Types
  - Announce, Signaling, Management
  - Sync, Follow_Up
  - Delay_Req, Delay_Resp
  - Pdelay_Req, Pdelay_Resp, Pdelay_Resp_-Follow_Up
- Transport Mappings
  - Annex D (IPv4)
  - Annex E (IPv6)
  - Annex F (Ethernet)
- Messaging Model
  - Unicast Negotiation
  - Unicast Static (non-negotiated)
  - Multicast
- Supports Transparent Clock correctionField
- Supports default BMCA, profile-specific Alternate BMCA and user-specified BMCA
- Interfaces to Microchip's Time Synchronization Algorithm and Microchip's Network Synchronizer PLLs
- Interfaces to IEEE 1588-capable PHY and switches with integrated timestamping
- Interfaces to variety of Transport Layers
- Independent of OS and CPU, from embedded SoC to home-grown

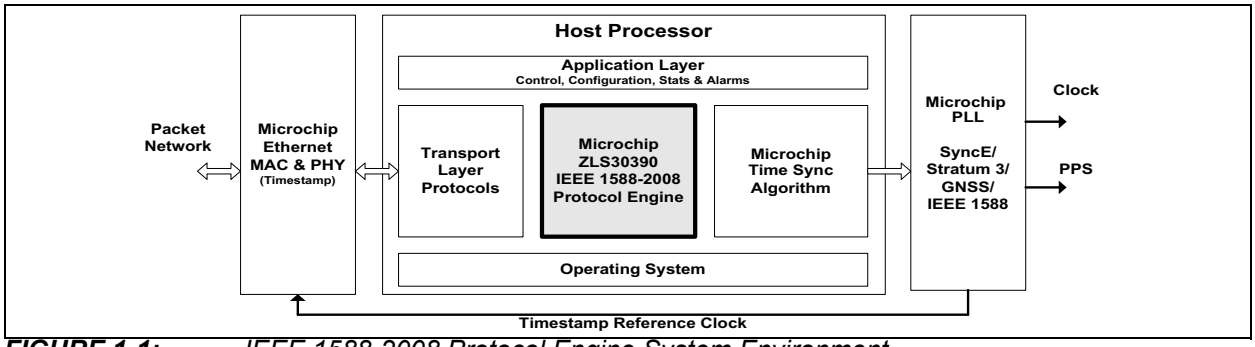

FIGURE 1-1: *IEEE 1588-2008 Protocol Engine System Environment.*

CONVENTIONS USED IN THIS GUIDE

This manual uses the following documentation conventions:

DOCUMENTATION CONVENTIONS

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	*MPLAB® IDE User's Guide*
	Emphasized text	...is the *only* compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	*File>Save*
Bold characters	A dialog button	Click **OK**
	A tab	Click the **Power** tab
N'Rnnnn	A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.	4'b0010, 2'hF1
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier New font:		
Plain Courier New	Sample source code	`#define START`
	Filenames	`autoexec.bat`
	File paths	`c:\mcc18\h`
	Keywords	`_asm, _endasm, static`
	Command-line options	`-Opa+, -Opa-`
	Bit values	`0, 1`
	Constants	`0xFF, 'A'`
Italic Courier New	A variable argument	*`file`*`.o`, where *`file`* can be any valid filename
Square brackets [ ]	Optional arguments	`mcc18 [options]` *`file`* `[options]`
Curly brackets and pipe character: { | }	Choice of mutually exclusive arguments; an OR selection	`errorlevel {0|1}`
Ellipses...	Replaces repeated text	`var_name [, var_name...]`
	Represents code supplied by user	`void main (void) { ... }`

WARNINGS, CAUTIONS, RECOMMENDATIONS, AND NOTES

Warnings, Cautions, Recommendations, and Notes attract attention to essential or critical information in this guide. The types of information included in each are displayed in a style consistent with the examples below.

WARNING
To avoid serious personal injury or death, do not disregard warnings. All warnings use this style. Warnings are installation, operation, or maintenance procedures, practices, or statements, that if not strictly observed, may result in serious personal injury or even death.

CAUTION
To avoid personal injury, do not disregard cautions. All cautions use this style. Cautions are installation, operation, or maintenance procedures, practices, conditions, or statements, that if not strictly observed, may result in damage to, or destruction of, the equipment. Cautions are also used to indicate a long-term health hazard.

Note: All notes use this style. Notes contain installation, operation, or maintenance procedures, practices, conditions, or statements that alert you to important information, which may make your task easier or increase your understanding.

THE MICROCHIP WEBSITE

Microchip provides online support via our website at www.microchip.com. This website is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the website contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the website at:
http://www.microchip.com/support.

DOCUMENT REVISION HISTORY

Revision A (February 2022)

- Initial release of this document as Microchip DS50003222A.

NOTES:

Table of Contents

List of Figures

List of Tables

Chapter 1. Products

For information about Microchip's Time Synchronization Algorithm, refer to:

- ZLS30380 data sheet

For information about Microchip's Network Synchronization PLLs with IEEE 1588 and SyncE capability, refer to:

- ZL3034x data sheet (ZL30342, ZL30343, ZL30347)
- ZL3036x data sheet (ZL30361, ZL30362, ZL30363, ZL30364, ZL30365, ZL30367)
- ZL3070x data sheet (ZL30701, ZL30702, ZL30703, ZL30704)
- ZL3072x data sheet (ZL30721, ZL30722, ZL30723)
- ZL3075x data sheet (ZL30752, ZL30753, ZL30754)
- ZL3077x data sheet (ZL30771, ZL30772, ZL30773)
- ZL3079x data sheet (ZL30791, ZL30793, ZL30795)

1.1 STATUS ON PROFILES IN DEVELOPMENT

IEEE 802.1as-2011 is in alpha release stage and some functionality is not available. Referring to standard section 11.1.3 "Transport of time-synchronization information", we do not transport SYNC from ingress PTP ports to egress PTP port, rather we generate new SYNC messages from local timebase and datasets. As a result, the local PTP clock must be synchronized (unsyntonized operation is not supported).

NOTES:

Chapter 2. Companion Documentation

This guide introduces the IEEE 1588-2008 Protocol Engine features & functionality.

For information about the software architecture, porting, bring-up and individual function calls, refer to:

- ZLS30390 data sheet
- ZLS30390 API Reference Manual (HTML)
- IEEE 1588-2008 Architecture, Porting & Integration Guide

For information about changes made in each software release version, refer to the following text files contained in the ZLS30390 zip file:

- Software Release Note.
- Static Analysis Summary.
- Compiler Warning Summary.
- Software Migration Notes.

For more detailed information about standards compliance, refer to:

- AN4024 (formerly ZLAN-255) IEEE 1588-2008 Edition 2 Conformance Information
- AN4026 (formerly ZLAN-661) IEEE 1588 Industry PTP Profiles Conformance Information

Microchip provides a number of application notes which detail various aspects regarding system configuration, management, integration, and debugging. The following list shows some key publications that may assist users:

- ZLAN-628: Understanding Distributed Structure for IEEE 1588
- ZLAN-525: Handling UTC Leap Second Events in IEEE 1588 Deployments
- ZLAN-473: Timing Card Redundancy - Use Case
- ZLAN-429: Applications of ZLS30390 - Debug Steps

NOTES:

Chapter 3. Software Architecture

3.1 SYSTEM LEVEL SOFTWARE ARCHITECTURE

A high-level view of the complete IEEE 1588-2008 system software architecture is shown in Figure 3-1.

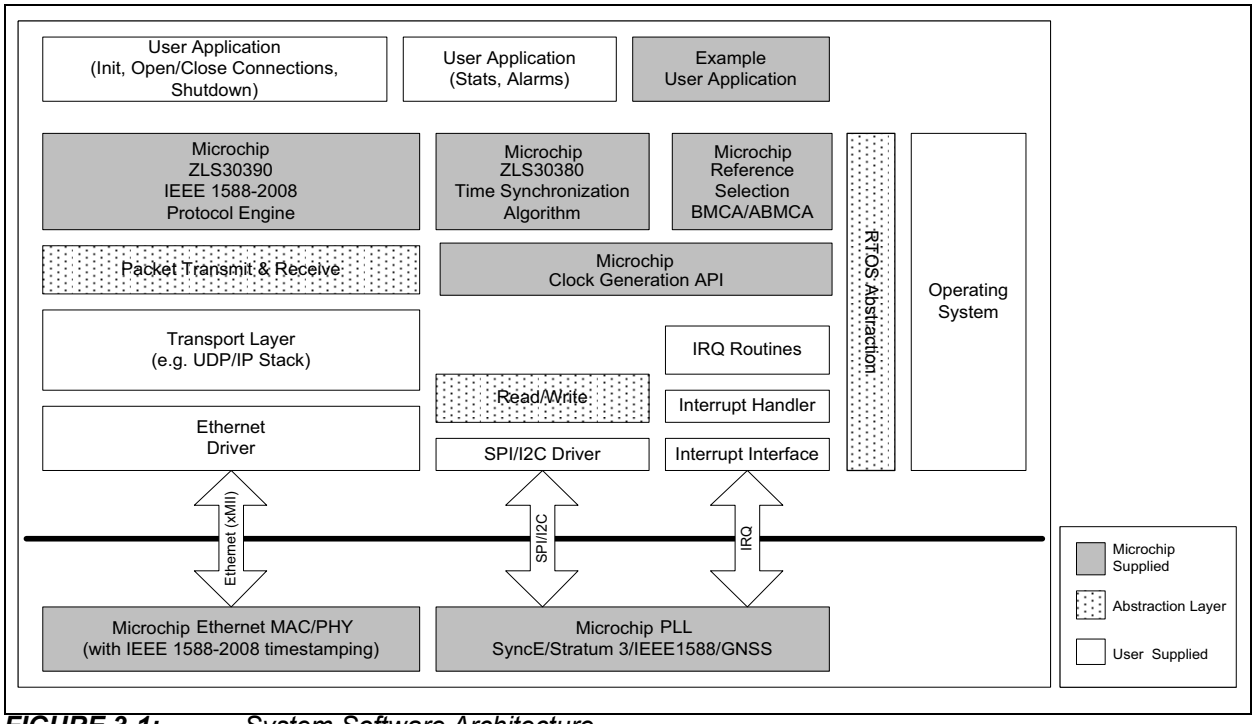

FIGURE 3-1: *System Software Architecture.*

The following modules are noted:

- Example User Application. This module is provided within the API package to provide as an example implementation to the user, to see how to use the API software.
- ZLS30390 IEEE 1588-2008 Protocol Engine. The core of the IEEE 1588-2008 operation, including the various supported profiles.
- ZLS30380 Time Synchronization Algorithm. Responsible to synchronize the local Microchip PLL to the selected GrandMaster.
- Reference Selection BMCA/ABMCA. Responsible to select the local PTP port state(s) and the best synchronization source for the system. Provided as example in the API package, and is expected to be modified by the user.
- Clock Generation API: Conversion of the synchronization commands into Microchip PLL register accesses.

The interaction between the various IEEE 1588-2008 modules within the system is shown in Figure 3-2: System Software Module Interaction.

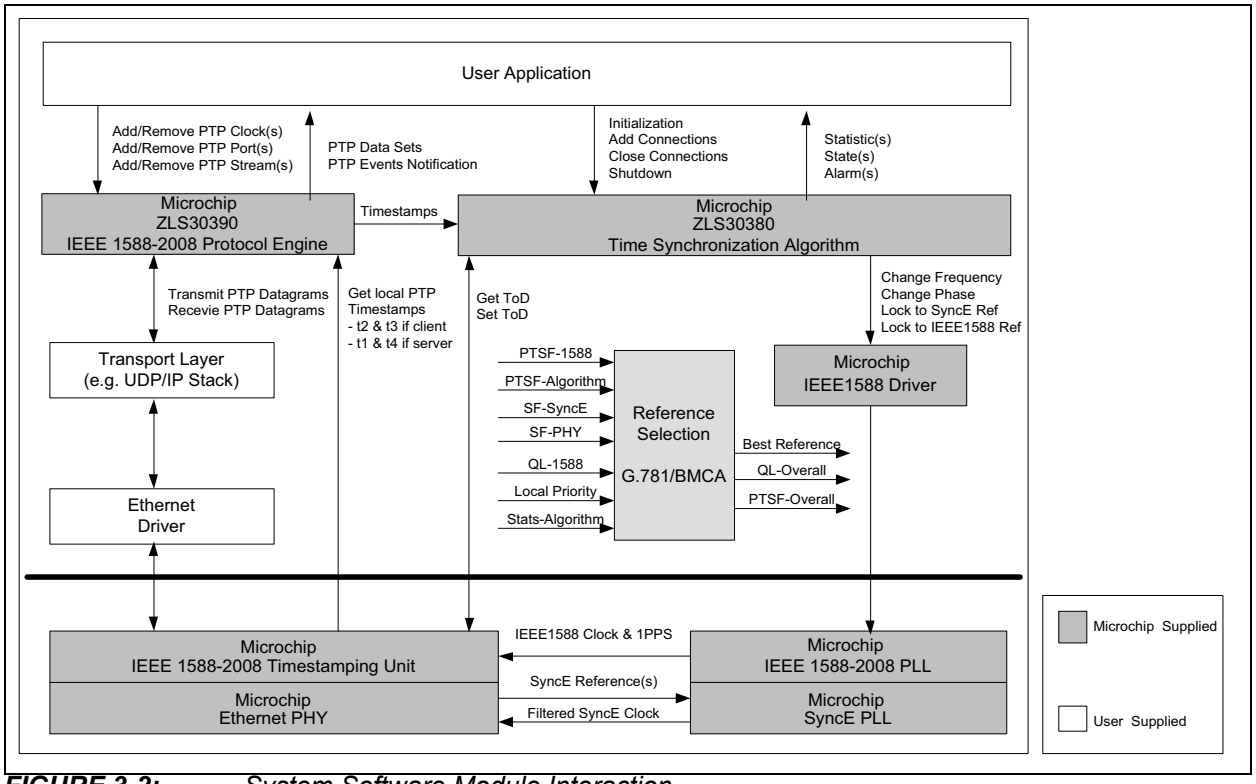

FIGURE 3-2: *System Software Module Interaction.*

3.2 PROTOCOL ENGINE SOFTWARE ARCHITECTURE

3.2.1 PTP Headers

The following PTP header files, found in `zlPtp/include`, contain the user-exposed functions. Other headers are for internal use and should not be used directly.

TABLE 3-1: PTP HEADER FILES

Filename	Description
`zl303xx_PtpApi.h`	Contains all configuration and utility functions required to start and close the PTP application.
`zl303xx_PtpApiTypes.h`	Contains implementation-specific data types required by `zl303xx_PtpApi.h` functions and additional data types passed to user function bindings.
`zl303xx_PtpStdTypes.h`	Contains data types and enums defined by the PTP standard.
`zl303xx_PtpConstants.h`	Contains PTP constants.
`zl303xx_PtpConstants_dep.h`	Contains PTP constants.
`zl303xx_PtpDeprecated.h`	Contains deprecated data types and functions that should not be referenced by a user's application. They are no longer supported and may be deleted in future software versions.

3.2.2 Structure Initialization

Configuration structures must be initialized prior to modification or passing to another function. This is done by calling the relevant `*StructInit()` or `*Get()` function (e.g., use `zl303xx_PtpClockCreateStructInit()` to initialize a `zl303xx_PtpClockCreateS` structure). This ensures that members added to structures in future API versions can be initialized such that backward compatibility is preserved.

3.2.3 Object Handles

The functions used to create a clock, port, and stream return a handle to that object that must be saved by the user application. This handle is a required input for other functions used to manipulate that object.

3.2.4 Time Stamp Format

All time stamp values passed into the PTP application and all values passed out via function bindings will be in PTP time stamp format: 6 bytes of seconds, 4 bytes of nanoseconds. If the time stamping hardware device or timing recovery block does not natively use this format, a conversion function must be called before passing time stamps across these layers. See the `zl303xx_TimeStamp` structure defined in `zlPtp/include/zl303xx_PtpStdTypes.h`.

NOTES:

Chapter 4. General Flow and Application References

4.1 PTP CONCEPTS

The following elements are listed from most general to least and each shares a 1-to-N relationship with the next element. That is, one clock may have many ports associated with it and one port may have many streams.

Understanding of this division is important since the majority of the ZLS30390 API functions relate to a PTP clock, port, or stream.

4.1.1 Node

A PTP node is any device that can send or receive PTP messages across a network. Each node involved in timing synchronization will generally contain a single clock.

4.1.2 Clock

A PTP clock is a collection of data about a hardware device that can measure the passage of time. In many scenarios, the "clock" may actually be the combination of two devices: one that generates a physical clocking signal, and a time stamp counter being driven by this signal. Each clock has its own task/thread (depending on implementation in the OS porting layer).

4.1.3 Port

A PTP port is a collection of data about a logical point of access to the network. There may be multiple PTP ports per PTP clock. Multiple PTP ports can be configured behind a single physical interface. Typical use with G.8275.1 is on PTP port per Ethernet interface.

4.1.4 Stream

A PTP stream is a Microchip-designed abstraction used to represent the flow of PTP messages between two endpoints. For unicast communication, this is the connection from one PTP node to another. For multicast, it is the connection from a PTP node to a multicast group. There may be multiple PTP streams per PTP port. Typical use with G.8275.1 is one PTP stream per PTP port.

4.2 SEQUENCE OF OPERATIONS

The following is a summary of steps to bring-up the target application:

- Open PTP Instantiation
  - Call `zl303xx_PtpInit()` to start the PTP application (or Node)
  - This routine takes no parameters.
- Add one or more PTP Clocks to the PTP Instantiation
  - Call `zl303xx_PtpClockCreateStructInit()` to get default initialization values
  - Modify any of the initialization values as required
  - Call `zl303xx_PtpClockCreate()` to start the PTP Clock

- Add one or more PTP Ports to a PTP Clock
  - Call `zl303xx_PtpPortCreateStructInit()` to get default initialization values
  - Modify any of the initialization values as required
  - Call `zl303xx_PtpPortCreate()` to start the PTP Port
- Add one or more PTP Streams to a PTP Port
  - Call `zl303xx_PtpStreamCreateStructInit()` to get default initialization values
  - Modify any of the initialization values as required
  - Call `zl303xx_PtpStreamCreate()` to start the PTP Stream

4.3 EXAMPLE ROUTINES

The following example routines for starting a PTP application can be found in `zl303xx_ExampleMain.c`:

- `exampleEnvInit()`
- `exampleAppStart()`

In this example code, the following example routines (in `zl303xx_ExamplePtp.c`) are called to complete the steps described above:

- `examplePtpEnvInit()`:Creates the PTP application
- `examplePtpClockCreate()`:Creates and registers a PTP Clock with the application
- `examplePtpPortCreate()`:Creates and registers a PTP Port with a specified PTP Clock
- `examplePtpStreamCreate()`: Creates and registers a PTP Stream with a specified PTP Port

4.4 CLOSING AND SHUTDOWN

Subsequently, the following example routine in `zl303xx_ExampleMain.c` can be referred to when shutting down (or removing) an application:

- `exampleAppStop()`

In this example code, the following example routine (in `zl303xx_ExamplePtp.c`) are called:

- `examplePtpStop()`:Stops the PTP application and removes all PTP Objects:
  - Removes all PTP Streams on each PTP Port
  - Removes all PTP Ports on each PTP Clock
  - Removes all PTP Clocks on the application

4.5 OTHER SOFTWARE INITIALIZATION AND CONFIGURATION

The following system resources and/or dependencies should be made available (where applicable).

> **Note:** The following list represents a general order. Some steps may not be required or will need to be performed in another order specific to the target platform and/or its architecture.

- Initialize the CPU standard IO (for APR Logging)
- Initialize the target processor's SPI or I^2C interface or drivers
- Initialize the target processor's interrupt handlers
- Initialize any required timers (POSIX or OS timers)
- Initialize any necessary network interfaces (LAN, MII, etc.).
- Initialize any other platform specific sub-systems
- Initialize any hardware clock devices. This initialization may include some (or all) of the following tasks:
  - Start any Clock controller software associated with the device and synchronize any internal signals
  - Read the initial settings via the SPI for the Clock Controller
  - Initialize any device interrupt configurations
  - Initialize any necessary clock outputs for PLL-type devices
- Start any Time-of-Day Manager associated with the platform and synchronize it to the hardware or other controls
- Configure any necessary Timestamp Hardware to:
  - Filter or identify timing packets and accurately timestamp them
  - Synchronize the timestamp signals with the local hardware
  - Configure any necessary interrupts
- Configure the local Timing Recovery Algorithm:
  - Initialize the Timing Recovery Algorithm application
  - Configure any objects associated with the Timing Recovery Algorithm (CGU, servers)

NOTES:

Chapter 5. Configuration

5.1 CONFIGURATION OF A PTP APPLICATION INSTANTIATION: ZL303XX_PTPINIT()

5.1.1 PTP Init: Initializing a PTP Application

To initialize the PTP Application, execute the following command:

- `zl303xx_PtpInit()`

This command takes no arguments, but must be run prior to adding any PTP Clocks (or other objects).

> **Note:** A single application instantiation should be created per CPU.

Launching a PTP application will perform the following actions:

- Create PTP tasks (1 or more)
  - PTP Clock Task (one per PTP clock as each clock is created)
    - Driven by a message queue.
    - Executes PTP state machines.
    - Handles received packets and generates packets to transmit.
  - PTP Timer Task (one per PTP node, if enabled):
    - Increments all PTP timers, some of which are used to periodically send packets.
    - Sends the PTP Clock Task a message when a timer expires.
    - Periodic events driven by either:
      - a task delay (VxWorks/ZL303XX_PTP_TIMER_TASK_USE_TICKDELAY) or
      - a POSIX timer (Linux/ZL303XX_PTP_TIMER_USE_HWTIMER). Recommended.
    - Runs approximately every 8 ms (125 Hz).
      - See ZL303XX_PTP_TIMER_TASK_INTVL_MS.
    - Alternatively, the compile-time option "ZL303XX_PTP_TIMER_TASK_DISABLED" may be used to disable the separate Timer Task altogether (i.e. timer task is not started).
      - In this mode, the Protocol Clock Task will directly receive hardware timer ticks for processing via event API `zl303xx_PtpSendHwTimerTick()`.
      - This is a CPU optimization option to reduce the overhead of timer tick processing and reduce mutex contention.
      - Note when using this optimization the "ZL303XX_PTP_TIMER_TASK_USE_TICKDELAY" option is not available and if not using "ZL303XX_PTP_TIMER_USE_HWTIMER" then user must periodically call `zl303xx_PtpSendHwTimerTick()` at the configured hardware timer tick interval (ZL303XX_PTP_TIMER_TASK_INTVL_MS).

5.1.2 PTP Init: Tasks

The following table lists the relative priority of these tasks. The priority numbers are based on:

- VxWorks system where 0 is the highest and 255 is the lowest possible priority
- Linux system where 99 is the highest and 0 is the lowest possible priority

TABLE 5-1: TASKS CREATED BY THE PTP APPLICATION

Task Name	Comments	Task Priority Constant	Stack Size
tPtpClock0zl303xx ... tPtpClockNzl303xx	PTP Clock Task (One or more where N is clock handle)	ZL303XX_PTP_DEFAULT_CLOCK_TASK_PRIORITY: • Linux Value: 96 • VxWorks Value: 64	16384 (16 KB)
tPtpTimerzl303xx	PTP Timer Task (one, if enabled)	ZL303XX_PTP_TIMER_TASK_PRIORITY: • Linux Value: 96 • VxWorks Value: 65	9000

5.2 CONFIGURATION OF A PTP CLOCK: ZL303XX_PTPCLOCKCREATE()

Creating a PTP clock will allocate memory and create the OS task, mutex, and message queue objects by calling the appropriate OS functions.

The following API routines are used to Add a PTP Clock Task (refer to the PTP API Reference Manual for complete interface, data type, and syntax details).

TABLE 5-2: PTP CLOCK CREATION ROUTINES

API Routine	Description
zl303xx_PtpClockCreateStructInit	Data fills the zl303xx_PtpClockCreateS data structure with a set of default clock creation parameters. The user may modify individual members prior to creating the PTP clock.
zl303xx_PtpClockCreate	Creates a new PTP Clock based on the zl303xx_PtpClockCreateS data structure.

Example code for creating a PTP Clock is provided in the module, `zlUserExamples/src/zl303xx_ExamplePtp.c`. This provides examples to:

- Initialize the creation structure with default values in `examplePtpClockCreateStructInit()`
- Create the actual clock in `examplePtpClockCreate()`

Full details about all members of the `zl303xx_PtpClockCreateS` structure can be found in the API reference manual.

The members of the `zl303xx_PtpClockCreateS` are listed in the following tables. These may be modified by a user when creating an application. (Complete details of the `zl303xx_PtpClockCreateS` structure can be found in the `zlPtp/include/zl303xx_PtpApiTypes.h` module).

5.2.1 PTP Clock: Task, General, and Miscellaneous

TABLE 5-3: PTP CLOCK CREATE PARAMETERS (TASK AND MISC)

Sub-Structure	Parameter	Type	Description
—	taskName	String	The task name for this PTP clock. It will be automatically appended with a unique clock ID and part number.
—	taskPriority	Sint32T	Priority of the clock task.
—	taskStackSize	Sint32T	Stack size of the clock task.
—	msgQLength	Sint32T	Length of the message queue used by the clock.
—	eventCallback	zl303xx_PtpEventFnT	Function to be called when a PTP event (see zl303xx_PtpEventE) occurs. This binding is not required (may be NULL).
—	hwCmdFn	zl303xx_PtpHwCmdFnT	Function to be called when PTP needs to query or set some information in the hardware. This binding is required (must NOT be NULL).
—	extData	void *	Pointer to some external user data. Memory management must be handled externally to PTP.
—	clockType	Uint16T	Value that will be placed in the clockType field of a CLOCK_DESCRIPTION management TLV. Set to 0 to have this parameter automatically filled based on the number of ports attached to the clock.
—	requestedHandle	zl303xx_PtpClockHandleT	Request clock to be created with a specific handle. Set to ZL303XX_PTP_INVALID to auto-generate a handle.
—	maxForeignRecords	Uint16T	The maximum number of foreign records that can be handled by this Clock. This should be determined by the number of remote master clocks that are expected to be discovered at any time on all associated Ports. This value must be at least 5 (see IEEE-1588-2008 section 9.3.2.4.5).
bmca	updateTrigger	zl303xx_PtpClockBmcaTriggerE	Determines the event that will trigger an upload to the BMCA engine.
bmca	updateEventSec	Uint32T	BMCA upload interval in seconds
—	autoUpdateStreamStates	zl303xx_Boolean	Flag indicating whether the Clock will automatically update the state of each local stream when a ParentDS update occurs or if the Clock will operate in a manual mode.

5.2.2 PTP Clock: Local defaultDS

TABLE 5-4: PTP CLOCK CREATE PARAMETERS (LOCAL DEFAULTDS)

Sub-Structure	Parameter	Type	Description
defaultDS	twoStepFlag	zl303xx_Boolean	Indicates whether this clock provides timing information using event and subsequent general messages (i.e., SYNC and then a FOLLOW_UP message).
defaultDS	clockIdentity	zl303xx_ClockIdentity	Defines the identity of the local clock. Must be unique for all clocks.
defaultDS	numberPorts	Uint16T	Defines the maximum number of PTP ports on the clock. Set to 0 to automatically fill with the greatest value of zl303xx-_PortDS::portIdentity::portNumber on the clock.
defaultDS	clockQuality	zl303xx_ClockQuality	A dynamic member defining the quality of the local clock. This includes: • clockClass • clockAccuracy • offsetScaledLogVariance Refer to the zl303xx_ClockQuality structure definition for sub-member descriptions.
defaultDS	priority1	Uint8T	Configurable member defining the priority1 attribute of the local clock, used in the Best Master Clock Algorithm. Values: 0 (highest priority) to 255 (lowest priority).
defaultDS	priority2	Uint8T	Configurable member defining the priority2 attribute of the local clock, used in the Best Master Clock Algorithm. Values: 0 (highest priority) to 255 (lowest priority).
defaultDS	domainNumber	Uint8T	Configurable member defining the domain attribute of the local clock. Messages from other domains will be ignored. Values: 0 to 127
defaultDS	slaveOnly	zl303xx_Boolean	Configurable member that defines if the clock can ever enter the MASTER state.
optionalDefaultDS	maxStepsRemoved	Uint16T	Parameter to determine the maximum value of the stepsRemoved member of a received ANNOUNCE message that the local clock will accept.
defaultDS	sdoId	Uint12T	Added for IEEE 1588-2019 v2.1 support and replaces clock transportSpecific configuraton (v2.0). It is used along with the domainNumber for PTP Instance isolation over shared networks (see IEEE 1588-2019 Table 2). The full 12-bits must be specified where: • most significant 4-bit nibble: transportSpecific/majorSdoId (v2.1) • least significant 8-bit byte: reserved/minorSdoId (v2.1) For v2.0 the least significant 8-bit byte should always be 0x00. Default value is 0x000. For 802.1as profile the value should be 0x100 (i.e. transportSpecific 1). Other values may be used by newer profiles defined for PTP v2.1. Note: This member replaces the deprecated zl303xx_PtpClockS transportSpecific configuration.

5.2.3 PTP Clock: Local timePropertiesDS

TABLE 5-5: PTP CLOCK CREATE PARAMETERS (LOCAL TIMEPROPERTIESDS)

Sub-Structure	Parameter	Type	Description
localTimeProperties	currentUtcOffset	Sint16T	The offset between TAI and UTC. This value is only meaningful in PTP systems whose epoch is the PTP epoch. This value is in units of seconds.
localTimeProperties	currentUtcOffsetValid	zl303xx_Boolean	This member is ZL303XX_TRUE if currentUtcOffset is known to be correct.
localTimeProperties	leap59	zl303xx_Boolean	This member is ZL303XX_TRUE if the epoch is the PTP epoch, and the last minute of the current UTC day contains 59 seconds.
localTimeProperties	leap61	zl303xx_Boolean	This member is ZL303XX_TRUE if the epoch is the PTP epoch, and the last minute of the current UTC day contains 61 seconds.
localTimeProperties	timeTraceable	zl303xx_Boolean	This member is ZL303XX_TRUE if the timescale and the value of currentUtcOffset are traceable to a primary reference.
localTimeProperties	frequencyTraceable	zl303xx_Boolean	This member is ZL303XX_TRUE if the frequency determining the timescale is traceable to a primary reference.
localTimeProperties	ptpTimescale	zl303xx_Boolean	This member is ZL303XX_TRUE if the clock timescale of the grandmaster clock is PTP.
localTimeProperties	timeSource	zl303xx_TimeSourceE	The source of time used by the grandmaster clock. Refer to the data type for values.
localTimeProperties	synchronization Uncertain	zl303xx_Boolean	This is an optional member and is ZL303XX_TRUE if the clock is in the process of synchronizing to the time source. If the synchronization is complete or the parameter is not used, this value is ZL303XX_FALSE.

5.2.4 PTP Clock: Profile-Specific and BMCA

TABLE 5-6: PTP CLOCK CREATE PARAMETERS (PROFILE AND BMCA)

Sub-Structure	Parameter	Type	Description
—	profile	zl303xx_PtpProfileE	The PTP profile that will be used on this clock.
—	profileStrict	zl303xx_Boolean	Strictly adhere to parameter ranges specified in the profile.
profileCfg	power	zl303xx_PtpC37p238ClockConfigS	Power Profile-specific clock creation members.
profileCfg	g8275p1	zl303xx_PtpG8275ClockConfigS	Telecom Phase Profile-specific clock creation members.
—	telecom	zl303xx_PtpTelecomClockCreateS	Telecom Profile-specific members.
uncalibrated	usePreviousClass	zl303xx_Boolean	Flag indicating if the clock will advertise the parentDS clockClass or the previously active clockClass while in the UNCALIBRATED state.
—	egressClassUpdateTrigger	zl303xx_PtpClockEgressQlUpdateE	Indicates whether update the Clock's egress class as soon as a server is selected or wait until the algorithm reaches a LOCK state.
pathTrace	enabled	zl303xx_Boolean	Flag indicating whether the Clock has enabled the optional PATH_TRACE functionality.
—	propagateEnabled	zl303xx_Boolean	Flag controlling whether the Clock should forward/propagate PATH_TRACE TLV when main functionality is disabled (pathTrace.enabled FALSE). Should be set TRUE when using PTPv2.1 Edition3 to comply with IEEE 1588-2019 14.2. Default is TRUE.
synchronization Uncertain	enabled	zl303xx_Boolean	Flag indicating whether the Clock has enabled the optional Synchronization-Uncertain functionality
bmca	revertiveEn	zl303xx_Boolean	Applicable to Telecom Profile only: indicates if the G.8265 revertive functionality is enabled.

5.2.4.1 PTP CLOCK: PROFILES ITU-T G.8275.X

For a list of the PTP Clock attributes, refer to the following data structure: `zl303xx_PtpG8275ClockConfigS`.

TABLE 5-7: CLOCK SETTINGS RELATED TO ITU-T G.8275.X

Structure	Parameter	Options/Type	Description
zl303xx_PtpG8275ClockConfigS	bypassEnabled	zl303xx_BooleanE	A configurable member allowing the implementation to bypass the G.8275 restrictions on advertised Announce parameters and allows the user to set the advertised values via the ParentDS.
zl303xx_PtpG8275ClockConfigS	localPriority	Uint8T	Configurable member defining the localPriority attribute of the local clock. Used in the Best Master Clock Algorithm of the G.8275.1 profile when comparing the local clock data set (D0) to that of a received potential grandmasters.
zl303xx_PtpG8275ClockConfigS	classEvalMethod	zl303xx_PtpG8275p1 ClassEvalMethodE	Configurable member defining which method is used to evaluate the clockClass that is advertised from the Clock. Refer to the zl303xx_PtpG8275p1ClassEvalMethodE enumeration for descriptions.
zl303xx_PtpG8275ClockConfigS	equipmentClass	Uint8T	Configurable member defining the alternate equipment class to use for T-GM clocks in holdover (Out-of-spec) for Table 6.4 of G.8275.1.
zl303xx_PtpG8275ClockConfigS	holdoverSupported	zl303xx_BooleanE	Configurable member applicable to T-TSC clocks only. • TRUE: Allows the T-TSC clock to use HOLDOVER Classes (135/165) in BMCA selection. (Like a T-BC). • FALSE: The T-TSC clock always uses class 255 in BMCA selection.

5.2.4.2 PTP CLOCK: PROFILES ITU-T G.8265.1

For a list of the PTP Clock attributes, refer to the following data structure: `zl303xx_PtpTelecomClockCreateS`.

TABLE 5-8: CLOCK SETTINGS RELATED TO ITU-T G.8265.1

Structure	Parameter	Options/Type	Description
zl303xx_PtpTelecomClockCreateS	waitToRestoreSec	Uint32T	Wait to restore time prevents a master from being selected by the BMCA. When a master fails and is later requalified, this timer starts. Set to 0 to disable this functionality.
zl303xx_PtpTelecomClockCreateS	qlHoldOffSec	Uint32T	If no masters are qualified, this determines how long the previous QL value will be retained before reporting QL-DNU/QL-DUS.
zl303xx_PtpTelecomClockCreateS	evtSquelchEn	zl303xx_Boolean	Determines if the ZL303XX_PTP_EVENT_SQUELCH is generated. Should not be set to ZL303XX_TRUE if qlHoldOffSec is greater than 0 (a warning message will be logged).

5.2.4.3 PTP CLOCK: PROFILE IEEE C37.238

For a list of the PTP Clock attributes, refer to the following data structure: `zl303xx_PtpC37p238ClockConfigS`.

TABLE 5-9: CLOCK SETTINGS RELATED TO IEEE C37.238

Structure	Parameter	Options/Type	Description
zl303xx_PtpC37p238ClockConfigS	grandMasterId	Uint16T	Clock ID: Range 0x0003 - 0x00FE for 2011 version (else assume unconfigured). Full range allowed for 2017 version.
zl303xx_PtpC37p238ClockConfigS	localTimeInaccuracy	Uint32T	Local Time Inaccuracy (ns) of this clock. On master clocks, this value is written to the grandmasterTimeInaccuracy field of the 2011 IEEE_C37_238 TLV (0xFFFFFFFF indicates overflow value). When operating as grandmaster, the local time inaccuracy should include the source time inaccuracy from chosen source and updated as conditions change. For 2017 version of the TLV this value is added to the totalTimeInaccuracy field.
zl303xx_PtpC37p238ClockConfigS	networkTimeInaccuracy	Uint32T	Network Time Inaccuracy (ns). This value is calculated by the system designer by summing the individual localTimeInaccuracy values for each TC in the cascade from the grandmaster clock. This value is set at either: a) The grandmaster clock and transmitted in the IEEE_C37_238 TLV as the networkTimeInaccuracy field. b) The end device clock and added to received grandmasterTimeInaccuracy value in the IEEE_C37_238 TLV to compute the TimeInaccuracy value. Note: This is only valid in 2011 profile version.

5.2.4.4 PTP CLOCK: PROFILE IEEE 802.1AS

For a list of the PTP Clock attributes, refer to the following data structure: `zl303xx_Ptp802p1ClockConfigS`.

TABLE 5-10: CLOCK SETTINGS RELATED TO IEEE 802.1AS

Structure	Parameter	Options/Type	Description
zl303xx_Ptp802p1ClockConfigS	gmCapable	zl303xx_BooleanE	Indicates if the time-aware system is capable of being a grandmaster and dictates the value of clockClass in defaultDs. If the value is TRUE the time-aware system is capable of being a grandmaster and if FALSE the timeaware system is not capable of being a grandmaster.

5.2.5 PTP Clock: Unicast Negotiation Capacity

TABLE 5-11: PTP CLOCK CREATE PARAMETERS (UNICAST NEGOTIATION CAPACITY)

Sub-Structure	Parameter	Type	Description
unicastNegotiation	maxAnnounceCount	Uint32T	The maximum number of ANNOUNCE contracts supported.
unicastNegotiation	maxSyncCount	Uint32T	The maximum number of SYNC contracts supported.
unicastNegotiation	maxDelayRespCount	Uint32T	The maximum number of DELAY RESPONSE contracts supported.
unicastNegotiation	maxPdelayRespCount	Uint32T	The maximum number of PEER DELAY RESPONSE contracts supported.
unicastNegotiation	totalPpsMax	Uint32T	The combined Maximum PPS of all contracts of a single message type allowed on this Clock

5.2.6 PTP Clock: User Overrides

TABLE 5-12: PTP CLOCK CREATE PARAMETERS (USER OVERRIDES)

Sub-Structure	Parameter	Type	Description
override	enabled	zl303xx_Boolean	Boolean flag indicating if an egress Override is configured for a particular parameter.
override	defaultDS	zl303xx_DefaultDS	The override values for certain default Data Set members. (Clock Quality, priority1/2 and domain).
override	timePropertiesDS	zl303xx_TimePropertiesDS	The override values for certain Time Properties Data Set members. (UTC Offset & Offset Valid, Time & Frequency Traceable, Synchronization Uncertain and Time Source).
override	currentDS	zl303xx_CurrentDS	The override values for certain Current Data Set members. (Steps Removed).

5.3 CONFIGURATION OF A PTP PORT: ZL303XX_PTPPORTCREATE()

The following API routines are used to Add a PTP Port (refer to the PTP API Reference Manual for complete interface, data type, and syntax details).

TABLE 5-13: PTP PORT CREATE ROUTINES

API Routine	Description
zl303xx_PtpPortCreateStructInit	Data fills the zl303xx_PtpPortCreateS data structure with a set of default port creation parameters. The user may modify individual members prior to creating the PTP port.
zl303xx_PtpPortCreate	Creates a new PTP Port based on the zl303xx_PtpPortCreateS data structure.

In the example code supplied in `zlUserExamples/src/zl303xx_ExamplePtp.c`, the structure initialization and port creation calls are found in functions `examplePtpPortCreateStructInit()` and `examplePtpPortCreate()`, respectively.

The members of the `zl303xx_PtpPortCreateS` are listed in the following table. These may be modified by a user when creating an application. (Complete details of the `zl303xx_PtpPortCreateS` structure can be found in the `zlPtp/include/zl303xx_PtpApiTypes.h` module).

5.3.1 PTP Port: General and Miscellaneous

TABLE 5-14: PTP PORT CREATE PARAMETERS (GENERAL AND MISCELLANEOUS)

Sub-Structure	Parameter	Type	Description
—	clockHandle	zl303xx_PtpClockHandleT	The handle to the clock that this port is attached to.
—	portNumber	Uint16T	Value that will be placed in the sourcePortIdentity field of egress PTP messages. Set to 0 to automatically generate a unique portNumber during port initialization.
—	ptpVersion	Uint16T	The PTP version in use on this port.
—	localAddr	zl303xx_PortAddress	The local address of this port. All messages are transmitted from this address, and unicast messages destined to this address will be processed.
—	extData	void *	Pointer to some external user data. This is passed to the packet-send porting functions. Memory management must be handled externally to PTP.
—	txMsgFn	zl303xx_PtpTxMsgFnT	Function used to transmit PTP messages.
—	requestedHandle	zl303xx_PtpPortHandleT	Request port to be created with a specific handle. Set to ZL303XX_PTP_INVALID to auto-generate a handle.
—	faultPassThroughEn	zl303xx_BooleanE	Configurable member. • TRUE: Allow the port to go to init after a fault • FALSE: The port will stay in faulty state unless explicitly forced to go to init.
—	alternateMaster	zl303xx_Boolean	Controls the behavior of alternateMasterFlag in some transmitted messages. • If FALSE, does NOT set alternateMasterFlag in transmitted messages. • If TRUE, sets the alternateMasterFlag in transmitted messages if the transmitting port is a non-MASTER state. Default TRUE for IEEE 1588-2008 standard behavior (see Table 5-19).

5.3.2 PTP Port: Profile-Specific

TABLE 5-15: PTP PORT CREATE PARAMETERS (PROFILE-SPECIFIC)

Sub-Structure	Parameter	Type	Description
—	delayMechanism	zl303xx_DelayMechanismE	Propagation delay measuring option. May be End-to-End or Peer-to-Peer depending on the Profile used.
—	pdRespTxMethod	zl303xx_PtpPdelayRespMethodE	The method of issuing a Peer-Delay Response. Refer to the data type definition of Clause 11.4.3.b of IEEE-1588-2008 for details.
—	logMinPdelayReqInterval	Sint8T	The minimum permitted mean time interval between successive PDELAY_REQ messages.
—	masterOnly	zl303xx_Boolean	Added by ITU G.8275.2: Optional parameter to prohibit a Port from entering the SLAVE or PASSIVE state. Any ANNOUNCE messages received on a masterOnly Port are not used in the Best Master Clock Algorithm of the profile.
profileCfg	power	zl303xx_PtpC37p238PortConfigS	Power Profile-specific port creation members.
profileCfg	g8275p1	zl303xx_PtpG8275PortConfigS	Telecom Phase Profile-specific port creation members.

5.3.2.1 PTP PORT: PROFILES ITU-T G.8275.x

For a list of the PTP Port attributes, refer to the following data structure: `zl303xx_PtpG8275PortConfigS`.

TABLE 5-16: PORT SETTINGS RELATED TO ITU-T G.8275.x

Structure	Parameter	Option/Type	Description
zl303xx_PtpG8275PortConfigS	localPriority	Uint8T	Configurable member defining the localPriority attribute of the local port. Used in the Best Master Clock Algorithm of the G.8275.1 profile.
zl303xx_PtpG8275PortConfigS	tsIfRate	zl303xx_PtpG8275 TimestampIfRateTlvS	Configurable member defining the Timestamp Interface attributes associated with this local PTP port. This data is used to issue the Timestamp Interface Rate TLV of the G.8275.2 profile. Refer to Table 5-17.
zl303xx_PtpG8275PortConfigS	notSlave	zl303xx_BooleanE	Deprecated: The following member value is ignored in favor of the optionalPortDS::masterOnly definition.

TABLE 5-17: SETTINGS RELATED TO INTERFACE RATE TLV OF THE G.8275.2

Structure	Parameter	Options/ Type	Description
zl303xx_PtpG8275TimestampIfRateTlvS	interfaceBitPeriod	Uint64S	The period of 1-bit of the transmitting PTP timestamp interface, excluding line encoding. The value is encoded as an unsigned integer in units of attoseconds (10^-18) to accommodate interface bit periods less than 1 ns.
zl303xx_PtpG8275TimestampIfRateTlvS	numberBitsBeforeTimestamp	Uint16T	The length of the packet prior to the timestamp point, in bits.
zl303xx_PtpG8275TimestampIfRateTlvS	numberBitsAfterTimestamp	Uint16T	The length of the packet after the timestamp point, in bits.

5.3.2.2 PTP PORT: PROFILE IEEE C37.238

For a list of the PTP Port attributes, refer to the following data structure: `zl303xx_PtpC37p238PortConfigS`.

TABLE 5-18: PORT SETTINGS RELATED TO IEEE C37.238

Structure	Parameter	Options/ Type	Description
zl303xx_PtpC37p238PortConfigS	empty	Uint32T	This structure is currently not used by the profile. It exists for the purpose of possible future expansion.

5.3.2.3 PTP PORT: PROFILE IEEE 802.1as

For a list of the PTP Port attributes, refer to the following data structure: `zl303xx_Ptp802p1PortConfigS`.

TABLE 5-19: PORT SETTINGS RELATED TO IEEE 802.1as

Structure	Parameter	Options/Type	Description
zl303xx_Ptp802p1PortConfigS	msgIntvlReq	zl303xx_Ptp802p1MsgIntvlReqTlvS	Defining the Message Interval attributes associated with this local PTP port. This data is used to issue the Message Interval Request TLV of the IEEE802.1AS profile.
zl303xx_Ptp802p1PortConfigS	neighborPropDelayThreshold	zl303xx_TimeInterval	Defining the neighbor propagation delay threshold for this port. This will be used to decide if the port is capable of running IEEE802.1AS profile (asCapable).
zl303xx_Ptp802p1PortConfigS	allowedLostResponses	Uint8T	Indicates the allowed number of Peer-Delay response packets dropped before PTSF-asCapable is declared.

5.3.2.4 PTP PORT: DEFAULT PROFILE EDITION 3 (v2.1)

For a list of all the PTP Port attributes refer to the following data structure: `zl303xx_PtpPortConfigS`.

TABLE 5-20: PORT SETTINGS RELATED TO DEFAULT PROFILE EDITION 3 (v2.1)

Structure	Parameter	Options/Type	Description
zl303xx_PtpPortConfigS	ptpMinorVersion	Uint4T	The PTP minor version in use on this port. This value is stored in the zl303xx_PortDS::minorVersionNumber field and used in egress header minorVersionPTP field for all message types sent on this port. Set to 1 for PTP v2.1 (Edition 3) operation on this port. See filterMinorVersionPtp for ingress filter options. Default: ZL303XX_PTP_DEFAULT_PORT_PTP_MINOR_VERSION (0) See zl303xx_PtpStreamEgressOverrideAncFieldSet() and ZL303XX_PTP_STREAM_OVERRIDE_MINOR_VERSION_PTP for stream-level egress header field override.
zl303xx_PtpPortConfigS	filterMinorVersionPtp	zl303xx_PtpMinorVersionPtpFilterE	For PTPv2.1, optionally discard ingress received packets based on minorVersionPtp header field value. This can be used to limit a port to work with only a particular minorVersionPtp. For example: accept only v2.1 packets, or accept only v2.0 packets, or accept any packet). Default ZL303XX_PTP_MINOR_VERSION_PTP_ACCEPT_ANY (do not filter ingress packets).

5.3.3 PTP Port: Acceptable Master and Acceptable Slave Tables

The Acceptable Master Table (AMT) and Acceptable Slave Tables (AST) provide access control on a port. Access is restricted based on received packet source network address as provided by the user (`zl303xx_PortAddress` type). Note the source address of a packet is provided by user as obtained from transport layer (e.g. socket) or similar means as it is not available in the PTP header or payload.

When enabled, the Acceptable Master Table (AMT):

- Drops ingress messages (all types) if the source address of a packet does not match an entry in AMT.
- If AMT entry is found, and message is ANNOUNCE and the AMT entry has non-zero alternatePriority1 configuration then the ANNOUNCE data grandmaster-Priority1 field is replaced with configured value.

When enabled, the Acceptable Slave Table (AST):

- Rejects unicast negotiation contract requests (SIGNALLING) if the source address of a packet does not match an entry in AST (i.e. replies with grant duration 0).
- Drops ingress DELAY_REQ messages if the source address of a packet does not match an entry in AST.
- Note when changing AST entries dynamically at runtime existing unicast negotiation contracts remain active until the granted contract term has expired. However, existing DELAY_RESP contract may not be honored due to above.

Related dynamic APIs:

- `zl303xx_PtpAcceptableMasterTableEnSet`
- `zl303xx_PtpAcceptableMasterAdd`
- `zl303xx_PtpAcceptableMasterRemove`
- `zl303xx_PtpAcceptableSlaveTableEnSet`
- `zl303xx_PtpAcceptableSlaveAdd`
- `zl303xx_PtpAcceptableSlaveRemove`

Configurable port parameters are presented in Table 5-21.

TABLE 5-21: PTP PORT CREATE PARAMETERS (ACCEPTABLE MASTER)

Sub-Structure	Parameter	Type	Description
—	acceptableMasterTableEnabled	zl303xx_Boolean	Enables the use of an acceptable master table.
—	acceptableSlaveTableEnabled	zl303xx_Boolean	Enables the use of an acceptable slave table.
—	acceptableMasterTableLength	Uint16T	The number of acceptable master table entries or acceptable slave table entries to allocate. Note same number of entries for AST and AMT.

5.3.4 PTP Port: Unicast Negotiation

TABLE 5-22: PTP PORT CREATE PARAMETERS (UNICAST NEGOTIATION)

Sub-Structure	Parameter	Type	Description
unicastNegotiation	enabled	zl303xx_Boolean	Indicates unicast negotiation is used on this port.
unicastNegotiation	stackedTlvEn	zl303xx_Boolean	Enables sending of stacked unicast negotiation TLVs in a single Signaling Message.
unicastNegotiation	maxAnnounceRate	Sint8T	Maximum packet rate per negotiated ANNOUNCE Contract on this Port
unicastNegotiation	maxAnnounceGrantSecs	Uint32T	The maximum duration per negotiated ANNOUNCE contract on this Port
unicastNegotiation	maxAnnounceCount	Uint32T	The maximum number of ANNOUNCE contracts to support on this Port.
unicastNegotiation	maxSyncRate	Sint8T	Maximum packet rate per negotiated SYNC Contract on this Port
unicastNegotiation	maxSyncGrantSecs	Uint32T	The maximum duration per negotiated SYNC contract on this Port
unicastNegotiation	maxSyncCount	Uint32T	The maximum number of SYNC contracts to support on this Port.
unicastNegotiation	maxDelayRespRate	Sint8T	Maximum packet rate per negotiated DELAY-RESP Contract on this Port
unicastNegotiation	maxDelayRespGrantSecs	Uint32T	The maximum duration per negotiated DELAY-RESP contract on this Port
unicastNegotiation	maxDelayRespCount	Uint32T	The maximum number of DELAY-RESP contracts to support on this Port.
unicastNegotiation	maxPdelayRespRate	Sint8T	Maximum packet rate per negotiated PEER-DELAY-RESP Contract on this Port
unicastNegotiation	maxPdelayRespGrantSecs	Uint32T	The maximum duration per negotiated PEER-DELAY-RESP contract on this Port
unicastNegotiation	maxPdelayRespCount	Uint32T	The maximum number of PEER-DELAY-RESP contracts to support on this Port.
unicastNegotiation	totalPpsMax	Uint32T	The combined Maximum PPS of all contracts of a single message type allowed on this Port.
unicastNegotiation	denyServiceRequests	zl303xx_Boolean	Added by ITU G.8275.2: Set to ZL303XX_FALSE to grant a contract request by default unless explicitly set otherwise using zl303xx_PtpPortDenyServiceRequests(). Set to ZL303XX_TRUE to deny requests by default. Also used to prevent a PTP Port from entering MASTER mode in ITU-T G.8275.1 operation. This is done to enable moritoring of multicast masters when unicastMonitorTiming is enabled (refer to "Monitoring Connections" & "PTP Stream: Unicast and Negotiation")
unicastNegotiation	grantTableLength	Uint16T	Maximum number of far-end PortAddresses that can be configured using zl303xx_PtpGrantConfigSet(). Must be set to a non-zero value if defaultGrant is set to ZL303XX_FALSE.
unicastMasterTable	maxTableSize	Uint16T	The maximum number of permitted entries in the Unicast Master Table.
unicastMasterTable	logQueryInterval	Sint8T	The log2 of the mean interval, in seconds, between requests from a node for a unicast ANNOUNCE message contract.

5.4 CONFIGURATION OF A PTP STREAM: ZL303XX_PTPSTREAMCREATE()

The following API routines are used to Add a PTP Stream (refer to the PTP API Reference Manual for complete interface, data type, and syntax details).

TABLE 5-23: PTP STREAM CREATION ROUTINES

API Routine	Description
zl303xx_PtpStreamCreateStructInit	Data fills the zl303xx_PtpStreamCreateS data structure with a set of default stream creation parameters. The user may modify individual members prior to creating the PTP stream.
zl303xx_PtpStreamCreate	Creates a new PTP Stream based on the zl303xx_PtpStreamCreateS data structure.

In the example code supplied in `zlUserExamples/src/zl303xx_ExamplePtp.c`, the structure initialization and stream creation calls are found in functions `examplePtpStreamCreateStructInit()` and `examplePtpStreamCreate()`, respectively.

The members of the `zl303xx_PtpStreamCreateS` are listed in the following table. These may be modified by a user when creating an application. Complete details of the `zl303xx_PtpStreamCreateS` structure can be found in the `zlPtp/include/zl303xx_PtpApiTypes.h` module.

5.4.1 PTP Stream: General and Miscellaneous

TABLE 5-24: PTP STREAM CREATE PARAMETERS (GENERAL AND MISCELLANEOUS)

Sub-Structure	Parameter	Type	Description
—	portHandle	zl303xx_PtpPortHandleT	The handle to the port that this stream is attached to.
—	destAddr	zl303xx_PortAddress	The destination port address of this connection.
—	requestedHandle	zl303xx_PtpStreamHandleT	Request stream to be created with a specific handle. Set to ZL303XX_PTP_INVALID to auto-generate a handle.
—	extData	void *	Pointer to some external user data. Memory management must be handled externally to PTP.
—	numTsRecords	Uint16T	The length of the time stamp record queue for this stream. The queue is used to store egress SYNC and DELAY_REQ time stamps, depending on the stream state (which determines what type of messages will be transmitted). Must be a power of 2.
—	numTwoStepRecords	Uint16T	The length of the two-step record queue for this stream. This queue is used to match time stamps for Sync and Follow_Up messages. Must be a power of 2.
—	createFlags	Uint32T	Boolean flags for various stream options.
—	delayReqTimeSetRequired	zl303xx_Boolean	According to the 1588-2008 Std., a SYNC message must be received before issuing a Delay Request. This parameter enforces a stricter condition that the Time (ToD) also needs to be set on the local hardware before a Delay Request message can be sent.
—	nextTxSequenceId	Uint16T	Initial values for the next transmitted sequenceId. Used for warm start operation to prevent jumps in sequenceId.
delayAsymmetry	ingress	zl303xx_TimeInterval	Specifies the delay asymmetry of a stream (if known). These are added to, or subtracted from correctionField values of received or transmitted event messages as per Clause 11 of the IEEE-1588-2008 Std.
delayAsymmetry	egress	zl303xx_TimeInterval	

5.4.2 PTP Stream: Message Rates, Timeouts, and Intervals

TABLE 5-25: PTP STREAM CREATE PARAMETERS (MESSAGE RATES, TIMEOUTS, AND INTERVALS)

Sub-Structure	Parameter	Type	Description
—	logAnnounceInterval	Sint8T	The log2 of the mean time interval between successive ANNOUNCE messages sent from this stream.
—	announceReceiptTimeout	Uint8T	The number of announceIntervals that have to pass without receipt of an ANNOUNCE message before the occurrence of the event ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES.
—	logSyncInterval	Sint8T	The mean time interval between successive SYNC messages.
—	logMinDelayReqInterval	Sint8T	The mean time interval between successive DELAY-REQUEST messages.
—	logMinPdelayReqInterval	Sint8T	The mean time interval between successive PEER_DELAY_REQUEST messages.

5.4.3 PTP Stream: Profile-Specific

TABLE 5-26: PTP STREAM CREATE PARAMETERS (PROFILE-SPECIFIC)

Sub-Structure	Parameter	Type	Description
—	mode	zl303xx_PtpStreamModeE	How this stream will determine if it is master or slave. This can be forced or automatically determined using BMCA.
profileCfg	power	zl303xx_PtpC37p238StreamConfigS	Power Profile-specific stream creation members.
profileCfg	g8275p1	zl303xx_PtpG8275StreamConfigS	Telecom Phase Profile-specific stream creation members.
—	padTlvLen	Uint16T	The number of bytes to pad at the end of each EVENT messageType in the PAD TLV (Clause 16.13 of IEEE-2017). The length includes the TYPE and LENGTH fields so therefore must have a minimum pad length of 4. Relates to IEC Power Profile.
—	maxClockClass	Uint8T	The maximum acceptable clock class that will allow a master to be qualified. Set to 0 to disable.
uncalibrated	anncIntervals	Uint8T	The number of Announce Intervals that have to pass before a stream in the UNCALIBRATED state can transition to the SLAVE state. Range: 0-255. A value of 0 disables this restriction.
uncalibrated	lockRequired	zl303xx_Boolean	Flag indicating if the clock must be in a LOCKED state before a stream can transition to the SLAVE State.
—	keepAliveSec	Uint32T	The duration in seconds for which to issue a 'keep-alive' event for the stream if no other related event has been issued (i.e. BMCA UPDATE, STATE CHANGE, CONFIG CHANGE all count as a stream event).
smpteSmTlvConfig	tlvAppend	zl303xx_Boolean	Flag indicating if SMPTE 2059-2 Sync Metadata TLV should be appended to outgoing management messages. If this is set to TRUE, the TLV shall be appended every one second on ports in MASTER state on Grandmaster clocks. zl303xx_PtpSmpteStreamSyncMetadataTlvSend (Table 6-17) can also be used to manually send the TLV on a given stream.
smpteSmTlvConfig	tlvProcess	zl303xx_Boolean	Flag indicating if incoming SMPTE 2059-2 Sync Metadata TLV should be processed.

5.4.3.1 PTP STREAM: PROFILES ITU-T G.8275.x

For a list of the PTP Port attributes, refer to the following data structure: `zl303xx_PtpG8275StreamConfigS`.

TABLE 5-27: STREAM SETTINGS RELATED TO ITU-T G.8275.x

Structure	Parameter	Options/Type	Description
zl303xx_PtpG8275StreamConfigS	localPriority	Uint8T	Configurable member defining the localPriority attribute of the local stream. Takes priority over the associated Port localPriority. Used in the Best Master Clock Algorithm of the G.8275.1 profile.
zl303xx_PtpG8275StreamConfigS	notSlave	zl303xx_BooleanE	Deprecated: The following member value is ignored in favour of the optionalPortDS::masterOnly and stream::config::operMode definitions.

5.4.3.2 PTP STREAM: PROFILE IEEE C37.238

For a list of the PTP Stream attributes, refer to the following data structure: `zl303xx_PtpC37p238StreamConfigS`.

TABLE 5-28: STREAM SETTINGS RELATED TO IEEE C37.238

Structure	Parameter	Options/Type	Description
zl303xx_PtpC37p238StreamConfigS	profileTlvRequired	zl303xx_BooleanE	Determines if the Profile TLV is required to be appended to received Announce messages. TRUE: any received Announce message with the TLV missing is discarded. Also, all transmitted Announce messages will append the TLV regardless of the 'append' setting.
zl303xx_PtpC37p238StreamConfigS	profileTlvAppend	zl303xx_BooleanE	Determines if the Profile TLV will be appended to transmitted Announce messages. Even if the ‘required’ field is FALSE, an application may still choose to append the TLV.
zl303xx_PtpC37p238StreamConfigS	profileTlvVersion	Uint32T	Profile TLV version (2011 or 2017)
zl303xx_PtpC37p238StreamConfigS	altTimeOffsetTlvRequired	zl303xx_BooleanE	Determines if the Alternate Time TLV is required to be appended to received Announce messages. TRUE: any received Announce message with the TLV missing is discarded. Also, all transmitted Announce messages will append the TLV regardless of the 'append' setting.
zl303xx_PtpC37p238StreamConfigS	altTimeOffsetTlvAppend	zl303xx_BooleanE	Determines if the Alternate Time TLV will be appended to transmitted Announce messages. Even if the ‘required’ field is FALSE, an application may still choose to append the TLV.
zl303xx_PtpC37p238StreamConfigS	networkTimeInaccuracyMax	Uint32T	Network Time Inaccuracy Max for this Stream (ns). When ingress networkTimeInaccuracy (or totalTimeInaccurcy 2017) exceeds this threshold the stream will raise ZL303XX_PTSF_NETWORK_TIME_INACC_EXCEEDED and stream will be disqualified from foreign master table (FMT) such that it will not take part in BMCA for selection.

5.4.4 PTP Stream: Unicast and Negotiation

TABLE 5-29: PTP STREAM CREATE PARAMETERS (UNICAST NEGOTIATION)

Sub-Structure	Parameter	Type	Description
—	unicast	zl303xx_Boolean	Indicates that destAddr corresponds to a unicast transport layer protocol address (not multicast or broadcast). This member also sets unicastFlag in the flagField of the common PTP message header.
—	unicastNegotiationDuration	Uint32T	Duration in seconds for which to request unicast contracts.
—	unicastMonitorTiming	zl303xx_Boolean	Flag indicating that a stream in the MASTER state that also has a UMT entry to a far-end-server will continue to request both ANNOUNCE and Timing service (contracts) with that server: • FALSE (default): request only ANNOUNCE service as long as the UMT entry exists, • TRUE: request ANNOUNCE, SYNC and DELAY services as configured by the UMT (even in MASTER mode). Also used to send Delay_Req messages from a PTP port in PASSIVE state with ITU-T G.8275.1 (refer to "Monitoring Connections").
—	unicastNegotiationRenew	Uint32T	The time in seconds before unicastNegotiationDuration expires that the unicast negotiation contract will be renewed.
unicastNegotiation:: bestEffort	enabled	zl303xx_Boolean	Enables a requester of a unicast contract to request a lower rate if its original request was denied.
unicastNegotiation:: bestEffort	intervalMax	Sint8T	The minimum packet interval to accept (intervalMax corresponds to a minimum PPS). Default = -128 (PTP_MESSAGE_INTERVAL_MIN); (so enabling this feature without changing intervalMax will inhibit negotiation).

5.4.5 PTP Stream: User Overrides

TABLE 5-30: PTP STREAM CREATE PARAMETERS (USER OVERRIDES)

Sub-Structure	Parameter	Type	Description
override	anncOvrdEn	zl303xx_Boolean	Boolean flags indicating if an Override is configured for the egress data set parameter (Refer to the zl303xx_PtpStreamOverrideAnncEgressE Type for a list of parameters).
override	ptpHeader	zl303xx_PtpV2MsgHeaderS	PTP Header egress override values (if enabled by anncOvrdEn)
override	timeProperties	zl303xx_TimePropertiesDS	Time Property egress override values (if enabled by anncOvrdEn)
override	anncData	zl303xx_PtpV2MsgAnnounceS	ANNOUNCE egress override values (if enabled by anncOvrdEn)

5.5 CONFIGURATION OF VIRTUAL PTP PORT (OPTIONAL)

A Virtual PTP Port is intended to model a physical clock signal as a PTP packet input. The PTP software was originally developed to only manage packet interfaces. The Virtual Port contains a number of configuration parameters that can be mapped to typical PTP attributes. Each time the BMC Algorithm is executed on the clock, the Virtual Port data is converted to equivalent comparison data and evaluated as if it were a regular packet interface.

An example of creating and managing a Virtual Port can be found in the sample routine, `examplePtpTelecomPhaseG8275p1Master`.

A summary of the API routines used to manage a Virtual Port is listed in the following table:

TABLE 5-31: PTP CLOCK VIRTUAL PORT API ROUTINES

API Routine	Description
zl303xx_PtpVirtualPortCreateStructInit	Initializes a zl303xx_PtpVirtualPortConfigS structure with default values.
zl303xx_PtpVirtualPortCreate	Creates a Virtual Port with the zl303xx_PtpVirtualPortConfigS parameters provided and reserves a portHandle value that may be used for future management of the port.
zl303xx_PtpVirtualPortDelete	Deletes the Virtual Port for the specified portHandle.
zl303xx_PtpVirtualPortGet	Gets the zl303xx_PtpVirtualPortConfigS configuration of a Virtual Port (for the specified portHandle).
zl303xx_PtpVirtualPortSet	Sets the zl303xx_PtpVirtualPortConfigS configuration of a Virtual Port (for the specified portHandle).
zl303xx_PtpVirtualPortPtsfSet	Enables or Disables a Virtual Port (for the specified portHandle). When enabled, the Virtual Port configuration is included in the BMCA selection.

TABLE 5-32: PTP VIRTUAL PORT CONFIGURATION PARAMETERS (ZL303XX_PTPVIRTUALPORTCONFIGS)

Sub-Structure	Parameter	Type	Description
—	clockHandle	zl303xx_PtpClockHandleT	The handle to the clock that this port is attached to.
—	portNumber	Uint16T	The identity of this port. This value is stored in the port's zl303xx_PortDS::portIdentity::portNumber field. Setting to INVALID (-1) will automatically generate a unique portNumber during initialization.
—	ptsf	zl303xx_BooleanE	Flag indicating whether the Virtual Port is Failed.
—	anncData	zl303xx_PtpV2MsgAnnounceS	ANNOUNCE egress override values (if enabled by anncOvrdEn)
—	prtcConnected	zl303xx_PtpPrtcLevelE	Flag indicating if this port is connected to a PRTC source. Options are: • ZL303XX_PTP_PRTC_NOT_CONNECTED, • ZL303XX_PTP_PRTC_CONNECTED, • ZL303XX_PTP_PRTC_ENHANCED
—	clockIdentity	zl303xx_ClockIdentity	Clock Identity of the GM being virtually received on this port.
—	clockQuality	zl303xx_ClockQuality	Clock Quality values of the GM being modeled on this port.
—	priority1	Uint8T	priority1 value of the GM being modeled on this port.
—	priority2	Uint8T	priority2 value of the GM being modeled on this port.
—	stepsRemoved	Uint16T	Steps removed to the GM being modeled on this port.
—	timeProperties	zl303xx_TimePropertiesDS	Time Property values of the GM being modeled on this port.
—	localPriority	Uint8T	Local Priority of the port (used by various profiles).
—	extData	void *	Pointer to some external user data. This is passed to the packet-send porting functions. Memory management must be handled externally to PTP.

5.6 CONFIGURATION OF EXTERNAL USER DATA (OPTIONAL)

5.6.1 How to Associate External User Data with a PTP Component

A user-defined data structure can be associated with certain PTP objects through the use of a generic (void*). Although not used by the PTP objects directly, it allows the user to associate proprietary data with each object so that when a PTP Event occurs, the pointer to the user data is supplied to allow quick access by the management system. Typically this functionality is not used.

5.6.1.1 EXTERNAL DATA ASSOCIATED WITH PTP OBJECTS

As seen from the previous sections, this pointer can be set in the following structures (see member 'void *extData' in each associated data structure):

- `zl303xx_PtpClockCreateS`
- `zl303xx_PtpPortCreateS`
- `zl303xx_PtpStreamCreateS`

It is possible to change the extData associated with a previously created Stream using the following routine (changes are not currently allowed to the Clock & Port extData configuration):

- `zl303xx_PtpStreamExtDataSet()`

The extData pointer for any created PTP object can be retrieved later by calling one of the following API commands (as appropriate for the object):

- `zl303xx_PtpClockExtDataGet()`
- `zl303xx_PtpPortExtDataGet()`
- `zl303xx_PtpStreamExtDataGet()`

It is the user's responsibility to manage the allocation and freeing of any external structure memory. This includes freeing the memory after manually calling a delete function and freeing it inside of a delete event handler.

5.6.1.2 EXTERNAL DATA ASSOCIATED WITH RECEIVED PACKETS

In rare situations, it may be necessary to pass some extra low level data about a received packet through the PTP application. This data can be written into the following 8-byte buffer; `zl303xx_PtpRxMsgDataS::extData`. This data is only available in the stream creation event, as structure member `zl303xx_PtpEventStreamCreateS::extData`.

5.7 CONFIGURATION FOR SHUTDOWN (STOPPING) PTP SERVICE

5.7.1 Terminating a PTP Application or its Components

The most straight-forward way of shutting down the PTP application is to call either following PTP Node API:

- `zl303xx_PtpShutdownWithOptions()`, or
- `zl303xx_PtpShutdown()` legacy interface

By default, this will delete all stream, port, and clock objects that are currently running (see forceDelete option) and events are generated (see fireEvent option) to notify the management system so that any other user resources associated with the objects can be freed. For more information on event handling, see "EVENT INTERFACE".

To prevent deleting existing objects, use `zl303xx_PtpShutdownWithOptions()` with forceDelete ZL303XX_FALSE which will return a failure status if there are objects still created.

The PTP timer task and the PTSF module will also be deleted by PTP shutdown function call.

An alternative sequence of closure is to first delete the PTP stream, then the PTP port then the PTP clock.

The individual clock, port, and stream objects can be deleted directly by calling the following functions:

- `zl303xx_PtpClockDelete()`
- `zl303xx_PtpPortDelete()`
- `zl303xx_PtpStreamDelete()`

Note: Calling these functions directly does not generate a corresponding event by default (see fireEvent options in the delete API parameters). Any user resources associated with the deleted object must be cleaned up by the management system after the direct call to these functions.

The following API routines are used to Delete a PTP Clock, Port, or Stream (refer to the PTP API Reference Manual for complete interface, data type, and syntax details).

TABLE 5-33: PTP NODE DELETION ROUTINES

API Routine	Description
zl303xx_PtpShutdownOptionsStructInit	Data fills the zl303xx_PtpShutdownOptionsS data structure with a set of default node deletion parameters.
zl303xx_PtpShutdownWithOptions	Deletes the PTP Node. Alternatively, see legacy zl303xx_PtpShutdown() interface.

TABLE 5-34: PTP CLOCK DELETION ROUTINES

API Routine	Description
zl303xx_PtpClockDeleteStructInit	Data fills the zl303xx_PtpClockDeleteS data structure with a set of default clock deletion parameters.
zl303xx_PtpClockDelete	Deletes the specified PTP Clock.

TABLE 5-35: PTP PORT DELETION ROUTINES

API Routine	Description
zl303xx_PtpPortDeleteStructInit	Data fills the zl303xx_PtpPortDeleteS data structure with a set of default port deletion parameters.
zl303xx_PtpPortDelete	Deletes the specified PTP Port.

TABLE 5-36: PTP STREAM CREATION AND DELETION ROUTINES

API Routine	Description
zl303xx_PtpStreamDeleteStructInit	Data fills the zl303xx_PtpStreamDeleteS data structure with a set of default stream deletion parameters.
zl303xx_PtpStreamDelete	Deletes the specified PTP Stream.

NOTES:

Chapter 6. Modify Configuration

6.1 MODIFY PTP CLOCK CONFIGURATION

6.1.1 Modify PTP Clock: Description Configuration

The following API routines are used to update a PTP Clock's Description settings. These are outlined in the 'CLOCK_DESCRIPTION' management TLV section of the IEEE-1588 standard (Section 15.5.3.1.2 at the time of this writing).

TABLE 6-1: PTP CLOCK DESCRIPTION UPDATE ROUTINES

API Routine	Description
zl303xx_PtpNodeProductDescriptionStructInit	Retrieves the default value of the 'Product Description' (from the zl303xx_PtpSetup.h file).
zl303xx_PtpNodeProductDescriptionSet	Sets the value of the PTP Clock's 'Product Description'. Use zl303xx_PtpNodeProductDescriptionGet() to retrieve the current setting
zl303xx_PtpNodeManufacturerIdentitySet	Sets the value of the PTP Clock's 'Manufacturer Identity'. Use zl303xx_PtpNodeManufacturerIdentityGet() to retrieve the current setting.
zl303xx_PtpNodeUserDescriptionStructInit	Retrieves the default value of the 'User Description' (from the zl303xx_PtpSetup.h file).
zl303xx_PtpNodeUserDescriptionSet	Sets the value of the PTP Clock's 'User Description'.
zl303xx_PtpNodeRevisionDataStructInit	Retrieves the default value of the 'Revision Data' (from the zl303xx_PtpSetup.h file).
zl303xx_PtpNodeRevisionDataSet	Sets the value of the PTP Clock's 'Revision Data'. Use zl303xx_PtpNodeRevisionDataGet() to retrieve the current setting

6.1.2 Modify PTP Clock: System Integration Parameters

When a PTP Clock is created, several parameters are configured in order to allow it to interface with the user platform. These include:

- An interface to the user management system (event notification).
- An interface to retrieve system specific hardware data (ex: Time-of-Day)
- Addition of user specific external data.

The following API routines can be used to modify these routines at run-time:

TABLE 6-2: PTP CLOCK INTEGRATION ROUTINES

API Routine	Description
zl303xx_PtpClockEventCallbackSet	Sets the PTP Event Call-back routine: all management and timing data are exported via this user provided binding
zl303xx_PtpClockHardwareCmdFnSet	Sets the local Hardware Command Interface: this is required for the software to access such things as the local time-of-day, etc.
zl303xx_PtpClockExtDataSet	User Specific External Clock Data: this user specific data is included any time a clock event is produced (allows quick access for external management processing).

6.1.3 Modify PTP Clock: Data Sets

The routines outlined in the table below are used to update the parameters of the following IEEE-1588 data sets:

- Default Data Set
- Time Properties Data Set

Updating of these parameters automatically triggers a re-evaluation of all known ports and local dynamic data related to the BMC algorithm.

Refer to the IEEE-1588 standard and the `zl303xx_PtpStdTypes.h` header for further details.

TABLE 6-3: PTP CLOCK DATA SET UPDATE ROUTINES

API Routine	Description
zl303xx_PtpClockDefaultDSDefaultParams	Data fills the zl303xx_DefaultDS data structure with a set of default parameters for the associated data set.
zl303xx_PtpClockDefaultDSSet	Updates the PTP Clock Default Data Set with the supplied values in the zl303xx_DefaultDS data structure.
zl303xx_PtpTimePropertiesDefaultParams	Data fills the zl303xx_TimePropertiesDS data structure with a set of default parameters for the associated data set.
zl303xx_PtpTimePropertiesSetLocal	Updates the Time Properties configuration of the local hardware. This may be propagated to the PTP Clock Time Properties Data Set depending on the state of the local PTP Clock and the BMC re-evaluation.

6.1.4 Modify PTP Clock: ClockQuality (or ClockClass) Value

The routines outlined in the table below can be used to update the IEEE-1588 clock quality settings of a PTP Clock. These include the following:

- Clock class
- Clock accuracy
- Offset scaled variance

Updating of these parameters automatically triggers a re-evaluation of all known ports and local dynamic data related to the BMC algorithm. Refer to the IEEE-1588 standard and the `zl303xx_PtpStdTypes.h` header for further details.

TABLE 6-4: PTP CLOCK QUALITY UPDATE ROUTINES

API Routine	Description
zl303xx_PtpClockQualityDefaultParams	Data fills the zl303xx_ClockQuality data structure with a set of default parameters.
zl303xx_PtpClockQualitySet	Updates the PTP Clock Quality with the supplied values in the zl303xx_ClockQuality data structure.
zl303xx_PtpClockClassSet	Updates the PTP Clock Class with the supplied class value.

6.1.5 Modify PTP Clock: Priority Value

This routine outlines how to update the IEEE-1588 clock priority values (P1 & P2) of a PTP Clock.

Updating either of these priority values automatically triggers a re-evaluation of all known ports and local dynamic data related to the BMC algorithm. Refer to the IEEE-1588 standard and the `zl303xx_PtpStdTypes.h` header for further details.

TABLE 6-5: PTP CLOCK PRIORITY UPDATE ROUTINE

API Routine	Description
zl303xx_PtpClockPrioritySet	Updates the Clock::priority1 and/or Clock::priority2 value(s) of the specified PTP Clock. If either the priority1 or priority2 arguments are set to NULL, then that parameter remains unchanged.

6.1.6 Modify PTP Clock: Two-Step Flag

Ordinarily, the two-step flag of a PTP Clock is a static attribute. However, an API has been provided to update this parameter at run-time. (It is assumed that the user platform has the ability to detect the flag from the PTP message header and take appropriate action). Refer to the IEEE-1588 standard and the `zl303xx_PtpStdTypes.h` header for further details.

TABLE 6-6: PTP CLOCK TWO-STEP UPDATE ROUTINE

API Routine	Description
zl303xx_PtpClockTwoStepFlagSet	Updates the Clock::twoStep flag of the specified PTP Clock.

6.1.7 Modify PTP Clock: Domain Number

Changing the Clock domain value at run-time will cause all existing foreign server data to be flushed because it was collected under the previous domain setting.

TABLE 6-7: PTP CLOCK DOMAIN UPDATE ROUTINE

API Routine	Description
zl303xx_PtpClockDomainNumberSet	Updates the Clock::domainNumber of the specified PTP Clock.

6.1.8 Modify PTP Clock: Maximum stepsRemoved Value

This parameter allows a clock to discard ANNOUNCE messages if the `stepsRemoved` value in the ingress message is above (or equal to) a configured level. By default, this limit is set to 255.

Additionally, the following PTP Event Notification is issued when the limit condition is detected:

- ZL303XX_PTP_EVENT_RX_STEPS_REMOVED_EXCEEDED: Packages the event data into a `zl303xx_PtpEventRxStepsRemovedExceededS` message and calls the routine, `examplePtpEventRxStepsRemovedExceeded()` (sample code provided).

To integrate this event, calls to the routines listed above would need to be executed at the event notification output of the PTP stack contained in `zl303xx_ExamplePtp.c`.

TABLE 6-8: PTP CLOCK MAXSTEPSREMOVED ROUTINE

API Routine	Description
zl303xx_PtpClockMaxStepsRemovedSet	Sets the stepsRemoved limit (MAX) for a clock. If the stepsRemoved field of a received ANNOUNCE Message is below the limit, it will be accepted. Otherwise, it is discarded.

6.1.9 Modify PTP Clock: Maximum Packet Rate Service Limit

A PTP clock may have multiple ports, each of which may have multiple clients connected. It is possible to set a maximum packet rate Service Limit for a clock. This prohibits the clock from granting too high of a transmit rate even though each individual client is within the port and stream packet rate limits.

If a lower rate than that of the current active contracts is provisioned, the existing contracts will not be canceled, but rather the new limit will be applied to any renewals. (Refer to the PTP API Reference Manual for complete interface, data type, and syntax details).

TABLE 6-9: PTP CLOCK PACKET RATE SERVICE LIMIT ROUTINE

API Routine	Description
zl303xx_PtpClockTotalPpsMaxSet	Sets the packet rate service limit for the specified PTP clock.
zl303xx_PtpClockTotalPpsGet	Returns the total PPS counters for all unicast negotiated message types (Announce, Sync, and Delay_Resp) on a specified clock.

6.1.10 Modify PTP Clock: PATH_TRACE TLV Operation

Refer also to IEEE-1588-2008: Clause 16.2 for a full description.

This functionality allows a clock to determine if a timing loop exists in its network. In general, the feature functions in the following manner:

- A Master clock issues an ANNOUNCE message with a PATH TRACE TLV containing its own clock identity appended.
- Each subsequent node receives the ANNOUNCE message and (assuming they select that Master as the best clock) issue their own ANNOUNCE messages with the received PATH-TRACE List and append their own clock identity.
- If a clock ever receives a PATH TRACE TLV containing its own clock identity, it will discard that message.

6.1.10.1 MEMORY CONSIDERATIONS

The number of hops in the PATH TRACE TLV may vary with each Master clock selected. The list itself is an array of `clockIdentity` structures (8 bytes each) and is stored at the clock level. By default the number of entries in the list is 0 and no memory is allocated.

When the first PATH TRACE TLV is received:

- The clock dynamically allocates enough memory for the list and updates the maximum and current size of the memory allocated with this value.
- If the next PATH TRACE TLV has fewer entries, then the current size of the memory allocated is updated but the maximum size remains the same (because the smaller list could fit into the larger buffer.
- If the next PATH TRACE TLV has more entries, then a new larger block of memory is allocated to store the larger list and the currently allocated block is freed.
- All memory is freed when the clock is deleted.

6.1.10.2 USE IN DISTRIBUTED SYSTEMS

In distributed systems (Line cards (LCs) and Timing card (TC) combinations) some things have to be considered:

- PATH TRACE information is stored on each Line Card;
- Other LCs that are a part of the clock may need to issue ANNOUNCE messages with the selected PATH TRACE information.
- The following sequence should be followed in that case:
  - All LCs send BMCA information to the Timing Card
  - The TC selects the best server from the data provided and prepares a ParentDS to load to each LC.
  - Prior to loading the ParentDS to each LC, the TC queries the PATH TRACE List from the selected LC.
  - The TC first loads this PATH TRACE List to each LC (using the command below).
  - The TC then loads the ParentDS to each LC.

6.1.10.3 API INTERFACE

The following commands may be used to manage the PATH TRACE functionality:

TABLE 6-10: PTP CLOCK PATH TRACE ROUTINES

API Routine	Description
zl303xx_PtpClockPathTraceEnable	Enables/Disables the PATH TRACE functionality on a target clock.
zl303xx_PtpClockPathTraceListGet	Retrieves the current PATH TRACE List from a target clock. (Typically used in distributed architectures)
zl303xx_PtpClockPathTraceForExtParent	Sets the active PATH TRACE List for a target clock that has selected an external ParentDS. (Typically used in distributed architectures)

6.1.11 Modify PTP Clock: Synchronization Uncertain Flag Operation

In some applications, a clock may begin to advertise the attributes of its parent data set before its local timing circuitry has achieved an acceptable level of stability (e.g. Default Profile advertises immediately after BMCA selects a server). In some cases, it may be desirable to advertise to downstream nodes that the local clock is unsynchronized and allow custom BMCAs to ignore that server until stability is achieved.

This mechanism is implemented using the optional Synchronization-Uncertain flag (included in the next revision of the IEEE-1588 Standard). In general:

- If the optional functionality is disabled, the flag (at Octet 1:Bit-6 of the ANNOUNCE flagField) is set to 0.
- If the optional functionality is enabled, the flag (in the ANNOUNCE messages) is set based on the following:
  - If the Synchronization-Uncertain flag of the ParentDS is TRUE then the advertised value from the local node will be TRUE.
  - If the state of the ingress Port/Stream is UNCALIBRATED then the advertised value from the local node will be TRUE.
  - If a PTP Profile specifies a performance condition that is not met (e.g. ITU G.8275) then the advertised value from the local node will be TRUE.
  - Otherwise, the advertised value from the local node will be FALSE.

TABLE 6-11: PTP SYNCHRONIZATION-UNCERTAIN FLAG API COMMANDS

API Routine	Description
zl303xx_PtpClockSynchronizationUncertainEn	Enables or Disables the Synchronization-Uncertain flag for a specified Clock.
zl303xx_DebugClockDataSet	Displays the configuration for the specified clock, including the Synchronization-Uncertain flag.

6.1.12 Modify PTP Clock: Slave-Only Operation

Changing the Clock slave-only configuration may be prohibited depending on the clockClass value. For example, if the current clockClass is less than 128, then attempting to set the slaveOnly setting to TRUE would fail.

Use the following API command to set the slave-only flag for valid clockClass configurations:

TABLE 6-12: PTP CLOCK SLAVE-ONLY UPDATE ROUTINE

API Routine	Description
zl303xx_PtpClockSlaveOnlySet	Updates the Clock::slaveOnly of the specified PTP Clock.

6.1.13 Modify PTP Clock: Profiles ITU-T

The following table outlines additional API routines available to manage these configuration parameters:

TABLE 6-13: PTP CLOCK AND PORT API ROUTINES FOR ITU-G.8275.X PARAMETERS

API Routine	Description
zl303xx_PtpG8275ClockStateGet	Evaluates (and returns) the current G.8275.1 clock state (refer to the zl303xx_PtpG8275p1ClockStateE definition).
zl303xx_PtpG8275ClockLocalPrioritySet	Sets the localPriority value of the G.8275.1 clock (used in BMCA selection).
zl303xx_PtpG8275p1ClockClassEvalMethodSet	Sets the classEvalMethod value of the G.8275.1 clock. This determines which G.8275.1 scheme is used to determine advertised clock class (refer to the zl303xx_PtpG8275p1ClassEvalMethodE definition).
zl303xx_PtpG8275p1ClockEquipmentClassSet	Sets the equipmentClass value of the G.8275.1 clock. This value is only advertised when the clock uses METHOD[2] to determine its advertised clock class (refer to the zl303xx_PtpG8275p1ClassEvalMethodE definition).
zl303xx_PtpG8275ClockHoldoverSupportedSet	T-TSC clocks only; determines if the T-TSC uses a holdover clock-class value in BMCA selection or its default 255 class.

6.1.14 Modify PTP Clock: Profile IEEE C37.238

The following table outlines additional API routines available to manage these configuration parameters:

TABLE 6-14: PTP CLOCK API ROUTINES FOR IEEE C37.238 PARAMETERS

API Routine	Description
zl303xx_PtpC37p238GrandmasterIdSet	Updates a clock's grandmasterId parameter.
zl303xx_PtpC37p238GrandmasterIdGet	Retrieves a clock's grandmasterId parameter.
zl303xx_PtpC37p238LocalTimeInaccSet	Sets a clock's Local Time Inaccuracy value.
zl303xx_PtpC37p238LocalTimeInaccGet	Retrieves a clock’s Local Time Inaccuracy value.
zl303xx_PtpC37p238NetworkTimeInaccSet	Sets a clock’s Network Time Inaccuracy value.
zl303xx_PtpC37p238NetworkTimeInaccGet	Retrieves a clock's Network Time Inaccuracy value.

6.1.15 Modify PTP Clock: Timestamp Interface Rate TLV

The Timestamp Interface Rate TLV is used by ITU-T G.8275.2 Profile to correct for path asymmetry due to different bit rates of the transmitting and receiving interfaces.

The following is the general procedure:

- Configure the PTP server with parameters required by the TLV
- Enable sending of the TLV on the server
- Retrieve the learned data on the client
- Set the Asymmetry Correction values on the client to offset the timestamp interface rate difference.

The following API commands are available:

TABLE 6-15: PTP TIMESTAMP INTERFACE RATE TLV API ROUTINES

API Routine	Description
zl303xx_PtpG8275ClockTimestampIfRateTlvEn	Enables or Disables appending the timestamp interface rate TLV on egress GRANT messages.
zl303xx_PtpG8275PortTimestampIfRateSet (zl303xx_PtpG8275TimestampIfRateTlvS)	Sets the characteristics of the timestamp interface TLV to be sent on egress GRANT messages.
zl303xx_PtpG8275StreamTimestampIfRateGet (zl303xx_PtpG8275TimestampIfRateTlvS)	Retrieves the timestamp interface rate characteristics learned on a stream connection.

The following structure is used to configure the TLV fields (`zl303xx_PtpG8275-TimestampIfRateTlvS`):

TABLE 6-16: PTP ZL303XX_PTPG8275TIMESTAMPIFRATETLVS FIELDS

Structure	Parameter	Options/ Type	Description
zl303xx_PtpG8275TimestampIfRateTlvS	interfaceBitPeriod	Uint64S	The period of 1-bit of the transmitting PTP timestamp interface, excluding line encoding. The value is encoded as an unsigned integer in units of attoseconds (10^-18) to accommodate interface bit periods less than 1 ns.
zl303xx_PtpG8275TimestampIfRateTlvS	numberBitsBeforeTimestamp	Uint16T	The length of the packet prior to the timestamp point, in bits.
zl303xx_PtpG8275TimestampIfRateTlvS	numberBitsAfterTimestamp	Uint16T	The length of the packet after the timestamp point, in bits.

6.1.16 Modify PTP Clock: SMPTE Sync Metadata TLV

The Sync Metadata TLV is used by SMPTE 2059-2 and AES-R16 profiles to convey video frame rate and other information related to a broadcast environment.

The following is the general procedure:

- Configure the values in `zl303xx_PtpClockCreateS::profileCfg:: smpte:: syncMetadata`
- Call `zl303xx_PtpClockCreate()` to create the clock with the required parameters
- Modify the data being sent on the server
- Retrieve the learned data on the client

The following API commands are available:

TABLE 6-17: PTP SMPTE SYNC METADATA TLV API ROUTINES

API Routine	Description
zl303xx_PtpSmpteClockConfigSet	Sets the characteristics of the SMPTE Sync Metadata TLV to be sent on egress management messages.
zl303xx_PtpSmpteClockConfigGet	Gets the characteristics of the SMPTE Sync Metadata TLV to be sent on egress management messages.
zl303xx_ PtpSmpteStreamSyncMetadataTlvSend	Sends a SMPTE Sync Metadata TLV on a particular stream connection.
zl303xx_PtpSmpteStreamSyncMetadataGet (zl303xx_PtpTlvSyncMetadataS)	Retrieves the SMPTE Sync Metadata characteristics learned on a stream connection.
zl303xx_PtpStreamSmpteSmTlvCfgSet	Configures whether to append or process SMPTE Sync Metadata TLV on a given stream.

The following structure is used to configure the TLV fields (`zl303xx_PtpTlvSyncMetadataS`):

TABLE 6-18: ZL303XX_PTPTLVSYNCMETADATAS FIELDS

Parameter	Options/Type	Description
defaultSystemFrameRate	Uint64S	Default video frame rate of the slave system as a lowest term rational.
masterLockingStatus	Uint8T	Complementary information to clockClass to convey the current locking status. 0 means unavailable
timeAddressFlags	Uint8T	Indicates the intended SMPTE ST 12-1 flags. Bit 0: Drop frame Bit 1: Color Frame Identification Bits 2-7: Reserved.
currentLocalOffset	Sint32T	Offset in seconds of Local Time from grandmaster PTP time.
jumpSeconds	Sint32T	The size of the next discontinuity, in seconds, of Local Time.
timeOfNextJump	Uint48T	The value of the seconds portion of the grandmaster PTP time at the time that the next jump will occur.
timeOfNextJam	Uint48T	The value of the seconds portion of the PTP time when the next Daily Jam will occur.
timeOfPreviousJam	Uint48T	The value of the seconds portion of the PTP time when the last Daily Jam occurred.
previousJamLocalOffset	Sint32T	The value of currentLocalOffset when the last Daily Jam happened.
daylightSaving	Uint8T	Bit 0: Current Daylight Saving Bit 1: Daylight Saving at next discontinuity Bit 2: Daylight Saving at previous Daily Jam event Bits 3-7: Reserved
leapSecondJump	Uint8T	The reason for the next jump in currentLocalOffset indicated by timeOfNextJump.

6.1.17 Modify PTP Clock: IEEE 802.1as Followup Information TLV Data

The Follow_Up Information TLV is used by IEEE 802.1as profiles to convey GM clock source information and the cumulative rate offset information of the path to the GM. When the received information changes, a PTP_EVENT_FOLLOWUP_INFO_TLV_CHANGE event message is generated locally (see Section 8.1.2.7).

The following API commands are available:

TABLE 6-19: PTP IEEE 802.1AS FOLLOWUP INFORMATION TLV API ROUTINES

API Routine	Description
zl303xx_PtpClock802p1FollowupInfoTlvLocalSet	API to configure local Followup Information TLV data. This data is used when the local clock becomes grandmaster.
zl303xx_PtpClock802p1FollowupInfoTlvLocalGet	API to retrieve local Followup Information TLV data. This data is used when the local clock becomes grandmaster.
zl303xx_PtpClock802p1FollowupInfoTlvGet	API to retrieve Followup Information TLV data as sent in Followup packets originating from this clock. When operating as grandmaster this matches the local configuration. When operating as bridge (BC), this combines data received from upstream grandmaster with the neighbor rate ratio of the BMCA best stream/port.
zl303xx_PtpStream802p1FollowupInfoTlvGet	API to retrieve raw Followup Information TLV data received on a stream (for debugging).

The following structure is used to configure the TLV fields (`zl303xx_Ptp802p1FollowUpInfoTlvS`):

TABLE 6-20: ZL303XX_PTP802P1FOLLOWUPINFOTLVS FIELDS

Parameter	Options/Type	Description
cumulativeScaledRateOffset	Sint32T	Equal to (rateRatio - 1.0) x (2^41) truncated to the next smaller signed integer, where rateRatio is the ratio of the frequency of the grandmaster to the frequency of the LocalClock entity in the time-aware system that sends the message.
gmTimeBaseIndicator	Uint16T	The timeBaseIndicator of the current grandmaster.
lastGmPhaseChange	ScaledNs96T	Value of the clock's lastGmFreqChange multiplied by 2^41 truncated to the next smaller signed integer.
scaledLastGmFreqChange	Sint32T	The fractional frequency offset of the current grandmaster relative to the previous grandmaster, at the time that the current grandmaster became grandmaster, or relative to itself prior to the last change in gmTimeBaseIndicator, multiplied by 2^41 and truncated to the next smaller signed integer.

6.1.18 Modify PTP Clock: Alternate Time Offset Indicator ATOI TLV Data

IEEE 1588-2008 Optional Feature in Section 16.3 specifies feature for propagating alternate timescale information to PTP instances using the Alternate Time Offset Indicator (ATOI) TLV appended to Announce messages sent by a grandmaster and propagated by boundary clocks.

The following APIs can be used on PTP grandmaster to configure the ATOI TLV.

TABLE 6-21: ALTERNATE TIME OFFSET INDICATOR ATOI TLV API ROUTINES

API Routine	Description
zl303xx_PtpClockAltTimeOffsetEntryAdd	Adds (or replaces) the specified entry in the Alternate Time Offset Table of the specified clock.
zl303xx_PtpClockAltTimeOffsetEntryDel	Removes the specified entry from the Alternate Time Offset Table of the specified clock.
zl303xx_PtpClockAltTimeOffsetEntryEnable	Enables/Disables the specified entry in the Alternate Time Offset Table of the specified clock.
zl303xx_PtpClockAltTimeOffsetEntryGet	Retrieves the configured entry in the Alternate Time Offset Table of the specified clock.
zl303xx_PtpClockAltTimeOffsetLearnedGet	Retrieves the learned (received) entry in the Alternate Time Offset Table of the specified clock. Also supports clearing the received data. Used on the boundary clock or client nodes.
ZL303XX_PTP_ALT_TIME_OFFSET_TBL_ENTRIES	Compile-time define for maximum number of supported local ATOI TLV entries configured or learned (default 5). Used as maxKey. This limit does not affect BC forwarding.

The following structure is used to configure the ATOI TLV fields (`zl303xx_PtpTlvAltTimeOffsetS`):

TABLE 6-22: ZL303XX_PTPTLVALTTIMEOFFSETS FIELDS

Parameter	Options/Type	Description
keyField	Uint8T	Key Field for the alternate timescale reported in this TLV entity. Supported values 0 to maxKey (ZL303XX_PTP_ALT_TIME_OFFSET_TBL_ENTRIES)
enabled	zl303xx_BooleanE	Indicates if the alternate timescale is enabled (or valid when learned/received).
currentOffset	Sint32T	Offset of the alternate time, in seconds, from the node's time. The alternate time is the sum of this value and the node's time.
jumpSeconds	Sint32T	The size of the next discontinuity, in seconds, of the alternate time. A value of zero indicates that no discontinuity is expected. A positive value indicates that the discontinuity will cause the currentOffset of the alternate time to increase.
timeOfNextJump	Uint64S	The value of the seconds portion of the transmitting node's time at the time that the next discontinuity will occur. The discontinuity occurs at the start of the second indicated by the value of timeOfNextJump.
displayName	char [10]	The text name of the alternate timescale. Maximum 10 bytes.

6.2 MODIFY PTP PORT CONFIGURATION

6.2.1 Modify PTP Port: Maximum Packet Rate Service Limit

As each PTP Port of a timing server may have multiple clients, it is possible to set a maximum packet rate Service Limit for a port. This prohibits the port from granting too high of a transmit rate even though each individual client is within the stream packet rate limit.

If a lower rate than that of the current active contracts is provisioned, the existing contracts will not be canceled, but rather the new limit will be applied to any renewals. (Refer to the PTP API Reference Manual for complete interface, data type, and syntax details).

TABLE 6-23: PTP PORT PACKET RATE SERVICE LIMIT ROUTINE

API Routine	Description
zl303xx_PtpPortTotalPpsMaxSet	Sets the packet rate service limit for the specified PTP port.
zl303xx_PtpPortTotalPpsGet	Returns the total PPS counters for all unicast negotiated message types (Announce, Sync, and Delay_Resp) on a specified port.

6.2.2 Modify PTP Port: Grant or Deny Unicast Service Requests

The ability to manage which clients and which PTP message types will be granted unicast service is done at the port level. A port's "grant table" can be configured to:

- Deny all requests in general, except for those clients and/or message types that have been enabled via a separate API interface and stored by the port table.
- Grant all requests in general, except for those clients and/or message types that have been disabled via a separate API interface and stored by the Port table.
- Set special Grant and/or Deny rules for specific Client nodes.
- Set special Grant and/or Deny rules for specific message types.
- Set special Grant and/or Deny rules for combinations of Client node and message type.

TABLE 6-24: PTP PORT GRANT TABLE ROUTINES

API Routine	Description
zl303xx_PtpGrantDefaultSet	Sets the default behavior for unicast negotiation requests. If an entry does not exist for the client node, this setting will be used.
zl303xx_PtpGrantConfigStructInit	Initializes a zl303xx_PtpGrantConfigS structure, which is used as a parameter to zl303xx_PtpGrantConfigSet().
zl303xx_PtpGrantConfigSet	Sets a bit-mask of message types that will be granted for a specific client. Requests for message types not set in the bit-mask will be rejected.
zl303xx_PtpGrantConfigGet	Returns the bit-mask of granted message types for a specific client.
zl303xx_PtpGrantConfigDelete	Removes an entry from a port's grant table.
zl303xx_PtpGrantConfigDeleteAll	Removes all entries from a port's grant table.
Zl303xx_PtpPortDenyServiceRequestsSet	Prohibits any stream on the associated port from granting service of any message-Type.

6.2.3 Modify PTP Port: Profiles ITU-T

The following table outlines additional API routines available to manage these configuration parameters:

TABLE 6-25: PTP PORT API ROUTINES FOR ITU-G.8275.X PARAMETERS

API Routine	Description
zl303xx_PtpG8275PortLocalPrioritySet	Sets the localPriority value of the G.8275.1 port (used in BMCA selection).
zl303xx_PtpPortMasterOnlySet	Sets the masterOnly flag of the port (indicating that the port cannot be used as a reference).

6.2.4 Modify PTP Port: Peer-Delay One-Step or Two-Step

Clocks using the Peer-Delay mechanism have a separate configuration option for setting the 1-step or 2-step operation. The available options are:

- 1-Step:
  - DROP: The Peer-Delay_Req is dropped. In this case, it is assumed that the hardware or some other mechanism (FPGA, etc.) will issue the appropriate Peer-Delay_Resp and no further action is need at the software level.
  - CORR: The correctionField of the Peer-Delay_Resp in 1-Step mode should be:
    - correctionField[rx] + (T3 - T2)

    Therefore the software layer seeds the correctionField of the Peer-Delay_Resp with:
    - correctionField[rx] - T2

    and assumes the hardware/firmware layer will add the appropriate T3 value during transmission.
- 2-Step
  - RAW: follows Clause 11.4.3.c.8 of the IEEE-1588-2008 Std.
  - DELTA: follows Clause 11.4.3.c.7 of the IEEE-1588-2008 Std.

By default, TWO-STEP-DELTA operation is set at clock creation unless otherwise changed by setting the <port::pdRespTxMethod> member (refer to the `zl303xx_PortCreateS` and `zl303xx_PtpPdelayRespMethodE` type for further details).

To change the configuration at run-time, refer to the following API routine:

TABLE 6-26: PTP PORT TELECOM PROFILE ATTRIBUTE UPDATE ROUTINES

API Routine	Description
zl303xx_PtpPortPdelayRespHandleMethodSet	Sets the method by which Peer-Delay Response messages are handled by a PTP Port (refer also to the zl303xx_PtpPdelayRespMethodE type).

6.2.5 Modify PTP Port: FAULTY State

A user can specify port behavior in case a fault occurs causing entry into the FAULTY PTP port state.

- Default behavior is to immediately reset the PTP engine and re-start through the INITIALIZING PTP port state
- Optional behavior is to stay in the FAULTY state

6.2.5.1 FORCE A PTP PORT INTO FAULTY STATE

A user can force a PTP port to go into the FAULTY state with the following command

- zlStatusE `zl303xx_PtpPortForceFaultySet` (`zl303xx_PtpPortHandleT` portHandle)

6.2.5.2 FORCE A PTP PORT OUT OF FAULTY STATE

A user can force a PTP port to come out of the FAULTY state with the following command

- zlStatusE `zl303xx_PtpPortForceFaultyClear` (`zl303xx_PtpPortHandleT` portHandle)

6.2.5.3 CHANGE DEFAULT PORT BEHAVIOR IN FAULTY STATE

The default behavior for a port which goes into faulty state is to immediately re-initialize the port and all associated streams. The user can change this behavior by calling the routine

- zlStatusE `zl303xx_ PtpPortFaultPassThroughEnSet` (`zl303xx_PtpPortHandleT` portHandle, `zl303xx_BooleanE` passThroughEn)

The argument `passThroughEn` is used to set the port behavior in case of a fault. The following table describes the possible values for this parameter and the actions which will be taken for a certain value.

TABLE 6-27: POSSIBLE VALUES OF PASSTHROUGHEN AND DESCRIPTION

Value	Description
TRUE	The port will only use faulty as a transition state. The port and all it's streams will be re-initialized. An event and error log will be generated. Fault counter will be updated.
FALSE	The port will stay in faulty state. All of the associated streams will be moved to IDLE state and stay there. An event and error log will be generated. Fault counter will be updated.

6.2.6 Modify PTP Port: IEEE 802.1as Neighbor Prop Delay Threshold

The IEEE 802.1as Neighbor Propagation Delay threshold is configurable. This threshold is used to determine if a PTP port is capable of running the 802.1as profile (i.e. asCapable).

TABLE 6-28: PTP PORT IEEE 802.1AS UPDATE ROUTINES

API Routine	Description
zl303xx_PtpPortNeighborPropDelayThresholdSet	Sets a port's threshold for neighbor propagation delay (nanoseconds)
zl303xx_PtpPortNeighborPropDelayThresholdGet	Gets a port's threshold for neighbor propagation delay (nanoseconds)

6.3 MODIFY PTP STREAM CONFIGURATION

6.3.1 Modify PTP Stream: Override Mode

Generally, a Stream on a PTP Clock will have its state (MASTER, SLAVE, etc.) controlled by the associated clock based on the condition of the overall clock and all other associated ports and streams. However, it is possible to manipulate the state of a PTP Stream by setting a preferred mode of operation using the following command:

TABLE 6-29: API ROUTINES FOR SETTING THE STREAM OVERRIDE MODE

API Routine	Parameter	Type	Description
zl303xx_PtpStreamOverrideModeSet	overrideMode	zl303xx_PtpStreamModeE	Sets (over-rides) the stream mode configuration to one of the following: • USE_BMC: Allows the Clock to change the Stream's State based on external reference selection (Default and Recommended setting). • SLAVE_ONLY: Forces the Stream into SLAVE state (can go to LISTENING or UNCALIBRATED as well depending on reference selection). • MASTER_ONLY: Forces the Stream into MASTER mode (Never PASSIVE).

The following diagram illustrates the BMCA stream states for each of the Stream Override conditions. Each of the shaded areas illustrate that the stream override setting is affecting the final state decision for the stream.

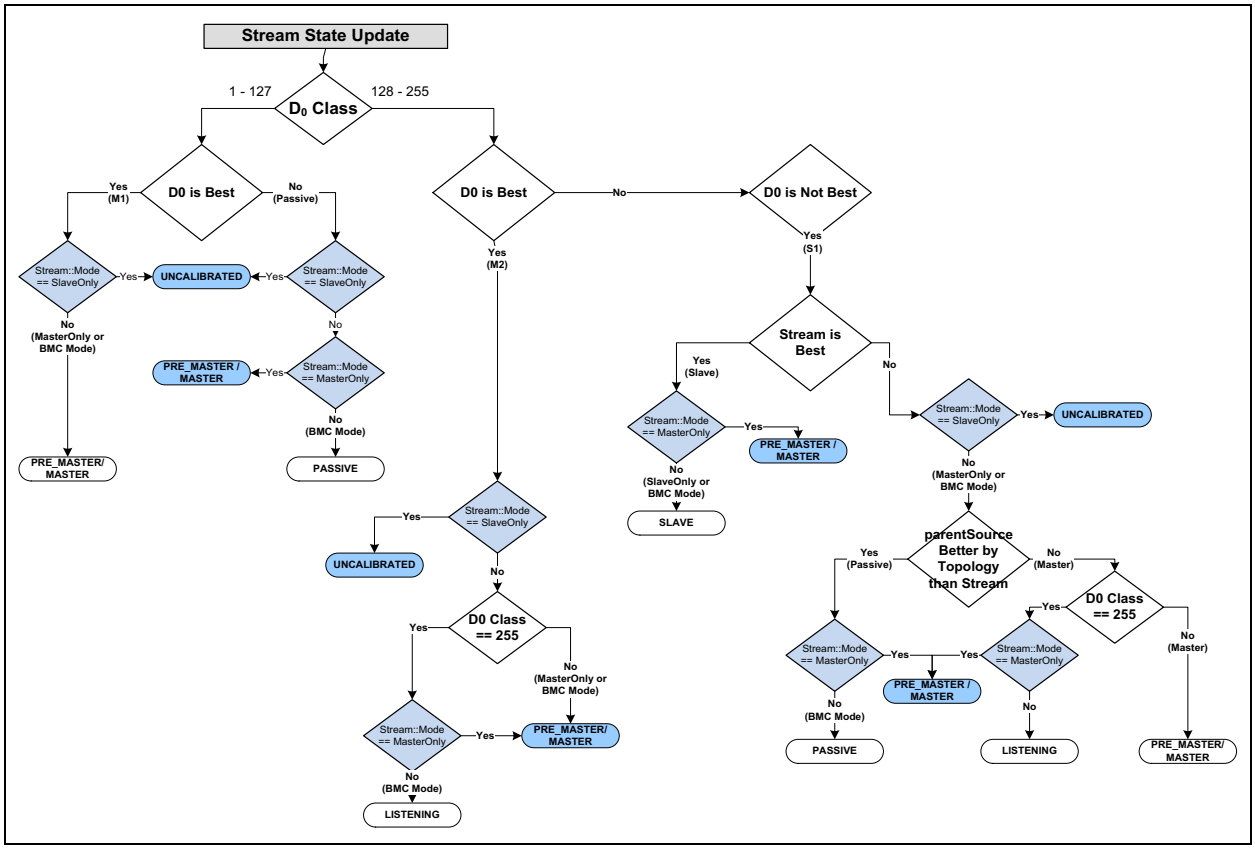

FIGURE 6-1: *Stream Override Affect on State Decision.*

6.3.2 Modify PTP Stream: Message Packet Rates

6.3.2.1 PRE-COMPILE OPTION

The default packet intervals for each of the message types are defined by a group of MACROS in `zl303xx_PtpSetup.h`. By changing the value of these MACROS prior to compiling, the default rate for each message type can be changed so that whenever a stream is created, the new rate will take effect.

TABLE 6-30: ZL303XX_PTPSETUP.H MACROS FOR DEFAULT PTP MESSAGE RATES

MACRO	Default Interval	Packet Rate	Description
ZL303XX_PTP_DEFAULT_STREAM_LOG_ANNC_INTVL	1	1 message per 2 seconds	Default ANNOUNCE Interval
ZL303XX_PTP_DEFAULT_STREAM_LOG_SYNC_INTVL	0	1 message per second	Default SYNC Interval
ZL303XX_PTP_DEFAULT_STREAM_LOG_DELAY_INTVL	0	1 message per second	Default DELAY_REQUEST Interval
ZL303XX_PTP_DEFAULT_STREAM_LOG_PEER_DELAY_INTVL	0	1 message per second	Default PEER_DELAY_REQUEST Interval

Affects both Multicast and Unicast configuration options.

6.3.2.2 STREAM CREATION OPTIONS

When creating streams, it is possible to set the desired packet rates the stream should use.

After issuing the `zl303xx_PtpStreamCreateStructInit()` command, the default values from the `zl303xx_PtpSetup.h` file are used. Before calling `zl303xx_PtpStreamCreate()` to create the new stream, the following structure members may be updated to set the desired rates.

TABLE 6-31: SETTING STREAM CREATION MESSAGE RATES ON NON-NEGOTIATED AND MULTICAST PTP STREAMS

MACRO	Description
::config.logAnnounceInterval	ANNOUNCE Interval
::config.logSyncInterval	SYNC Interval
::config. logMinDelayReqInterval	DELAY_REQUEST Interval
::config. logMinPDelayReqInterval	PEER_DELAY_REQUEST Interval

In the multicast and non-negotiated unicast scenarios, there are no contracts (and therefore no contract limits). However, the master can change "logMinDelayReqInterval" in the Delay_Response messages to inform the slave to change its Delay_Request.

Negotiated clients negotiates message contracts with master for each message type sent by master (Sync, Dresp, and Announce).

6.3.2.3 RUN-TIME OPTIONS

The rate of an active stream can be changed using the `zl303xx_PtpStreamLogMsgIntvlSet()` command. In this case, the user can specify the messageType and new message interval.

For Unicast Negotiated streams, a new request is sent immediately and the existing contract rate is updated once the master accepts the request.

6.3.2.4 ADDITIONAL UNICAST NEGOTIATED MESSAGE OPTIONS

6.3.2.4.1 Slave's Acceptable Rate (Best-effort contract negotiation option)

Because master nodes can only accept or reject contract requests (it does not propose a rate to the slave in its reply), best-effort negotiation can be used by slave to auto-negotiate with the master using slower rates until it reaches its acceptable minimum.

By default, the functionality is disabled.

It may be enabled and configured when the stream is created or during runtime.

At creation-time by modifying the following parameters:

- stream.config.unicastNegotiation.bestEffort[ZL303XX_MSG_ID_ANNOUNCE].enabled
- stream.config.unicastNegotiation.bestEffort[ZL303XX_MSG_ID_ANNOUNCE].intervalMax
- stream.config.unicastNegotiation.bestEffort[ZL303XX_MSG_ID_SYNC].enabled
- stream.config.unicastNegotiation.bestEffort[ZL303XX_MSG_ID_SYNC].intervalMax
- stream.config.unicastNegotiation.bestEffort[ZL303XX_MSG_ID_DELAY_RESP].enabled
- stream.config.unicastNegotiation.bestEffort[ZL303XX_MSG_ID_DELAY_RESP].intervalMax

At run-time by calling the following routines:

- `zl303xx_PtpStreamBestEffortEn()`
- `zl303xx_PtpStreamBestEffortIntervalSet()`

6.3.2.4.2 Master Contract Limits

On the unicast negotiated master, streams are not created until requested by the slave and granted by the master. It is possible to configure the master to limit the contract parameters for each new request.

1. Port, Per-Contract MessageType Limits: Allows a Master Port to limit the individual contract request parameters.

- Max packet rate per contract messageType
- Used to control maximum packet rate per-client per message type.
- Creation-time configuration:
  - port.config.unicastNegotiation.maxAnnounceRate
  - port.config.unicastNegotiation.maxSyncRate
  - port.config.unicastNegotiation.maxDelayRespRate
- Run-time:
  - `zl303xx_PtpPortContractIntervalMinSet()`

2. Port, Total (sum) packet per second limit for all clients on the port (per message type):

- Maximum packet rate for all combined contracts on the port.
- Used to control the maximum throughput of the port. May affect the number of clients supported (e.g. 2048 pps max sync = 32 clients at 64 sync/sec)
- Creation-time configuration:
  - port.config.unicastNegotiation.totalPpsMax[ZL303XX_MSG_ID_*]
- Run-time:
  - `zl303xx_PtpPortTotalPpsMaxSet()`

If these limits are changed at run-time, the change will not affect existing, granted contracts for the duration that was negotiated. Once renewed however, the new limits will take effect.

6.3.3 Modify PTP Stream: UNCALIBRATED State Operation

When a Port is selected as the reference source, it will transition through the UNCALIBRATED state prior to entering the SLAVE state. Prior to Release 4.6.0 this transition was not implemented, making the UNCALIBRATED period effectively zero (0).

The UNCALIBRATED phase can now be enabled in one of 2 ways (or both):

- Configuring the UNCALIBRATED timer: determines the number of Announce Intervals that a Port will remain in the UNCALIBRATED state before transitioning to the SLAVE state. This waits the required number of intervals regardless of whether Announce messages are actually received.
- Forcing the Port to wait for an external indication that a desired level of performance has been achieved (i.e. LOCKED) before transitioning to the SLAVE state.

If both a timer and external LOCK condition are configured then both conditions must be met before transitioning to the SLAVE state. The following table lists the API commands provided:

TABLE 6-32: API ROUTINES FOR MANAGING THE UNCALIBRATED PORT STATE

API Routine	Description
zl303xx_PtpStreamUncalibratedIntervalSet	Sets the number of ANNOUNCE intervals that a Port will remain in the UNCALIBRATED state before transitioning to the SLAVE state.
zl303xx_PtpStreamUncalibratedLockRequiredSet	API to set whether a user define level of performance is required prior to transitioning to the SLAVE state. This is an external user implemented routine so that the level of acceptable performance is determined by the user application (refer to ZL303XX_PTP_HW_CMD_LOCK_STATUS_GET definition for details.

6.3.4 Modify PTP Stream: Unicast Negotiation Packet Rate

The following API routines are used to change a PTP Stream's Packet Rates for each message type of a unicast negotiated connection.

The new rate is immediately re-negotiated regardless of the remaining duration of any existing contract. Refer to the PTP API Reference Manual for complete interface, data type, and syntax details.

TABLE 6-33: PTP STREAM PACKET RATE RESET ROUTINES

API Routine	Description
zl303xx_PtpStreamLogSyncIntvlSet	Resets the SYNC message rate of the specified PTP Stream.
zl303xx_PtpStreamLogDelayIntvlSet	Resets the DELAY_REQUEST message rate of the specified PTP Stream.
zl303xx_PtpStreamLogAnnounceIntvlSet	Resets the ANNOUNCE message rate of the specified PTP Stream.

6.3.5 Modify PTP Stream: Unicast Negotiation Contract Duration

The following API routine is used to change a PTP Stream's Negotiated Contract Duration.

The new value applies to all message types and will not be applied until the next contract renewal. (Refer to the PTP API Reference Manual for complete interface, data type, and syntax details).

TABLE 6-34: PTP STREAM NEGOTIATED CONTRACT

API Routine	Description
zl303xx_PtpStreamUniNegDurationSet	Resets the contract duration of the specified PTP Stream.

6.3.6 Modify PTP Stream: Asymmetry Correction

Clause 11.6 of the IEEE-1588-2008 Standard specifies the following asymmetry corrections that may be made to PTP V2 event messages:

TABLE 6-35: APPLICATION OF ASYMMETRY CORRECTION VALUES

messageType	Transmit Correction	Receive Correction
Sync	no correction (Per 11.6.2.a of the Standard)	Add the value of the ingress path delayAsymmetry to the correctionField (Per 11.6.2.b of the Standard)
Delay Request	Subtract the value of the egress path delayAsymmetry from the correctionField (Per 11.6.3.b of the Standard)	no correction (Per 11.6.3.a of the Standard)
Peer Delay Request	Subtract the value of the egress path delayAsymmetry from the correctionField (Per 11.6.4.b of the Standard)	no correction (Per 11.6.4.a of the Standard)
Peer Delay Response	no correction (Per 11.6.5.a of the Standard)	Add the value of the ingress path delayAsymmetry to the correctionField (Per 11.6.5.b of the Standard)

The following diagram illustrates the implications of these corrections in a typical OC client-server system:

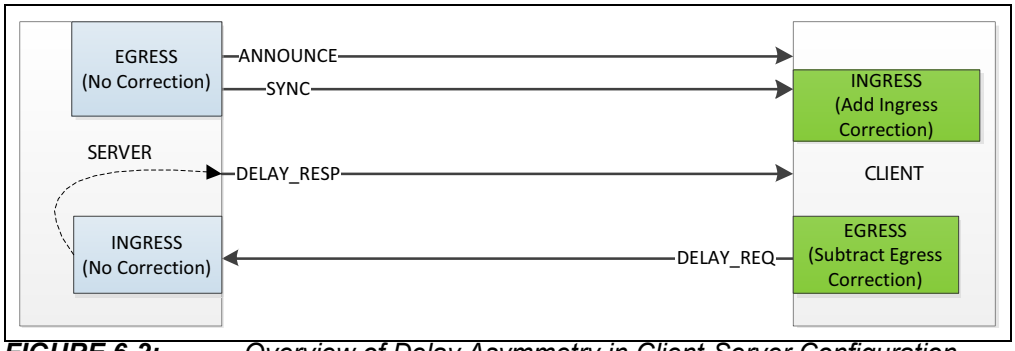

FIGURE 6-2: *Overview of Delay Asymmetry in Client-Server Configuration.*

Configuring the Asymmetry Correction values for a stream may be done in two ways:

1. Setting default values to be applied during stream creation by editing the following #define values in the `zl303xx_PtpSetup.h` module:

- ZL303XX_PTP_DEFAULT_STREAM_DELAY_ASYM_INGRESS
- ZL303XX_PTP_DEFAULT_STREAM_DELAY_ASYM_EGRESS

2. Using either of the Run-Time API calls to modify the correction values dynamically:

- Set one or both using the zl303xx_TimeInterval structure:

```
zlStatusE zl303xx_PtpStreamDelayAsymmetrySet(
      zl303xx_PtpStreamHandleT streamHandle,
      zl303xx_TimeInterval     *ingressDelay,
      zl303xx_TimeInterval     *egressDelay);
```

Where:

streamHandle:	Handle to an existing stream.
ingressDelay:	The ingress delay asymmetry value of type zl303xx_TimeInterval (nSec * (2^16)). If NULL, the current value is maintained.
egressDelay:	The egress delay asymmetry value of type zl303xx_TimeInterval (nSec * (2^16)). If NULL, the current value is maintained.

- Set a single direction using only a nanoseconds value:

```
zlStatusE zl303xx_PtpStreamDelayAsymmetryNsSet(
      zl303xx_PtpStreamHandleT streamHandle,
      zl303xx_BooleanE         isIngress,
      Sint32T                  delayNs);
```

Where:

streamHandle:	Handle to an existing stream.
isIngress:	Boolean indicating the direction on which to apply the correction. • TRUE => ingress • FALSE => egress
delayNs:	The delay, in nanoseconds, to apply to the direction specified.

Referring back to the diagram in Figure 6-2, a positive value at the Egress has the same effect as the same positive value at Ingress (this is because the Egress value is subtracted instead of added)

If the server and client are phase aligned, the following Figure 6-3 illustrates how setting correction values will affect the client signal relative to the server signal on an oscilloscope:

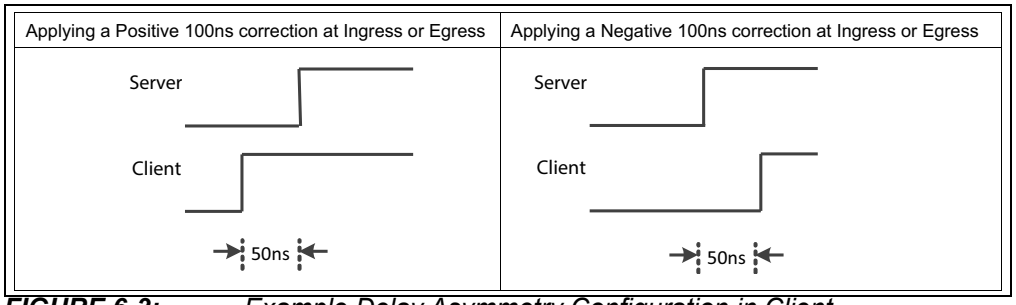

FIGURE 6-3: *Example Delay Asymmetry Configuration in Client.*

6.3.7 Modify PTP Stream: Override Ingress and Egress clockClass

Note: a) It is recommended when overriding the Egress `clockClass` value of a stream to refer to Section 6.3.8 instead of using the egress routine provided in this section.
b) It is fine to use the Ingress override routine provided here.

This allows a system to:

- Override the `clockClass` contained in a received PTP message on a per stream basis.
- Override the `clockClass` transmitted on a per stream basis.

The stream override values can be set when creating a stream. By default, no override is configured to maintain backward compatibility. Refer to the API User's Guide for parameter and usage details of the routines listed below.

TABLE 6-36: PTP STREAM CLOCKCLASS OVERRIDE ROUTINES

API Routine	Description
zl303xx_PtpStreamIngressClockClassSet	Sets the ingress override clock class value for the specified stream.
zl303xx_PtpStreamEgressClockClassSet	Sets the egress override clock class value for the specified stream.

6.3.8 Modify PTP Stream: Override Egress ANNOUNCE Message Fields

The values that a clock advertises in transmitted ANNOUNCE messages are generally determined by the IEEE-1588-2008 Standard and the Profile that is in operation. The current software allows the user to configure over-ride values that will be sent instead of the expected value (on a per-Stream basis).

The values advertised in ANNOUNCE messages are applied in the following order:

1. Values determined by the IEEE-1588-2008 Standard
2. Values applicable for the operating profile replace the values from step (1).
3. Any configured stream override values replace the values from steps (1) & (2) above.

WARNING

The software DOES NOT verify override parameters. It is expected that if an override value is configured, the user is aware of its implications.

The list of advertised ANNOUNCE values that may be over-written is defined in the enumeration, `zl303xx_PtpStreamOverrideAnncEgressE`.

The API routines used to manage this functionality is listed in the following table:

TABLE 6-37: API ROUTINES FOR MANAGING THE PER-STREAM ANNOUNCE OVERRIDE VALUES

API Routine	Description
zl303xx_PtpStreamEgressOverrideAnncFieldSet	API to set individual egress ANNOUNCE fields. Parameters are: • streamHandle • parameterId (refer to the enum definition from above). • Void * (pointer to the over-ride data value).
zl303xx_PtpStreamEgressAnncDataGet	Data fills an ANNOUNCE message structure with the values that will be advertised for this stream.

6.3.9 Modify PTP Stream: Maximum clockClass to Qualify

If the `clockClass` received by a stream is above the configured level, then the server will be disqualified. Additionally, the PTSF-QLTooLow flag will be raised. When this is configured to '0', this extra qualification action is not performed.

TABLE 6-38: PTP STREAM MAXIMUM CLOCKCLASS ROUTINES

API Routine	Description
zl303xx_PtpStreamMaxClockClassSet	Sets the maximum received clockClass at which this stream will be considered qualified.

6.3.10 Modify PTP Stream: IEEE 802.1as Message Interval Request TLV

An IEEE 802.1as Message Interval Request TLV can be sent on a PTP stream to the remote peer on the other end of the link using functions in table below.

TABLE 6-39: PTP STREAM IEEE 802.1AS MESSAGE INTERVAL REQUEST TLV ROUTINES

API Routine	Description
zl303xx_PtpStream802p1MsgIntvlReqTlvSend	Sends a Message Interval Request TLV to the remote peer on a specified stream.
zl303xx_PtpStream802p1MsgIntvlReqTlvGetTotal	Debug utility to diagnose total (cumulative) received or sent Message Interval Request TLV states.

Chapter 7. Dynamic Operation

7.1 HANDLING LEAP SECOND EVENTS

For detailed information on handling leap seconds events in the ZL30390 software, refer to the application note:

- ZLAN-525: Handling UTC Leap Second Events in IEEE 1588 Deployments

7.1.1 Available APIs

The following APIs and steps may be taken to handle leap seconds events in Release 4.x.x and above.

TABLE 7-1: API CALLS FOR LEAP SECONDS AND UTC MANAGEMENT

API Function	Description
zl303xx_PtpClockUtcGet()	Get the current UTC time. Uses `currentUtcOffset` from the active `timePropertiesDS`. Use `zl303xx_PtpClockTimePropertiesDSGet()` to check `currentUtcOffsetValid`.
zl303xx_PtpTimePropertiesGetLocal()	Get the local timePropertiesDS.
zl303xx_PtpTimePropertiesSetLocal()	Set the local timePropertiesDS.
zl303xx_PtpTimePropertiesDefaultParams()	Initialize given timePropertiesDS to hard-coded default values.
zl303xx_PtpClockTimePropertiesDSGet()	Get the active timePropertiesDS (connected grandmaster's DS or local DS when disconnected)

7.1.2 Grandmaster

The following describes the sequence of steps a grandmaster should perform to update the `timePropertiesDS` on the day of a UTC leap second event.

1. Within the 12 hours prior to midnight (UTC) on the day of the leap seconds event, use the `zl303xx_PtpTimePropertiesGetLocal()` and `zl303xx_PtpTimePropertiesSetLocal()` functions to update the grandmaster local `timePropertiesDS.leap59` or `leap61` flags depending on the type of event.

   This change immediately triggers a BMCA update and the new `timePropertiesDS` will be advertised to clients in all subsequent ANNOUNCE messages.

   [v4.6.0+ only] The PTP event `ZL303XX_PTP_EVENT_LEAP_SECONDS_FLAG_CHANGE` will be generated at the grandmaster.
2. Configure a system timer to execute the next step at midnight (UTC).
3. At midnight (UTC), update the local time properties of the grandmaster using the `zl303xx_PtpTimePropertiesGetLocal()` and `zl303xx_PtpTimePropertiesSetLocal()` command with the following values:

- `leap59 = FALSE`
- `leap61 = FALSE`
- `currentUtcOffset` ±1 (based on previous leap seconds flags)
- all other values unchanged

This change immediately triggers a BMCA update and the new timePropertiesDS will be advertised to clients in all subsequent ANNOUNCE messages. It is recommended to store these updated properties to non-volatile memory for next reboot.

[v4.6.0+ only] The PTP events ZL303XX_PTP_EVENT_LEAP_SECONDS_FLAG_CHANGE and ZL303XX_PTP_EVENT_UTC_OFFSET_CHANGE will be generated at the grandmaster for confirmation purposes (no further action required).

7.1.3 Client (including Boundary Clocks)

The following describes the sequence of steps a client performs automatically to receive the updated timePropertiesDS on the day of a UTC leap second event.

1. Up to 12 hours before the UTC leap second event the client receives the grandmaster's ANNOUNCE messages with the timePropertiesDS.leap59 or leap61 flags set specifying the type of event.

   [v4.6.0+ only] The PTP event ZL303XX_PTP_EVENT_LEAP_SECONDS_FLAG_CHANGE will be generated at the client for confirmation purposes (no further action required).
2. Optional. Configure a system timer to execute the next step at midnight (UTC) to safeguard against the possibility that the client loses connectivity to the grandmaster and accurate UTC time is still required.
3. At midnight (UTC), or a few seconds after, the client will receive an ANNOUNCE message from the grandmaster with the new timePropertiesDS.currentUtcOffset value and the leap59 and leap61 flags cleared.

   Internally, the PTP clock's active dataset is updated with the grandmaster's timePropertiesDS and the zl303xx_PtpClockUtcGet() function reports accurate, leap-second adjusted UTC time.

   [v4.6.0+ only] The PTP events ZL303XX_PTP_EVENT_LEAP_SECONDS_FLAG_CHANGE and ZL303XX_PTP_EVENT_UTC_OFFSET_CHANGE will be generated at the client for confirmation purposes (no further action required).

Note: Although the dynamic or active timePropertiesDS has been updated, the local timePropertiesDS values have not changed. Therefore, if connection to the grandmaster is lost, the client will revert to its own local data which may be incorrect. It is recommended to update the local time properties with zl303xx_PtpTimePropertiesSetLocal() and storing these properties to non-volatile memory for next reboot.

Chapter 8. Reporting

8.1 EVENT INTERFACE

8.1.1 Event Handler

The PTP application generates events by calling a function binding that can be optionally set by a user application. The user function is bound to `zl303xx_PtpClockCreateS::eventCallback`, which is configured when calling the `zl303xx_PtpClockCreate()` routine. This eventCallback function will be called for every event generated on the PTP clock. Inside the event handler, a switch() statement should be used to handle only the desired events.

The prototype for the event handler function is defined as:

```
void (* zl303xx_PtpEventFnT)(zl303xx_PtpEventE event, void
*pEventData)
```

For every event generated, the event handler is passed the event type and a data structure with additional information about that event. The data structure is passed as a (void *) and must be cast to the proper type.

Note that the PTP event handler runs within the PTP Clock task context with the PTP Clock mutex held (taken). User event handler implementations should be careful to avoid below:

- Do not call PTP APIs that take the PTP clock mutex inside an event handler. This would cause a double mutex take request which may not be supported by the OS (reentrant mutex support would be required). Refer to examples for APIs that are usable from within event handlers (e.g. BMCA event calling `zl303xx_PtpClockApplyCompareData`).
- Do not block the event handler for long time periods. Generally should not block for more than the fastest packet rate being processed in the system (e.g. 64 pps max block time of 1/64 sec is approximately 16 milliseconds).

To satisfy above requirements it is recommended to process PTP events asynchronously in a separate custom task using a non-blocking message queue implementation.

8.1.2 Event Details

Refer to Table 8-1 for information about how/when each event is generated and which data type the supplied pointer should be cast.

For additional information about the members of each event structure, refer to the ZLS30390 API Reference Manual.

An example event handler function, called `examplePtpEventHandler()`, that illustrates the required interface details can be found in module `zlUserExamples/src/zl303xx_ExamplePtp.c`.

8.1.2.1 PTP EVENT NOTIFICATIONS (GENERAL)

TABLE 8-1: PTP EVENT NOTIFICATIONS (GENERAL AND MISCELLANEOUS)

EVENT ID and Data Structure	Description
ZL303XX_PTP_EVENT_MULTIPLE_PEER_RESP (zl303xx_PtpEventMultiplePeerRespS)	Occurs when multiple PEER_DELAY_RESP messages are received to a single PEER_DELAY_REQ message.
ZL303XX_PTP_EVENT_UNKNOWN_TLV (zl303xx_PtpEventUnknownTlvS)	Occurs when a TLV is received that is non-standard and for which no customTLV handler has been configured. This can be for a non-standard tlvType, managementId or unhandled ORGANIZATION_EXTENSION tlvType.
ZL303XX_PTP_EVENT_RX_STEPS_REMOVED_EXCEEDED (zl303xx_PtpEventRxStepsRemovedExceededS)	Occurs when the local node receives an ANNOUNCE message in which the stepsRemoved field exceeds the local stepsRemoved limit configured.
ZL303XX_PTP_EVENT_PORT_FAULTY (zl303xx_PtpEventPortFaultS)	Occurs when a port goes into faulty state.

8.1.2.2 PTP EVENT NOTIFICATIONS (TIME SYNC ALGORITHM AND TIME OF DAY)

TABLE 8-2: PTP EVENT NOTIFICATIONS (TIME SYNC ALGORITHM AND TIME OF DAY)

EVENT ID and Data Structure	Description
ZL303XX_PTP_EVENT_SERVO_DATA (zl303xx_PtpEventServoDataS)	A transmit/receive time stamp pair is available to be processed by a clock recovery algorithm.
ZL303XX_PTP_EVENT_MSG_INTVL_CHANGE (zl303xx_PtpEventMsgIntvlChangeS)	A message interval change has been detected. This is used to change the parameters of a clock recovery algorithm.
ZL303XX_PTP_EVENT_LEAP_SECONDS_FLAG_CHANGE (zl303xx_PtpEventLeapSecondsFlagChangeS)	Occurs when there is a change in either of the leap51 or leap61 flags of the clock's dynamic Time Properties Data Set.
ZL303XX_PTP_EVENT_UTC_OFFSET_CHANGE (zl303xx_PtpEventUtcOffsetChangeS)	Occurs when there is a change in UTC Offset value of the clock's dynamic Time Properties Data Set.

8.1.2.3 PTP EVENT NOTIFICATIONS (BMCA AND PARENT UPDATES)

TABLE 8-3: PTP EVENT NOTIFICATIONS (BMCA AND PARENT UPDATES)

EVENT ID and Data Structure	Description
ZL303XX_PTP_EVENT_STREAM_STATE_CHANGE (zl303xx_PtpEventStreamStateChangeS)	Occurs when the operating state of a stream changes as a result of a ParentDS update on the associated Clock.
ZL303XX_PTP_EVENT_STREAM_KEEP_ALIVE (zl303xx_PtpEventStreamKeepAliveS)	Occurs when no Stream related Event has been issued for the stream for an extended period of time.
ZL303XX_PTP_EVENT_CLOCK_BMCA_UPDATE (zl303xx_PtpEventClockBmcaUpdateS)	Occurs when there is a change in a clock's ordered list of best servers (evaluated by BMCA) or at periodic intervals (2 seconds).
ZL303XX_PTP_EVENT_COUNTER_ALARM (zl303xx_PtpEventCounterAlarmS)	An alarm generated when the rate of received messages drops below 50% of the expected rate.
ZL303XX_PTP_EVENT_SQUELCH (zl303xx_PtpEventSquelchS)	Will only be generated when using ZL303XX_PTP_PROFILE_TELECOM. Occurs when all grandmasters are in failure conditions.
ZL303XX_PTP_EVENT_PARENT_DATA_SET_CHANGE (zl303xx_PtpEventParentDsChangeS)	A message indicating that the Parent Data Set (ParentDS) of the PTP Clock generating the event has been updated. (This update might have also triggered ZL303XX_PTP_EVENT_STREAM_STATE_CHANGE events). By default this event fires whenever ParentDS is updated regardless of whether data changed or not. See zl303xx_PtpClockCreateS::fireParentDsChangeOnlyDelta to modify this behavior.

8.1.2.4 PTP EVENT NOTIFICATIONS (CREATE/DELETE)

TABLE 8-4: PTP EVENT NOTIFICATIONS (CREATE/DELETE)

EVENT ID and Data Structure	Description
ZL303XX_PTP_EVENT_CLOCK_CREATE (zl303xx_PtpEventClockCreateS)	Occurs when a PTP clock is created.
ZL303XX_PTP_EVENT_CLOCK_DELETE (zl303xx_PtpEventClockDeleteS)	Occurs when a PTP clock is automatically deleted. When zl303xx_PtpShutdown() is called, this event will be generated for each existing clock. This event is not triggered when calling zl303xx_PtpClockDelete() directly by default (see fireEvent option).
ZL303XX_PTP_EVENT_PORT_CREATE (zl303xx_PtpEventPortCreateS)	Occurs when a PTP port is created, see zl303xx_PtpPortCreateS.fireEvent or zl303xx_PtpVirtualPortConfigS.fireCreateEvent. Not triggered by default.
ZL303XX_PTP_EVENT_PORT_DELETE (zl303xx_PtpEventPortDeleteS)	Occurs when a PTP port is automatically deleted. When zl303xx_PtpClockDelete() (or zl303xx_PtpShutdown()) is called, this event will be generated for every port associated with the clock. Calling zl303xx_PtpPortDelete() directly will not trigger this event by default (see fireEvent option).
ZL303XX_PTP_EVENT_STREAM_CREATE (zl303xx_PtpEventStreamCreateS)	Occurs when a PTP stream is automatically created. This happens when a unicast negotiation server receives a valid transmission request signaling message from a client node. Calling zl303xx_PtpStreamCreate() directly will not trigger this event by default (see createFlags option).
ZL303XX_PTP_EVENT_STREAM_DELETE (zl303xx_PtpEventStreamDeleteS)	Occurs when a PTP stream is automatically deleted. This can happen when a clock or port is deleted when streams are still associated with it. Directly calling zl303xx_PtpStreamDelete() will not trigger this event by default (see fireEvent option).

8.1.2.5 PTP EVENT NOTIFICATIONS (UNICAST NEGOTIATION)

TABLE 8-5: PTP EVENT NOTIFICATIONS (UNICAST NEGOTIATION)

EVENT ID and Data Structure	Description
ZL303XX_PTP_EVENT_CONTRACT_REJECTED (zl303xx_PtpEventContractRejectedS)	Occurs when a clock is configured with ZL303XX_PTP_PROFILE_TELECOM, and a slave stream is not granted the requested contract (rate or duration) for a message type.
ZL303XX_PTP_EVENT_CONTRACT_GRANTED (zl303xx_PtpEventContractGrantedS)	Occurs when a unicast negotiation contract is granted.
ZL303XX_PTP_EVENT_CONTRACT_EXPIRED (zl303xx_PtpEventContractExpiredS)	Occurs when a unicast negotiation contract expires (does not get renewed).

8.1.2.6 PTP EVENT NOTIFICATIONS (FAULTY STATE)

If at any time the PTP engine determines a port has gone into FAULTY state it will issue a ZL303XX_PTP_EVENT_PORT_FAULTY Event Message to inform the application. This message uses the `zl303xx_PtpEventPortFaultS` structure.

TABLE 8-6: ZL303XX_PTPEVENTPORTFAULTS STRUCTURE FOR ZL303XX_PTP_EVENT_PORT_FAULTY EVENT

zl303xx_PtpEventPortFaultS	Type	Description
portHandle	zl303xx_PtpPortHandleT	Handle to the port which went into faulty state.
portExtData	void *	Pointer to the port's external data (which may contain a mapping to the socket this port is bound to).
portFaultType	zl303xx_PtpPortFaultTypeE	The fault type which occurred on the port. Refer to Section 8.2.1 for a description of all the fault types.
txError	zl303xx _PtpTxStatusE	The transmit error which caused the port to go into faulty. Refer to Section 8.2.1.3 for a description of all transmit errors.

A PTP port can go into faulty state for the following reasons:

- Unable to transmit one of the following packet types as there is no transmit function bound to `zl303xx_PtpPortCreateS::txMsgFn` (See Table 5-14 and Section 10.5.3.1).
  - Announce Packet
  - Sync Packet
  - Follow_Up Packet
  - Delay_Req Packet
  - Delay_Resp Packet
  - Pdelay_Req Packet
  - Pdelay_Resp Packet
  - Signaling Packet
  - Management Packet

Additionally a PTP port can go into faulty state if strict transmit checks are enabled. These checks can be enabled at compile time. Please refer to ZLS303XX_Architecture_Porting_And_Integration_Guide for a list of compile time options for port faulty state. Below is a list of all the optional reasons a port can go into faulty state.

- If internal stream state is out of defined range (e.g. memory corruption).
- If there is a NULL pointer.
- Network Protocol is not IPv4, IPv6 or IEEE 802.3.
- Message to be transmitted is too long.
- There is a problem sending data on the socket.

8.1.2.7 PTP EVENT NOTIFICATIONS (802.1AS PROFILE)

When PTP engine detects a change in the Followup Information TLV from the best grandmaster it will issue a ZL303XX_PTP_EVENT_FOLLOWUP_INFO_TLV_CHANGE event message to inform the application. This message uses the `zl303xx_PtpEventFollowupInfoTlvChangeS` structure. This conforms to the IEEE 802.1as-2011 Section 9.6 “ClockTargetPhaseDiscontinuity” interface requirements. Note this event does not fire from changes in the dynamic cumulativeScaledRateOffset field.

TABLE 8-7: ZL303XX_PTPEVENTFOLLOWUPINFOTLVCHANGES STRUCTURE FOR ZL303XX_PTP_EVENT_FOLLOWUP_INFO_TLV_CHANGE EVENT

zl303xx_PtpEventFollowupInfoTlvChangeS	Type	Description
streamHandle	zl303xx_PtpStreamHandleT	Handle to the best stream which received Followup Info TLV triggering the change, or -1 if best is local clock.
gmIdentity	zl303xx_ClockIdentity	ClockIdentity of the current grandmaster, or all 0 if best is local clock.
followupInfoTlvOld	zl303xx_Ptp802p1FollowupInfoTlvS	Old Followup Information TLV data (previous best ingress).
followupInfoTlvNew	zl303xx_Ptp802p1FollowupInfoTlvS	New Followup Informaton TLV data (current best ingress).

8.2 POLLING STATISTICS AND COUNTERS

8.2.1 PTP Port State Reporting

The PTP port state when the port is active is dynamically computed based on the state of PTP streams attached to the port.

In some cases the port state may be FAULTY as described in next sections.

8.2.1.1 PTP PORT STATE FAULTY GET LAST FAULT TYPE

The following command can be used to poll the PTP engine for the last fault type which occurred on a specific port

- zlStatusE `zl303xx_PtpPortGetLastFaultType` (`zl303xx_PtpPortHandleT` portHandle, `zl303xx_PtpPortFaultTypeE` *lastFaultType)

The returned lastFaultType is a `zl303xx_PtpPortFaultTypeE` with the following fault types.

TABLE 8-8: ZL303XX_PTPPORTFAULTTYPEE FAULT TYPES AND DESCRIPTION

zl303xx_PtpPortFaultTypeE Types	Description
ZL303XX_PTP_PORT_FAULT_ANNC_SEND	There was a fault while sending an announce message.
ZL303XX_PTP_PORT_FAULT_SYNC_SEND	There was a fault while sending a sync message.
ZL303XX_PTP_PORT_FAULT_FOLLOW_UP_SEND	There was a fault while sending a follow up message.
ZL303XX_PTP_PORT_FAULT_DELAY_REQ_SEND	There was a fault while sending a delay request message.
ZL303XX_PTP_PORT_FAULT_DELAY_RESP_SEND	There was a fault while sending a delay response message.
ZL303XX_PTP_PORT_FAULT_PDELAY_REQ_SEND	There was a fault while sending a peer delay message.
ZL303XX_PTP_PORT_FAULT_PDELAY_RESP_SEND	There was a fault while sending a peer delay response message.
ZL303XX_PTP_PORT_FAULT_SIGNALING_SEND	There was a fault while sending a signaling message.
ZL303XX_PTP_PORT_FAULT_MGMT_SEND	There was a fault while sending a management message.
ZL303XX_PTP_PORT_FAULT_USER_ENFORCED	User forced the port to go into faulty.

8.2.1.2 PTP PORT STATE FAULTY GET FAULT COUNTER

Every time a port goes into faulty state it updates the fault counter associated with it. The following routine can be used to get the counter.

- zlStatusE `zl303xx_PtpPortFaultCounterGet` (`zl303xx_PtpPortHandleT` portHandle, Uint32T *pFaultCounter)

8.2.1.3 PTP PORT STATE FAULTY GET LAST TRANSMIT ERROR

The following command can be used to get the last transmit error which occurred on a specific port

- zlStatusE `zl303xx_PtpPortGetLastTxError`(`zl303xx_PtpPortHandleT` portHandle, `zl303xx_PtpTxStatusE` *pLastTxError)

The returned pLastTxError is a `zl303xx _PtpTxStatusE` type with the following default transmit status types.

TABLE 8-9: ZL303XX _PTPTXSTATUSE DEFAULT TRANSMIT STATUS TYPES AND DESCRIPTION

zl303xx _PtpTxStatusE Default Types	Description
ZL303XX_PTP_TX_OK	There was no transmit error.
ZL303XX_PTP_TX_USER_ENFORCED_ERROR	User forced the port to go into a faulty state resulting in a transmit error.
ZL303XX_TX_MSG_BINDING_NULL	There is no transmit function (see 10.5.3.1) bound to the stack.

Users can turn on optional checks on egress packet path for debugging purposes. These checks might affect peak packet rates so they are off by default. Please refer to ZLS303XX_Architecture_Porting_And_Integration_Guide for a list of compile time options to turn on these checks. Below is a description of all transmit status types which can be returned if extra compile time flags are used.

TABLE 8-10: ZL303XX _PTPTXSTATUSE OPTIONAL TRANSMIT STATUS TYPES AND DESCRIPTION

zl303xx _PtpTxStatusE Optional Types	Description
ZL303XX_PTP_TX_MSG_BUFFER_DATA_NULL	The transmit buffer passed to transmit routine is null.
ZL303XX_PTP_TX_PORT_NULL	Port data passed to transmit routine is null.
ZL303XX_PTP_TX_MSG_TOO_BIG	Length of the message to be transmitted is more than allowed.
ZL303XX_PTP_TX_STREAM_STATE_OUT_OF_RANGE	The transmit stream is not in a valid state.
ZL303XX_PTP_TX_SOCKET_UNSUPPORTED_OPERATION	Destination address of the packet is neither of IPv4, IPv6 or IEEE 802.3 type.
ZL303XX_PTP_TX_SOCKET_ERROR	There was a problem connecting to the socket.

8.2.2 PTP Stream State Reporting

PTP stream states can be obtained with the APIs described in Table 8-11.

TABLE 8-11: API CALLS FOR PTP STREAM STATE REPORTING

API Function	Description
zl303xx_PtpStreamStateGet	Returns PTP stream “operating” state (zl303xx_PtpStreamOperatingStateE). It is provided for information/debugging purposes. The operating state is an internal representation of the behavior of the stream and is not related to IEEE 1588-2008 “portState” (Table 10).
zl303xx_PtpStreamSubStateHistoryGet	Returns PTP stream “sub-state” (zl303xx_PtpStreamSubStateE) current and historical values including state transition reasons, time, and trigger source to aid debugging. The stream sub-state is related to IEEE 1588-2008 “portState” (Table 10) except for FAULTY and DISABLED (see PTP port states). Note a maximum of ZL303XX_PTP_STREAM_SUBSTATE_HISTORY_SIZE (default 20) elements are stored per-stream in a FIFO buffer (i.e. the oldest entries are removed as new transition events occur). A count of the total transition events that have occurred can be obtained as an optional argument. This API returns the most recent transition event first and can be used to obtain the current stream sub-state. See Table 8-12 for stream sub-state transition reason strings.

TABLE 8-12: STREAM SUB-STATE TRANSITION REASON STRINGS

String	Description (Reason for State Transition)
PORT_INIT	Port initializing.
PORT_DISABLED	Port disabled.
PORT_FAULTY	Port faulty.
INIT	Stream initializing.
IDLE_LISTEN	Stream idle.
S0_MO	BMCA slave on slave-only clock with master-only stream mode override.
S0_NOT_BEST	BMCA slave on slave-only clock BMCA inferior stream.
S0_NOT_BEST_NOT_QUAL	BMCA slave on slave-only clock BMCA inferior FMT unqualified stream.
S0_BEST	BMCA slave on slave-only clock BMCA best stream.
M1_SO	BMCA master on master-only clock with slave-only stream mode override.
M1_LOCAL	BMCA master on master-only clock with local best source.
M1_OTHER	BMCA master on master-only clock with other best source.
M2_SO_OR_UNG	BMCA master with slave-only stream mode override OR contractless unicast negotiated stream.
M2_DENY_SERVICE	BMCA master with denyServiceRequests set true.
M2_LOOPBACK	BMCA master with loopback condition.
M2_MASTER	BMCA master.
S1M3_SO_MONITOR	BMCA slave with slave-only stream mode override BMCA inferior stream.
S1M3_SO_BEST	BMCA slave with slave-only stream mode override BMCA best stream.
S1M3_DENY_SERVICE_PASSIVE	BMCA slave with denyServiceRequests set true for BMCA inferior stream.
S1M3_G8275_PASSIVE	BMCA slave with G.8275.1 or G.8275.2 worse by topology for BMCA inferior stream.
S1M3_PASSIVE	BMCA slave worse by topology for BMCA inferior stream.
S1M3_LOOP_PASSIVE	BMCA slave with loopback condition for BMCA inferior stream.
S1M3_MASTER	BMCA slave for BMCA inferior stream.
S1M3_SLAVE	BMCA slave for BMCA best stream.

TABLE 8-12: STREAM SUB-STATE TRANSITION REASON STRINGS (CONTINUED)

String	Description (Reason for State Transition)
PRE_MASTER_EXPIRED	PRE_MASTER state timer has expired.
M1M2_PRE_MASTER_SKIP	PRE_MASTER timer skipped.
M3_PRE_MASTER_FAILED	PRE_MASTER timer failed to start.
UNCAL_EXPIRE	Uncalibrated timer expired.
SLAVE_LOST_LOCK	Slave with uncalibrated lockRequired lost lock.
PASSIVE_UNINEG	Passive stream with a unicast negotiated expired announce contract.
UMT_START	Unicast negotiated stream initializing.
MASTER_SO	Master state requested on stream with slave-only mode override.
LISTENING_MO	Listening state requested on stream with master-only mode override.

8.2.3 PTP Stream Packet Count Reporting

PTP streams maintain counters of events for debugging and reporting available through the APIs in Table 8-13. The counts available are provided in zl303xx_PtpStreamCounterS data structure described in Table 8-14. A maximum of 4,294,967,295 (2^{32} - 1) events can be counted before the counter rolls over (51 years at typical max rate of 64 pps).

TABLE 8-13: API CALLS FOR PTP STREAM PACKET COUNT REPORTING

API Function	Description
zl303xx_PtpStreamCountersGet	Returns the message counter data collected by a stream into a user allocated data structure (zl303xx_PtpStreamCountersS).
zl303xx_PtpStreamCountersReset	Initializes all stream message counters to 0.

TABLE 8-14: ZL303XX_PTPSTREAMCOUNTERS FIELDS

Member	Sub-Member	Description
msg[id]	tx	Number of ingress packets on this stream (received packets).
	rxPorcessed	Number of ingress packets successfully processed on this stream (received packets not dropped).
	lost	Estimated number of lost packets on this stream (based on gaps in sequenceId). Note it may not accurately detect large outage periods where losses may exceed the size of sequenceId field (Uint16T, 65535).
	tx	Number of egress packets on this stream (transmitted packets)
tlv	rx	Number of ingress TLVs received on this stream.
	rxProp	Number of ingress TLVs received on this stream that had propagate requirements from IEEE 1588-2019 clause 14.2.
	tx	Number of egress TLVs sent on this stream.
	txProp	Number of egress TLVs sent on this stream due to propagate requirements from IEEE 1588-2019 clause 14.2.

Chapter 9. Trace and Logs

9.1 TRACING FACILITIES

The API includes tracing facilities to allow users to gain visibility of what is going in inside its functions. Microchip Support may ask a user to turn on tracing to troubleshoot a problem. The key operations available are listed in the following sections:

9.1.1 Trace Macros

Trace messages can help immensely with the debug process. Each major software module has one or more trace identifiers associated with it. A complete list of trace identifiers are defined in the enum `zl303xx_ModuleIdE` found in the file: `zlUserPorting\include\zl303xx_Trace.h`.

Trace numbers are incremental, that is, trace level 5 will include all trace messages from level 5 through 1. To turn off tracing, set the level back to 0.

The trace invocation macros are as follows:

- ZL303XX_TRACE(Uint16T modId, Uint8T level, char *fmtStr, arg0, arg1, arg2, arg3, arg4, arg5)
- ZL303XX_TRACE_ALWAYS(char *fmtStr, arg0, arg1, arg2, arg3, arg4, arg5)
- ZL303XX_TRACE_ERROR(char *fmtStr, arg0, arg1, arg2, arg3, arg4, arg5)

These macros take six parameters after the format string which are formatted at run-time. They may be set to zero if they are not required.

To ensure the trace message is displayed correctly the following rules must be followed:

- The fmtStr must be a constant pointer to a constant string
- The optional parameters must be either integers or constant pointers to constant strings cast as Uint32T.

9.1.2 Trace Example

Timestamp Packet Stream Trace

```
-> zl303xx_TraceSetLevel 10,3

Output:

0x1b9eb48 (zlPtpTask): zl303xxPtpGetTimestampRx:

0x1b9eb48 (zlPtpTask): Matched record with Rx: sysTs=
B3CE9C8D,

insertTs= 479B1AEC.007D6B9A, ptpTs= 479B1AEC.007D7736, Seq-
Num= 59450

0x1b9eb48 (zlPtpTask): zl303xxPtpGetTimestampRx:

0x1b9eb48 (zlPtpTask): Matched record with Rx: sysTs=
B3DAD39E,

insertTs= 479B1AEC.01161C85, ptpTs= 479B1AEC.01162785, Seq-
Num= 59451

-> zl303xx_TraceSetLevel 10,0
```

Look for anything that does not make sense:

- Timestamp values of 0 (or seconds portion that is 0)
- Sequence numbers that don't increment or skip numbers.
- Timestamp values that don't changes or increase at a normal rate.

9.1.3 Logging Modules

The following table outlines the module IDs available:

TABLE 9-1: AVAILABLE PTP LOGGING MODULES AND DESCRIPTIONS

Module Name	Description
ZL303XX_MOD_ID_PTP_ENGINE	Generic PTP messages.
ZL303XX_MOD_ID_PTP_BMC	Displays logs relates to the Best-Master-Clock process that determines the preferred server in a network.
ZL303XX_MOD_ID_PTP_UNICAST	Displays logs relates to Unicast Negotiation Signal Messages that are exchanged between servers & clients.
ZL303XX_MOD_ID_PTP_UNI_DISC	Deprecated.
ZL303XX_MOD_ID_PTP_STATE_UPD	Displays logs relates to State changes of the PTP Streams.
ZL303XX_MOD_ID_PTP_FMT	Displays logs related to PTP Foreign Master Table.
ZL303XX _MOD_ID_PTP_TIMER	Displays logs related to PTP Timers.
ZL303XX _MOD_ID_PTP_TLV	Displays logs related to PTP TLV processing.
ZL303XX_MOD_ID_TRACK_PKT_PROCESS	Displays logs related to PTP packet processing.
ZL303XX_MOD_ID_TS_RECORD_MGR	Displays logs related to PTP timestamp pair management.
ZL303XX_MOD_ID_PTSF	Displays logs related to the PTSF module.

9.1.4 Log Levels

Each of the modules specified in the previous table may have a different log level associated with it. In general, the following behavior applied:

- A log level of '0' disables logging for the module.
- Error messages are generally triggered at level '1' (although more detailed error logs may also be produced at higher levels as well).
- Enabling log messages at one level enables all messages at the lower levels as well (for example, setting the log level at '2' for a PTP module will generate all level '2' and '1' messages).
- Generally, logs get more detailed as the log level is increased.

Chapter 10. External PTP Interfaces

10.1 INTERFACE INTRODUCTION (CONFIG, EVENTS, TRANSMIT AND RECEIVE)

The following figure illustrates the interfaces of the PTP application that must be integrated by the user:

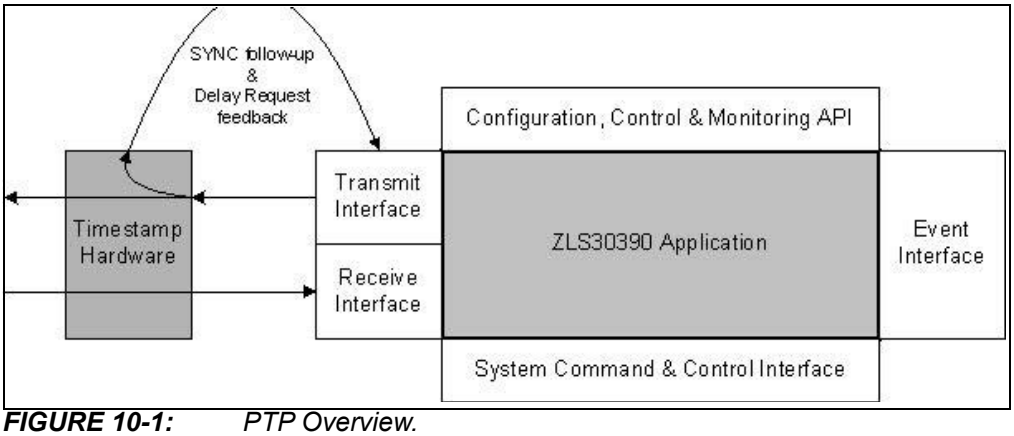

FIGURE 10-1: *PTP Overview.*

10.1.1 Interface for Configuration, Control, and Monitoring

The configuration interface is the set of API functions used to create and delete various PTP objects. These are listed and described in

- **Section 5.1 "Configuration of a PTP Application Instantiation: zl303xx_PtpInit()"**,
- **Section 5.2 "Configuration of a PTP Clock: zl303xx_PtpClockCreate()"**,
- **Section 5.3 "Configuration of a PTP Port: zl303xx_PtpPortCreate()"**,
- **Section 5.4 "Configuration of a PTP Stream: zl303xx_PtpStreamCreate()"**,
- **Chapter 8. "Reporting"**,

and the code-level API descriptions are found in the ZL30390 API Reference Manual.

10.1.2 Interface for Packet Transmit Bindings

The transmit interface is implemented using a function binding. This process and the data available to the function binding are described in Section 10.5.3 "Interface Binding: Transport for Transmit Packets (Egress)".

10.1.3 Interface for Packet Receive Bindings

The receive interface is the set of API functions used to pass the PTP datagram and receive time stamp (if applicable) to the PTP application. These are listed and described in Section 10.5.4 "Interface Binding: Transport for Receive Packets (Ingress)".

10.1.4 Interface for Events

The event interface is implemented as a function binding that is called for every event generated by the PTP application. More information is available in Section 8.1 "Event Interface".

10.1.5 Interface for System Commands

The system command interface is used by the PTP application to access or control an underlying platform resource or associated device. This is achieved with the implementation of a Command Handler Interface for a list of well-defined actions.

10.1.5.1 COMMAND HANDLER

The command handler is a required user function binding (cannot be NULL) set in `zl303xx_PtpClockCreateS::hwCmdFn` prior to calling `zl303xx_PtpClockCreate()`. This function will be called for every command requested by the PTP application. Inside the command handler, a switch() statement should be used to handle only the applicable commands.

The prototype for the command handler is defined as:

Sint32T (* `zl303xx_PtpHwCmdFnT`)(`zl303xx_PtpHwCmdE` cmdType, void *pCmdParams)

For every command issued by the PTP application, the handler is passed the command type and a data structure with command inputs and outputs. The data structure is passed as a (void *) and must be cast to the proper type.

10.1.5.2 COMMAND DETAILS

For additional information about the members of each command structure, refer to the ZLS30390 API Reference Manual.

An example command handler function that illustrates the required interface details can be found in the `zlUserExamples/src/zl303xx_ExamplePtp.c` function, `examplePtpHwCmdHandler()`.

Refer to the following table for information about how/when each command is issued and which data type the supplied pointer should be cast.

TABLE 10-1: PTP COMMAND INTERFACE DEFINITIONS

COMMAND ID and Data Structure	Description
ZL303XX_PTP_HW_CMD_CLOCK_TIME_GET (zl303xx_PtpHwClockTimeGetS)	A command to get the estimated time of a hardware clock. This time is used to fill the originTimestamp field of general packets and SYNC packets transmitted from a two-step clock. The returned time must be accurate to within ±1 second or set to zero (as per the IEEE-1588 standard).
ZL303XX_PTP_HW_CMD_CLOCK_TIME_SET (zl303xx_PtpHwClockTimeSetS)	A command to set the time of a hardware clock. Generally, this function is called to set the seconds portion of a time-of-day hardware register.
ZL303XX_PTP_HW_CMD_PHYS_ADDR_GET (zl303xx_PtpHwPhysAddrGetS)	A command to gather data when a GET is requested for the CLOCK_DESCRIPTION management TLV.
ZL303XX_PTP_HW_CMD_LOCK_STATUS_GET (zl303xx_PtpHwLockStatusGetS)	A command to get the hardware clock LOCK status (may be from the actual hardware or timing algorithm).
ZL303XX_PTP_HW_CMD_CLOCK_STABILITY_GET (zl303xx_PtpHwClockStabilityGetS)	A command to get the hardware clock STABILITY status (may be from the actual hardware or timing algorithm).
ZL303XX_PTP_HW_CMD_TIMESET_STATUS_GET (zl303xx_PtpHwTimeStatusGetS)	A command to get information from the hardware clock about whether the Time has been set (may be from the actual hardware or timing algorithm).
ZL303XX_PTP_HW_CMD_PLL_PERF_DATA_GET (zl303xx_PtpHwPllPerformanceGetS)	A command to get information from the hardware clock about the current operating status of the PLL (may be from the actual hardware or timing algorithm).

10.2 INTERFACE TO TIME SYNCHRONIZATION ALGORITHM

10.2.1 Interface Event: Timing Packet Timestamps (Egress)

The IEEE 1588-2008 Protocol Engine passes timestamp information to the Time Synchronization Algorithm. The raw timestamp information itself is provided for either the forward path (Sync, Follow_Up), the reverse path (Delay_Req, Delay_Resp), the path delay (Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up), or some combination thereof. For the forward path the timestamp information is the arrival timestamp (as recorded by the client when the Sync packet arrived) and the departure timestamp (as inserted by the server in the Sync or Follow_Up message). For the reverse path the timestamp information is the arrival timestamp (as recorded by the server when the Delay_Req packet arrived and inserted into the return Delay_Resp message) and the departure timestamp (as recorded by the client when the Delay_Req packet was sent).

In addition to the raw timestamp information the connection identifier associated with the connection is provided to allow the Time Synchronization Algorithm to track multiple servers. A correction field is also provided when the network supports transparent clock nodes. A full timestamp field is 48-bits seconds and 32-bits nanoseconds, however not all Ethernet MAC/PHYs support this full timestamp field. The timestamp size is configured when opening a PTP connection by the application.

Each time the protocol determines a set of timing data (timing packet pairs, etc.); the data is exported through the Event interface using the ZL303XX_PTP_EVENT_SERVO_DATA message that uses the `zl303xx_PtpEventServoDataS` structure:

TABLE 10-2: ZL303XX_PTPEVENTSERVODATAS STRUCTURE FOR ZL303XX_PTP_EVENT_SERVO_DATA EVENT

zl303xx_PtpEventServoDataS	Type	Description
streamHandle	zl303xx_PtpStreamHandleT	Handle to the stream the time stamp pair is associated with.
streamExtData	void *	Pointer to the stream's external data (which may contain a mapping to a Timing Server ID).
txTs	zl303xx_TimeStamp	The transmit time stamp of the event message in PTP format
rxTs	zl303xx_TimeStamp	The receive time stamp of the event message in PTP format
correctionField	Uint64S	The value of the correctionField in the PTP message
peerMeanPathDelay	Uint64S	The value of the peerMeanPathDelay calculated at the current node (i.e. An estimate of the one-way propagation delay to its Peer Node). If the value of zl303xx_PortDS::delayMechanism is DELAY_-MECHANISM_P2P. Otherwise, this is 0.
tsPairType	zl303xx_PtpTsPairE	Whether this time stamp pair is from a SYNC or DELAY_REQ/DELAY_RESP.
sequenceId	Uint16T	SequenceId of the PTP message associated with these time stamps.
offsetFromMaster Valid	zl303xx_Boolean	Flag indicating if the offsetFromMaster value provided is valid. It may be invalid if there was a correctionField overflow or the source of the SYNC and Delay-Request messages are different.
offsetFromMaster	zl303xx_TimeStamp	The offsetFromMaster value calculated from the last SYNC and Delay-Request messages received. The offsetFromMasterValid flag should be checked to determine if this value is accurate or may contain error.
meanPathDelayValid	zl303xx_Boolean	Flag indicating if the meanPathDelay value provided is valid. It may be invalid if there was a correctionField overflow or the source of the SYNC and Delay-Request messages are different.

TABLE 10-2: ZL303XX_PTPEVENTSERVODATAS STRUCTURE FOR ZL303XX_PTP_EVENT_SERVO_DATA EVENT (CONTINUED)

zl303xx_PtpEventServoDataS	Type	Description
meanPathDelay	zl303xx_TimeStamp	The meanPathDelay value calculated from the last SYNC and Delay-Request messages received. The meanPathDelayValid flag should be checked to determine if this value is accurate or may contain error.

A sample of a timestamp integration is provided in the ZL30390 package. Refer to the following example function in `zl303xx_ExamplePtpBinding.c` if the ZL30380 package is available.

- *examplePtpEventServoData()*

10.2.2 Interface Event: Timing Packet Rate Change Notification (Egress)

If at any time, the timing protocol determines the timing packet rate has changed, it will issue the ZL303XX_PTP_EVENT_MSG_INTVL_CHANGE Event Message to inform the Timing Algorithm. This message uses the `zl303xx_PtpEventMsgIntvlChangeS` structure.

TABLE 10-3: ZL303XX_PTPEVENTMSGINTVLCHANGES STRUCTURE FOR ZL303XX_PTP_EVENT_MSG_INTVL_CHANGE EVENT

zl303xx_PtpEventMsgIntvlChangeS	Type	Description
streamHandle	zl303xx_PtpStreamHandleT	Handle to the stream the interval change was detected on.
streamExtData	void *	Pointer to the stream's external data (which may contain a mapping to a Timing Server ID).
messageType	zl303xx_MessageTypeE	The message type that changed interval.
prevIntvl	Sint8T	Previous message interval.
currIntvl	Sint8T	New message interval.

A sample of the timing packet rate change notification integration is provided in the ZL30390 package. Refer to the *examplePtpEventMsgIntvlChange()* example function if the ZLS30380 package is available.

10.3 INTERFACE TO SYSTEM SYNCHRONIZATION INFO

10.3.1 Interface Command: Sync Lock Status (Ingress)

At certain times the timing protocol software may need to determine the LOCK status of the current recovered clock or hardware PLL. This may be done using the `zl303xx_PtpHwLockStatusGetS` structure to issue the ZL303XX_PTP_HW_CMD_LOCK_STATUS_GET command.

TABLE 10-4: ZL303XX_PTPHWLOCKSTATUSGETS STRUCTURE FOR ZL303XX_PTP_HW_CMD_LOCK_STATUS_GET COMMAND

Sub-Structure	Parameter	Type	Description
input	streamHandle	zl303xx_PtpStreamHandleT	Handle to the stream the interval change was detected on.
input	streamExtData	void *	Pointer to the stream's external data (which may contain a mapping to a Timing Server ID).
input	phaseRequired	Uint32T	A minimum of FREQUENCY lock is expected. However, some profile require PHASE lock as well. Provide this criteria so the driver can reply.
input	profile	zl303xx_PtpProfileE	For compatibility purposes, provide the PTP profile in case future profiles are added.

TABLE 10-4: ZL303XX_PTPHWLOCKSTATUSGETS STRUCTURE FOR ZL303XX_PTP_HW_CMD_LOCK_STATUS_GET COMMAND (CONTINUED)

Sub-Structure	Parameter	Type	Description
output	lockStatus	Uint32T	Return the clock Lock Status (TRUE or FALSE).

A sample of the LOCK Status command integration is provided in the ZL30390 package. Refer to the following example function in `zl303xx_ExamplePtpBinding.c` if the ZL30380 package is available.

- *examplePtpHwLockStatusGet()*

10.3.2 Interface Command: Sync Stability Status (Ingress)

At certain times the timing protocol software may need to determine the Stability of the hardware clock. This may be done using the `zl303xx_PtpHwClockStabilityGetS` structure to issue the ZL303XX_PTP_HW_CMD_CLOCK_STABILITY_GET command.

TABLE 10-5: ZL303XX_PTPHWCLOCKSTABILITYGETS STRUCTURE FOR ZL303XX_PTP_HW_CMD_CLOCK_STABILITY_GET COMMAND

Sub-Structure	Parameter	Type	Description
input	clockHandle	zl303xx_PtpClockHandleT	Handle to the clock being queried.
input	clockExtData	void *	Pointer to the clock's external data (which may contain a mapping to a Timing Device ID).
output	freqStability	Uint32T	Frequency Stability value.
output	phaseStability	Sint32T	Phase Stability value.

A sample of the Stability Status command integration is provided in the ZL30390 package. Refer to the following example function in `zl303xx_ExamplePtpBinding.c` if the ZL30380 package is available.

- *examplePtpHwClockStabilityGet()*

10.3.3 Interface Command: Sync Performance Data (Ingress)

At times, the timing protocol software will require PLL status information in order to make certain state decisions. This is queried using the `zl303xx_PtpHwPllPerformanceGetS` structure to issue the ZL303XX_PTP_HW_CMD_PLL_PERF_DATA_GET command.

TABLE 10-6: ZL303XX_ PTPHWPLLPERFORMANCEGETS STRUCTURE FOR ZL303XX_PTP_HW_CMD_PLL_PERF_DATA_GET COMMAND

Sub-Structure	Parameter	Type	Description
input	clockHandle	zl303xx_PtpClockHandleT	Handle to the clock being queried.
input	clockExtData	void *	Pointer to the clocks' external data (which may contain a mapping to a Timing Device ID).
output	pllOperatingMode	zl303xx_PtpPllOperatingModeE	PLL Operating mode (ELEC, PKT, HYBRID, etc.).
output	hwPllOperatingState	zl303xx_PtpPllOperatingStateE	Hardware PLL/Algorithm lock status (Freerun, Holdover, Acquiring, Locked, etc.). Valid when pllOperatingMode = ELECTRIC or HYBRID.
output	swPllOperatingState	zl303xx_PtpPllOperatingStateE	Software PLL/Algorithm lock status (Freerun, Holdover, Acquiring, Locked, etc.). Valid when pllOperatingMode = PACKET_MODE.
output	pllHoldoverQuality	zl303xx_PtpPllFreqHoldoverQualityE	Holdover quality category (Category 1,2,or 3).
output	bHoldoverInSpec	zl303xx_BooleanE	Holdover IN/OUT of Specification.
output	synceTraceable	zl303xx_BooleanE	Flag indicating if the SyncE is frequency traceable (for HYBRID mode).

TABLE 10-6: ZL303XX_ PTPHWPLLPERFORMANCEGETS STRUCTURE FOR ZL303XX_PTP_HW_CMD_PLL_PERF_DATA_GET COMMAND (CONTINUED)

Sub-Structure	Parameter	Type	Description
output	settlingTimeActive	zl303xx_BooleanE	Flag indicating that a reference switch or other event may have recently occurred and that the current reference should be viewed as LOCKED. Once this flag goes to FALSE, the actual state of the PLL will be used internally. Refer to the next sub-section.

A sample of the PLL Performance Status command integration is provided in the ZL30390 package. Refer to the following example function in `zl303xx_ExamplePtpBinding.c` if ZL30380 package is available.

- *examplePtpHwPllPerformanceGet()*

10.3.3.1 EGRESS ANNOUNCE USING THE 'SETTLINGTIMEACTIVE' PARAMETER

In the G.8275 Profiles, priority is given to settling the topology of the network as fast as possible. When switching from one source to another, the PLL may enter a Holdover and/or Acquiring state and begin advertising a HOLDOVER-Class other than the class provided by the new parent reference. In turn, this may trigger switches on downstream nodes.

To avoid this, the 'settlingTimeActive' Parameter has been added to the `zl303xx_PtpPllPerformanceDataS` data structure (refer to previous section). When active, the G.8275 engine ignores the computed Holdover output values and uses the selected ParentDS value directly.

It is recommended that the 'settlingTimeActive' flag be set for a finite time period. If the new reference is unable to LOCK within that time then the output should revert to the Holdover values calculated by the G.8275 engine.

A possible procedure would be to:

1. Set 'settlingTimeActive' = TRUE when the BMCA makes a new selection (prior to applying the new ParentDS).
2. Start a Timer to limit how long the PLL has to LOCK.
3. If the Timer expires or the new reference LOCKs, set 'settlingTimeActive' = FALSE (allowing the G.8275 engine to take back output control).

10.4 INTERFACE TO TIME OF DAY

10.4.1 Interface Event: Leap Seconds Flag (Egress)

Refer to Section 7.1 “Handling Leap Second Events” for additional information.

When the timing protocol software detects an upcoming Leap Seconds event, it uses the `zl303xx_PtpEventLeapSecondsFlagChangeS` structure to issue the ZL303XX_PTP_EVENT_LEAP_SECONDS_FLAG_CHANGE event in order to allow the user application to manage the event.

TABLE 10-7: ZL303XX_PTPEVENTLEAPSECONDSFLAGCHANGES STRUCTURE FOR ZL303XX_PTP_EVENT_LEAP_SECONDS_FLAG_CHANGE EVENT

Sub-Structure	Parameter	Type	Description
—	clockHandle	zl303xx_PtpClockHandleT	Handle to the clock for which the event occurred.
—	clockExtData	void *	Pointer to the clocks’ external data (which may contain a mapping to a Timing Device ID).
—	localTimeProperties	zl303xx_TimePropertiesDS	The configured TimePropertiesDS of the exporting clock.
—	currentTimeProperties	zl303xx_TimePropertiesDS	The current, dynamic TimePropertiesDS of the exporting clock which includes the current leap59 & leap 61 values.
—	previousLeap59	zl303xx_BooleanE	The previous values of the leap59 flag.
—	previousLeap61	zl303xx_BooleanE	The previous values of the leap61 flags.
—	currentPtpTime	zl303xx_TimeStamp	The current PTP Time on the clock.

A sample of the information from the Leap Seconds Event is provided in the ZL30390 package. Refer to the following example function in `zl303xx_ExamplePtp.c`.

- *examplePtpEventLeapSecondsFlagChange()*

10.4.2 Interface Event: UTC Offset Change (Egress)

When the timing protocol software detects a change in the UTC offset or status, it uses the `zl303xx_PtpEventUtcOffsetChangeS` structure to issue the ZL303XX_PTP_EVENT_UTC_OFFSET_CHANGE event in order to allow the user application to manage the event.

TABLE 10-8: ZL303XX_PTPEVENTUTCOFFSETCHANGES STRUCTURE FOR ZL303XX_PTP_EVENT_UTC_OFFSET_CHANGE EVENT

Sub-Structure	Parameter	Type	Description
—	clockHandle	zl303xx_PtpClockHandleT	Handle to the clock for which the event occurred.
—	clockExtData	void *	Pointer to the clocks’ external data (which may contain a mapping to a Timing Device ID).
—	localTimeProperties	zl303xx_TimePropertiesDS	The configured TimePropertiesDS of the exporting clock.
—	currentTimeProperties	zl303xx_TimePropertiesDS	The current, dynamic TimePropertiesDS of the exporting clock which includes the current leap59 & leap 61 values.
—	previousUtcOffset	Sint32T	The previous values of the UTC Offset
—	previousUtcOffsetValid	zl303xx_BooleanE	The previous values of the UTC Offset Valid flag.
—	currentPtpTime	zl303xx_TimeStamp	The current PTP Time on the clock.

A sample of the information from the UTC Change Event is provided in the ZL30390 package. Refer to the following example function in `zl303xx_ExamplePtp.c`.

- *examplePtpEventUtcOffsetChange()*

10.4.3 Interface Command: Time of Day Get (Ingress)

Periodically the timing protocol requires the estimated time of a hardware clock for such things as filling the originTimestamp field of general messages (e.g., Announce), the originTimestamp field of Sync messages transmitted from a two-step clock, and the currentTime field of a TIME Management TLV. It is also used when the `zl303xx_PtpClockTaiGet()` and `zl303xx_PtpClockUtcGet()` functions are called. The returned time must be accurate to within ±1 second or set to all zeros (as per IEEE Std 1588-2008).

If you require an even more accurate timer then you require a H/W Real-Time Clock with a custom interface to the PTP servo.

The timing protocol software uses the `zl303xx_PtpHwClockTimeGetS` structure to issue the ZL303XX_PTP_HW_CMD_CLOCK_TIME_GET command in order to get the current time.

TABLE 10-9: ZL303XX_PTPHWCLOCKTIMEGETS STRUCTURE FOR ZL303XX PTP_HW_CMD_CLOCK_TIME_GET COMMAND

Sub-Structure	Parameter	Type	Description
input	clockHandle	zl303xx_PtpClockHandleT	Handle to the clock for which the event occurred.
input	clockExtData	void *	Pointer to the clocks's external data (which may contain a mapping to a Timing Device ID).
input	twoStepFlag	zl303xx_BooleanE	Two-step clock flag. If this is false (egress timestamps can be inserted on the fly), it may still be necessary to return part of the time stamp. E.g., a hardware device cannot write the top 16 bits of the time stamp.
input	interface	zl303xx_PtpInterfaceE	Interface that the message will be sent on.
output	ptpTime	zl303xx_TimeStamp	Return the current time of the clock in PTP format here.

A sample of the information from the Time query command is provided in the ZL30390 package. Refer to the following example function in `zl303xx_ExamplePtp.c`.

- *examplePtpHwClockTimeGet()*

10.4.4 Interface Command: Time of Day Set (Egress)

The timing protocol provides an interface to set the Time-of-Day on the local system. Since PTP may not be aware of the Timing Algorithm or PLL specifics that control the Time locally, it is the user responsibility to ensure that the controls in setting the time on the system are implemented properly.

The timing protocol software can set the local Time-of-Day using the `zl303xx_PtpHwTimeStatusSetS` structure to issue the ZL303XX_ PTP_HW_CMD_TIMESET_STATUS_SET command.

TABLE 10-10: ZL303XX_PTPHWTIMESTATUSSETS STRUCTURE FOR ZL303XX_ PTP_HW_CMD_TIMESET_STATUS_SET COMMAND

Sub-Structure	Parameter	Type	Description
input	clockHandle	zl303xx_PtpClockHandleT	Handle to the clock for which the time is being set.
input	clockExtData	void *	Pointer to the clocks's external data (which may contain a mapping to a Timing Device ID).
input	ptpTime	zl303xx_Timestamp	Time to set in PTP format.

A sample of the command information and steps required to set a local Time-of-Day is provided in the ZL30390 package. Refer to the following example function in `zl303xx_ExamplePtp.c`.

- *examplePtpHwClockTimeSet()*

10.4.5 Interface Command: Time of Day Set Status (Ingress)

Some PTP profiles limit the data sent downstream in Announce messages until a local Time-of-Day has been set. This avoids the situation where a downstream node starts to acquire to a master clock but then the master jumps its own Time-of-Day and affects the client's ability to LOCK properly (or causes the client to lose LOCK, etc).

The timing protocol software can query the local Time-of-Day status using the `zl303xx_PtpHwTimeStatusGetS` structure to issue the ZL303XX_PTP_HW_CMD_TIMESET_STATUS_GET command.

TABLE 10-11: ZL303XX_PTPHWTIMESTATUSGETS STRUCTURE FOR ZL303XX_PTP_HW_CMD_TIMESET_STATUS_GET COMMAND

Sub-Structure	Parameter	Type	Description
input	clockHandle	zl303xx_PtpClockHandleT	Handle to the clock for which the event occurred.
input	clockExtData	void *	Pointer to the clocks' external data (which may contain a mapping to a Timing Device ID).
output	bTimeSet	zl303xx_BooleanE	Return the Time-of-Day Set Status.

A sample of the information from the Time status query command is provided in the ZL30390 package. Refer to the following example function in `zl303xx_ExamplePtp.c` if ZL30380.

- *examplePtpHwTimeSetStatusGet()*

10.5 INTERFACE TO TRANSPORT LAYER (TRANSMIT AND RECEIVE)

10.5.1 Host Processor and Real-Time OS Dependencies

To achieve the IEEE1588-2008 requirements of packet rates within –30% with 90% confidence for Delay_Req and PDelay_Req messages, and ±30% with 90% confidence for Sync messages, it is required that the software timers run at minimum 125 Hz (8 milliseconds) which would usually require the kernel to tick at twice that rate (250 Hz).

The OS adaptation layer in PTP provides a set of OS independent functions to allow PTP to manipulate OS resources and task operations. These functions should be implemented by the user based on the individual OS.

10.5.2 Interface Command: Ethernet Physical Address (Ingress)

The timing protocol software can query the local MAC Address using the `zl303xx_PtpHwPhysAddrGetS` structure to issue the ZL303XX_PTP_HW_CMD_PHYS_ADDR_GET command. This is used for supporting the PTP2_MGMT_ID_CLOCK_DESCRIPTION management TLV GET (see IEEE 1588-2008 section 15.5.3.1.2 and Table 41).

TABLE 10-12: ZL303XX_PTPHWPHYSADDRGETS STRUCTURE FOR ZL303XX_PTP_HW_CMD_PHYS_ADDR_GET COMMAND

Sub-Structure	Parameter	Type	Description
input	portHandle	zl303xx_PtpPortHandleT	Handle to the port being queried.
input	portExtData	void *	Pointer to the port's external data.
output	physicalLayerProtocol	char[32]	Return the physical layer protocol string.
output	physicalAddressLength	Uint16T	Return the length of the physical address string.
output	physicalAddress	Uint8T[16]	Return the physical address string.

10.5.3 Interface Binding: Transport for Transmit Packets (Egress)

10.5.3.1 TRANSMIT FUNCTION BINDING

PTP message transmission is handled by a function callout that must be setup by a user application. The user function is bound to `zl303xx_PtpPortCreateS::txMsgFn`, which must be set prior to calling `zl303xx_PtpPortCreate()`.

This function will be called for every packet that needs to be sent from this port.

The prototype for the message transmit function is defined as:

- Sint32T (* `zl303xx_PtpTxMsgFnT`)(Uint8T *buffer, Uint16T bufLen, `zl303xx_PtpTxMsgDataS` *msgData);

where **buffer** and **bufLen** contain the PTP message and message length, respectively. A Sint32T value can be returned by the user function where 0 indicates success. By default this return value is only used for logging purposes; it has no effect on state machine operation. However, if the optional compile-time option ENABLE_EXT_TX_API_CHECKS is defined then an error return value (non zero) may cause the PTP Port to enter FAULTY state (see Section 6.2.5 for more details about FAULTY state).

10.5.3.2 TRANSMIT DATA TYPE DESCRIPTION

The following table details the members of the `zl303xx_PtpTxMsgDataS` data structure used for this interface.

TABLE 10-13: PTP TRANSMIT INTERFACE DEFINITIONS

Structure Member	Description
portHandle	Handle of the port this message is being transmitted from.
streamHandle	Handle of the stream this message is associated with. Must be used along with sequenceId to register transmit time stamp information back to the PTP application.
clockExtData	Pointer to external clock data.
portExtData	Pointer to external port data.
streamExtData	Pointer to external stream data.
srcAddr	Source port address of this message. This will be the value set during port creation, in zl303xx_PtpPortCreateS:localAddr.
destAddr	Destination port address of this message.
interface	Which PTP "interface" this message should be sent on. If UDP transport is being used, event messages are sent on port 319, and general messages are sent on port 320.
messageId	Value of the messageType field in the PTP header. Can be used to determine if this message should have its transmit time stamp recorded.
sequenceId	Value of the sequenceId field in the PTP header. Must be used along with streamHandle to register transmit time stamp information back to the PTP application.

10.5.3.3 TRANSMIT TIME STAMPS

The transmit time stamp of a PTP packet will need to be retrieved from hardware for the following cases:

- A Sync message is transmitted from a two-step clock.
- A Delay_Req message is transmitted.
- A PDelay_Req message is transmitted.
- A PDelay_Resp message is transmitted from a two-step clock.

When the transmit time stamp is available from hardware, it must passed to the PTP application with the following data from `zl303xx_PtpTxMsgDataS`:

- streamHandle
- messageId
- sequenceId

It is the responsibility of the user transport layer to associate and recombine this data with transmit time stamps.

Transmit time stamps are registered with the PTP application by calling `zl303xx_PtpTsRecordProcess()`. In the case of registering the transmit time of a PDelay_Resp or Sync message from a two-step clock, calling this triggers the PTP application to generate a PDelay_Resp_Follow_Up or Sync Follow_Up message.

The messageType field of the `zl303xx_PtpTsRecordS` data structure must be set properly when calling the `zl303xx_PtpTsRecordProcess()` routine. Prior to the addition of the PDelay mechanism (Release 4.5.0) this was not required since there were enough other flags available to determine whether the time stamp was for a Sync or Delay_Request message.

The internal table of collected timestamps can be copied to a local memory for debugging via API function `zl303xx_PtpStreamTsRecordTableCopy`. See example in zlUserUtils folder.

10.5.4 Interface Binding: Transport for Receive Packets (Ingress)

10.5.4.1 PRIMARY RECEIVE FUNCTION BINDING

PTP message reception is handled by API function calls. The most common way to register a received packet with PTP is by calling (prototyped in `zlPtp/include/zl303xx_PtpApi.h`):

```
• zlStatusE  zl303xx_PtpRxMsgProcess(
    Uint8T *buffer,
    Uint16T bufferLen,
    zl303xx_PtpRxMsgDataS *pMsgData);
```

where **buffer** and **bufferLen** contain the PTP message and message length, respectively. The `zl303xx_PtpRxMsgDataS` is used to store implementation-specific data that is not available in the PTP message itself.

> **Note:** It is recommended to use the more efficient interface `zl303xx_PtpPortRxMsgProcess()` when the receiving port handle is known (see Section 10.5.4.2).

The following table details the members of the `zl303xx_PtpRxMsgDataS` data structure used for this interface.

TABLE 10-14: PTP RECEIVE INTERFACE DEFINITIONS

Structure Member	Description
srcAddr	Source port address of the PTP message.
destAddr	Destination port address of the PTP message.
rxTs	Receive PTP time stamp. Only applies to event packets.
extData	Small buffer for additional implementation specific data.

This function uses the srcAddr and destAddr pair to resolve which PTP stream the message is intended for. It assumes that the address pair is sufficient to uniquely identify each stream.

> **Note:** A stream may not exist in the case of a master node running unicast negotiation. In this case, only the destAddr is used to look up which port the message is intended for.

If the address pair is insufficient to uniquely identify a stream, it is possible for a user application to specify the handle to the stream the message is intended for. In this case, it is the user's responsibility to maintain a table or other data structure that maps each received packet to a specific stream handle. The following alternative APIs may also be used.

10.5.4.2 ALTERNATE RECEIVE FUNCTION BINDINGS

The following routine is used to send a received message to the PTP task for processing on a specific PTP Port. It searches a single PTP Port for the first Stream that either:

a) Has the same destination address as the received message source address.
b) Has the same destination address as the received message destination address (in the case of multicast)

```
• zlStatusE zl303xx_PtpPortRxMsgProcess(
    Uint8T *buffer,
    Uint16T bufferLen,
    zl303xx_PtpPortRxMsgDataS *pMsgData);
```

The **buffer** and **bufferLen** contain the PTP message and message length, respectively as above. The `zl303xx_PtpPortRxMsgDataS` is described below:

TABLE 10-15: ZL303XX_PTPPORTRXMSGDATAS DEFINITION

Structure Member	Description
portHandle	Handle to a previously created port that this message was received on.
srcAddr	Source port address of the PTP message.
destAddr	Destination port address of the PTP message.
rxTs	Receive PTP time stamp. Only applies to event packets.
extData	Small buffer for additional implementation specific data.

The following routine is used to send a received message to the PTP task for processing on a specific PTP Stream.

- `zlStatusE zl303xx_ PtpStreamRxMsgProcess (`

```
Uint8T *buffer,
Uint16T bufferLen,
zl303xx_PtpStreamRxMsgDataS *pMsgData);
```

The **buffer** and **bufferLen** contain the PTP message and message length, respectively as above. The `zl303xx_PtpStreamRxMsgDataS` is described below:

TABLE 10-16: ZL303XX_PTPSTREAMRXMSGDATAS DEFINITION

Structure Member	Description
streamHandle	Handle to a previously created stream that this message was received on.
rxTs	Receive PTP time stamp. Only applies to event packets.
extData	Small buffer for additional implementation specific data.

10.5.4.3 RECEIVE TIME STAMPS

It is the responsibility of the user's transport layer to associate the receive time stamp with the correct packet. The time stamp may be stored in a hardware queue or appended to the end of the packet. It must also be converted to PTP format. If the time stamp is stripped from the end of the packet, ensure the value of bufferLen passed to `zl303xx_PtpRxMsgProcess()` is updated to "remove" the time stamp. Processing of certain PTP signaling TLVs require the buffer length to be correct.

NOTES:

Chapter 11. Test Interfaces

The following table lists some APIs that are available for testing purposes.

TABLE 11-1: API ROUTINES FOR TESTING

API Routine	Description
zl303xx_PtpClockAnnounceTlvEnqueue	API to send a single custom TLV on the next announce packet sent by all streams that are currently sending ANNOUNCE packets. This can be used to test PTP Edition 3 Announce TLV Propagation Requirements (Clause 14.2)
zl303xx_PtpStreamAnnounceTlvEnqueue	API to send a single custom TLV on the next announce packet sent by this stream. This can be used to test PTP Edition 3 Announce TLV Propagation Requirements (Clause 14.2)

NOTES:

Chapter 12. Example Reference Selection Application

12.1 INTRODUCTION

A sample Reference Selection Application is provided with the ZLS30390. The purpose of this application is to collect data from multiple sources within the user system, perform necessary comparisons and re-distribute system control messages to the various components. These components may include:

- Multiple line-card units executing a timing protocol (such as the ZLS30390)
- A Time-Sync Algorithm module to process protocol timing data.
- Local PLL modules.
- Other proprietary modules specific to the user system.

12.1.1 Code Inclusion and Modules

To enable the new content, compile the ZLS30390 with the following compile-time definition:

- ZL303XX_REFERENCE_SELECTION

The code for the application is contained within the following modules:

- `zl303xx_ExampleRefSelectApi.h`
- `zl303xx_ExampleRefSelectSetup.h`
- `zl303xx_ExampleRefSelectInternal.h`
- `zl303xx_ExampleRefSelectApp.c`
- `zl303xx_ExampleRefSelectData.c`
- `zl303xx_ExampleRefSelectTask.c`

12.2 ARCHITECTURE

The following diagram illustrates the general architecture of the Reference Selection Application:

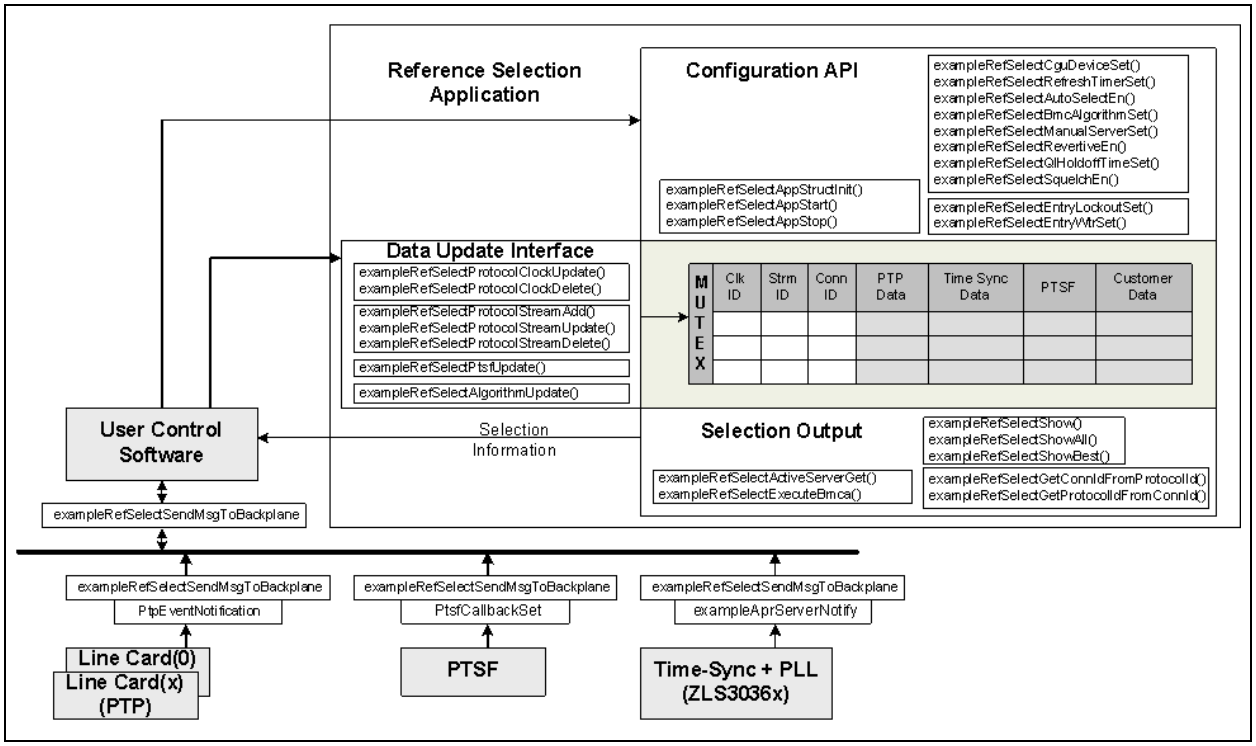

FIGURE 12-1: *Reference Selection Overview.*

As shown in the diagram:

1. All data is collected in a centralized table managed by the Application Task.
2. Data Entries may contain:

a) Local Protocol Clock information: in this case no protocol stream or Time-Sync connection Id is required.

b) Remote server information: in this case, the protocol clock of the protocol stream must be specified along with the associated Time-Sync connection Id. The following routines are provided to cross-reference these paired-Ids:

- `exampleRefSelectGetConnIdFromProtocolId()`
- `exampleRefSelectGetProtocolIdFromConnId()`

3. The Reference Selection Task is responsible for:

a) Controlling the configuration API of the Application.

b) Receiving data from the various system modules, verifying the content and updating the data table accordingly.

c) Running the Reference Selection Algorithm (as configured).

d) Sending the selected server data to the various modules.

4. The Messaging Backplane is an exchange interface on the local system. The following routine is provided as a sample to translate and exchange message between the Reference Selection Application and the individual system modules:

- `exampleRefSelectSendMsgToBackplane()`

12.2.1 Application API

12.2.1.1 APPLICATION INITIALIZATION

The following routines are provided to start, configure and stop the Reference Selection Application:

- `exampleRefSelectAppStructInit()`: retrieves default, start-up values for the application as defined in the `zl303xx_ExampleRefSelectSetup.h` header.
- `exampleRefSelectAppStart()`: starts the application with parameters provided. This includes creating the application task and initializing the server table. If no configuration is provided, the default values are used automatically.
- `exampleRefSelectAppStop()`: terminates the application.

Refer to the provided sample code for examples of how to use these routines.

12.2.1.2 APPLICATION CONFIGURATION

The following routines are provided to manage and configure the Reference Selection Application. Refer to each function description for appropriate data types, return values, and usage.

General Application Management

- `exampleRefSelectAppConfigGet()`: retrieves the configuration of an active application (may be used for Non-Volatile Storage (NVS) or other functionality).
- `exampleRefSelectAppConfigSet()`: changes the configuration of an active application.

Server Selection Management

- `exampleRefSelectRefreshTimerSet()`: set the interval in which the application executes default processing (such as server selection if enabled).
- `exampleRefSelectAutoSelectEn()`: enables automatic selection of the best know server. When enabled, the application refresh period should be set so that updates occur at an acceptable interval.
- `exampleRefSelectBmcAlgorithmSet()`: set the server selection algorithm used to evaluate the best server (expected algorithms are Default 1588, Telecom Profile for Frequency, Power Profile).
- `exampleRefSelectManualServerSet()`: force a particular server to be selected. This overrides the evaluation of the algorithm.
- `exampleRefSelectActiveServerGet()`: receives the data for the current active server last selected by the application.
- `exampleRefSelectExecuteBmca()`: executes the currently configured selection algorithm on the latest data collected and returns the best server (if any). This does not execute internal routines or transmit the data to other system components.

Time Sync Algorithm Interface

- `exampleRefSelectCguDeviceSet()`: sets the Time-Sync data parameter associated with the Time-Sync module.

General Telecom Profile Parameters

The following parameters apply only when the Telecom Profile for Frequency selection algorithm is used.

- `exampleRefSelectRevertiveEn()`: enables revertive switching.
- `exampleRefSelectQlHoldoffTimeSet()`: sets a QL Hold-Off time for drops in received QL.

- exampleRefSelectSquelchEn(): enables squelch protection.

12.2.1.3 SERVER ENTRY CONFIGURATION

- exampleRefSelectEntryLockoutSet(): prohibits a server from being selected.
- exampleRefSelectEntryWtrSet(): sets the time period that a server is prohibited from being re-selected after it has been removed as the selected server.

12.2.2 Message Interfaces

12.2.2.1 FROM PROTOCOL CLOCKS

The following routines provide updates to the Reference Selection Application regarding Protocol Clocks in the system:

- exampleRefSelectProtocolClockUpdate(): provides data from a Protocol Clock when it is created or changed.
- exampleRefSelectProtocolClockDelete(): indicates that a Protocol Clock has been deleted and its entry should be removed from the data table.

To integrate these items, calls to the routines listed above would need to be executed at the event notification output of the PTP stack contained in zl303xx_ExamplePtp.c. Specifically for the following events:

- ZL303XX_PTP_EVENT_CLOCK_CREATE: Package the event data into an exampleProtocolClockUpdateMsgS message and call exampleRefSelectProtocolClockUpdate() (sample code provided).
- ZL303XX_PTP_EVENT_CLOCK_DELETE: Package the event data into an exampleProtocolClockDeleteMsgS message and call exampleRefSelectProtocolClockDelete() (sample code provided).
- ZL303XX_PTP_EVENT_CLOCK_BMCA_UPDATE: This event contains all of the collected server data for a particular line-card including the Default data set of the line-card itself. Package the Default data set of the line-card into an exampleProtocolClockUpdateMsgS message and call exampleRefSelectProtocolClockUpdate() (sample code provided).

12.2.2.2 FROM PROTOCOL STREAMS

The following routines provide updates to the Reference Selection Application regarding Protocol Streams in the system:

- exampleRefSelectProtocolStreamAdd(): indicates a Protocol Stream has been added to the system.
- exampleRefSelectProtocolStreamUpdate(): data for a known Protocol Stream has been updated that may affect the evaluation of the server selection algorithm.
- exampleRefSelectProtocolStreamDelete(): indicates that a Protocol Stream has been deleted and its entry should be removed from the data table.

To integrate these items, calls to the routines listed above would need to be executed at the event notification output of the PTP stack contained in zl303xx_ExamplePtp.c. Specifically for the following events:

- ZL303XX_PTP_EVENT_STREAM_CREATE: Package the event data into an exampleProtocolStreamAddMsgS message and call exampleRefSelectProtocolStreamAdd() (sample code provided).
- ZL303XX_PTP_EVENT_STREAM_KEEP_ALIVE: Package the event data into an exampleProtocolStreamUpdateMsgS message and call exampleRefSelectProtocolStreamUpdate() (sample code provided).

- ZL303XX_PTP_EVENT_CLOCK_BMCA_UPDATE: This event contains all of the collected server data for a particular line-card including the Default data set of the line-card itself. Package the updated data of each server contained in the event into an `exampleProtocolStreamUpdateMsgS` message and call `exampleRefSelectProtocolStreamUpdate()` (sample code provided).
- ZL303XX_PTP_EVENT_STREAM_DELETE: Package the event data into an `exampleProtocolStreamDeleteMsgS` message and call `exampleRefSelectProtocolStreamDelete()` (sample code provided).

12.2.2.3 FROM TIME-SYNC ALGORITHM

The Reference Selection Application expects to receive LOCK status and other status flags from the Time-Sync Algorithm using the following routine:

- `exampleRefSelectAlgorithmUpdate()`:

One recommendation for this data is the list of status flags contained in the `zl303xx_AprServerNotifyS` type (provided with Microchip products that contain the APR Time-Sync Algorithm). In this case, the call to send the APR status flags to the Reference Selection Application would be integrated in the `exampleAprServerNotify()` routine contained with the APR software.

12.2.2.4 FROM PACKET TIMING SIGNAL FAIL (PTSF)

The Reference Selection Application expects to receive PTSF status flags from the various modules in the system. In the current implementation, the Reference Selection Application starts a PTSF collector locally and Protocol Line-Cards, the Time-Sync module and other custom blocks (user, for example) perform PTSF updates using the PTSF API routines (refer to `zl303xx_Ptsf.h`)

The Reference Selection Application listens for updates via the PTSF notification routine and applies these same changes to the server data (use `zl303xx_PtsfCallbackSet()` to intercept updates to the PTSF module). Each PTSF update is then repackaged into an `examplePtsfUpdateMsgS` type and forwarded to the application using the following routine:

- `exampleRefSelectPtsfUpdate()`: indicates that a PTSF status bit for a connection has changed.

12.2.3 Backplane Interface

As mentioned above, all messages from system modules to the Reference Selection Application (and vice versa) are assumed to traverse a Message Backplane. In the sample code provided, this is modeled in a single routine:

- `exampleRefSelectSendMsgToBackplane()`

However, in actual implementations this is most likely implemented as a socket, message queue, or some other data transfer interface (or there may be no interface, in which case function calls are made directly).

Because the details of this interface are proprietary to the user system, it is expected that extensive rework of this particular routine may be required for application integration.

12.2.4 Debug Routines

The following routines are provided in order to help monitor and/or debug the application:

- `exampleRefSelectShow()`: Shows all VALID entries in the table.
- `exampleRefSelectShowAll()`: Dumps the entire table (including empty rows).
- `exampleRefSelectShowBest()`: Shows only the BEST entry from the table (if

it exists).

12.2.5 Typical Operating Sequence

The following outlines the steps that would typically be carried out in using the application:

1. Application Initialization:

a) Executed during system initialization.

b) Call `exampleRefSelectAppStructInit()` to retrieve default configuration values.

c) Modify the configuration as required.

d) Call `exampleRefSelectAppStart()` to initialize the application

2. Add clock and/or server entries to the list of available references. There are two approaches here:

a) The User Control Software may fill the table with a list of known acceptable entries using the routines:
   - `exampleRefSelectProtocolClockUpdate()`
   - `exampleRefSelectProtocolStreamAdd()`

b) Entries may be automatically added as they are created by other modules and notification is received by the application. For example, when Line-Card Clocks and Streams are created, the subsequent event messages (ZL303XX_PTP_EVENT_CLOCK_CREATE and ZL303XX_PTP_EVENT_STREAM_CREATE) may be translated to appropriate Reference Selection Application messages and the server database updated accordingly.

3. Status flags from the Time-Sync Algorithm or PTSF Alarms are monitored and may affect server selection.

4. The Application evaluates all entries and provided data related to the best selection available.

5. The following updates regarding reference entries or configuration changes are constantly evaluated for a change in selection:

a) Addition or Deletion of entries

b) Change in application configuration

c) Updates to local clock configuration

d) Updates to connection data (or re-configuration)

Chapter 13. Redundant Timing Card Application for Chassis Systems

The ZLS30390 API provides functions to support systems that require switching control between an active and standby node. The following parameters determine what actions should be taken when performing a redundant switchover:

- Protocol type: multicast or unicast negotiation
- Operating state: server or client
- Node type: active or standby

For an active node, there are following events to consider:

- Starting a node in active mode
- Controlled switch from active to standby
- Node fails unexpectedly

In the case of an unexpected failure, it may not be possible to perform cleanup or transfer data to the standby node. So, functions and event callbacks are provided to allow a user application to periodically save the state of the active node's streams. Note that it is assumed that there is no internal dynamic clock or port data that needs to be transferred during a switchover. Any runtime configuration changes should be applied to both the active and standby nodes.

For the standby node, there are the following events to consider:

- Starting a node in standby mode
- Controlled switch from standby to active

The following API routines can be used to implement redundant switching and are referenced in the following sections (refer to the PTP API Reference Manual for complete interface, data type, and syntax details).

TABLE 13-1: PTP REDUNDANCY ROUTINES

API Routine	Description
zl303xx_PtpContractManualCancel	Manually send a CANCEL TLV to a destination address. If a stream matching the destination address already exists, the function reverts to calling zl303xx_PtpContractCancel(). If no stream exists, only a single CANCEL TLV will be sent.
zl303xx_PtpTxContractStop	Silently stops unicast message service. No CANCEL TLV will be sent to inform the grantee that the service is being stopped.
zl303xx_PtpContractStatusGet	Returns the status of the most recent unicast negotiation contract (active or inactive).
zl303xx_PtpContractTxStatusSet	Function to manually create TX unicast service without receiving a REQUEST TLV or modify an existing contract. This function will create a new stream if required.
zl303xx_PtpClockMessagingEnabled	If clock messaging is enabled, all RX and TX PTP message processing happens normally. If it's disabled, no TX messages will be sent and all RX messages will be blocked.
zl303xx_PtpStreamConfigGet	Returns the zl303xx_PtpStreamCreateS structure used to initialize a PTP stream. Note that this can be used to retrieve sequenceId.

In addition, the following PTP events can be used to save data about the state of streams in unicast negotiation mode (see Table 8-1):

- ZL303XX_PTP_EVENT_CONTRACT_REJECTED
- ZL303XX_PTP_EVENT_CONTRACT_GRANTED
- ZL303XX_PTP_EVENT_CONTRACT_EXPIRED

13.1 HOW TO CREATE A NODE IN A REDUNDANT SYSTEM

13.1.1 Active Node

No changes are necessary from the included example code for a master or slave node.

13.1.2 Standby Node

These steps can be used for a master or slave node.

1. Create a PTP clock. See "Configuration of a PTP Clock: zl303xx_PtpClockCreate()" for details.
2. Disable clock messaging, so no packets will be sent until the node becomes the active node. See function `zl303xx_PtpClockMessagingEnabled()`.
3. Create the PTP port(s). See "Configuration of a PTP Port: zl303xx_PtpPortCreate()" for details.

No streams should be created until this node becomes the active one.

13.2 HOW TO MANAGE A REDUNDANT NODE

13.2.1 Active Master Node

13.2.1.1 MULTICAST

The user application can optionally save the sequenceId of all messageType using `zl303xx_PtpStreamConfigGet()`, and transfer this data to the standby node.

13.2.1.2 UNICAST NEGOTIATION

Use the ZL303XX_PTP_EVENT_CONTRACT_GRANTED and ZL303XX_PTP_EVENT_CONTRACT_EXPIRED events to transfer contract data to the standby node. This data can be used on the standby card to manually create contracts when it becomes the active.

Refer to "PTP Event Notifications (Unicast Negotiation)" for Event details.

13.2.2 Active Slave Node

13.2.2.1 MULTICAST

The user application can optionally save the sequenceId of all messageType using `zl303xx_PtpStreamConfigGet()`, and transfer this data to the standby node.

13.2.2.2 UNICAST NEGOTIATION

Use the ZL303XX_PTP_EVENT_CONTRACT_GRANTED and ZL303XX_PTP_EVENT_CONTRACT_EXPIRED events to transfer contract data to the standby node. This data can be used on the standby card to request new contracts when it becomes the active.

Refer to "PTP Event Notifications (Unicast Negotiation)" for Event details.

13.3 HOW TO SWITCH THE REDUNDANT MODE

13.3.1 Active Master to Standby

13.3.1.1 MULTICAST

Disable clock messaging, so no packets will be sent while this node is in standby. See function `zl303xx_PtpClockMessagingEnabled()`.

13.3.1.2 UNICAST NEGOTIATION

The active master may optionally cancel all contracts it has by calling `zl303xx_PtpContractCancel()`, but this is not recommended. The slave may choose to not negotiate a new contract with this port address.

Instead, the following steps should be executed:

1. Disable clock messaging, so no packets will be sent while this node is in standby. See function `zl303xx_PtpClockMessagingEnabled()`.
2. Call `zl303xx_PtpTxContractStop()` for all streams and messageType.
3. Call `zl303xx_PtpContractStatusGet()` for all streams and messageType.
4. Transfer all contract information to the standby node.
5. Optionally, delete all streams. See "Terminating a PTP Application or its Components" for details.

13.3.2 Standby Master to Active

13.3.2.1 MULTICAST

Add all streams that existed on the active master. Streams are not created dynamically in multicast mode, and there may only be a single multicast stream per PTP port. See "Configuration of a PTP Stream: zl303xx_PtpStreamCreate()" for details.

13.3.2.2 UNICAST NEGOTIATION

The active master may optionally cancel all contracts it has by calling `zl303xx_PtpContractManualCancel()`, but this is not recommended. The slave may choose to not negotiate a new contract with this port address.

Instead, the following steps should be executed:

1. Call `zl303xx_PtpContractTxStatusSet()` for all saved contracts
2. Enable clock messaging to allow packet transmit and receive. See function `zl303xx_PtpClockMessagingEnabled()`.

13.3.3 Active Slave to Standby

13.3.3.1 MULTICAST

1. Disable clock messaging, so no packets will be sent while this node is in standby. See function `zl303xx_PtpClockMessagingEnabled()`.
2. Delete all PTP streams. See "Terminating a PTP Application or its Components" for details.

13.3.3.2 UNICAST NEGOTIATION

1. Optionally, cancel all contracts with `zl303xx_PtpContractCancel()`.
2. Disable clock messaging, so no packets will be sent while this node is in standby. See function `zl303xx_PtpClockMessagingEnabled()`.
3. Call `zl303xx_PtpContractStatusGet()` for all streams and messageType.
4. Transfer all contract information to the standby node.
5. Optionally, delete all streams. See "Terminating a PTP Application or its Components" for details.

13.3.4 Standby Slave to Active

13.3.4.1 MULTICAST

Add all streams that existed on the active slave. Streams are not created dynamically in multicast mode, and there may only be a single multicast stream per PTP port. See "Configuration of a PTP Stream: zl303xx_PtpStreamCreate()" for details.

13.3.4.2 UNICAST NEGOTIATION

1. Optionally, call `zl303xx_PtpContractManualCancel()` to clear out contract data on the master.
2. Add all streams.
3. Enable clock messaging to allow packet transmit and receive. See function `zl303xx_PtpClockMessagingEnabled()`.

Chapter 14. Acronyms

- ABMCA: Alternate Best Master Clock Algorithm
- APR: Advanced Phase Recovery
- BC: Boundary Clock
- BMCA: Best Master Clock Algorithm
- CGU: Clock Generation Unit
- DCO: Digitally Controlled Oscillator
- DPLL: Digital Phase Lock Loop
- E2E: End-to-end
- FCL: Frequency Change Limit
- FDD: Frequency Division Duplex
- FPE: Flexible Protocol Engine
- GM: Grandmaster
- NCO: Numerically Controlled Oscillator
- NTP: Network Time Protocol
- OC: Ordinary Clock
- OS: Operating System
- P2P: Peer-to-peer
- PEC-M: Packet Equipment Clock Master
- PEC-S: Packet Equipment Clock Slave
- PLL: Phase Lock Loop
- ppb: Parts per billion (10E-9)
- ppt: Parts per trillion (10E-12)
- PSL: Phase Slope Limit
- PSN: Packet Switched Network
- PTP: Precision Time Protocol
- PTSF: Packet Timing Signal Fail
- QL: Quality Level
- SDK: Software Development Kit
- SF: Signal Fail
- SOOC: Slave-Only Ordinary Clock
- SPLL: Software Phase Lock Loop
- SyncE: Synchronous Ethernet
- RTP: Real-Time Protocol
- T-BC: Telecom Boundary Clock
- T-BC-P: Partial Support Telecom Boundary Clock
- T-GM: Telecom Grandmaster
- T-TSC: Telecom Time Slave Clock
- T-TSC-A: Assisted Partial Support Telecom Time Slave Clock
- T-TSC-P: Partial Support Telecom Time Slave Clock
- TC: Transparent Clock

- TDD: Time Division Duplex
- ToD: Time of Day
- ToP: Timing over Packet
- TSU: Timestamp Unit
- ZL: Zarlink
- ZLE: Zarlink Evaluation Board
- ZLS: Zarlink Software

Chapter 15. Change History

The following is a list of changes and improvements made to the ZLS30390 User's Guide for each of the software versions indicated.

15.1 RELEASE 5.3.8

- TS-4144 TS-4217 Updated Section 5.1.1 "PTP Init: Initializing a PTP Application" with new compile-time options:
  - ZL303XX_PTP_TIMER_TASK_DISABLE
  - ZL303XX_PTP_TIMER_TASK_USE_TICKDELAY
  - ZL303XX_PTP_TIMER_USE_HWTIMER
- TS-4144 TS-4258 Corrected Section 5.1.2 "PTP Init: Tasks" default task priorities and stack size.
- TS-4217 Updated
  - Table 9-1: Available PTP Logging Modules and Descriptions.
  - Section 10.5.3.1 "Transmit Function Binding" details for FAULTY state option.
  - Section 10.5.4.1 "Primary Receive Function Binding" for recommended alternative option.

15.2 RELEASE 5.3.6

- TS-4000 Updated PTP Port config alternateMaster definition and moved to Table 5-14.

15.3 RELEASE 5.3.4

- TS-2022 Updated PTP node deletion Section 5.7 "Configuration for Shutdown (Stopping) PTP Service".

15.4 RELEASE 5.3.0

- TS-2794 Added PTP stream state reporting APIs to Section 8.2.2 "PTP Stream State Reporting" and Table 8-11.
- TS-2053 Updated Acceptable Master Table and Acceptable Slave Table Section 5.3.3 "PTP Port: Acceptable Master and Acceptable Slave Tables".
- TS-2186 Fixed figure captions in Asymmetry correction Section 6.3.6 "Modify PTP Stream: Asymmetry Correction".
- TS-2330 Added PTP stream message counter Section 8.2.3 "PTP Stream Packet Count Reporting".
- TS-2152 Updated PTP event handler Section 8.1.1 "Event Handler" with implementation guidelines.

15.5 RELEASE 5.2.6

- TS-3083 Added pathTrace.propagateEnabled to Section 5.2.4 "PTP Clock: Profile-Specific and BMCA" in Table 5-6.
- TS-3082 Added Alternate Time Offset Indicator ATOI TLV configuration to

Section 6.1.18 "Modify PTP Clock: Alternate Time Offset Indicator ATOI TLV Data".

15.6 RELEASE 5.2.4

- TS-3080 Added Section Chapter 11. "Test Interfaces" and Table 11-1: API Routines for Testing.
- TS-3179 Added sdoId configuration for PTPv2.1 to Section 5.2.2 "PTP Clock: Local defaultDS" and Table 5-4.

15.7 RELEASE 5.1.0

- Added Section 5.3.2.4 "PTP Port: Default Profile Edition 3 (v2.1)" and Table 5-20.

15.8 RELEASE 5.0.6

- Added Section 1.1 "Status on Profiles in Development".
- Added configuration parameters for IEEE 802.1as in Section 5.2.4.4 "PTP Clock: Profile IEEE 802.1as" and Section 5.3.2.3 "PTP Port: Profile IEEE 802.1as".
- Added APIs for IEEE 802.1as Followup Information TLV in Section 6.1.17 "Modify PTP Clock: IEEE 802.1as Followup Information TLV Data", Section 6.2.6 "Modify PTP Port: IEEE 802.1as Neighbor Prop Delay Threshold", and Section 6.3.10 "Modify PTP Stream: IEEE 802.1as Message Interval Request TLV".
- Added ZL303XX_PTP_EVENT_FOLLOWUP_INFO_TLV_CHANGE event documentation in Section 8.1.2.7 "PTP Event Notifications (802.1as Profile)".

15.9 RELEASE 5.0.5

- Media Profiles additions
  - Added support for AES67 2016 Profile
  - Added support for SMPTE 2059-2 2019 Profile
  - Added support for AES R16 2016 Profile
  - Added support for SMPTE 2059-2 Sync Metadata TLV in Section 6.1.6 "Modify PTP Clock: Two-Step Flag"
- Updated delete APIs with fireEvent option (Section 5.7.1 "Terminating a PTP Application or its Components" and Section 8.1.2.4 "PTP Event Notifications (Create/Delete)")
- Updated ZL303XX_PTP_EVENT_PARENT_DATA_SET_CHANGE event documentation for new option fireParentDsChangeOnlyDelta (Section 8.1.2.3 "PTP Event Notifications (BMCA and Parent Updates)")
- Added ZL303XX_PTP_EVENT_PORT_CREATE event documentation (Section 8.1.2.4 "PTP Event Notifications (Create/Delete)")

15.10 RELEASE 5.0.3

- Renamed clockId attribute in zl303xx_PtpC37p238ClockConfigS to grandmasterId.
- Added set and get APIs for grandmasterId attribute in zl303xx_PtpC37p238ClockConfigS.
- Added set and get APIs for localTimeInaccuracy attribute in zl303xx_PtpC37p238ClockConfigS.
- Added set and get APIs for networkTimeInaccuracy attribute in zl303xx_PtpC37p238ClockConfigS.

15.11 RELEASE 5.0.0

- Added debug API zl303xx_PtpStreamTsRecordTableCopy
- Updated zl303xx_PtpC37p238ClockConfigS table and removed deprecated PTP Clock parameter localTimeZoneInfo.
- Updated zl303xx_PtpC37p238StreamConfigS table and removed deprecated PTP Stream parameter localTimeInaccuracyMax.

15.12 RELEASE 4.10.1

- Added optional checks for PTP port state FAULTY

15.13 RELEASE 4.10.0

- Added handling of PTP port state FAULTY

15.14 RELEASE 4.9.0

- Updated format, structure & template of the document

15.15 RELEASE 4.8.5

- Mux PTP streams to APR slots

15.16 RELEASE 4.7.2

- ITU-T G.8275.1 additions
  - Added support for ITU-T G.8275.1 Edition 2 profile
  - Updated support of offsetScaledLogVariance for ITU-T G.8275.1 edition 2 and G.8275.2
  - Added holdoverSupported Boolean for T-TSC clocks in ITU G.8275 to allow use of local holdover capability (i.e. use holdover clockClass in BMCA).
  - Updated handling of PTSF TRUE to treat Erbest as the empty set (as per ITU-T G.8275.1 Edition 2).
- ITU-T G.8275.2
  - Added support for ITU-T G.8275.2 Profile
  - Added support for ITU-T G.8275.2 Annex D TLV for PTP interface rate
  - Added a user option, 'settlingTimeActive', to enable a settling time after a reference switch or mode switch, during which time the egress Announce messages will assume PLL is in PHASE_LOCK.
  - Added enum for CableLabs R-DTI Profile, based on ITU-T G.8275.2
- Added support for Synchronization Uncertain flag (based on ITU-T G.8275 but available for all profiles).
- Added support for Acceptable Slave Table for all Unicast Negotiated profiles (previously supported only for ITU-T G.8265.1)

15.17 RELEASE 4.7.0

- Addition of PATH TRACE functionality (Refer to IEEE-1588-2008: Clause 16.2).
- Virtual Port configuration enhancements (including all Time Properties parameters, configurable clockIdentity and stepsRemoved).
- Inclusion of ITU G.8275.1 – Edition 2
- Enhancements to ITU G.8275.2 (Alternate Master Flag, Signal Fail)

15.18 RELEASE 4.6.3

- Ability to configure the stepsRemoved parameter for the ZLS30390 application.
- Addition of Acceptable Master Table 'dropped message' statistics.
- Enhancements in the G.8275.1 state machine including BMCA logging.

15.19 RELEASE 4.6.0

- Support for ITU-G.8275.1 (Telecom Profile for Phase)
- Addition of the UNCALIBRATED port state:
  - Refer also to IEEE-1588-2008: Clause 9.2.5, Clause 9.2.6.11 and Clause 9.2.6.13.
  - In past releases, ports went directly to the SLAVE state without entering the UNCALIBRATED state. In this release, ports may be kept in the UNCALIBRATED state to allow the local system to achieve a desired performance condition.
  - This mechanism is OFF by default but can be activated via the provided API.
- Addition of PTP Events to notify users when Leap-Seconds flags and/or UTC Offset (including UTC offset valid flag) change.
- Ability of users to over-ride field values in transmitted Announce messages.

15.20 RELEASE 4.5.0

- Addition of the Peer-Delay mechanism for path delay measurements:
- Asymmetry correction capability at the PTP Stream layer:
- Support for C37.238 (Power Profile):
- Addition of the PRE_MASTER port state:
  - Activated only when a PTP Clock is entering the M3 state (according to IEEE-1588-2008: Figure 26) and the recommended state of the associated port is MASTER (according to IEEE-1588-2008: Figure 23).
  - Refer also to IEEE-1588-2008: Clause 9.2.6.10
  - This mechanism is always ON. There is no API to enable/disable use of this state.

15.21 RELEASE 4.4.0

- Improvements to the logging for improved diagnosis
- Improvements to the OS_SIGNAL_HANDLER

Appendix A. Configuration of T-BC using the G.8275.1 Profile

A.1 HIGH LEVEL SYSTEM VIEW

The following diagram outlines the general logical flow for G.8275.1 operation:

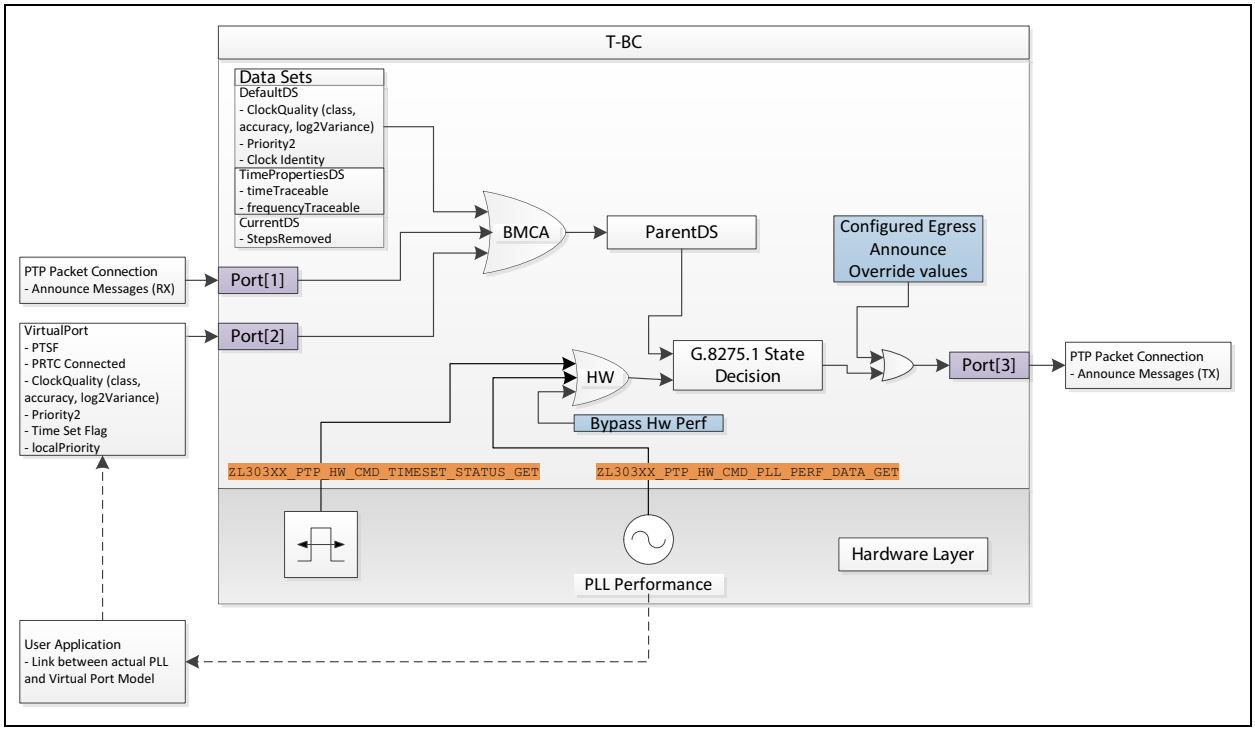

FIGURE A-1: *General G.8275.1 System Operation.*

A.1.1 Component Overview

- Local Data Sets (DefaultDS, TimePropertiesDS and CurrentDS):
  - These are standard data sets available on all configurations.
  - Refer to Configuration of a PTP Clock: zl303xx_PtpClockCreate() and Modify PTP Clock: Data Sets for configuration details.
- Input PTP Packet Ports:
  - These provide input ANNOUNCE messages to the T-BC from upstream master clocks.
  - This diagram shows a single instance but more may be added as required.
- Input Virtual Port (Optional):
  - Refer to Configuration of Virtual PTP Port (Optional) for structure and parameter descriptions.
  - This is an optional input interface intended to model the PLL electrical input as a PTP input to the G.8275 state machine (for example, the PLL may have a GPS reference and sync signal as its primary input).
  - The configured VP parameters are mapped to appropriate ANNOUNCE fields and provided to the BMCA module as if an ANNOUNCE message was

received on that port.

- User Application:
  - This monitors the status of the PLL electrical input and updates the Virtual Port configuration to accurately reflect the Electrical signal(s).
  - For example, should the input fail, the PTSF of the Virtual Port should be set to TRUE to invalidate the input so that a packet reference is selected instead.
- Hardware Layer Interface:
  - Whether or not the system has selected a PTP packet input or a Virtual Port input for a modeled electrical source, the profile state machine requires information about the status of the PLL.
  - Two key items are:
    - Time-of-Day status:
      - The profile waits for the local ToD to be set in order to avoid doing a Step-Time and impacting downstream clients.
    - PLL Performance status:
      - Refer to Interface Command: Sync Performance Data (Ingress) for parameter details.
      - The data from this input may be ignored by enabling the "bypass" mode. In this case, the PLL status is assumed to be PHASE LOCKED.
  - These inputs impact what values are advertised in ANNOUNCE messages to T-TSC nodes.
- BMCA Selection:
  - As with other profiles, this takes inputs from the local clock (DefaultDS) and PTP Masters (ANNOUNCE Messages) but also from the Virtual Port.
  - This selects the best input according to the profile selection algorithm and updates the ParentDS.
- G.8275 Engine:
  - This component decided what information to advertise to downstream clients.
  - It takes as inputs:
    - The best server source from the ParentDS.
    - The ToD input from the system.
    - The PLL performance status from the hardware.
- Egress ANNOUNCE Override:
  - Refer to Modify PTP Stream: Override Egress ANNOUNCE Message Fields for parameter details.
  - This layer is available to all profiles and allows the user to force the value of an egress ANNOUNCE field.
  - It is not recommended typically as settings are not validated by the software.

A.2 OPERATION

The typical operation for the G.8275 system above is:

- Create a T-BC clock using the G.8275 profile:
  - Refer to examplePtpTelecomPhaseG8275p1Bc() for details.
  - profile = ZL303XX_PTP_PROFILE_TELECOM_G8275_1
  - clockType = ZL303XX_PTP_G8275p1_CLOCK_TYPE_T_BC
- Add any PTP Packet Ports as required:
  - Refer to Configuration of a PTP Port: zl303xx_PtpPortCreate() for creating

PTP Ports and Streams.

- Add a Virtual Port to model any electrical reference + sync to the PLL
  - Refer to Configuration of Virtual PTP Port (Optional) for creating Virtual Ports (`zl303xx_PtpVirtualPortCreate()`).
- Update Virtual Port configuration to reflect the electrical input as changes are detected.
  - Individual parameters associated with the VP may be updated as follows:
    - Read the current VP configuration (`zl303xx_PtpVirtualPortGet()`).
    - Change the required members of the structure.
    - Write this new configuration back to the VP (`zl303xx_PtpVirtualPortSet()`).
  - To indicate that the VP input has experienced a Reference Failure, the PTSF flag may be updated individually.
    - Use the `zl303xx_PtpVirtualPortPtsfSet()` command to toggle the flag.
  - Key parameters include:
    - ptsf
    - prtcConnected
    - clockQuality (class, accuracy, variance).
    - localPriority
    - timeSetFlag
    - timeTraceable
    - frequencyTraceable

The BMCA will select the best master (electrical or packet) while the G.8275 engine will monitor the PLL performance and set the egress ANNOUNCE fields as required.

- When the PLL is in a PHASE LOCKED state, the egress ANNOUNCE messages will reflect the ParentDS data provided by the BMCA selection.
- When the PLL is anything less than PHASE LOCKED, the egress ANNOUNCE message fields will be updated by the G.8275 engine as per the ITU-T G.8275 Standard. Refer to the following Tables in particular:
  - ITU-T G.8275.1: Table 2 – Applicable clockClass values.
  - ITU-T G.8275.1: Table V.3 – T-BC Announce message contents
- If egress ANOUNCE overrides are configured, they will be applied after the G.8275 engine.

A.3 MONITORING CONNECTIONS

In some cases it may be required that a port is dedicated to the upstream (master) direction and that the node should never send ANNOUNCE or SYNC messages on a port. For example, in the diagram above Port[1] may be dedicated to a PTP master and even when the Virtual Port is selected (Port[2] becomes the SLAVE port), it is desired that Port[1] enters the PASSIVE mode rather than MASTER (as it might normally).

At the same time it may also be desirable to continue to monitor the timing packets (SYNC and DELAY) on the packet port so that the PLL can acquire faster if a switch to the port occurs.

To configure this scenario, use the following two steps:

1. Block the target port from entering the MASTER state when another port is the SLAVE:

- Issue the following command:

```
zlStatusE zl303xx_PtpPortDenyServiceRequestsSet(
  zl303xx_PtpPortHandleT portHandle,
  zl303xx_BooleanE bDenyServiceRequest)
```

- This will signal the state machine that the port cannot be a MASTER and therefore set it to PASSIVE.

2. Inform the port to continue collecting timing data:

- Because the local port is forced to PASSIVE it will not issue ANNOUNCE messages. The consequence is that the far-end port will determine itself to be better and maintain MASTER state.
- Because the far-end is MASTER, it will issue ANNOUNCE and SYNC messages that will be received by the local port.
- Typically in this state, the local port will not issue DELAY_REQUESTS. However, this can be enabled by issuing the following command:

```
zlStatusE zl303xx_PtpStreamUniNegMonitorTimingSet(
  zl303xx_PtpStreamHandleT streamHandle,
  zl303xx_BooleanE bUniNegMonitorTiming)
```

- Even though the command was originally intended for unicast negotiation monitoring, it has now been extended for G.8275.1 monitoring as well.

A.4 STATE EVALUATION

A.4.1 Clock State Evaluation Logic

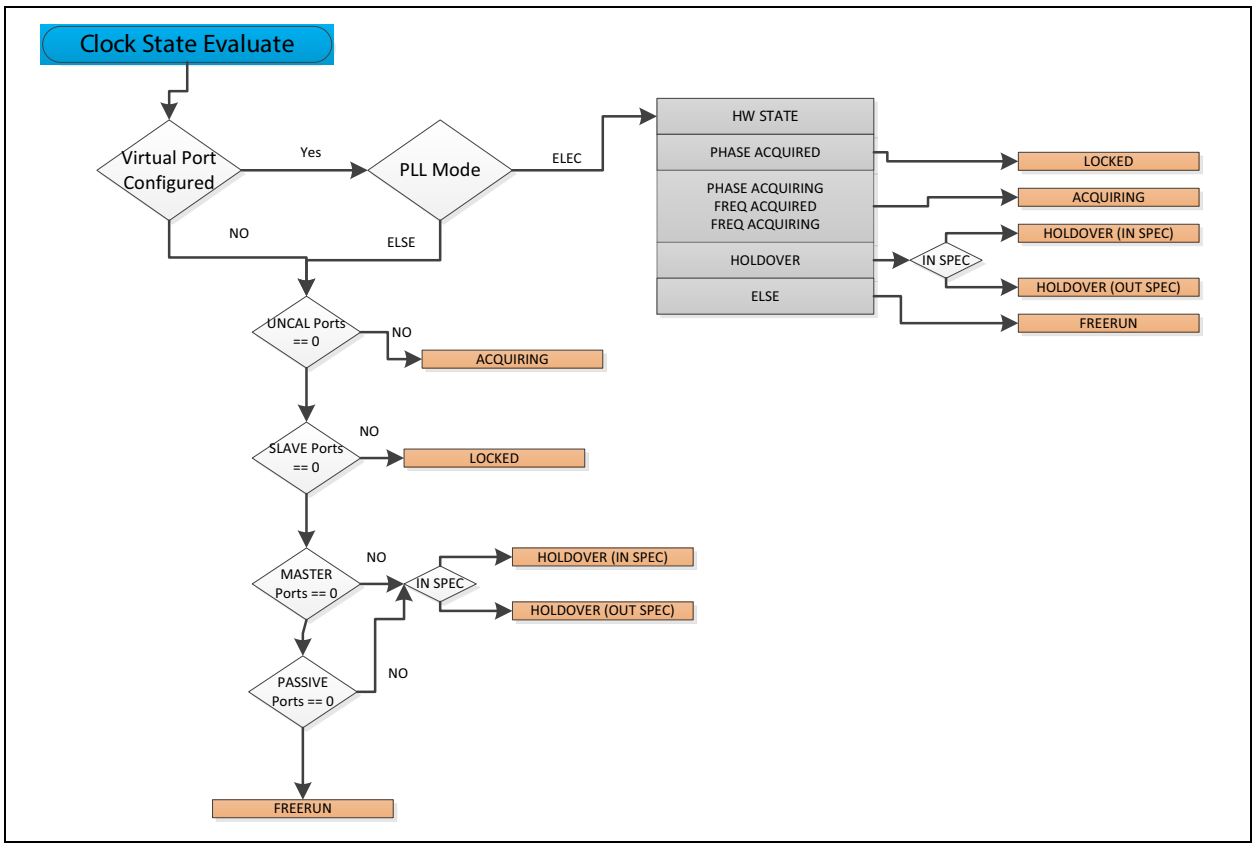

FIGURE A-2: *G.8275.1 Clock State Evaluation.*

A.4.2 PLL Status Evaluation

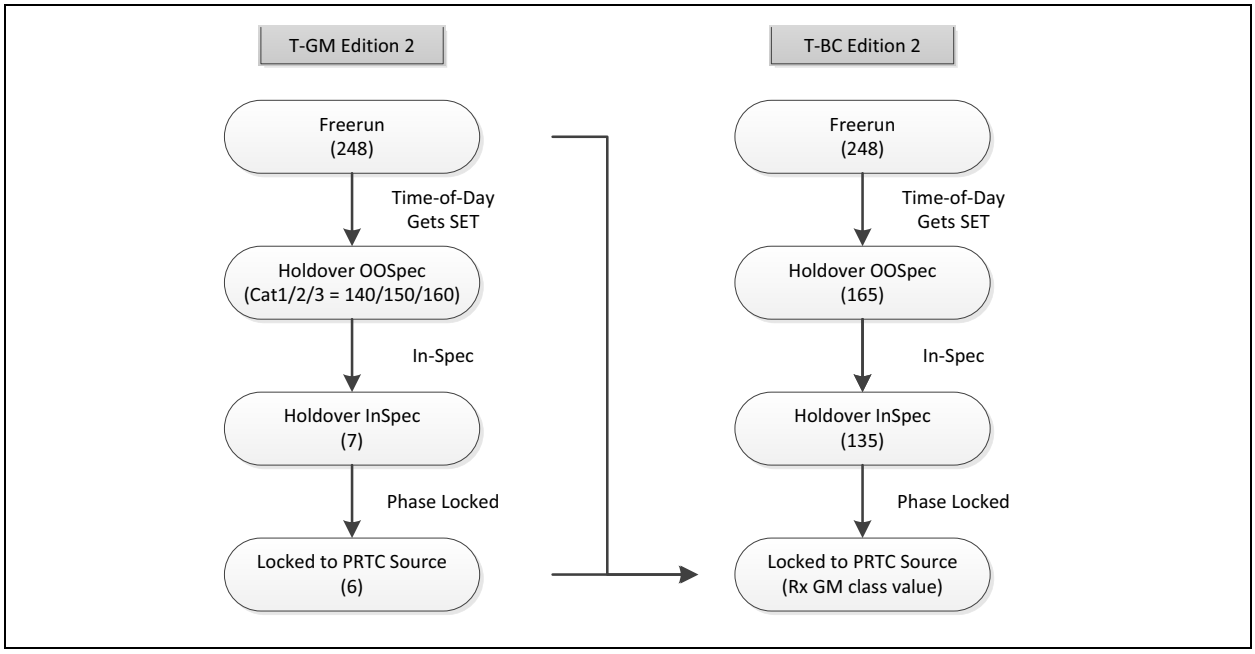

FIGURE A-3: *G.8275.1 PLL Status and clockClass Values.*

A.5 DEBUGGING

The following debug routines may be used to troubleshoot G.8275.1 system operation.

TABLE A-1: G.8275.1 DEBUG AND TROUBLESHOOTING ROUTINES

API Routine	Description
zl303xx_ DebugClockDataSet	To display the local configured Clock data including: • DefaultDS • TimePropertiesDS). Also displays the ParentDS as selected by the BMCA engine. This may reflect either a packet source ANNOUNCE data or a VP data.
zl303xx_ DebugPortDataSet	To display the local configured Port data. If the port corresponds to a Virtual Port, then all of the parameters of the zl303xx_PtpVirtualPortConfigS will be displayed.
zl303xx_DebugStreamEgressAnncShow	Displays the fields of the output ANNOUNCE message for the stream specified. This includes the values inserted by the G.8275.1 engine as well as any override values.

NOTES:

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-7288-4388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820