Cortina Systems® CS604x Optical Transport Processor & FEC Device Software Driver

User Guide

4 November 2011

Document Number 450371

Revision 5.5

Cortina Systems® CS604x Optical
Transport Processor & FEC Device
Software Driver User Guide
450371, Revision 5.5
4 November 2011

Cortina Systems, Inc. Confidential – Approval and NDA Required

This document contains information proprietary to Cortina Systems, Inc. (Cortina). Any use or disclosure, in whole or in part, of this information to any unauthorized party, for any purposes other than that for which it is provided is expressly prohibited except as authorized by Cortina in writing. Cortina reserves its rights to pursue both civil and criminal penalties for copying or disclosure of this material without authorization.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH CORTINA SYSTEMS® PRODUCTS.

NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT.

EXCEPT AS PROVIDED IN CORTINA'S TERMS AND CONDITIONS OF SALE OF SUCH PRODUCTS, CORTINA ASSUMES NO LIABILITY WHATSOEVER, AND CORTINA DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO THE SALE AND/OR USE OF CORTINA PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Cortina products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Contents

Revision History

Revision 5.5
Revision Date: 31 October 2011

- Section 5.7 was added to document a new high-resolution timer API.
- Section 8.7 was updated to include new information on the Amplitude Recentering functionality.

Revision 5.4
Revision Date: 31 August 2011

- Section 2.10 was updated to include compatibility information with previous CS600x/CS604x SW releases.
- Section 8.3.1 was added to explain sample interrupt code.
- Section 8.7 was added on elastic store amplitude recentering.

Revision 5.3
Revision Date: 20 July 2011

- Section 2.10 was updated to include compatibility information with previous CS600x/CS604x SW releases.
- Added Section 5.1.8 to describe the CUPLL APIs.
- Section 7.4.2 was updated for the OC-768 → OTU3(V) mapping, where a device fix has been implemented in the CS604x devices for a SADECO issue with the CS600x devices. 7.4.2 was split into two sub-sections: 7.4.2.1 for the CS600x and 7.4.2.2 for the CS604x.

Revision 5.2.1
Revision Date: 20 June 2011

- Section 2.9 was updated to indicate that the Socket Server is part of Release 5.2.1.
- Section 2.10 was updated to include compatibility information with previous CS600x/CS604x SW releases.

Revision 5.2
Revision Date: 16 May 2011

- Section 2.9 was updated to indicate that the Socket Server is not part of Release 5.2.
- Section 2.10 was updated to include compatibility information with previous CS600x/CS604x SW releases.

Revision 5.1
Revision Date: 4 March 2011

- Section 2.9 was updated to indicate that the Socket Server is not part of Release 5.1.
- Section 2.10 was updated to include compatibility information with previous CS600x/CS604x SW releases. It also contains a description of the intent behind the "_t41" suffix on the new Release 5.x API functions.
- Section 3.5 includes four new debug functions
- Section 4.4 was updated to include a listing of the most useful API functions for provisioning traffic, based on API usage in the full.pl Perl scripts used on the Evaluation Platforms.

- Section 5.1.3.1 was updated to include new Fiber Channel and 40GE traffic.
- API parameters were updated in Sections 7.5.4.2 and 7.6.10.2
- Infiniband is not supported in Release 5.1. This is noted in Sections 7.5.7 and 7.6.12.
- Section 8.6 was added to describe a fiber pull recovery procedure and interrupt handling routines.
- Changed references of “T40” to “CS600x”.
- Changed references of “T41” to “CS604x”.

Revision 5.0
Revision Date: 23 December 2010

- Most instances of CS600x were replaced with CS604x.
- References to the CS600x Data Sheet were not updated. They will be in the next revision of this document.
- Many abbreviations were added to Table 1.
- The directory structure of the SW release was updated to show T41 instead of TENABO. It also shows the t41_registers.h file instead of ten_registers.h.
- The Socket Server and Client is not part of Release 5.0.
- The Cleanup PLL is mentioned where the external SiLabs was referenced before.
- The CFP interface is mentioned along with the XFP interface.
- The Perl-based FEC functions were removed from Section 6.2 and replaced by a reference to the new High-Level functions that include FEC provisioning.
- The mappings in Section 7 were updated to reflect new T41 API functions and T41 device capabilities.
- Datasheet references were updated from the CS600x to the CS604x Datasheet.

Revision 4.3
Revision Date: 13 December 2010

- Section 2.6.2 was updated to include all of the latest source directories in the distribution.
- More compile-time options were added to Section 2.7.2.
- Section 3.5 was added to describe the new debug APIs that dump register settings.
- Updated Section 5.1 to reflect changes in provisioning sequence.
- Section 5.6.3.2 documents the new High-Level KPGA and HSIF KPGA APIs.
- Section 7.4.2 was added to describe the SW workaround for a device issue that effects OC-768 to OTU3 mapping.
- An issue with muxponder idle channels was documented in Section 7.5.6.2.
- Full.pl was removed from Appendix B since it is available in the same source code distribution as this file.

Revision 4.2
Revision Date: 31 August 2010

- Added description for XCON loopbacks.
- Added several new API descriptions: ten_hl_config_pbert, ten_hl_config_8gfc_enh_odtu23, ten_hl_config_8gfc_enh_otu2, ten_hl_config_pbert_8gfc_odtu23, ten_hl_config_otu2_cc_loopback, ten_hl_config_otu2e_odtu23, ten_hl_config_otu1e_odtu23, ten_hl_config_10ge6_2_odtu23, ten_hl_config_10ge7_2_odtu23, ten_hl_config_10ge7_2_otu1e, ten_hl_config_oc192_async_odtu23, ten_hl_config_pbert_8gfc_otu2, ten_hl_config_8gfc_ra_otu2, ten_hl_config_sfi41, ten_hl_config_xaui
- Reordered traffic type sections to group similar types together.

- Changed PBERT description from PERL-based to C-based call.
- Removed descriptions of obsolescent perl APIs.
- Moved text in “Functional Description” sections for some of the traffic function API descriptions to match format of other traffic type APIs.
- Updated the “Warm Initialization” procedure.
- Removed description of ten_hl_hsif_config_xaui, since ten_hl_config_xaui supersedes it.
- Made a number of formatting corrections.
- Changed references of “T40” to “CS600x”.
- Updated full.pl appendix.

Revision 4.1
Revision Date: 21 June 2010

- Rearranged chapter order.
- Completely revised Sync/Desync section. Added 5 new APIs, and revised the two remaining ones (get_fec_parameters and the configuration routine).
- Added several new traffic functions: ten_hl_config_40g_monolithic, config_otu2e_odtu23, ten_hl_config_10gfc_tc_odtu23, ten_hl_config_10gfc_otu2e, ten_hl_config_10gfc_otu1f, ten_hl_config_10ge_10ge
- Added section listing supported traffic types and FEC rates (Section 5.1)
- Revised the bring-up section (Section 9.0) to include changes to Cortina configurations and APIs.
- Revised Appendix B to reflect the new version of the reference script.
- Clarified cs_rtos.h description.
- Revised fractional divider descriptions and API parameter explanations.
- Added section 2.10 to discuss lack of backward compatibility.
- Revised/expanded description of Lock Detect Filter Control APIs (Section 7.2.1)
- Revised definition/description of ten_hl_config_xfi API.
- Added section for XAUI (section 7.2.5)

Revision 4.0
Revision Date: 8 April 2010

- Complete revision to accompany SW release 4.0.
- Rearranged chapters.
- Added high-level API descriptions.
- Added chapters for clocking, sync/desync, fractional dividers.
- Added sections describing traffic flow APIs.

Revision 3.1
Revision Date: 12 November 2009

- Added a note in Section 4.3.9.1 about XFI microcode registers.

Revision 3.0
Revision Date: 5 October 2009

- Added Sections 4.3 to 4.13 that go into more detail about provisioning the device for traffic.
- Added Section 11.0 (Appendix A) that documents the API calls for Config 12x.

Revision 2.2
Revision Date: 31 August 2009

- In Section 1.1 the distribution file suffix was changed from .tgz, which confused some tools, to .tar.gz.
- Removed configurations from Section 4.2 that are not available as part of the formal release package. Configurations 13-21, 23, 25, and 30-32 were removed.

Revision 2.1
Revision Date: 14 August 2009

- In Section 1.2 Visual Studio was removed as a supported compiler.

Revision 2.0
Revision Date: 10 July 2009

- The script organization in Section 4.1 was updated to reflect that script logs are now being distributed
- Added configs 12x, 30, 31, and 32 to Section 4.2.
- Section 5.7.3.1 was updated to clarify some of the 10GE Performance Monitoring statistics. Some sub-sections were renumbered.

Revision 1.9
Revision Date: 5 June 2009

- Added abbreviations in Section 1.3.
- The sample code was removed from Section 3.8 in lieu of pointing to the Perl scripts described in Section 4.0.
- Added descriptions of the Perl script environment to Section 4.0 and reorganized Section 4.
- Added top-configs 17-21 and 23-27 to Section 4.3.
- Added a description of the source code “modules” directory to Section 7.2.
- Added Section 10 to describe the Socket Client and Server.

Revision 1.6
Revision Date: 30 April 2009

- Added configurations 13-16 to Section 4.
- Updated Performance Monitoring details in Sections 5.6.2 and 5.7.1.5 to reflect the resolution of bug #12830.
- Updated Performance Monitoring details in Section 5.6.3.1 to reflect the resolution of bug #13216.
- Added details on Performance Monitoring functions for GFP to Sections 5.6.3.4 and 5.7.3.2.

Revision 1.5
Revision Date: 27 March 2009

- Added Sections 1.1 and 1.2 on the release distribution and supported compilers (the information was previously in the Release Notes).
- Added Sections 2.5 and 2.5.1 on the GFEC numbering.
- Enhanced Section 3.8 to add details from the top-config Perl script.
- Added Section 4 on the top configurations.
- Added details on Performance Monitoring functions for 10GE.

Revision 1.4 **Revision Date: 11 March 2009**
Adding details on Performance Monitoring functions for OTN and SONET/SDH.

Revision 1.3 **Revision Date: 30 January 2009**
Add explanations of the various compile parameters and miscellaneous updates.

Revision 1.2 **Revision Date: 19 December 2008**
Revised Interrupts Section.

Revision 1.1 **Revision Date: 28 August 2008**
Minor Revisions.

Revision 1.0 **Revision Date: 04 June 2008**
Initial Version.

1.0 Introduction

This document is the specification and user manual for software device driver library for the Cortina Systems® CS604x Optical Transport Processor & FEC Device. It describes in detail:

- The various components of the driver and the SW architecture.
- The driver features and the functionality it provides.
- The external interfaces for the driver.
- Porting and integrating the driver with customer software.

This document along with the driver software is provided to the customers of Cortina Systems, Inc. (Cortina) to help facilitate them in bringing up their hardware and software with minimal effort and enhanced flexibility.

The driver has been implemented in a very broad and general fashion so that it can be utilized for different customer applications and environments. As such, some functionality that is provided in the driver may not be used depending on the customer's specific needs. Also, the customer can opt either to integrate the driver software with their application software and RTOS, or to use it as a reference guide to develop their own driver to suit their specific application, embedded and RTOS environment.

The main features of the driver are:

- RTOS independent
- Written in Standard ANSI C and compiled with GNU-C compiler.
- Multithread and Multiprocessor safe.
- The code is structured, modular and well documented.
- Clean and simple interfaces.
- Provide various default configurations and diagnose the current running configuration.
- Support performance monitoring and interrupt handling.
- Provide extensive debug capabilities for use in bring-up and diagnostic applications.
- This driver can be easily integrated with the customer's systems and protocol software and ported to any embedded hardware and real-time operating system (RTOS).

1.1 Release Distribution

The release distribution is a Unix/Linux gzipped tar file (.tar.gz). It can be extracted on a Linux system with the following command, which will create a directory called “CS604x_Release_<major>.<minor>” with all of the files in it.

```
tar xzf CS604x_Release_<major>.<minor>.tar.gz
```

On a Windows machine, WinZip or 7-Zip can extract the files from the archive.

The files included in the distribution are listed below.

- There is Software Driver Manual in the “doc” directory that identifies all of the API functions and their parameters
- The register header file, which identifies all of the device’s registers and bit-fields, is in the “T41” directory; the register documentation will be available separately from the Cortina Customer Service (CCS) web site https://ccs.cortina-systems.com/.
- Source code is in the “T41/modules” directory
- Perl scripts for the Top-Level Configurations are in the “configs” directory
- Code that is common to all Cortina Systems SW drivers is in the “platform” directory; this code may need modifications for your system

1.2 Supported Compilers

This distribution has been tested against the GNU Compiler Collection (gcc) versions identified below.

- 4.1.0 for i686
- 3.3.2 for i686
- 2.95.4 for i686
- 2.7.2 for i686
- 3.3.3 cross-compiled for ppc_8xx on an i686

1.3 Abbreviations

The following abbreviations are used extensively in the Device Driver and documentation.

Table 1 Abbreviations

Abbreviation	Explanation
AIS	Alarm Indication Signal
AMP	Asynchronous Mapping Procedure (G.709 v3, 2009) – what we refer to as “asynchronous”
API	Application Programming Interface
BER	Bit error rate
BIP	Bit interleaved parity
BMP	Bit-synchronous Mapping Procedure (G.709 v3, 2009) – what we refer to as “synchronous”
CFP	A hot-pluggable optical transceiver- 100G Form Factor Pluggable
COR	Cortina Proprietary
Datasheet	CS604x Datasheet Revision 0.2
DT	Deterministic Justification (CS604x Datasheet

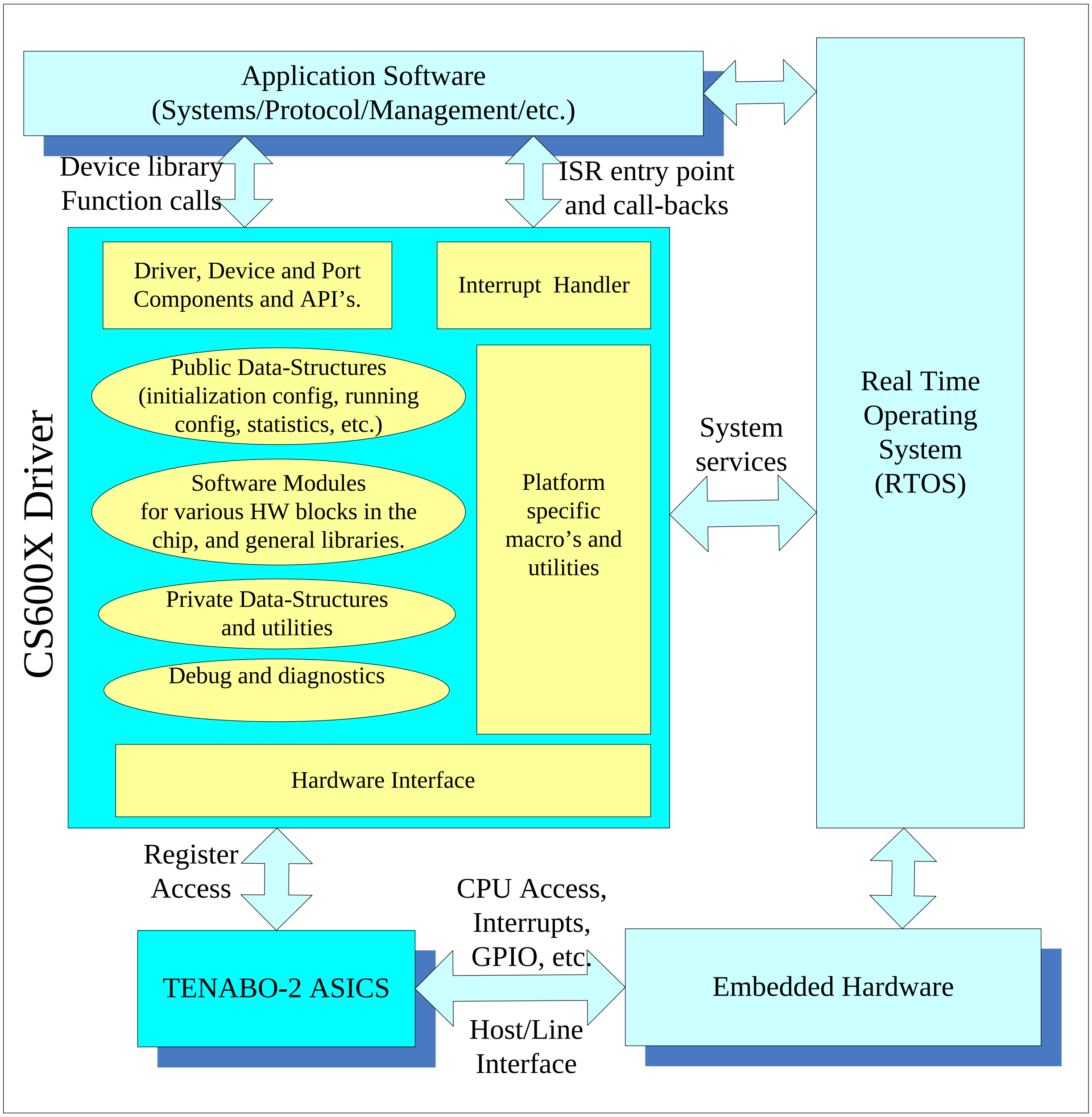
Abbreviation	Explanation
	Section 2.13.4 10G Class Wrapper (OxU2/flex Wrapper) and Section 2.13.7 ODTU23(e)/ODTU3(e).ts Wrapper for Aggregation over HO OxU3(e)
ES	Elastic Store
ETH	Ethernet
FC	Fiber Channel; Flow control
FRACDIV	Fractional Divider
GFEC	Generic Forward Error Correction
GFP	Generic Framing Procedure
GLB	Global
GMP_HO	Generic Mapping Procedure – High order (G.709 v3, 2009)
GMP_LO	Generic Mapping Procedure – Low order (G.709 v3, 2009)
HSIF	Host Interface
IB, IBA	Infiniband - a switched fabric communication link (signal rate 2.5,10,12.5 Gbps support)
IRQ	Interrupt
KPG	Known Pattern Generator
KPA	Known Pattern Analyzer
MAC	Media Access Control
MON	Monitoring
MLD	Multi Lane Deskew (A generic term used to describe OTL3.4 and STL256.4)
MPIF	Microprocessor Interface
MR	Multi-Rate SERDES
MSREGEN	Multiplex Section OH Regenerator (SDH) which includes first three overhead rows (A1, A2, J0, B1, E1, F1, D1-3)
N10G	Circuit Processor 10G
N40G	Circuit Processor 40G
OPU	Optical Channel Payload Unit (G.709 v3, 2009)
ODU	Optical Channel Data Unit (G.709 v3, 2009)
OUT	Optical Channel Transport Unit (G.709 v3, 2009)
PBERT	Packet BERT
PCS	Physical Coding Sub layer (IEEE 802.3)
PN11	2047-bit polynomial number 11 (PN-11) repeating sequence
PP10G	Packet Processor 10G
PP40G	Packet Processor 40G
PRBS	Pseudo Random Binary Sequence
PROP	Cortina Proprietary (Also referred to as COR)
RA	Rate Adjust/ Rate Adaptation
RSREGEN	Regenerator Section OH regenerator (SDH) which includes OH rows 5-9 (B2, K1, K2, D4-12, S1, M1, E2)

Abbreviation	Explanation
SADECO	Skew and Delay Compensator - bring the 10G frame boundaries into phase with a common frame boundary reference
SFU	Signaling Forwarding Unit - generates automatic consequent actions
SPOH	SONET Path Overhead
STATS	Statistics
SYNCDSYNC	Synchronizer/Desynchronizer
TBD	To be determined
TC, XC	Transcoding – covert 64B/66B blocks to 512B superblock
TRIB	Tributary
TS	Timeslots
UFEC	Ultra Forward Error Correction
VC-4-64C	16704 - column by 9 - row structure into which the 64B/66B coded Ethernet data stream is mapped
XAUI	10G Attachment Unit Interface (AUI)- 15 pin connection between MAC and PHY
XCON	Cross Connect – allows connecting side B slice to side A slice in CS6041
XFI	10G XFP Electrical Interface
XFP	10G Small Form Factor Pluggable optical transceiver
XGADJ	10G Rate Adjust
10GE6_1-10GE7-3	Methods to map 10GE client into OTN as per G.Sup43 (Nov 2006)

2.0 SW Organization, Architecture and Drivers

A typical system implementation incorporating the CS604x Transport Processor driver software might look as follows:

Figure 1 Driver Internals



2.1 Architecture

There are three main software components in the driver:

- Driver
- Device
- Module

The application software can communicate with these components via the Application Program Interfaces (APIs). These APIs are function calls which indicate to the driver what specific action is to be performed.

2.2 Driver

There is only one instance of the DRIVER. The DRIVER has to be loaded before using any of its services. When the DRIVER is loaded, it allocates resources and sets up the data structures needed by the driver software. The API to load the DRIVER is:

```
ten_drvr_load (void)
```

For any reason if the user application wants, it can unload the DRIVER component. Unloading the DRIVER will do the necessary clean-up - close any previously opened instances of DEVICE and MODULE components and free up any dynamic memory allocated by the driver software. The API to unload the DRIVER is:

```
ten_drvr_unload (void)
```

There are only a few DRIVER level APIs. These APIs may take none or at the most, one or two arguments and could potentially impact all the device and module level components.

2.3 Device

There can be multiple instances of the DEVICE depending on how many CS604x Transport Processor chips are being interfaced by the driver. Each DEVICE instance corresponds to a unique CS604x Transport Processor chip in the system. The current setting (TEN_MAX_NUM_DEVICES) is at 16. This can be changed at the customer's discretion provided that the system hardware and RTOS have the resources to support it.

Once the DRIVER has been loaded, each of the DEVICE instance needs to be registered by providing a base address for the DEVICE. This device registration is accomplished by calling the API:

```
ten_dev_register (cs_uint16 dev_id, cs_uint32 base_addr)
```

The driver uses read (CS_REG_READ) and write (CS_REG_WRITE) macros for register accesses, which take the physical address of the register as an argument. This physical address is computed by the driver by using the above registered base address for the specific DEVICE instance. The macros can be customized to use this address as the physical address to access the register or map it as desired before doing the access. This API binds the user provided dev_id with the device base address and the driver keeps track of it.

The DEVICE component also supports a method to un-register a particular DEVICE instance which will essentially delete the DEVICE instance. Any opened modules that belong to this DEVICE will be closed when it gets un-registered. This is accomplished by calling the API:

```
ten_dev_unregister (cs_uint16 dev_id)
```

After the device registration has been completed, the particular DEVICE instance can be initialized (see the Initializationsection for details).

The DEVICE initialization will initialize the associated data-structures in software and will configure the specified chip hardware as per the user provided configuration. After the initialization of the device is done, individual modules belonging to this device can be opened and configured. More description of this is provided later in this document.

The DEVICE component of the driver supports more APIs for the user to call. These APIs take the device id (dev_id) as the first parameter. In general, these APIs operate in a global scope as far as the underlying modules are concerned and so calling these APIs will impact the associated MODULE components.

Each instance of the DEVICE component is uniquely identified by dev_id of type cs_uint16. When the DEVICE instance gets registered, that is the time when the binding or mapping is made in the driver between a dev_id and the device's address space on the board. The application is responsible in ensuring that the dev_id values are unique.

2.4 Module

There are two instances of the module component associated with each DEVICE instance. In a general sense, the MODULE component corresponds to the physical A and B sides of the chip.

The MODULE component can be activated only after the corresponding DEVICE instance has been initialized and the associated device-level hardware configured. Firstly, the MODULE instance needs to be initialized and this is done by calling the API:

```
module_id = ten_mod_main_init (cs_uint16 dev_id, cs_uint8
module_num)
```

This creates the MODULE instance and returns a unique module handle (or module id) to identify the MODULE instance on successful completion. It also enables statistics and interrupt tables for using these APIs.

For any reason, if the user wants to delete an instance of the MODULE that was created previously, then this can be accomplished by calling the API:

```
ten_mod_close (cs_uint16 module_id)
```

More description of the module initialization can be found later in the document.

The module_id is of type cs_uint16 (i.e. the module handle) generated by the driver is based on the corresponding device id and the specific MODULE instance. The current driver implementation has module_id computed as follows:

```
module_id = (dev_id << 8) | module_num;
```

This module_id computation is internal to the driver software and the application software should not in any way interpret this module_id or derive any information from the module_id value. The application software is expected to use the driver provided module_id value as is when subsequently calling module-related APIs associated with this module.

The 40G APIs require only module_id; however for 10G APIs, an additional parameter is required. “Slice” represents a physical 10G slice and is numbered 0-3, since there are four 10G slices on each side of CS604x. So there is Module A with slices 0-3, which corresponds the first 0-3 instances in the register map and Module B with slices 0-3, which corresponds to the register map’s 4-7 instances.

The APIs will automatically calculate the correct register for accessing module A or B and also calculate the stride, which is the distance between the various 10G slices. This distance is different for the various blocks and is abstracted away by the APIs.

The APIs that operate on the entire device take dev_id as a parameter, like the cross connect and mpif. Most APIs operate on Module A or Module B and thus require the module_id parameter. Furthermore, the APIs that operate on 10G slices also require the “slice” parameter.

There are also some APIs that operate on registers with multiple “instances.” These are not the four physical 10G slices, but operate in a similar way: the instance number is given and the stride is automatically calculated.

Every API requires device id OR module id. Some APIs also require slice and/or instance to further specify which register is affected.

Hardware View	SW View	
Register Slice	Module	Slice
0	A	0
1	A	1
2	A	2
3	A	3
4	B	0
5	B	1
6	B	2
7	B	3

2.5 Exceptions

There are some non-intuitive exceptions to the device, module, port/slice, and instance mapping described above. These are described below.

2.5.1 GFEC

Refer to <u>CS600x Transport Processor Datasheet</u> document number 400486 Sections 2.1.1 and 2.5 for more information.

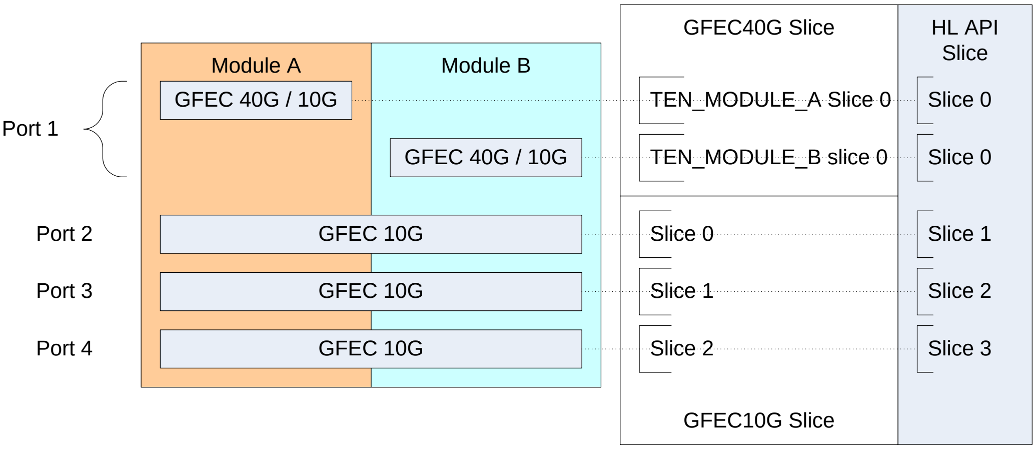
Mapping for the 10G GFEC is based on the figure below. The complexity of GFEC API port, module, and slice numbering can be summarized as:

- Port 1 GFECs run at 40G or 10G rates whereas the Port 2, 3, and 4 GFECs run at 10G only.
- Port 1 GFECs are in the GFEC40G block, even if it is running at 10G. Port 2, 3, and 4 GFECs are in the GFEC10G block. The CS604x registers are different for the GFEC40G and GFEC10G blocks, so this

makes defining common 10G API routines between Port 1 and Ports 2, 3, and 4 difficult.

- Port 2, 3, and 4 GFECs can be assigned to either Module A or Module B.

Figure 2 GFEC Slice Numbering



There is a difference between the way the Low-Level APIs and the High-Level APIs handle GFEC numbering. The Low-Level APIs are closely related to the GFEC40G and GFEC10G blocks, so Port 1, at 40G or 10G rates, is handled by GFEC40G APIs while Ports 2, 3, and 4 are handled by GFEC10G APIs. The High-Level APIs take a more user-friendly approach to provisioning the FECs by allowing the user to specify the module_id, the slice (0 – 3 for 10G and 0 for 40G), whether the FEC is 10G or 40G, GFEC or UFEC, and the UFEC overhead (which is not applicable for the GFEC).

For the Low-Level APIs, the rate for the two Port 1 GFECs is provisioned in the MPIF block and corresponding APIs, while the remainder of the GFEC provisioning is in the GFEC APIs. MPIF APIs use dev_id, not module_id, so the module (TEN_MODULE_A or TEN_MODULE_B) must be explicitly specified. The GFEC40G APIs use module_id as a parameter to indicate which of the two Port 1 GFECs is being accessed. The table below provides parameter examples for GFEC APIs.

Table 2 GFEC API Module Port, Module, and Slice Numbering

Port	Module	Rate	Low-Level APIs	High-Level API

1	A	40G	ten_mpif_gfec_cfg_port1(dev_id, TEN_MODULE_A, TEN_SLICE_40G)
			ten_gfec_40g_???(module_id for TEN_MODULE_A, …)
				ten_hl_mpif_fec_init(module_id for TEN_MODULE_A, TEN_SLICE0, TEN_FEC_GFEC_40G, …)
				ten_hl_fec_config(module_id for TEN_MODULE_A, TEN_SLICE0, TEN_FEC_GFEC_40G, …)

1	A	10G	ten_mpif_gfec_cfg_port1(dev_id, TEN_MODULE_A, TEN_SLICE_10G) ten_gfec_40g_???(module_id for TEN_MODULE_A, …)	ten_hl_mpif_fec_init(module_id for TEN_MODULE_A, TEN_SLICE0, TEN_FEC_GFEC_10G, …) ten_hl_fec_config(module_id for TEN_MODULE_A, TEN_SLICE0,

				TEN_FEC_GFEC_10G, …)

1	B	40G	ten_mpif_gfec_cfg_port1(dev_id, TEN_MODULE_B, TEN_SLICE_40G)
			ten_gfec_40g_???(module_id for TEN_MODULE_B, …)
				ten_hl_mpif_fec_init(module_id for TEN_MODULE_B, TEN_SLICE0, TEN_FEC_GFEC_40G, …)
				ten_hl_fec_config(module_id for TEN_MODULE_B, TEN_SLICE0, TEN_FEC_GFEC_40G, …)

1	B	10G	ten_mpif_gfec_cfg_port1(dev_id, TEN_MODULE_B, TEN_SLICE_10G)
			ten_gfec_40g_???(module_id for TEN_MODULE_B, …)
				ten_hl_mpif_fec_init(module_id for TEN_MODULE_B, TEN_SLICE0, TEN_FEC_GFEC_10G, …)
				ten_hl_fec_config(module_id for TEN_MODULE_B, TEN_SLICE0, TEN_FEC_GFEC_10G, …)

2	A	10G	ten_mpif_gfec_cfg_port2_4(dev_id, TEN_MODULE_A, TEN_SLICE0)
			ten_gfec_10g_???(module_id for TEN_MODULE_A, TEN_SLICE0, …)
				ten_hl_mpif_fec_init(module_id for TEN_MODULE_A, TEN_SLICE1, TEN_FEC_GFEC_10G, …)
				ten_hl_fec_config(module_id for TEN_MODULE_A, TEN_SLICE1, TEN_FEC_GFEC_10G, …)

2	B	10G	ten_mpif_gfec_cfg_port2_4(dev_id, TEN_MODULE_B, TEN_SLICE0)
			ten_gfec_10g_???(module_id for TEN_MODULE_B, TEN_SLICE0, …)
				ten_hl_mpif_fec_init(module_id for TEN_MODULE_B, TEN_SLICE1, TEN_FEC_GFEC_10G, …)
				ten_hl_fec_config(module_id for TEN_MODULE_B, TEN_SLICE1, TEN_FEC_GFEC_10G, …)

3	A	10G	ten_mpif_gfec_cfg_port2_4(dev_id, TEN_MODULE_A, TEN_SLICE1)
			ten_gfec_10g_???(module_id for TEN_MODULE_A, TEN_SLICE1, …)
				ten_hl_mpif_fec_init(module_id for TEN_MODULE_A, TEN_SLICE2, TEN_FEC_GFEC_10G, …)
				ten_hl_fec_config(module_id for TEN_MODULE_A, TEN_SLICE2, TEN_FEC_GFEC_10G, …)

3	B	10G	ten_mpif_gfec_cfg_port2_4(dev_id, TEN_MODULE_B, TEN_SLICE1)
			ten_gfec_10g_???(module_id for TEN_MODULE_B, TEN_SLICE1, …)
				ten_hl_mpif_fec_init(module_id for TEN_MODULE_B, TEN_SLICE2, TEN_FEC_GFEC_10G, …)
				ten_hl_fec_config(module_id for TEN_MODULE_B, TEN_SLICE2, TEN_FEC_GFEC_10G, …)

4	A	10G	ten_mpif_gfec_cfg_port2_4(dev_id, TEN_MODULE_A, TEN_SLICE2)
			ten_gfec_10g_???(module_id for TEN_MODULE_A, TEN_SLICE2, …)
				ten_hl_mpif_fec_init(module_id for TEN_MODULE_A, TEN_SLICE3, TEN_FEC_GFEC_10G, …)
				ten_hl_fec_config(module_id for TEN_MODULE_A, TEN_SLICE3, TEN_FEC_GFEC_10G, …)

4	B	10G	ten_mpif_gfec_cfg_port2_4(dev_id, TEN_MODULE_B, TEN_SLICE2)
			ten_gfec_10g_???(module_id for TEN_MODULE_B, TEN_SLICE2, …)
				ten_hl_mpif_fec_init(module_id for TEN_MODULE_B, TEN_SLICE3, TEN_FEC_GFEC_10G, …)
				ten_hl_fec_config(module_id for TEN_MODULE_B, TEN_SLICE3, TEN_FEC_GFEC_10G, …)

Statistics APIs are closely coupled to the device blocks, so the parameters shown in the table below should be used for the statistics APIs. The actual rate Port 1 GFECs are running is irrelevant for the statistics functions.

Table 3 GFEC Statistics APIs Parameters

Port	Module	module_id	block_id	sect_id (slice)
1	A	module_id for TEN_MODULE_A	TEN_ID_GFEC_40G	0
1	B	module_id for TEN_MODULE_B	TEN_ID_GFEC_40G	0
2	A	module_id for TEN_MODULE_A	TEN_ID_GFEC_10G	0
2	B	module_id for TEN_MODULE_B	TEN_ID_GFEC_10G	0
3	A	module_id for TEN_MODULE_A	TEN_ID_GFEC_10G	1
3	B	module_id for TEN_MODULE_B	TEN_ID_GFEC_10G	1
4	A	module_id for TEN_MODULE_A	TEN_ID_GFEC_10G	2
4	B	module_id for TEN_MODULE_B	TEN_ID_GFEC_10G	2

2.6 Driver Code Organization

2.6.1 Driver Package

The driver package consists of the following:

CS604x Transport Processor-specific driver source code (under T41 sub-directory).

Common top-level platform related source code where the basic data-types are defined, along with any RTOS-dependent defines are put in (under platform subdirectory). In general, any modifications needed by the user to customize the driver to their specific hardware and RTOS environment should be done here.

CS604x Transport Processor-specific documentation (under doc) which contains a PDF copies of the “Cortina Systems® CS604x Optical Transport Processor & FEC Device Software Driver User Guide” and “Cortina Systems® CS604x Optical Transport Processor & FEC API User Guide”.

2.6.2 Source Code Layout

The CS604x Transport Processor-specific driver code is organized in a tree directory structure ("T41" being the top-level). In this directory:

CS604x register header file (T41/t41_registers.h)

There is a top-level Makefile which will do a recursive make by invoking the module-level Makefiles.

All the block or module-specific source code (the *.[ch] files) which comprise the driver code are organized in the appropriate sub-directories. These sub-directories are under “T41/modules” and are named appropriately corresponding to the respective blocks.

“targets” sub-directory which will hold all the driver object files that get created when the driver library (binary) archive image gets built. The driver library archive itself after successfully building, can be found in “T41/<OS>/tenabo.a”.

The relationship between the functional blocks specified in the datasheet and the driver functions are as follows:

Debug Facilities	T41/modules/debug
Forward Error Correction	T41/modules/fec
Fractional Divider	T41/modules/fracdiv
Initialization and General Purpose	T41/modules/general
Global	T41/modules/glb
Global PLLs	T41/modules/gpllx1
HIgh Level Block-specific	T41/modules/highlevel
HIgh Level Traffic Provisioning	T41/modules/hl_config
High-speed Interfaces	T41/modules/hsif
Interrupts	T41/modules/irq
Microprocessor Interface	T41/modules/mpif
10G Circuit Processor	T41/modules/n10g
40G Circuit Processor	T41/modules/n40g
Overhead Processor	T41/modules/ohpp
10G Packet Processor	T41/modules/pp10g
40G Packet Processor	T41/modules/pp40g
Statistics	T41/modules/stats
Clocking and Synchronization	T41/modules/syncdsync
Cross connect and elastic store	T41/modules/xcon

2.7 Porting and Integration

The makefiles take a least-common-denominator approach to compiling for the different compiler versions that are supported. All advanced compiler features beyond ANSI C89 have been disabled so that the code will compile for any of these compilers without changes to the makefiles. The only exception is cross-compiling on a machine with different endianness than the target machine. In that case, the endianness test in Makefile.common should be overridden and the ENDIAN variable made equal to CS_BIG_ENDIAN or CS_LITTLE_ENDIAN based on the architecture of the target processor. Refer to Section 5 at the bottom of Makefile.common.

For other compiler options, refer to Sections 2, 3, 4, and 6 at the bottom of Makefile.common. Also refer to the definition of the make variable CFLAGS in Makefile.common.

2.7.1 Main Files for Porting

All platform and RTOS related methods are collected under “platform” directory so that the user can easily customize to suit their specific embedded hardware and RTOS environment. The main files in this directory are:

2.7.1.1 cs_types.h

Contains basic data-types that the CS604x Transport Processor driver uses.

2.7.1.2 cs_rtos.h

All the RTOS system-calls are put here as C-macros such that the rest of the source code can call these wrapper macros.

Some clarifications for cs_rtos.h items are described below:

2.7.1.2.1 cs_rtos.h Interrupts

If the user does not use interrupts, then functions/macros dealing with interrupts can be left blank (although they should still be defined). Also, the “modules/irq” directory contains an interrupt-tree walker for servicing interrupts. The data structure to support this is fairly large and is purely optional. The user can remove the directory from Makefile.common and use the ten_reg_read function to read interrupt registers directly if you prefer.

2.7.1.2.2 cs_rtos.h Semaphores

The driver is written so that it will work in a single thread, or, optionally, in a multi-threaded architecture. Thus we support semaphores. By itself provisioning the device can be done in a single thread. There are some software workarounds for device issues that may require a polling thread to handle specific events. Also, statistics gathering can be handled by a separate thread. For this reason, threads are useful, but not required. If threads are not required then you can leave the semaphore support stubbed-out.

If the driver uses multi-thread/multi-process execution for its own purposes, then the semaphore mechanism required is a counting mutex, because the same semaphore will be locked multiple times by one thread as the subroutines are nested. For example, function A locks a semaphore then calls function B which locks the same semaphore. The interrupt code can be compiled with or without the semaphore support. If called from an interrupt context you will not want to use the semaphore.

2.7.1.3 which_endian.c

This is a simple utility used at compile-time to figure out the endian-ness of the processor. If a cross-compiler is used, then one of the flags, CS_FLAG_BIG_ENDIAN or CS_FLAG_LITTLE_ENDIAN should be set to 1, depending on the target environment.

2.7.1.4 cs_utils.h

This has macro’s for the driver software to figure out what kind of platform (the physical layer) it is running on. By default, for the external customers, only the macro CS_IN_CUSTOMER_ENV () would return a TRUE value. The external customers of this driver software should not modify this file without consulting with Cortina, as the implications of any change is extensive in the way the driver operates.

2.7.2 Macros and Defines

As mentioned earlier, the driver provides a variety of macros and defines which are basically wrapper utilities that the user can easily customize to their environment. Some of them are the following:

Read/Write Access: All the register accesses in the driver code is basically done via CS_REG_READ () and CS_REG_WRITE () where the physical register address is passed in as one of the parameters. The customer is expected to implement the actual accesses based on their environment. For example, for memory-mapped I/O and using a flat-memory model, it will be just a matter of de-referencing the address (pointer) parameter.

Timer delays: The TEN_UDELAY and TEN_MDELAY are delay utilities to specify timer delay with micro-second or milli-second granularity.

Semaphores: The driver has macros to create, take, give and delete semaphores and these macros are called internally in the driver code. It is recommended that only mutually-exclusive semaphore implementation be used, which gives the flexibility of a single process/task to be able to call nested APIs. If there is no need for semaphore protection, then these macros could be left blank.

Dynamic memory utilities: There are macros for doing dynamic allocation, freeing of memory, copying memory, and setting memory. The driver code uses these macros for maintaining its internal data-structures and so the customer is expected to call the right memory library functions from inside these macros.

There are certain macros that the driver supports which are used at compile-time. These may be disabled or enabled by default (as indicated below), but when enabled provide additional flexibility to support the customer’s RTOS environment if needed. If needed, the customer can easily enable these macros by specifying it during the compile (defined in CFLAGS or as -D<macro-name> command line option), or defining them in platform/cs_types.h as shown below,

```
#define <macro-name>
```

These macros are:

CS_BIG_ENDIAN or CS_LITTLE_ENDIAN: Define this to select either big endian or little endian for the target CPU. Either (but not both) of the endian macros should be defined, or the compiling process will fail. This is automatically detected during the make process. However, for cross-compiling, it may necessary to override and manually set this in the file Makefile.common.

CS_DONT_USE_INLINE: The driver implementation uses some C in-line functions and this flag controls whether these in-line functions are supported by the C-compiler (flag = FALSE) or if they should not be used (flag = TRUE). By default, this flag is defined (TRUE) for maximum compatibility with C89 compilers.

CS_DONT_USE_STDLIB: This is provided so that in some customer applications, the standard C-libraries cannot be used so the driver will not use those system utilities. By default, this flag is not defined (FALSE).

CS_DONT_USE_VAR_ARGS: This is provided for some customer applications where the compiler does not support the handling of variable argument list. By default, this flag is not defined (FALSE).

CS_DONT_USE_VSNPRINTF: This is provided for customer applications where the compiler does not support the handling of a variable printf argument list. By default, this flag is defined (TRUE).

RELEASE_PLATFORM: Settings other than RELEASE_PLATFORM are used internally by Cortina Systems for debugging purposes and should be left as-is.

TENABO: This define is used internally by Cortina Systems and should be left as-is.

Linux: This define is used internally by Cortina Systems for the Linux lab3 build and is not needed for other types of projects.

TEN_IRQ_PRINT_INTERRUPTS is defined in modules/irq/ten_irq_priv.h. When it is defined as ‘1’ then the interrupt data structures will contain string representations of the interrupt registers and bit-field names. This will take significant memory so the feature can be disabled by defining this as ‘0’.

The file general/ten_bld_flags.h contains additional build-time options. These are:

INCLUDE_TEN_DEBUG_APIs, if defined, will include a significant amount of debug code that contains functions for dumping the registers and bit-fields in a human readable format. The functions operate on the block and slice level, or the entire device can be dumped with the ten_debug_dev_dump_settings(dev_id) function.

TEN_CHECK_FOR_BUS_ERRORS, if defined, will include a test for MPIF bus errors for every register read and write.

TEN_LOG_XFI_UCODE, if defined at compile-time and register read/write logging is enabled at run-time, then the microcode download will be included in the log. This can fill a log file. If this is not defined then the log will not be filled with microcode download data even if register read/write logging is enabled.

2.7.3 Integrating the Driver

The primary goal in the design, development and testing of the driver is to facilitate the customer in easily integrating it with the higher layer software components like System Software for device (interface) control and management, event handling, provisioning, accounting, and for use by upper protocol layers. As such, the APIs have been implemented such that they are re-entrant and are semaphore protected so that it can be used by more than one process or task in the system and/or in a multi-processor environment.

2.8 Getting Started

Cortina provides the customer with the source code for the driver and the associated PDF documentation for the APIs and also a copy of this document, as a tarred (archived) file called CS604x_Release_<release number>.tgz.

To extract the files and to build the driver library archive:

Copy CS604x_Release_<release number>.tgz to the desired location:

```
cp CS604x_Release_<major>.<minor>.tgz <path_to_install_dir>
```

Uncompress the files:

```
cd <path_to_install_dir>
```

Extract (un-archive) and restore the files:

```
tar xvf CS604x_Release_<major>.<minor>.tgz
```

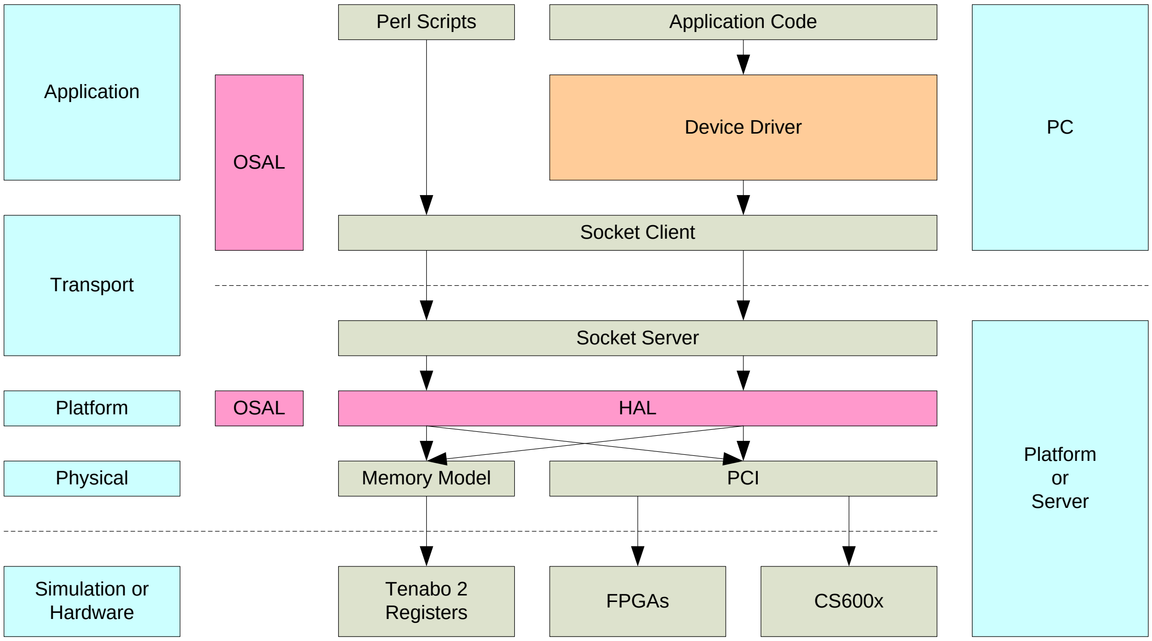
Build the driver library:

```
cd T41
make
```

2.9 Socket Server

The Socket Client/Server is an optional component of the CS604x release that provides a mechanism for lab testing. It is used as a communication path between the Cortina Systems® CS604x Evaluation Platform and a host computer that may be running the Device Driver.

Figure 3 Socket Client/Server Architecture



The Cortina Systems® lab environment is structured in layers, as shown to the left in the figure above. Those layers are, working from the bottom up:

- The *Simulation or Hardware Layer* represents the CS604x device and associated hardware, such as FPGAs. This layer can be implemented in one of two forms.
  - o The devices can be represented in virtual form, either through an RTL simulation of the device itself or a software memory model simulator.
  - o The devices can exist in physical form on a printed circuit board.

- The *Physical Layer* is the software that interacts with the Simulation or Hardware Layer. There are three types of physical layers.
    - The memory model simulator represents the CS604x device registers. The file socket_server/targets/bin/t41.init contains the reset values for all of the CS604x registers. These are loaded into an array and used to simulate the reads and writes of the device. This is not a functional model of the device, however. The memory model simulator is included in this release as part of the Socket Server.
    - The RTL/SW co-simulator ties the CS604x RTL to the Device Driver SW and the Perl scripting environment so that Device Driver reads and writes operate on the registers within the simulated device. Test programs can then verify that the API functions for traffic provisioning actually work as they are intended and that simulated traffic flows through the device. The co-simulation physical layer is not part of the CS604x SW release.
    - The actual CS604x device on a circuit board is the final physical layer.
- The *Transport Layer* is made up of the Socket Server and the Socket Client. There is a simple protocol between these two to enable the client to read and write registers.
- The *Application Layer* could be a simple register read/write program, like the socket_client application described below, or a more complex program that contains both the Socket Client and the CS604x Device Driver. The more complex version could use the API functions to interact with the CX604x device if the Device Driver is compiled in.

2.9.1 Building the Socket Server

The Socket Server will be built with the Memory Module Simulator. To build it:

```
cd socket_server
make
```

The executable will be socket_server/targets/bin/socket_server.

2.9.2 Building the Socket Client

There is an example Socket Client in the socket_server/client directory. This is a very simple application that connects to a Socket Server to read and write a few registers in the Evaluation Platform FPGA and in the CS604x device. It is built when the Socket Server is built, or it can be compiled manually:

```
cd socket_server/client
make
```

The executable will be socket_server/targets/bin/socket_client.

2.10 Compatibility

In general refer to the Release Notes with each release for any specific release incompatibilities.

Release 5.3 changed how CUPLL fixed clocks are implemented. Previous to Release 5.3 fixed clock was implemented with the CUPLL synthesizer (MPIF_CUPLL_LOCAL_FORCE.on=1). Starting with Release 5.3 the ten_hl_cupll_config_t41 no longer sets this local_force mode. Instead, it is expected that RX_DIV will be switched to an internal pilot clock (MPIF_RXDIV_CFG0.pilot must be set to the appropriate port number, MPIF_RXDIV_CFG0.mode=2) using the ten_mpif_rxdiv_cfg0_mode_t41 API.

Release 5.2.1 is generally compatible with previous CS600x releases and will automatically detect whether a CS600x or CS604x is in use. A CUPLL function that was defined in Release 5.2 has changed parameters, but all other functions should be compatible. Scripts using the CUPLL generated for Release 5.2 will require rework to be compatible with Release 5.2.1.

Release 5.2 is generally compatible with previous CS600x releases and will automatically detect whether a CS600x or CS604x is in use. Some functions that were defined in Releases 5.0, 5.1, and 5.1.1 have changed parameters.

Release 5.1 is generally compatible with previous CS600x releases and will automatically detect whether a CS600x or CS604x is in use. Some functions that were defined in Release 5.0 have changed parameters. New Release 5.0/5.1 functions will continue to be subject to change until they are frozen in Release 5.2. Function ten_gpllx1_analog_config was obsolete and was removed from the code. There are many new functions that end in “_t41”. This indicates that the function is new to the Release 5.x CS604x driver. These functions will also provision traffic in a CS600x device so that the same customer code base will operate with either the CS600x or CS604x devices.

Release 5.0 is generally compatible with Releases 4.3 and 4.2. Some functions had the enum type of a parameter changed but the integer enum values stayed the same. This shouldn’t impact compilers but it may be flagged by a static code analysis tool if customer code isn’t changed to match the new enums.

Release 4.2 added a number of new functions, and it has migrated most of the legacy perl code to C. However, it is generally compatible with release 4.1, although there may be some issues with old scripts (as it was not possible to retest all old scripts under the new release).

Some non-backward compatible changes were made as of Release 4.1. Release 4.0 scripts will not run against the Release 4.1 (or later) driver because syncdsync configuration code was moved out of some of the functions introduced in Release 4.0. Also, FEC parameters had to be added to some of the functions that were introduced in Release 4.0 to accommodate the “nofec” option.

For a similar reason, some 3.2 scripts may not run adequately under the 4.2 driver (e.g. 10GE scripts may have issues). Also, a number of APIs (particularly related to sync/desync, clocking, and fractional dividers) have had their names changed to reflect their migration from perl to C code source.

3.0 Driver Configuration

The application software can configure the various blocks in the ASSP during initialization. In addition, at a later time this configuration can be updated by the application software based on some event trigger or user request.

The initialization and the configuration of the hardware involve two steps. Firstly, the device needs to be globally initialized and configured. This configures all the shared sections in the chip that are common to both modules A and B. Secondly, each separate module in the device can be configured individually.

Driver APIs are provided to the application for the following purposes:

- Set or commit a configuration.
- Print and retrieve the run-time configuration read from the hardware.

3.1 Device Configuration Retrieval

The driver supports the capability for retrieving the current device configuration. These configurations are retrieved at run time directly from the ASIC, not from information cached in the device's private control block. For example:

```
ten_dev_cfg_t *p_cfg;
cs_status ret_val;
p_cfg = (ten_dev_cfg_t *) CS_MALLOC(sizeof(ten_dev_cfg_t));
ret_val = ten_dev_get_cfg (dev_id, p_cfg);
            …
CS_FREE(p_cfg);
return (ret_val);
```

Individual configuration block can also be retrieved by the block APIs in the following format:

```
ten_<block name>dev_get_cfg (cs_dev_id_t dev_id,
ten_<block name>_dev_cfg_t * p_cfg)
```

3.2 Device Configuration and/or Status Printing

The current configuration can be "printed" to the console (see CS_PRINT customization for details). These APIs are available in the following format:

```
ten_<optional block name>_print_cfg(cs_uint16 dev_id)
```

Use this API to print out the status of the block or the entire device:

```
ten_<optional block name>_print_status(cs_uint16 dev_id)
```

This API will print both the configuration and status of the block or the entire device:

```
ten_<optional block name>_print(cs_uint16 dev_id)
```

3.3 Module Configuration Retrieval

The driver supports the capability for retrieving the current module configuration. These configurations are retrieved at run time directly from the ASIC, not from information cached in the module’s private control block. The application should follow these steps:

```
ten_mod_cfg_t * p_cfg;
cs_status ret_val;
p_cfg = (ten_mod_cfg_t *)CS_MALLOC(sizeof(ten_mod_cfg_t));
ret_val = ten_get_cfg (mod_id, p_cfg);
…
CS_FREE(p_cfg);
return (ret_val);
```

Individual configuration block can also be retrieved by the block APIs in the following format:

```
ten_<block name>_get_cfg (cs_uint16 mod_id,
ten_<block name>_mod_cfg_t * p_cfg)
```

3.4 Module Configuration and / or Status Printing

The current configuration can be “printed” to the console (see CS_PRINT customization for details). These APIs are available in the following format:

```
ten_<optional block name>_print_cfg(cs_uint16 mod_id)
```

Use this API to print out the status of the block or the entire module:

```
ten_<optional block name>_print_status(cs_uint16 mod_id)
```

This API will print both the configuration and status of the block or the entire module:

```
ten_<optional block name>_print(cs_uint16 mod_id)
```

3.5 Printing Debug Information

Releases 4.2 and 4.3 added new APIs to print out the registers in the device in a human readable promat, using the symbolic register and bit-field names. This code is optional and must be compiled in by uncommenting the INCLUDE_TEN_DEBUG_APIS line in general/ten_bld_flags.h.

To dump the whole chip, use:

```
ten_debug_dev_dump_settings
```

Note that this will generate MPIF bus errors for any blocks that are powered-down,

The other debug functions are:

```
ten_debug_slice_dump_settings
ten_debug_gfec10g_dump_settings
ten_debug_gfec40g_dump_settings
ten_debug_ufec_dump_settings
ten_debug_hsif_dump_settings
ten_debug_hsif_slc0_mr10x4_dump_settings
ten_debug_hsif_slc1_mr10x5_dump_settings
ten_debug_hsif_slc2_mr10x4_dump_settings
```

```
ten_debug_hsif_slc3_mr10x5_dump_settings
ten_debug_mpif_dump_settings
ten_debug_glb_dump_settings
ten_debug_frac_div_dump_settings
ten_debug_gpllx1_dump_settings
ten_debug_spoh_dump_settings
ten_debug_n10g_dump_settings
ten_debug_n10g_oohr_dump_settings
ten_debug_n40g_dump_settings
ten_debug_n40g_oohr_dump_settings
ten_debug_ohpp_dump_settings
ten_debug_pp10g_dump_settings
ten_debug_syncdsync_dump_settings
ten_debug_xcon_dump_settings
ten_debug_xfi_dump_settings
ten_debug_cfp_dump_settings
ten_debug_cupll_dump_settings
ten_debug_pp40g_dump_settings
```

There is also a debug function that will clear all interrupts. It will walk the interrupt tree and clear all interrupts whether they are enabled or not. It will also enter blocks that are not clocked and will therefore generate MPIF Bus Errors.

```
ten_dev_irq_clear
```

4.0 Initialization

4.1 Device Initialization

The application can initialize the device hardware using the following API:

```
ten_dev_main_init (cs_uint16 dev_id)
```

As part of the device initialization, the driver will verify that the chip's JTAG-ID is correct. It will also accept the CS604x JTAG-ID (for upward compatibility). There are APIs to perform a hard reset on the various blocks if not in warm initialization mode (see section 3.7). The various soft resets are also asserted (if not in warm initialization mode), so that the blocks can be configured without disrupting the state machines during the provisioning process.

After the device initialization is done, the application can call any of the other device APIs to configure or reconfigure during run-time. There is also an API to perform an orderly removal of soft resets, which should be called when all configuration APIs are complete.

4.2 Module Initialization

Similar to the device initialization, the driver can initialize the module hardware using the following API:

```
ten_mod_main_init (cs_uint16 dev_id, cs_uint8 mod_num);
```

When the API gets successfully executed, it will create a valid module handle/id which is returned to the application. This handle is used in all subsequent module APIs.

As part of the module initialization, the driver will hard reset any blocks that are unique to this module if not currently in warm initialization mode (see section 3.7). The various soft resets are also asserted (if not in warm initialization mode), so that the blocks can be configured without disrupting the state machines during the provisioning process.

After the module initialization is done, the application can call any of the other module APIs to configure or reconfigure during run-time. There is an API to perform an orderly removal of soft resets, which should be called when all configuration APIs are complete.

4.3 Warm Initialization

The driver supports the following method for warm initialization:

TEN_WS_METHOD_TOP_DOWN: in this mode, the provisioning sequence from the higher layer software has to be identical to the normal (cold initialization) sequence. The driver will make sure that traffic will not be affected by avoiding resetting and writing to the device.

The procedure and sequence of API calls for warm initialization is the following:

```
1. New CPU needs to initialize without impacting
   existing traffic in the device
2. ten_dev_main_init(dev_id)
3. ten_mod_main_init(dev_id, TEN_MODULE_A)
4. ten_mod_main_init(dev_id, TEN_MODULE_B)
```

```
5. ten_dev_start_ws(dev_id, TEN_WS_TOP_DOWN)
6. Provision the device for the traffic that is
   currently running through it; device writes will not
   occur but data structures will be updated
7. ten_dev_stop_ws(dev_id)
8. Enable interrupts and performance monitoring
```

4.4 Sample Driver Initialization

Refer to the Perl scripts described in Section 9.0 for pointers to code samples.

The Perl script full.pl is used on the Evaluation Board platform for provisioning of traffic. It supports a wide range of traffic parameters. It also serves as a good indication of which API functions are the most useful for provisioning traffic, and therefore which functions should receive the most study to customers just learning the driver. These are the API functions in alphabetical order that are called from full.pl:

```
ten_cb_rates
ten_dev_is_t41
ten_dev_main_init
ten_drvr_ctl_logging
ten_glb_misc_xlos_inv_mra
ten_glb_misc_xlos_inv_mrb
ten_hl_config_10g_cbr_kpga_t41
ten_hl_config_10ge_10ge
ten_hl_config_10ge_otu2v_t41
ten_hl_config_10ge_otu3v_t41
ten_hl_config_10ge_xcon_loopback
ten_hl_config_10g_hsif_kpga
ten_hl_config_10g_wire_t41
ten_hl_config_40g_cbr_kpga_t41
ten_hl_config_40ge_otu3v_t41
ten_hl_config_40g_hsif_kpga
ten_hl_config_40g_monolithic
ten_hl_config_40g_wire_t41
ten_hl_config_aggregation_idle
ten_hl_config_fc_otu2v_t41
ten_hl_config_fc_otu3v_t41
ten_hl_config_fc_ra_fc
ten_hl_config_fec_t41
ten_hl_config_global
ten_hl_config_idle
ten_hl_config_oc192_kpga
ten_hl_config_oc192_otu2_kpga
ten_hl_config_oc192_otu2v_t41
ten_hl_config_oc192_otu3v_t41
ten_hl_config_oc192_xcon_loopback
ten_hl_config_oc768_kpga_t41
ten_hl_config_oc768_otu3v_kpga_t41
ten_hl_config_oc768_otu3v_t41
ten_hl_config_oc768_xcon_loopback
ten_hl_config_otu2_kpga
ten_hl_config_otu2v_otu2v_t41
ten_hl_config_otu2v_otu3v_t41
ten_hl_config_otu2_xcon_loopback
ten_hl_config_otu3v_kpga_t41
ten_hl_config_otu3v_odtu_otu3v_t41
ten_hl_config_otu3v_otu3v_t41
ten_hl_config_otu3v_t41
ten_hl_config_otu3_xcon_loopback
ten_hl_config_pbert_odtu23
ten_hl_config_pbert_otu2
ten_hl_config_remove_module_soft_resets (from configs.pm)
```

```
ten_hl_config_remove_soft_resets (from configs.pm)
ten_hl_fracdiv_config (from configs.pm)
ten_hl_gpllx1_waitfor_gpll_lock
ten_hl_gpllx1_waitfor_gpll_lock_40g
ten_hl_hsif_mr_switch_clock
ten_hl_mr_40g_config_clockmux
ten_hl_mr_config_clockmux
ten_hl_n10g_config_lom_from_traffic
ten_hl_n40g_async_dewrap_generic_40g_client_sw_workaround (from
configs.pm)
ten_hl_n40g_config_lom_from_bps
ten_hl_n40g_config_lom_from_traffic
ten_hl_ohpp_and_shadow_ram_init
ten_hl_resets_block (from configs.pm)
ten_hl_syncdsync_datapath_config_t41
ten_hl_syncdsync_reset (from configs.pm)
ten_hl_syncdsync_threadsafe_datapath_config
ten_hl_ufec_ed_config
ten_hl_xfi_config_clockmux
ten_hsif_provision_datapath
ten_hsif_set_clk_40g
ten_mod_main_init
ten_mpif_clock_select_gpll_in
ten_mpif_global_clock_disable_common
ten_mpif_global_reset_common
ten_mpif_global_reset_hsif (from configs.pm)
ten_mpif_global_reset_syncdsync (from configs.pm)
ten_mpif_global_reset_ufec
ten_mpif_set_clock_switch_force (from configs.pm)
ten_n10g_gblr_set_kpasel
ten_n10g_set_global_resets (from configs.pm)
ten_n40g_gblr4x_set_kpasel
ten_n40g_otnr4x_oacfg0_dsyhyst
ten_n40g_otnr4x_set_ncols
ten_n40g_otnr4x_set_nparb_npar
ten_n40g_otnt4x_set_ncols
ten_n40g_otnt4x_set_nparb_npar
ten_n40g_sdfr_select_global_timing_source (from configs.pm)
ten_n40g_set_global_resets (from configs.pm)
ten_reg_read (from configs.pm)
ten_reg_write
ten_syncdsync_config_1_to_1 (from configs.pm)
ten_syncdsyncrx_cfg (from configs.pm)
ten_syncdsyncrx_control_sreset (from configs.pm)
ten_syncdsyncrx_set_ac1 (from configs.pm)
ten_syncdsyncrx_set_ac2 (from configs.pm)
ten_syncdsyncrx_set_ad1 (from configs.pm)
ten_syncdsyncrx_set_ad2 (from configs.pm)
ten_syncdsyncrx_set_denominator (from configs.pm)
ten_syncdsyncrx_set_mode (from configs.pm)
ten_syncdsyncrx_set_numerator (from configs.pm)
ten_syncdsync_set_cb_10g_transponder_t41
ten_syncdsync_set_cb_40g_muxponder_t41
ten_syncdsync_set_cb_40g_transponder_t41
ten_syncdsync_thread_set_cb_10g_transponder
ten_syncdsync_thread_set_cb_40g_muxponder
ten_syncdsync_thread_set_cb_40g_transponder
ten_syncdsync_thread_set_cb_transcode
ten_syncdsynctx_cfg (from configs.pm)
ten_syncdsynctx_cfgtx_cfgpd0 (from configs.pm)
ten_syncdsynctx_control_sreset (from configs.pm)
ten_syncdsynctx_jcgen_configuration (from configs.pm)
ten_syncdsynctx_set_ac1 (from configs.pm)
ten_syncdsynctx_set_ac2 (from configs.pm)
ten_syncdsynctx_set_ad1 (from configs.pm)
ten_syncdsynctx_set_ad2 (from configs.pm)
ten_syncdsynctx_set_configuration_init_force (from configs.pm)
```

```
ten_syncdsynctx_set_configuration_rx_div_mux (from configs.pm)
ten_syncdsynctx_set_denominator (from configs.pm)
ten_syncdsynctx_set_mode (from configs.pm)
ten_syncdsynctx_set_numerator (from configs.pm)
ten_ufec_select_global_timing_source (from configs.pm)
ten_xcon_datapath_reset (from configs.pm)
ten_xcon_es_reset (from configs.pm)
ten_xcon_sadeco_reset (from configs.pm)
ten_xfi32x1_stxp0_tx_clkdiv_ctrl_ctv_div
ten_xfi32x1_stxp0_tx_clkdiv_ctrl_rdiv_sel
```

5.0 Global Functions

5.1 Clocking

The CS604x supports a wide variety of clocking options. Configuring the clocking for the chip can be an overwhelming task. It may involve the provisioning of external filters, VCOs, and phase detector devices, but is highly dependent on the system architecture. On the Cortina Systems® CS600x Evaluation Platform this involves programming the Silicon Labs* Precision Clock Multiplier/Jitter Attenuators. The Cortina Systems® CS604x Evaluation Platform will use the internal clean-up PLLs instead of the external PLLs.

With this release, clock configuration has been simplified via a number of high-level APIs. The user still must separately configure the GPLLs, fractional dividers, and sync/dsync, but each can now be done with one API rather than several.

5.1.1 Clocking Overview

5.1.1.1 Clock Configuration General Flow

The following steps show the general flow of clock configuration. This should give an idea of the number of things involved in configuring clocks. However, the user does not have to directly perform all of these tasks. The newer high-level APIs will do some of this automatically, based on the traffic, synchronization, etc.

This section gives background information. Implementation details will be shown in the bringup section.

- Setup interface TX reference clock
- Setup system reference clock
- Set first-level clock muxing
- Configure fractional dividers
- ConfigureSiLabs/Cleanup PLL
- Setup the sys GPLL and check for sys GPLL lock
- Setup internal pilot clock using internal fraction divider
- Setup TX protection clock using internal fraction divider
- Setup XFI/CFP SERDES and check for VCO tune
- Setup MR SERDES and check for VCO tune
- Setup GPLL and check for VCO tune
- Configure syncdsync
- Configure the rest of chip
- Adjust clock MUX for XFI/OTL3.4/STL256.4/XLAUI interface
- Adjust clock MUX for MR interface
- Switch inputs for GPLL and check for GPLL lock
- Remove software resets

- Check for XFI/CFPSERDES lock
- Check for MR SERDES lock

The goal is to configure the CS604x independent of the incoming traffic. The absolute requirement for the external clock is system reference clock. All TX reference clocks need to be stable after the device configuration and before carrying traffic. (Note that a port can be configured without carrying traffic.)

5.1.1.2 Detailed Explanation

Setup TX reference clock

There are two options to setup the interface TX reference clock. First, use the on-board clock oscillator. Second, use on-board PLL to generate the TX reference clock based on the output clock (RXDIV) from SYNCDESYNC.

Setup system reference clock

The internal system clock frequency range is 300 MHz – 425 MHz The on-board system reference clock frequency needs to be the same as the intended internal system clock frequency or one quarter of the system clock frequency. The minimum system clock frequency is 1000 ppm above the line rate.

Setup the sys GPLL and check for sys GPLL lock

Configure the first instance of GPLL to match the external system reference clock frequency. SYS GPLL must be locked before moving to the next step.

Setup internal pilot clock using internal fraction divider

There are 16 fractional dividers inside of the CS604x. The first eight dividers can be used as internal pilot clocks for each channel (four on the A side and four on the B side). The fractional divider should be programmed to the frequency that matches with the data rate of the channel. The same fractional divider can also be used to protect the RX data path clock in the case of fiber–pull.

Setup TX protection clock using internal fraction divider

The next eight fractional dividers inside of the CS604x can be used to provide TX protection clock. The frequency of the divider should be matched with the expected RXDIV clock that is generated by the SYNCDESYNC. During the configuration or fiber pull stage, the TX protection clock is absolutely necessary for recovery.

Setup SYNCDESYNC

See sync/desync section below.

Setup XFI/CFP SERDES and check for VCO tune

There are total four XFI/CFP SERDES in the CS604x.

1. Configure the divider and MUX inside of XFI SERDES to match TX reference clock.
2. Choose the fractional divider as the pilot clock
3. Force the RX SERDES to lock on the pilot clock
4. Force the TX protection clock onto RXDIV pin

5. Remove the SERDES reset and select the XFI/CFP output clock to source the next block
6. Check for TX XFI/CFP SERDES VCO tune
7. Check for RX XFI/CFP SERDES VCO tune
8. Check for RX XFI/CFP SERDES LOCK

Setup MR SERDES and check for VCO tune

This is similar to setting up the XFI SERDES. There are total eight MR SERDES in the CS604x.

Setup GPLL and check for VCO tune

There are total 17 GPLLs in the CS604x as identified below.

Table 4 GPLL Instances

Instance	GPLL
0	System
1 – 4	Side A Receive
5 – 8	Side A Transmit
9 – 12	Side B Receive
13 – 16	Side B Transmit

1. Force the RX SERDES provides the input clock to both RX and TX GPLL for that channel.
2. Configure the divider and MUX based on the channel protocol.
3. Check for VCO tune of the GPLL

Configure the rest of chip

At this point, all clocks from HSIF and GPLL block are tuned and locked to the internal pilot clock. The TX protection clock provides a stable frequency to external clocks. The rest of device is ready to be configured.

Adjust clock MUX for XFI/CFP interface

1. Switch the RX SERDES to lock on the incoming data.
2. Switch the internal pilot clock source to TX reference clock. This is due to a known bug inside of the XFI block.
3. Adjust the XFI/CFP SERDES dividers to match the TX reference clock.
4. Enable the TX protection switch so the RXDIV clock will switch to TX protection clock if the XFI/CFP SERDES RX lose lock. This is important for recovering from a fiber-pull.

Adjust clock MUX for MR interface

1. Switch the RX SERDES to lock on the incoming data.
2. Enable the TX protection switch so the RXDIV clock will switch to TX protection clock if the MR SERDES RX loose lock. This is important for recovering from a fiber-pull. If SFI4.1 is configured for the MR channel,

Both RX and TX protection should NOT be enabled. There is no SERDES for SFI4.1 application.

Switch inputs for GPLL and check for GPLL lock

Switch the input for TX GPLL to TX SERDES. The assumption is there is a stable TX reference clock.

Remove software resets

1. Remove all block soft reset.
2. Disable force TX protection clock to RXDIV. This step allows RXDIV taking clock from SYNCDESYNC block.

Check for XFI/CFP SERDES lock

Check SERDES lock status for all XFI/CFP channels that is enabled.

Check for MR SERDES lock

Check SERDES lock status for all MR channels that is enabled. If SFI4.1 is used, skip this step.

5.1.2 PLLs

5.1.2.1 Datasheet Sections

Section 2.14.1: Clocking

5.1.2.2 Key APIs

For release 4.1, GPLL configuration has mostly been moved inside the associated interface APIs. Therefore, there are no comprehensive high-level configuration APIs for the GPLLs. There are some specific APIs that may need to be called, which are listed below.

5.1.2.2.1 GPLL Lock-related APIs

These APIs wait for the GPLLs to lock; they will return CS_ERROR if lock is not achieved. There are two versions, one for 40G (which checks for 40G-required GPLLs) and one for 10G (which will only check for the specified slice).

```
ten_hl_gpllx1_waitfor_gpll_lock_40g
ten_hl_gpllx1_waitfor_gpll_lock
```

5.1.2.2.2 Other high-level APIs:

(These are lower-level APIs that do more specific configuration functions. It is possible – but not recommended – to use them instead of the more comprehensive APIs described above. For information on these, consult the API User's Guide.)

```
ten_hl_gpllx1_analog_config
ten_hl_gpllx1_check_lock
ten_hl_gpllx1_config_SYSGPLL
ten_hl_gpllx1_init
ten_hl_gpllx1_waitfor_vcotune
```

5.1.3 Traffic/FEC Parameters For Fracdiv/Syncdsync APIs

The syncdsync APIs (as well as the Fractional Divider APIs) require a precise specification of the traffic configuration. To simplify parameter passing, the Cortina SW usually generates special code words to describe traffic configurations. These are generated by routines like "combine_signal_type_and_fec". The user should employ the generator APIs to produce special traffic types, and avoid guessing at or hard-coding the special code words.

However, some basic types need to be passed to the combining routines; possible candidates are listed in the following sections.

5.1.3.1 Traffic Types (line, client)

The following lists shows many of the basic traffic types that may be available for the syncdsync and fracdiv routines.

Supported basic line parameters: otu1e, otu1f, otu2, otu2e, otu3, otu3e, otu3e2, oc192, oc768, 10ge, 10ge6_1, 10ge6_2, 10ge7_1, 10ge7_2, 10ge7_3, 10ge_ra, 10gfc, 10gfc_tc, 8gfc_enh, 8gfc_ra, 4gfc_ra, 2gfc_ra, 1gfc_ra, 10gelan, 40gelan, 40ge

5.1.3.2 FEC Rates

Supported FEC rate parameters: nofec, 0fec, gfec, ufec7p, ufec10p, ufec12p, ufec13p, ufec15p, ufec20p, ufec25p, ufec26p

5.1.4 Sync/Desync

The CS604x Transport Processor has the capability to synchronously or asynchronously map and demap four 10G clients or one 40G client. It employs the Synchronizer/Desynchronizer logic to adjust clock rates and provide justification opportunities. See datasheet section 2.14.2 for details.

Release 4.2 has enhanced sync/desync support, which calculates the sync/desync parameters based on traffic types. The addition of this new code adds support for different UFEC percentages, including "nofec" (aka ODUk). It also simplifies the passing of data between the sync/desync functions.

5.1.4.1 Supported Configurations

The following clocking diagrams show the supported sync/desync configurations. Elements inside of the dotted line are CS604x functions. The boxes shaded with diagonal lines represent the sync/desync logic for each side (client/line) and direction (RX/TX). If the sync/desync is omitted for a direction, that indicates that it is not used for that configuration.

5.1.4.1.1 40G Muxponder

Table 5 40G Muxponder Sync/Desync configurations

Client Mode	40G Transponder Sync/Desync Configuration
OC192 10GE7.1 10GE7.2	40G Muxponder Sync/Desync configuration 1 8GFC ASYNC+Deterministic

4GFC 8GFC 10GFC INFINIBAND	Rest SYNC+ASYNC
OTU2 OTU2e	40G Muxponder Sync/Desync configuration 2
	40G Muxponder Sync/Desync configuration 3
10GE7.3	40G Muxponder Sync/Desync configuration 4

Figure 4 40G Muxponder Sync/Desync configuration 1

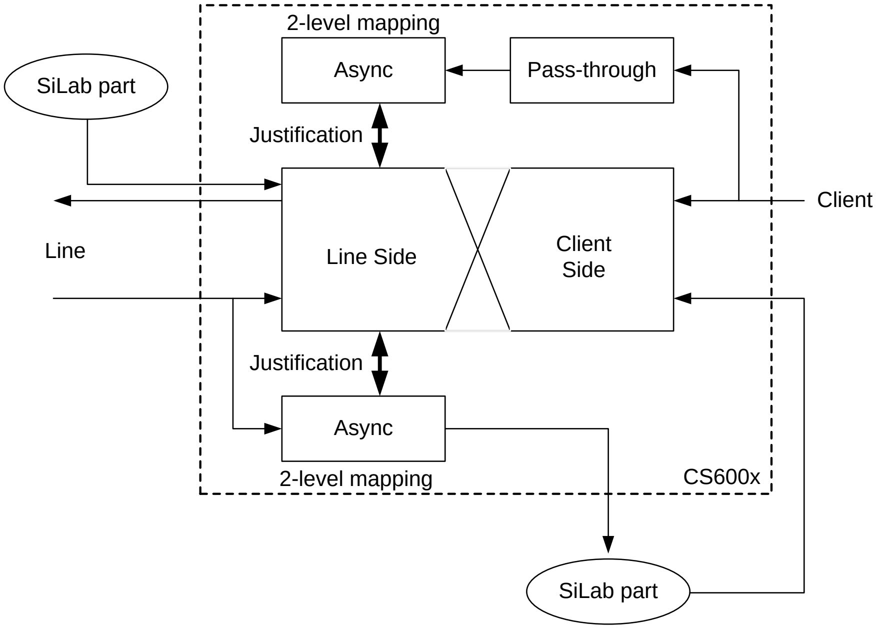


Figure 5 40G Muxponder Sync/Desync configuration 2

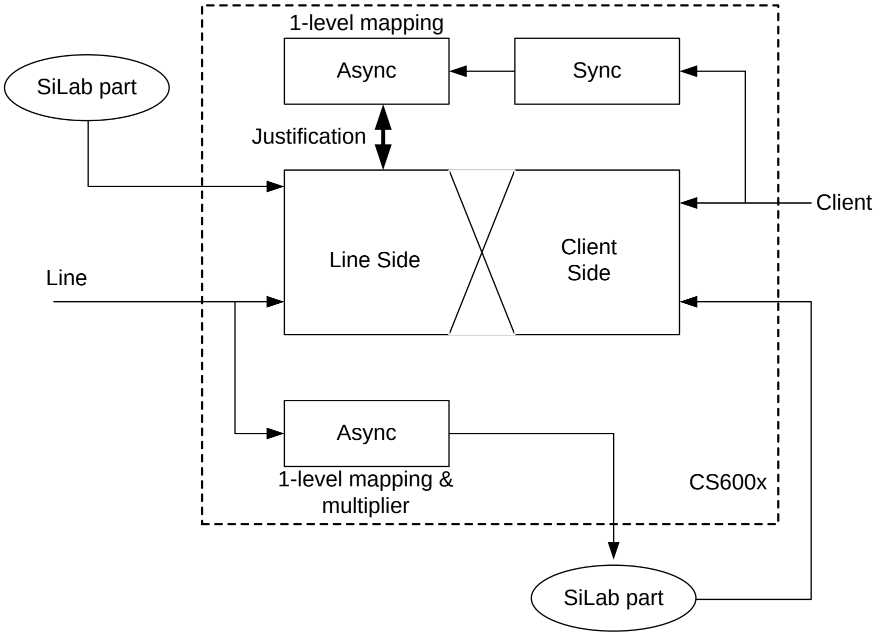


Figure 6 40G Muxponder Sync/Desync configuration 3

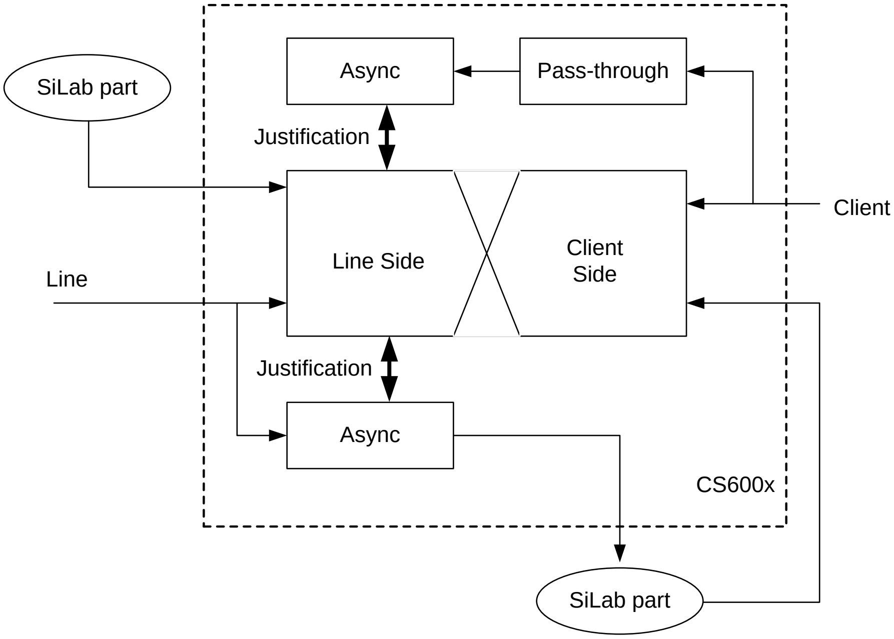
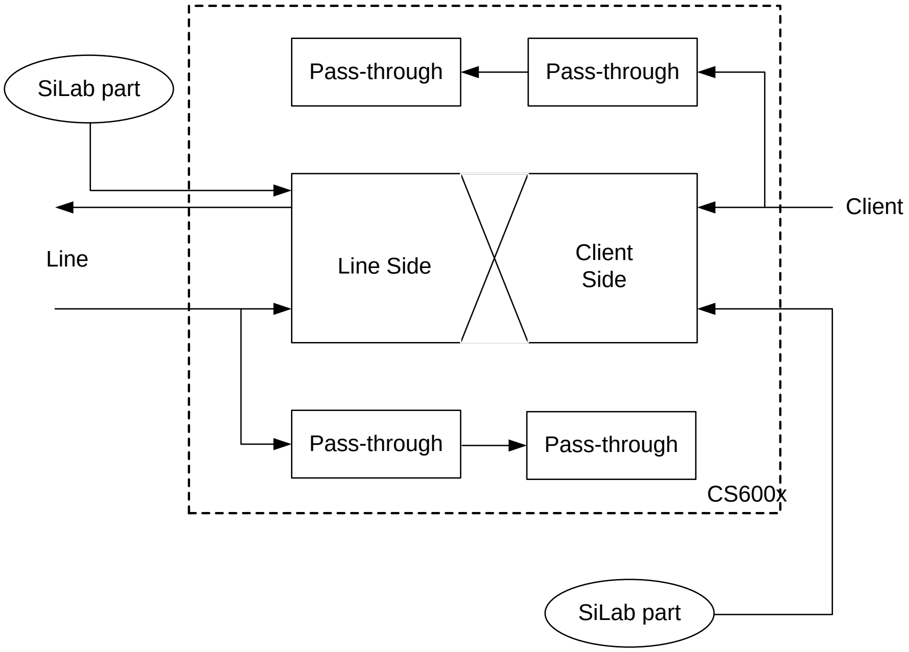


Figure 7 40G Muxponder Sync/Desync configuration 4



5.1.4.1.2 10G Transponder

Table 6 10G Transponder Sync/Desync configurations

Client Mode	Line Mode	10G Transponder Sync/Desync Configuration
OC192	OC192	Pass-through (1:1)
OTU2	OTU2	Synchronous or Asynchronous
10GE6.2	OTU2	(sync/desync not used)
10GE7.1	OTU2e	Synchronous
10GE7.3	OTU2	(sync/desync not used)
OTU2e	OTU2e	Synchronous or Asynchronous
10GFC	OTU2e	Synchronous
10GFC	OTU1f	Synchronous
OTU1f	OTU1f	Synchronous
INFINIBAND	OTU2	Asynchronous

Figure 8 10G Transponder Sync/Desync synchronous configuration

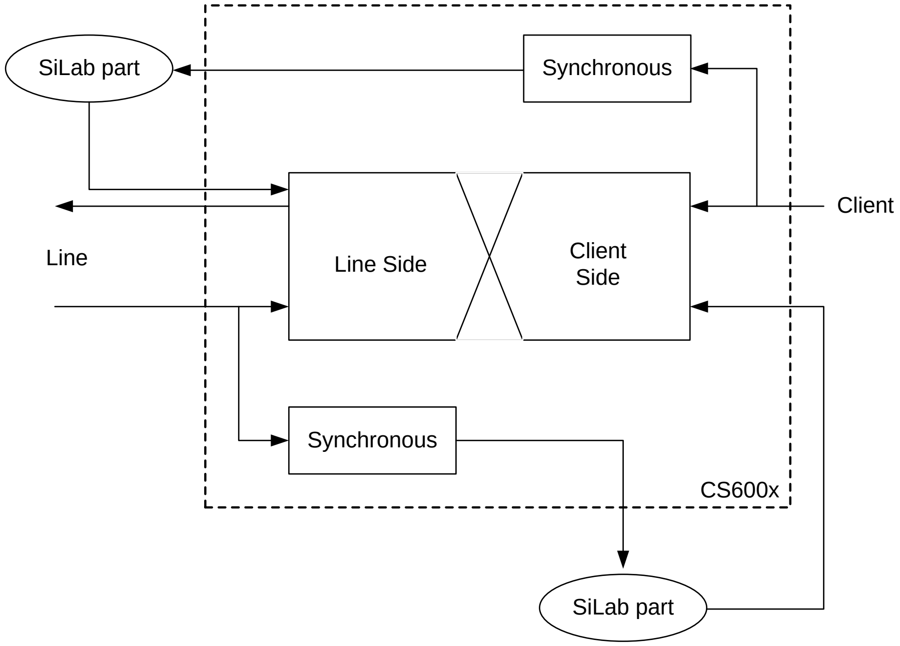


Figure 9 10G Transponder Sync/Desync asynchronous configuration

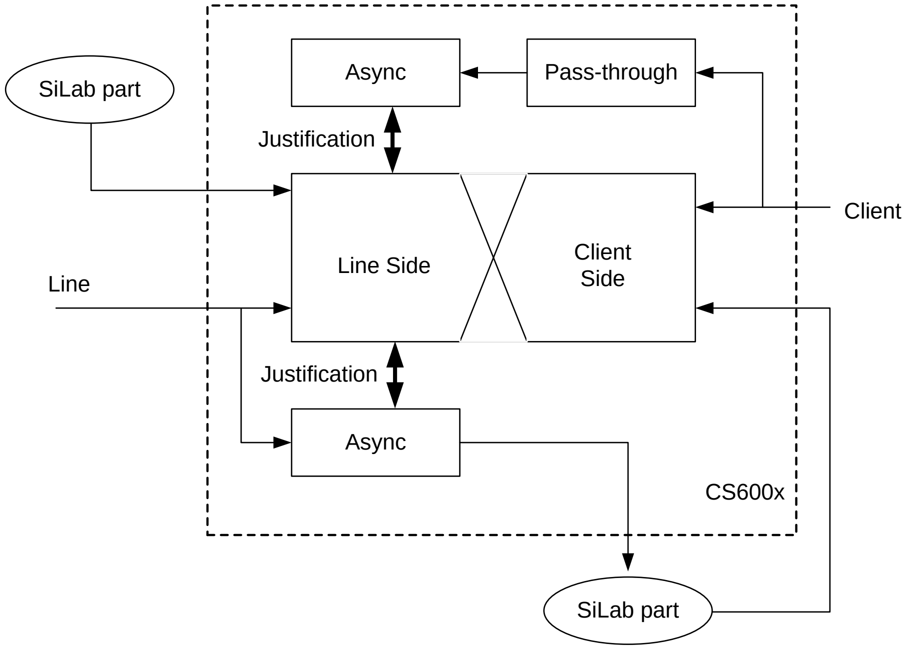
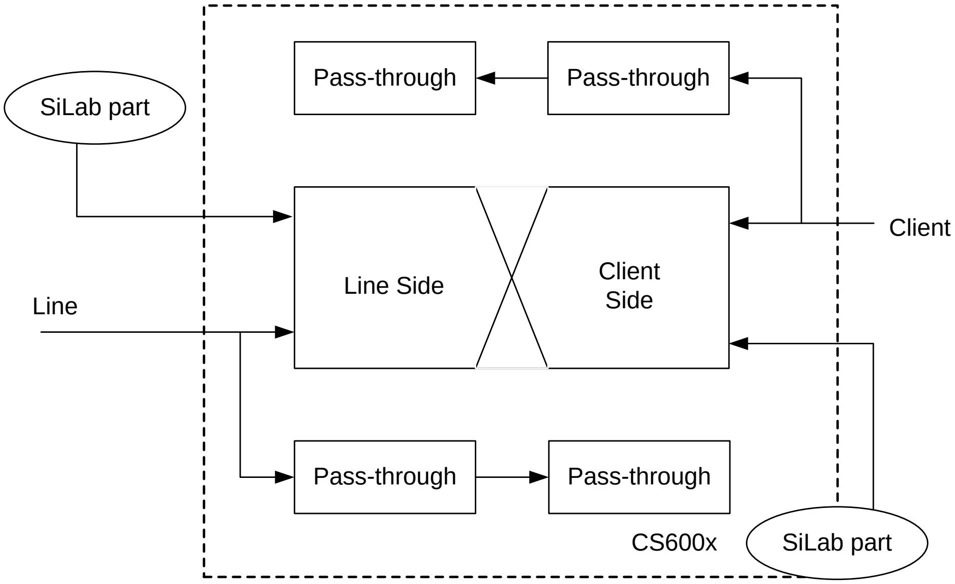


Figure 10 10G Transponder Sync/Desync pass-through (1:1) configuration



5.1.4.2 Datasheet Sections

Section 2.14.2: Synchronizer/Desynchronizer

5.1.4.3 Key APIs

There are a number of sync/desync APIs required to configure the device. There are three main types. The first type generates the required parameters. The second type stores these parameters into specific data structures (called the control block). The third type is the actual configuration routine. The various APIs are summarized below:

Table 7 Sync/desync API Summary

APIs	Purpose of API	Type	Language
get_syncdsync_params get_line_client_params	Get number of pjos and justifications for OxU2 and OxU3 traffic from syncdsync.pm Compute P/Q and line/client infofrom clocks.pm	Parameter generation	perl
ten_syncdsync_set_cb_transcode ten_syncdsync_set_cb_40g_transponder ten_syncdsync_set_cb_10g_transponder ten_syncdsync_set_cb_40g_muxponder	Takes justification values, P/Q values, and line/client info and stores it in data structure. ten_syncdsync_cb_t. Choose appropriate routines for traffic type. (Note: ten_syncdsync_set_cb_transcode must be called for all applications)	Populate control block	C
ten_hl_syncdsync_datapath_config_t41	Configures sync/desync logic in CS604x based on information in data structures and API parameters	Configure sync/desync	C

The configuration is set via the ten_hl_syncdsync_datapath_config () API. This is a complex function which requires a large number of parameters to do its job.

Justification parameters for sync/desync are computed by the get_syncdsync_params() API. The control block will also need OTN frame (P/Q) and line/client information provided by the get_line_client_params() API.

Once the parameters are computed, they must be stored in the sync/desync control block. This minimizes the amount of information that must be explicitly passed to the configuration routine. Only one data-structure API needs to be called, but it must match the type of traffic for the device.

After the parameters and data structures are set, then the configuration routine can be called.

5.1.4.3.1 Parameter Generator APIs

The perl API get_syncdsync_params() computes four parameters: (num_just_oxu2, num_pjo_oxu2, num_just_oxu3, num_pjo_oxu3). These are the OTN justification parameters needed by the sync/desync configuration routine. This API returns a list of these parameters (in that order), which can then be applied to the data-storage APIs.

```
get_syncdsync_params(
      line_type, client_type, mode, sync
)
```

The meanings of the API parameters are as follows:

```
      client_type, line_type
            These can be any one of the available client
            and line types. See Traffic Types for available
            types.

      mode
            One of:
            mux – for muxponder traffic
            trans10 – for 10G transponders
            trans40 – for 40G transponders

      sync
            0 for asynchronous
            1 for synchronous
```

The perl API get_line_client_params() returns eight parameters: (P_Lvalue, QLvalue, PCvalue, QCvalue, line_client_same, line_client_rate_equal, trans_num, trans_den) . As with the previous API, these are returned as a list which can then be applied to the data-storage APIs.

```
get_line_client_params (line_type, client_type, line_fec,
      client_fec, mode, sync)
```

The meanings of the API parameters are as follows:

```
      client_type, line_type
            These can be any one of the available client
            and line types. See Traffic Types for available
            types.

      client_fec, line_fec
            These can be any one of the available client
            and line fec rates. See FEC Rates for available
            types.


      mode
            One of:
            mux – for muxponder traffic
            trans10 – for 10G transponders
            trans40 – for 40G transponders

      sync
            0 for asynchronous
            1 for synchronous
```

5.1.4.3.2 Populate Control Block APIs

As noted above, the user should select one or more of the control-block routine that matches the application.

IMPORTANT: The transcoding routine **must** be called for all applications. It provides a multiplier for the sync/desync computations. If transcoding is not being used, the num and denom parameters must be set to 1.

For all applications; only has unique function for transcoding

This API sets the transcoding multiplier in the control block. If transcoding is not used, this routine must still be called, and num and denom must each be set to 1.

```
ten_syncdsync_set_cb_transcode(dev_id, num, denom)
```

Parameter descriptions:

```
dev_id:
      The device ID

num:
      The Transcode numerator

denom:
      The Transcode denominator
```

Transponder, 40G

This API should be used only for 40G transponder applications

```
ten_syncdsync_set_cb_40g_transponder(
      dev_id, num_just_oxu3, num_pjo_oxu3, P_line, Q_line,
      P_client, Q_client, line_client_same,
      line_client_rate_equal)
```

Parameter descriptions:

```
dev_id:
      The device ID

num_just_oxu3, num_pjo_oxu3:
      Justification and PJO parameters from
      get_syncdsync_params();

      num_just_oxu3 specifies the number of justification
      bytes (column15 of FS)
```

```
          num_pjo_oxu3 specifies the number of positive
          justification bytes (0-6)


    P_line, Q_line:
          Line Numerator (P) and Denominator (Q)

          P_line specifies the number of columns in the OTN
          frame on the line side (239 – 231)

          Q_line specifies the number of columns in the payload
          on the line side (237 – 238)

    P_client, Q_client:
          Client Numerator (P) and Denominator (Q)

          P_client specifies the number of columns in the OTN
          frame on the client side (239 – 231)


          Q_client specifies the number of columns in the
          payload on the client side (237 – 238)

    line_client_same:
          This used for 10G/40G transponder applications to
          indicate line and client types are identical - for
          syncdsync to be programmed 1:1

          0 - line and client type are different
          1 - line and client type are identical

    line_client_rate_equal:
          This is used in conjunction with line_client_same
          parameter to indicate presence of FECs on the client
          side if line and client type are identical

          0 - line and client data rates are different
          1 - line and client data rates are identical
```

Transponder, 10G

```
ten_syncdsync_set_cb_10g_transponder(
      dev_id, num_just_oxu2, num_pjo_oxu2, P_line, Q_line,
      P_client, Q_client, line_client_same,
      line_client_rate_equal)
```

Parameter descriptions:

(See 40G transponder; num_just_oxu2 and num_pjo_oxu2 have the same explanation as num_just_oxu3 and num_pjo_oxu3

Muxponder, 40G

```
ten_syncdsync_set_cb_40g_muxponder(
      dev_id, num_just_oxu2, num_pjo_oxu2, num_just_oxu3,
      num_pjo_oxu3, P_line, Q_line, P_client, Q_client)
```

Parameter descriptions: Most are as above. Exceptions:

```
num_just_oxu2
      Specifies the number of justification bytes
      (C15 of FS) (client -> OxU2)

num_pjo_oxu2
      Specifies the number of positive justification bytes
      (client -> OxU2) (value of 0-6)

num_just_oxu3
      Specifies the number of justification bytes (C15 of
      FS) (OxU2 -> OxU3)

num_pjo_oxu3
      Specifies the number of positive justification bytes
      (OxU2 -> OxU3) (value of 0 – 6)
```

5.1.4.3.3 Syncdsync Configuration API

```
ten_hl_syncdsync_datapath_config_t41(
      module_id_line, slice_line, module_id_client,
      slice_client,  traffic_type_line, fec_line,
      traffic_type_client, fec_client, mode, sync_mode,
      muxponder_method, k_divider,
      map_odtu, map_oxuv)


slice_line, slice_client
      Specified as
            TEN_SLICE0 (0)
            TEN_SLICE1 (1)
            TEN_SLICE2 (2)
            TEN_SLICE3 (3)
            TEN_SLICE_ALL (0xFF)

fec_line parameter defines the line fec type:
fec_client parameter defines the line fec type:
    TEN_FEC_MODE_NOFEC = 15, or
    Any other FEC% as per ten_fec_ovhd_t = 0 -14

mode
      Specifies one of the three modes
            TEN_40G_TRANSPONDER  = 0 (40G to 40G)
            TEN_40G_MUXPONDER    = 1 (10G to 40G)
            TEN_10G_TRANSPONDER  = 2 (10G to 10G)
sync_mode
```

```
            0 for asynchronous
            1 for synchronous

muxponder_method
      Specifies how syncdysnc blocks can be configured for
      40G muxponder applications (asynchronous mapping
      only)
            TEN_MUX_1TO1_ASYNC_2LEVEL = 0
                  Client -  1:1
                  Line   -  Async 2 level mapping
            TEN_MUX_SYNC_ASYNC_1LEVEL = 1
                  Client -  Sync  1 level mapping
                  Line   -  Async 1 level mapping
            TEN_MUX_1TO1_ASYNC_1LEVEL = 2
                  Client -  1:1
                  Line   -  Async 1 level mapping
            TEN_MUX_BOTH_1TO1         = 3
                  Client -  1:1
                  Line   -  1:1


k_divider
      Specifies the value of constant 'k'
            Async : 16 – 64
            Sync  : 16 – 255

map_odtu parameter specifies the mapping method
 map_oxuv parameter specifies the mapping method
  TEN_MAP_BMP,           = 0
  TEN_MAP_AMP,           = 1
  TEN_MAP_AMP_DT,        = 2
  TEN_MAP_AMP_PROP,      = 3
  TEN_MAP_AMP_PROP_DT,   = 4
  TEN_MAP_GMP_LO,        = 5
  TEN_MAP_GMP_HO,        = 6
  TEN_MAP_GMP_HO_DT,     = 7
  TEN_MAP_RATE_ADJUST,   = 8
  TEN_MAP_TRANSCODE,     = 9
```

5.1.5 Fractional Dividers

Fractional dividers are located on the system clock for clock protection purposes. It supports two clock protection switch modes: the client clock protection switch and the receive line clock protection switch.

5.1.5.1 Datasheet Sections

2.14.3: Clock Protection
2.14.4: Top Level Fractional Dividers

5.1.5.2 Applicable APIs

The following API combines the signal type and the FEC type into one parameter, which encodes the corresponding bitrate for use by the autoconfig_fracdiv API.

```
combine_signal_type_and_fec (signal_type, fec)
```

Parameter description:

```
      signal_type
            The traffic type (string). See Traffic Types
            for available types.

      fec
            The FEC type (string). See FEC Rates for
            available types.

      Return value:
            String code corresponding to signal/fec pair
```

Once the corresponding signal type is generated, the fractional divider can be configured with the following API:

```
autoconfig_fracdiv(module_id, slice, direction, mode,
      signal_type, interface_type, k, sysclk_freq)
```

Parameter description:

```
      module
            Module can be 0 (A) or 1 (B)

      slice
            Channel (0..3)

      direction
            An integer noting which fractional dividers
            will be configured:
            0 - RX (pilot)
            1 - TX (protection)
            2 - RX & TX

      mode
            An integer denoting the operation mode of the
            device; one of
                  TEN_GLOBAL_MODE_S_40G - 40G
                  TEN_GLOBAL_MODE_QUAD_10G - 10G

      signal_type
            A string that describes the client traffic.
            This parameter is the output of the
            combine_signal_type_and_fec() API (see above).

      interface_type
```

```
        A string to indicate the HSIF interface; one
        of:
              sfi5_1
              sfi4_2_40g
              sfi4_2_10g
              xfi

  k
        An integer denoting the syncdsync divider value

  sysclk_freq
        The system reference clock frequency in Hz
        (integer)

  Returns:  Nothing
```

5.1.6 Clock Muxes

One of the key configuration requirements for CS604x is to properly set the clock muxes. The correct muxing depends on many factors, including the clock relationships of the line and client, the source clocks, and the interface that is being used.

The ten_hl_mr_config_clockmux (for 10G interfaces) and ten_hl_mr_40g_config_clockmux (for 40G interfaces) APIs set up clock the clock mux to handle multirate traffic, including clock divider muxing.

The XFI interface provides additional clock muxing possibilities. These are handled with additional parameters for the XFI high-level clockmux API. The ten_hl_xfi_config_clockmux API sets up clock the clock mux to handle XFI traffic, including clock divider muxing. It incorporates the functionality of a number of lower-level APIs; refer to the documentation for these functions for more information:

ten_xfi32x1_srx0_rx_clkdiv_ctrl_rdiv_sel
ten_xfi32x1_srx0_rx_clkdiv_ctrl_ctvdiv_sel
ten_xfi32x1_stxp0_tx_clkdiv_ctrl_ctr_div

5.1.6.1.1 Relevant CS600x Datasheet Sections

2.14: Clocking

5.1.6.1.2 Associated APIs

40G Clockmux

```
cs_status ten_hl_mr_40g_config_clockmux(
      cs_uint16,
      module_id)
```

Parameter description:

```
      module_id
            Module can be A (0) or B (1)
```

10G Clockmux

```
cs_status ten_hl_mr_config_clockmux(
      cs_uint16 module_id, cs_uint8 slice)
```

Parameter description:

```
      module_id
            Module can be A (0) or B (1)
      slice
            Integer which indicates the channel (0..3)
```

XFI Clockmux

The XFI version is more complex, since there are more XFI-related clock muxes that can be set:

```
cs_status ten_hl_xfi_config_clockmux(
      cs_uint16 module_id,
      cs_uint8 slice,
      cs_uint16 srx_rdiv_sel
      cs_uint16 srx_ctvdiv_sel,
      cs_uint16 stxp_ctrdiv_sel,
      cs_uint16 aux_clk)
```

Parameter description:

```
module_id
      Side of the CS604x (TEN_MODULE_A, TEN_MODULE_B)

slice
      Specified as
      TEN_SLICE0 (0)
      TEN_SLICE1 (1)
      TEN_SLICE2 (2)
      TEN_SLICE3 (3)
      TEN_SLICE_ALL (0xFF)

srx_rdiv_sel, srx_ctvdiv_sel, stxp_ctrdiv_sel
      See API Guide entries for
      ten_xfi32x1_srx0_rx_clkdiv_ctrl_rdiv_sel
      ten_xfi32x1_srx0_rx_clkdiv_ctrl_ctvdiv_sel
      ten_xfi32x1_stxp0_tx_clkdiv_ctrl_ctr_div

aux_clk
      If 1, use aux_clk as clock source
```

5.1.7 Aux Clock

There are a number of low-level APIs that can utilize the AUX reference clocks (e.g. MRB_TXn_AUXPILOTP/N or XFIn_TX_AUX_LPTIME_CKREF). However, only a few high-level APIs currently support it, and they are all associated with HSIF interfaces.

To apply an AUX clock, see the Interfaces section for details on any high-level API which supports it.

5.1.7.1 Related CS604x Datasheet Sections

Section 2.16.1: Clocking

5.1.8 XFI Clean Up PLL

Use of the Clean Up PLL (CUPLL) in the XFI TX CMU requires the use of only two high-level APIs. ten_hl_cupll_config_t41 provisions the CUPLL specified by the module (a formality for XFI APIs, this will always be TEN_MODULE_B) and slice. The ten_hl_cupll_config_t41 uses data in a control block (pdevcb->cupll_cb) to calculate required values. This control block must be provisioned before calling ten_hl_cupll_config_t41 with the ten_set_cb_cupll_calc_t41 API.

5.1.8.1.1 Relevant CS600x Datasheet Sections

2.16: Clocking and Synchronization

5.1.8.1.2 Associated APIs

```
cs_status ten_set_cb_cupll_calc_t41(cs_uint16 dev_id,
                                    double    rate,
                                    cs_uint8  source,
                                    cs_uint16 k_value,
                                    cs_uint32 freq,
                                    cs_uint32 pdoffset)
```

Parameter descriptions:

```
dev_id
  Device ID

Rate
  Line rate for the line or client in hertz. For
example, for OTU2 7% FEC, 10709225320

Source
  Unused

k_value
  16, 32, or 64

Freq
  Clock rate for the "clean" CUPLL System Clock.
  311040000, 622080000, 400000000, or 200000000

Pdoffset
  Phase Detector Offset. Presently always 0x00000320.
```

```
cs_status ten_hl_cupll_config_t41(cs_uint16 module_id,
                                  cs_uint8  slice)
```

Parameter descriptions:

```
module_id
  Side of the CS604x (TEN_MODULE_A, TEN_MODULE_B)

slice
  Specified as
  TEN_SLICE0 (0)
  TEN_SLICE1 (1)
  TEN_SLICE2 (2)
  TEN_SLICE3 (3)
  TEN_SLICE_ALL (0xFF)
```

5.1.8.1.3 Limitations

1. The CUPLL APIs do not currently work with k_value = 8, 16, or 32.
2. For populating the CUPLL control block, the ten_set_cb_cupll_calc_t41 API must be used for drivers 5.2.1 and later. For drivers previous to 5.2.1 ten_set_cb_cupll_t41 must be used.

5.2 Performance Monitoring

The CS604x device will constantly count various events for performance monitoring, but for some counters a capture operation has to be triggered to be able to read the data out of the counters. Without triggering a capture some counters will always read zero. Latching counters simultaneously at the block level is useful so that related counts (such as Tx vs. Rx packet counts) can be compared.

The capture operation is described in Section 2.16.7.1 the CS604x Data Sheet. A latch operation can be triggered via hardware (using the six on-chip global statistics GSTI timers), via SW (a register bit that is toggled to trigger the capture), or from an external pin. The device’s capture operation can be synchronized with the SW via interrupts (but note that Sighting #3 describes a problem with the GSTI interrupt that requires a SW workaround; see below).

The flow for setting up the counting/triggering and reading the count values is described below.

1. Globally, statistics can be either accumulated in the driver in 64-bit counters so that the statistics reporting functions always report the accumulated values until cleared, or statistics can be reported as they are read from the device (not accumulated). The default is to not accumulate statistics. The following function will select the mode of operation:

```
ten_drvr_ctl_stats_timing_interval
```

2. Program the statistics unit of each block with the source of the trigger. The table below identifies the APIs to be called for each block.

Table 8 Statistics Functions

Block	Dev or Mod	Functions
SYNCDSYNC	Mod	ten_syncdsyncrx_select_global_timing_source ten_syncdsynctx_select_global_timing_source
GFEC_40G	Mod	ten_gfec_40g_set_gi_sel
GFEC_10G	Mod	ten_gfec_10g_set_excess_error_flag
UFEC	Mod	ten_ufec_select_global_timing_source
N40G	Mod	ten_n40g_otnr4x_select_global_timing_source ten_n40g_sdfr_select_global_timing_source ten_n40g_config_kpa ten_n40g_config_kpa_prbs_interval ten_n40g_config_kpa_fixed_interval ten_n40g_oohr_cfg5_bipssel ten_n40g_oohr_cfg5_beissel
N10G	Mod	ten_n10g_otnr_select_global_timing_source ten_n10g_sdfr_select_global_timing_source ten_n10g_config_kpa ten_n10g_config_kpa_prbs_interval ten_n10g_config_kpa_fixed_interval ten_n10g_oohr_cfg5_bipssel ten_n10g_oohr_cfg5_beissel
PP10G	Mod	See note ten_pp10g_pm_control
XCON	Dev	See note 2
HSIF	Mod	See note 2

Note 1: PP10G counters operate in either TICK or IEEE mode. In TICK mode counters are updated every timer tick, and the source of the timing source is programmable like the other blocks. In IEEE mode the counters are cumulative within the device and rollover when the maximum is reached. The function to set the count mode and the source of the timer tick is ten_pp10g_pm_control and the sel_tick1sec parameter is used to define which of the eight trigger sources is used for the timer tick.

Note 2: The XCON and HSIF blocks do not use the same trigger mechanism as the rest of the device. The generic statistics functions in ten_stats_dev and ten_stats_mod do work for the counters in these blocks, however.

3. If GSTI timers are being used then program them with the proper interval using the functions

```
ten_glb_misc_set_gsti_prescale
ten_glb_misc_set_gsti
```

4. Review the sighting described in the next section and decide which of the workarounds will be used for the system. Captures can be triggered manually or to synchronize counters by calling

```
ten_glb_misc_gsti_update
```

5. Whenever the counts are latched (GSTI, SW, or HW originated) call one of the statistics functions to read out the counts. The various statistics functions are in ten_stats_dev.c and ten_stats.mod.c. Also refer to Section 5.0 of this document.

An easy way to check counters during lab testing is to use the SW trigger mechanism. Perform steps 1 and 2 above and set the trigger to “Software controlled timing generator” for all of the blocks whose counters need to be read. Skip step 3. For step 4 call the following function prior to reading counters

```
ten_glb_misc_gsti_update( dev_id, 1, 0 );
```

The counters can then be read with a function call like the following. Refer to Section 5 for additional information on statistics reporting functions.

```
ten_mod_print_stats( module_id );
```

5.2.1 Handling GSTI Interrupts

NOTE: This section applies to the CS600x and does not apply to the CS604x.

From document *CS600x Transport Processor Sightings Report, 400989, Revision 4*: “The MPIF_GS_INTERRUPT will only indicate the expiration of every other global timer interval (GSTI). Each interrupt bit of the register is affected: the eight internal hardware timers (see GLB_MISC_GSTI, GLB_MISC_GSTI0-7), the software timer configuration bit (see GLB_MISC_GSTI.SW) and the external timer via the FSIG pin. The capturing of statistics is not affected; statistics will still be captured each time the selected timer expires.”

All of the workarounds identified in the *Sightings Report* require some amount of SW involvement.

One-half of the timer interrupts will be missed due to the sighting but the statistics capturing will operate as normal. This means that if the processor depends on the hardware interrupt alone then one-half of the statistics data will be missed and not collected. Consider the following scenario where the timer is programmed to expire every T seconds.

Time	Example Statistics Value	Processor View	Calculated Accumulated Statistics Value	Correct Accumulated Statistics Value
0	n/a	n/a	0	0
T	2	Reads 2 from the device	2	2
2T	1	Interrupt is missed so the statistics value of 1 is not accumulated	2	3
3T	3	Reads 3 from the device and accumulates it	5	6
4T	2	Interrupt is missed so the statistics value of 2 is not	5	8

Time	Example Statistics Value	Processor View	Calculated Accumulated Statistics Value	Correct Accumulated Statistics Value
		accumulated		

The calculated accumulated value is incorrect because of the missed interrupts. At interval 4T the processor has only accumulated 5 counts where the actual value should be 8. All of the workarounds identify a mechanism to interrupt the processor at a regular timer interval synchronized with the timer expirations within the device instead of relying solely on the timer interrupt.

5.2.1.1 Workaround #1

This workaround involves using two GSTI timers to capture statistics normally associated with one timer. The first timer (#1) is programmed normally with statistics interval T. Due to the sighting, the interrupt will only be generated every other interval expiration. This workaround will ignore the interrupt since it is inaccurate. The second timer (#2) is used to generate the interrupt that will be used by the processor. It is programmed to have a statistics interval of T/2 but since every-other interrupt is missed then it will actually generate the interrupt at the desired interval of T. The interrupt generated from this second timer is the one that will actually be used to interrupt the processor and collect the statistics data. It is important that the two timers be synchronized, however. This is accomplished by writing a 0→1 transition to the configuration register bits GLB_MISC_GSTI:INIT. For example, if Timer 0 is the first timer (which is being used for statistics capture but not for the interrupt) and Timer 1 is the second timer (which is being used for the interrupt only), then the following would implement the INIT transition.

```
ten_glb_misc_gsti_update( dev_id, 0, 3 );
```

The system SW should ignore Timer 0's interrupt. Timer 1's interrupt should be used instead to kick-off reading the statistics associated with Timer 0.

The advantage of this workaround is that it is relatively simple from a SW perspective and requires no HW changes. The disadvantage is that it consumes two interval timers for each one actually used to latch statistics.

5.2.1.2 Workaround #2

This workaround uses the timer interrupt for the intervals where it correctly appears and a SW timer for the missing timer interrupts. The steps to use this approach are:

1. Initialize the GSTI timer for interval T
2. When the HW interrupt appears:
    a. Read the statistics values
    b. Start a SW timer set to expire in interval T
3. When the SW timer expires
    a. Read the statistics values

The timing interval must be long enough that the SW timer is accurate. If the SW timer is not accurate, by occurring at less than interval T or after interval 2T, then statistics counters will either be accumulated multiple times or missed. It is also important that the SW timer not be started and then free-run, interrupting multiple times, as it will drift compared to the HW interrupt. The SW timer must be re-started at each HW interrupt.

The advantage is that only one GSTI counter is required for each interval. The disadvantage is that a SW timer is required and that it needs to be reasonably accurate for the given timing interval.

5.2.1.3 Workaround #3

This is a SW-only approach where a SW timer is used instead of the timer HW interrupt. The GLB_MISC_GSTI:SW bit will be used to trigger the statistics update. All of the statistics blocks need to be provisioned to use the SW timer option and they will all be captured and read based on the same timer.

The API to execute a SW trigger is:

```
ten_glb_misc_gsti_update( dev_id, 1, 0 );
```

The advantage of this approach is that it is very simple and does not rely on the timer interrupt at all. The disadvantages are that a SW timer is required, the SW timer will likely be asynchronous to SYS_REFCLK and could be less accurate, and all of the statistics have to be triggered and captured at the same time.

5.2.1.4 Workaround #4

This HW-based approach used the FSIG pin to trigger the statistics update based on an external timer that could be synchronous with SYS_REFCLK. All blocks would have to update at the same time. This can be synchronized to the SW by interrupting the processor with the same signal that drives FSIG. If a free interrupt input is not available to the processor, then one of the CS604x GPIO inputs can be tied to FSIG and used to generate the interrupt.

The advantage is that this approach does not require a SW timer and it can be made synchronous with SYS_REFCLK. The disadvantage is that a HW timer and interrupt are required.

5.2.2 Performance Monitoring Debug Tips

Q: My PM Counts are not right. I am not sure if my PM ticks are set up correctly. What should I check?

A: See datasheet (rev 1.0) section 2.14.6.1 “Global Timers for Performance Monitoring (PM Tick(s))”.

Q: I updated my software to follow the driver guide but PM Counts still are not right. What should I check?

A: Here is a summary of items to check regarding global time (external FSIG in this case) and interrupt timing details.

Sequence and timing

```
         ____  _  _  _
FSIG ___/    \__________\__________________________________
INT     |------→Interrupt Set
```

```
READ                 |---- Wait 1msec → Start Reading Stats
```

Checks/Suggestions:

- Does FSIG (or alternate global timer) have a 1 second period?
- Is the pulse >2 SysClk periods wide?
- Is the TICK Interrupt from an FPGA, processor or CS604x? (i.e., PM period done, ready to start reading stats)
- Is the TICK Interrupt triggered by the rising edge of FSIG? Is it triggering on both edges?
    - The internal CS604x interrupt triggers after just a few clocks (very short time). If the interrupt is from the FPGA or processor, a short delay after FSIG rising edge is fine
- If the interrupt is from CS604x, sightings report item #3 will cause issues. See the report for workarounds.
- If the interrupt is from CS604x, but the wrong bit of MPIF_GS_INTERRUPT:GSnIFi is used, that also explains bad counts.
- After the TICK interrupt, is there a 1msec or longer wait period before reading stats?
- For PP10G, add a check for PP10G_PM_INTERRUPT.pm_avI before reading stats
- For debug, poll PP10G_PM_INTERRUPT.pm_avI at ~100msec to see if it triggers a second time before the next TICK
- Each blocks statistics engine has an independent selection of the PM timer TICK, so check each block separately to confirm the PM timer selection is correct.
- Are you reading the current interval after latching with the tick?
    - i.e., are your status reads synchronized after the tick latch?
    - If not you can read some counts from two different timing windows.
    - See Section 2.14.6.1 of revision 1.0 of the CS600x Data Sheet.
- Are you using MPIF_GS_INTERRUPT:GSnIFi (with n=7 <not 9>for the FSIG global timer) to trigger a stats call?
- Also check sightings report item #3, workaround 4.
- Is FSIG sourced by an FPGA? Can your FPGA provide a software interrupt to use for your stats call?

Q: My FC total counts read from CS604x are misaligned with the test set

A: Suggestions:

- Do some simple debug tests such as:
    - Send no frames – check counts

- Send 1 frame at a time – check counts – do it several times
- Send a short burst – check counts – do it several times
- Send a low rate – check counts
- Check 10GE counts. FC and 10GE PM counts use similar circuits.

Q: You mention a 1msec wait before reading status. Why is this needed?

A: The PP10G needs some time after FSIG for latch processing. This is due to a RAM based circuit holding many stat counts. Analysis indicates a delay time of 1usec should work, but many systems can budget 1msec for high margin. The PP10G block uses indirect addressing to RAM based stats registers via this register:

Register: PM: Statistics Access Control Register

Name: PP10G_PM_STATS_ACCESS

There is a long list of statistics registers that are included. Processing that list can take some time, so PP10G provides the “available” interrupt bit for confirmation.

Q: How can I confirm my wait is really long enough?

A: After the delay, you can check this PP10G bit to confirm it is ready for stats access:

Name: PP10G_PM_INTERRUPT

Address(es): 0x19243, 0x19643, 0x19a43, 0x19e43, 0x1a243, 0x1a643, 0x1aa43, 0x1ae43

pm_avI Only applicable when CTRL[mode]=TICK.

When asserted, indicates that the tick switched the banks and new PM counts are available for reading. Alternately you can use this interrupt bit to trigger the call of your stats routine.

Q: Is there an API that we can use to check the PP10G_PM_INTERRUPT. pm_avI bit?

A: There is no API for reading this since it’s an interrupt. You can enable the interrupt and register an interrupt handler for it, then walk the tree periodically, or read the register directly.

Q: Do other blocks besides PP10G have a bit that indicates stat counts are ready to read? Some configurations do not use the PP10G block.

A: The other blocks don’t really need this. A short delay after the FSIG edge and before reading is fine. The worst case delay is for PP10G and ~1msec should suffice as a worst case. That same delay can be applied to all the blocks as a conservative number. Checking the PP10G_PM_INTERRUPT.pm_avI bit is not required normally, but it can confirm the delay is sufficient. Once the process is working; using a worst case delay should be fine.

Q: During my read of stat counters, how can I tell if the later reads are delayed into the next time interval?

A: Try a longer window (e.g. 1sec vs 100msec). Note that the driver reads the counters in the order listed in ten_stats.h ten_pp10g_stats_e.

Q: How can I confirm whether TICK or IEEE mode is active?

A: Confirm the PP10G configuration is right for each slice you are using with the PP10G monitor. Check if PP10G_PM_CTRL is set correctly on A-side and B-side after configuration is complete.

Example: This register looks right for external TICK - WR: 0x1A640 = 0x0701

API call with parameters: ten_pp10g_pm_control

This example sets sel_tick1sec to 7:

```
ten_pp10g_pm_control( 1, 0xFF, 7, 1);
```

In our log this is followed by registers including:

> Name: PP10G_PM_CTRL
>
> Address(es): 0x19240, 0x19640, 0x19a40, 0x19e40, 0x1a240, 0x1a640, 0x1aa40, 0x1ae40

After all configuration steps, read this register and check bit 0 to confirm the mode is TICK versus IEEE. Look for other writes to these same registers later in the script, such as an API call with parameters: ten_pp10g_pm_reset.

Again, check the register writes.

Q: Do I need to configure PM_CTRL every time there is a configuration change to the slice. For example, if we change the port from OTN to FC, do I need to re-configure PM_CTRL if I called the API when port is in OTN mode.

A: Yes. You will reset this and other registers during your reconfig so you will need to reconfigure it again.

5.3 Statistics (Performance Monitoring)

The CS604x Transport Processor supports a wide variety of hardware counters which keep track of various error counts and also serve as performance monitors. The driver has the basic set of APIs which the application SW can use to get, print or clear statistics.

The main characteristics of the statistics support in the driver are:

- The statistics are maintained by the driver at a device as well as a module basis.

- All counters are maintained as 64-bit counters in the driver. The hardware counters come in different sizes (16, 32, 40, or 48 bits) but the driver uses unified 64-bit values for all of them internally in the implementation as well as in the interface to the application SW.
- The driver will internally take care of the special sequences that are required to read or clear the counters that span across multiple registers (i.e. counters that are more than 16-bits in size). Also, the hardware has some counters which are cleared-on-read, while others are cleared by writing 0 to them. The driver will internally do the appropriate processing as needed for the hardware counters.
- The driver will accumulate counters. This is especially needed for those hardware counters which are cleared on read. For the other counters, the driver will accumulate when these counters wrap around.

The statistics are provided on an on-demand basis for the application. When the application SW calls the appropriate API for the statistics, the driver will retrieve the hardware counters, accumulate as needed, and also do the necessary computation for software counters before giving it to the user or printing it. It is expected that the application SW will periodically gather the statistics from the driver.

The basic unit of the driver statistics is made up of two 64-bit value data variables; one for Transmit (TX) counter and one for Receive (RX) counter. It should be noted that some statistics may have only one counter (either TX or RX) and not both.

By default, values are printed in Decimal format. If desired, the user can opt for a hexadecimal format by calling the driver-level API: ten_stats_hex_format_ctl(hex_ctl=> TEN_ENABLE)

5.3.1 Counter Rollover Time

The hardware counters in the ASIC must be retrieved periodically to avoid counters roll over or overflow. The fastest counter roll over time depends on the counter size and the maximum rate of counter increments.

5.3.2 Statistics APIs

The driver provides the user with a collection of device and module APIs which operate on different blocks to get, clear and print statistics.

The available block name for the device is the following:

- TEN_ID_XCON

The available block names for the module are the following:

- TEN_ID_SYNCDSYNC
- TEN_ID_HSIF
- TEN_ID_GFEC_40G
- TEN_ID_GFEC_10G

- TEN_ID_UFEC
- TEN_ID_N40G
- TEN_ID_N10G
- TEN_ID_PP10G
- TEN_ID_PP40G

5.3.2.1 Device API Parameters

Device statistics generally use the following parameters to define the specific counter(s) that need to be read.

dev_id The Device Identifier created when ten_dev_register was called

block_id TEN_ID_<block> that specifies the device block

section_id The slice; refer to Section 2.4

unit_id This is the counter to be captured; it should be an enum from ten_stats.h that is in the enum definition corresponding to the block_id

dir CS_TX, CS_RX, or CS_RX_AND_TX to specify whether transmit, receive, or both sets of counters should be read

5.3.2.2 Module API Parameters

Module statistics generally use the following parameters to define the specific counter(s) that need to be read.

module_id The Module Identifier that was returned by the call to ten_mod_main_init

block_id TEN_ID_<block> that specifies the device block

section_id The slice; refer to Section 2.4

unit_id This is the counter to be captured; it should be an enum from ten_stats.h that is in the enum definition corresponding to the block_id

dir CS_TX, CS_RX, or CS_RX_AND_TX to specify whether transmit, receive, or both sets of counters should be read

5.3.3 Retrieve Statistics

These APIs will retrieve the values from the hardware counters, compute any software counters, accumulate if the counter is clear-on-read, and then finally return a data-structure with the performance counters filled in. The caller of these APIs is expected to provide a pre-allocated buffer (array of cs_uint64s) which the driver will populate. The following APIs are available:

- ten_dev_get_stats
- ten_dev_get_blk_stats

- ten_dev_get_unit_stats
- ten_mod_get_stats
- ten_mod_get_blk_stats
- ten_mod_get_unit_stats

Additionally, the “ten_dev_read_stat” and “ten_mod_read_stat” APIs are available to retrieve an individual counter using the unit id. This is useful for scripting because no C pointers are needed.

5.3.4 Print Statistics

These APIs will print the statistics in a user-friendly format and are provided mainly for debugging purposes. Below is the list of the available APIs that print statistics for both device and module:

- ten_dev_print_stats
- ten_dev_print_blk_stats
- ten_dev_print_unit_stats
- ten_mod_print_stats
- ten_mod_print_blk_stats
- ten_mod_print_unit_stats

When any of the statistics print APIs are called, the driver displays both the TX and RX values for each statistics unit on a single output line. A blank is printed if the corresponding TX or RX counter does not exist (i.e. the unit counter is uni-directional).

Below is an output sample of the ten_mod_print_stats API:

```
--Dev 00 PP10G_MAC------------RX-------------TX-----------
    good_frames       :      100            100
    err_frames        :        5              0
```

Since the set of statistic counters is quite extensive, the user can opt to suppress the printing of those counters which have 0 as value in them, and print only the non-zero value counters. The driver provides an API to enable this option:

```
ten_stats_mask_zero_cntr (mask_zero_cntl => CS_ENABLE)
```

5.3.5 Clear Statistics

These APIs will reset/clear the statistics values in the driver maintained data-structure and also clear the hardware counters that are clearable (i.e. either by doing clear-on-read or write-to-clear). Below is the list of APIs that clear statistics for both device and module objects:

- ten_dev_clear_stats_clear
- ten_dev_clear_blk_stats

- ten_dev_clear_unit_stats
- ten_mod_clear_stats_clear
- ten_mod_clear_blk_stats
- ten_mod_clear_unit_stats

5.3.6 Performance Monitoring – Defects

This section will capture details about defects: which ones are supported; the status registers and API functions to retrieve current status; the interrupt registers and the SW Interrupt Node data structures that facilitate handling interrupts; and finally the counter enums that are associated with some defects. The following format is used.

Status Register: Identifies the CS604x register and bitfield for the defect's status, in the form register.bitfield

Status API: The API function to retrieve the defect's status is shown here. Parameters dev_id, module, and slice are shown as '…'. Parameters specific to the defect are indicated.

Interrupt Registers: Interrupt register.bitfield
Interrupt Enable register.bitfield

Interrupt Node: The ten_irq_node_t structure defined for this interrupt register

Counter: Not all defects have associated counters. If one exists, the block_id and stat enum (from ten_stats.h) are identified, which should be used in statistics calls as described in Section 3.5.2.1.

5.3.6.1 OTN

The OTN defects are identified below.

5.3.6.1.1 OTN LOS

OTN LOS 40G

Status Register:	N40G_GBLR4X_LOSSTAT.LOSF
Status API:	ten_n40g_get_global_los_status(...)
Interrupt Registers:	N40G_GBLR4X_INTR0.LOSFC N40G_GBLR4X_INTR0E.LOSFCE
Interrupt Node:	TEN_IRQ_NODE_N40G_GBLR4X_INTR0
Counter:	None

OTN LOS 10G

Status Register:	N10G_GBLR_GBLRStatus.LOSF
Status API:	ten_n10g_get_global_losf_status(…)
Interrupt Registers:	N10G_GBLR_INTR.LOSFC N10G_GBLR_INTRE.LOSFCE
Interrupt Node:	TEN_IRQ_NODE_N10G_GBLR_INTR
Counter:	None

5.3.6.1.2 OTN LOF

OTN LOF 40G

Status Register:	N40G_OTNR4X_OFSTAT.SLOF
Status API:	ten_n40g_otnr4x_get_otu_framer_status(…)
Interrupt Registers:	N40G_OTNR4X_INTR.ILOF N40G_OTNR4X_INTR.ILOFE
Interrupt Node:	TEN_IRQ_NODE_N40G_OTNR4X_INTR
Counter block_id:	TEN_ID_N40G
Counter stat:	TEN_N40G_OTNR4X_LOFSTAT _STAT

OTN LOF 10G

Status Register:	N10G_OTNR_OFSTAT.SLOF
Status API:	ten_n10g_get_otu_framer_status(…, TEN_NX0G_OTNR_OFSTAT_SLOF)
Interrupt Registers:	N10G_OTNR_INTR.ILOF N10G_OTNR_INTRE.ILOFE
Interrupt Node:	TEN_IRQ_NODE_N10G_OTNR_INTR
Counter:	TEN_N10G_OTNR_LOFSTAT_STAT

5.3.6.1.3 OTN OOF / SEF

There is no hardware SEF indication; use the OOF defect instead.

OTN OOF/SEF 40G

Status Register:	N40G_OTNR4X_OFSTAT.SOOF
Status API:	ten_n40g_otnr4x_get_otu_framer_status(…)
Interrupt Registers:	N40G_OTNR4X_INTR.IOOF N40G_OTNR4X_INTR.IOOFE
Interrupt Node:	TEN_IRQ_NODE_N40G_OTNR4X_INTR
Counter block_id:	TEN_ID_N40G
Counter stat:	TEN_N40G_OTNR4X_OOFSTAT_STAT

OTN OOF/SEF 10G

Status Register:	N10G_OTNR_OFSTAT.SOOF
Status API:	ten_n10g_get_otu_framer_status(…, TEN_NX0G_OTNR_OFSTAT_SOOF)
Interrupt Registers:	N10G_OTNR_INTR.IOOF N10G_OTNR_INTRE.IOOFE
Interrupt Node:	TEN_IRQ_NODE_N10G_OTNR_INTR
Counter:	TEN_N10G_OTNR_OOFSTAT_STAT

5.3.6.1.4 OTN LOM

OTN LOM 40G

Status Register:	N40G_OOHR_SMSTAT.LOM
Status API:	ten_n40g_oohr_get_sm_status(…, TEN_NX0G_OOHR_SM_STATUS_LOM)
Interrupt Registers:	N40G_OOHR_SMINT.LOMI N40G_OOHR_SMINTE.LOMIE
Interrupt Node:	TEN_IRQ_NODE_ N40G_OOHR_SMINT
Counter:	None

OTN LOM 10G

Status Register:	N10G_OOHR_SMSTAT.LOM
Status API:	ten_n10g_oohr_get_sm_status(…, TEN_NX0G_OOHR_SM_STATUS_LOM
Interrupt Registers:	N10G_OOHR_SMINT.LOMI N10G_OOHR_SMINTE.LOMIE
Interrupt Node:	TEN_IRQ_NODE_N10G_OOHR_SMINT
Counter:	None

5.3.6.1.5 OTU BDI

OTU BDI 40G

Status Register:	N40G_OOHR_SMSTAT.DBDI
Status API:	ten_n40g_oohr_get_sm_status(…, TEN_NX0G_OOHR_SM_STATUS_DBDI)
Interrupt Registers:	N40G_OOHR_SMINT.DBDII N40G_OOHR_SMINTE.DBDIIE
Interrupt Node:	TEN_IRQ_NODE_N40G_OOHR_SMINT
Counter:	None

OTU BDI 10G

Status Register:	N10G_OOHR_SMSTAT.DBDI
Status API:	ten_n10g_oohr_get_sm_status(…, TEN_NX0G_OOHR_SM_STATUS_DBDI)
Interrupt Registers:	N10G_OOHR_SMINT.DBDII N10G_OOHR_SMINTE.DBDIIE
Interrupt Node:	TEN_IRQ_NODE_N10G_OOHR_SMINT
Counter:	None

5.3.6.1.6 ODU AIS

ODU AIS 40G

Status Register:	N40G_OOHR_PMSTAT.DAIS

Status API:	ten_n40g_oohr_get_pm_status(..., TEN_NX0G_OOHR_PM_STATUS_DAIS)
Interrupt Registers:	N40G_OOHR_PMINT0.DAISI N40G_OOHR_PMINT0E.DAISIE
Interrupt Node:	TEN_IRQ_NODE_N40G_OOHR_PMINT0
Counter:	None

ODU AIS 10G

Status Register:	N10G_OOHR_PMSTAT.DAIS
Status API:	ten_n10g_oohr_get_pm_status(..., TEN_NX0G_OOHR_PM_STATUS_DAIS)
Interrupt Registers:	N40G_OOHR_PMINT0.DAISI N40G_OOHR_PMINT0E.DAISIE
Interrupt Node:	TEN_IRQ_NODE_N40G_OOHR_PMINT0
Counter:	None

5.3.6.1.7 ODU BDI

ODU BDI 40G

Status Register:	N40G_OOHR_PMSTAT.DBDI
Status API:	ten_n40g_oohr_get_pm_status(..., TEN_NX0G_OOHR_PM_STATUS_DBDI)
Interrupt Registers:	N40G_OOHR_PMINT0.DBDII N40G_OOHR_PMINT0E.DDBIIE
Interrupt Node:	TEN_IRQ_NODE_N40G_OOHR_PMINT0
Counter:	None

ODU BDI 10G

Status Register:	N10G_OOHR_PMSTAT.DBDI
Status API:	ten_n10g_oohr_get_pm_status(..., TEN_NX0G_OOHR_PM_STATUS_DBDI)
Interrupt Registers:	N10G_OOHR_PMINT0.DBDII N10G_OOHR_PMINT0E.DBDIIE
Interrupt Node:	TEN_IRQ_NODE_N40G_OOHR_PMINT0
Counter:	None

5.3.6.1.8 ODU OCI

ODU OCI 40G

Status Register:	N40G_OOHR_PMSTAT.DOCI
Status API:	ten_n40g_oohr_get_pm_status(…, TEN_NX0G_OOHR_PM_STATUS_DOCI)

Interrupt Registers:	N40G_OOHR_PMINT0.DOCII N40G_OOHR_PMINT0E.DOCIIE
Interrupt Node:	TEN_IRQ_NODE_N40G_OOHR_PMINT0
Counter:	None

ODU OCI 10G

Status Register:	N10G_OOHR_PMSTAT.DOCI
Status API:	ten_n10g_oohr_get_pm_status(…, TEN_NX0G_OOHR_PM_STATUS_DOCI)
Interrupt Registers:	N10G_OOHR_PMINT0.DOCII N10G_OOHR_PMINT0E.DOCIIE
Interrupt Node:	TEN_IRQ_NODE_N10G_OOHR_PMINT0
Counter:	None

5.3.6.1.9 ODU LCK

ODU LCK 40G

Status Register:	N40G_OOHR_PMSTAT.DLCK
Status API:	ten_n40g_oohr_get_pm_status(…, TEN_NX0G_OOHR_PM_STATUS_DLCK)
Interrupt Registers:	N40G_OOHR_PMINT0.DLCKI N40G_OOHR_PMINT0E.DLCKIE
Interrupt Node:	TEN_IRQ_NODE_N40G_OOHR_PMINT0
Counter:	None

ODU LCK 10G

Status Register:	N10G_OOHR_PMSTAT.DLCK
Status API:	ten_n10g_oohr_get_pm_status(…, TEN_NX0G_OOHR_PM_STATUS_DLCK)
Interrupt Registers:	N10G_OOHR_PMINT0.DLCKI N10G_OOHR_PMINT0E.DLCKIE
Interrupt Node:	TEN_IRQ_NODE_N10G_OOHR_PMINT0
Counter:	None

5.3.6.2 SONET / SDH

5.3.6.2.1 SONET / SDH LOS

SONET/SDH LOS 40G

Status Register:	N40G_SDFR_SDFSTAT.LOSS
Status API:	ten_n40g_get_sdfr_rx_status(…, TEN_SDFR_SDFSTAT_LOSS)
Interrupt Registers:	N40G_SDFR_SDFIST.LOS N40G_SDFR_SDFISTE.LOSE

Interrupt Node:	TEN_IRQ_NODE_N40G_SDFR_SDFIST
Counter:	None

SONET/SDH LOS 10G

Status Register:	N10G_SDFR_SDFSTAT.LOSS
Status API:	ten_n10g_get_sdfr_rx_status(…, TEN_SDFR_SDFSTAT_LOSS)
Interrupt Registers:	N10G_SDFR_SDFIST.LOS N10G_SDFR_SDFISTE.LOSE
Interrupt Node:	TEN_IRQ_NODE_N10G_SDFR_SDFIST
Counter:	None

5.3.6.2.2 SONET / SDH LOF

SONET/SDH LOF 40G

Status Register:	N40G_SDFR_SDFSTAT.LOFS
Status API:	ten_n40g_get_sdfr_rx_status(…, TEN_SDFR_SDFSTAT_LOFS)
Interrupt Registers:	N40G_SDFR_SDFIST.LOF N40G_SDFR_SDFISTE.LOFE
Interrupt Node:	TEN_IRQ_NODE_N40G_SDFR_SDFIST
Counter block_id:	TEN_ID_N40G
Counter stat:	TEN_N40G_SDFR_LOFEC_STAT

SONET/SDH LOF 10G

Status Register:	N10G_SDFR_SDFSTAT.LOFS
Status API:	ten_n10g_get_sdfr_rx_status(…, TEN_SDFR_SDFSTAT_LOFS)
Interrupt Registers:	N10G_SDFR_SDFIST.LOF N10G_SDFR_SDFISTE.LOFE
Interrupt Node:	TEN_IRQ_NODE_N10G_SDFR_SDFIST
Counter block_id:	TEN_ID_N10G
Counter stat:	TEN_N10G_SDFR_LOFEC_STAT

5.3.6.2.3 SONET / SDH OOF / SEF

There is no hardware SEF indication; use the OOF defect instead.

SONET/SDH OOF/SEF 40G

Status Register:	N40G_SDFR_SDFSTAT.OOFS
Status API:	ten_n40g_get_sdfr_rx_status(…, TEN_SDFR_SDFSTAT_OOFS)

Interrupt Registers:	N40G_SDFR_SDFIST.OOF N40G_SDFR_SDFISTE.OOFE
Interrupt Node:	TEN_IRQ_NODE_N40G_SDFR_SDFIST
Counter block_id:	TEN_ID_N40G
Counter stat:	TEN_N40G_SDFR_OOFEC_STAT

SONET/SDH OOF/SEF 10G

Status Register:	N10G_SDFR_SDFSTAT.OOFS
Status API:	ten_n10g_get_sdfr_rx_status(…, TEN_SDFR_SDFSTAT_OOFS)
Interrupt Registers:	N10G_SDFR_SDFIST.OOF N10G_SDFR_SDFISTE.OOFE
Interrupt Node:	TEN_IRQ_NODE_N10G_SDFR_SDFIST
Counter block_id:	TEN_ID_N10G
Counter stat:	TEN_N10G_SDFR_OOFEC_STAT

5.3.6.2.4 SONET/SDH Section/RST AIS

SONET/SDH Section/RST AIS 40G

Status Register:	N40G_SDFR_SDFSTAT.PN11S
Status API:	ten_n40g_get_sdfr_rx_status(…, TEN_SDFR_SDFSTAT_PN11S)
Interrupt Registers:	N40G_SDFR_SDFIST.IPN11 N40G_SDFR_SDFISTE.IPN11E
Interrupt Node:	TEN_IRQ_NODE_N40G_SDFR_SDFIST
Counter:	None

SONET/SDH Section/RST AIS 10G

Status Register:	N10G_SDFR_SDFSTAT.OOFS
Status API:	ten_n10g_get_sdfr_rx_status(…, TEN_SDFR_SDFSTAT_PN11S)
Interrupt Registers:	N10G_SDFR_SDFIST.IPN11 N10G_SDFR_SDFISTE.IPN11E
Interrupt Node:	TEN_IRQ_NODE_N10G_SDFR_SDFIST
Counter:	None

5.3.6.2.5 SONET/SDH Line/MST AIS

SONET/SDH Line/MST AIS 40G

Use 10G slice 3 for 40G Line Overhead.

Status Register:	N10G_SOHR_OHPSTAT.LAIS (slice 3)

Status API:	ten_n10g_get_mst_status(…, 3, TEN_NX0G_POHR_OHPSTAT_LAIS)
Interrupt Registers:	N10G_SOHR_OHINST0.LAISI (slice 3) N10G_SOHR_OHINST0E.LAISIE (slice 3)
Interrupt Node:	TEN_IRQ_NODE_N10G_SOHR_OHINST0 (slice 3)
Counter:	None

SONET/SDH Line/MST AIS 10G

Status Register:	N10G_SOHR_OHPSTAT.LAIS
Status API:	ten_n10g_get_mst_status(…, TEN_NX0G_POHR_OHPSTAT_LAIS)
Interrupt Registers:	N10G_SOHR_OHINST0.LAISI N10G_SOHR_OHINST0E.LAISIE
Interrupt Node:	TEN_IRQ_NODE_N10G_SOHR_OHINST0
Counter:	None

5.3.6.2.6 SONET/SDH Line/MST RDI

SONET/SDH Line/MST RDI 40G

Use 10G slice 3 for 40G Line Overhead.

Status Register:	N10G_SOHR_OHPSTAT.LRDI (slice 3)
Status API:	ten_n10g_get_mst_status(…, 3, TEN_NX0G_POHR_OHPSTAT_LRDI)
Interrupt Registers:	N10G_SOHR_OHINST0.LRDII (slice 3) N10G_SOHR_OHINST0E.LRDIIE (slice 3)
Interrupt Node:	TEN_IRQ_NODE_N10G_SOHR_OHINST0 (slice 3)
Counter:	None

SONET/SDH Line/MST RDI 10G

Status Register:	N10G_SOHR_OHPSTAT.LRDI
Status API:	ten_n10g_get_mst_status(…, TEN_NX0G_POHR_OHPSTAT_LRDI)
Interrupt Registers:	N10G_SOHR_OHINST0.LRDII N10G_SOHR_OHINST0E.LRDIIE
Interrupt Node:	TEN_IRQ_NODE_N10G_SOHR_OHINST0
Counter:	None

5.3.6.2.7 SONET/SDH Path AIS

SONET/SDH Path AIS 40G

Use 10G slice 3 for 40G Path Overhead.

Status Register:	N10G_POHR_OHPSTAT0.VCAIS (slice 3)

Status API:	ten_n10g_get_poh_status(…, 3, TEN_NX0G_POHR_OHPSTAT0_VCAIS)
Interrupt Registers:	N10G_POHR_OHINST0.VCAISI (slice 3) N10G_POHR_OHINST0E.VCAISIE (slice 3)
Interrupt Node:	TEN_IRQ_NODE_N10G_POHR_OHINST0 (slice 3)
Counter:	None

SONET/SDH Path AIS 10G

Status Register:	N10G_POHR_OHPSTAT0.VCAIS
Status API:	ten_n10g_get_poh_status(…, TEN_NX0G_POHR_OHPSTAT0_VCAIS)
Interrupt Registers:	N10G_POHR_OHINST0.VCAISI N10G_POHR_OHINST0E.VCAISIE
Interrupt Node:	TEN_IRQ_NODE_N10G_POHR_OHINST0
Counter:	None

5.3.6.2.8 SONET/SDH Path RDI

SONET/SDH Path RDI 40G

Use 10G slice 3 for 40G Path Overhead.

Status Register:	N10G_POHR_OHPSTAT0.RDIPS[2:0] (slice 3)
Status API:	ten_n10g_get_poh_status(…, 3, TEN_NX0G_POHR_OHPSTAT0_RDIPS)
Interrupt Registers:	N10G_POHR_OHINST0.RDIPSI (slice 3) N10G_POHR_OHINST0E.RDIPSIE (slice 3)
Interrupt Node:	TEN_IRQ_NODE_N10G_POHR_OHINST0 (slice 3)
Counter:	None

SONET/SDH Path RDI 10G

Status Register:	N10G_POHR_OHPSTAT0.RDIPS[2:0]
Status API:	ten_n10g_get_poh_status(…, TEN_NX0G_POHR_OHPSTAT0_RDIPS)
Interrupt Registers:	N10G_POHR_OHINST0.RDIPSI N10G_POHR_OHINST0E.RDIPSIE
Interrupt Node:	TEN_IRQ_NODE_N10G_POHR_OHINST0
Counter:	None

5.3.6.3 10G Ethernet and Fiber Channel

5.3.6.3.1 10GE LAN

LOS

Loss of Sync

Status Register:	PP10G_PCS_RX_RXSTATUS. syncdetS
Status API:	ten_pp10g_pcs_rx_status(..., TEN_PP10G_PCS_RX_SYNCDETS)
Interrupt Registers:	PP10G_PCS_RX_RXINT.syncdetI PP10G_PCS_RX_RXINTENABLE.syncdetE
Interrupt Node:	TEN_IRQ_NODE_PP10G_PCS_RX_RXINT
Counter:	None

High BER

Status Register:	PP10G_PCS_RX_RXSTATUS.berhighS
Status API:	ten_pp10g_pcs_rx_status(..., TEN_PP10G_PCS_RX_BERHIGHS)
Interrupt Registers:	PP10G_PCS_RX_RXINT.berhighI PP10G_PCS_RX_RXINTENABLE.berhighE
Interrupt Node:	TEN_IRQ_NODE_PP10G_PCS_RX_RXINT
Counter:	None

5.3.6.3.2

PCS 64/66 Errors

Status Register:	**None**
Status API:	**None**
Interrupt Registers:	PP10G_PCS_RX_RXINT.errordecI PP10G_PCS_RX_RXINTENABLE.errordecE
Interrupt Node:	TEN_IRQ_NODE_PP10G_PCS_RX_RXINT
Counter block_id:	TEN_ID_PP10G
Counter stat receive:	TEN_PP10G_PCS49_RX_ERR_FRAMES_STAT

RS Remote Fault

Status Register:	PP10G_RS_RX_RXSTATUS.RFS
Status API:	ten_pp10g_rs_rx_get_status(..., TEN_PP10G_RS_RX_STATUS_RFS)
Interrupt Registers:	PP10G_RS_RX_RXINT.RFI PP10G_RS_RX_RXINTENABLE.RFE
Interrupt Node:	TEN_IRQ_NODE_PP10G_RS_RX_RXINT
Counter:	None

RS Local Fault

Status Register:	PP10G_RS_RX_RXSTATUS.LFS
Status API:	ten_pp10g_rs_rx_get_status(..., TEN_PP10G_RS_RX_STATUS_LFS)
Interrupt Registers:	PP10G_RS_RX_RXINT.LFI PP10G_RS_RX_RXINTENABLE.LFE
Interrupt Node:	TEN_IRQ_NODE_PP10G_RS_RX_RXINT
Counter:	None

XGXS Remote Fault

Status Register:	PP10G_BASEX_XAUI_RX_RX_INTSTAT.RemoteFault
Status API:	ten_pp10g_xaui_rx_get_status(..., TEN_PP10G_XAUI_RX_REMOTEFAULT
Interrupt Registers:	PP10G_BASEX_XAUI_RX_RX_INTERRUPT.RemoteFaultI PP10G_BASEX_XAUI_RX_RX_INTENABLE.RemoteFaulte
Interrupt Node:	TEN_IRQ_NODE_PP10G_BASEX_XAUI_RX_RX_INTERRUPT
Counter:	None

XGXS Local Fault

Status Register:	PP10G_BASEX_XAUI_RX_RX_INTSTAT.LocalFault
Status API:	ten_pp10g_xaui_rx_get_status(..., TEN_PP10G_XAUI_RX_LOCALFAULT
Interrupt Registers:	PP10G_BASEX_XAUI_RX_RX_INTERRUPT.LocalFaultI PP10G_BASEX_XAUI_RX_RX_INTENABLE.LocalFaulte
Interrupt Node:	TEN_IRQ_NODE_PP10G_BASEX_XAUI_RX_RX_INTERRUPT
Counter:	None

Overflow

5.3.6.3.3 10GE WAN

LFD

5.3.6.3.4 Fiber Channel

Loss of Sync / Local Fault

This corresponds to the 'loss_of_sync' signal described in ANSI INCITS 364-2003 section 9.7.2.

Status Register:	PP10G_FC_RX_R_INTSTATUS.prim_lfS
Status API:	ten_pp10g_fc_rx_get_psq_status(..., TEN_PP10G_FC_RX_STATUS_PRIM_LFS)
Interrupt Registers:	PP10G_FC_RX_R_INTERRUPT.prim_lfI PP10G_FC_RX_R_INTENABLE.prim_lfE
Interrupt Node:	TEN_IRQ_NODE_PP10G_FC_RX_R_INTERRUPT
Counter:	None

5.3.6.3.5 GFP

Loss of Frame Delineation (LFD)

Status Register:	PP10G_GFP_RX_INTSTATUS.lfdS
Status API:	ten_pp10g_gfp_rx_get_status(..., TEN_PP10G_GFP_RX_STATUS_LFDS)
Interrupt Registers:	PP10G_GFP_RX_INTERRUPT.lfdI PP10G_GFP_RX_INTENABLE.lfdE
Interrupt Node:	TEN_IRQ_NODE_PP10G_GFP_RX_INTERRUPT
Counter:	None

5.3.7 Performance Monitoring – Counters

5.3.7.1 OTN

5.3.7.1.1 OTN FEC

OTN GFEC Byte Errors 40G

Counter block_id:	TEN_ID_GFEC_40G
Counter stat:	TEN_GFEC40G_FRX40_RBYTER0_STAT

OTN GFEC Byte Errors 10G

Counter block_id:	TEN_ID_GFEC_10G
Counter stat:	TEN_GFEC10G_FRX_RBYTER0_STAT

OTN GFEC Uncorrectable Block Errors 40G

Counter block_id:	TEN_ID_GFEC_40G
Counter stat:	TEN_GFEC40G_FRX40_RERROV0_STAT

OTN GFEC Uncorrectable Block Errors 10G

Counter block_id: TEN_ID_GFEC_10G

Counter stat: TEN_GFEC10G_FRX_RERROV0_STAT

5.3.7.1.2 OTN FEC One-Bit Errors

OTN GFEC One-Bit Errors 40G

Counter block_id: TEN_ID_GFEC_40G

Counter stat: TEN_GFEC40G_FRX40_ROBTER0_STAT

OTN GFEC One-Bit Errors 10G

Counter block_id: TEN_ID_GFEC_10G

Counter stat: TEN_GFEC10G_FRX_ROBTER0_STAT

OTN UFEC One-Bit Errors

This is the total error count.

Counter block_id: TEN_ID_UFEC

Counter stat: TEN_UFEC_URX_TC10_STAT

5.3.7.1.3 OTN FEC Zero-Bit Errors

OTN GFEC Zero-Bit Errors 40G

Counter block_id: TEN_ID_GFEC_40G

Counter stat: TEN_GFEC40G_FRX40_RZBTER0_STAT

OTN GFEC Zero-Bit Errors 10G

Counter block_id: TEN_ID_GFEC_10G

Counter stat: TEN_GFEC10G_FRX_RZBTER0_STAT

OTN UFEC Zero-Bit Errors

This is the total error count.

Counter block_id: TEN_ID_UFEC

Counter stat: TEN_UFEC_URX_TC00_STAT

OTN UFEC Overflow

This is the total error count.

Counter block_id: TEN_ID_UFEC

Counter stat: TEN_UFEC_URX_TEO0_STAT

5.3.7.1.4 OTU BIP8

OTU BIP8 40G

Counter block_id: TEN_ID_N40G

Counter stat: TEN_N40G_OOHR_BIP7ERC1_STAT

OTU BIP8 10G

Counter block_id: TEN_ID_N10G

Counter stat: TEN_N10G_OOHR_BIP7ERC1_STAT

5.3.7.1.5 ODU BIP8

ODU BIP8 40G

Counter block_id: TEN_ID_N40G

Counter stat: TEN_N40G_OOHR_BIP0ERC1_STAT

ODU BIP8 10G

Counter block_id: TEN_ID_N10G

Counter stat: TEN_N10G_OOHR_BIP0ERC1_STAT

5.3.7.1.6 OTU BEI

OTU BEI 40G

Counter block_id: TEN_ID_N40G

Counter stat: TEN_N40G_OOHR_BEI7ERC1_STAT

OTU BEI 10G

Counter block_id: TEN_ID_N10G

Counter stat: TEN_N10G_OOHR_BEI7ERC1_STAT

5.3.7.1.7 ODU BEI

ODU BEI 40G

Counter block_id: TEN_ID_N40G

Counter stat: TEN_N40G_OOHR_BEI0ERC1_STAT

ODU BEI 10G

Counter block_id: TEN_ID_N10G

Counter stat: TEN_N10G_OOHR_BEI0ERC1_STAT

5.3.7.2 SONET / SDH

5.3.7.2.1 SONET / SDH B1 Bit-Level

SONET/SDH B1 Bit-Level 40G

Counter block_id: TEN_ID_N40G

Counter stat: TEN_N40G_SDFR_B1BTER_STAT

SONET/SDH B1 Bit-Level 10G

Counter block_id: TEN_ID_N10G

Counter stat: TEN_N10G_SDFR_B1BTER_STAT

5.3.7.2.2 SONET/SDH B1 Block-Level

SONET/SDH B1 Block-Level 40G

Counter block_id:	TEN_ID_N40G
Counter stat:	TEN_N40G_SDFR_B1BLER_STAT

SONET/SDH B1 Block-Level 10G

Counter block_id:	TEN_ID_N10G
Counter stat:	TEN_N10G_SDFR_B1BLER_STAT

5.3.7.2.3 SONET/SDH B2 Bit-Level

SONET/SDH B2 Bit-Level 40G

Counter block_id:	TEN_ID_N40G
Counter stat:	TEN_N40G_SDFR_B2BTER_STAT

SONET/SDH B2 Bit-Level 10G

Counter block_id:	TEN_ID_N10G
Counter stat:	TEN_N10G_SDFR_B2BTER_STAT

5.3.7.2.4 SONET/SDH B2 Block-Level

SONET/SDH B2 Block-Level 40G

Counter block_id:	TEN_ID_N40G
Counter stat:	TEN_N40G_SDFR_B2BLER_STAT

SONET/SDH B2 Block-Level 10G

Counter block_id:	TEN_ID_N10G
Counter stat:	TEN_N10G_SDFR_B2BLER_STAT

5.3.7.2.5 SONET/SDH REI Bit-Level

Use 10G slice 3.for 40G counter.

Counter block_id:	TEN_ID_N10G
Counter stat:	TEN_N10G_POHR_REIPBTER_STAT

5.3.7.2.6 SONET/SDH REI Block-Level

Use 10G slice 3.for 40G counter.

Counter block_id:	TEN_ID_N10G
Counter stat:	TEN_N10G_POHR_REIPBLER_STAT

5.3.7.3 10G Ethernet, Fiber Channel, and GFP

5.3.7.3.1 10GE LAN

These are some of the more important 10GE LAN statistics counters. Refer to ten_pp10g_stats_e in ten_stats.h for a complete list of the available counters.

Octets

Counter block_id:	TEN_ID_PP10G
Counter stat receive:	TEN_PP10G_MAC_RX_OCTETS_STAT
Counter stat transmit:	TEN_PP10G_MAC_TX_OCTETS_STAT

Good (Valid) Octets

This counter implements 802.3 30.3.1.1.14 aOctetsReceivedOK which does not include the MAC header and CRC. To include the MAC header and CRC use TEN_PP10G_MAC_RX_OCTETS_STAT (for good and bad packets) or use the equation:

MAC_RX_GOOD_OCTETS + MAC_RX_GOOD_FRAMES*(18+(VLANtag*4)) for only ‘good’ frames counting.

This equation will be accurate only if the VLANtag (=0/1/2/3/4) is known and fixed on the link.

Counter block_id:	TEN_ID_PP10G
Counter stat receive:	TEN_PP10G_MAC_RX_GOOD_OCTETS_STAT
Counter stat transmit:	TEN_PP10G_MAC_TX_GOOD_OCTETS_STAT

Bad (Invalid) Octets

Bad octets are not reported by the CS604x devices. Customer SW will need to subtract Good Octets from Octets to get Bad Octets.

Frames

Counter block_id:	TEN_ID_PP10G
Counter stat receive:	TEN_PP10G_MAC_RX_FRAMES_STAT
Counter stat transmit:	TEN_PP10G_MAC_TX_FRAMES_STAT

Valid Frames

Counter block_id:	TEN_ID_PP10G
Counter stat receive:	TEN_PP10G_MAC_RX_GOOD_FRAMES_STAT
Counter stat transmit:	TEN_PP10G_MAC_TX_GOOD_FRAMES_STAT

Errored Frames

Counter block_id:	TEN_ID_PP10G

Counter stat receive: TEN_PP10G_MAC_RX_ERR_FRAMES_STAT

Counter stat transmit: TEN_PP10G_MAC_TX_ERR_FRAMES_STAT

Jabbers

Counter block_id: TEN_ID_PP10G

Counter stat receive: TEN_PP10G_MAC_RX_JABBER_FRAMES_STAT

Fragments

Use the Runt Frame statistics.

Pause Frames

Counter block_id: TEN_ID_PP10G

Counter stat receive: TEN_PP10G_MAC_RX_PAUSE_FRAMES_STAT

Counter stat transmit: TEN_PP10G_MAC_TX_PAUSE_FRAMES_STAT

The CS604x device does not differentiate between *Pause Frames Zero* and *Pause Frames Non-Zero*.

Control Frames

Oversize Frames

Use the Giant or Jabber statistics.

Undersize Frames

Counter block_id: TEN_ID_PP10G

Counter stat receive: TEN_PP10G_MAC_RX_UNDERSIZE_FRAMES_STAT

Discarded Frames

Use the GCADJ_DISCARDED statistics.

Symbol Errors

For a PCS 64b/66b client use PCS49_RX_ERR_DEC.

For a XAUI client use XAUI_RX_ICC_LNx.

For a PCS 8b/10b client use EGPCS_RX_TC_ERR.

5.3.7.3.2 GFP

Good Frames

Counter block_id: TEN_ID_PP10G

Counter stat receive: TEN_PP10G_GFP_RX_GOOD_FRAMES_STAT

Payload Header Multiple Bit Error or Uncorrected Single Bit Error (n_FDis_tHEC)

Counter block_id:	TEN_ID_PP10G
Counter stat receive:	TEN_PP10G_GFP_RX_PH_ERR_FRAMES_STAT

Payload Header EXI not equal to "0000" (n_FDis_eHEC_EXI)

Counter block_id:	TEN_ID_PP10G
Counter stat receive:	TEN_PP10G_GFP_RX_EXM_FRAMES_STAT

Core Header Errored Frames (cHEC)

Counter block_id:	TEN_ID_PP10G
Counter stat receive:	TEN_PP10G_GFP_RX_CH_ERR_FRAMES_STAT

Payload FCS Error (n_FCSErr)

Counter block_id:	TEN_ID_PP10G
Counter stat receive:	TEN_PP10G_GFP_RX_FCS_ERR_FRAMES_STAT

Discarded Frames

Counter block_id:	TEN_ID_PP10G
Counter stat receive:	TEN_PP10G_GFP_RX_DISCARD_FRAMES_STAT

5.4 Dynamic Reprovisioning

5.4.1 Overview

The device is programmed for a certain mixture of traffic initially. (Note: There must be at least one channel of valid traffic.) Dynamic reprovisioning is the changing of a port from one type of traffic to another without affecting the traffic being carried on other ports. In general, the CS604x supports dynamic reprovisioning of 10G slices, with certain limitations.

This release allows the user to perform dynamic reprovisioning on either a line channel or a line/client pair.

For example, in an aggregation configuration, the 40G port can't be changed from OTU3 to OTU3e without impacting all of the traffic. Dynamic reprovisioning will impact traffic being carried on the port that is being reprovisioned. Reprovisioning the 40G port will impact all four client ports in an aggregation configuration, or the client 40G port in a transponder configuration.

In either a muxponder or transponder (10G → 10G or 40G → 40G) configuration the line side is the port that is intended to have a minimal traffic hit when the client-side is reprovisioned.

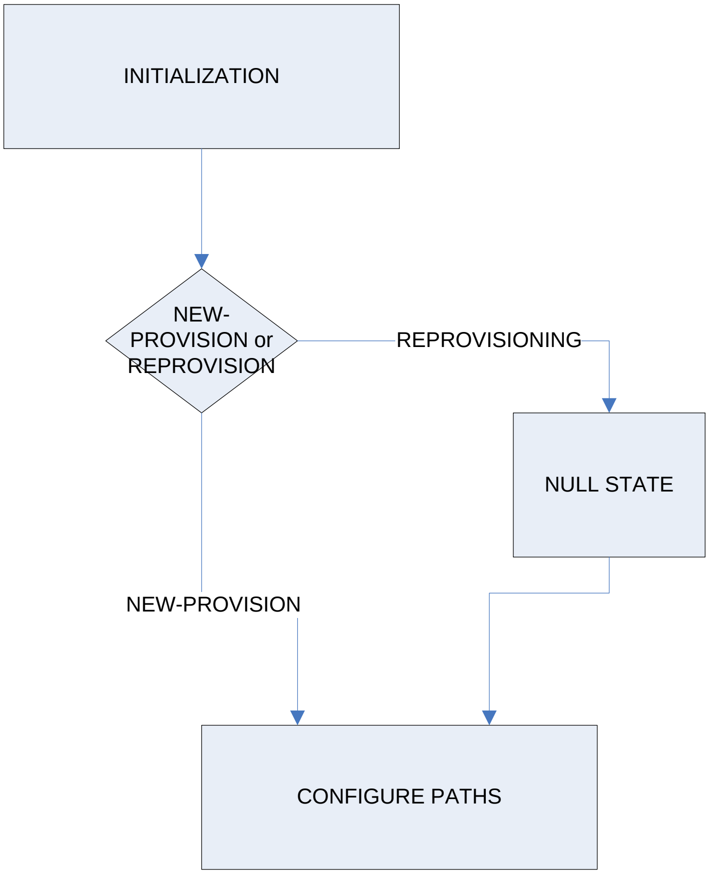
5.4.2 Procedure

In the discussion below, 'n' is the original configuration for a port and 'm' is the desired reprovisioned configuration.

Ports must be reprovisioned in a two-step process. The first step is to change the port from ‘n’ to an unconfigured state. (If the port has not been configured yet, then it is not necessary to do this.) The second step is to change the port from the unconfigured state to ‘m’.

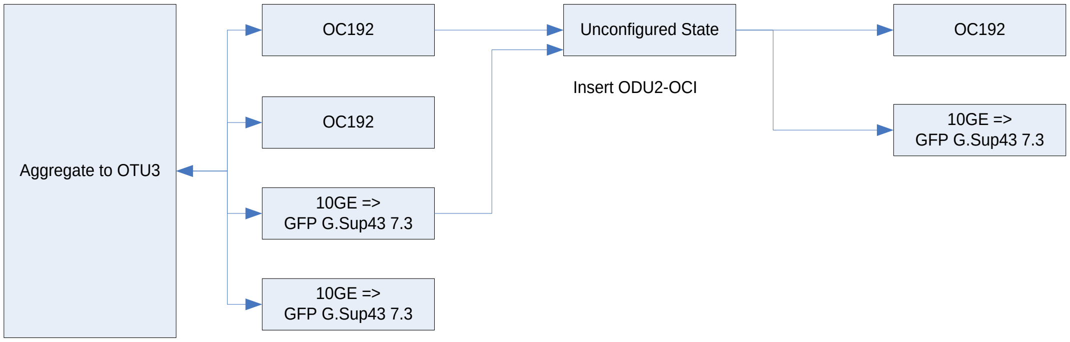
The “unconfigured state” is like the power-on-reset state for a 10G channel. Adding the unconfigured state to the process minimizes the total number of transitions that need to be tested and supported. Let’s say there are ‘x’ client mappings supported by the system. Without the intermediate unconfigured state, there would be x*(x-1) different configuration changes that would need to be tested. If ‘x’ is 10 then this is 90 tests. With the intermediate unconfigured state there are 2x configuration changes. If ‘x’ is 10 then this is only 20 tests.

Figure 11 Dynamic Reconfiguration



For example, the steps required to reprovision an aggregation configuration are:

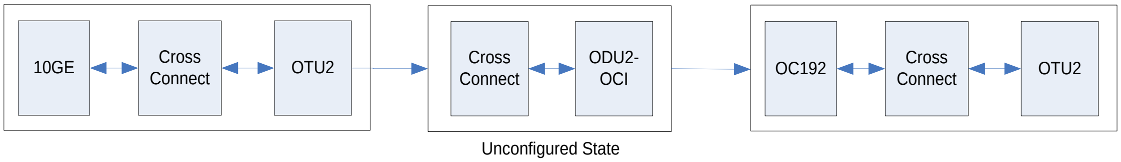
Figure 12 Muxponder Dynamic Reconfiguration



- Set N40G to indicate a 10G channel is disabled
  - Insert ODU2-OCI on the line side of the 10G channel being disabled
- Block the 10G Transmitter from sending a bad signal to the client
- Reset the 10G channel to the “unconfigured state”
  - (optional) Set hard reset on the client side of the 10G channel being disabled.
  - (optional) Disable clock on the client side of the 10G channel.
  - Set soft reset on the elastic store FIFOs on the both line side and client side of the 10G channel.
- Configure the 10G channel to the new state
  - Reconfigure HSI on the client side of the 10G channel.
  - Remove soft reset on the elastic store FIFOs on the both line side and client side of the 10G channel.
  - Remove hard reset the client side of the 10G channel.
  - Enable clock on the client side of the 10G channel.
  - Reconfigure 10G framer and packet processor on both the line side and client side of the 10G channel.
  - Remove ODU2-OCI on the line side of the 10G channel.

The steps required to reprovision a 10G transponder application are:

Figure 13 Transponder Dynamic Reconfiguration



- Set N10G to indicate a 10G channel is disabled
  - Insert ODU2-OCI on the line side of the 10G channel to indicate a 10G channel is disabled
- Block the 10G Transmitter from sending a bad signal to the client
- Reset the 10G channel to the “unconfigured state”
  - (optional) Set hard reset on the client side of the 10G channel being disabled.
  - (optional) Disable clock on the client side of the 10G channel.
  - Set soft reset on the elastic store FIFOs on the both line side and client side of the 10G channel.
- Configure the 10G channel to the new state
  - Reconfigure HSI on the client side of the 10G channel.
  - Remove soft reset on the elastic store FIFOs on the both line side and client side of the 10G channel.
  - Remove hard reset the client side of the 10G channel.
  - Enable clock on the client side of the 10G channel.
  - Reconfigure 10G framer and packet processor on both the line side and client side of the 10G channel.
  - Remove ODU2-OCI on the line side of the 10G channel.

5.4.3 Key APIs

ten_hl_n10g_reconfig_rx_init

ten_hl_n10g_reconfig_tx_init

5.5 Registering Callbacks

There are a number of areas where the Device Driver can call user functions. In order to limit changes to the Device Driver code by users, the driver instead supports a callback registration methodology. This section will describe the registered callbacks available.

5.5.1 Register Writes

Should the user code need to monitor register writes, for support of redundant processors for example, a callback function can be registered that will be called whenever a CS604x register is written to. The functions that support this are:

```
// This registers a callback function
ten_register_reg_write( ten_register_reg_write_t
reg_write_cb );
// This deregisters a callback function
ten_unregister_reg_write( void );
```

If a register write occurs, the callback function will be called with the following parameters:

```
volatile cs_uint16* address
         cs_uint16  value
```

5.5.2 Register Reads

Should the user code need to monitor register reads, a callback function can be registered that will be called whenever a CS604x register is read from. The functions that support this are:

```
// This registers a callback function
ten_register_swerr( ten_register_swerr_t swerr_cb )
// This deregisters a callback function
ten_unregister_reg_read( void )
```

If a register read occurs, the callback function will be called with the following parameters:

```
volatile cs_uint16* address
         cs_uint16  value
```

5.5.3 Software Errors

The function ten_error_handler is called when there is an error detected within the Device Driver, for example when a parameter is invalid. CS_PRINT is called by this function to print useful debugging information. Some users may want to perform additional logging outside of the CS_PRINT function, however. A callback may be registered that gets called by ten_error_handler.

```
// This registers a callback function
ten_register_swerr( ten_register_swerr_t swerr_cb )
// This deregisters a callback function
ten_unregister_swerr( void )
```

If a software error occurs, the callback function will be called with the following parameters:

```
cs_uint16 id
cs_uint16 errorNumber
cs_char8* errorString_p
```

5.5.4 SFU Shadow RAM Updates

For redundancy it may be necessary to track changes to the SFU Shadow Ram. When function ten_ohpp_copy_shadow_sfc_ram is called, the registered callback function will also be called so that the user can track changes to the Shadow RAM.

```
// This registers a callback function
ten_register_shadow_ram( ten_register_shadow_ram_t
reg_shadow_ram_cb )
// This deregisters a callback function
ten_unregister_shadow_ram( void )
```

If a software error occurs, the callback function will be called with the following parameters:

```
cs_uint16  module_id
cs_uint16  bank
cs_uint16 *shadowRam_p
cs_uint16  shadowRamSize
```

5.6 Diagnostics and Test Functions

5.6.1 Loopbacks

There are a number of loopbacks in CS604x which can be controlled via software.

It is also possible to use the cross-connect (XCON) block to implement a loopback. However, doing so is usually complicated, because it user must pay careful attention to the timing modes (e.g. synchronous or asynchronous) and rates on both sides of the XCON. It is much easier to apply the loopbacks described below.

5.6.1.1 Interface (HSIF) Loopbacks

5.6.1.1.1 Relevant CS604x Datasheet Sections

Figures 161 and 162

Section 2.4.15: Loopbacks and Test Modes of Operation

Section 4.1: Loopbacks

5.6.1.1.2 High-level APIs

There are two high-level APIs that can be used to control any of the HSIF loopbacks:

ten_hl_hsif_control_loopback (module_id, slice, loopback, ctl)

ten_hl_hsif_control_loopback_with_user_settings(module_id, slice, loopback, ctl, *settings)

Parameter description:

```
The [slice] parameter is specified as:
      0x00 = TEN_SLICE0
      0x01 = TEN_SLICE1
      0x02 = TEN_SLICE2
      0x03 = TEN_SLICE3
      0xFF = TEN_SLICE_ALL (ALL 10G SLICES OR MR40G)

[loopback] parameter is specified as:
      TEN_HSIF_LOOPBACK_NONE = 0
      TEN_HSIF_LOOPBACK_FACILITY = 1 (MR40G)
      TEN_HSIF_LOOPBACK_FACILITY_EXT = 2
      TEN_HSIF_LOOPBACK_FACILITY_MR10G = 3 (MR10G)
      TEN_HSIF_LOOPBACK_FAR_END = 4
      TEN_HSIF_LOOPBACK_MR_TX2RX = 5
      TEN_HSIF_LOOPBACK_MR_RX2TX = 6
      TEN_HSIF_LOOPBACK_XFI_TX2RX = 7
      TEN_HSIF_LOOPBACK_XFI_RX2TX = 8
      TEN_HSIF_LOOPBACK_MR_SERIAL = 9
      TEN_HSIF_LOOPBACK_FACILITY_XFI = 10
```

(RX2TX means that the input source for the TX datapath is RX loopback data.)

```
[ctl] Controls loopback. Each loopback keeps the last state
written to it(enabled or disabled)
      0 = CS_DISABLE
      1 = CS_ENABLE

[*settings] is a pointer to buffer space that holds values
for a ten_hsif_loopback_settings_t struct.
```

To set a loopback, execute the high-level API with the loopback field set to the value for the desired loopback type and the ctl field set to 1. To clear a loopback, do the same but with the ctl parameter set to 0.

The high-level loopbacks operate on a per-slice basis; all lanes of that slice will be placed in loopback.

User-settings API vs. non-user-settings API

These APIs are both able to apply any of the interface loopbacks. But there is a key difference:

- The "_with_user_settings" API also allows the user to supply a pointer to buffer space that holds settings needed to go out of loopback.
- The non-"with_user_settings" version saves the state of the device in a driver data structure.

If there is a CPU fault and a redundant CPU needs to take over while in loopback then the saved state information will not be available, and the removal of loopback will not cause traffic to be properly restored.

The solution to this is to move the data structure out of the driver via the "with_user_settings" functions. Here, the System would be responsible for replicating the data structure on the redundant CPU following the loopback function call

5.6.1.1.3 HSIF Parallel Loopback APIs

There are also HSIF APIs which act on individual HSIF slices or lanes. There is a separate API for each loopback type. The high-level APIs can set any loopback that the specific HSIF APIs can, and it is generally advisable to use the high-level APIs where possible.

The examples below show the complexity of using the HSIF APIs for loopbacks, and should encourage the user to use the high-level APIs.

However, in certain cases the user may need to use an HSIF loopback. For instance, if you want to loop back ONE lane using the multi-rate serial loopback, you cannot use the high-level version (which operates on a per-slice basis).

To implement the HSIF parallel loopbacks, use one of the following:

```
ten_hsif_facility_loopback (module_id, instance, ctl)
ten_hsif_far_end_loopback (module_id, instance, ctl)

The module_id indicates Module A or B.
The instance is the channel number (0..3).
The ctl field is either 0 (disable) or 1 (enable).
```

Once an RX-TX HSIF loopback has been applied, you should reinitialize the RX-TX elastic stores using:

ten_hsif_rx_tx_elastic_store_control (module_id, instance, act)

```
The act parameter specifies the reset option and is one of
the following:
      0 = CS_RESET_DEASSERT
      1 = CS_RESET_ASSERT
      2 = CS_RESET_TOGGLE
      (The CS_RESET_TOGGLE option will assert reset, hold
      the reset for a while and then de-assert the reset
      bit(s).)
```

You can also resync the TX and RX elastic stores individually,

To resync the TX elastic stores, use the following function (one API per slice)

```
ten_hsif_slc{n}_mr10x4_sds_common_txelst0_resync(module_id,
resync)
```

The RX elastic stores accessed either per slice or lane, so the user needs to make 4 or 18 API calls to resync the RX elastic stores using the HSIF APIs.

```
ten_hsif_slc{n}_mr10x4_sds_common_rxelst0_resync(
module_id, instance, resync)

(Note: {n} is one of 0..3)
```

Parameter description:

```
Module_id denotes module A or B
```

```
Instance is the lane number, which is one of
      0..3 or TEN_INSTANCE_ALL (0xFF) for slices 0 and 2
      0..4 or TEN_INSTANCE_ALL (0xFF) for slices 1 and 3
resync parameter is specified as
      0 (no resync)
      1 (resync)
```

(Note: The resync bit must be cleared to 0 to resume dataflow.)

5.6.1.1.4 Multirate Serial Loopbacks using HSIF APIs

To implement the serial loopbacks, you can use the following APIs. It requires a large number of API calls to implement the serial loopback using the HSIF APIs.

TX: On the TX side the serial loopback has one enable bit per slice. There are 4 slices per module.

```
ten_hsif_slc{n}_mr10x4_sds_common_stxp0_tx_config_lpbken
(module_id, stxp_lpbken)

(Note: {n} is the slice number, from 0..3)
```

Parameter description:

```
Module_id denotes module A or B
srx_lpbken
      0 = loopback disabled
      1 = loopback enabled
```

This API configures enables the TX to send traffic over the loopback. There is one TX bit per slice to set. You must set the TX to transmit to make the loopback work.

RX: For the RX side the loopbacks must be enabled per instance (lane). There are 18 lanes per module.

```
ten_hsif_slc{n}_mr10x4_sds_common_srx0_srx_lpbken
(module_id, instance, srx_lpbken)

(Note: {n} is the slice number, from 0..3)
```

Parameter description:

```
Module_id denotes module A or B
Instance is the lane number, which is one of
      0..3 or TEN_INSTANCE_ALL (0xFF) for slices 0 and 2
      0..4 or TEN_INSTANCE_ALL (0xFF) for slices 0 and 2
srx_lpbken
      0 = loopback disabled
      1 = loopback enabled
```

Once you have enabled the loopback, you must then set the TX data source to actually have the TX path use the loopback data:

```
ten_hsif_slc{n}_mr10x4_sds_common_tx_config (module_id,
instance, data_source)
```

Module_id and instance are as above. The data_source parameter values are:

```
data_source
      0 = DIG_TX_DIN (regular data input)
      1 = PRBS (from HSIF PRBS generator, which must be
      configured separately)
      2 = RX Loopback Data
```

5.6.1.1.5 XFI Serial Loopbacks Using XFI APIs

The XFI interfaces have a serial loopback capability similar to the multi-rate interface. It is configured in a similar manner. However, since the XFI has no lanes, everything is done on a per-slice basis. The applicable APIs are:

```
ten_xfi32x1_stxp0_tx_config(module_id, slice,
stxp_lptime_sel, stxp_pilot_sel, stxp_fcen, stxp_buswidth,
stxp_lpbken)
```

which enables the drivers for TX-RX serial loopback

```
ten_xfi32x1_rx0_config (module_id, slice, data_source)
```

which chooses the RX data source (loopback or other traffic sources), and

```
ten_xfi32x1_tx0_config (module_id, slice, data_source)
```

which selects which data is used by the transmit (datapath, PRBS, RX loopback data).

5.6.1.2 PP10G Loopbacks

There are two loopbacks that are available in the PP10G (the packet processor).

Applying the PP10G loopbacks requires that the PP10G path be configured and in use. The PP10G loopbacks cannot be used for non-PP10G traffic.

A transmit to receive PCS loopback implementing the loopbacks in CS604x Datasheet Sections 2.8.2 and 2.12.7 can be enabled using the PCS loopback APIs (see below).

For debugging purposes, a transmit to receive loopback (see Datasheet Figure 395) can be enabled using the debugging APIs indicated below.

Once a PCS loopback is enabled, the loopback elastic store should be resynchronized.

5.6.1.2.1 Relevant CS604x Datasheet Sections

Section 4.1: Loopbacks
Section 2.8.2: Receive 64b/66b PCS

Section 2.8.9: Receive XGMAC
Section 2.12.3: Transmit XGMAC
Section 2.12.7: Transmit 64b/66b PCS
Figure 395
Figure 396

5.6.1.2.2 Relevant APIs

PCS loopback:

```
ten_pp10g_pcs_rx_loopback(module_id, slice,
rx_estore_resync, rx_loopback)
ten_pp10g_pcs_tx_loopback (module_id, slice,
tx_estore_resync, tx_loopback, tx_loopback_data_en)
```

Debugging (tx-to-rx) loopback:

```
ten_pp10g_mac_ctl_loopback (module_id, slice, dir, enbl)
ten_pp10g_tx2rx_ctrl (module_id, slice, tx2rx_bp,
tx2rx_clken, tx2rx_en)
```

5.6.2 Cross-Connect Loopbacks

It is possible to implement loopbacks in the CS604x using the cross-connect (XCON) block. However, part of the cross-connect function is to transfer data between different timing domains, and it is not always easy to configure the correct timing relationships when reflecting data back to the source.

There are several high-level APIs available to allow for loopbacks via the cross-connect. The user must choose one that matches the data rate (10G, 40G) and traffic type (OTN, OC192). No other traffic types are supported at present.

5.6.2.1 10G Cross Connect Loopbacks

5.6.2.1.1 OC192 Cross Connect Loopbacks

The following API configures a 10G (OC192) cross connect loopback:

```
cs_status ten_hl_config_oc192_xcon_loopback(
      cs_uint16 module_id,
      cs_uint8  slice,
      cs_uint16 traffic_type,
      cs_uint16 dyn_repro,
      cs_uint16 term_sonet)
```

Parameter description:

```
module_id specifies module ID
      0xXX00 = Module A
      0xXX01 = Module B

slice specifies slice:
      0x00 = TEN_SLICE0
      0x01 = TEN_SLICE1
      0x02 = TEN_SLICE2
      0x03 = TEN_SLICE3

dyn_repro specifies the type of dynamic reprovisioning:
      0x00 = TEN_INITIAL_CONFIG
      0x01 = TEN_REPRO_CLIENT
      0x02 = TEN_REPRO_LINE_AND_CLIENT

traffic_type specifies the traffic type of the line:
      0x08 = TEN_TRAFFIC_TYPE_OC192

term_sonet defines the way the OC192 will be sinked and
      sourced
      0x01 = TEN_OC192_TERM_TRANSPARENT_CBR10
      0x02 = TEN_OC192_TERM_TRANSPARENT_REGENERATOR
      0x03 = TEN_OC192_TERM_RS_LAYER_REGENERATOR
      0x01 = TEN_OC192_TERM_MS_LAYER_REGENERATOR
```

5.6.2.1.2 OTU2 Cross Connect Loopbacks

The following API configures a 10G (OTU2) cross connect loopback:

```
cs_status ten_hl_config_otu2_xcon_loopback(
      cs_uint16 module_id,
      cs_uint8  slice,
      cs_uint16 traffic_type,
      cs_uint8  dyn_repro,
      cs_uint8  term_otu,
      cs_uint8  tcm,
      cs_uint16 sysclk);
```

5.6.2.2 40G Cross Connect Loopbacks

5.6.2.2.1 OTU3 Cross Connect Loopbacks

The following API configures a 40G (OTU3) cross connect loopback:

```
cs_status ten_hl_config_otu3_xcon_loopback(
      cs_uint16 module_id,
      cs_uint16 traffic_type,
      cs_uint16 dyn_repro,
      cs_uint16 sync_mode,
      cs_uint16 term_otu,
      cs_uint16 tcm_bits)
```

Parameter description:

```
module_id specifies line's module ID
      0xXX00 = Module A
      0xXX01 = Module B


The traffic_type parameters are specified as:
      TEN_TRAFFIC_TYPE_OTU3                = 1,
      TEN_TRAFFIC_TYPE_OTU3E3              = 23,

The dyn_repo parameter is specified as:
      0 - full provision
      1 - dynamic reprovision.
```

```
The sync_mode parameter specifies the clock sync method
      0 = TEN_ASYNC_MODE
      1 = TEN_SYNC_MODE

The term_otu parameter specifies the otu3 to otu3
termination and is specified as:
      TEN_OTU_TERM_TRANSPARENT = 0x0000,
      TEN_OTU_TERM_SECTION     = 0x0001,

The tcm_bits parameters
specifies the termination for TCM 1 through 6.
This is a bit encoded parameter with bits 1 through 6
specifying TCM 1 through 6 respectively.
These bits are defined as follows:
      0 = disable
      1 = enable
```

5.6.3 Test Pattern (PRBS) Generators/Analyzers

CS604x contains a number of PRBS generators and analyzers. There are generator/analyzer pairs in HSIF, N40G, N10G, and the PP10G.

5.6.3.1 Relevant CS604x Datasheet Sections

Section 4.2: Test Pattern Generators / Analyzers
Figures 397 to 401

5.6.3.2 Relevant APIs

The relevant APIs are listed below, although their parameters will not be detailed. See the API guide for further information.

High-Level KPGA APIs introduced in Release 4.3:

```
ten_hl_config_otu2_kpga
ten_hl_config_oc192_otu2_kpga
ten_hl_config_oc192_kpga
ten_hl_config_otu3_kpga
ten_hl_config_oc768_otu3_kpga
ten_hl_config_oc768_kpga
```

High-Level HSIF APIs introduced in Release 4.3:

```
ten_hl_config_10g_hsif_kpga
ten_hl_config_40g_hsif_kpga
```

XFI interface PRBS APIs:

```
ten_xfi32x1_prbsgen_config
ten_xfi32x1_prbsgen_enable
ten_xfi32x1_prbsgen_control
ten_xfi32x1_prbsgen_set_fixed0_pattern
ten_xfi32x1_prbsgen_fixed1_pattern1
ten_xfi32x1_prbsgen_set_repeat_pattern
ten_xfi32x1_prbschk_config
ten_xfi32x1_prbschk_enable
```

```
ten_xfi32x1_prbschk_control_hunt
```

HSIF interface prbs APIs:

```
ten_hsif_slc0_mr10x4_sds_common_prbschk0_enable
ten_hsif_slc2_mr10x4_sds_common_prbschk0_enable
ten_hsif_slc1_mr10x5_sds_common_prbschk0_enable
ten_hsif_slc3_mr10x5_sds_common_prbschk0_enable
ten_hsif_slc0_mr10x4_sds_common_prbsgen0_prbs_enable
ten_hsif_slc2_mr10x4_sds_common_prbsgen0_prbs_enable
ten_hsif_slc1_mr10x5_sds_common_prbsgen0_prbs_enable
ten_hsif_slc3_mr10x5_sds_common_prbsgen0_prbs_enable
ten_hsif_slc0_mr10x4_sds_common_prbschk0_enable
ten_hsif_slc2_mr10x4_sds_common_prbschk0_enable
ten_hsif_slc0_mr10x4_sds_common_prbsgen0_prbs_enable
ten_hsif_slc2_mr10x4_sds_common_prbsgen0_prbs_enable
ten_hsif_slc1_mr10x5_sds_common_prbschk0_enable
ten_hsif_slc3_mr10x5_sds_common_prbschk0_enable
ten_hsif_slc1_mr10x5_sds_common_prbsgen0_prbs_enable
ten_hsif_slc3_mr10x5_sds_common_prbsgen0_prbs_enable
```

Datapath (KPA/KPG) prbs APIs:

```
ten_n10g_config_kpa_prbs_interval
ten_n10g_soht_prbsie
ten_n10g_config_kpa_prbs_interval
ten_n10g_soht_prbsie
ten_n40g_config_kpa_prbs_interval
ten_n40g_config_kpa_prbs_interval
```

MPIF prbs functions:

```
ten_mpif_prbs_sync
ten_mpif_prbs_sync
```

5.6.4 PBERT

The Packet BERT (PBERT) can generate and analyze packets with generated test patterns.

PBERT should be implemented with the high-level function “ten_hl_config_pbert”. This function everything required to enable and configure a PBERT generator/checker on a specific slice of the cross-connect.

Note: Using low-level APIs to configure PBERT is quite involved; there are at least 11 API calls that need to be completed. Although the APIs are listed below, it is recommended that the high-level API be used where possible.

5.6.4.1 Relevant CS604x Datasheet Sections

Section 4.2: Test Pattern Generators / Analyzers
Figure 400
Section 2.10.4: 10GPacket / Circuit BERT

5.6.4.2 Relevant APIs

The PBERT traffic-based APIs are described in the following sections. There are some special considerations that the user should be aware of when employing them.

First of all, when establishing a muxponder configuration, the PBERT path needs to be configured first as a “static” (i.e. non-dynamic) configuration, with a specific line/client pair on one path and dummy traffic on the other channels. The PBERT must be configured for the client side, with the test/reference device on the line side. Once the PBERT channel has been established, the other channels can be added.

Second, there is a special perl API available to do a sanity check for the PBERT operation. The checking API will be described below.

5.6.4.2.1 PBERT into an ODTU23

This API is used to connect an onboard PBERT to an ODTU23 for insertion into an OTU3-type datastream.

```
cs_status ten_hl_config_pbert_odtu23(
      cs_uint16 module_id_line,
      cs_uint8  slice_line,
      cs_uint16 module_id_client,
      cs_uint8  slice_client,
      cs_uint16 line,
      cs_uint16 client,
      cs_uint8  dyn_repro,
      cs_uint16 sysclk)
```

Parameter description:

```
module_id_line specifies line's module ID

slice_line specifies line's slice:
      0x00 = TEN_SLICE0
      0x01 = TEN_SLICE1
      0x02 = TEN_SLICE2
      0x03 = TEN_SLICE3

module_id_client specifies the client's module ID

line specifies the traffic type of the line:
      0x02 = TEN_TRAFFIC_TYPE_OTU3E
      0x03 = TEN_TRAFFIC_TYPE_OTU3P
      0x19 = TEN_TRAFFIC_TYPE_OTU3P2

client specifies the traffic type of the client:
      0x13 = TEN_TRAFFIC_TYPE_10GFC
      0x14 = TEN_TRAFFIC_TYPE_8GFC

slice_client  specifies client's slice:
      0x00 = TEN_SLICE0
      0x01 = TEN_SLICE1
      0x02 = TEN_SLICE2
      0x03 = TEN_SLICE3

dyn_repro  specifies the type of dynamic reprovisioning:
      0x00 = TEN_INITIAL_CONFIG
      0x01 = TEN_REPRO_CLIENT
      0x02 = TEN_REPRO_LINE_AND_CLIENT

sysclk specifies the sysclk frequency in Hz,
      For example, 400000000.
```

5.6.4.2.2 PBERT into an OTU2

This API is used to connect an onboard PBERT to an OTU2.

```
cs_status ten_hl_config_pbert_otu2(
      cs_uint16 module_id_line,
      cs_uint8  slice_line,
      cs_uint16 module_id_client,
      cs_uint8  slice_client,
      cs_uint16 client,
      cs_uint8  dyn_repro,
      cs_uint16 sysclk)
```

Parameter description:

```
module_id_line specifies line's module ID

slice_line specifies line's slice:
      0x00 = TEN_SLICE0
      0x01 = TEN_SLICE1
      0x02 = TEN_SLICE2
      0x03 = TEN_SLICE3

module_id_client specifies the client's module ID

slice_client  specifies client's slice:
      0x00 = TEN_SLICE0
      0x01 = TEN_SLICE1
      0x02 = TEN_SLICE2
      0x03 = TEN_SLICE3

client specifies the traffic type of the client:
      0x13 = TEN_TRAFFIC_TYPE_10GFC
      0x14 = TEN_TRAFFIC_TYPE_8GFC

dyn_repro  specifies the type of dynamic reprovisioning:
      0x00 = TEN_INITIAL_CONFIG
      0x01 = TEN_REPRO_CLIENT
      0x02 = TEN_REPRO_LINE_AND_CLIENT

sysclk specifies the sysclk frequency in Hz,
      For example, 400000000.
```

5.6.4.2.3 Checking Routine For Traffic-Based PBERT APIs

There is a special perl API to check PBERT operation. It should be called once the PBERT connection has been established. It will zero out PBERT counters, restart PBERT operation, wait for a while, and then check the counters.

The API will print statistics if it passes, and print "Fail" and return an error if it does not.

The API is invoked by the call:

```
check_pbert_counters($module_id, $slice)

"module_id" is the module (side) of the device, and "slice"
is the channel.
```

Note: This API runs for 30 seconds, so it shouldn't be run indiscriminately.

5.6.4.2.4 General configuration:

The general register-configuration PBERT API is described below. However, it is almost always preferable to use one of the traffic-based PBERT routines described in the previous sections. Configuring the registers is only one part of the process of establishing PBERT traffic. The user still must configure an appropriate datapath, set up muxes and clocking, and so on. This generic routine does not do that. However, it is described here in case the user has a specific need to configure PBERT registers that is not covered by the traffic-based APIs.

Use the following API to write PBERT configuration registers for traffic specified by "profile". The profile parameter denotes the associated Ethernet type; possible values are shown below.

```
cs_status ten_hl_config_pbert(
      cs_uint16 module_id,
      cs_uint8  slice,
      cs_uint16 profile,
      cs_uint16 min_len,
      cs_uint16 max_len,
      cs_uint16 ifg);
```

Parameter description:

```
The module_id parameter specifies module ID

The slice parameter specifies slice:
      0x00 = TEN_SLICE0
      0x01 = TEN_SLICE1
      0x02 = TEN_SLICE2
      0x03 = TEN_SLICE3

The profile parameter is specified as:
      0 = XCON_PBERT_PROFILE_ETH_0
      Ethernet : Transport of MAC frames (DA/SA/../FCS)
      1 = XCON_PBERT_PROFILE_ETH_1
      Ethernet : Transport of MAC frames (DA/SA/../FCS) and
      Ordered Sets
      2 = XCON_PBERT_PROFILE_ETH_2
      Ethernet : Transport of entire MAC frames
      (Preamble/SFD/DA/SA/../FCS) and Ordered Sets
      3 = XCON_PBERT_PROFILE_ETH_3
      Ethernet : Transparent transport via the IEEE
      Std 802.3-2005 Clause 49 PCS transmitter
      4 = XCON_PBERT_PROFILE_ETH_4
      Ethernet : Transparent transport via the IEEE
      Std 802.3-2005 Clause 47/48 XAUI transmitter
      5 = XCON_PBERT_PROFILE_GFP_S
      GFP-F    : transport of MAC frames (DA/SA/../FCS)
      6 = XCON_PBERT_PROFILE_GFP_T
      GFP-T    : transparent transport of entire MAC frames
      (Preamble/SFD/DA/SA/../FCS) and Ordered Sets
      7 = XCON_PBERT_PROFILE_FC
      Fibre Channel : Transport of FC frames (SOF/../EOF),
      Primitive Signals & Primitive Sequences

The min_len parameter selects the unit minimum length value
(number of octets per unit). For MODE[traffic]=OS unit
minimum length must satisfy 'minlen % 4 = 0'
When ULEN[mode]=LIN, set MINLEN=MAXLEN for fixed length
units
      0 to 0x7FFF

The max_len paramter selects the unit maximum length value
(number of octets per unit). For MODE[traffic]=OS, unit
maximum length must satisfy 'maxlen % 4 = 0'
When ULEN[mode]=LIN, set MAXLEN=MINLEN for fixed length
units
      0 to 0x7FFF

The ifg parameter is the inter frame gap specified as:
      0 = Line rate
      1 to 0x3FFF columns of inter frame gap
```

5.6.4.2.5 Low-level APIs:

(Note: Additional configuration needs to be done for the associated MAC to allow PBERT to operate correctly.)

ten_xcon_set_pbert_mode → Configures the TX/RX PACKET BERT mode
ten_xcon_control_pbert_os_payload → Controls ordered-set payload for PBERT
ten_xcon_select_bert → Selects the BERT source
ten_xcon_set_pbert_protocol → Configures the TX/RX PACKET BERT protocol settings
ten_xcon_control_pbert_frame_payload → Configures the BERT TX/RX patterns
ten_xcon_control_pbert_unit_len →Configures the PBERT TX unit length
ten_xcon_pbert_control_traffic →Controls the PBERT traffic
ten_xcon_pbert_control_ifg → Controls the PBERT IFG
ten_xcon_pbert_select_fixed_pattern → Selects specific PBERT fixed patterns

5.6.5 CBERT

The Circuit BERT (CBERT) is a circuit-based test pattern generator/analyzer located in the cross-connect block. It can be used in circuit applications to generate and check test patterns.

Enabling and using CBERT is somewhat complicated. However, there are high-level APIs available to simplify the implementation. They should be employed if possible if CBERT is desired. Low-level APIs are listed below, but their proper use is beyond the scope of this document.

5.6.5.1 Relevant CS604x Datasheet Sections

Section 43.2: Test Pattern Generators / Analyzers
Figure 400
Section 2.10.4: 10G Packet / Circuit BERT

5.6.5.2 Relevant APIs

High-level APIs:

To set up the CBERT checker, use:

```
ten_hl_xcon_set_cbert_checker(dev_id, channel)
```

To set up the CBERT generator, use:

```
ten_hl_xcon_set_cbert_generator(dev_id, channel)
```

To check the CBERT error status, use:

```
ten_hl_xcon_check_cbert(dev_id, channel, threshold)
```

For all of the high-level APIs, the dev_id is the device ID, channel is the channel to apply CBERT to (from 0..7, where 0 is A0 and 7 is B3), and threshold is the minimum number of errors that CBERT must have detected to return an error.

Low-level APIs:

```
ten_xcon_cbert_config → Set CBERT PRBS configuration
ten_xcon_cbert_config_err_threshold → Set CBERT PRBS Config
Error Threshold
ten_xcon_cbert_control_checker → ENABLE/DISABLE CBERT
checker
ten_xcon_cbert_control_generator → Enable/disable CBERT
generator
ten_xcon_cbert_get_sync_status →Get status of CBERT
interrupt
ten_xcon_cbert_inject_err → Inject error in CBERT TX stream
ten_xcon_cbert_resync → Reset checker into Hunt state
```

5.7 High-Resolution Software Timer

Software timers in the CS604x driver—TEN_MDELAY() and TEN_UDELAY()—are dependent on the API's and chip's run time environment, so their performance is nondeterministic and varies widely between platforms. Beginning with software release 5.3, to create consistent timer behavior in a diverse array of implementations, the API sets a 6ms floor for these timers. This quality assurance step exposed a requirement for a timer with finer resolution for some applications. This high resolution timer must operate consistently in most environments to ensure that the driver operates in the customer environment as tested in Cortina's development environment.

This feature provides a new (for release 5.5) software timer which takes advantage of the CS604x MPIF miscellaneous counter registers. This counter increments at a rate proportional to the system clock.

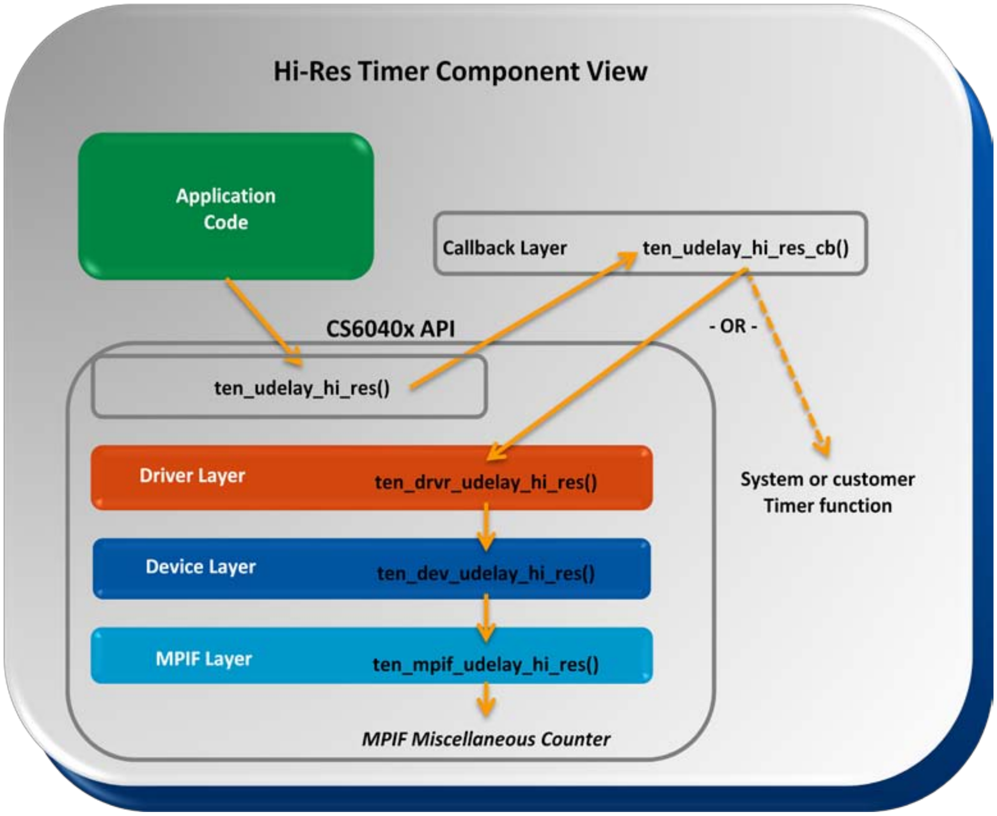
The new timer can be accurate down to 2µS, depending on hosting platform. The timer is implemented in the MPIF software block as a busy-wait timer. It is available at several software levels as shown.

API	Function	API doc?	Purpose
MPIF	ten_mpif_udelay_hi_res(dev_id, usecs)	Yes	Implementation
Device	ten_dev_udelay_hi_res(usecs)	No	Knows which device
Driver	ten_drvr_udelay_hi_res(usecs)	No	Failsafe, reverts to TEN_UDELAY()
System	ten_udelay_hi_res(usecs)	Yes	Generic Access
Call-back	ten_udelay_hi_res_cb(usecs)	No	Default registration is to ten_drvr_udelay_hi_res, can be re-registered.

Since this is a busy-wait timer, it is encapsulated in a call-back construct so it can be substituted with a customer-provided timer utility, if desired.

Calibration and other setup is required. All of this is done by default as driver APIs are called to initialize and configure the chip. It can also be done manually on demand.

Figure 14 High Resolution Timer Component View



5.7.1 Timer Core

5.7.1.1 Description

The timer's core is a busy-wait timer which monitors the MPIF Miscellaneous Counter. This counter is a 64-bit, 4-register free-running counter that increments with every MPIF clock cycle. The MPIF clock is ¼ of the system clock. Only the lower two registers are used for this counter, though the high-word register must be read to clock the current counter value into the remaining registers.

The timer automatically operates in three different modes, depending on the register-read access time (measured during calibration), system clock rate, and the requested duration.

1. Short duration: When the requested duration is short compared to the measured access time, the timer issues a calculated number of blind register reads to approximate the desired delay. Since this scheme is prone to overruns, it is tuned to under-run about 1 in 4 times to improve average accuracy.

   This mode is used when the requested time is less than about 10 times the access time.

2. Standard duration, slow access: This mode is used when the measured access time is very long, as it might be if register access is via a LAN or encumbered by printed output. In this mode, the timer polls the MPIF

counter but does not read the counter's least significant word. This reduces error introduced by long read times.

3. Standard duration, normal access. The timer reads the lower 32 bits of the counter in a polling loop until the requested duration has passed.

Before the timer can be used the first time, it must be 'calibrated'. This is done by calling function ten_mpif_init_access_times(). Note that this function must be provided with the frequency of the system clock. Calibration values are stored in a new 'mpif' block of the device control block.

The APIs used at this level are:

Timer: ten_mpif_udelay_hi_res(dev_id, usecs)

Calibration: ten_mpif_init_access_times(dev_id, sysclk_freq)

These functions are described in detail in their function headers.

The system clock in MHz must be passed as an argument into the calibration function `ten_mpif_init_access_times()` because prior to this feature the system clock rate is not stored by the driver.

5.7.2 Device Layer

5.7.2.1 Device level wrapper

This timer must be accessible as if it were a 'system' timer, so its higher level APIs must not include a parameter for device ID. In a multi-device system the driver must select a CS604x to use as the timer core, and then fill in the device ID for the MPIF timer. The device layer is where this is done.

During device initialization, the first device to be initialized that can be used is selected. When subsequent devices are initialized, they do not overwrite the selection.

The device chosen as the core for the timer is maintained as a private global in ten_dev.c. This is set automatically by ten_dev_init(). To provide a mechanism to substitute during device reset on multi-chip systems, it can also be set manually by API `ten_dev_udelay_high_res_device().`

The device level timer function is `ten_dev_udelay_hi_res(usecs)`. Though it is public, it is not intended to be part of the driver API, so it is not included in driver documentation as an API. Application code should access the timer via the documented APIs.

5.7.2.2 System Clock

To provide automatic bring up and calibration, the high res timer needs to know the MPIF cycle frequency. This is derived from the system clock frequency, which until now has not been stored by the API. This is now added to the device control block, along with the new mpif sub-control block.

System clock frequency can be set using new API `ten_dev_update_sysclock()`. This API checks to see if the system clock frequency has changed, and if so, calls additional functions to process system clock dependencies as needed. For example, when the system clock is updated, the high resolution timer is recalibrated.

To ensure that the system clock value is captured and set, this new API is called from within `ten_hl_fracdiv_config()`, which is called with the correct system clock for every functional configuration.

5.7.2.3 Notes

The device to be used for the high res timer is selected automatically. If, on a multi-device system, that device is taken out of service, the **system** must select a new CS604x device for this timer using the facilities provided by this feature

There are no device checks to ensure that the device is out of reset. If the selected device is taken out of operation after system initialization, counter reads will loop until they time out. This timeout is calculated to be about 25% longer than the requested delay period, but this is a safety mechanism with no claims to accuracy.

A warning message is printed when the timer fails or times out.

5.7.3 Driver Layer

The high res timer must work at the system level even if no device has been initialized. For this reason, the device-level timer—`ten_dev_udelay_hi_res()`—is wrapped in the driver layer function `ten_drvr_udelay_hi_res()`. If the call to the device timer fails, it means that no supporting device has been initialized, so this function substitutes the legacy timer `TEN_UDELAY(usecs)`.

5.7.4 System Layer

The new high res timer can be accessed from several driver layers. However, system and application code should access it using system function `ten_udelay_hi_res(usecs)`.

This function abstracts the timer call through a call-back facility. On driver load, function `ten_register_udelay_hi_res()` is registered for callback.

The default callback can be overwritten in systems that have a good OS or other system timer, as discussed in the function description in the code excerpt above.

There are two replacement options.

a) If there is a better system call than that used for TEN_UDELAY(), register it for callback, or

b) If the same system call made by TEN_UDELAY() is capable of higher resolution than the 6ms floor in TEN_UDELAY(), register it for callback to get around the TEN_UDELAY( ) constraint.

To be registered for callback, the timer utility must have, or be wrapped in a function with, a C signature of the form “`void my_timer_function(cs_uint32 usecs)`”.

5.7.5 API Function Summary

These are the functions added to support the software high-resolution timer.

Layer	Function	API doc	Purpose
MPIF	**ten_mpif_udelay_hi_res (dev_id, usecs)**	Yes	Implementation
	ten_mpif_init_access_times (dev_id, sysclk_freq)	Yes	Calibration, can be called directly, is called for device initialization and sysclock update.
Device	ten_dev_udelay_hi_res(usecs)	No	Knows which device
	ten_dev_udelay_high_res_device ()	Yes	Override auto-selected device to use for timer.
	ten_dev_update_sysclock()	Yes	Updates system clock in CB, calls functions to process sysclk dependencies. Can be called directly. Is also called by ten_hl_fracdiv_config().
Driver	ten_drvr_udelay_hi_res(usecs)	No	Failsafe, reverts to TEN_UDELAY()
System	**ten_udelay_hi_res(usecs)**	Yes	Generic Access, calls
	ten_udelay_hi_res_cb(usecs)	No	Default registration is to ten_drvr_udelay_hi_res, during driver load. Other functions can be registered for this.
	ten_register_udelay_hi_res (my_timer_function)	Yes	Registers my_timer_function to be called by ten_udelay_hi_res_cb()
	ten_unregister_udelay_hi_res()	Yes	Unregisters the function registered to ten_udelay_hi_res_cb
	ten_is_registered_udelay_hi_res()	Yes	Test to prevent overwrites.

5.7.6 Compatibility with CS600x (T40)

This feature has value for CS600x systems. The callback feature can be used to replace the default call to ten_`drvr_udelay_hi_res()`. There are two replacement options.

c) If there is a better system call than that used for TEN_UDELAY(), register it for callback, or

d) If the same system call made by TEN_UDELAY() is capable of higher resolution than the 6ms floor in TEN_UDELAY(), register it for callback to get around the TEN_UDELAY( ) constraint.

This feature is fully compatible with CS600x systems even when a new callback for `ten_udelay_hi_res()` is not registered. CS604x register reads are avoided in a couple of ways.

a) For direct calls to the timer's core `ten_mpif_udelay_hi_res()`: Calibration of the timer's core is not allowed for CS600x devices. Individual calls to the timer's core can only run on CS604x because the timer function will not run when the timer is not calibrated. Therefore, an explicit check in the core timer function for CS6040x is not necessary.

b) For high-level calls to `ten_udelay_hi_res()`**:** When the callback for API `ten_udelay_hi_res()`hasn't been changed , by default it calls `ten_drvr_-udelay_hi_res().` When there is no CS604x, its call to `ten_dev_udelay_hi_-res()` fails. A warning is printed and TEN_UDELAY() is called.

During setup, a device can't be selected to be the timer unless it's a T41, either by the automatic init code or by explicit selection with `ten_dev_udelay_high_res_device().`

5.7.7 How to Use the Software Timer

For clarity, this procedure references discreet stages of initialization. However, it is possible to bring up the device and driver completely and then go back through to set up this feature as desired.

5.7.7.1 If a system timer is to replace the default

1. In application code, change timer function calls, as needed, from TEN_UDELAY(usecs) to ten_udelay_hi_res(usecs).
2. Compile and build.
3. Start the driver (e.g. `ten_drvr_load());`
4. Call ten_unregister_udelay_hi_res();
5. Call ten_register_udelay_hi_res(<*my delay function*>);
6. Continue initialization, you're done.

5.7.7.2 If the default timer is to be used

1. In application code, change timer function calls, as needed, from TEN_UDELAY(usecs) to ten_udelay_hi_res(usecs).
2. Compile and build.
3. Start the driver (e.g. `ten_drvr_load()`);
4. Initialize your devices (e.g. `ten_dev_init()`);
5. Call ten_dev_update_sysclock() for each device that might be used for the timer. *Note: In case this step is skipped, the T41 driver harvests the system clock rate from calls to ten_hl_fracdiv_config(). If system clock frequency passed in here is different that the stored frequency, the device's system clock setting is updated and timer calibration is kicked off.*
6. Call ten_dev_udelay_high_res_device() to select which device will be used. *Note: This is necessary only if the system-selected device is not preferred.*

7. Call ten_mpif_init_access_times() to calibrate the timer for operation in its environment. *Note: This is only necessary if the device selected for the timer is changed, or if recalibration is desired.*

8. Continue initialization, you're done.

5.7.7.3 When the selected device is to be taken out of service

This is relevant only for multi-device systems.

1. Initialize the replacement device if necessary (e.g. `ten_dev_init()`);

2. Call ten_dev_update_sysclock() for the next device to be used for the timer. *Note: In case this step is skipped, the T41 driver harvests the system clock rate from calls to ten_hl_fracdiv_config(). If system clock frequency passed in here is different that the stored frequency, the device's system clock setting is updated and timer calibration is kicked off*.

3. Call ten_mpif_init_access_times() to for that device to calibrate the timer for operation in its environment. *Note: This is only necessary if the device selected for the timer is changed, or if recalibration is desired.*

4. Call ten_dev_udelay_high_res_device() to select the device to be used.

5. You're done.

6.0 Block Functions

6.1 Functional Block Configuration

The main functional blocks of CS604x (PP10G, N10G, and N40G) each have high-level APIs that can configure the block in accordance with the desired connections and traffic.

Configuring the block is not the same as configuring traffic through the block. Once the block configurations are done, the user must still call traffic-configuration APIs in order to send and receive.

6.1.1 Common Block Functions

There are several new high-level functions that apply to all of the functional blocks. They are described here.

6.1.1.1 Soft Resets

These APIs remove soft resets from N10G, PP10G, ES, and SYNCDSYNC, depending on the traffic. They assume that the line and client slices are the same for aggregation.

(Note: The ten_hl_config_* functions are written in C, while the config_* functions are written in perl. Though the descriptions may imply that they do the same things, this is not the case; the C functions may not support everything that the perl functions do. This will be noted in the descriptions. Also, the data types for the parameters may differ.)

6.1.1.1.1 10G Reset-Control APIs

There are several different high-level APIs available for 10G reset assertion/removal. Two of them are listed here.

Traffic-Specific Soft Reset Removal

The following function removes the 10G functional block soft resets, removing soft resets from N10G, PP10G, ES, and SYNCDSYNC depending on the traffic. The API assumes that the line and client slice are the same for aggregation.

```
cs_status ten_hl_config_remove_soft_resets(
      cs_uint16 mod_line,
      cs_uint8 slice_line,
      cs_uint16 mod_client,
      cs_uint8 slice_client,
      cs_uint16 client,cs_uint16 aggr)
```

Parameter description:

```
slice_line
      Specified as:
      0x00 = TEN_SLICE0
      0x01 = TEN_SLICE1
      0x02 = TEN_SLICE2
      0x03 = TEN_SLICE3
      0xFF = TEN_SLICE_ALL.

slice_client
      Specified as:
      0x00 = TEN_SLICE0
      0x01 = TEN_SLICE1
      0x02 = TEN_SLICE2
      0x03 = TEN_SLICE3
      0xFF = TEN_SLICE_ALL.

client
      The type of traffic. One of:
      TEN_TRAFFIC_TYPE_10GE
      TEN_TRAFFIC_TYPE_10GE_6_1
      TEN_TRAFFIC_TYPE_10GE_6_2
      TEN_TRAFFIC_TYPE_10GE_7_1
      TEN_TRAFFIC_TYPE_10GE_7_2
      TEN_TRAFFIC_TYPE_10GE_7_3
      TEN_TRAFFIC_TYPE_10GE_GFPF_OC192_ODU2
      TEN_TRAFFIC_TYPE_10GE_RA
      TEN_TRAFFIC_TYPE_10GFC
      TEN_TRAFFIC_TYPE_8GFC
      TEN_TRAFFIC_TYPE_OC192
      TEN_TRAFFIC_TYPE_OTU1E
      TEN_TRAFFIC_TYPE_OTU1F
      TEN_TRAFFIC_TYPE_OTU2
      TEN_TRAFFIC_TYPE_OTU2E

aggr
      1 = if this is aggregating into an ODTU23 and the
      other module's resets need to be removed also.
      0 = otherwise.
```

Selectable Soft Reset Assertion/Removal

The following API sets or removes 10G functional block soft resets.

It performs a soft reset on the specified (masked on) selections. The selections that are masked off will not be affected.

```
cs_status ten_n10g_set_global_resets(
      cs_uint16 module_id,
      cs_uint8 slice,
      cs_reset_action_t act,
      cs_uint16 bitfield)
```

Parameter description:

```
The slice parameter is specified as:
      0x00 = TEN_SLICE0
      0x01 = TEN_SLICE1
      0x02 = TEN_SLICE2
      0x03 = TEN_SLICE3
      0xFF = TEN_SLICE_ALL

The act parameter specifies the reset option and is one
of the following:
      CS_RESET_DEASSERT
      CS_RESET_ASSERT
      CS_RESET_TOGGLE
      The CS_RESET_TOGGLE option will assert reset, hold
      the reset for a while and then de-assert the reset
      bit(s).

The bitfield parameter is specified as:
      TEN_N10G_GLOBAL_RESETS_RSTTX    = 0x0001,
      TEN_N10G_GLOBAL_RESETS_RSTTXNG  = 0x0002,
      TEN_N10G_GLOBAL_RESETS_RSTOT    = 0x0004,
      TEN_N10G_GLOBAL_RESETS_RSTCT    = 0x0008,
      TEN_N10G_GLOBAL_RESETS_RSTRX    = 0x0010,
      TEN_N10G_GLOBAL_RESETS_RSTOR    = 0x0020,
      TEN_N10G_GLOBAL_RESETS_RSTCR    = 0x0040,
      TEN_N10G_GLOBAL_RESETS_RSTSYS   = 0x0100,
      TEN_N10G_GLOBAL_RESETS_ALL      = 0x017F
```

6.1.1.1.2 40G Reset-Control APIs

The following API removes 40G functional block soft resets.

It performs a soft reset on the specified (masked on) selections. The selections that are masked off will not be affected.

```
cs_status ten_n40g_set_global_resets(
      cs_uint16 module_id,
      cs_reset_action_t act,
      cs_uint16 bitfield)
```

Parameter description:

```
The module_id is the side of the device.

The act parameter specifies the reset option and is one of
the following:
      CS_RESET_DEASSERT
      CS_RESET_ASSERT
      CS_RESET_TOGGLE → The CS_RESET_TOGGLE option will
      assert reset, hold the reset for a while and then
      deassert the reset bit(s).

The bitfield parameter is specified as:
      TEN_N40G_GLOBAL_RESETS_RSTTX  = 0x0001
      TEN_N40G_GLOBAL_RESETS_RSTOT  = 0x0002
      TEN_N40G_GLOBAL_RESETS_RSTST  = 0x0004
      TEN_N40G_GLOBAL_RESETS_RSTRX  = 0x0010
      TEN_N40G_GLOBAL_RESETS_RSTOR  = 0x0020
      TEN_N40G_GLOBAL_RESETS_RSTSR  = 0x0040
```

```
        TEN_N40G_GLOBAL_RESETS_RSTSYS = 0x0100
        TEN_N40G_GLOBAL_RESETS_ALL    = 0x0177
```

6.1.2 PP10G

6.1.2.1 Relevant CS604x Datasheet Sections

2.2: Mapping of Data Client Signals
2.8: Receive 10G Packet Processor
2.12: Transmit 10G Packet Processor

6.1.2.2 Associated APIs

```
cs_status ten_hl_pp10g_config(
      cs_uint16 module_id,
      cs_unit8 slice,
      cs_uint8 mode_rx,
      cs_uint8 mode_tx)
```

Parameter description:

```
[slice]
      One of:
      0x00 = TEN_SLICE0
      0x01 = TEN_SLICE1
      0x02 = TEN_SLICE2
      0x03 = TEN_SLICE3
      0xFF = TEN_SLICE_ALL

[mode_rx]
      The 'mode' parameter for ten_hl_pp10g_rx_init; one
      of:
            TEN_PP10G_RX_DISABLE = 0,
            TEN_PP10G_RX_ETH_0 (no Idles or Ordered Sets
            transported for GFP-F mapping) = 1,
            TEN_PP10G_RX_ETH_2 (Ordered Sets but no idles
            transported for GFP-T mapping) = 2,
            TEN_PP10G_RX_ETH_4 (Idles and Ordered Sets
            processed for Rate Adjust) = 3,
            TEN_PP10G_RX_GFP_S = 4,
            TEN_PP10G_RX_GFP_T = 5,
            TEN_PP10G_RX_ETH2_GFP_T = 6,
            TEN_PP10G_RX_ETH0_GFP_S = 7,
            TEN_PP10G_RX_XAUI_FC = 8,
            TEN_PP10G_RX_FC = 9,
            TEN_PP10G_RX_8FC = 10,
            TEN_PP10G_RX_XC_GE = 11,
            TEN_PP10G_RX_XC_FC = 12,
            TEN_PP10G_RX_ETH_4_RA = 13,
            TEN_PP10G_RX_FC_RA = 14,
            TEN_PP10G_RX_8FC_RA = 15,
            TEN_PP10G_RX_XAUI_ETH_4_RA = 16,
            TEN_PP10G_RX_XAUI_ETH2_GFP_T = 17,
            TEN_PP10G_RX_XAUI_FC_RA = 18,
            TEN_PP10G_RX_XAUI_ETH0_GFP_S = 19.
```

```
[mode_tx]
      The 'mode' parameter for ten_hl_pp10g_tx_init; one
      of:
            TEN_PP10G_TX_DISABLE = 0,
            TEN_PP10G_TX_ETH_0 = 1,
            TEN_PP10G_TX_ETH_2 = 2,
            TEN_PP10G_TX_ETH_4 = 3,
            TEN_PP10G_TX_GFP_S = 4,
            TEN_PP10G_TX_GFP_T = 5,
            TEN_PP10G_TX_GFP_T_ETH2 = 6,
            TEN_PP10G_TX_GFP_S_ETH0 = 7,
            TEN_PP10G_TX_XAUI_FC = 8,
            TEN_PP10G_TX_FC = 9,
            TEN_PP10G_TX_8FC = 10,
            TEN_PP10G_TX_XC_FC = 11,
            TEN_PP10G_TX_XC_GE = 12,
            TEN_PP10G_TX_XAUI_ETH_4 = 13,
            TEN_PP10G_TX_XAUI_GFP_T_ETH2 = 14,
            TEN_PP10G_TX_XAUI_GFP_S_ETH0 = 15.
```

6.1.2.3 Additional Concerns: Enabling Jumbo Frames

The default maximum frame length is 1518 bytes. To enable Jumbo Frames (up to 9.6KB) execute the following commands.

```
ten_pp10g_mac_rx_maxlen( module_id, slice, 9600 );
ten_pp10g_xgadj_tx_maxframe( module_id, slice, 0, 9600 );
```

6.1.3 N10G

ten_hl_n10g_config() provisions the N10G in accordance with the selected RX and TX configurations. In addition, the user can specify that the API is doing either a “full” provisioning or dynamic reprovisioning.

6.1.3.1 Relevant CS604x Datasheet Sections

6.1.3.2 Associated APIs

```
cs_status ten_hl_n10g_config(
      cs_uint16 module_id,
      cs_unit8 slice,
      cs_uint8 mode_rx,
      cs_uint8 mode_tx,
      cs_unit16 dyn_repro)
```

Parameter description:

```
[slice]
      One of:
      0x00 = TEN_SLICE0
      0x01 = TEN_SLICE1
      0x02 = TEN_SLICE2
      0x03 = TEN_SLICE3
      0xFF = TEN_SLICE_ALL
```

```
[mode_rx]
      The 'mode' parameter for ten_hl_n10g_rx_init; one of:
            TEN_N10G_RX_DISABLE = 0,
            TEN_N10G_RX_BYPASS = 1,
            TEN_N10G_RX_OTU2_ODU2_FS = 2,
            TEN_N10G_RX_OTU2_U15_10GFC = 3,
            TEN_N10G_RX_OTU2_OPU2 = 4,
            TEN_N10G_RX_ODU2J_8GFC = 5,
            TEN_N10G_RX_OTU2_8GFC = 6,
            TEN_N10G_RX_ODU2_ODU2 = 7,
            TEN_N10G_RX_OC192_OC192 = 8,
            TEN_N10G_RX_OXU3J_ODU2E_10GE = 9,
            TEN_N10G_RX_OTU2E_ODU2 = 10,
            TEN_N10G_RX_OXU3_OC192 = 11,
            TEN_N10G_RX_OXU3_OC192P = 12,
            TEN_N10G_RX_OXU3_ODU2 = 13,
            TEN_N10G_RX_OXU3_ODU2P = 14,
            TEN_N10G_RX_OXU3_10GEP = 15,
            TEN_N10G_RX_OXU3_10GE = 16,
            TEN_N10G_RX_OXU3_10GFC = 17,
            TEN_N10G_RX_OXU3_OC192_10GE = 18,
            TEN_N10G_RX_OTU2E_ODU2E = 19,
            TEN_N10G_RX_OTU1E_ODU1E = 20,
            TEN_N10G_RX_OTU2_ODU2 = 21,
            TEN_N10G_RX_OTU2E_10GE = 22,
            TEN_N10G_RX_OXU3_ODU2E_10GE = 23,
            TEN_N10G_RX_OTU2FC_10GFC = 24,
            TEN_N10G_RX_OXU3_ODU2FC_10GFC = 25,
            TEN_N10G_RX_OTU1E_10GE = 26,
            TEN_N10G_RX_OXU3_ODU1E_10GE = 27,
            TEN_N10G_RX_OTU2EJ_10GE = 28,
            TEN_N10G_RX_OTU2EJ_10GFC = 29,
            TEN_N10G_RX_OTU2XJ_10GE = 30,
            TEN_N10G_RX_OTU2EJ_8GFC = 31,
            TEN_N10G_RX_ODU2_OC192 = 32,
            TEN_N10G_RX_OTU2EJ_OC192 = 33,
            TEN_N10G_RX_OTU2_10GE = 34,
            TEN_N10G_RX_OTU2_10U_ODU2 = 35,
            TEN_N10G_RX_OTU2_OC192_10GE = 36,
            TEN_N10G_RX_OC192_10GE = 37,
            TEN_N10G_RX_OTU2_OC192 = 38,
            TEN_N10G_RX_BYPASS_P = 39,
            TEN_N10G_RX_OTU2_OC192PT = 40,
            TEN_N10G_RX_OXU3_ODU2_10GE = 41,
            TEN_N10G_RX_OXU3_192_10GE = 42,
            TEN_N10G_RX_OTU2_34_10GE = 43,
            TEN_N10G_RX_OTU2_OC192_10GE_20U = 44,
            TEN_N10G_RX_OTU2_GENRM_ODU3 = 45,
            TEN_N10G_RX_ODU3_GENRM_OTU2 = 46

[mode_tx]
      The 'mode' parameter for ten_hl_n10g_tx_init; one of:
            TEN_N10G_TX_DISABLE = 0,
            TEN_N10G_TX_BYPASS = 1,
            TEN_N10G_TX_10GFC_OTU2_U15 = 2,
            TEN_N10G_TX_OPU2_OTU2 = 3,
            TEN_N10G_TX_ODU2_ODU2 = 4,
            TEN_N10G_TX_8GFC_OTU2 = 5,
            TEN_N10G_TX_8GFC_ODU2J = 6,
            TEN_N10G_TX_10GE_ODU2E_OXU3J = 7,
            TEN_N10G_TX_10GE_OTU2E = 8,
            TEN_N10G_TX_10GE_ODU2E_OXU3 = 9,
            TEN_N10G_TX_10GE_OTU1E = 10,
            TEN_N10G_TX_10GE_ODU1E_OXU3 = 11,
            TEN_N10G_TX_10GFC_OTU2FC = 12,
            TEN_N10G_TX_10GFC_ODU2FC_OXU3 = 13,
            TEN_N10G_TX_OC192_OXU3 = 14,
            TEN_N10G_TX_OC192_OXU3P = 15,
            TEN_N10G_TX_ODU2_OXU3 = 16,
            TEN_N10G_TX_10GFC_OXU3 = 17,
            TEN_N10G_TX_10GE_OXU3P = 18,
            TEN_N10G_TX_10GE_OXU3 = 19,
            TEN_N10G_TX_ODU2_OXU3P = 20,
            TEN_N10G_TX_OC192_ODU2 = 21,
            TEN_N10G_TX_OC192_OTU2EJ = 22,
            TEN_N10G_TX_10GE_OTU2 = 23,
            TEN_N10G_TX_10GE_OTU2XJ = 24,
            TEN_N10G_TX_10GE_OTU2EJ = 25,
            TEN_N10G_TX_10GFC_OTU2EJ = 26,
            TEN_N10G_TX_8GFC_OTU2EJ = 27,
            TEN_N10G_TX_ODU1E_OTU1E = 28,
            TEN_N10G_TX_ODU2_OTU2 = 29,
            TEN_N10G_TX_ODU2E_OTU2E = 30,
            TEN_N10G_TX_ODU2_OTU2_10U = 31,
            TEN_N10G_TX_10GE_OC192_OTU2 = 32,
            TEN_N10G_TX_10GE_OC192 = 33,
            TEN_N10G_TX_OC192_OTU2 = 34,
            TEN_N10G_TX_OC192_OC192PT = 35,
            TEN_N10G_TX_10GE_192_OXU3 = 36,
            TEN_N10G_TX_10GE_ODU2_OXU3 = 37,
            TEN_N10G_TX_10GE_ODU2E_OXU3_DT = 38,
            TEN_N10G_TX_10GE_OTU2_34 = 39,
            TEN_N10G_TX_OTU2_OXU3_NT = 40,
            TEN_N10G_TX_FC_TRANS_OXU3_NT = 41,
            TEN_N10G_TX_OTU2E_OXU3_NT = 42,
            TEN_N10G_TX_OC192_OTU2_10U = 43,
            TEN_N10G_TX_10GE_OC192_OTU2_20U = 44,
            TEN_N10G_TX_OTU2_GENRM_ODU3 = 45,
            TEN_N10G_TX_ODU3_GENRM_OTU2 = 46.
[dyn_repro]
```

0 = full provision

1 = dynamic reprovision

6.1.4 N40G

APIs that Configure an N40G transmit/receive path.

ten_hl_n40g_config() sets up the N40G blocks so that they can carry traffic of the types indicated by the “mode” fields. Not all possible mode combinations are supported, since there are dozens of possible combinations and they cannot all be tested.

Table 9 N40G Receive Modes

Mode	Description	Example in Config
TEN_N40G_RX_BYPASS	Bypass the N40G receiver	20b
TEN_N40G_RX_OTU3_ODTU23TEN_N40G_RX_OTU3_ODU3_ORX	OTU3 → ODTU23	11
TEN_N40G_RX_OC768_ODU3TEN_N40G_RX_OTU3_ODTU23	OC768 → ODU3OTU3 → ODTU23	2611
TEN_N40G_RX_ODU3_ODU3TEN_N40G_RX_OC768_ODU3	ODU3 → ODU3OC768 → ODU3	26
TEN_N40G_RX_OTU3_ODTU23PTEN_N40G_RX_OTU3_ODTU23P_T21TEN_N40G_RX_OTU3_ODU3_ORX	OTU3 → ODTU23 with Cortina Enhanced Mapping	1
TEN_N40G_RX_OTU3_ODTU23P14TEN_N40G_RX_OTU3_ODTU23PTEN_N40G_RX_OTU3_ODTU23	OTU3 → ODTU23 with Cortina Enhanced MappingOTU3 → ODTU23	111
TEN_N40G_RX_OTU3_ODU3_LSTEN_N40G_RX_OTU3_ODTU23P14TEN_N40G_RX_OC768_ODU3	OC768 → ODU3	26
TEN_N40G_RX_OTU3_ODTU23P3TEN_N40G_RX_OTU3_ODU3_LSTEN_N40G_RX_ODU3_ODU3	OTU3e → ODTU23ODU3 → ODU3	3 or 12e
TEN_N40G_RX_OTU3_ODTU23P7TEN_N40G_RX_OTU3_ODTU23P3TEN_N40G_RX_OTU3_ODU3	OTU3+ (44.569 GHz) → ODTU23OTU3e → ODTU23OTU3 → ODU3	73 or 12e
TEN_N40G_RX_ODU3_ODTU23TEN_N40G_RX_OTU3_ODTU23P7TEN_N40G_RX_OTU3_ODTU23P_T21	ODU3 → ODTU23OTU3+ (44.569 GHz) → ODTU23	247
TEN_N40G_RX_OTU2_GENRM_ODU3TEN_N40G_RX_ODU3_ODTU23TEN_N40G_RX_OTU3_ODTU23P	ODU3 → ODTU23OTU3 → ODTU23 with Cortina Enhanced Mapping	241
TEN_N40G_RX_OTU3E_ODTU23PTEN_N40G_RX_OTU3_ODTU23P3	OTU3e → ODTU23 with Cortina Enhanced MappingOTU3e → ODTU23	3 or 12e
TEN_N40G_RX_OTU3_ODTU23P7	OTU3+ (44.569 GHz) → ODTU23	7
TEN_N40G_RX_ODU3_ODTU23	ODU3 → ODTU23	24

Table 10 N40G Transmit Modes

Mode	Description	Example in Config
TEN_N40G_TX_BYPASS	Bypass the N40G transmitter	20b
TEN_N40G_TX_ODTU2_OTU3	ODTU23 → OTU3	11
TEN_N40G_TX_ODTU2_OTU3P3	ODTU23 → OTU3e	3 or 12x

Mode	Description	Example in Config
TEN_N40G_TX_ODTU2_OTU3P7	ODTU23 → OTU3+ (44.569 GHz)	7
TEN_N40G_TX_ODTU2_ODU3	ODTU23 → ODU3	24

6.1.4.1 Relevant CS604x Datasheet Sections

Section 2.6: Receive 40G Circuit Processor

Section 2.14: Transmit 40G Circuit Processor

6.1.4.2 Associated APIs

ten_hl_n40g_config: This API configures the RX and TX to enable specific traffic types. It sets all required register bits and muxes to enable the chosen data type.

RX and TX can be configured differently.

6.1.4.2.1 Key High-level API Call and Parameters

```
cs_status ten_hl_n40g_config(
      cs_uint16 module_id,
      cs_uint8 mode_rx,
      cs_uint8 mode_tx)
```

Parameter description:

```
mode_rx
      The 'mode' parameter for ten_hl_n40g_rx_init; one of:
            TEN_N40G_RX_DISABLE = 0,
            TEN_N40G_RX_BYPASS = 1,
            TEN_N40G_RX_BYPASS_SNT_MON = 2,
            TEN_N40G_RX_OTU3_ODTU23_10U_6J = 3,
            TEN_N40G_RX_OTU3_ODTU23_10U = 4,
            TEN_N40G_RX_OTU3_ODTU23_7U = 5,
            TEN_N40G_RX_OTU3_ODU3_ORX = 6,
            TEN_N40G_RX_OTU3_ODTU23 = 7,
            TEN_N40G_RX_OC768_ODU3 = 8,
            TEN_N40G_RX_ODU3_ODU3 = 9,
            TEN_N40G_RX_OTU3_ODU3 = 10,
            TEN_N40G_RX_OTU3_ODTU23P_T21 = 11,
            TEN_N40G_RX_OTU3_ODTU23P = 12,
            TEN_N40G_RX_OTU3_ODTU23P14 = 13,
            TEN_N40G_RX_OTU3_ODU3_LS = 14,
            TEN_N40G_RX_OTU3_ODTU23P3 = 15,
            TEN_N40G_RX_OTU3_ODTU23P7 = 16,
            TEN_N40G_RX_ODU3_ODTU23 = 17,
            TEN_N40G_RX_OTU2_GENRM_ODU3 = 18,
            TEN_N40G_RX_ODU3_GENRM_OTU2 = 19,
            TEN_N40G_RX_OTU3E_ODTU23P = 20

mode_tx
```

```
The 'mode' parameter for ten_hl_n40g_tx; one of:
      TEN_N40G_TX_DISABLE = 0,
      TEN_N40G_TX_BYPASS = 1,
      TEN_N40G_TX_BYPASS_SNT_MON = 2,
      TEN_N40G_TX_ODTU2_OTU3_10U_6J = 3,
      TEN_N40G_TX_ODTU2_OTU3_10U = 4,
      TEN_N40G_TX_ODTU2_OTU3_7U = 5,
      TEN_N40G_TX_ODU3_OTU3_ORV = 6,
      TEN_N40G_TX_ODTU2_OTU_T21A = 7,
      TEN_N40G_TX_ODTU2_OTU3 = 8,
      TEN_N40G_TX_ODTU2_OTU3P_T21 = 9,
      TEN_N40G_TX_ODTU2_OTU3P = 10,
      TEN_N40G_TX_ODTU2_OTU3P14 = 11,
      TEN_N40G_TX_OC192_OTU3 = 12,
      TEN_N40G_TX_ODU3_ODU3 = 13,
      TEN_N40G_TX_ODU3_OTU3 = 14,
      TEN_N40G_TX_ODU3_OTU3_SL = 15,
      TEN_N40G_TX_ODTU2_OTU3P3 = 16,
      TEN_N40G_TX_ODTU2_OTU3P7 = 17,
      TEN_N40G_TX_ODTU2_ODU3 = 18,
      TEN_N40G_TX_OTU2_GENRM_ODU3 = 19,
      TEN_N40G_TX_ODU3_GENRM_OTU2 = 20,
      TEN_N40G_TX_OTU3E_ODTU23P = 21.
```

6.2 FEC Provisioning

The CS604x hardware supports the GFEC coding scheme for both OTU3(V) 40G ports and for up to four OTU2(V) 10G ports. It supports the Ultra-FEC on both OTU3(V) 40G ports or for up to four OTU2(V) 10G ports. This proprietary Ultra FEC operates with overhead rates from 3.4% to 26%.

However, not all of these rates and configurations are supported by the software. The following UFEC percentages are supported for both line-side and client-side OTN protocols: 7, 10, 12, 13, 15, 20, 25, and 26%, plus “0fec” and “nofec” (or ODUk).

FECs should be provisioned immediately after the N40G or N10G provisioning since FEC provisioning over-rides n40g/n10g programming (e.g. setting or changing the number of columns).

6.2.1 FEC, 0FEC and No-FEC

If no FEC is implemented for a particular data stream, that channel is said to be in the “no-FEC” state. If FEC columns are provided, but are not being used (set to “0”) then the FEC is said to be in the “0FEC” (zero-FEC) state. Otherwise, the channel is in the FEC state.

The FEC configuration APIs perform all configuration required to add a FEC to the OTN payload. The config_fec_* APIs assume that no FECs are allocated. They can be successfully run after reset, or after a successful FEC de-allocation. The FEC configureation APIs will support 0FEC in the future, but they do not support it for this release.

FEC reallocation may produce traffic hits if a previous FEC config is not deallocated first.

6.2.2 Supported FEC Types And Rates

The options for provisioning the FECs are:

- No FEC (e.g. no FEC columns present in OTN traffic)
- FEC columns present but zeroed out (e.g. FEC will be added by an external device)
- GFEC
- UFEC at various overhead percentages

Release 4.0 only supports gfec, 7% ufec, and 0fec. No-fec is only supported for one specific mapping. Other ufec rates do not currently work with the new syncdsync code and will not be supported until Release 4.1.

If the user knows the proper syncdsync settings then the FEC APIs will operate correctly to provision higher-percentage UFEC rates.

6.2.3 Related CS604x Datasheet Sections

Section 2.5: Forward Error Correction

6.2.4 Associated APIs

The newer High-Level APIs documented in Sections 7.4 and 7.5 configure the FEC along with the datapath, so the Low-Level functions are generally not required.

6.3 Overhead Ports

There are a number of high-level overhead port (OHPP) APIs that are new with this release. They are listed below. In general, overhead port functions should be much simpler. After initializing the OHPP, and enabling the function, then each activity can be carried out using one or two API calls.

6.3.1 Relevant CS604x Datasheet Sections

Section 2.15: Centralized Overhead and Alarms Processing

6.3.2 Key OHPP APIs

Due to the large number of APIs, they are only listed here. See the API User Guide for parameters and specifics.

6.3.2.1 Forcing APIs

ten_hl_ohpp_force_otu3_ais()
ten_hl_ohpp_force_otu2_ais()
ten_hl_ohpp_force_odu3_maintenance_signal()
ten_hl_ohpp_enable_odu3_maintenance_signal()
ten_hl_ohpp_force_odu2_maintenance_signal()
ten_hl_ohpp_enable_odu2_maintenance_signal()
ten_hl_ohpp_force_opu3_pn11()
ten_hl_ohpp_enable_opu3_pn11()
ten_hl_ohpp_force_opu2_pn11()
ten_hl_ohpp_enable_opu2_pn11()
ten_hl_ohpp_force_aisl()
ten_hl_ohpp_enable_aisl()
ten_hl_ohpp_force_circuit_pp10g_insert()

ten_hl_ohpp_enable_circuit_pp10g_insert()

6.3.2.2 Information Retrieval

ten_hl_ohpp_dump_msignal()
ten_hl_ohpp_dump_osignal()
ten_hl_ohpp_switch_sfc_ram()
ten_hl_ohpp_commit_shadow_sfc_ram()
ten_hl_ohpp_dump_doe_ram()
ten_hl_ohpp_dump_doi_ram()

6.3.2.3 Consequent Action Provisioning

ten_hl_ohpp_dump_consequent_action_events ()
ten_hl_ohpp_dump_shadow_consequent_action_events ()

6.3.2.4 OHPP Init and Control

ten_hl_ohpp_and_shadow_ram_init()
ten_hl_ohpp_init()
ten_hl_ohpp_control_oduk_oci()

6.3.3 Key SFU Low-Level APIs

These are some of the key SFU APIs. They are not currently supported in a high-level API, so the user may want to employ them to drive the SFU. They are all covered in the API User Guide, so they will not be covered here.

ten_ohpp_sfu_set_iaesel
ten_ohpp_sfu_set_mssel
ten_ohpp_sfu_set_msignal (great for SFU debug)
ten_ohpp_sfu_set_m3bcfg
ten_ohpp_sfu_set_nstcfg
ten_ohpp_sfu_set_siggene
ten_ohpp_sfu_get_state
ten_ohpp_set_sfc_ram_bit
ten_ohpp_set_shadow_sfc_ram_bit
ten_ohpp_set_sfu_beicfg
ten_ohpp_sfu_beiovrd

6.3.4 Defect Handling

6.3.4.1 dTIM Defects

The String Extractor has two modes of operation: Default Mode, and dTIM Mode. The operation of these two modes is described below.

Default String Extractor Mode – The accepted string is available for reading via the microcontroller registers. There is no ability to provision an expected value for use in dTIM processing. The actual dTIM status and interrupt bits are not valid but are not automatically disabled. The user must ignore these bits when in this mode.

dTIM String Extractor Mode – The accepted string is not available for reading via the microcontroller registers. The microcontroller registers instead provide access an expected string, which is provisioned by the user with the string value expected to be received. This value will be compared with the actual received string and the dTIM value set accordingly.

If the user has DTIM mode off (default string extractor mode), then the user must ensure that any consequent actions based on dTIM or associated interrupts are disabled.

6.3.4.2 Using the SFU for 10GE and Fiber Channel Defects

The SFU is not set up to look specifically for 10GE (or FC) faults. However the Global Events, bits 0..39 in the list of consequent action triggers, are agnostic about protocol. They include very coarse-grained faults like XLOS, lock-detect-fail, justification generator out-of-range, and phase-detector out of range. So while you can't trigger consequent actions on 10GE (or FC) specific faults, you can trigger on something like a fiber pull.

6.3.5 Overhead Provisioning

6.3.5.1 Receive Expected Values

The provisioning of the expected TTI should be done with the string extractor turned off (SEOFF=1). It is acceptable to enable DTIM mode after the expected TTI, but SEOFF should remain set to 1 until after DTIM mode is set to 1. The string extractor should not be turned on until both the expected value is provisioned and DTIM mode is turned on.

6.3.6 Handling OTN OHPP functions

6.3.6.1 Direct Insertion of Overhead Bytes

The CS604x devices allows for the direct insertion of OTU, ODU, and OPU overhead (OH) bytes. Refer to Section 2.11.5.4 of Revision 0.6 of the CS604x Data Sheet. Note that the following procedure requires the OHPP block to be out of reset and to have clocks applied. These two API functions will do that:

```
ten_mpif_global_clock_disable_common(
      dev_id, CS_ENABLE,
      TEN_MPIF_GLOBAL_CLOCK_DISABLE_COMMON_OHPP_A or
      TEN_MPIF_GLOBAL_CLOCK_DISABLE_COMMON_OHPP_B );

ten_mpif_global_reset_common(
      dev_id, CS_RESET_DEASSERT,
      TEN_MPIF_GLOBAL_RESET_COMMON_OHPPA or
      TEN_MPIF_GLOBAL_RESET_COMMON_OHPPB );

ten_ohpp_sfu_sreset( module_id, CS_RESET_DEASSERT );
```

To enable direct overhead insertion, perform the following steps:

```
ten_ohpp_sfu_set_otnohen( module_id, otnohen );
```

Write the desired OxUk OH value into the DOI RAM:

```
ten_ohpp_set_doi_ram_byte(module_id, string, byte, value);
```

Define the multiframe (MFM) events triggering the insertion:

```
ten_n40g_otnt4x_mfmcfg( module_id, match_unit, mfmcmp,
mfmmsk );
```

```
OR
ten_n10g_otnt_mfmcfg( module_id, slice, match_unit, mfmcmp,
mfmmsk );
```

Enable the direct overhead insertion:

```
ten_ohpp_sfu_set_doaien( module_id, doaien );
```

Configure the direct insertion field for the specific OH byte to either insert the value programmed into the DOI RAM when triggered by the selected MFM event or disable insertion from the DOI RAM for that particular OH byte:

```
ten_n40g_otnt4x_doaie( module_id, byte, insertion );
OR
ten_n10g_otnt_doaie( module_id, slice, byte, insertion );
```

6.3.6.2 Tributary Timeslot Assignment

The default tributary timeslot assignment is shown below, based on the reset values for the N40G_OTNT4X_WRPTSCFG0/1 and N40G_OTNR4X_ODWTSCFG0/1 registers.

Table 11 Tributary Timeslots

Port	Timeslots
1	3, 7, 11, 15
2	2, 6, 10, 14
3	1, 5, 9, 13
4	0, 4, 8, 12

To reverse the timeslot assignments so that port 1 uses timeslots 0, 4, 8, and 12 and the other ports use the next timeslots in increasing order, then write the following values to the device:

```
ten_reg_write( dev_id, N40G_OTNR4X_ODWTSCFG0, 0x1b1b );
ten_reg_write( dev_id, N40G_OTNR4X_ODWTSCFG1, 0x1b1b );
ten_reg_write( dev_id, N40G_OTNT4X_WRPTSCFG0, 0x1b1b );
ten_reg_write( dev_id, N40G_OTNT4X_WRPTSCFG1, 0x1b1b );
```

6.4 Cross-Connect and Elastic Store

The cross-connect (XCON) can be configured using either high-level or low-level APIs. The high-level API allows the user to configure a bidirectional unicast cross-connect path for any pair of channels in CS604x. This should be sufficient for most (if not all) applications.

If low-level APIs are employed, the function used to connect an input channel to an elastic store is ten_xcon_es_select_source. This API effects both receive and transmit blocks connected to the elastic store. In Figure 167, selecting an input channel for the elastic store controls which receiver is sending its data to the transmit block (via ES_SOURCE_SELECT_n). But it also selects which transmit block is feeding threshold information back to the receiver (via ES_RX_THRESHOLD_SELECT_n), as shown in Figure 168. Because of the bidirectional nature of this function it may not be applicable for broadcast applications.

6.4.1.1.1 Relevant CS604x Datasheet Sections

2.10: Cross-Connect and Elastic Stores
Figure 254

6.4.1.1.2 Associated APIs

The API below is similar to the release-4.0 API config_xcon, but it uses a mode field. Only connections on the supported mode list are possible.

```
cs_status ten_hl_xcon_cfg(cs_uint16 dev_id,
                          cs_uint8 slice,
                          cs_uint8 mode)
```

Parameter description:

```
dev_id
      Device ID

slice
      This is one of:
      0 = XCON_ELASTIC_STORE_0
      1 = XCON_ELASTIC_STORE_1
      2 = XCON_ELASTIC_STORE_2
      3 = XCON_ELASTIC_STORE_3
      4 = XCON_ELASTIC_STORE_4
      5 = XCON_ELASTIC_STORE_5
      6 = XCON_ELASTIC_STORE_6
      7 = XCON_ELASTIC_STORE_7Client


mode
      The exact connection type desired. One of:
      TEN_XCON_DISABLE = 0,
      TEN_XCON_10GE_GFP_OTU3 = 1,
      TEN_XCON_OTU3_GFP_10GE = 2,
      TEN_XCON_10GE_RA_FC_OTU3 = 3,
      TEN_XCON_OTU3_RA_FC_10GE = 4,
      TEN_XCON_10GFC_TRANS_OTU3P = 5,
      TEN_XCON_10GE_RA_GFP = 6,
      TEN_XCON_GFP_RA_10GE = 7,
      TEN_XCON_10GE_RA_OTU2E = 8,
      TEN_XCON_OTU2E_RA_10GE = 9,
      TEN_XCON_10GE_OC192_OTU2 = 10,
      TEN_XCON_OTU2_OC192_10GE = 11,
      TEN_XCON_10GE_RA = 12,
      TEN_XCON_10GE_MON_OTU2 = 13,
      TEN_XCON_10GFC_TRANS_OTU2 = 14,
      TEN_XCON_8GFC_MON_OTU2 = 15,
      TEN_XCON_10GE_OTU2E_OTU3P = 16,
      TEN_XCON_10GE_OTU1E_OTU3P = 17,
      TEN_XCON_10GE_OC192_OTU3P = 18,
      TEN_XCON_OTU3P_OC192_10GE = 19,
      TEN_XCON_GRP_T_OTU3P = 20,
      TEN_XCON_10GFC_OTU2FC = 21,
      TEN_XCON_8GFC_OTU2 = 22,
      TEN_XCON_OTU2_8GFC = 23,
      TEN_XCON_TABLE15P4 = 24,
      TEN_XCON_TABLE16P0 = 25,
      TEN_XCON_TABLE16MIX_P0 = 26,
      TEN_XCON_TABLE16MIX_P1 = 27,
      TEN_XCON_TABLE16MIX_P2 = 28,
      TEN_XCON_TABLE19P0 = 29,
      TEN_XCON_TABLE19MIX_P0 = 30,
      TEN_XCON_TABLE19MIX_P1 = 31,
      TEN_XCON_TABLE19MIX_P2 = 32.
```

Low-level APIs:

(See the API User’s Guide XCON section for a complete list.)

7.0 Traffic Functions

This section will briefly describe CS604x high-level traffic-related APIs. In this context, “traffic” refers to the manipulations of the data and control information (as distinct from infrastructure items like reset or clocking). This section will discuss interfaces, allocation of FECs, overhead ports, OTN, SONET, and PP10G functions.

Most of the functions described here are new for the CS604x. The older Release 4.3 and earlier API functions will still provision traffic for the CS600x and the CS604x, limited to the types of mappings that the CS600x supports. The older functions are not described in this document; refer to the Release 4.3 version of *CS600x Software Driver User Guide* for the documentation on those older functions. Newer CS604x mappings require use of the new functions described here.

7.1 No Traffic

There are two main components to sending traffic over the CS604x: block configuration, and traffic configuration. Block configuration must be done first, in a manner consistent with the sequence in the bring-up section. If the block configuration is done, but no traffic API is called to send specific payloads, then the CS604x will not send traffic.

7.2 Interfaces

Release 4.0 adds high-level API support for sfi5, sfi4.2, and XFI. Their use is described below.

Other interface types can be implemented, but the user must use the low-level HSIF APIs. The API calls required would depend upon the user’s specific requirements. Therefore, other interfaces will not be described here.

7.2.1 Locking

7.2.1.1 Lock Checking

There are three functions that can be used to check for SERDES lock. They are:

```
check_mr_filt_lock_40g (module_id)
check_mr_filt_lock (module_id, slice)
check_xfi_filt_lock (module_id, slice)
```

They return 0 if lock achieved, 1 if lock not achieved.

7.2.1.2 Lock Detect Filter Control

By default, CS604x sets the XFI SerDes lock-detect filter control settings too low (e.g. register XFI32X1_SDS_XFI32X1_SDS_COMMON_RXLOCKD0_FILTER for XFI slice 1). This can cause unstable lock status reporting during certain conditions (e.g. fiber-pull).

The XFI high-level interface API (ten_hl_hsif_config_xfi) automatically sets the settings in the Lock Detect Filter Control register to a higher value, which should avoid the issue.

For multi-rate SERDES, there are specific APIs that can be called to change the filter settings. These include:

For TX:

```
ten_hsif_slc1_mr10x5_sds_common_txlockd0_filter(
      module_id, instance, stable_period);

ten_hsif_slc3_mr10x5_sds_common_txlockd0_filter(
      module_id, instance, stable_period);

ten_hsif_slc0_mr10x4_sds_common_txlockd0_filter(
      module_id, instance, stable_period);

ten_hsif_slc2_mr10x4_sds_common_txlockd0_filter(
      module_id, instance, stable_period);
```

For RX

```
ten_hsif_slc1_mr10x5_sds_common_rxlockd0_filter(
      module_id, instance, stable_period);

ten_hsif_slc3_mr10x5_sds_common_rxlockd0_filter(
      module_id, instance, stable_period);


ten_hsif_slc0_mr10x4_sds_common_rxlockd0_filter(
      module_id, instance, stable_period);

ten_hsif_slc0_mr10x4_sds_common_rxlockd0_filter(
      module_id, instance, stable_period);
```

7.2.2 SFI5.1

The SFI5.1 high-level API configures and initializes the SFI5.1 interface, in accordance with a user-specified traffic type and reference clock frequency. The high-level API does not support all traffic types.

The high-level SFI51 API calls a number of other high-level APIs, including:

```
config_clocks
config_gpll_40g
ten_hl_hsif_config_sfi51
ten_hl_hsif_mr_waitfor_vcotune
```

It uses these to set up the SFI51 configuration and allow the SerDes to tune. It also configures clocks and GPLLs to support the interface protocol.

7.2.2.1 Relevant CS604x Datasheet Sections

2.4.3: SFI-5.1 Interface

7.2.2.2 Associated APIs

```
cs_status ten_hl_config_sfi51(cs_uint16 module_id)
```

Parameter description:

```
module_id         Module can be A (0) or B (1)
```

7.2.3 SFI4.2

The SFI4.2 high-level API configures and initializes the SFI4.2 interface, in accordance with a user-specified traffic type and reference clock frequency. The high-level API does not support all traffic types.

The high-level SFI42 APIs calls a number of other high-level APIs, including:

```
config_clocks
config_gpll_40g or config_gpll
ten_hl_hsif_config_sfi42_40g or
ten_hl_hsif_config_sfi42_10g
ten_hl_hsif_mr_waitfor_vcotune
```

It uses these to set up the SFI42 configuration and allow the SerDes to tune. It also configures clocks and GPLLs to support the interface protocol.

SFI4.2 can be run in two different modes: 10G and 40G. For 10G mode, each module can have up to four ports running SFI4.2. 40G makes us of all of the multirate data lanes, so there can be only one port per module.

7.2.3.1 Relevant CS604x Datasheet Sections

2.4.6: SFI-4.2 Interface (10G Capable)
2.4.7: 16-lane SFI-4.2 Interface (40G Capable)

7.2.3.2 Associated APIs

7.2.3.2.1 10G mode

```
cs_status ten_hl_config_sfi42_10g(
      cs_uint16 module_id,
      cs_uint8 slice,
      cs_uint16 line,
      cs_uint32 sysclk_freq,
      cs_uint16 sync,
      cs_uint16 enhanced_deskew)
```

Parameter description:

```
      module_id
            Module can be A (0) or B (1)

      slice
            Integer which indicates the channel (0..3)

      line
            This is a string which indicates the line
            traffic type.

      sysclk_freq
            An integer denoting the system reference clock
            frequency in Hz

      sync
            0 = synchronous
            1 = asynchronous

      enhanced_deskew
            0 = regular deskew mode
            1 = enhanced deskew mode

      Returns: '0' if waitfor_vcotune fails
```

7.2.3.2.2 40G mode

```
cs_status ten_hl_config_sfi42_40g(cs_uint16 module_id,
                                  cs_uint16 divider,
                                  cs_uint16 internal_pilot,
                                  cs_uint16 mr_protect)
```

Parameter description:

```
      module_id
            Module can be A (0) or B (1)

      divider
            One of:
```

```
            TEN_HSIF_CKREFDIV1 = 0,  (div1 to serdes)
            TEN_HSIF_CKREFDIV2 = 1,  (div2 to serdes)
            TEN_HSIF_CKREFDIV4 = 2   (div4 to serdes)

      internal_pilot
            Specified as:
            0 = do not use internal pilot
            1 = use internal pilot.

      mr_protect
            0 = Select SERDES clock from interface
            1 = Select protection clock
```

7.2.4 SFI4.1

The SFI4.1 high-level APIs allow the user to configure a SFI4.1 port without the need for additional calls to HSIF low-level APIs.

This API configures the HSIF SFI41, configures the GPLL and checks for GPLL lock.

As SFI4.1 consumes 16 data pins for an interface, only one SFI4.1 channel can be established per side of the device. Therefore, only 10G of bandwidth can be transmitted over the multirate interface when this is used.

(Note: The API is hardcoded for electrical traffic on Module A and optical on module B. The APIs listed below are used to control this:

```
      Module A:
      ten_hsif_sfi41_cfg_spare_launch_data_txclk(
      module_id, 1)

      On Module B:
      ten_hsif_sfi41_cfg_spare_launch_data_rxclk(
      module_id, 0)
      ten_hsif_sfi41_cfg_spare_launch_data_txclk(
      module_id, 0)
```

)

7.2.4.1 Relevant CS604x Datasheet Sections

2.4.5: SFI-4.1 / XSBI Interface

7.2.4.2 Associated APIs

```
cs_status ten_hl_config_sfi41(
      cs_uint16 module_id,
      cs_uint8 slice,
      cs_uint16 internal_pilot)
```

Parameter description:

```
internal_pilot parameter is specified as:
      0 = do not use internal pilot
      1 = use internal pilot.
```

```
slice Specified as
      TEN_SLICE0 (0),
      TEN_SLICE1 (1),
      TEN_SLICE2 (2),
      TEN_SLICE3 (3),
      TEN_SLICE_ALL (0xFF)
```

7.2.5 XFI

(Note: the user must power down MR functions for any B-side channels that are being used for XFI. Use the ten_hl_hsif_powerdown_mr(module_id, slice, ctl) function.)

The XFI high-level APIs allow the user to configure an XFI port with the need for additional calls to HSIF low-level APIs. The high-level API does not support all traffic types.

The ten_hl_config_xfi API has some additional complexity over SFI4.2 or SFI5.1, because it has support for the protection clocks. The function itself configures the XFI interface and associated GPLLs, downloads microcode if needed, then waits for VCO tuning. It also downloads microcode into the XFI.

The ten_hl_xfi_config_clockmux API sets up clock the clock mux to handle XFI traffic, including clock divider muxing. See XFI Clockmux for more details.

There are two APIs which allow the user to control aspects of the microcode process: ten_hl_hsif_xfi_control_micro_code_verify() controls whether or not the microcode is verified after it is downloaded; ten_hl_hsif_xfi_control_micro_code_running_check() controls whether or not the microcode is checked for “running” after download.

7.2.5.1 Relevant CS604x Datasheet Sections

2.4.2: XFI Interface

7.2.5.2 Associated APIs

7.2.5.2.1 XFI Configuration

```
cs_status ten_hl_config_xfi(
                  cs_uint16 module_id,
                  cs_uint8 slice,
                  cs_uint16 client,
                  cs_uint16 invert_dir,
                  cs_uint16 invert,
                  cs_uint32 sysclk_freq,
                  cs_uint16 prot,
                  cs_uint16 sync,
                  cs_uint16 aux_clk,
                  cs_uint16 waitfor_vcotune)
```

Parameter description:

```
module_id
      Side of the CS604x (TEN_MODULE_A, TEN_MODULE_B)

slice
```

```
      Specified as:
      0x00 = TEN_SLICE0
      0x01 = TEN_SLICE1
      0x02 = TEN_SLICE2
      0x03 = TEN_SLICE3
      0xFF = TEN_SLICE_ALL

client
      Specified as:
      TEN_TRAFFIC_TYPE_NONE = 0
      TEN_TRAFFIC_TYPE_OTU3 = 1
      TEN_TRAFFIC_TYPE_OTU3E = 2
      TEN_TRAFFIC_TYPE_OTU3P = 3
      TEN_TRAFFIC_TYPE_ODTU23 = 4
      TEN_TRAFFIC_TYPE_OTU2 = 5
      TEN_TRAFFIC_TYPE_OTU2E = 6
      TEN_TRAFFIC_TYPE_OTU1F = 7
      TEN_TRAFFIC_TYPE_OC192 = 8
      TEN_TRAFFIC_TYPE_10GE_WAN = 9
      TEN_TRAFFIC_TYPE_10GE_6_2 = 10
      TEN_TRAFFIC_TYPE_10GE_7_1 = 11
      TEN_TRAFFIC_TYPE_10GE_7_3 = 12
      TEN_TRAFFIC_TYPE_10GFC = 13
      TEN_TRAFFIC_TYPE_8GFC = 14
      TEN_TRAFFIC_TYPE_4GFC = 15
      TEN_TRAFFIC_TYPE_2GFC = 16
      TEN_TRAFFIC_TYPE_1GFC = 17
      TEN_TRAFFIC_TYPE_5GB_IB = 18

invert_dir
      Specified as:
      0 = CS_TX
      1 = CS_RX
      2 = CS_TX_AND_RX

invert
      Specified as:
      0 = do invert the bit polarity of all TX/RX data bits
      1 = invert the bit polarity of all TX/RX data bits


sysclk_freq
      Specified as:
      Sys clock frequency in Hertz, ie 400000000

prot
      Specified as:
      0 = do not set up the receive protection clock
      1 = set up the receive protection clock

Sync
      Specified as:
      0 = async
      1 = sync
```

```
aux_clk
      Specified as:
      0 = do not use auxiliary clock
      1 = use auxiliary clock

waitfor_vcotune
      Specified as:
      0 = do not wait for vcotune (not recommended)
      1 = wait for vcotune (recommended)
```

7.2.5.2.2 Microcode Operation Checking

ten_hl_hsif_xfi_control_micro_code_verify

The XFI highlevel configuration routine automatically downloads and checks the microcode. However, there are situations where the user HW configuration can generate false microcode verification errors. The ten_hl_hsif_xfi_control_micro_code_verify() API allows the user to disable microcode verification after microcode download. This should only be done when the user specifically wants to avoid known false errors.

```
cs_status ten_hl_hsif_xfi_control_micro_code_verify(
      cs_ctl_t ctl)
```

Parameter description:

```
The ctl parameter controls checking option:
      0 = CS_DISABLE (do not verify microcode)
      1 = CS_ENABLE (perform microcode verify.
```

ten_hl_hsif_xfi_control_micro_code_running_check

The XFI highlevel configuration routine automatically runs the microcode. However, there are situations where the user HW configuration can generate false errors when verifying that the microcode is running. The ten_hl_hsif_xfi_control_micro_code_running_check() API allows the user to disable microcode-running verification after microcode download. This should only be done when the user specifically wants to avoid known false errors.

```
cs_status ten_hl_hsif_xfi_control_micro_code_running_check(
      cs_ctl_t ctl)
```

Parameter description:

```
The [ctl] parameter controls checking option:
      0 = CS_DISABLE (do not check microcode running)
      1 = CS_ENABLE (perform check for microcode running)
```

7.2.6 XAUI

The XAUI high-level APIs allow the user to configure a XAUI port without the need for additional calls to HSIF low-level APIs.

This API configures the HSIF XAUI SERDES and GPLLX1. There is no need to call a separate API to set the XAUI-related GPLL.

7.2.6.1 Relevant CS604x Datasheet Sections

2.2.2: XAUI-Based Ethernet Client
2.2.4: XAUI-Based 10GFC Client
2.4.4: XAUI Interface

7.2.6.2 Associated APIs

```
cs_status ten_hl_config_xaui(
      cs_uint16 module_id,
      cs_uint8 slice,
      cs_uint16 internal_pilot,
      cs_uint16 mr_protect)
```

Parameter description:

```
module_id
      Side of the CS604x (TEN_MODULE_A, TEN_MODULE_B)

slice Specified as
      TEN_SLICE0 (0),
      TEN_SLICE1 (1),
      TEN_SLICE2 (2),
      TEN_SLICE3 (3),
      TEN_SLICE_ALL (0xFF).

internal_pilot parameter is specified as:
      0 = do not use internal pilot
      1 = use internal pilot.

mr_protect parameter is specified as:
      0 = do not use mr protection
      1 = use mr protection.
```

```
cs_status ten_hl_hsif_config_xaui(cs_uint16 module_id,
                                  cs_uint8 slice,
                                  cs_uint16 internal_pilot,
                                  cs_uint16 mr_protect)
```

7.3 Circuit Protocols

7.3.1 Common Circuit Protocol Modes

The high-level traffic functions operate in specific modes. The mode selected determines which traffic-related registers get set, the FEC percentage that can be applied, the handling of defects (for example, whether or not they are forwarded), and the actions of the SFU.

For example, for OxU to OxU applications, two possible configuration modes are Termination Mode/Transparent Regenerator and Termination Mode/Section Terminator. For the Section Terminator case, for TX traffic, the SFU inserts SM-BEI based on SM-BIP errors detected on same side RX. For the Regenerator case, it does not.

This section summarizes the effects of the modes for different traffic types. These effects must be taken into account when the user is applying a high-level traffic API.

7.3.1.1 OxU2 to OxU2 Applications

For OxU2-to-OxU2 applications, the OxU mode for the A-side, the OxU mode for the B-side, and the termination mode may each be set independently.

7.3.1.1.1 OxU Mode

1. OTU2 (G.709 standard FEC, can be standard rate ODU2 or non-standard rate such as ODU2e/ODU1e)

   RX

   - Set OTNR_OFCFG1.NPAR to 16 (0x10)
   - Set OTNR_OFCFG0.NCOLS = 254 (0xfe)
   - OTNR_OACFG0.FECMD = 0
   - OTNR_OACFG0.POSTDSCR = 0, PREDSCR = 1
   - OTNR_OACFG0.ALEN = 1
   - OTNR_OACFG0.FECEN = 1

   TX

   - Set OTNT_WRPCFG2.NPAR to 16 (0x10)
   - Set OTNT_WRPCFG0.NCOLS = 254 (0xfe)
   - OTNT_OCFG.FECMD = 0
   - OTNT_OCFG.POSTSCR = 1, PRESCR = 0
   - OTNR_OACFG0.FECEN = 1

2. OTU2V (strong FEC – UFEC or other externally-applied FEC, can be standard 7% or non-standard rate with higher/lower FEC redundancy)

   RX

   - Set OTNR_OFCFG1.NPAR to desired # of FEC columns
   - Set OTNR_OFCFG0.NCOLS = 238 + NPAR
   - OTNR_OACFG0.FECMD = 1
   - OTNR_OACFG0.POSTDSCR = 1, PREDSCR = 0
   - OTNR_OACFG0.ALEN = 1
   - Internal UFEC Decoder : OTNR_OACFG0.FECEN = 1
   - External FEC Decoder : OTNR_OACFG0.FECEN = 0

TX

- Set OTNT_WRPCFG2.NPAR to desired # of FEC columns
- Set OTNT_WRPCFG0.NCOLS = 238 + NPAR
- OTNT_OCFG.FECMD = 1
- OTNT_OCFG.POSTSCR = 0, PRESCR = 1
- Internal UFEC Encoder : OTNR_OACFG0.FECEN = 1
- External FEC Encoder : OTNR_OACFG0.FECEN = 0

3. ODU2 (or ODU2e/ODU1e) – technically an OTU2V with no FEC columns

   RX

   - Set OTNR_OFCFG1.NPAR to 0
   - Set OTNR_OFCFG0.NCOLS = 238 (0xee)
   - OTNR_OACFG0.POSTDSCR = 0, PREDSCR = 1
   - OTNR_OACFG0.ALEN = 1
   - OTNR_OACFG0.FECEN = 0

   TX

   - Set OTNT_WRPCFG2.NPAR to 0
   - Set OTNT_WRPCFG0.NCOLS = 238 (0xee)
   - OTNT_OCFG.POSTSCR = 1, PRESCR = 0
   - OTNR_OACFG0.FECEN = 0

4. ODU2 demultiplexed/multiplexed from/to ODTU23/ODTU23e

   RX

   - Set OTNR_OFCFG1.NPAR to 0
   - Set OTNR_OFCFG0.NCOLS = 238 (0xee)
   - OTNR_OACFG0.POSTDSCR = 0, PREDSCR = 0
   - OTNR_OACFG0.ALEN = 1
   - OTNR_OACFG0.FECEN = 0
   - OTNR_OACFG2.MFCMPES = 1
   - OTNR_OACFG2.MFCMPED = 1

   TX

   - Set OTNT_WRPCFG2.NPAR to 0
   - Set OTNT_WRPCFG0.NCOLS = 238 (0xee)
   - OTNT_OCFG.POSTSCR = 0, PRESCR = 0
   - OTNR_OACFG0.FECEN = 0

7.3.1.1.2 Termination Mode

1. Transparent Regenerator

RX Side:

- GBLR_DPCFG.XSEL = 1, OBY = 0, SBYFRM = 1 (Framed OTN through the XCON)
- OTNR_ODWCFG.FSOMD = 1
- LOS , LOF, and OTUk-AIS reported to OHPP
- TCM and PM layers may optionally be used for non-intrusive monitoring
- SDH Processor and PP10G may optionally be used for non-intrusive monitoring

TX Side:

- GBLT.CFGTX0.OBY = 0, SBY = 1, KPGBY = 1
- OTNT_CFG0.FASIE = 0, OTNT_CFG0.MFASINS = 0x0
- OTNT_CFG0.PTIE = 0, OTNT_WRPCFG3.WRMD = 0x3
- SFU inserts ODUk-AIS as consequent action if LOS/LOF/OTUk-AIS detected on opposite RX (this must include setting OTNT_CFG0.FASIE=1, OTNT_CFG0.MFASINS=0x3)

2. Section Terminator

RX Side:

- GBLR_DPCFG.XSEL = 1, OBY = 0, SBYFRM = 1 (Framed OTN through the XCON)
- OTNR_ODWCFG.FSOMD = 1
- OOHR_CFG5.MFALGNE = 1
- OOHR_CFG5.BIPMSK = 0xFF
- LOS , LOF, and OTUk-AIS reported to OHPP
- SM-dTIM, SM-dDEG, SM-dIAE, SM-dBDI, SM-dBIAE defects reported to OHPP
- TCM and PM layers may optionally be used for non-intrusive monitoring
- SDH Processor and PP10G may optionally be used for non-intrusive monitoring

TX Side:

- GBLT.CFGTX0.OBY = 0, SBY = 1, KPGBY = 1
- OTNT_CFG0.FASIE = 1, MFASINS = 0x3, MFALGNE = 1
- OTNT_CFG4.BIPMSK = 0x80 (Generate Section BIP)
- OTNT_CFG0.PTIE = 0 , OTNT_WRPCFG3.WRMD = 0x3
- SFU inserts ODUk-AIS as consequent action if LOS/LOF/OTUk-AIS or optionally SM-dTIM detected on opposite RX

- SFU inserts SM-IAE as consequent action if LOS/LOF/OTUk-AIS detected on opposite RX, else set SM-IAE = 0
- SFU inserts SM-BEI based on SM-BIP errors detected on same side RX
- SFU inserts SM-BIAE as consequent action when SM-dIAE detected on same side RX
- SFU inserts SM-BDI as consequent action when LOS/LOF/OTUk-AIS or optionally SM-dTIM detected on same side RX, else set SM-BDI=0
- SM-TTI, GCC0, and RES overhead may optionally be inserted.
- ODU-layer GCC1/GCC2, FTFL, APS/PCC, EXP, and RES overhead may optionally be inserted.

3. TCM Layer "I" Terminator

RX TCM-Terminating Side:

- GBLR_DPCFG.XSEL = 1, OBY = 0, SBYFRM = 1 (Framed OTN through the XCON)
- OTNR_ODWCFG.FSOMD = 1
- OOHR_CFG5.MFALGNE = 1
- OOHR_CFG5.BIPMSK = 0xFF
- LOS , LOF, and OTUk-AIS reported to OHPP
- SM-dTIM, SM-dDEG, SM-dIAE, SM-dBDI defects reported to OHPP
- TCMi-dLTC, TCMi-dODUkAIS, TCMi-dOCI, TCMi-dLCK, TCMi-dTIM, TCMi-dDEG, TCMi-dIAE, TCMi-dBDI, TCMi-dBIAE defects reported to OHPP
- PM and other TCM layers may optionally be used for non-intrusive monitoring
- SDH Processor and PP10G may optionally be used for non-intrusive monitoring

TX TCM-Terminating Side:

- GBLT.CFGTX0.OBY = 0, SBY = 1, KPGBY = 1
- OTNT_CFG0.FASIE = 1, MFASINS = 0x3, MFALGNE = 1
- OTNT_CFG4.BIPMSK[7] = 1 (Generate Section BIP)
- OTNT_CFG4.BIPMSK[i] = 1 (Generate TCM #i BIP)
- OTNT_CFG4.BIPMSK[n] = 0 (for n=0..6 and n != i)
- OTNT_CFG0.PTIE = 0, OTNT_WRPCFG3.WRMD = 0x3
- SFU inserts ODUk-AIS as consequent action if LOS/LOF/OTUk-AIS or optionally SM-dTIM detected on opposite RX

- SFU inserts SM-IAE as consequent action if LOS/LOF/OTUk-AIS detected on opposite RX, else set SM-IAE = 0
- SFU inserts SM-BEI based on SM-BIP errors detected on same side RX
- SFU inserts SM-BIAE as consequent action when SM-dIAE detected on same side RX
- SFU inserts SM-BDI as consequent action when LOS/LOF/OTUk-AIS or optionally SM-dTIM detected on same side RX, else set SM-BDI=0
- SM-TTI, GCC0, and RES overhead may optionally be inserted.
- ODU-layer GCC1/GCC2, FTFL, APS/PCC, EXP, and RES overhead may optionally be inserted.
- SFU inserts TCMi-BEI based on TCMi-BIP errors detected on TCM-Terminating side RX
- SFU inserts TCMi-BIAE as consequent action when TCMi-dIAE detected on TCM-Terminating side RX
- SFU inserts TCMi-BDI as consequent action when LOS/LOF/OTUk-AIS or optionally SM-dTIM or TCMi-ODUkAIS/dOCI/dLCK/dLTC or optionally TCMi-dTIM detected on TCM-Terminating side RX, else set TCMi-BDI=0
- SFU inserts TCMi-aIAE (TCMi -STAT = 010) as consequent action if LOS/LOF/OTUk-AIS on opposite RX, else set TCMi- STAT = 001.
- TCMi-TTI inserted

RX non-TCM-Terminating Side:

- Same as #2 (Section Terminating above)

TX non-TCM-Terminating Side:

- GBLT.CFGTX0.OBY = 0, SBY = 1, KPGBY = 1
- OTNT_CFG0.FASIE = 1, MFASINS = 0x3, MFALGNE = 1
- OTNT_CFG4.BIPMSK = 0x80 (Generate Section BIP)
- OTNT_CFG0.PTIE = 0, OTNT_WRPCFG3.WRMD = 0x3
- SFU inserts ODUk-AIS as consequent action if LOS/LOF/OTUk-AIS or optionally SM-dTIM or TCMi-ODUkAIS/dOCI/dLCK/dLTC or optionally TCMi-dTIM detected on opposite RX
- SFU inserts SM-IAE as consequent action if LOS/LOF/OTUk-AIS detected on opposite RX, else set SM-IAE = 0
- SFU inserts SM-BEI based on SM-BIP errors detected on same side RX

- SFU inserts SM-BIAE as consequent action when SM-dIAE detected on same side RX
- SFU inserts SM-BDI as consequent action when LOS/LOF/OTUk-AIS or optionally SM-dTIM detected on same side RX, else set SM-BDI=0
- SM-TTI, GCC0, and RES overhead may optionally be inserted.
- ODU-layer GCC1/GCC2, FTFL, APS/PCC, EXP, and RES overhead may optionally be inserted.
- TCMi OH inserted as 0x00

7.3.1.2 Mapping OC-192 Client into OTN

When an OC192 is mapped into an OTN wrapper, there are several different ways that overhead, defects, and consequent actions can behave. Not all combinations of these are supported by the software, or by CS604x.

The modes that are supported are as follows:

1. Transparent Mode (could also be called CBR10 mode)

RX Client Side:

- XSEL = 2, SBYPLD = 1 (Bypass SONET/SDH processing)
- LOS reported to OHPP
- SDH Processor may optionally be used for non-intrusive monitoring

TX Line Side (Mapping):

- SBY = 1 (Bypass SONET/SDH processing)
- SFU inserts CBRGENAIS as consequent action if LOS detected on client RX

RX Line Side (Demapping):

- XSEL = 2, SBYPLD = 1 (Bypass SONET/SDH processing)
- OTN-layer defects reported to OHPP
- SDH Processor may optionally be used for non-intrusive monitoring, otherwise set SDHE = 0 to save power

TX Client Side:

- SBY = 0 (Must not bypass SDH Processor for consequent-action insertion)
- SOHT_OHBEN0 = 0x00, SOHT_OHBEN1 = 0x00, SOHT_OHBEN2 = 0x00 (disable all SONET/SDH insertion)

- SFU inserts STM-AIS as consequent action if OTN defect detected on line side

2. Fully Transparent Regenerator

   RX Client Side:

   - XSEL = 1, SBYFRM = 0 (Framed SONET/SDH through the XCON)
   - SDFR_SDFCFG.SCREN = 1
   - LOS , LOF, and STM-AIS reported to OHPP
   - SDH Processor and PP10G may optionally be used for non-intrusive monitoring

   TX Line Side (Mapping):

   - SBY = 0 (Do not bypass SONET/SDH processing)
   - SOHT_OHBEN0 = 0x08, SOHT_OHBEN1 = 0x00, SOHT_OHBEN2 = 0x00 (disable all SONET/SDH insertion but turn on scrambling)
   - SFU inserts MS-AIS (AIS-L) as consequent action if LOS/LOF/STM-AIS detected on client RX (this must include setting SOHT_OHBEN0.B1IE=1)

   RX Line Side (Demapping):

   - XSEL = 1, SBYFRM = 0 (Framed SONET/SDH to the XCON)
   - SDFR_SDFCFG.SCREN = 1
   - LOF and STM-AIS reported to OHPP
   - OTN-layer defects reported to OHPP
   - SDH Processor and PP10G may optionally be used for non-intrusive monitoring

   TX Client Side:

   - SBY = 0 (Do not bypass SDH Processor)
   - SOHT_OHBEN0 = 0x08, SOHT_OHBEN1 = 0x00, SOHT_OHBEN2 = 0x00 (disable all SONET/SDH insertion but turn on scrambling)
   - SFU inserts MS-AIS (AIS-L) as consequent action if LOF/STM-AIS or OTN defect detected on line RX (this must include setting SOHT_OHBEN0.B1IE=1)

3. RS-layer Regenerator

   RX Client Side:

   - XSEL = 1, SBYFRM = 0 (Framed SONET/SDH through the XCON)
   - SDFR_SDFCFG.SCREN = 1
   - LOS , LOF, STM-AIS, RS-TIM, AIS-L defects reported to OHPP

- Set SOHR_S1CFG.B1TRANS=1
- SDH Processor and PP10G may optionally be used for non-intrusive monitoring

TX Line Side (Mapping):

- SBY = 0 (Do not bypass SONET/SDH processing)
- In SOHT, set SCREN=1, A1A2RSH=1, B1IE =1
- In SOHT, enable RSOH bytes that are to be sourced (others will be transparently passed through)
- If B1 error transparency is desired, set B1TRANS=1 in SOHT_OHBEN1
- SFU inserts MS-AIS (AIS-L) as consequent action if LOS/LOF/STM-AIS/MS-AIS or optionally RS-TIM detected on client RX

RX Line Side (Demapping):

- XSEL = 1, SBYFRM = 0 (Framed SONET/SDH through the XCON)
- SDFR_SDFCFG.SCREN = 1
- LOF, STM-AIS, RS-TIM, AIS-L defects reported to OHPP
- OTN-layer defects reported to OHPP
- Set SOHR_S1CFG.B1TRANS=1
- SDH Processor and PP10G may optionally be used for non-intrusive monitoring

TX Client Side:

- SBY = 0 (Do not bypass SDH Processor)
- In SOHT, set SCREN=1, A1A2RSH=1, B1IE =1
- In SOHT, enable RSOH bytes that are to be sourced (others will be transparently passed through)
- If B1 error transparency is desired, set B1TRANS=1 in SOHT_OHBEN1
- SFU inserts MS-AIS (AIS-L) as consequent action if LOF/STM-AIS/MS-AIS or optionally RS-TIM or OTN defect detected on line RX

4. MS-layer Regenerator

RX Client Side:

- XSEL = 1, SBYFRM = 0 (Framed SONET/SDH through the XCON)
- SDFR_SDFCFG.SCREN = 1
- LOS , LOF, STM-AIS, RS-TIM, AIS-L, dEXC, dLOP, AU-AIS defects reported to OHPP
- Set SOHR_S1CFG.B1TRANS=1

- Set SOHR_S1CFG.B2TRANS=1
- SDH Processor and PP10G may optionally be used for non-intrusive monitoring

TX Line Side (Mapping):

- SBY = 0 (Do not bypass SONET/SDH processing)
- In SOHT, set SCREN=1, A1A2RSH=1, B1IE =1, B2IE=1
- In SOHT, enable RSOH/MSOH bytes that are to be sourced (others will be transparently passed through)
- If B1 error transparency is desired, set B1TRANS=1 in SOHT_OHBEN1
- If B2 error transparency is desired, set B2TRANS=1 in SOHT_OHBEN1
- SFU inserts MS-RDI (RDI-L) as consequent action if LOF/MS-AIS detected on line RX
- SFU inserts MS-REI (REI-L) based on number of B2 errors detected on line RX
- SFU inserts MS-AIS (AIS-L) as consequent action if LOS/LOF/STM-AIS/MS-AIS or optionally RS-TIM/dEXC detected on client RX
- Optionally insert AU-AIS as consequent action if dLOP/AU-AIS detected on client RX

RX Line Side (Demapping):

- XSEL = 1, SBYFRM = 0 (Framed SONET/SDH through the XCON)
- SDFR_SDFCFG.SCREN = 1
- LOF, STM-AIS, RS-TIM, AIS-L, dEXC, dLOP, AU-AIS defects reported to OHPP
- OTN-layer defects reported to OHPP
- Set SOHR_S1CFG.B1TRANS=1
- Set SOHR_S1CFG.B2TRANS=1
- SDH Processor and PP10G may optionally be used for non-intrusive monitoring

TX Client Side:

- SBY = 0 (Do not bypass SDH Processor)
- In SOHT, set SCREN=1, A1A2RSH=1, B1IE =1, B2IE=1
- In SOHT, enable RSOH/MSOH bytes that are to be sourced (others will be transparently passed through)
- If B1 error transparency is desired, set B1TRANS=1 in SOHT_OHBEN1

- If B2 error transparency is desired, set B2TRANS=1 in SOHT_OHBEN1
- SFU inserts MS-RDI (RDI-L) as consequent action if LOS/LOF/MS-AIS detected on client RX
- SFU inserts MS-REI (REI-L) based on number of B2 errors detected on client RX
- SFU inserts MS-AIS (AIS-L) as consequent action if LOF/STM-AIS/MS-AIS or optionally RS-TIM/dEXC or OTN defect detected on line RX
- Optionally insert AU-AIS as consequent action if dLOP/AU-AIS detected on line RX

7.3.2 Common Circuit Protocol Received Defect Handling

7.3.2.1 OTN Defect Handling

The effects of OTN defects transmitted to the CS604x differs depending on the mode that the CS604x OTN is in. The table below summarizes the handling of defects for the different OTN modes. If the defect is listed as “Transmitted”, then it is passed through the CS604x in that case. If it is listed as “Not Transmitted”, then it is not passed through.

(Note: These effects have been verified by test equipment.)

Transmitted Error	Transparent Regenerator	Section Terminator	TCM Layer “I” Terminator
Frame	Transmitted	Not Transmitted	
MFAS	Transmitted	Not Transmitted	
OTU BIP8	Transmitted	Not Transmitted	
OTU BEI	Transmitted	Transmitted	
ODU BIP8	Transmitted	Transmitted	
ODU BEI	Transmitted	Transmitted	
FEC Block	Transmitted	Transmitted	

Similarly, the OTN alarms that are passed through are as follows:

Transmitted Error	Transparent Regenerator	Section Terminator	TCM Layer “I” Terminator
LOF	Transmitted	Not Transmitted	
OOF	Transmitted	Transmitted	
LOM	Transmitted	Not Transmitted	
OOM	Transmitted	Not Transmitted	
OTU AIS	Transmitted	Transmitted	
OTU IAE	Transmitted	Transmitted	
OTU BDI	Transmitted	Transmitted	
ODU AIS	Transmitted	Transmitted	
ODU OCI	Transmitted	Transmitted	
ODU LCK	Transmitted	Transmitted	

ODU BDI	Transmitted	Transmitted	

The following table summarizes the impact of received defects. The impact of received defects was measured with commercial test equipment and is subject to the equipment's generation and detection capabilities.

	1	**2**
	Transparent	**Repeater**
Transmit OH in both directions:		
FAS (A1/A2)	Passthru	Refreshed on error
BIP-8	Passthru	Passthru if no consequent actions are defined; regenerated if consequent actions are defined
Impact of client defects on the line:		
LOS (with consequent action)	Framed random data	AIS
LOS (without consequent action)	LOF	OOF; goes in and out of frame with random data
LOF (with consequent action)	Framed random data	AIS
LOF (without consequent action)	LOF	Framing errors are corrected
BIP-8 error (with consequent action)	BIP-8 error	Nothing, BIP-8 is regenerated
BIP-8 error (without consequent action)	BIP-8 error	BIP-8 error
AIS	AIS	AIS
Bit error	Passthru; could result in BIP-8 errors	Passthru; could result in BIP-8 errors
Impact of line defects on the client:		
LOS (with consequent action)	LOF	AIS
LOS (without consequent action)	LOF	Framed random data
LOF (with consequent action)	LOF	Nothing
LOF (without consequent action)	LOF	Framing errors are corrected
BIP-8 error (with consequent action)	BIP-8 error	Nothing, BIP-8 is regenerated
BIP-8 error (without consequent action)	BIP-8 error	BIP-8 error
AIS	AIS	AIS
Bit error	Passthru; could result in BIP-8 errors	Passthru; could result in BIP-8 errors

7.3.2.2 SONET over OTN Defect Handling

The following table summarizes the provisioning and impact of received defects. The impact of received defects was measured with commercial test equipment and is subject to the equipment's generation and detection capabilities.

	1	2		
	Transparent (CBR10)	**Fully Transparent Regenerator**	**RS-layer Regenerator**	**MS-layer Regenerator**
Transmit OH in both directions:				
FAS (A1/A2)	Passthru	Passthru (unless AIS-L insertion)	Refreshed	Refreshed
B1	Passthru	Passthru (unless AIS-L insertion)	Regenerated	Regenerated
B2	Passthru	Passthru (unless AIS-L insertion)	Passthru	Regenerated
J0 and other OH bytes (see note 1)	Passthru	Passthru	Regen for select RSOH bytes, passthru for others (user-specific)	Regen for select RSOH and MSOH bytes, passthru for others (user-specific)
Impact of client defects on the line:				
LOS (with consequent action)	Unframed random data (PN-11)	AIS-L	AIS-L (MS-AIS)	AIS-L (MS-AIS)
LOS (without consequent action)	LOF	LOF	No backup clock: OOF; goes in and out of frame with random data Backup clock: good framing, payload and any inserted OH	No backup clock: OOF; goes in and out of frame with random data Backup clock: good framing, payload and any inserted OH
LOF (with consequent action)	Framed random data	AIS-L	AIS-L (MS-AIS)	AIS-L (MS-AIS)
LOF (without consequent action)	LOF	LOF	Framing errors are corrected	Framing errors are corrected (and any OH inserted will be good)
B1 error (without consequent action)	B1 error	B1 error	B1 regen	B1 regen
B2 error	B2 error	B2 error	B2 error	B2 regen
AIS-L (MS-AIS)	AIS-L	AIS-L	AIS-L	AIS-L
Bit error	Passthru; could result in B1/B2 errors	Passthru; could result in B1/B2 errors	Passthru; could result in B1/B2 errors	Passthru; could result in B1/B2 errors
Impact of line defects on the client:				

LOS (with consequent action)	Unframed PN-11	AIS-L	AIS-L (MS-AIS)	AIS-L (MS-AIS)
LOS (without consequent action)	LOF	LOF	Backup clock used: Framed random data; No backup clock: will go in and out of frame	Backup clock used: Framed random data; No backup clock: will go in and out of frame
SONET LOF (with consequent action)	LOF	AIS-L	AIS-L	AIS-L
SONET LOF (without consequent action)	LOF	LOF	Framing errors are corrected	Framing errors are corrected
B1 error (without consequent action)	B1 error	B1 error	B1 regen	B1 regen
B2 error	B2 error	B2 error	B2 error	B2 regen
AIS-L (MS-AIS)	AIS-L	AIS-L	AIS-L	AIS-L
Bit error	Passthru; could result in B1/B2 errors	Passthru; could result in B1/B2 errors	Passthru; could result in B1/B2 errors	Passthru; could result in B1/B2 errors

Note 1: OH bytes can be optionally inserted to override the received values. Refer to the “Additional Operations Following Provisioning” section below.

7.4 40G Transponder Traffic Types

7.4.1 40G-rate Traffic Configuration

The function ten_hl_config_40g_monolithic is used to map any 40G-to-40G transponder (no aggregation) traffic configuration.

This API is used for all supported traffic types: OTN, OC768, 40GE rate (Note that CS600x does not process 40GE payloads but the CS604x does), 40G CBR. The required traffic types are set with parameters, instead of calling a specific function for a specific connection type.

By default, all typical scrambling and other functions are set automatically in accordance with the specified traffic types.

The 40G to 40G modes that are currently supported are: OTU3 to OTU3, OTU3 to OC768, OTU3E to 40GELAN CBR; other rates are possible.

7.4.1.1 Relevant CS604x Datasheet Sections

2.6: Receive 40G Circuit Processor
2.14: Transmit 40G Circuit Processor

7.4.1.2 Function Call and Parameters

```
cs_status ten_hl_config_40g_monolithic(
      cs_uint16 mod_line,
      cs_uint16 mod_client,
      cs_uint16 line,
      cs_uint16 client,
      cs_uint16 dyn_repro,
      cs_uint16 line_fec,
      cs_uint16 client_fec,
      cs_uint16 sync_mode,
      cs_uint16 term_otu,
      cs_uint16 line_tcm_bits,
      cs_uint16 client_tcm_bits,
      cs_uint16 term_sonet)


Parameter descriptions:

      mod_line and mod_client
            0 for side A and 1 for side B

      line and client are:
            0x01 = TEN_TRAFFIC_TYPE_OTU3
            0x02 = TEN_TRAFFIC_TYPE_OTU3E
            0x15 = TEN_TRAFFIC_TYPE_40GELAN
            0x16 = TEN_TRAFFIC_TYPE_OC768
            0x17 = TEN_TRAFFIC_TYPE_OTU3E3

      dyn_repro is:
            0 for disable and 1 for enable although this
            parameter is not currently used

      line_fec and client-fec are as follows (although the
      FEC provisioning is currently done outside of this
      API):
            0x00 = TEN_FEC_MODE_OTUkV (7%)
            0x01 = TEN_FEC_MODE_OTUkV_262 (10%)
            0x02 = TEN_FEC_MODE_OTUkV_273 (15%)
            0x03 = TEN_FEC_MODE_OTUkV_285 (20%)
            0x04 = TEN_FEC_MODE_OTUkV_297 (25%)
            0x05 = TEN_FEC_MODE_OTUkV_5_4 (25%)
            0x06 = TEN_FEC_MODE_OTUkV_301 (26%)
            0x07 = TEN_FEC_MODE_OTUkV_SDH_A (7%)
            0x08 = TEN_FEC_MODE_OTUkV_SDH_B (7 %)
            0x09 = TEN_FEC_MODE_OTUkV_4080_3929 (3.4%)
            0x10 = TEN_FEC_MODE_OTUkV_68_65 (4.2%)
            0x11 = TEN_FEC_MODE_OTUkV_267 (12%)
            0x12 = TEN_FEC_MODE_OTUkV_270 (13%)
            0x13 = TEN_FEC_MODE_GFEC (7%)
            0x14 = TEN_FEC_MODE_ZEROFEC
            0x15 = TEN_FEC_MODE_NOFEC

      sync_mode:
            0 for Async and 1 for Sync

      term_otu is the OTU3 to OTU3 termination mode:
            0x00 = TEN_OTU_TERM_TRANSPARENT
            0x01 = TEN_OTU_TERM_SECTION
            0x02 = TEN_OTU_TERM_FULL
```

```
line_tcm_bits and client_tcm_bits define which tandem
pairs are terminated. Bits 1 through 6 being set will
enable the termination of that pair.
      bit 1 TCM1
      bit 2 TCM2
      bit 3 TCM3
      bit 4 TCM4
      bit 5 TCM5
      bit 6 TCM6

term_sonet:
      Defines the way the OC768 will be sinked and
      sourced (despite the enum names shown below,
      this is an OC768 function):
      0x01 = TEN_OC192_TERM_TRANSPARENT_CBR10
      0x02 = TEN_OC192_TERM_TRANSPARENT_REGENERATOR
      0x03 = TEN_OC192_TERM_RS_LAYER_REGENERATOR
      0x04 = TEN_OC192_TERM_MS_LAYER_REGENERATOR
```

7.4.2 OC-768 to OTU3

There is an issue with OC-768 to OTU3 mappings with the CS600x devices. The failure is due to excessive skew between the four 10G data lanes coming from the SADECO in the Cross Connect to the N40G block. This problem can be detected in the chip by observing that the TXDSKERR interrupt in the N40G_GBL4X_GBLINT register is persistently active after the SYNC bit in the XCON_SADECO_SDSTAT register goes active when data is present. The software fix for this is to resynchronize the generated frame alignment in the SADECO by toggling the MANRSY bit in the XCON_SADECO_SADCFG3 register.

7.4.2.1 CS600x Software Fix

The software fix for this has been implemented in the ten_sadeco_n40g_deskew_error_handler() routine. This software fix is only required for the CS600x devices – not for the CS604x devices. This routine will toggle the MANRSY bit in the XCON_SADECO_SADCFG3 if it detects that the SYNC bit in the XCON_SADECO_SDSTAT register is active and that the TXDSKERR interrupt in the N40G_GBL4X_GBLINT register is active after it has been cleared for one SONET frame. The ten_sadeco_n40g_deskew_error_handler() routine is to be called when the SYNCI interrupt in the XCON_SADECO_SADINT register goes active.

The ten_sadeco_n40g_deskew_error_handler() interrupt handler can be registered and the SYNCI interrupt in the XCON_SADECO_SADINT register can be enabled by calling the ten_sadeco_n40g_deskew_error_handler_enable() API.   The ten_irq_isr() routine can be used to walk the interrupt tree whenever an interrupt occurs.

7.4.2.2 CS604x OC768 → OTU3(V) Operation

There are two mechanisms for aligning the four 10G streams in the CS604x devices. The first uses the SADECO and is the default method when the SW detects a CS604x device. This method uses a hardware fix that was implemented in the CS604x devices. If a CS600x device is detected then the hardware fix is not available and the software workaround described in Section 7.4.2.1 must be used.

The second method uses the PP40G block and is designed for minimum latency across the cross-connect. This method is not available for use as of Release 5.3; it is planned for Release 5.4.

7.5 40G Muxponder Traffic Types

7.5.1 OC192 aggregation into an OTU3(V)

The following mappings are supported for OC192 aggregation. Lines highlighted in yellow are supported by the CS600x.

- **OC192 (BMP) -> ODU2(AMP 1.25/2.5) -> ODTU23 -> ODU3**
- **OC192 (BMP) -> ODU2(AMP 1.25/2.5) -> ODTU23 -> ODU3 -> OTU3**
- **OC192 (AMP) -> ODU2(DT AMP 1.25/2.5) -> ODTU23 -> ODU3**
- **OC192 (AMP) -> ODU2(DT AMP 1.25/2.5) -> ODTU23 -> ODU3 -> OTU3**
- **OC192 (BMP) -> ODU2(COR 2.5) -> ODTU23 -> ODU3ex**
- **OC192 (BMP) -> ODU2(COR 2.5) -> ODTU23 -> ODU3ex-> OTU3ex**
- **OC192 (BMP) -> ODU2(COR 1.25/2.5) -> ODTU23 -> ODU3+**
- **OC192 (BMP) -> ODU2(COR 1.25/2.5) -> ODTU23 -> ODU3 -> OTU3+**
- **OC192 (BMP) -> ODU2(GMP_HO 1.25) -> ODTU3ex.8 -> ODU3ex**
- **OC192 (BMP) -> ODU2(GMP_HO 1.25) -> ODTU3ex.8 -> ODU3ex-> OTU3ex**
- **OC192 (AMP) -> ODU2(DT GMP_HO 1.25) -> ODTU3ex.8 -> ODU3ex**
- **OC192 (AMP) -> ODU2(DT GMP_HO 1.25) -> ODTU3ex.8 -> ODU3ex->OTU3ex**

The function ten_hl_config_oc192_otu3v_t41 is used to provision OC192 SONET/SDH traffic into a container for an OTU3 or OTU3e. The OC192 traffic will ingress from the *client* port, be wrapped by the OPU2 and egress in the OTU3/3e via the *line* port.

The OC192 will be scrambled within the ODU2.

7.5.1.1 Relevant CS604x Datasheet Sections

2.7.1: ODTU23(e)/ODTU3(e).ts Dewrapper for De-Aggregation from an HO OxU3(e)
2.7.11: OC-192 Receive Transport Termination Functions
2.13.3: OC-192 / STM-64 Transmit Transport Termination Functions
2.14.5.6: Quad ODTU23(e)/ODTU3(e).ts to HO OPU3(e) Multiplexing / Aggregation
2.15.2: Signal Forwarding Unit

7.5.1.2 Function Call and Parameters

```
cs_status ten_hl_config_oc192_otu3v_t41(cs_uint16 module_id_line,
                                        cs_uint16 slice_line,
                                        cs_uint16 slice_client,
                                        cs_uint16 dyn_repro,
                                        cs_uint16 traffic_type_line,
                                        cs_uint16 fec_line,
                                        cs_uint16 term_oc192,
                                        cs_uint16 tcm_line,
                                        cs_uint16 mld_line,
                                        cs_uint16 map_odtu,
                                        cs_uint16 map_oxuv,
                                        cs_uint16 timeslots,
                                        cs_uint16 gmp_timeslot_mask)
```

Parameter description:

```
module_id_line
      Specifies the line's module ID

slice_line
      Specifies line's slice and is one of the following:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

slice_client
      Specifies client’s slice and is one of the following:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

dyn_repro
      Specifies the type of dynamic reprovisioning and is
      one of the following:
            0x00 = TEN_INITIAL_CONFIG
            0x01 = TEN_REPRO_CLIENT
            0x02 = TEN_REPRO_LINE_AND_CLIENT

traffic_type_line
      Specifies the traffic type of the line:
            0x01 = TEN_TRAFFIC_TYPE_OTU3
            0x02 = TEN_TRAFFIC_TYPE_OTU3E
            0x37 = TEN_TRAFFIC_TYPE_OTU3E2

fec_line
      Defines fec type
            0x00 = TEN_FEC_MODE_OTUkV (7%)
            0x01 = TEN_FEC_MODE_OTUkV_262 (10%)
            0x02 = TEN_FEC_MODE_OTUkV_273 (15%)
            0x03 = TEN_FEC_MODE_OTUkV_285 (20%)
            0x04 = TEN_FEC_MODE_OTUkV_297 (25%)
            0x05 = TEN_FEC_MODE_OTUkV_5_4 (25%)
            0x06 = TEN_FEC_MODE_OTUkV_301 (26%)
            0x07 = TEN_FEC_MODE_OTUkV_SDH_A (7%)
            0x08 = TEN_FEC_MODE_OTUkV_SDH_B (7 %)
            0x09 = TEN_FEC_MODE_OTUkV_4080_3929 (3.4%)
            0x10 = TEN_FEC_MODE_OTUkV_68_65 (4.2%)
            0x11 = TEN_FEC_MODE_OTUkV_267 (12%)
            0x12 = TEN_FEC_MODE_OTUkV_270 (13%)
            0x13 = TEN_FEC_MODE_GFEC (7%)
            0x14 = TEN_FEC_MODE_ZEROFEC
            0x15 = TEN_FEC_MODE_NOFEC

term_oc192
      Defines the way the OC192 will be sinked and sourced:
            0x01 = TEN_OC192_TERM_TRANSPARENT_CBR10
            0x02 = TEN_OC192_TERM_TRANSPARENT_REGENERATOR
            0x03 = TEN_OC192_TERM_RS_LAYER_REGENERATOR
            0x04 = TEN_OC192_TERM_MS_LAYER_REGENERATOR
tcm_line
      Specifies the termination for TCM 1 through 6. This
      is a bit encoded parameter with bits 1 through 6
      specifying TCM 1 through 6 respectively. These bits
      are defined as follows:
            0 = disable
            1 = enable

mld_line
      Enables distributing data across multiple lanes on
      line side i.e. OTL3.4
            0 = disable
            1 = enable

map_odtu
      Specifies the line mapping method
            0x00 = TEN_MAP_AMP
            0x02 = TEN_MAP_AMP_DT
            0x03 = TEN_MAP_AMP_PROP
            0x06 = TEN_MAP_GMP_HO

map_oxuv
      Specifies the client mapping method
            0x00 = TEN_MAP_AMP
            0x01 = TEN_MAP_BMP

timeslots
      Specifies number of timeslots. Bandwidth is always
      1.25G per timeslot for GMP_HO. Whereas for AMP it is
      as follows:                                    */
            16 timeslots - 2.5G per timeslot
            32 timeslots - 1.25G per timeslot
            timeslots = 1 – 8 for [map_proc] - GMP_HO
            timeslots = 16/32 for [map_proc] – Any AMP

gmp_timeslot_mask
      Specifies the timeslots to use for GMP mapping.
      Applicable to GMP_HO only.
            Bit 0 - Timeslot 1 through Bit 7 – Timeslot 8
```

7.5.2 OTU1e / 2 / 2e aggregation into an OTU3

The following mappings are supported for OTU1e aggregation. Lines highlighted in yellow are supported by the CS600x.

- **OTU1e-> ODU1e(COR 2.5)->ODTU23->ODU3ex**

- **OTU1e-> ODU1e(COR 2.5)->ODTU23->ODU3ex->OTU3ex**
- **OTU1e-> ODU1e(GMP_HO 1.25)->ODTU3ex.8->ODU3ex**
- **OTU1e-> ODU1e(GMP_HO 1.25)->ODTU3ex.8->ODU3ex->OTU3ex**

The following mappings are supported for OTU2 aggregation. Lines highlighted in yellow are supported by the CS600x.

- **OTU2-> ODU2(AMP 1.25/2.5) -> ODTU23 -> ODU3**
- **OTU2 -> ODU2(AMP 1.25/2.5) -> ODTU23 -> ODU3 -> OTU3**
- **OTU2 -> ODU2(COR 2.5) -> ODTU23 -> ODU3ex**
- **OTU2 -> ODU2(COR 2.5) -> ODTU23 -> ODU3ex-> OTU3ex**
- **OTU2 -> ODU2(GMP_HO 1.25) -> ODTU3ex.8 -> ODU3ex**
- **OTU2 -> ODU2(GMP_HO 1.25) -> ODTU3ex.8 -> ODU3e2->OTU3ex**

The following mappings are supported for OTU2e aggregation. Lines highlighted in yellow are supported by the CS600x.

- **OTU2e-> ODU2e(COR 2.5)->ODTU23->ODU3ex**
- **OTU2e-> ODU2e(COR 2.5)->ODTU23->ODU3ex->OTU3ex**
- **OTU2e-> ODU2e(GMP_HO 1.25)->ODTU3ex.8->ODU3ex**
- **OTU2e-> ODU2e(GMP_HO 1.25)->ODTU3ex.8->ODU3ex->OTU3ex**

The function ten_hl_config_otu2v_otu3v_t41 is used to provision OTU2 traffic into a container for an OTU3 or OTU3e. The OTU2 traffic will ingress from the *client* port, be wrapped by the ODU2 and egress in the OTU3/3e via the *line* port.

In other words, the flow is OTU2 → ODU2 → ODTU23 → OTU3/3e.

7.5.2.1 Relevant CS604x Datasheet Sections

2.7.1: ODTU23(e)/ODTU3(e).ts Dewrapper for De-Aggregation from an HO OxU3(e)
2.14.5.6: Quad ODTU23(e)/ODTU3(e).ts to HO OPU3(e) Multiplexing / Aggregation
2.15.2: Signal Forwarding Unit

7.5.2.2 Function Call and Parameters

```
cs_status ten_hl_config_otu2v_otu3v_t41(cs_uint16 module_id_line,
                                        cs_uint16 slice_line,
                                        cs_uint16 slice_client,
                                        cs_uint16 dyn_repro,
                                        cs_uint16 traffic_type_line,
                                        cs_uint16 fec_line,
                                        cs_uint16 traffic_type_client,
                                        cs_uint16 fec_client,
                                        cs_uint16 term_otu,
                                        cs_uint16 tcm_line,
                                        cs_uint16 tcm_client,
                                        cs_uint16 mld_line,
                                        cs_uint16 map_odtu,
                                        cs_uint16 map_oxuv,
                                        cs_uint16 timeslots,
                                        cs_uint16 gmp_timeslot_mask)
```

Parameter description:

```
module_id_line
      Specifies the line's module ID

slice_line
      Parameter specifies line's slice and is one of the
      following:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

slice_client
      Parameter specifies client's slice and is one of the
      following:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

dyn_repro
      Parameter specifies the type of dynamic
      reprovisioning and is one of the following:
            0x00 = TEN_INITIAL_CONFIG
            0x01 = TEN_REPRO_CLIENT
            0x02 = TEN_REPRO_LINE_AND_CLIENT

traffic_type_line
      Specifies the traffic type of the line:
            0x01 = TEN_TRAFFIC_TYPE_OTU3
            0x02 = TEN_TRAFFIC_TYPE_OTU3E
            0x37 = TEN_TRAFFIC_TYPE_OTU3E2

fec_line
      Defines fec type
            0x00 = TEN_FEC_MODE_OTUkV (7%)
            0x01 = TEN_FEC_MODE_OTUkV_262 (10%)
            0x02 = TEN_FEC_MODE_OTUkV_273 (15%)
            0x03 = TEN_FEC_MODE_OTUkV_285 (20%)
            0x04 = TEN_FEC_MODE_OTUkV_297 (25%)
            0x05 = TEN_FEC_MODE_OTUkV_5_4 (25%)
            0x06 = TEN_FEC_MODE_OTUkV_301 (26%)
            0x07 = TEN_FEC_MODE_OTUkV_SDH_A (7%)
            0x08 = TEN_FEC_MODE_OTUkV_SDH_B (7 %)
            0x09 = TEN_FEC_MODE_OTUkV_4080_3929 (3.4%)
            0x10 = TEN_FEC_MODE_OTUkV_68_65 (4.2%)
            0x11 = TEN_FEC_MODE_OTUkV_267 (12%)
            0x12 = TEN_FEC_MODE_OTUkV_270 (13%)
            0x13 = TEN_FEC_MODE_GFEC (7%)
            0x14 = TEN_FEC_MODE_ZEROFEC
            0x15 = TEN_FEC_MODE_NOFEC
```

```
traffic_type_client
      Specifies the traffic type of the client:
            0x05 = TEN_TRAFFIC_TYPE_OTU2
            0x06 = TEN_TRAFFIC_TYPE_OTU2e
            0x07 = TEN_TRAFFIC_TYPE_OTU1f
            0x1d = TEN_TRAFFIC_TYPE_OTU1e

fec_client
      Defines fec type
            0x00 = TEN_FEC_MODE_OTUkV (7%)
            0x01 = TEN_FEC_MODE_OTUkV_262 (10%)
            0x02 = TEN_FEC_MODE_OTUkV_273 (15%)
            0x03 = TEN_FEC_MODE_OTUkV_285 (20%)
            0x04 = TEN_FEC_MODE_OTUkV_297 (25%)
            0x05 = TEN_FEC_MODE_OTUkV_5_4 (25%)
            0x06 = TEN_FEC_MODE_OTUkV_301 (26%)
            0x07 = TEN_FEC_MODE_OTUkV_SDH_A (7%)
            0x08 = TEN_FEC_MODE_OTUkV_SDH_B (7 %)
            0x09 = TEN_FEC_MODE_OTUkV_4080_3929 (3.4%)
            0x10 = TEN_FEC_MODE_OTUkV_68_65 (4.2%)
            0x11 = TEN_FEC_MODE_OTUkV_267 (12%)
            0x12 = TEN_FEC_MODE_OTUkV_270 (13%)
            0x13 = TEN_FEC_MODE_GFEC (7%)
            0x14 = TEN_FEC_MODE_ZEROFEC
            0x15 = TEN_FEC_MODE_NOFEC

term_otu
      Parameter specifies the OTU overhead termination and
      is one of the following:
            0x00 = TEN_OTU_TERM_TRANSPARENT
            0x01 = TEN_OTU_TERM_SECTION
            0x02 = TEN_OTU_TERM_FULL

tcm_line
      Parameter specifies the termination for TCM 1 through
      6. This is a bit encoded parameter with bits 1
      through 6 specifying TCM 1 through 6 respectively.
      These bits are defined as follows:
            0 = disable
            1 = enable

tcm_client
      Parameter specifies the termination for TCM 1 through
      6. This is a bit encoded parameter with bits 1
      through 6 specifying TCM 1 through 6 respectively.
      These bits are defined as follows:
            0 = disable
            1 = enable

mld_line
      Enables distributing data across multiple lanes on
      line side i.e. OTL3.4
            0 = disable
            1 = enable

map_odtu
      Specifies the line mapping method
            0x00 = TEN_MAP_AMP
            0x02 = TEN_MAP_AMP_DT
            0x03 = TEN_MAP_AMP_PROP
            0x06 = TEN_MAP_GMP_HO

map_oxuv
      Specifies the client mapping method
            0x00 = TEN_MAP_AMP
            0x01 = TEN_MAP_BMP

timeslots
      Specifies number of timeslots. Bandwidth is always
      1.25G per timeslot for GMP_HO. Whereas for AMP it is
      as follows:                                    */
            16 timeslots - 2.5G per timeslot
            32 timeslots - 1.25G per timeslot
            timeslots = 1 – 8 for [map_proc] - GMP_HO
            timeslots = 16/32 for [map_proc] – Any AMP

gmp_timeslot_mask
      Specifies the timeslots to use for GMP mapping.
      Applicable to GMP_HO only.
            Bit 0 - Timeslot 1 through Bit 7 – Timeslot 8
```

7.5.3 OTU2e aggregation into an OTU23

See Section 7.5.2 above.

7.5.4 10G Ethernet Aggregation

The following mappings are supported for 10GE G.Sup43 6.1 aggregation. Lines highlighted in yellow are supported by the CS600x.

- **10GE6.1(RA/FC)->OC192(BMP)->ODU2(DT AMP 1.25/2.5)-> ODTU23->ODU3**
- **10GE6.1(RA/FC)->OC192(BMP)->ODU2(DT AMP 1.25/2.5)-> ODTU23->ODU3->OTU3**
- **10GE6.1(RA/FC)->OC192(BMP)->ODU2(DT GMP_HO 1.25)-> ODTU3ex.8->ODU3ex**
- **10GE6.1(RA/FC)->OC192(BMP)->ODU2(DT GMP_HO 1.25)-> ODTU3ex.8->ODU3ex->OTU3ex**
- **10GE6.1(RA/FC)->OC192(BMP)->ODU2(DT COR 2.5)-> ODTU23->ODU3ex**
- **10GE6.1(RA/FC)->OC192(BMP)->ODU2(DT COR 2.5)-> ODTU23->ODU3ex->OTU3ex**

The following mappings are supported for 10GE G.Sup43 6.2 aggregation. Lines highlighted in yellow are supported by the CS600x.

- **10GE6.2(RA->GFP-F)->ODU2(DT AMP 1.25/2.5)-> ODTU23->ODU3**
- **10GE6.2(RA->GFP-F)->ODU2(DT AMP 1.25/2.5)-> ODTU23->ODU3->OTU3**
- **10GE6.2(RA->GFP-F)->ODU2(DT GMP_HO 1.25)-> ODTU3e2.8->ODU3ex**
- **10GE6.2(RA->GFP-F)->ODU2(DT GMP_HO 1.25)-> ODTU3e2.8->ODU3ex-> OTU3ex**
- **10GE6.2(RA->GFP-F)->ODU2(DT COR 2.5)-> ODTU23->ODU3ex**
- **10GE6.2(RA->GFP-F)->ODU2(DT COR 2.5)-> ODTU23->ODU3ex->OTU3ex**

The following mappings are supported for 10GE G.Sup43 7.1 aggregation. Lines highlighted in yellow are supported by the CS600x.

- **10GE7.1(BMP)->ODU2e(GMP_HO 1.25)-> ODTU3.8->ODU3e**
- **10GE7.1(BMP)->ODU2e(GMP_HO 1.25)-> ODTU3.8->ODU3e->OTU3e**
- **10GE7.1(BMP)->ODU2e(COR 2.5)-> ODTU23->ODU3e**
- **10GE7.1(BMP)->ODU2e(COR 2.5)-> ODTU23->ODU3e->OTU3e**

The following mappings are supported for 10GE G.Sup43 7.2 aggregation. Lines highlighted in yellow are supported by the CS600x.

- **10GE7.2(BMP)->ODU1e(GMP_HO 1.25)-> ODTU3.8->ODU3e**
- **10GE7.2(BMP)->ODU1e(GMP_HO 1.25)-> ODTU3.8->ODU3e->OTU3e**
- **10GE7.2(BMP)->ODU1e(COR 2.5)-> ODTU23->ODU3e**
- **10GE7.2(BMP)->ODU1e(COR 2.5)-> ODTU23->ODU3e->OTU3e**

The following mappings are supported for 10GE G.Sup43 7.3 aggregation. Lines highlighted in yellow are supported by the CS600x.

- **10GE7.3(G43MD/GFP-F)->ODU2(DT AMP 1.25/2.5)-> ODTU23->ODU3**
- **10GE7.3(G43MD/GFP-F)->ODU2(DT AMP 1.25/2.5)-> ODTU23->ODU3->OTU3/3x**
- **10GE7.3(G43MD/GFP-F)->ODU2(DT GMP_HO 1.25)-> ODTU3.8->ODU3ex**
- **10GE7.3(G43MD/GFP-F)->ODU2(DT GMP_HO 1.25)-> ODTU3.8->ODU3ex->OTU3ex**
- **10GE7.3(G43MD/GFP-F)->ODU2(DT COR 2.5)-> ODTU23->ODU3ex**
- **10GE7.3(G43MD/GFP-F)->ODU2(DT COR 2.5)-> ODTU23->ODU3ex-> OTU3ex**

7.5.4.1 Relevant CS604x Datasheet Sections

Section 2.2: Mapping of Data Client Signals
Section 2.8.1: Receive GFP
Section 2.8.1.4.1
Section 2.12.8: Transmit GFP
Section 2.12.8.2.1

7.5.4.2 Function Call and Parameters

```
cs_status ten_hl_config_10ge_otu3v_t41 (cs_uint16 module_id_line,
                                        cs_uint16 slice_line,
                                        cs_uint16 slice_client,
                                        cs_uint16 dyn_repro,
                                        cs_uint16 traffic_type_line,
                                        cs_uint16 fec_line,
                                        cs_uint16 traffic_type_client,
                                        cs_uint16 sysclk,
                                        cs_uint16 tcm_line,
                                        cs_uint16 mld_line,
                                        cs_uint16 map_odtu,
                                        cs_uint16 map_oxuv,
                                        cs_uint16 timeslots,
                                        cs_uint16 gfp_frame_format,
                                        flow_ctrl_mode)
```

Parameter description:

```
module_id_line
      Specifies line's module ID

slice_line
      Specifies line's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

slice_client
      Specifies client's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

dyn_repro
      Specifies the type of dynamic reprovisioning:
            0x00 = TEN_INITIAL_CONFIG
            0x01 = TEN_REPRO_CLIENT
            0x02 = TEN_REPRO_LINE_AND_CLIENT

traffic_type_line
      Specifies the traffic type of the line:
            0x01 = TEN_TRAFFIC_TYPE_OTU3
            0x02 = TEN_TRAFFIC_TYPE_OTU3E
            0x37 = TEN_TRAFFIC_TYPE_OTU3E2

fec_line
      Defines fec type
            0x00 = TEN_FEC_MODE_OTUkV (7%)
            0x01 = TEN_FEC_MODE_OTUkV_262 (10%)
            0x02 = TEN_FEC_MODE_OTUkV_273 (15%)
            0x03 = TEN_FEC_MODE_OTUkV_285 (20%)
            0x04 = TEN_FEC_MODE_OTUkV_297 (25%)
            0x05 = TEN_FEC_MODE_OTUkV_5_4 (25%)
            0x06 = TEN_FEC_MODE_OTUkV_301 (26%)
            0x07 = TEN_FEC_MODE_OTUkV_SDH_A (7%)
            0x08 = TEN_FEC_MODE_OTUkV_SDH_B (7 %)
            0x09 = TEN_FEC_MODE_OTUkV_4080_3929 (3.4%)
            0x10 = TEN_FEC_MODE_OTUkV_68_65 (4.2%)
            0x11 = TEN_FEC_MODE_OTUkV_267 (12%)
            0x12 = TEN_FEC_MODE_OTUkV_270 (13%)
            0x13 = TEN_FEC_MODE_GFEC (7%)
            0x14 = TEN_FEC_MODE_ZEROFEC
            0x15 = TEN_FEC_MODE_NOFEC

traffic_type_client
      Specifies the traffic type of the client:
            0x09 = TEN_TRAFFIC_TYPE_10GE_WAN
            0x0a = TEN_TRAFFIC_TYPE_10GE_6_2
            0x0b = TEN_TRAFFIC_TYPE_10GE_7_1
            0x0c = TEN_TRAFFIC_TYPE_10GE_7_3
            0x1a = TEN_TRAFFIC_TYPE_10GE_6_1
            0x1b = TEN_TRAFFIC_TYPE_10GE_7_2
            0x1c = TEN_TRAFFIC_TYPE_10GE_GFPF_OC192_ODU2

sysclk

      Specifies the sysclk rate in MHz

tcm_line
      Parameter specifies the termination for TCM 1 through
      6. This is a bit encoded parameter with bits 1
      through 6 specifying TCM 1 through 6 respectively.
      These bits are defined as follows:
            0 = disable
            1 = enable

mld_line
      Enables distributing data across multiple lanes on
      line side i.e. OTL3.4
            0 = disable
            1 = enable

map_odtu
      Specifies the line mapping method
            0x00 = TEN_MAP_AMP
            0x02 = TEN_MAP_AMP_DT
            0x03 = TEN_MAP_AMP_PROP
            0x06 = TEN_MAP_GMP_HO

map_oxuv
      Specifies the client mapping method
            0x00 = TEN_MAP_AMP
            0x01 = TEN_MAP_BMP
timeslots
      Specifies number of timeslots. Bandwidth is always
      1.25G per timeslot for GMP_HO. Whereas for AMP it is
      as follows:
            16 timeslots - 2.5G per timeslot
            32 timeslots - 1.25G per timeslot
            timeslots = 1 – 8 for [map_proc] - GMP_HO
            timeslots = 16/32 for [map_proc] – Any AMP

gfp_frame_format
      Specifies the GFP frame format
            0x00 = TEN_28_BLOCKS_5_SUPERBLOCKS_CRC4
            0x01 = TEN_28_BLOCKS_11_SUPERBLOCKS_CRC4
            0x02 = TEN_32_BLOCKS_4_SUPERBLOCKS_NOCRC
            0x03 = TEN_32_BLOCKS_8_SUPERBLOCKS_NOCRC

flow_ctrl_mode
      Specifies the flow control mode
            0x00 = TEN_PP10G_FLOW_GFP_S_P
            0x01 = TEN_PP10G_FLOW_GFP_S
            0x02 = TEN_PP10G_FLOW_ETH_4
            0x03 = TEN_PP10G_FLOW_ETH_4_P
            0x04 = TEN_PP10G_FLOW_MAX_MODE
```

7.5.5 10G Ethernet Using GFP-F G.Sup43.7.3 Aggregation

Refer to Section 7.5.4.

7.5.6 Aggregation Idle State (for dynamic reprovisioning)

This API inserts ODUk-OCI in the channel in preparation for dynamic reprovisioning.

When a path has been provisioned, it cannot be directly transitioned to another payload-carrying traffic type. Instead, it must first be assigned to the idle traffic type. Once the idle state has been achieved, the path may be reprovisioned to the desired end traffic type.

7.5.6.1 Relevant CS604x Datasheet Sections

Section 2.7: Receive 10G Circuit Processor
Section 2.13: Transmit 10G Circuit Processor

7.5.6.2 Function Call and Parameters

There is an issue where muxponder idle channel ODU2s will not be in frame when initially provisioned. This is a bug in the driver, not in the device. The workaround is to provision all four channels with some sort of traffic. If channels need to be idle then they can be idled after the initial traffic provisioning with a call to the following API function.

```
cs_status ten_hl_config_aggregation_idle (
      cs_uint16 mod_line,
      cs_uint8 slice_line,
      cs_uint16 mod_client,
      cs_uint8 slice_client)
```

Parameter description:

```
mod_line
      Specifies line's module ID

slice_line
      Specifies line's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3
            0xFF = TEN_SLICE_ALL

mod_client
      Specifies the client's module ID

slice_client
      Specifies client's slice:
            0x00 = TEN_SLICE0
```

```
0x01 = TEN_SLICE1
0x02 = TEN_SLICE2
0x03 = TEN_SLICE3
0xFF = TEN_SLICE_ALL
```

7.5.7 Infiniband

NOTE: Infiniband is not currently supported in Release 5.1.

The following mappings are supported for 10GE G.Sup43 6.1 aggregation.

- **5G IB (GMP_LO)->ODU2->ODTU23(DT AMP 1.25/2.5) ->ODU3**
- **5G IB (GMP_LO)->ODU2->ODTU23(DT AMP 1.25/2.5) ->ODU3->OTU3**
- **5G IB (GMP_LO)->ODU2e->ODTU23(DT AMP 1.25/2.5) ->ODU3**
- **5G IB (GMP_LO)->ODU2e-> ODTU23(DT AMP 1.25/2.5)->ODU3->OTU3e**
- **5G IB (GMP_LO)->ODU2->ODTU23(DT GMP_HO 1.25)->ODU3**
- **5G IB (GMP_LO)->ODU2->ODTU23(DT GMP_HO 1.25)->ODU3->OTU3**
- **5G IB (GMP_LO)->ODU2e->ODTU23(DT GMP_HO 1.25)->ODU3**
- **5G IB (GMP_LO)->ODU2e-> ODTU23(DT GMP_HO 1.25)->ODU3->OTU3e**
- **5G IB (BMP)->ODUflex(GMP_HO 1.25)->ODTU3.5->ODU3**
- **5G IB (BMP)->ODUflex(GMP_HO 1.25)->ODTU3.5->ODU3->OTU3**
- **5G IB (BMP)->ODUflex(GMP_HO 1.25)->ODTU3.5->ODU3e**
- **5G IB (BMP)->ODUflex(GMP_HO 1.25)->ODTU3.5->ODU3e->OTU3e**
- **5G IB (COR 2.5)->ODU2->ODTU23(DT AMP)->ODU3**
- **5G IB (COR 2.5)->ODU2->ODTU23(DT AMP)->ODU3->OTU3**
- **5G IB (COR 2.5)->ODU2->ODTU23(DT COR)->ODU3e**
- **5G IB (COR 2.5)->ODU2->ODTU23(DT COR)->ODU3/3e->OTU3e**
- **10G IB (GMP_LO)->ODU2e->ODTU23(DT AMP 1.25/2.5) ->ODU3e**
- **10G IB (GMP_LO)->ODU2e-> ODTU23(DT AMP 1.25/2.5)->ODU3e->OTU3e**
- **10G IB (GMP_LO)->ODU2e->ODTU23(DT GMP_HO 1.25)->ODU3e**
- **10G IB (GMP_LO)->ODU2e-> ODTU23(DT GMP_HO 1.25)->ODU3e->OTU3e**
- **10G IB (GMP_LO)->ODU2e->ODTU23(COR 2.5)->ODU3e**
- **10G IB (GMP_LO)->ODU2e->ODTU23(COR 2.5)->ODU3e->OTU3e**
- **10G IB (BMP)->ODUflex(GMP_HO 1.25)->ODTU3.8->ODU3e**
- **10G IB (BMP)->ODUflex(GMP_HO 1.25)->ODTU3.8->ODU3e->OTU3e**
- **10G IB (COR 2.5)->ODU2->ODTU23(DT AMP 2.5)->ODU3**
- **10G IB (COR 2.5)->ODU2->ODTU23(DT AMP 2.5)->ODU3->OTU3**
- **10G IB (COR 2.5)->ODU2->ODTU23(DT COR)->ODU3e->OTU3e**

7.5.7.1 Relevant CS604x Datasheet Sections

Section 2.2.6: Infiniband Clients

7.5.7.2 Function Call and Parameters

```
cs_status ten_hl_config_infiniband_otu3v_t41(cs_uint16 module_id_line,
                                             cs_uint16 slice_line,
                                             cs_uint16 slice_client,
                                             cs_uint16 dyn_repro,
                                             cs_uint16 traffic_type_line,
 cs_uint16 fec_line,
cs_uint16 traffic_type_client,
 cs_uint16 sysclk,
 cs_uint16 tcm_line,
 cs_uint16 mld_line,
 cs_uint16 map_odtu,
 cs_uint16 map_oxuv,
 cs_uint16 timeslots,
 cs_uint16 gmp_timeslot_mask)
```

Parameter description:

```
mod_id_line
      Specifies line's module ID

slice_line
      Specifies line's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3
            0xFF = TEN_SLICE_ALL

slice_client
      Specifies client's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3
            0xFF = TEN_SLICE_ALL

dyn_repro
      Specifies the type of dynamic reprovisioning:
            0x00 = TEN_INITIAL_CONFIG
            0x01 = TEN_REPRO_CLIENT
            0x02 = TEN_REPRO_LINE_AND_CLIENT

traffic_type_line
      Specifies the traffic type of the line:
            0x01 = TEN_TRAFFIC_TYPE_OTU3
            0x02 = TEN_TRAFFIC_TYPE_OTU3E
            0x37 = TEN_TRAFFIC_TYPE_OTU3E2

fec_line
      Defines fec type
            0x00 = TEN_FEC_MODE_OTUkV (7%)
            0x01 = TEN_FEC_MODE_OTUkV_262 (10%)
            0x02 = TEN_FEC_MODE_OTUkV_273 (15%)
            0x03 = TEN_FEC_MODE_OTUkV_285 (20%)
            0x04 = TEN_FEC_MODE_OTUkV_297 (25%)
            0x05 = TEN_FEC_MODE_OTUkV_5_4 (25%)
            0x06 = TEN_FEC_MODE_OTUkV_301 (26%)
            0x07 = TEN_FEC_MODE_OTUkV_SDH_A (7%)
            0x08 = TEN_FEC_MODE_OTUkV_SDH_B (7 %)
            0x09 = TEN_FEC_MODE_OTUkV_4080_3929 (3.4%)
            0x10 = TEN_FEC_MODE_OTUkV_68_65 (4.2%)
            0x11 = TEN_FEC_MODE_OTUkV_267 (12%)
            0x12 = TEN_FEC_MODE_OTUkV_270 (13%)
            0x13 = TEN_FEC_MODE_GFEC (7%)
            0x14 = TEN_FEC_MODE_ZEROFEC
            0x15 = TEN_FEC_MODE_NOFEC

traffic_type_client
      Specifies the traffic type of the client:
            0x12 = TEN_TRAFFIC_TYPE_5GIB
            0x21 = TEN_TRAFFIC_TYPE_10GIB
            0x22 = TEN_TRAFFIC_TYPE_12_5GIB

sysclk
      Specifies the sysclk frequency in Hz. For example,
      400000000.

tcm_line
      Specifies the termination for TCM 1 through 6. This
      is a bit encoded parameter with bits 1 through 6
      specifying TCM 1 through 6 respectively. These bits
      are defined as follows:
            disable = 0
            enable  = 1

mld_line
      Enables distributing data across multiple lanes on
      line side i.e. OTL3.4
            disable = 0
            enable  = 1

map_odtu
      Specifies the client mapping method
            0x02 = TEN_MAP_AMP_DT
            0x03 = TEN_MAP_AMP_PROP
            0x04 = TEN_MAP_AMP_PROP_DT
            0x06 = TEN_MAP_GMP_HO
            0x07 = TEN_MAP_GMP_HO_DT

map_oxuv
      Specifies the client mapping method
            0x01 = TEN_MAP_BMP
            0x03 = TEN_MAP_AMP_PROP
            0x05 = TEN_MAP_GMP_LO

timeslots
      Specifies number of timeslots. Bandwidth is always
      1.25G per timeslot for GMP_HO. Whereas for AMP it is
      as follows:
            16 timeslots  - 2.5G per timeslot
            32 timeslots  - 1.25G per timeslot
            timeslots = 1 - 8      for [map_proc] - GMP_HO
            timeslots = 16/32      for [map_proc]  Any AMP
```

```
gmp_timeslot_mask
      Specifies the timeslots to use for GMP mapping.
      Applicable to GMP_HO only.
            Bit 0- Timeslot 1 through Bit 7  Timeslot 8
```

7.6 10G Transponder Traffic Types

7.6.1 Idle State (for Dynamic Reprovisioning)

This API inserts ODUk-OCI in the channel in preparation for dynamic reprovisioning.

When a path has been provisioned, it cannot be directly transitioned to another payload-carrying traffic type. Instead, it must first be assigned to the idle traffic type. Once the idle state has been achieved, the path may be reprovisioned to the desired end traffic type.

7.6.1.1 Relevant CS604x Datasheet Sections

Section 2.7: Receive 10G Circuit Processor
Section 2.13: Transmit 10G Circuit Processor

7.6.1.2 Function Call and Parameters

```
ten_hl_config_idle (
      cs_uint16 mod_line,
      cs_uint8 slice_line,
      cs_uint16 mod_client,
      cs_uint8 slice_client,
      cs_uint16 dyn_repro)
```

Parameter description:

```
mod_line
      Specifies line's module ID

slice_line
      Specifies line's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3
            0xFF = TEN_SLICE_ALL

mod_client
      Specifies the client's module ID

slice_client
      Specifies client's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3
            0xFF = TEN_SLICE_ALL
```

```
dyn_repro
      Specifies the type of dynamic reprovisioning:
            0x00 = TEN_INITIAL_CONFIG
            0x01 = TEN_REPRO_CLIENT
            0x02 = TEN_REPRO_LINE_AND_CLIENT
```

7.6.2 OC192 over ODTU23

This section removed intentionally.

7.6.3 OC192 over OTU2

The following mappings are supported for OC192 into an OTU2. Lines highlighted in yellow are supported by the CS600x.

- **OC192 wire**
- **OC192 (BMP) -> ODU2**
- **OC192 (BMP) -> ODU2 -> OTU2**
- **OC192 (AMP) -> ODU2**
- **OC192 (AMP) -> ODU2 -> OTU2**

The function ten_hl_config_oc192_otu2v_t41 is used to map an OC192 → OTU2.

7.6.3.1 Relevant CS604x Datasheet Sections

2.7: Receive 10G Circuit Processor
2.7.11: OC-192 Receive Transport Termination Functions
2.13: Transmit 10G Circuit Processor
2.13.3: OC-192/STM-64 Transmit Transport Termination Function

7.6.3.2 Associated APIs

ten_hl_config_oc192_otu2v_t41

7.6.3.3 High-level API Call and Parameters

```
cs_status ten_hl_config_oc192_otu2v_t41(cs_uint16 module_id_line,
                                        cs_uint16 slice_line,
                                        cs_uint16 module_id_client,
                                        cs_uint16 slice_client,
                                        cs_uint16 dyn_repro,
                                        cs_uint16 traffic_type_line,
                                        cs_uint16 fec_line,
                                        cs_uint16 term_oc192,
                                        cs_uint16 tcm_line,
                                        cs_uint16 map_oxuv)
```

Parameter description:

```
module_id_line
      Specifies line's module ID

slice_line
```

```
      Specifies line's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

module_id_client
      Specifies the client's module ID

slice_client
      Specifies client's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

dyn_repro
      Specifies the type of dynamic reprovisioning:
            0x00 = TEN_INITIAL_CONFIG
            0x01 = TEN_REPRO_CLIENT
            0x02 = TEN_REPRO_LINE_AND_CLIENT

traffic_type_line
      Specifies the traffic type of the line:
            0x05 = TEN_TRAFFIC_TYPE_OTU2

fec_line
      Defines fec type
            0x00 = TEN_FEC_MODE_OTUkV (7%)
            0x01 = TEN_FEC_MODE_OTUkV_262 (10%)
            0x02 = TEN_FEC_MODE_OTUkV_273 (15%)
            0x03 = TEN_FEC_MODE_OTUkV_285 (20%)
            0x04 = TEN_FEC_MODE_OTUkV_297 (25%)
            0x05 = TEN_FEC_MODE_OTUkV_5_4 (25%)
            0x06 = TEN_FEC_MODE_OTUkV_301 (26%)
            0x07 = TEN_FEC_MODE_OTUkV_SDH_A (7%)
            0x08 = TEN_FEC_MODE_OTUkV_SDH_B (7 %)
            0x09 = TEN_FEC_MODE_OTUkV_4080_3929 (3.4%)
            0x10 = TEN_FEC_MODE_OTUkV_68_65 (4.2%)
            0x11 = TEN_FEC_MODE_OTUkV_267 (12%)
            0x12 = TEN_FEC_MODE_OTUkV_270 (13%)
            0x13 = TEN_FEC_MODE_GFEC (7%)
            0x14 = TEN_FEC_MODE_ZEROFEC
            0x15 = TEN_FEC_MODE_NOFEC

term_oc192
      Defines the way the OC192 will be sinked and sourced:
            0x01 = TEN_OC192_TERM_TRANSPARENT_CBR10
            0x02 = TEN_OC192_TERM_TRANSPARENT_REGENERATOR
            0x03 = TEN_OC192_TERM_RS_LAYER_REGENERATOR
            0x04 = TEN_OC192_TERM_MS_LAYER_REGENERATOR

tcm_line
```

```
            Parameter specifies the termination for TCM 1 through
            6. This is a bit encoded parameter with bits 1
            through 6 specifying TCM 1 through 6 respectively.
            These bits are defined as follows:
                  0 = disable
                  1 = enable

      map_oxuv
            Specifies the client mapping method
                  0x00 = TEN_MAP_AMP
                  0x01 = TEN_MAP_BMP
```

7.6.4 OTU1f to OTU1f

Refer to Section 7.6.6.

7.6.5 OTU1e to ODTU23

This section removed intentionally.

7.6.6 OTU2 to OTU2

The following mappings are supported for OTU1e into an OTU1e. Lines highlighted in yellow are supported by the CS600x.

- **OTU1e->ODU1e**
- **OTU1e->ODU1e->OTU1e**

The following mappings are supported for OTU1f into an OTU1f. Lines highlighted in yellow are supported by the CS600x.

- **OTU1f->ODU1f**
- **OTU1f->ODU1f->OTU1f**

The following mappings are supported for OTU2 into an OTU2. Lines highlighted in yellow are supported by the CS600x.

- **OTU2 wire**
- **OTU2 -> ODU2**
- **OTU2 -> ODU2 -> OTU2**
- **OTU2 -> ODU2(AMP)->client-> ODU2(AMP)->OTU2**

The following mappings are supported for OTU2e into an OTU2e. Lines highlighted in yellow are supported by the CS600x.

- **OTU2e wire**
- **OTU2e->ODU2e**
- **OTU2e->ODU2e->OTU2e**

The following mappings are supported for OTU2f into an OTU2f.

- **OTU2f->ODU2f**
- **OTU2f->ODU2f->OTU2f**

The function ten_hl_config_otu2v_otu2v_t41 is used to provision traffic from line to client.

7.6.6.1 Relevant CS604x Datasheet Sections

2.7: Receive 10G Circuit Processor
2.13: Transmit 10G Circuit Processor

7.6.6.2 Associated APIs

ten_hl_config_otu2v_otu2v_t41

7.6.6.3 High-level API Call and Parameters

```
cs_status ten_hl_config_otu2v_otu2v_t41(cs_uint16 module_id_line,
                                        cs_uint16 slice_line,
                                        cs_uint16 module_id_client,
                                        cs_uint16 slice_client,
                                        cs_uint16 dyn_repro,
                                        cs_uint16 traffic_type_line,
                                        cs_uint16 fec_line,
                                        cs_uint16 traffic_type_client,
                                        cs_uint16 fec_client,
                                        cs_uint16 term_otu,
                                        cs_uint16 tcm_line,
                                        cs_uint16 tcm_client,
                                        cs_uint16 map_oxuv)
```

Parameter description:

```
module_id_line
      Specifies the line's module ID

slice_line
      Specifies line's slice and is one of the following:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

module_id_client
      Specifies the client's module ID

slice_client
      Specifies client’s slice and is one of the following:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

dyn_repro
      Specifies the type of dynamic reprovisioning and is
      one of the following:
            0x00 = TEN_INITIAL_CONFIG
            0x01 = TEN_REPRO_CLIENT
            0x02 = TEN_REPRO_LINE_AND_CLIENT
```

```
traffic_type_line and traffic_type_client
      Specify the traffic type of the line and the client:
            0x05 = TEN_TRAFFIC_TYPE_OTU2
            0x06 = TEN_TRAFFIC_TYPE_OTU2E
            0x07 = TEN_TRAFFIC_TYPE_OTU1F
            0x1d = TEN_TRAFFIC_TYPE_OTU1E

fec_line and fec_client
      Defines fec type
            0x00 = TEN_FEC_MODE_OTUkV (7%)
            0x01 = TEN_FEC_MODE_OTUkV_262 (10%)
            0x02 = TEN_FEC_MODE_OTUkV_273 (15%)
            0x03 = TEN_FEC_MODE_OTUkV_285 (20%)
            0x04 = TEN_FEC_MODE_OTUkV_297 (25%)
            0x05 = TEN_FEC_MODE_OTUkV_5_4 (25%)
            0x06 = TEN_FEC_MODE_OTUkV_301 (26%)
            0x07 = TEN_FEC_MODE_OTUkV_SDH_A (7%)
            0x08 = TEN_FEC_MODE_OTUkV_SDH_B (7 %)
            0x09 = TEN_FEC_MODE_OTUkV_4080_3929 (3.4%)
            0x10 = TEN_FEC_MODE_OTUkV_68_65 (4.2%)
            0x11 = TEN_FEC_MODE_OTUkV_267 (12%)
            0x12 = TEN_FEC_MODE_OTUkV_270 (13%)
            0x13 = TEN_FEC_MODE_GFEC (7%)
            0x14 = TEN_FEC_MODE_ZEROFEC
            0x15 = TEN_FEC_MODE_NOFEC

term_otu
      Specifies the OTU overhead termination and is one of
      the following:
            0x00 = TEN_OTU_TERM_TRANSPARENT
            0x01 = TEN_OTU_TERM_SECTION
            0x02 = TEN_OTU_TERM_FULL

tcm_line
      Specifies the termination for TCM 1 through 6.  This
      is a bit encoded parameter with bits 1 through 6
      specifying TCM 1 through 6 respectively. These bits
      are defined as follows:
            0 = disable
            1 = enable

tcm_client
      Specifies the termination for TCM 1 through 6.  This
      is a bit encoded parameter with bits 1 through 6
      specifying TCM 1 through 6 respectively. These bits
      are defined as follows:
            0 = disable
            1 = enable

map_oxuv
      Specifies the client mapping method
            0x00 = TEN_MAP_AMP
            0x01 = TEN_MAP_BMP
```

7.6.7 OTU2e to OTU2e

Refer to Section 7.6.6.

7.6.8 OTU2e to ODTU23

This section removed intentionally.

7.6.9 10G Ethernet Into 10G Ethernet

This API maps a 10GE client into a 10GE line.

7.6.9.1 Relevant CS604x Datasheet Sections

Section 2.2: Mapping of Data Client Signals
Section 2.8: Receive 10G Packet Processor
Section 2.12: Transmit 10G Packet Processor

7.6.9.2 Function Call and Parameters

```
cs_status ten_hl_config_10ge_10ge(
      cs_uint16 module_id_line,
      cs_uint8  slice_line,
      cs_uint16 module_id_client,
      cs_uint8  slice_client,
      cs_uint8  dyn_repro,
      cs_uint16 sync,
      cs_uint16 sysclk)
```

Parameter description:

```
module_id_line
      Specifies line's module ID

slice_line
      Specifies line's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

module_id_client
      Specifies the client's module ID

slice_client
      Specifies client's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

dyn_repro
      Specifies the type of dynamic reprovisioning:
            0x00 = TEN_INITIAL_CONFIG
            0x01 = TEN_REPRO_CLIENT
            0x02 = TEN_REPRO_LINE_AND_CLIENT
```

```
sync
      Specifies synchronous or asynchronous:
            0=async
            1=sync

sysclk
      Specifies the sysclk rate (in MHz)
```

7.6.10 10G Ethernet into an OTU1e/2/2e

The following mappings are supported for 10GE G.Sup43 6.1 into an OTU2. Lines highlighted in yellow are supported by the CS600x.

- **10GE6.1(RA/FC)->OC192->ODU2**
- **10GE6.1(RA/FC)->OC192->ODU2->OTU2**

The following mappings are supported for 10GE G.Sup43 6.2 into an OTU2. Lines highlighted in yellow are supported by the CS600x.

- **10GE6.2(RA->GFP-F)->ODU2**
- **10GE6.2(RA->GFP-F)->ODU2->OTU2**

The following mappings are supported for 10GE G.Sup43 7.1 into an OTU2e. Lines highlighted in yellow are supported by the CS600x.

- **10GE7.1(BMP)->ODU2e**
- **10GE7.1(BMP)->ODU2e->OTU2e**

The following mappings are supported for 10GE G.Sup43 7.2 into an OTU1e. Lines highlighted in yellow are supported by the CS600x.

- **10GE7.2(BMP)->ODU1e**
- **10GE7.2(BMP)->ODU1e->OTU1e**

The following mappings are supported for 10GE G.Sup43 7.3 into an OTU2. Lines highlighted in yellow are supported by the CS600x.

- **10GE7.3(G43MD->GFP-F)->ODU2**
- **10GE7.3(G43MD->GFP-F)->ODU2->OTU2**

7.6.10.1 Relevant CS604x Datasheet Sections

Section 2.2: Mapping of Data Client Signals
Section 2.8.1: Receive GFP
Section 2.12.8: Transmit GFP

7.6.10.2 Function Call and Parameters

```
cs_status ten_hl_config_10ge_otu2v_t41(cs_uint16 module_id_line,
                                       cs_uint16 slice_line,
                                       cs_uint16 module_id_client,
                                       cs_uint16 slice_client,
                                       cs_uint16 dyn_repro,
                                       cs_uint16 traffic_type_line,
                                       cs_uint16 fec_line,
                                       cs_uint16 traffic_type_client,
                                       cs_uint16 sysclk,
                                       cs_uint16 tcm_line
                                       cs_uint16 gfp_frame_format,
                                       cs_uint16 flow_ctrl_mode)
```

Parameter description:

```
module_id_line
      Specifies line's module ID

slice_line
      Specifies line's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

module_id_client
      Specifies the client's module ID

slice_client
      Specifies client's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

dyn_repro
      Specifies the type of dynamic reprovisioning:
            0x00 = TEN_INITIAL_CONFIG
            0x01 = TEN_REPRO_CLIENT
            0x02 = TEN_REPRO_LINE_AND_CLIENT

traffic_type_line
      Specify the traffic type of the line:
            0x05 = TEN_TRAFFIC_TYPE_OTU2
            0x06 = TEN_TRAFFIC_TYPE_OTU2E
            0x07 = TEN_TRAFFIC_TYPE_OTU1F
            0x1d = TEN_TRAFFIC_TYPE_OTU1E

fec_line
      Defines fec type
            0x00 = TEN_FEC_MODE_OTUkV (7%)
            0x01 = TEN_FEC_MODE_OTUkV_262 (10%)
            0x02 = TEN_FEC_MODE_OTUkV_273 (15%)
            0x03 = TEN_FEC_MODE_OTUkV_285 (20%)
            0x04 = TEN_FEC_MODE_OTUkV_297 (25%)
            0x05 = TEN_FEC_MODE_OTUkV_5_4 (25%)
            0x06 = TEN_FEC_MODE_OTUkV_301 (26%)
```

```
                  0x07 = TEN_FEC_MODE_OTUkV_SDH_A (7%)
                  0x08 = TEN_FEC_MODE_OTUkV_SDH_B (7 %)
                  0x09 = TEN_FEC_MODE_OTUkV_4080_3929 (3.4%)
                  0x10 = TEN_FEC_MODE_OTUkV_68_65 (4.2%)
                  0x11 = TEN_FEC_MODE_OTUkV_267 (12%)
                  0x12 = TEN_FEC_MODE_OTUkV_270 (13%)
                  0x13 = TEN_FEC_MODE_GFEC (7%)
                  0x14 = TEN_FEC_MODE_ZEROFEC
                  0x15 = TEN_FEC_MODE_NOFEC

traffic_type_client
      Specifies the traffic type of the client:
            0x09 = TEN_TRAFFIC_TYPE_10GE_WAN
            0x0a = TEN_TRAFFIC_TYPE_10GE_6_2
            0x0b = TEN_TRAFFIC_TYPE_10GE_7_1
            0x0c = TEN_TRAFFIC_TYPE_10GE_7_3
            0x1a = TEN_TRAFFIC_TYPE_10GE_6_1
            0x1b = TEN_TRAFFIC_TYPE_10GE_7_2
            0x1d = TEN_TRAFFIC_TYPE_10GE_GFPF_OC192_ODU2

sysclk
      Specifies the sysclk rate (in MHz)

tcm_line
      Specifies the termination for TCM 1 through 6.  This
      is a bit encoded parameter with bits 1 through 6
      specifying TCM 1 through 6 respectively. These bits
      are defined as follows:
            0 = disable
            1 = enable

gfp_frame_format
      Specifies the GFP frame format
            0x00 = TEN_28_BLOCKS_5_SUPERBLOCKS_CRC4
            0x01 = TEN_28_BLOCKS_11_SUPERBLOCKS_CRC4
            0x02 = TEN_32_BLOCKS_4_SUPERBLOCKS_NOCRC
            0x03 = TEN_32_BLOCKS_8_SUPERBLOCKS_NOCRC

flow_ctrl_mode
      Specifies the flow control mode
            0x00 = TEN_PP10G_FLOW_GFP_S_P
            0x01 = TEN_PP10G_FLOW_GFP_S
            0x02 = TEN_PP10G_FLOW_ETH_4
            0x03 = TEN_PP10G_FLOW_ETH_4_P
            0x04 = TEN_PP10G_FLOW_MAX_MODE
```

7.6.11 10G Ethernet Using GFP-S G.Sup43.6.2 into an ODTU23

This section removed intentionally.

7.6.12 10G Ethernet Using GFP-F G.Sup43.7.2 into an ODTU23

This section removed intentionally.

7.6.13 10G Ethernet Using GFP-F G.Sup43.7.2 into an OTU1e

Refer to Section 7.6.10.

7.6.14 10G Ethernet Using GFP-F G.Sup43.7.3 into an OTU2

Refer to Section 7.6.10.

7.6.15 10G Ethernet into an OTU2e

Refer to Section 7.6.10.

7.6.16 FibreChannel into an OTU1f/2/2e

The following mappings are supported for Fibre Channel into an OTU2. Lines highlighted in yellow are supported by the CS600x.

- **4/8GFC (RA)->ODU2**
- **4/8GFC (RA)->ODU2->OTU2**
- **4/8GFC (COR AMP)->ODU2**
- **4/8GFC (COR AMP)->ODU2->OTU2**
- **4/8GFC(GMP_LO)-> ODU2**
- **4/8GFC(GMP_LO)-> ODU2->OTU2**
- **4/8GFC(BMP)-> ODUFlex(GMP_HO)->ODU2**
- **4/8GFC(BMP)-> ODUFlex(GMP_HO)->ODU2->OTU2**

The following mappings are supported for Fibre Channel into an OTU1f. Lines highlighted in yellow are supported by the CS600x.

- **10GFC(BMP)->ODU1f**
- **10GFC(BMP)->ODU1f->OTU1f**

The following mappings are supported for Fibre Channel into an OTU2f. Lines highlighted in yellow are supported by the CS600x.

- **10GFC(BMP)->ODU2f**
- **10GFC(BMP)->ODU2f->OTU2f**

The following mappings are supported for Fibre Channel into an OTU2e. Lines highlighted in yellow are supported by the CS600x.

- **10GFC(TC)->ODU2e**
- **10GFC(TC)->ODU2e->OTU2e**

7.6.16.1 Relevant CS604x Datasheet Sections

Section 2.2.3: Serial 10GFC Client

7.6.16.2 Function Call and Parameters

```
cs_status ten_hl_config_fc_otu2v_t41(cs_uint16 module_id_line,
                                     cs_uint16 slice_line,
                                     cs_uint16 module_id_client,
                                     cs_uint16 slice_client,
                                     cs_uint16 dyn_repro,
                                     cs_uint16 traffic_type_line,
                                     cs_uint16 fec_line,
                                     cs_uint16 traffic_type_client,
                                     cs_uint16 sysclk,
                                     cs_uint16 tcm_line,
                                     cs_uint16 map_oxuv,
                                     cs_uint16 timeslots,
                                     cs_uint16 gmp_timeslot_mask)
```

Parameter description:

```
module_id_line
      Specifies line's module ID

slice_line
      Specifies line's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

module_id_client
      Specifies the client's module ID

slice_client
      Specifies client's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

dyn_repro
      Specifies the type of dynamic reprovisioning:
            0x00 = TEN_INITIAL_CONFIG
            0x01 = TEN_REPRO_CLIENT
            0x02 = TEN_REPRO_LINE_AND_CLIENT

traffic_type_line
      Specify the traffic type of the line:
            0x05 = TEN_TRAFFIC_TYPE_OTU2
            0x06 = TEN_TRAFFIC_TYPE_OTU2E
            0x07 = TEN_TRAFFIC_TYPE_OTU1F
            0x1d = TEN_TRAFFIC_TYPE_OTU1E

fec_line
      Defines fec type
            0x00 = TEN_FEC_MODE_OTUkV (7%)
            0x01 = TEN_FEC_MODE_OTUkV_262 (10%)
            0x02 = TEN_FEC_MODE_OTUkV_273 (15%)
            0x03 = TEN_FEC_MODE_OTUkV_285 (20%)
            0x04 = TEN_FEC_MODE_OTUkV_297 (25%)
            0x05 = TEN_FEC_MODE_OTUkV_5_4 (25%)
            0x06 = TEN_FEC_MODE_OTUkV_301 (26%)
            0x07 = TEN_FEC_MODE_OTUkV_SDH_A (7%)
            0x08 = TEN_FEC_MODE_OTUkV_SDH_B (7 %)
            0x09 = TEN_FEC_MODE_OTUkV_4080_3929 (3.4%)
            0x10 = TEN_FEC_MODE_OTUkV_68_65 (4.2%)
            0x11 = TEN_FEC_MODE_OTUkV_267 (12%)
            0x12 = TEN_FEC_MODE_OTUkV_270 (13%)
            0x13 = TEN_FEC_MODE_GFEC (7%)
            0x14 = TEN_FEC_MODE_ZEROFEC
            0x15 = TEN_FEC_MODE_NOFEC

traffic_type_client
      Specifies the traffic type of the client:
      0x0d = TEN_TRAFFIC_TYPE_10GFC
      0x0e = TEN_TRAFFIC_TYPE_8GFC
      0x0f = TEN_TRAFFIC_TYPE_4GFC

sysclk
      Specifies the sysclk rate in MHz

tcm_line
      Specifies the termination for TCM 1 through 6.  This
      is a bit encoded parameter with bits 1 through 6
      specifying TCM 1 through 6 respectively. These bits
      are defined as follows:
            0 = disable
            1 = enable

map_oxuv
      Specifies the client mapping method
            0x01 = TEN_MAP_BMP
            0x03 = TEN_MAP_AMP_PROP
            0x05 = TEN_MAP_GMP_LO
            0x08 = TEN_MAP_RATE_ADJUST
            0x09 = TEN_MAP_TRANSCODE

gmp_timeslots
      Specifies number of timeslots. Applicable to GMP_HO
      only if [map_proc] = TEN_MAP_GMP_HO. Bandwidth is
      always 1.25G per timeslot.
            1..8

gmp_timeslot_mask
      Specifies the timeslots to use for GMP mapping.
      Applicable to GMP_HO only.
            Bit 0- Timeslot 1 through Bit 7  Timeslot 8
```

7.6.17 10G FibreChannel into an ODTU23

This section removed intentionally.

7.6.18 10G FibreChannel into an OTU2e

Refer to Section 7.6.16.

7.6.19 8G FibreChannel into an ODTU23

This section removed intentionally.

7.6.20 8G FibreChannel into an OTU2

Refer to Section 7.6.16.

7.6.21 Infiniband

NOTE: Infiniband is not currently supported in Release 5.1.

The following mappings are supported for Infiniband into an OTU2.

- **5/10G IB (GMP_LO)->ODU2**
- **5/10G IB (GMP_LO)->ODU2->OTU2**
- **5/10G IB (GMP_LO)->ODU2e**
- **5/10G IB (GMP_LO)->ODU2e->OTU2e**
- **5G IB (BMP)->ODUflex(GMP_HO)->ODTU2.5->ODU2**
- **5G IB (BMP)->ODUflex(GMP_HO)->ODTU2.5->ODU2->OTU2**
- **5G IB (BMP)->ODUflex(GMP_HO)->ODTU2.5->ODU2e**
- **5G IB (BMP)->ODUflex(GMP_HO)->ODTU2.5->ODU2->OTU2e**
- **10G IB (BMP)->ODUflex(GMP_HO)->ODTU2.8->ODU2e**
- **10G IB (BMP)->ODUflex(GMP_HO)->ODTU2.8->ODU2->OTU2e**
- **5/10G IB (BMP)->ODUflex(GMP_HO)->ODTU2.5->ODU2**
- **5/10G IB (BMP)->ODUflex(GMP_HO)->ODTU2.5->ODU2->OTU2**
- **5/10G IB (BMP)->ODUflex(GMP_HO)->ODTU2.5->ODU2e**
- **5/10G IB (BMP)->ODUflex(GMP_HO)->ODTU2.5->ODU2->OTU2e**
- **5/10G IB (COR)->ODU2**
- **5/10G IB (COR)->ODU2->OTU2**
- **5/10G IB (COR)->ODU2e**
- **5/10G IB (COR)->ODU2e->OTU2e**

7.6.21.1 Relevant CS600x Datasheet Sections

Section 2.2.6: Infiniband Clients

7.6.21.2 Function Call and Parameters

```
cs_status ten_hl_config_infiniband_otu2v_t41(cs_uint16 module_id_line,
                                             cs_uint16 slice_line,
                                             cs_uint16 module_id_client,
                                             cs_uint16 slice_client,
                                             cs_uint16 dyn_repro,
                                             cs_uint16 traffic_type_line,
                                             cs_uint16 fec_line,
                                            cs_uint16 traffic_type_client,
                                             cs_uint16 sysclk,
                                             cs_uint16 tcm_line,
                                             cs_uint16 map_oxuv,
                                             cs_uint16 timeslots,
                                             cs_uint16 gmp_timeslot_mask)
```

Parameter description:

```
module_id_line
      Specifies line's module ID

slice_line
      Specifies line's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

module_id_client
      Specifies the client's module ID

slice_client
      Specifies client's slice:
            0x00 = TEN_SLICE0
            0x01 = TEN_SLICE1
            0x02 = TEN_SLICE2
            0x03 = TEN_SLICE3

dyn_repro
      Specifies the type of dynamic reprovisioning:
            0x00 = TEN_INITIAL_CONFIG
            0x01 = TEN_REPRO_CLIENT
            0x02 = TEN_REPRO_LINE_AND_CLIENT

traffic_type_line
      Specify the traffic type of the line:
            0x05 = TEN_TRAFFIC_TYPE_OTU2
            0x06 = TEN_TRAFFIC_TYPE_OTU2E
            0x07 = TEN_TRAFFIC_TYPE_OTU1F
            0x1d = TEN_TRAFFIC_TYPE_OTU1E

fec_line
      Defines fec type
            0x00 = TEN_FEC_MODE_OTUkV (7%)
            0x01 = TEN_FEC_MODE_OTUkV_262 (10%)
            0x02 = TEN_FEC_MODE_OTUkV_273 (15%)
            0x03 = TEN_FEC_MODE_OTUkV_285 (20%)
            0x04 = TEN_FEC_MODE_OTUkV_297 (25%)
            0x05 = TEN_FEC_MODE_OTUkV_5_4 (25%)
            0x06 = TEN_FEC_MODE_OTUkV_301 (26%)
            0x07 = TEN_FEC_MODE_OTUkV_SDH_A (7%)
            0x08 = TEN_FEC_MODE_OTUkV_SDH_B (7 %)
            0x09 = TEN_FEC_MODE_OTUkV_4080_3929 (3.4%)
            0x10 = TEN_FEC_MODE_OTUkV_68_65 (4.2%)
            0x11 = TEN_FEC_MODE_OTUkV_267 (12%)
            0x12 = TEN_FEC_MODE_OTUkV_270 (13%)
            0x13 = TEN_FEC_MODE_GFEC (7%)
            0x14 = TEN_FEC_MODE_ZEROFEC
            0x15 = TEN_FEC_MODE_NOFEC

traffic_type_client
      Specifies the traffic type of the client:
            0x12 = TEN_TRAFFIC_TYPE_5G_IB
            0x21 = TEN_TRAFFIC_TYPE_10_GIB
            0x22 = TEN_TRAFFIC_TYPE_12_5G_IB

sysclk
      Specifies the sysclk rate in MHz

tcm_line
      Specifies the termination for TCM 1 through 6.  This
      is a bit encoded parameter with bits 1 through 6
      specifying TCM 1 through 6 respectively. These bits
      are defined as follows:
            0 = disable
            1 = enable

map_oxuv
      Specifies the client mapping method
            0x01 = TEN_MAP_BMP
            0x03 = TEN_MAP_AMP_PROP
            0x05 = TEN_MAP_GMP_LO
            0x08 = TEN_MAP_RATE_ADJUST
            0x09 = TEN_MAP_TRANSCODE

timeslots
      Specifies number of timeslots. Applicable to GMP_HO
      only if [map_proc] = TEN_MAP_GMP_HO. Bandwidth is
      always 1.25G per timeslot.
            1..8

gmp_timeslot_mask
      Specifies the timeslots to use for GMP mapping.
      Applicable to GMP_HO only.
            Bit 0- Timeslot 1 through Bit 7  Timeslot 8
```

8.0 Interrupt Control and Handling

In a typical board or line-card implementation using the CS604x Transport Processor, it is expected that the interrupt signal(s) from one or more CS604x Transport Processor's, along with interrupt signals from other devices, go to some on-board device controller (CPLD/FPGA). This device-controller will then propagate the interrupt signal to the appropriate (if more than one) interrupt pin on the board CPU.

On the CS604x Transport Processor, the interrupt signal is asserted on the INTN pin (active-low). It is assumed that interrupt signal on this pin is propagated to the CPU. It is also assumed that the customer application (or RTOS) will provide the First-Level Interrupt Handler (FLIH) which is connected to the appropriate interrupt vector of the CPU. The driver provides a Second-Level Interrupt Handler (SLIH) which should be called by the customer application (from their FLIH processing).

The device driver IRQ code works off of the concept of *interrupt nodes*. An interrupt node is a data structure that defines an interrupt register, interrupt enable register, an interrupt status register (if one exists for the node), the node's parent in the hierarchy, all of the children in the hierarchy, the number of slices, and the stride. Optionally, the register and bit-field names can be included in the structures, which can be used for printing enabled interrupts or interrupts that have "fired". The names of interrupt nodes are formed by prefixing the interrupt register name with TEN_IRQ_NODE_. Therefore, the top-level interrupt node in the tree is TEN_IRQ_NODE_MPIF_GLOBAL_INTERRUPT. The interrupt node data structure is in the auto-generated file modules/irq/ten_irq_tree.c.

8.1 Interrupt Control

The driver does not enable or disable interrupts by default – interrupt enables initialize in the default state following device reset, which is "disabled" in GLOBAL_INTENABLE. This gives the application flexibility to enable them at a later time when it is ready to handle interrupts from the device.

The functions to enable or disable interrupts operate on an interrupt node and can perform the action in one of four ways:

Specific interrupt bits at the specified node can be enabled or disabled. This is the best way to selectively disable a leaf interrupt: select the leaf node, the bits to disable, and then use the TEN_IRQ_DIR_ONLY option.

Specific interrupt bits at the specified node can be enabled or disabled, and then the interrupt tree traversed upward toward the GLOBAL_INTERRUPT, enabling or disabling the path to the specified node. This is the best way to fully enable a leaf interrupt: select the leaf node, the bits to enable, and then use the TEN_IRQ_DIR_UP option.

Specific interrupt bits at the specified node can be enabled or disabled, and then the interrupt tree traversed downward toward the leaves, enabling or disabling the path from the specified node. This is the best way to fully enable all of the interrupts for a selected block: select the global or intermediate node, the bits to enable, and then use the TEN_IRQ_DIR_DOWN option. An example of using this approach to enable all interrupts in the device is below.

A combination of the three previous options is also available with the TEN_IRQ_DIR_BOTH option. Select an intermediate node and the path to it from the GLOBAL_INTERRUPT will be enabled, along with all subtending interrupts beneath it down to the leaves.

There are two types of enable/disable functions. The first, ten_dev_irq_enable and ten_dev_irq_disable, operate on the global blocks mpif, glb_misc, spoh, and xcon. These functions are indiscriminate about modules or slices. Enabling interrupts from TEN_IRQ_NODE_MPIF_GLOBAL_INTERRUPT on down the tree would enable interrupts in every slice of all subtending blocks. So, to enable every interrupt in the device you would use:

```
ten_dev_irq_enable(dev_id,
TEN_IRQ_NODE_MPIF_GLOBAL_INTERRUPT,
0xFFFF,
TEN_IRQ_DIR_DOWN);
```

The second type of enable/disable function are the module/slice-aware functions ten_mod_irq_enable and ten_mod_irq_disable. These operate on interrupts in blocks where multiple instances exist (syncdsyncrx, syncdsynctx, gpllx1_sds, xfi32x1_sds, gfec40g, gfec10g, ufec, ohpp, n10g, and pp10g). Interrupts are only enabled or disabled for the module and slice specified, going up or down the tree. To enable the SYNC_LOSTi interrupt in the XCON_ES_INTERRUPT for all slices, place the following API call in a loop and iterate over the module_id and slice:

```
ten_mod_irq_enable(module_id, slice,
TEN_IRQ_NODE_XCON_ES_INTERRUPT,
0x200, /* This is the SYNC_LOSTi interrupt bit */
TEN_IRQ_DIR_UP);
```

The leaf-level interrupts can be enabled as part of device or module initialization but the interrupt signal on INTN pin will only be asserted when the intermediate node(s) and top-level interrupt(s) are enabled. Enabling a leaf interrupt up the tree is an easy way to make sure that the interrupt is enabled all the way to the INTN pin. The previous SYNC_LOSTi interrupt example illustrates how to enable at a leaf and on up the tree, by using the TEN_IRQ_DIR_UP option.

8.2 Interrupt Handling

The CS604x Transport Processor driver provides the Second-Level Interrupt Handler (SLIH) which the application's FLIH can call. This handler's entry point is the following:

```
ten_irq_isr (cs_dev_id_t dev_id);
```

When the CPU detects external device interrupt, the application's FLIH, after de-multiplexing the interrupt and determining that it is one of the CS604x Transport Processor devices being the source of the interrupt, should then call the above driver handler.

In the CS604x Transport Processor, there is one interrupt pin (INTN) on the chip which collects all the top-level pending interrupts from various blocks and propagates to the board CPU. Internally, there are intermediate interrupt enable control and status bits for the many sub-blocks within each block, and similarly in the sub-sub-blocks, etc. That is, there is an interrupt hierarchy tree where at the leaf-level is the actual interrupt source bits which can be cleared. All the intermediate interrupt bits are just status bits propagating the interrupt signal from the leaf interrupt sources. The purpose of the intermediate interrupt enable bits is to determine whether interrupt signals from below in the hierarchy tree get propagated further or not.

Each node in the interrupt hierarchy can have an associated user-supplied interrupt handler that will be called whenever there is an active and enabled interrupt at that node. Active but disabled interrupts will be ignored. The handler will be called by the interrupt service routine whenever one or more interrupt bits at that node are ‘1’ and the corresponding enable bit is also ‘1’. Only one handler can be registered at each node. The handler is registered with the following call:

```
ten_irq_register_handler(cs_uint16 dev_id, ten_irq_node_ptr
node, void *handler);
```

To register a handler for the XCON_ES_INTERRUPT:

```
ten_irq_register_handler(dev_id,
&TEN_IRQ_NODE_XCON_ES_INTERRUPT,
&handler);
```

The handler would be of the form:

```
void handler(ten_irq_handler_data_t data)
```

When the driver’s ISR is called, it will identify all the valid and pending interrupt sources in each of the blocks. It does this by traversing the interrupt hierarchy for the various blocks and sub-blocks in the chip for which the appropriate interrupts are enabled. On finding each valid interrupt source at any level of the interrupt hierarchy, the driver clears the interrupt source if the interrupt is a leaf. It then calls the registered interrupt handler for that node, if any has been registered. Handlers are passed a structure with information on the interrupt:

```
typedef struct ten_irq_handler_data_s {
    cs_uint16                dev_id;
    ten_irq_node_ptr         node;
    cs_uint16                ireg_data;
    cs_uint16                ereg_data;
    cs_uint16                sreg_data;
    cs_uint8                 slice;
} ten_irq_handler_data_t;
```

ireg_data, ereg_data, and sreg_data will be the current values of the interrupt register, enable register, and status register respectively, before the active and enabled interrupts were cleared. Not all nodes have status registers; for those that do not the sreg_data field will be 0.

The value of *node* can be used to retrieve additional information about the interrupt node, such as the addresses of the interrupt, enable, and status registers, the number of slices and the stride. If TEN_IRQ_PRINT_INTERRUPTS has been enabled then a string representation of the node and each of the interrupt bits is available.

```
typedef struct ten_irq_node_s {
    cs_uint32                ireg;
    cs_uint32                ereg;
    cs_uint32                sreg;
    cs_uint16                 slices;
    cs_uint16                 stride;
    cs_uint8                  num_children;
    cs_uint16                 mask;
    ten_irq_node_children_t *children;
    ten_irq_node_parents_t  *parents;
    ten_irq_handler_t         handler;
#if TEN_IRQ_PRINT_INTERRUPTS
    char                     *name;
    cs_uint8                  last_child;
    char                     *child_name[];
#endif
} ten_irq_node_t;
```

Before calling the registered interrupt handler, the driver will clear any leaf-level interrupt bits in the source register. Note that the clearing of the interrupt source is for this instance of the interrupt event only. If the interrupt condition is persistent (level-triggered) then the interrupt could be detected again immediately upon returning from the interrupt processing and when the top-level interrupt to the CPU is re-enabled.

8.3 Interrupt Integration

The interrupt mechanism in general is very useful, as the application can detect certain events and appropriately take user-defined actions in order to respond to the interrupt events. User-defined interrupt handlers have register granularity allowing for very structured interrupt handling code.

The chip and the driver do not support prioritization of the interrupts. The driver's interrupt handler (SLIH) when invoked will walk the tree recursively working from the least significant bit to the most significant. For example, starting at the GLOBAL_INTERRUPT, it will follow the branch from bit 0 (if bit 0 is enabled and active) before advancing to bit 1. It does not follow branches that do not have enabled and active interrupts.

The SLIH can be polled or called as part of an interrupt service routine. The semaphore TEN_ID_IRQ is used to protect the interrupt enabled read-modify-write operation and the interrupt clearing mechanism.

It is expected of the application to have some mechanism to be able to throttle the number and frequency of interrupts from the devices on the board. During initial board bring-up and debug, this could be very useful to have. If there is no throttling and if there is some interrupt that is persistently detected then it could starve the RTOS processes of the CPU resource.

8.3.1 Example Interrupt Code

The driver does not implement any required interrupt routines because each system's interrupt mechanism is different. The driver is tested with a polled interrupt scheme and some interrupt handlers have been implemented to test specific software work-arounds, like elastic store re-centering upon fiber pull.

The actual implementation of an interrupt service routine depends on whether the system supports multiple processes or threads, a hardware-driven interrupt or polling, and message queues. Some systems will insert a message into a message queue when an interrupt has been received so that the interrupt handler does not run in the interrupt context.

The example driver interrupt handler is implemented as a polling tread, however this functionality is disabled in the released code because threads may not be available in all customers' systems. This thread is created in ten_drvr_load().The thread's background routine is irq_polling_thread() and it consists of calling ten_irq_isr() and then going to sleep for a period of time. ten_irq_isr() is the interrupt service routine and is responsible for walking the interrupt tree and calling registered interrupt handlers for active enabled interrupts. The interrupt service routine will clear an interrupt after the interrupt handler has been called.

8.4 Special Considerations

8.4.1 Handling the UFEC Interrupt Differences

This section only applies to the CS600x; it does not apply to the CS604x.

Interrupts are generated in the leaf nodes and, if enabled, propagate up through the intermediate nodes to the top of the interrupt tree. For most blocks, clearing the interrupt at the leaf node clears the intermediate nodes if there are no other enabled and active interrupts below the interrupt nodes.

Interrupts are handled differently in the UFEC block. Clearing an interrupt in the leaf node does not clear the interrupt in the intermediate nodes – this must be done manually. SW Release 2.2 introduced code to automatically walk up the tree and clear the intermediate nodes. The code is in ten_irq_walk_node which is called by ten_irq_isr and ten_irq_walk_tree.

8.5 Handling errors and defects

8.5.1 dLOS

8.5.1.1 Handling External LOS

This section only applies to the CS600x; it does not apply to the CS604x.

From document *CS600x Transport Processor Sightings Report, 400989, Revision 4*: "The filter circuits on the external loss-of-signal inputs do not exhibit the expected behavior. An active external loss-of-signal input may cause an immediate activation of the filter output and activation of the MPIF_XLOS_INTERRUPT. The hardware enabled internal protection clock switching would also be immediate." The internal filter on the EXT_LOS inputs should not be used. There are three workarounds identified.

8.6 Fiber Pull Recovery

A fiber pull recovery mechanism should be implemented that re-centers the elastic stores following removal of the LOS state. This recovery mechanism could be interrupt driven. An interrupt service routive API will facilitate this development and is described below.

A pair of interrupt driven recovery sequence examples have been provided that will, based on the selected interrupts, reset the CS600x/CS604x datapath when the interrupts assert and associated status bits indicate a recovery sequence is warranted. Separate threads from the main execution thread are required to monitor these interrupts for transitions. The example code consists of the ten_lof_recovery_error_handler_enable and ten_pcs_recovery_error_handler_enable APIs which launch threads to poll the interrupts and call the ten_hl_config_perform_recovery_sequence API to execute the recovery sequence on interrupt assertion. ten_mpif_n10g_rx_is_active and ten_mpif_pp10g_rx_is_active are helper APIs that determine if applicable circuit mode and packet mode configurations have been provisioned.

For a circuit mode configuration, LOF is the example interrupt used to trigger the recovery sequence. The ten_lof_recovery_error_handler_enable API verifies that the n10g is out of reset and the OTN block is enabled. If so, the API launches a new thread to poll the N10G_OTNR_INTR.ILOF interrupt and launch the recovery sequence API on assertion.

For a packet mode configuration, PCS sync is the example interrupt used to trigger the recovery sequence. The ten_pcs_recovery_error_handler_enable API verifies that the pp10g is out of reset and the PCS block is enabled. If so, the API launches a new thread to poll the PP10G_PCS_RX_RXINT.syncdetI interrupt and launch the recovery sequence API on assertion.

The recovery sequence is implemented in the ten_hl_config_perform_recovery_sequence API. This API is hardwired to the provided circuit mode and packet mode interrupt examples, ILOF and syncdetI. Should different interrupts be desired the API should be changed to recover based on those interrupts. Because the example interrupts assert on both edges of their associated status bits, and a recovery sequence is only warranted when the datapath is going back into service, the API should determine if the interrupts indicate a rising edge or falling edge of the associated status bits.

For example, for the circuit mode configuration, when the API is called based on an assertion of the ILOF interrupt, the API looks at the NX0G_OTNR_OFSTAT.SLOF bit. If the SLOF bit is asserted, which indicates the ILOF interrupt is signaling that the framer has just lost its in-frame state, the ten_hl_config_perform_recovery_sequence API exits because no recovery sequence is warranted when the framer is out-of-frame. However, if the SLOF bit is deasserted when the API is called, then the ILOF interrupt is signaling that the framer has just regained its in-frame status and the recovery sequence executes. The ten_hl_config_perform_recovery_sequence API behaves similarly for the packet mode example based on the syncdetI interrupt and the PP10G_PCS_RX_RXSTATUS.syncdetS status bit.

The interrupt thread described above is disabled by default so that it won’t conflict with customer threads or interrupt routines.

8.7 Elastic Store Amplitude Recentering

The CS604x has a new method for elastic store recentering called “Amplitude Recentering”. This feature requires some support from software whenever an OTN has gone into frame.

Amplitude recentering uses measurements that must be taken with stable framed traffic. The CS604x makes these measurements in real-time when recenter is triggered. If amplitude recentering is triggered with no traffic or unstable traffic, results are not predictable. Thus, software must intervene when traffic becomes stable again.

A system is defined which includes a definition of “traffic-stable” events and new functions to set up and recenter as those events are received. To provide the greatest flexibility in defining and reacting to these events, the 5.4 release provides the setup tools and the handlers, which must be called as desired by application code.

8.7.1 Overview of Amplitude Recentering

There are two types of amplitude recentering. Both apply to 10 gigabit and 40 gigabit traffic.

- Measured amplitude recentering is based on the measured amplitude of the variation in elastic store levels over a short duration.
- Frame Pulse Alignment can be used to recentering configures elastic store according to the anticipated amplitude, and then coerces a deterministic frame alignment by recentering at a precise moment in the received frame.

Because amplitude recentering can only be initiated on stable framed traffic, it requires three steps over a sequence of time for configuration, recentering, and follow-up.

The CS604x C API supports full access to CS604x recentering configurations and parameters. There are three sets of APIs.

- Configure-and-recenter: This set provides a discrete function for each of 10G/40G and each of the recenter types, for a total of four. Each function configures, initiates, and completes amplitude recentering. This was available in CS604x API release 5.3.
- Configure-and-wait: A consolidated set of flexible API is used to configure the driver and CS604x. To support autonomous amplitude recentering, event handlers are provided to trigger and complete the recenter process. This was available in CS604x API release 5.4. This set supports
    - Offline configuration.
    - Autonomous recenter when framing is detected.
    - Update of ephemeral circuit threshold registers after FPA recentering is complete.
- Simplified hybrid (release 5.5): A consolidated API function is used to configure the driver and CS604x, and a new API function triggers and completes the recenter process on demand.

.

The release 5.3 API provides separate functions for 10G and 40G AMP recentering, requiring arguments for upper and lower margin, upper and lower correction (a ‘pinching’ mechanism), measurement duration, and system clock frequency. For recentering on frame pulse alignment, the API provides two more functions for 10G and 40G, requiring arguments for static depth, and upper and lower circuit thresholds.

The release 5.4 API provides separate functions for setup and for event handling. Setup uses one function to write the recenter parameters (as described for 5.3) to a new XCON Elastic Store control block, and another to configure the device using the Control Block data. The remaining functions are for event handling.

8.7.2 Recentering Based on Amplitude Measurement Period

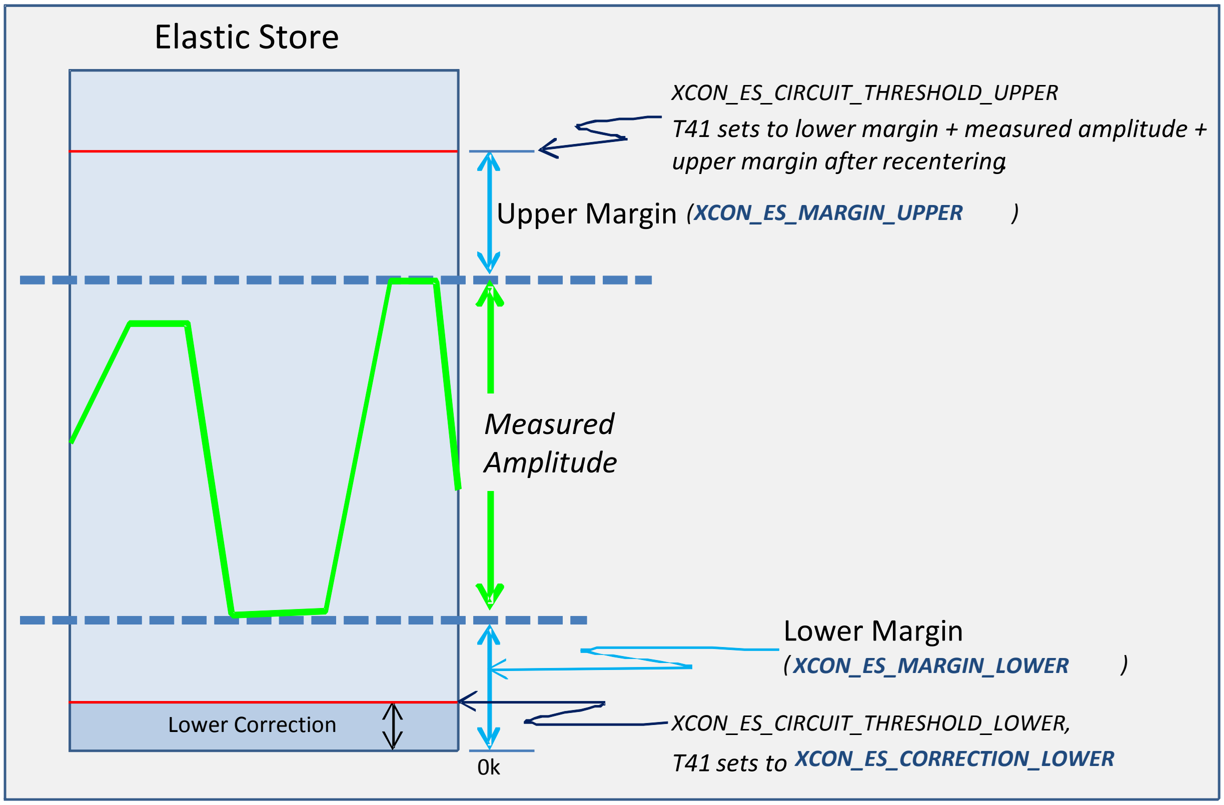
Three different recentering margins are recommended:

- Wide: This is an accommodating choice for normal configurations where amplitude centering makes sense. Use this when a) it is not certain what the amplitude measurements for elastic store requirements will work out to, or b) it is known that a wide margin is necessary. The threshold settings are very conservative (see table below). The measurement duration is longer than the others because of the wide qualifier's intent for uncertain ES characteristics.
- Typical: Use this for configurations known to have lesser elastic store requirements, e.g., for typical FEC wrap/dewrap.
- Narrow: use this for configurations known to have minimal elastic store requirements.

The following table describes how these threshold levels are mapped to API arguments. These are subject to change.

Margin range	lower margin	lower correction	upper margin	upper correction	duration
wide	90	15	100	0	75ms
typical	10	5	15	0	10ms
narrow	5	2	10	0	5ms

After the measurement period, the CS604x sets XCON_ES_CIRCUIT_THRESHOLD_LOWER to “lower margin” plus “lower margin correction.” It sets XCON_ES_CIRCUIT_THRESHOLD_UPPER to “lower margin” plus measured amplitude plus “upper margin” minus “upper margin correction”. Margin-exceeded interrupts are triggered when the amount of data in the ES goes below XCON_ES_CIRCUIT_THRESHOLD_LOWER. However, data transfer is not affected until there is no data in the ES to clock out.


Elastic Store
XCON_ES_CIRCUIT_THRESHOLD_UPPER
T41 sets to lower margin + measured amplitude + upper margin after recentering
Upper Margin (XCON_ES_MARGIN_UPPER)
Measured Amplitude
Lower Margin
(XCON_ES_MARGIN_LOWER)
Lower Correction
XCON_ES_CIRCUIT_THRESHOLD_LOWER,
T41 sets to XCON_ES_CORRECTION_LOWER
0k

Parameters supplied to the API to configure the CS604x are elastic store number (for 10G configurations) or module number, XCON_ES_MARGIN_LOWER/ _UPPER, XCON_ES_CORRECTION_LOWER/_UPPER (upper is usually 0), duration of the measurement period in milliseconds, and the system clock rate in MHz. The API used the system clock rate to convert the measurement duration in milliseconds to units of 1024 system clock tics.

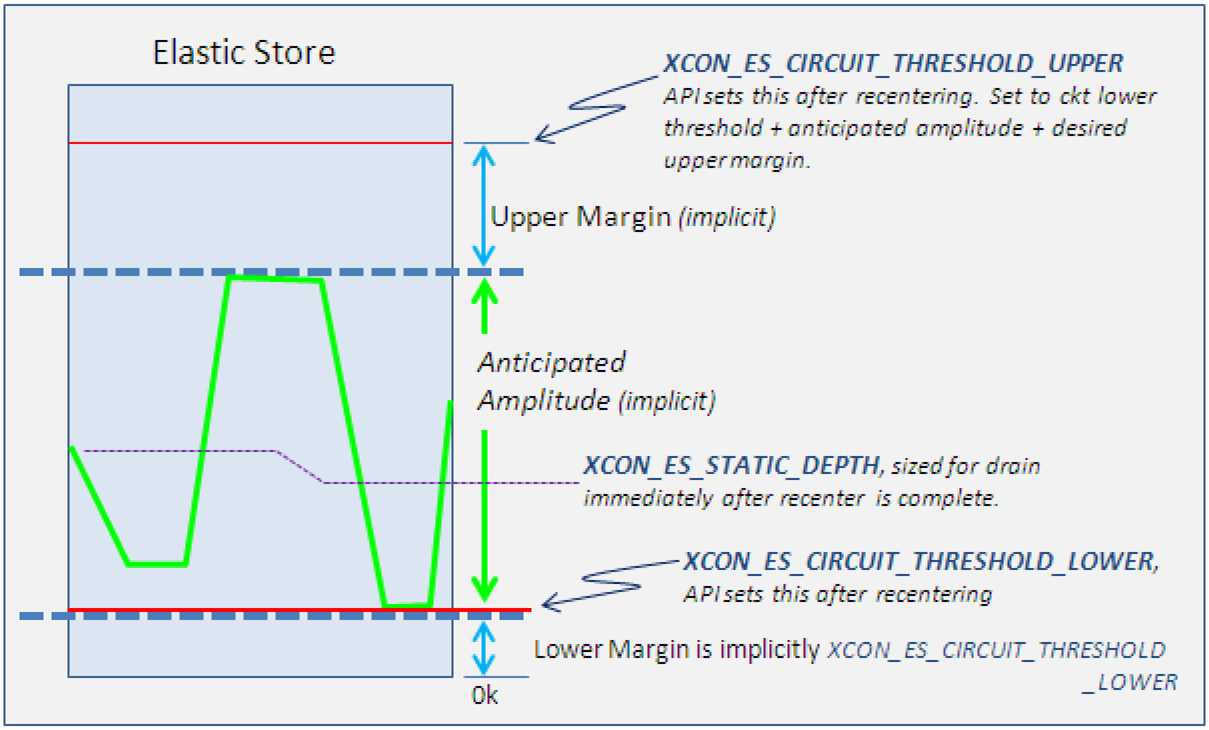
8.7.3 Recentering Based on Deterministic Amplitude Using Frame Pulse Alignment

Three different recentering margins are recommended:

- Wide: This is intended for normal configurations where amplitude centering makes sense. Use this when a) it is not certain what the amplitude measurements for elastic store requirements will work out to, or b) it is known that a wide margin is necessary. The threshold settings are very conservative (see table). The lower circuit threshold is higher than the others because this configuration can be used when ES characteristics are uncertain.
- Typical: Use this qualifier for configurations known to have lesser elastic store requirements, e.g., for typical FEC wrap/dewrap.
- Narrow: use this for configurations known to have minimal elastic store requirements.

The following table describes how these threshold levels are mapped to API arguments. These are subject to change.

Margin range	static depth	lower circuit threshold	upper circuit threshold
wide	150	35	260
typical	75	10	160
narrow	35	2	100


Elastic Store
XCON_ES_CIRCUIT_THRESHOLD_UPPER
API sets this after recentering. Set to ckt lower threshold + anticipated amplitude + desired upper margin.
Upper Margin (implicit)
Anticipated Amplitude (implicit)
XCON_ES_STATIC_DEPTH, sized for drain immediately after recenter is complete.
XCON_ES_CIRCUIT_THRESHOLD_LOWER,
API sets this after recentering
Lower Margin is implicitly XCON_ES_CIRCUIT_THRESHOLD _LOWER
0k

Parameters supplied to the API to configure the CS604x are elastic store number (for 10G configurations) or module number, XCON_ES_STATIC_DEPTH (the lower margin plus amount of elastic store drain after recenter) and XCON_ES_-CIRCUIT_THRESHOLD_LOWER/_UPPER. The circuit thresholds are configured after recenter is complete.

8.7.4 Triggering Amplitude Recentering

8.7.4.1 Configure-and-Recenter API

Configure the CS604x for the desired traffic type.

Set up the driver and the device for amplitude recentering and trigger the recenter using one of the following. Note that these functions polls interrupt registers to determine when recenter is complete.

- ten_hl_xcon_es_amplitude_recenter_t41() for an individual ES.
- ten_hl_xcon_es_amplitude_recenter_with_fpa_t41()
- ten_hl_xcon_es_amplitude_recenter_40g_t41() for a 40G traffic config.
- ten_hl_xcon_es_amplitude_recenter_with_fpa_40g_t41()

8.7.4.2 Configure-and-Wait API

Configure the CS604x for the desired traffic type.

Set up the driver and configure the CS604x for amplitude recentering. Call:

- ten_xcon_es_write_cb_t41()
- ten_xcon_es_setup_amp_recntr_from_cb_10g_t41() for an individual ES or
- ten_xcon_es_setup_amp_recntr_from_cb_40g_t41() for a 40G traffic config.

Set up application code to monitor for events indicating that traffic has become stable. These events should be raised after system clocks are stable and traffic has begun to flow—e.g., for OTU2 traffic, when the frame indicates it has framed.

When traffic has become stable after setup or recovery, call one of the following to trigger recenter. These functions rely on mutex, so run them on an OS thread, not on a processor interrupt thread.

- ten_xcon_es_handle_traffic_stable_event_10g_t41() for a 10G stream,
- ten_xcon_es_handle_traffic_stable_event_40g_t41() for a 40G traffic config.

Set up application code to monitor for recenter complete events (see registers XCON_ES_INTERRUPT and XCON_ES_INTSTATUS).

When recenter has completed, call the following function to complete follow-up work. This function requires mutual exclusion, so it should be run on an OS thread, not on a processor interrupt thread.

- ten_xcon_es_handle_recenter_complete_event_t41().

New for release 5.5: Recentering can be initiated by application code for any event, such as on elastic store threshold violations, by using new API function ten_hl_xcon_es_amplitude_recenter_from_cb_t41().

8.7.5 Measured Amplitude Recentering High level APIs

These were developed and delivered in CS604x release 5.3. There is one high-level API each for 40G and 10G traffic. These are shown below.

```
cs_status ten_hl_xcon_es_amplitude_recenter_t41(
                              cs_uint16 dev_id,
                              ten_xcon_es_num_t es_num,
                              cs_uint16 margin_lower,
                              cs_uint16 margin_upper,
                              cs_uint16 correct_lower,
                              cs_uint16 correct_upper,
                              cs_uint8  duration,
                              cs_uint32 sysclk_freq);

cs_status ten_hl_xcon_es_amplitude_recenter_40g_t41(
                              cs_uint16 dev_id,
                              cs_uint8  module_num,
                              cs_uint16 margin_lower,
                              cs_uint16 margin_upper,
                              cs_uint16 correct_lower,
                              cs_uint16 correct_upper,
                              cs_uint8  duration,
                              cs_uint32 sysclk_freq);
```

8.7.6 Recentering with Frame Pulse Alignment High level APIs

These were developed and delivered in release 5.3. There is one high-level API each for 40G and 10G traffic.

```
cs_status ten_hl_xcon_es_amplitude_recenter_with_fpa_t41(
                              cs_uint16 dev_id,
                              ten_xcon_es_num_t es_num,
                              cs_uint16 static_depth,
                              cs_uint16 circuit_th_lower,
                              cs_uint16 circuit_th_upper);

cs_status
ten_hl_xcon_es_amplitude_recenter_with_fpa_40g_t41(
                              cs_uint16 dev_id,
                              cs_uint8  module_num,
                              cs_uint16 static_depth,
                              cs_uint16 circuit_th_lower,
                              cs_uint16 circuit_th_upper);
```

8.7.7 Amplitude Recentering after Signal Recovery

It is the hosting system’s responsibility to initiate amplitude recentering after recovery, or after traffic is running stably for initial configurations. To support this, the CS604x API provides a combination of high level APIs ad traffic status ISRs, and retains configuration data. These can be used to configure, trigger, and finish amplitude recentering. Please refer to 8.7.4.2 “Configure-and-Wait API” above for more information on how to use this.

8.7.8 Amplitude Recentering after Other Events

Beginning with Revision 5.5, once amplitude recentering has been configured, amplitude recenter APIs can trigger and complete recenter with a single API function call. This simplifies recovery and reaction to other events, such as upper/lower elastic store threshold violations.

The function uses amplitude recenter parameters which have been stored in the XCON elastic store control block.

- cs_status ten_hl_xcon_es_amplitude_recenter_from_cb_t41(dev_id, es_num);

Notice: For this function to finish the recentering operation, amplitude recenter configuration parameters must be overloaded to configure the driver with the desired recenter and completion behavior when recovery events are handled. These are

- Default: This is 0 (zero) for backward compatibility. It is mapped to 'none' for the configure-and-recenter APIs, 'interrupt driven' for configure-and-wait APIs.
- Auto-complete is new, to simplify recovery. Use this from recovery event handlers to recenter from trigger to completion.
- Interrupt driven: This is the default behavior for configure-and-wait APIs. Each step of recentering must be triggered by an event handler.
- None: This is the default behavior for the configure-and-recenter APIs. Event handlers exit without action.

The enumerations for these recovery options are:

```
TEN_XCON_ES_RECENTER_RECOVERY_DFLT = 0x00,
TEN_XCON_ES_RECENTER_RECOVERY_AUTO = 0x40,
TEN_XCON_ES_RECENTER_RECOVERY_INTP = 0x80,
TEN_XCON_ES_RECENTER_RECOVERY_NONE = 0xC0,
```

To configure the driver for auto-completion on recovery, the recovery argument is combined with other arguments. For configure-and-recenter API's, this is the Elastic Store or module number. For example:

ten_hl_xcon_es_amplitude_recenter_t41(dev_id, es_num | 0x40, …)

For configure-and-wait APIs, the recenter type is overloaded. For example:

ten_xcon_es_write_cb_t41(dev_id, es_num, (0x40<<8) | type, …).

8.7.9 Checking Configuration and Operation

All amplitude recentering parameters are stored in the Device Control Block. These can be printed by using the API command ten_dev_dump_cb(). To avoid clutter, Elastic Store blocks not configured for amplitude recentering are not displayed.

Operation of the amplitude recenter configuration can be checked by viewing minimum and maximum elastic store levels. The minimum and maximum capture registers can be reset and displayed using the following API functions.

ten_xcon_es_reset_level_min_max()
ten_xcon_es_dump_level_min_max()

These will reflect normal values when stable traffic is running and the recenter has been properly configured. To ensure the recorded minimum and maximums are accurate, there should be a small delay between recenter complete and min/max reset, and between reset and min/max dump.

8.7.10 Interrupt-Driven Recentering

Some APIs that enable interrupts and register a handler and the interrupt handler itself have been written. The main focus of the handlers is to re-center the elastic store upon the out of frame to in frame transition. This out of frame to in frame transition can be caused by the initial flow of traffic after provisioning or after a fiber reconnect. The APIs to enable interrupts and register a handler are called from the EVB lab script environment (full.pl) before the CS604x is brought out of reset.

Instead of having a polling thread, the same functionality could be achieved by calling ten_irq_isr() when the CS604x interrupt occurs. In this case, the part of the interrupt handler that does writes to CS604x registers that may be accessed and locked by other threads that can be interrupted should be done in a deferred procedure to prevent lock conflicts.

The following are example interrupt enable APIs.

- ten_n40g_otnr4x_intr_handler_enable() for OTU3v in frame transition:

This API can enable the IIF interrupt in the N40G_OTNR4X_INTR register and registers the interrupt handler ten_n40g_otnr4x_intr_handler(). This interrupt is activated when the SIF bit in the N40G_OTNR4X_OFSTAT register changes . ten_n40g_otnr4x_intr_handler() will call an elastic store re-centering API when a SIF = 0 to SIF = 1 transition occurs.

- ten_n40g_sdfr_sdfist_handler_enable() for OC768 in frame transition:

This API can enable the OOF interrupt in the N40G_SDFR_SDFIST register and registers the interrupt handler ten_n40g_sdfr_sdfist_handler(). This interrupt is activated when the OOFS bit in the N40G_SDFR_SDFSTAT register changes. ten_n40g_sdfr_sdfist_handler() will call an elastic store re-centering API when a OOFS=1 to OOFS=0 transition occurs.

- ten_n10g_otnr_intr_handler_enable() for OTU2v in frame transition:

This API can enable the ILOF interrupt in the N10G_OTNR_INTR register and register the interrupt handler ten_n10g_otnr_intr_handler(). This interrupt is activated when the SLOF bit in the N10G_OTNR_OFSTAT register changes. ten_n10g_otnr_intr_handler() will call an elastic store re-centering API when a SLOF=1 to SLOF=0 transition occurs.

- ten_n10g_sdfr_sdfist_handler_enable() for OC192 in frame transition:

This API can enable the LOF interrupt in the N10G_SDFR_SDFIST register and registers the interrupt handler ten_10g_sdfr_sdfist_handler(). This interrupt is activated when the LOFS bit in the N10G_SDFR_SDFSTAT register changes. ten_10g_sdfr_sdfist_handler() will call an elastic store re-centering API when a LOFS=1 to LOFS=0 transition occurs.

- ten_pp10g_pcs_rx_rxint_handler_enable() for 10GE and 10GFC in sync transition:

This API can enable the SYNCDETI interrupt in the PP10G_PCS_RX_RXINT register and registers the interrupt handler ten_pp10g_pcs_rx_rxint_handler(). This interrupt is activated when the syncdetS bit in the PP10G_PCS_RX_RXSTATUS register changes. ten_pp10g_pcs_rx_rxint_handler() will call an elastic store re-centering API when a syncdetS=0 to syncdetS=1 transition occurs.

- ten_pp40g_pcs_rx_interrupt_handler_enable() for 40GE in sync transition:

This API can enable the XDCBLOCKLOCKI interrupt in the PP40G_PCS_RX_INTERRUPT register and registers the interrupt handler ten_pp40g_pcs_rx_interupt_handler(). This interrupt is activated when the xdcblocklockS bit of the PP40G_PCS_RX_INTSTATUS register changes. ten_pp40g_pcs_rx_interupt_handler() will call an elastic store re-centering API when a xdcblocklockS=0 to xdcblocklockS=1 transition occurs.

9.0 Configuring the Device for Traffic

This section describes the tasks required to configure the CS604x for traffic in a relatively generic way. We'll use a typical transponder configuration to demonstrate the concepts, and show examples where required for additional illustration. Appendix B shows the sample script of API calls referenced in this section.

The sample script will provision the device for aggregation with the following features:

- The line can be assigned to either side A or B, as can the client.
- Not all interfaces are supported by the driver, and not all sides can be used with all interfaces. See the HSIF interface table below.
- 100 MHz, 400 MHz, or 425 MHz system reference clock

9.1 Hard Reset the Device

The device should be hard reset by bringing the RSTN pin to a logic low. While the device is in hard reset, the microprocessor bus mode should be selected via the MBMODE pin.

9.2 System Reference Clock and Internal Clock Frequencies

The SYS_REFCLK_IN pins must have a valid clock for the microprocessor interface to function. Refer to Section 2.14.1 of the CS600x datasheet, select an appropriate SYSREFCLK frequency and set the MBCLKSEL pin appropriately. Examples are below.

Table 12 SYSREFCLK versus MBCLKSEL

SYS_REFCLK_IN Frequency	MBCLKSEL value	Internal SYSREFCLK Frequency
100 MHz	0	100 MHz
400 MHz	1	100 MHz
425 MHz	1	106.25 MHz

9.3 Remove Hard Reset

Once the SYS_REFCLK_IN pins are running at the appropriate rate and MBCLKSEL and MBMODE are correct, remove the hard reset of the CS604x device by bringing pin RSTN to logic high. It is now possible to access the CS604x TEN_MPIF_CHIP_ID_LSB and TEN_MPIF_CHIP_ID_MSB registers to confirm that the microprocessor interface is functional.

9.4 Initialize the Device Driver

The Device Driver requires that the JTAG ID in TEN_MPIF_CHIP_ID_LSB and TEN_MPIF_CHIP_ID_MSB be accessible, so the preceding steps must have been accomplished successfully.

The Device Driver needs memory allocated for a device data structure. The size of this structure is the number of chips being controlled.

Load and register the device driver. Loading is done once regardless of the number of devices being controlled. The “dev_id” parameter is used to distinguish between the different CS604x devices being controlled. “dev_id” should be ‘0’ for the first device, ‘1’ for the second, and so on. Each device must be registered, so dev_id may need to be iterated.

```
dev_id = 0;
ten_dev_register(dev_id, (cs_uint32)&Chips[dev_id]);
ten_dev_main_init( dev_id );
ten_mod_main_init( dev_id, TEN_MODULE_A );
ten_mod_main_init( dev_id, TEN_MODULE_B );
```

9.5 Global Configuration

The global configuration function ten_hl_config_global performs the following steps that are common to all configurations of the device.

- Calls ten_hl_mpif_wait_for_bist_done which waits for Built-In Self-Test (BIST) completion. BIST is a process internal to the CS604x devices that performs diagnostics of the embedded memories. It runs automatically when hardware reset is de-asserted. BIST must be complete before soft-resets are de-asserted and the device is provisioned. If BIST never completes or reports a failure then something is wrong with the device or the board and the software should report a failure.
- Calls ten_mpif_global_reset_common to removes CHIP and MPIF resets.
- Calls ten_mpif_global_reset_syspll to remove the System PLL reset.
- Calls ten_hl_gpllx1_config_SYSGPLL to configure the System PLL.
- Calls ten_hl_gpllx1_check_lock to check PLL lock status.
- Provisions the GPLL clocking multiplexor.
- Configures receive and transmit GPLL reference and pilot clocks to come from the receive SERDES.
- Calls ten_mpif_clock_select_40g to set the 40G clocking multiplexor as appropriate for the configuration.
- Calls ten_mpif_global_cnfg to set the quad-mode multiplexor as appropriate for the configuration.
- Calls ten_mpif_set_lvds_div_pd to enable the LVDS pads for DIV clocks.
- Calls ten_hl_hsif_sfi42_reset_fix to apply the SFI-4.2 reset work-around.

The prototype for this function is:

```
ten_hl_config_global( dev_id, a_mod_mode, b_mod_mode,
mdclksel, mr_mode_a, mr_mode_b )
```

a_mod_mode and b_mod_mode specify the N40G bypass enable for side A and side B and are specified as:

0 = TEN_GLOBAL_MODE_S_40G for N40G module enabled
1 = TEN_GLOBAL_MODE_QUAD_10G for four N10G modules enabled

mdclksel specifies the system clock reference frequency select pin state and is specified as:

0 for system reference frequency <= 200MHz
1 for system reference frequency > 200MHz

mr_mode_a and mr_mode_b specify the multi-rate mode for side A and side B and are specified as:

0 if the port's Multi-Rate mode is SFI-4.1
1 if the port's Multi-Rate mode is not SFI-4.1 or Multi-Rate mode is disabled (e.g. for XFI)

An example of the mr_mode_a and mr_mode_b parameters for different configurations is below.

Table 13 ten_hl_config_global Parameters

Configuration	mr_mode_a	mr_mode_b
Muxponder	TEN_GLOBAL_MODE_S_40G	TEN_GLOBAL_MODE_QUAD_10G
40G Transponder	TEN_GLOBAL_MODE_S_40G	TEN_GLOBAL_MODE_S_40G
10G Transponder	TEN_GLOBAL_MODE_QUAD_10G	TEN_GLOBAL_MODE_QUAD_10G
Side B only	TEN_GLOBAL_MODE_QUAD_10G	TEN_GLOBAL_MODE_QUAD_10G

9.5.1 ten_hl_config_global examples

If sys_refclk is 400 MHz or 425 MHz, and Side A is an overclocked OTU3 on the SFI-4.2 interface, and Side B is four 10G clients, then the function call would be:

```
ten_hl_config_global( dev_id, TEN_GLOBAL_MODE_S_40G,
TEN_GLOBAL_MODE_QUAD_10G, 1, 1, 1 );
```

Using a sys_refclk of 100 MHz the function call would be:

```
ten_hl_config_global ( dev_id, TEN_GLOBAL_MODE_S_40G,
TEN_GLOBAL_MODE_QUAD_10G, 0, 1, 1 );
```

With Side A using the SFI-5.1 interface, and a sys_refclk of 400 MHz the function call would be:

```
ten_hl_config_global ( dev_id, TEN_GLOBAL_MODE_S_40G,
TEN_GLOBAL_MODE_QUAD_10G, 1, 1, 1);
```

A 40G transponder application with a 400 MHz or 425 MHz sys_refclk and both Side A and Side B using the SFI-5.1 interfaces would use:

```
ten_hl_config_global ( dev_id, TEN_GLOBAL_MODE_S_40G,
TEN_GLOBAL_MODE_S_40G, 1, 1, 1 );
```

A 4x10G transponder application using SFI-4.2 on Side A and XFI on Side B at a sys_refclk of 400 or 425 MHz would use:

```
ten_hl_config_global ( dev_id,
TEN_GLOBAL_MODE_QUAD_10G, TEN_GLOBAL_MODE_QUAD_10G,
1, 1, 1 );
```

9.6 Initialize OHPP and Shadow Ram

Run ten_hl_ohpp_and_shadow_ram_init().

9.7 Set First-Level Clock Muxing

Knowing the desired line and client traffic modes, the user will call:

ten_mpif_clock_select_gpll_in()

to set the first-level clock muxing. The arguments chosen for the function will depend on the mode and client/line identity.

9.8 Configure the Fractional Dividers

See the Fractional Dividers explanation in this document.

9.9 Configure the SiLabs

The SiLabs part is not made or supported by Cortina, so it will not be detailed here. However, if one is being used, it should be configured at this point. See Appendix E for additional information.

9.10 Determine the Sync/Desync parameters

See the Sync/Desync section in this document.

9.11 Configure the Line-Side High-Speed Interfaces (HSIF)

The next step is to provision the HSIF interfaces for the configuration.

Before provisioning the HSIF for the specific interface, the user should first override the default datapath state (with all HSIF ports disabled). Use the low-level function ten_hsif_provision_datapath() to do this. This must be called for each side that is being provisioned.

If the configuration is using aggregation, the user will also need to set the 40G clock using the appropriate aggregation parameter.

A high-level API can then be used to configure the specific HSIF interface type.

Summary of APIs to call (this should be done for both line and client):

- ten_hsif_provision_datapath()
- If aggregation call → ten_hsif_set_clk_40g()
- Now run the desired high-level HSIF API (see below).
- If the HSIF type is 40G, then call tb_ical($mod). Otherwise, call tb_ical($mod, $slice). (See Appendix E, section b for an explanation of this).

The table below shows the available high-level HSIF APIs. See the HSIF interface section for a description of these APIs and their parameters.

Call only those functions that are relevant for the configuration. For example, if the platform isn't using the 10G SFI-4.2 interface then don't call the config_sfi42_10g() function.

NOTE: The user must power down B-side MR slices which are being used for XFI interfaces.

Table 14 HSIF Provisioning Functions

Interface	API	Modules Supported
SFI-5.1	ten_hl_config_sfi51	Module A , Module B
SFI-4.2 40G	ten_hl_config_sfi42_40g	Module A , Module B
SFI-4.2 10G	ten_hl_config_sfi42_10g	Module A , Module B
SFI-4.1	ten_hl_config_sfi41	API is hardcoded for electrical traffic on Module A and optical on module B
XFI	ten_hl_hsif_config_xfi	Module B
XAUI	ten_hl_hsif_config_xaui	Module A , Module B
MR Disable	ten_hl_hsif_powerdown_mr	Module A , Module B

9.12 Configure the Client-Side High-Speed Interfaces (HSIF)

Repeat the line-side HSIF configuration steps for the client side.

9.13 Provision the 40G Circuit Processor (N40G)

9.13.1 Baseline Configuration

The user should configure the N40G using the API:

```
ten_hl_n40g_config()
```

The parameters and a description of their use for this API can be found in the N40G block API section. The block config must occur before the FEC config.

9.13.2 Configure 40G FEC

Once the 40G block is configured, the user should configure the 40G FEC, using the high-level FEC API (config_fec_40g). See the FEC API section for more details.

9.13.3 Adjust N40G for Aggregation

If this is an aggregation configuration, then a few other N40G APIs must be called to prepare for it:

Set up BYPASS and DISABLE configurations (as needed) using something like

```
ten_hl_n40g_config(mod_b, TEN_N40G_RX_BYPASS_SNT_MON,
TEN_N40G_TX_DISABLE)
```

Note that the parameter choice could depend on the application.

Now disable the N40G deskew function (which aligns four incoming streams by frame pulse) since all four channels may not be provisioned

```
ten_n40g_set_dsbldskw($mod_a, 1);
```

Set an appropriate value for PTI, using the high-level PTI config API:

ten_hl_n40g_config_pti($mod_a, TEN_TRAFFIC_TYPE_ODTU23);

Now disable unneeded clocks (for each side that is doing aggregation), using low-level APIs::

```
ten_mpif_global_clock_disable_n40g_n10g( $dev_id,
TEN_MODULE_A, TEN_SLICE_ALL, TEN_MPIF_N40G_DATAPATH,
CS_TX_AND_RX, CS_ENABLE );

ten_mpif_global_clock_disable_n40g_n10g( $dev_id,
TEN_MODULE_B, TEN_SLICE_ALL, TEN_MPIF_N40G_DATAPATH,
CS_TX_AND_RX, CS_ENABLE );
```

9.14 Start Cross-Connect and Clocks

Apply the low-level clock disable function to remove resets from the XCON and allow clocks to operate:

```
ten_mpif_global_clock_disable_common( $dev_id,
CS_ENABLE, TEN_MPIF_GLOBAL_CLOCK_DISABLE_COMMON_XCON
);
ten_mpif_global_reset_common( $dev_id,
CS_RESET_DEASSERT, TEN_MPIF_GLOBAL_RESET_COMMON_XCON
);
ten_mpif_global_reset_common( $dev_id,
CS_RESET_DEASSERT,
TEN_MPIF_GLOBAL_RESET_COMMON_TOPSYS );
```

9.15 Provision the 10G Traffic

Now the user can apply one of the high-level traffic-provisioning APIs. The traffic APIs are discussed in the Traffic Functions section of this document.

The user only needs to call one API to set all aspects of the traffic configuration. This greatly simplifies provisioning of CS604x. However, the user is limited to the supported traffic functions.

In addition to functions involving OTN, Sonet, and 10GE, there are also idle function for channels that are not carrying traffic.

9.16 Configure 10G FEC

Once the traffic type is configured, the user should configure the 10G FEC, using the high-level FEC API (config_fec_10g). See the FEC API section for more details.

If FEC is not needed (e.g. a null-traffic channel), then it should be deallocated at this time.

9.17 Configure Clock Muxes

Set all of the clock muxes at this time. See the clock-mux section of this document for more details.

9.18 Wait For GPLLs To Lock

Use the low-level API ten_mpif_clock_select_gpll_in() to do the initial level of clock muxing. Then call the high-level API to wait for lock. A typical example would be:

```
ten_mpif_clock_select_gpll_in(dev_id,
      mod_line, slice_line, CS_TX, 0);
config_waitfor_gpll_lock(mod_line, slice_line);
```

9.19 Remove Soft Resets

The final step in device provisioning is removing the soft resets applied to all blocks. This will enable the flow of traffic through the device. Blocks that are being used can have SW reset removed via the following calls. Do not remove resets for blocks that are not being used.

```
/* N40G */
ten_n40g_set_global_resets( module_id, CS_RESET_DEASSERT,
TEN_N40G_GLOBAL_RESETS_ALL );

/* N10G, PP10G, XCON Elastic Store, Syncdsync */
ten_hl_config_remove_soft_resets($mod_line, $slice_line,
$mod_client, $slice_client, $hl_client, $aggregation);

/* Force clock switch */
ten_mpif_set_clock_switch_force( dev_id,
TEN_MODULE_A_AND_B, TEN_SLICE_ALL, CS_TX_AND_RX, CS_DISABLE
);

/* HSIF */
/* module_num = TEN_MODULE_A | TEN_MODULE_B |
TEN_MODULE_A_AND_B */
ten_mpif_global_reset_hsif( dev_id, module_num,
CS_TX_AND_RX, CS_RESET_DEASSERT );
```

9.20 Check / Wait For HSIF SERDES Lock

Now that the HSIF is out of reset, check the lock status for the SERDES for whichever type of HSIF interface you are using. Use one or more of:

```
check_mr_filt_lock_40g
check_mr_filt_lock
check_xfi_filt_lock_40g
```

A. Appendix: Script Organization

The Perl script is located in the “configs” directory.

r5	This directory contains the new Release 5.0 “full.pl” configuration script and the associated Perl modules.

Refer to the Perl scripts for examples of using the Device Driver to provision the CS604x for different types of traffic.

B. Appendix: Sample Configuration Routine (full.pl)

The Perl script used to configure traffic on the Evaluation Board (EVB) platform is included in the distribution in configs/r5/full.pl.

Also, there are numerous auto-generated scripts for many different traffic types that are available that can serve as examples for developing code to provision the device. Your Applications Engineer can send you the sample scripts.

C. Appendix: Using Low-level FracDiv APIs

Older scripts may use a more low-level approach to configuring the fractional dividers. Cortina does not recommend this approach, but it is detailed here for reference for those with older scripts.

The fractional dividers (if needed) can be provisioned using the functions below.

```
ten_hl_fracdiv_init( dev_id, instance, sysclk_freq,
desired_freq );

Any of the ten_frac_div_cfg_frac_div_* APIs; APIs are
available to configure all of the aspects of the fractional
dividers.
```

Internal Pilot Fractional Dividers

The first eight fractional dividers are used for internal pilot clocks.

Table 15 Fractional Dividers 0-7

Fractional Divider	Receive Interface
0	10G Side A Port 1 40G Side A
1	10G Side A Port 2
2	10G Side A Port 3
3	10G Side A Port 4
4	10G Side B Port 1 40G Side B
5	10G Side B Port 2
6	10G Side B Port 3
7	10G Side B Port 4

Example: OTU3 Side A Pilot Clock Fractional Dividers

With a 400 MHz sys_refclk and Side A running at the OTU3 rate of 43.018 GHz, the desired frequency is 1/128 of the OTU3 rate.

```
/* Underscores have been inserted in the numbers for
clarity */
ten_hl_fracdiv_init( dev_id, 0, 400_000_000, 336_081_250 );
ten_frac_div_cfg_frac_div_s2en( dev_id, 0, 0 );
ten_hl_fracdiv_init( dev_id, 1, 400_000_000, 336_081_250 );
ten_frac_div_cfg_frac_div_s2en( dev_id, 1, 0 );
ten_hl_fracdiv_init( dev_id, 2, 400_000_000, 336_081_250 );
ten_frac_div_cfg_frac_div_s2en( dev_id, 2, 0 );
ten_hl_fracdiv_init( dev_id, 3, 400_000_000, 336_081_250 );
ten_frac_div_cfg_frac_div_s2en( dev_id, 3, 0 );
```

Example: OC192 Side B Pilot Clock Fractional Dividers

With a 400 MHz sys_refclk and the OC192 ports running at 9.953 GHz, the desired frequency is 1/32 of the OC192 rate.

```
/* Underscores have been inserted for clarity */
ten_hl_fracdiv_init( dev_id, 4, 400_000_000, 311_040_000 );
ten_frac_div_cfg_frac_div_s2en( dev_id, 4, 0 );
ten_hl_fracdiv_init( dev_id, 5, 400_000_000, 311_040_000 );
ten_frac_div_cfg_frac_div_s2en( dev_id, 5, 0 );
```

Example: 10GE Side B Pilot Clock Fractional Dividers

With a 400 MHz sys_refclk and the 10GE LAN ports running at 10.313 GHz, the desired frequency is 1/32 of the 10GE LAN rate.

```
/* Underscores have been inserted for clarity */
ten_hl_fracdiv_init( dev_id, 6, 400_000,000, 322_265_624 );
ten_frac_div_cfg_frac_div_s2en( dev_id, 6, 0 );
ten_hl_fracdiv_init( dev_id, 7, 400_000,000, 322_265_624 );
ten_frac_div_cfg_frac_div_s2en( dev_id, 7, 0 );
```

Backup Clock Fractional Dividers

The remaining eight fractional dividers are used for client backup clocks. This is necessary for proper protection switching on fiber pull events. Refer to Sections 2.14.3.1 and 2.14.3.2 of revision 0.6 of the CS600x Data Sheet for information on line clock and client clock protection.

In addition to the ten_frac_div_cfg_frac_div_enable function described above, the following function will also be used.

```
ten_mpif_set_clock_switch_force(dev_id, module_num, slice,
dir, ctl );
```

Table 16 Fractional Dividers 8-15

Fractional Divider	Receive Interface
8	RX Side A Port 1
9	RX Side A Port 2
10	RX Side A Port 3
11	RX Side A Port 4
12	RX Side B Port 1
13	RX Side B Port 2
14	RX Side B Port 3
15	RX Side B Port 4

Example: Disabling the Side A Backup Fractional Dividers

```
ten_frac_div_cfg_frac_div_enable( dev_id, 8, 0 );
ten_frac_div_cfg_frac_div_enable( dev_id, 9, 0 );
ten_frac_div_cfg_frac_div_enable( dev_id, 10, 0 );
ten_frac_div_cfg_frac_div_enable( dev_id, 11, 0 );
```

Example: OC192 Backup Fractional Dividers

```
/* Underscores have been inserted for clarity */
ten_hl_fracdiv_init( dev_id, 12, 400_000_000, 9_720_000 );
ten_mpif_set_clock_switch_force( dev_id, TEN_MODULE_B,
TEN_SLICE0, CS_TX, CS_DISABLE );
ten_hl_fracdiv_init( dev_id, 13, 400_000_000, 9_720_000 );
ten_mpif_set_clock_switch_force( dev_id, TEN_MODULE_B,
TEN_SLICE1, CS_TX, CS_DISABLE );
```

Example: Disabling the 10GE LAN Backup Fractional Dividers

```
ten_frac_div_cfg_frac_div_enable( dev_id, 14, 0 );
ten_frac_div_cfg_frac_div_enable( dev_id, 15, 0 );
```

D. Appendix: Logging

The driver can log all the major configuration activity, including register reads and writes.

By default, only errors are logged for normal operation. For debugging purposes, the API ten_drvr_ctl_logging can be used to selected additional levels of logging .

CS_DISABLE: Nothing is logged, including errors.

CS_ENABLE: Enable all logging.

CS_LOG_ERRORS_AND_DELAYS_ONLY: Logs errors, and also logs each time the driver uses a delay.

CS_LOG_DELAYS_ONLY: Logs only when driver uses a delay.

CS_LOG_ERRORS_ONLY: This is the default mode, only errors are logged.

```
/*****************************************************************/
/* $rtn_hdr_start   CONTROL LOGGING OUTPUT                       */
/* CATEGORY   : Driver                                           */
/* ACCESS     : Public                                           */
/* BLOCK      : GENERAL                                          */
/* CHIP       : Tenabo                                           */
void ten_drvr_ctl_logging(cs_ctl_t log_ctl)
/* INPUTS     : o Logging Control                                */
/* OUTPUTS    : ----                                             */
/* RETURNS    : ----                                             */
/* DESCRIPTION:                                                  */
/* Enables/disables debug logging messages to stout.             */
/* By default, only errors are logged (CS_LOG_ERRORS_ONLY).      */
/*                                                               */
/* The [ctl] parameter is specified as:                          */
/*   CS_DISABLE                = 0, (nothing logged)             */
/*   CS_ENABLE                 = 1, (everything logged)          */
/*   CS_LOG_DELAYS_AND_ERRORS = 2, (only delays/errors logged)  */
/*   CS_LOG_DELAYS_ONLY        = 3, (only delays logged)         */
/*   CS_LOG_ERRORS_ONLY        = 4  (only errors logged).        */
/*                                                               */
/* $rtn_hdr_end                                                  */
/*****************************************************************/
```

E. Appendix: SiLabs

There are a number of testbench (tb_{FUNCTION}) that can be used to set up or alter clocking external to the CS604x. Several of these relate to the SiLabs part.

a. Configuring the SiLabs

The SiLabs part itself can be configured with the high-level API config_silab().

```
config_silab ($line_module, $line_ch, $client_module,
      $client_ch, $line_type, $client_type,
      $line_interface, $client_interface, $k_divider,
      $line_mode, $client_mode, $sync, $dyn_repro,
      $aggregation, $line_fec, $client_fec))
```

Parameter description:

```
module_id
      A (0) or B (1)

client_module, line_module
      MODULE_A, MODULE_B
client_ch, line_ch
      0 - 3
client_type
      oc192, otu2, otu210geopu2e(otu2e), 10gelan,
      fc800(8gfc), fc1200(10gfc),
      otu210gfcopu1e(otu1f)
line_type
      otu2, otu210geopu2e(otu2e),
      otu210gfcopu1e(otu1f)
k_divider
      1 - 128
client_interface
      SFI41, XAUI, SFI42_10G, SFI42_40G, SFI51_40G,
      XFI
line_interface
      SFI41, XAUI , SFI42_10G, SFI42_40G, SFI51_40G
client_mode, line_mode
      S_40G, QUAD_10G
sync
      0 = ASYNC, 1 = SYNC
```

b. Manipulating the SiLabs ical bit

Two APIs which are particularly important for configuration are:

- tb_ical_40g($mod)
- tb_ical($mod, $slice)

These are used to toggle the ical bit in the SiLabs, which must be done after the HSIF interfaces are configured. Usage of these bits enables the software to speed up the locking process.

Contact Information

Cortina Systems, Inc.
840 W. California Ave
Sunnyvale, CA 94086

408-481-2300

sales@cortina-systems.com
www.cortina-systems.com

To provide comments on this document:

documentation@cortina-systems.com