

Project Zipline Shared Support Block

Micro Architecture Specification

**Authors:**

**Microsoft Corporation**

**Broadcom Corporation**

**Revision History**

|  |  |
| --- | --- |
| Date | Description |
| 04/03/2019 | Version 1.0 |
|  |  |

**License**

Contributions to this Specification are made under the terms and conditions set forth in the Open Web Foundation Contributor License Agreement (“OWF CLA 1.0”) (“Contribution License”) by:

Microsoft Corporation

Broadcom Corporation

Usage of this Specification is governed by the terms and conditions set forth in Open Web Foundation Final Specification Agreement (“OWFa 1.0”) (“Specification License”).

**Note**: The following clarifications, which distinguish technology licensed in the Contribution License and/or Specification License from those technologies merely referenced (but not licensed), were accepted by the Incubation Committee of the OCP:

**None.**

NOTWITHSTANDING THE FOREGOING LICENSES, THIS SPECIFICATION IS PROVIDED BY OCP "AS IS" AND OCP EXPRESSLY DISCLAIMS ANY WARRANTIES (EXPRESS, IMPLIED, OR OTHERWISE), INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, OR TITLE, RELATED TO THE SPECIFICATION. NOTICE IS HEREBY GIVEN, THAT OTHER RIGHTS NOT GRANTED AS SET FORTH ABOVE, INCLUDING WITHOUT LIMITATION, RIGHTS OF THIRD PARTIES WHO DID NOT EXECUTE THE ABOVE LICENSES, MAY BE IMPLICATED BY THE IMPLEMENTATION OF OR COMPLIANCE WITH THIS SPECIFICATION. OCP IS NOT RESPONSIBLE FOR IDENTIFYING RIGHTS FOR WHICH A LICENSE MAY BE REQUIRED IN ORDER TO IMPLEMENT THIS SPECIFICATION. THE ENTIRE RISK AS TO IMPLEMENTING OR OTHERWISE USING THE SPECIFICATION IS ASSUMED BY YOU. IN NO EVENT WILL OCP BE LIABLE TO YOU FOR ANY MONETARY DAMAGES WITH RESPECT TO ANY CLAIMS RELATED TO, OR ARISING OUT OF YOUR USE OF THIS SPECIFICATION, INCLUDING BUT NOT LIMITED TO ANY LIABILITY FOR LOST PROFITS OR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL OR PUNITIVE DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND EVEN IF OCP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contents

[1 Terminology 10](#_Toc5184633)

[2 Overview 11](#_Toc5184634)

[3 Inbound Stream FIFO (ISF) 13](#_Toc5184635)

[3.1 Features 13](#_Toc5184636)

[3.2 Description 13](#_Toc5184637)

[3.2.1 FIFO 14](#_Toc5184638)

[3.2.2 Command Protocol Checking 15](#_Toc5184639)

[3.2.3 Command TLV Modifications 15](#_Toc5184640)

[3.3 Debug Support 26](#_Toc5184641)

[3.4 Dataflow Control 26](#_Toc5184642)

[3.5 Debug Counters 28](#_Toc5184643)

[3.6 Selectable Event Counter Support 29](#_Toc5184644)

[3.7 Pipeline Status Support 29](#_Toc5184645)

[3.8 Miscellaneous Debug Support 29](#_Toc5184646)

[3.9 Statistics 29](#_Toc5184647)

[3.10 Self-Test 31](#_Toc5184648)

[3.11 Interrupts 31](#_Toc5184649)

[3.12 Engine Errors 32](#_Toc5184650)

[3.13 One-Time Programmable (OTP) Inputs 32](#_Toc5184651)

[3.14 Interfaces 32](#_Toc5184652)

[4 CRC Generator and Checker (CRCGC) 33](#_Toc5184653)

[4.1 CRC Generation (CRCG) 33](#_Toc5184654)

[4.1.1 Features 33](#_Toc5184655)

[4.1.2 Data Flow Description 33](#_Toc5184656)

[4.1.3 Debug Support 36](#_Toc5184657)

[4.1.4 Statistics 37](#_Toc5184658)

[4.1.5 Interrupts 37](#_Toc5184659)

[4.1.6 Engine Errors 37](#_Toc5184660)

[4.1.7 Interfaces 37](#_Toc5184661)

[4.2 CRC Checking (CRCC) 37](#_Toc5184662)

[4.2.1 Features 37](#_Toc5184663)

[4.2.2 Data Flow Description 37](#_Toc5184664)

[4.2.3 Statistics 41](#_Toc5184665)

[4.2.4 Debug Support 42](#_Toc5184666)

[4.2.5 Statistics 42](#_Toc5184667)

[4.2.6 Interrupts 42](#_Toc5184668)

[4.2.7 Engine Errors 42](#_Toc5184669)

[4.2.8 Interfaces 42](#_Toc5184670)

[4.3 CRC Generation and Checking (CRCGC) 42](#_Toc5184671)

[4.3.1 Data Flow Description 43](#_Toc5184672)

[4.3.2 Statistics 46](#_Toc5184673)

[4.3.3 Debug Support 46](#_Toc5184674)

[4.3.4 Statistics 47](#_Toc5184675)

[4.3.5 Interrupts 47](#_Toc5184676)

[4.3.6 Engine Errors 47](#_Toc5184677)

[4.3.7 Interfaces 47](#_Toc5184678)

[4.4 CRC Polynomials 47](#_Toc5184679)

[5 Completion Generator (CG) 49](#_Toc5184680)

[5.1 Features 49](#_Toc5184681)

[5.2 Operational Description 49](#_Toc5184682)

[5.2.1 Command TLV Modifications 49](#_Toc5184683)

[5.3 Debug Support 57](#_Toc5184684)

[5.4 Statistics 57](#_Toc5184685)

[5.5 Interrupts 58](#_Toc5184686)

[5.6 Engine Errors 58](#_Toc5184687)

[5.7 Interfaces 58](#_Toc5184688)

[6 Outbound Stream FIFO (OSF) 59](#_Toc5184689)

[6.1 Features 59](#_Toc5184690)

[6.2 Description 59](#_Toc5184691)

[6.2.1 Dataflow 60](#_Toc5184692)

[6.2.2 Command TLV Modifications 61](#_Toc5184693)

[6.3 Debug Support 62](#_Toc5184694)

[6.3.1 Dataflow Control 62](#_Toc5184695)

[6.3.2 Debug Counters 63](#_Toc5184696)

[6.3.3 Post-data TLV History Buffers 64](#_Toc5184697)

[6.3.4 Pipeline Status Support 64](#_Toc5184698)

[6.3.5 Miscellaneous Debug Support 64](#_Toc5184699)

[6.4 Statistics 64](#_Toc5184700)

[6.5 Interrupts 65](#_Toc5184701)

[6.6 Self-Test 65](#_Toc5184702)

[6.7 Outbound Halting 65](#_Toc5184703)

[6.8 Interfaces 65](#_Toc5184704)

[6.9 Engine Errors 66](#_Toc5184705)

[7 Statistics Accumulator (SA) 67](#_Toc5184706)

[7.1 Features 67](#_Toc5184707)

[7.2 Operational Description 67](#_Toc5184708)

[7.2.1 Controls 67](#_Toc5184709)

[7.2.2 Countable Events 68](#_Toc5184710)

[7.3 Debug Support 92](#_Toc5184711)

[7.4 Interfaces 92](#_Toc5184712)

[8 Scheduler Update (SU) 93](#_Toc5184713)

[8.1 Features 93](#_Toc5184714)

[8.2 Description 93](#_Toc5184715)

[8.3 Debug Support 95](#_Toc5184716)

[8.3.1 Debug Counters 95](#_Toc5184717)

[8.3.2 Scheduler Update Message History Buffer 95](#_Toc5184718)

[8.3.3 Miscellaneous Debug Support 96](#_Toc5184719)

[8.4 Self-Test 96](#_Toc5184720)

[8.5 Interrupts 96](#_Toc5184721)

[8.6 Interfaces 96](#_Toc5184722)

[9 CCEIP and CDDIP Support Blocks 97](#_Toc5184723)

[9.1 Description 97](#_Toc5184724)

[9.2 Idle Interface 97](#_Toc5184725)

[9.3 KME Interface Support 97](#_Toc5184726)

[9.4 Debug Support 97](#_Toc5184727)

[9.5 Dataflow Mux 97](#_Toc5184728)

[9.6 Interrupts 98](#_Toc5184729)

[9.7 Control and Status Registers 98](#_Toc5184730)

[10 Inter-Stage Monitor (ISM) 99](#_Toc5184731)

[10.1 Features 99](#_Toc5184732)

[10.2 Description 99](#_Toc5184733)

[10.3 Interface Description 100](#_Toc5184734)

[11 TLV Parser 102](#_Toc5184735)

[11.1 Features 102](#_Toc5184736)

[11.2 Description 102](#_Toc5184737)

[11.2.1 AXI4S Slave 103](#_Toc5184738)

[11.2.2 TLV Disassembler 103](#_Toc5184739)

[11.2.3 TLV Reassembler 104](#_Toc5184740)

[11.2.4 AXI4S Master 105](#_Toc5184741)

[11.2.5 Debug Support 108](#_Toc5184742)

[11.3 Interfaces 110](#_Toc5184743)

[11.4 Implementation Details 114](#_Toc5184744)

[11.4.1 Verilog Files 114](#_Toc5184745)

[11.4.2 Parameters 114](#_Toc5184746)

[12 Interface Descriptions 116](#_Toc5184747)

[13 References 120](#_Toc5184748)

Table of Figures

[Figure 1 : CCEIP Block Diagram 10](#_Toc5184599)

[Figure 2 : CDDIP Block Diagram 11](#_Toc5184600)

[Figure 3 : ISF Block Diagram 13](#_Toc5184601)

[Figure 4 : ISF TLV Modifications for Simple Commands 15](#_Toc5184602)

[Figure 5 : ISF TLV Modifications for Compound Command Frames 15](#_Toc5184603)

[Figure 6 : ISF TLV Modifications for Simple Commands with User Prefix Data 18](#_Toc5184604)

[Figure 7 : ISF TLV Modifications for Compound Command Frames with User Prefix Data 18](#_Toc5184605)

[Figure 8 : CRCG Block Diagram 32](#_Toc5184606)

[Figure 9 : Simple Command Modifications for Encrypted/Compressed CRC Generation 32](#_Toc5184607)

[Figure 10 : Compound Command Modifications for Encrypted/Compressed CRC Generation 33](#_Toc5184608)

[Figure 11 : Simple Command Modifications for CRC Generation for FRMD\_USER\_PI16 34](#_Toc5184609)

[Figure 12 : Compound Command Modifications for CRC Generation for FRMD\_USER\_PI16 34](#_Toc5184610)

[Figure 13 : CRCC Block Diagram 36](#_Toc5184611)

[Figure 14 : Simple Command Raw CRC Checker Stack-up 37](#_Toc5184612)

[Figure 15 : Compound Command Raw CRC Checker Stack-up 37](#_Toc5184613)

[Figure 16 : CDDIP Simple Command Compressed/Encrypted CRC Checker Stack-up with Prefix TLV 38](#_Toc5184614)

[Figure 17 : CDDIP Compound Command Compressed/Encrypted CRC Checker Stack-up with Prefix TLV 38](#_Toc5184615)

[Figure 18 : CCEIP Simple Command Compressed/Encrypted CRC Checker Stack-up 39](#_Toc5184616)

[Figure 19 : CCEIP Compound Command Compressed/Encrypted CRC Checker Stack-up 39](#_Toc5184617)

[Figure 20 : Simple Command Modifications for FRMD\_USER\_PI64 (CMD.MD\_TYPE {1,2}) and FRMD\_USER\_VM 42](#_Toc5184618)

[Figure 21 : Compound Command Modifications for FRMD\_USER\_PI64 (CMD.MD\_TYPE {1,2}) and FRMD\_USER\_VM 43](#_Toc5184619)

[Figure 22 : Simple Command Modifications for FRMD\_USER\_NULL and FRMD\_USER\_PI64 (CMD.MD\_TYPE {0,3}) 43](#_Toc5184620)

[Figure 23 : Compound Command Modifications for FRMD\_USER\_NULL and FRMD\_USER\_PI64 (CMD.MD\_TYPE {0,3}) 43](#_Toc5184621)

[Figure 24 : Simple Command Modifications for FRMD\_USER\_PI16 43](#_Toc5184622)

[Figure 25 : Compound Command Modifications for FRMD\_USER\_PI16 44](#_Toc5184623)

[Figure 26 : CG TLV Modifications for Simple Commands 47](#_Toc5184624)

[Figure 27 : CG TLV Modifications for Compound Command Frames 47](#_Toc5184625)

[Figure 28: OSF Block Diagram 58](#_Toc5184626)

[Figure 29 : CCEIP OSF TLV Modifications for Simple Commands 59](#_Toc5184627)

[Figure 30 : CCEIP OSF TLV Modifications for Compound Command Frames 59](#_Toc5184628)

[Figure 31 : CDDIP OSF TLV Modifications for Simple Commands 59](#_Toc5184629)

[Figure 32 : CDDIP OSF TLV Modifications for Compound Command Frames 60](#_Toc5184630)

[Figure 33 : CCEIP/CDDIP OSF Output Stack-Up 60](#_Toc5184631)

[Figure 34 : ISM Block Diagram 97](#_Toc5184632)

Table of Tables

[Table 1 : ISF FIFO Entry Format 13](#_Toc4072295)

[Table 2 : Expected TLV Order from Simple Commands 14](#_Toc4072296)

[Table 3 : Expected TLV Order from Compound Commands 14](#_Toc4072297)

[Table 4 : PREFIX DATA TLV Format 18](#_Toc4072298)

[Table 5 : Mini-FOOTER TLV Format 21](#_Toc4072299)

[Table 6 : CCEIP Footer Creation 22](#_Toc4072300)

[Table 7 CDDIP Footer Creation 23](#_Toc4072301)

[Table 8 : RawDataMACSize Logic for USER\_VM FRMDs in CCEIP 24](#_Toc4072302)

[Table 9 : Trigger Freeze Configuration Registers 27](#_Toc4072303)

[Table 10 : ISF Statistics Events 29](#_Toc4072304)

[Table 11 : ISF Interrupts 30](#_Toc4072305)

[Table 12 : ISF OTP Inputs 31](#_Toc4072306)

[Table 13 : ISF Interfaces 31](#_Toc4072307)

[Table 14 : CDDIP CRCG #0 Function Table 34](#_Toc4072308)

[Table 15 : CRC History Buffer Entry Format 35](#_Toc4072309)

[Table 16 : CCEIP CRCC #0 Checking Cases 38](#_Toc4072310)

[Table 17 : CRCC Error Codes 41](#_Toc4072311)

[Table 18 : CCEIP CRCGC Function Table 43](#_Toc4072312)

[Table 19 : CRCGC Interfaces 46](#_Toc4072313)

[Table 20 : CRC Polynomials 47](#_Toc4072314)

[Table 21 : FOOTER TLV Format 51](#_Toc4072315)

[Table 22 : Function Table for RawDataCkSumProtocol Footer Field 51](#_Toc4072316)

[Table 23 : CCEIP Output FRMD Creation 52](#_Toc4072317)

[Table 24 : CDDIP Output FRMD Creation 53](#_Toc4072318)

[Table 25 : STATS TLV Format 55](#_Toc4072319)

[Table 26 : CG Statistics Events 57](#_Toc4072320)

[Table 27 : CG Interfaces 57](#_Toc4072321)

[Table 28 : OSF Statistics Events 64](#_Toc4072322)

[Table 29 : ISF Interrupts 65](#_Toc4072323)

[Table 30 : OSF Interfaces 66](#_Toc4072324)

[Table 31 : Per-Module Statistic Events 68](#_Toc4072325)

[Table 32 : CCEIP Statistic Events Description 85](#_Toc4072326)

[Table 33 : CDDIP Statistic Events Description 92](#_Toc4072327)

[Table 34 : SA Interfaces 92](#_Toc4072328)

[Table 35 : SCH TLV Format (64-bit words) 94](#_Toc4072329)

[Table 36 : Serialized SCH TLV Format 95](#_Toc4072330)

[Table 37 : Scheduler Update History Buffer Entry Format 95](#_Toc4072331)

[Table 38 : SU Interrupts 96](#_Toc4072332)

[Table 39 : SU Interfaces 96](#_Toc4072333)

[Table 40 : ISM Interface Signals 101](#_Toc4072334)

[Table 41 : 64-bit AXI4S Slave Interface 116](#_Toc4072335)

[Table 42 : 64-bit AXI4S Master Interface 117](#_Toc4072336)

[Table 43 : 8-bit AXI4S Master Interface to Scheduler 118](#_Toc4072337)

[Table 44 : SU Interface from Huffman Compressor/XP10 Decompressor 118](#_Toc4072338)

[Table 45 : RBUS Ring Input Interface 119](#_Toc4072339)

[Table 46 : RBUS Ring Output Interface 119](#_Toc4072340)

# Terminology

AXI4S: AXI4 Streaming Interface

CCEIP: Compression Encryption and Integrity Pipeline

CDDIP: Decompression Decryption and Integrity Pipeline

CG: Completion Generator

CKMIC: Message Integrity Code (un-keyed)

CKMICMAC: Message Integrity Code/Message Authentication Code (keyed)

CRCC: CRC Checker

CRCG: CRC Generator

CRCGC: CRC Generator/Checker

CSR: Control and Status Registers

ISM: Inter-Stage Monitor

ISF: Input Stream FIFO

MIC: Message Integrity Code

OSF: Output Stream FIFO

System: Zipline blocks external to CDDIP/CCEIP (e.g. TXC)

SA: Statistics Accumulator

SU: Scheduler Update

SW: Software

TT: TLV Type

TVLP: TLV parser

# Overview

This document describes the microarchitecture of several support blocks that are found in the CCEIP and CDDIP engines. The blocks described in this document are provided in the list below.

* Input Stream FIFO
* Output Stream FIFO
* CRC Generator
* CRC Checker
* CRC Generator/Checker
* Statistics Accumulator
* Completion Generator
* Scheduler Update
* CDDIP and CCEIP support blocks
* ISM
* TLV Parser

Block diagrams of the CCEIP and CDDIP are shown below in Figure 1 and Figure 2.



Figure : CCEIP Block Diagram



Figure : CDDIP Block Diagram

# Inbound Stream FIFO (ISF)

## 3.1 Features

* 1K x 92 input FIFO
* Protocol checking of inbound command order
* Modifications to the TLV for inbound commands
  + Create the CMD.trace bit
  + Append the mini-FOOTER TLV and copy fields from FRMD and CMD TLVs to the mini-mini-FOOTER
  + Force the entire 32-bits of the CMD.debug\_ctl to all 0’s if the dbg\_cmd\_disable OTP bit is set to 1 to prevent the use of any debug commands.
  + Create Prefix TLV for frames with user prefix data embedded in the DATA TLV
* SA Interface
  + Configurable and fixed statistic events
* Debug Support
  + Debug Counters
  + Debug hooks to support SW control of input data flow through CCEIP and CDDIP
    - Single step
    - Trigger freeze
  + Interfaces to CCEIP and CDDIP support blocks to provide pipeline status information
  + Overflow, protocol error, and system stall interrupts, amongst others.
* Operation configurable for any CDDIP/CCEIP differences via tie-off

## Description

The location of the ISF in the CCEIP and CDDIP is shown in Figure 1 and Figure 2. A block diagram of the ISF is shown in Figure 3.



Figure : ISF Block Diagram

### FIFO

The Inbound Stream FIFO (ISF) is a 1K x 92 FIFO that connects to the primary 64-bit AXI4S slave interface of the CCEIP and CDDIP. It provides a more robust interface between the system and the CCEIP/CDDIP to minimize thrashing due to short term back-pressure from the engines. The format of each FIFO entry is shown in Table 1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Bits** | 91:82 | 81 | 80:73 | 72:65 | 64 | 63:0 |
| **Field** | Reserved | TID | TSTRB[7:0] | TUSER[7:0] | TLAST | TDATA[63:0] |

Table : ISF FIFO Entry Format

The ISF also has 2 selectable watermarks that are used to initiate and remove backpressure to the system and to provide hysteresis.

* Use watermark: Do not allow the ISF to fill above this mark. This sets the depth at which the ISF asserts backpressure. Controlled via *cr\_isf\_ctl\_config.use\_wmark\_sel.*
* Request watermark: Do not request new transactions until below this watermark. This sets the depth at which the ISF de-asserts backpressure. Controlled via *cr\_isf\_ctl\_config.req\_wmark\_sel.*

### Command Protocol Checking

The ISF implements protocol checking by checking the order of the TLVs for a command. The expected order for both simple and compound commands is shown in Table 2 and Table 3. Any deviations from this order result in a Protocol Error interrupt being generated, and the ISF stops processing this command and any subsequent commands as soon as this error is detected. An engine reset is required to recover from this error.

|  |
| --- |
| RQE |
| CMD |
| FRMD |
| DATA |
| CQE |

Table : Expected TLV Order from Simple Commands

|  |
| --- |
| RQE |
| CMD |
| FRMD |
| DATA |
| FRMD |
| DATA |
| …….. |
| FRMD |
| DATA |
| CQE |

Table : Expected TLV Order from Compound Commands

### Command TLV Modifications

In order to reduce complexity in the blocks that deliver the commands to the CCEIP/CDDIP, the ISF in supports modifications to the TLV stack-up received. For commands without “user prefix” data embedded in the DATA TLV, the changes are shown in Figure 4 and Figure 5 and summarized below.



Figure : ISF TLV Modifications for Simple Commands



Figure : ISF TLV Modifications for Compound Command Frames

* Create the CMD.trace bit. The details of this are provided in 3.3.3.
* Force the entire 32-bits of the CMD.debug\_ctl to all 0’s if the dbg\_cmd\_disable OTP bit is set to 1. This prevents the use of any debug commands.
* Append a mini- FOOTER TLV and populate with fields from the FRMD and CMD TLVs . The details of this are provided in 3.2.3.2.

#### User Prefix TLV

When CMD.comp\_ctl.xp10\_prefix\_mode is set to USER\_DEFINED\_PREFIX (0x1) and CMD.comp\_ctl.algorithm is either XPRESS10, XPRESS10\_COMP\_4KB or XPRESS10\_COMP\_8KB[[1]](#footnote-2), the DATA TLV contains 1K-64K of “user prefix” data, as indicated by CMD. comp\_ctl.xp10\_prefix\_selector, and an associated CRC that are embedded at the start of the opaque data. In this case, the ISF performs the additional modifications described below and shown in Figure 6 and Figure 7:

* Create a PREFIX DATA TLV out of the user prefix data by inserting word 0 of this type of TLV, as shown in Table 4, just before the user prefix data/CRC-64 embedded in the opaque data.

* Move word 0 of the DATA TLV after the user prefix data.

|  |  |  |  |
| --- | --- | --- | --- |
| **Fields** | **Size** | **Word** | **Notes** |
| Type | 8 | 0 | Value for Prefix TLV is 0x04 |
| Length | 8 | 0 | 0x00 |
| Engine ID | 4 | 0 | 0-3 CCEIP, 4-7 CDDIP (copied from DATA TLV) |
| Sequence Num | 8 | 0 | Increments per RQE/Command (copied from DATA TLV) |
| Frame Num | 11 | 0 | Copied from DATA TLV  0 – Simple Command  0-2047 for Compound Commands |
| XP10\_Prefix\_Sel | 6 | 0 | PrefixSrc =0:  Attached prefix number from Prefix Engine  PrefixSrc = 1:  Size of User Prefix in 1KB units +1KB (copied from CMD. comp\_ctl.xp10\_prefix\_selector)  Example:  0x0 = 1KB  0x1 = 2KB  .  .  0x3f = 64KB |
| Prefix\_Src | 1 | 0 | Prefix Source  0 : Prefix Engine  1: User Prefix from ISF |
| Parity | 2 | 0 | BIP-2 (Even) |
| Prefix Data | 1KB-64KB | 128-8192 | Supports 1KB-64KB prefixes in 1KB increments. |
| CRC-64 | 64 | 129-8193 | CRC for Prefix Data |

Table : PREFIX DATA TLV Format



Figure : ISF TLV Modifications for Simple Commands with User Prefix Data



Figure : ISF TLV Modifications for Compound Command Frames with User Prefix Data

**Error Handling**

The ISF has the ability to detect and accommodate TLV misconfigurations where the user prefix size in **CMD. comp\_ctl.xp10\_prefix\_selector** is greater than or equal to the size of the DATA TLV it receives. It flags these cases as errors by inserting an ISF\_PREFIX\_ERR (180) in the FOOTER TLV. Additionally, the ISF takes the following action:

* Pads out the User Prefix TLV data to the length specified in **CMD. comp\_ctl.xp10\_prefix\_selector** using all 0’s for the data
* Creates a DATA TLV with a Word 0 followed by 8-bytes of data which is all 0’s

#### FOOTER TLV

The ISF appends a mini-footer (i.e. no GUID or IV fields) for both simple and compound commands. Table 5 shows the internal format of the Mini-FOOTER TLV. The ISF uses fields from the FRMD and CMD TLVs to create the FOOTER TLV.

Table 6 and Table 7 show the various recipes for creating the FOOTER for each applicable FRMD for both the CCEIP and CDDIP. Sources that are different than those used for FRMD\_USER\_NULL are highlighted in yellow. The various FRMD formats are defined in [3].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | | **Size** | **Word** | **Notes** |
| Type | | 8 | 0 | Type = 6 |
| Length | | 8 | 0 | Length = 28 |
| Sequence Number | | 8 | 0 | Increments per RQE/Command |
| Engine ID | | 4 | 0 | 0-3 CCEIP, 4-7 CDDIP |
| Reserved | | 4 | 0 | Reserved |
| Frame Number | | 11 | 0 | 0 – Simple Command  0-2047 for Compound Commands |
| Reserved | | 1 | 0 | Reserved |
| Status | Coding | 2 | 0 | 0: Raw  1: Parsable  2: XP10CFH 4K  3: XP10CFH 8K |
| EncCmpDataMacSize | 2 | 0 | 0: 64 bits valid (upper 64 bits of the field)  1: 128 bits valid (upper 128 bits of the field)  2: 256 bits valid (full size of the field)  3: Reserved |
|  | RawDataMACSize | 2 | 0 | 0: 64 bits valid (upper 64 bits of the field1: 128 bits valid (upper 128 bits of the field)  2: 256 bits valid (full size of the field)  3: Reserved |
|  | Reserved | 2 | 0 | Reserved |
| GenFRMDOutType | | 8 | 0 |  |
| Reserved | | 2 |  |  |
| Parity | | 2 | 0 | BIP-2 (Even) |
| RawDataMAC | | 256 | 1-4 | SHA2, SHA2-HMAC (Big Endian format with bits [255:192] in Word1) |
| RawDataCkSum | | 64 | 5 | CRC-64 |
| RawDataCkSumProtocol | | 64 | 6 | CRC added to the compressed data in the Huffman Encoder for XP10, GZIP, and ZLIB,. CRC can be one of has 4 different types per the protocol specified in CMD. comp\_ctl.algorithm and the CRC specified in CMD. comp\_ctl.xp10\_crc\_mode and described in Table 22. |
| EncCmpDataMac | | 256 | 7-10 | SHA2, SHA2-HMAC, AES-GMAC (Big Endian format with bits [255:192] in Word7) |
| EncCmpDataCkSum | | 64 | 11 | CRC-64 |
| Bytes Out | | 24 | 12 | Bytes Out |
| Bytes In | | 24 | 12 | Bytes In (xfer’d over PCIe) (ISF updates) |
| NVMERawCkSum | | 16 | 12 | CRC-16T |
| Errored Frame Number | | 11 | 13 | 0 for simple commands; otherwise has the first frame in the compound command that generated the error. |
| Reserved | | 1 | 13 | Reserved |
| Error Code | | 8 | 13 | See [1] |
| Compressed Length | | 24 | 13 | Compressed Size (Huffman Encoder builds) |
| Reserved | | 20 | 13 | Reserved |

Table : Mini-FOOTER TLV Format

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FOOTER Field** | **Source of Field Values for Mini-FOOTER TLV for Supported FRMD Types (CCEIP)** | | | |
|  | **FRMD**  **USER**  **NULL** | **FRMD**  **USER**  **PI16** | **FRMD**  **USER**  **PI64** | **FRMD**  **USER**  **VM** |
| Type | Constant | Same As Null | Same As Null | Same As Null |
| Length | Constant (28) | Same As Null | Same As Null | Same As Null |
| Sequence Num | From FRMD | Same As Null | Same As Null | Same As Null |
| Engine ID | From FRMD | Same As Null | Same As Null | Same As Null |
| Reserved | All 0’s | Same As Null | Same As Null | Same As Null |
| Frame Num | From FRMD | Same As Null | Same As Null | Same As Null |
| Reserved | All 0’s | Same As Null | Same As Null | Same As Null |
| Coding | ISF Gen (Note 5) | Same As Null | Same As Null | Same As Null |
| EncCmpDataMacSize | 3 (Reserved) | Same As Null | Same As Null | Same As Null |
| RawDataMACSize | 3(Reserved) | Same As Null | Same As Null | 2(256b) or 3(Reserved)  Note 6 |
| Reserved | All 0’s | Same As Null | Same As Null | Same As Null |
| GenFRMDOutType | CMD.frame\_ctl.  out\_type | Same As Null | Same As Null | Same As Null |
| Reserved | All 0’s | Same As Null | Same As Null | Same As Null |
| Parity | ISF Gen | Same As Null | Same As Null | Same As Null |
| RawDataMAC  (Big Endian) | All 0’s | Same As Null | Same  As Null | FRMD.  RawDataMAC  Note 3 |
| RawDataCkSum | All 0’s | Same As Null | FRMD.  NVMEMetadata  CRC64 | FRMD.  RawDataCkSum |
| RawDataCkSumProtocol | All 0’s | Same As Null | Same As Null | Same As Null |
| EncCmpDataMac  (Big Endian) | All 0’s | Same As Null | Same As Null | Same As Null |
| EncCmpDataCkSum | All 0’s | Same As Null | Same As Null | Same As Null |
| BytesOut | All 0’s | Same As Null | Same As Null | Same As Null |
| BytesIn | ISF insert | Same As Null | Same As Null | Same As Null |
| NVMERawCkSum | All 0’s | FRMD.  NVMEMetadata  CRC16T | Same As Null | Same As Null |
| Errored Frame Number | All 0’s unless have ISF\_PREFIX\_ERR | Same As Null | Same As Null | Same As Null |
| Reserved | All 0’s | Same As Null | Same As Null | Same As Null |
| Error Code | All 0’s unless have ISF\_PREFIX\_ERR | Same As Null | Same As Null | Same As Null |
| Compressed Length | All 0’s | Same As Null | Same As Null | Same As Null |
| Reserved | All 0’s | Same As Null | Same As Null | Same As Null |

Table : CCEIP Footer Creation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Source of Field Values for Mini-FOOTER TLV for Supported FRMD Types (CDDIP)** | | | | | |
|  | **FRMD**  **INT**  **APP** | **FRMD**  **INT**  **SIP** | **FRMD**  **INT**  **LIP** | **FRMD**  **INT**  **VM**  **SHORT** | **FRMD**  **INT**  **VM**  **LONG** |
| Type | Constant | Same As APP | Same As APP | Same As APP | Same As APP |
| Length | Constant (28) | Same As APP | Same As APP | Same As APP | Same As APP |
| Sequence Num | From FRMD | Same As APP | Same As APP | Same As APP | Same As APP |
| Engine ID | From FRMD | Same As APP | Same As APP | Same As APP | Same As APP |
| Reserved | All 0’s | Same As APP | Same As APP | Same As APP | Same As APP |
| Frame Num | From FRMD | Same As APP | Same As APP | Same As APP | Same As APP |
| Reserved | All 0’s | Same As APP | Same As APP | Same As APP | Same As APP |
| Coding | FRMD.  Coding | Same As APP | Same As APP | Same As APP | Same As APP |
| EncCmpDataMacSize | 1(128bits) | 3(0bits) | 3(bits) | 2(256bits) | 2(256bits) |
| RawDataMACSize | 3(0bits) | 0(64bits) | 2(256bits) | 0(64b) | 2(256bits) |
| Reserved | All 0’s | Same As APP | Same As APP | Same As APP | Same As APP |
| GenFRMDOutType | CMD.frame\_ctl.  out\_type | Same As APP | Same As APP | Same As APP | Same As APP |
| Reserved | All 0’s | Same As APP | Same As APP | Same As APP | Same As APP |
| Parity | ISF Gen | Same As APP | Same As APP | Same As APP | Same As APP |
| RawDataMAC  (Big Endian) | All 0’s | Same As APP | FRMD.  RawDataMAC  Note 3 | FRMD.  RawDataMAC  Note 2 | FRMD.  RawDataMAC  Note 3 |
| RawDataCkSum | All 0’s | Same As APP | Same As APP | Same As APP | Same As APP |
| RawDataCkSumProtocol | All 0’s | Same As APP | Same As APP | Same As APP | Same As APP |
| EncCmpDataMac  (Big Endian) | FRMD.  CmpEncMAC  Note 1 | All 0’s | All 0’s | FRMD.  CmpEncMAC  Note 4 | FRMD.  CmpEncMAC  Note 4 |
| EncCmpDataCkSum | FRMD.  CmpEncCkSum | Same As APP | Same As APP | Same As APP | Same As APP |
| BytesOut | All 0’s | Same As APP | Same As APP | Same As APP | Same As APP |
| BytesIn | ISF Insert | Same As APP | Same As APP | Same As APP | Same As APP |
| NVMERawCkSum | All 0’s | Same As APP | Same As APP | Same As APP | Same As APP |
| Errored Frame Number | All 0’s unless have ISF\_PREFIX\_ERR | Same As APP | Same As APP | Same As APP | Same As APP |
| Reserved | All 0’s | Same As APP | Same As APP | Same As APP | Same As APP |
| Error Code | All 0’s unless have ISF\_PREFIX\_ERR | Same As APP | Same As APP | Same As APP | Same As APP |
| Compressed Length | FRMD.  Compressed  Length | Same As APP | Same As APP | Same As APP | Same As APP |
| Reserved | All 0’s | Same As APP | Same As APP | Same As APP | Same As APP |

Table CDDIP Footer Creation

**Notes**

1. FRMD. CmpEncMAC has the upper 128-bits of the MAC. Populate FOOTER.EncCmpDataMac[255:128] with FRMD. CmpEncMAC and FOOTER.EncCmpDataMac[127:0]=0.
2. FRMD.RawDataMAC has the upper 64-bits of the MAC. Populate FOOTER.RawDataMAC[255:192] with FRMD.RawDataMAC and FOOTER.RawDataMAC[191:0] =0.
3. FRMD.RawDataMAC has a 256-bit MAC and this is copied to FOOTER.RawDataMAC .
4. FRMD. CmpEncMAC has a 256-bit MAC and this is copied to FOOTER. EncCmpDataMac .
5. ISF maps CMD.comp\_ctl.algorithm[3:0] to FOOTER.coding[1:0] as shown in the table below:

|  |  |
| --- | --- |
| CMD.comp\_ctl.algorithm | FOOTER.Coding |
| 0 : Passthrough (NOP) | 0 : Raw |
| 1: ZLIB | 1 : Parsable |
| 2 : GZIP | 1 : Parsable |
| 3 : XP9 | 1 : Parsable |
| 4 : XP10 | 1 : Parsable |
| 5 : XP10CHU4K | 2 : XP10CFH 4K |
| 6 : XP10CHU8K | 3 : XP10CFH 8K |
| 7-15: Reserved | 0: Raw |

1. See Special Considerations section below.

**Special Considerations**

The FOOTER generation logic in the CCEIP supports a special mode that can optionally prevent the RawDataMAC supplied with a USER\_VM FRMD from being checked in the CKMIC. This is done by using the state of CMD.frame\_ctrl.md\_type[1] to control the FOOTER.RawDataMACSize value per Table 8.

|  |  |  |  |
| --- | --- | --- | --- |
| FRMD type | CMD.frame\_ctrl.  md\_type | FOOTER.  RawDataMACSize | Comments |
| USER\_VM | 1 | 0x3 (reserved) | CKMIC will not check the calculated RawDataMAC against the one provided in the Footer. |
| USER\_VM | 0 | 0x2 (256-bits) | CKMIC will check the calculated RawDataMAC againstthe one provided in the Footer. |

Table : RawDataMACSize Logic for USER\_VM FRMDs in CCEIP

## Debug Support

## Dataflow Control

The ISF has the debug support features described below that allow SW to control the flow of data through the CCEIP and CDDIP. The various modes are controlled via *cr\_isf\_debug\_ctl\_config.debug\_mode,* and the values of the modes described below are included with the heading. For non-debug operation, this field is set to NORMAL (0x0).

**SW Data Entry (RW\_DISABLE 0x1)**

Load ISF via SW and then transfer entire contents to engines in real time via SW enable.

**Sample procedure for loading the ISF FIFO via SW**

1. Write the following values to *cr\_isf\_debug\_ctl\_config* tp put the ISF into the proper mode:

*cr\_isf\_debug\_ctl\_config.debug\_mode* = 0x1

*cr\_isf\_debug\_ctl\_config.force\_sw\_access* = 0x1

1. Enable ISF FIFO into by writing *cr\_isf\_isf\_fifo\_ia\_config.op*=0x3
2. Poll *cr\_isf\_isf\_fifo\_ia\_status.code* until value is 0x0 (READY)
3. Setup write data in the format of an AXI transaction (see Table 41)

*cr\_isf\_isf\_fifo\_ia\_wdata\_part0* = TDATA[31:0]

*cr\_isf\_isf\_fifo\_ia\_wdata\_part1* = TDATA[63:0]

*cr\_isf\_isf\_fifo\_ia\_wdata\_part2* = {11’b0, TID, TSTRB, TUSER, 3’b0, TLAST}

1. Write the following values to *cr\_isf\_isf\_fifo\_ia\_config* to write the data from Step 3 to the ISF FIFO address 0x0:

*cr\_isf\_isf\_fifo\_ia\_config.op*=0x3

*cr\_isf\_isf\_fifo\_ia\_config.addr*=0x0

1. Poll *cr\_isf\_isf\_fifo\_ia\_status.code* until value is 0x0 (READY)
2. Repeat steps 4-6 while incrementing *cr\_isf\_isf\_fifo\_ia\_config.addr*  to load more locations in the ISF FIFO

**DMA Debug Buffer (RD\_DISABLE 0x2)**

In this mode, the ISF holds the system input data for inspection by SW to verify the DMA data from the system.

**Single Step (SINGLE\_STEP 0x3)**

Load ISF via SW and single step the transfer of this data to the engines also via SW. The ISF can also single step the contents of the ISF FIFO that was loaded via SW while in the DMA Debug Buffer mode described above.

**Trigger Freeze Mode (TRIG\_FREEZE 0x4)**

The Trigger Freeze mode supports a TLV based breakpoint (trigger). The trigger value is detected at the output of the Pre-TLVP FIFO shown in Figure 3. Once the trigger occurs, the ISF will halt and can be single-stepped via a debugger or other means. Any data downstream of the Pre-TLVP FIFO at the time the ISF is halted will exit the ISF normally. At any time, the ISF can be put back in NORMAL mode to resume normal operation. Any data stored in the ISF Debug or Pre-TLVP FIFOs during the time the ISF is halted will exit normally. Table 9 lists the registers involved in configuring the trigger value.

The actual contents of trigger word are stored in the *cr\_isf\_debug\_trig\_cap\_lo* and *cr\_isf\_debug\_trig\_cap\_high* registers. Since the trigger word has to be read out of the Pre-TLVP FIFO to hit the trigger, it is processed and sent out of the ISF after the trigger occurs. While the ISF is halted, the data in the ISF Debug FIFO is directly available to SW. Since the data in the Pre-TLVP FIFO is not directly available to SW, registers are provided to observe this data as it is single-stepped out of the ISF. After each single-step, which is done by writing *cr\_isf\_debug\_ss\_ctl\_config*, the last data read from the Pre-TLVP FIFO is stored in the *cr\_isf\_debug\_ss\_cap\_sb*, *cr\_isf\_debug\_ss\_cap\_lo*, and *cr\_isf\_debug\_ss\_cap\_hi* registers.

|  |  |  |
| --- | --- | --- |
| **Trigger**  **Configuration** | **Description** | **Register(s)** |
| TLV type | 8-bit type field in TLV word 0 | *cr\_isf\_debug\_trig\_tlv\_config.tlv\_type* |
| TLV word number | 21-bit word number of the TLV with the  word with the type field being 0 | *cr\_isf\_debug\_trig\_tlv\_config.tlv\_word\_num* |
| TLV data match | 64-bit data match value for the contents of the  TLV word | *cr\_isf\_debug\_trig\_match\_lo\_config*  *cr\_isf\_debug\_trig\_match\_hi\_config* |
| TLV data mask | 64-bit mask applied to the match value to determine which bits are looked at for the trigger. Bits of the data used for the match have the associated mask bit set to 1. | *cr\_isf\_debug\_trig\_mask\_lo\_config*  *cr\_isf\_debug\_trig\_mask\_hi\_config* |

Table : Trigger Freeze Configuration Registers

## Debug Counters

The ISF provides a local 50-bit debug counter for aggregate inbound data bytes. This count only includes non-TLV Word 0 bytes in the incoming DATA TLVs. Any user prefix data embedded in the DATA TLVs is also included. The counter value is read via *cr\_isf\_ib\_agg\_data\_bytes\_0\_count\_part0\_a* and *cr\_isf\_ib\_agg\_data\_bytes\_0\_count\_part1\_a*. This is a clear-on-read counter.

**Usage Note**

* Prior to reading the count registers, a read of the associated “global\_read” register (*cr\_isf\_ib\_agg\_data\_bytes\_global\_read.read\_strobe* is required to latch the counter values in the registers.
* Since this is a clear-on-read counter, the order in which the count registers is important. The upper 18-bits (e.g. *cr\_isf\_ib\_agg\_data\_bytes\_0\_count\_part1\_a)* must be read first and then the lower 32-bits can be read (*cr\_isf\_ib\_agg\_data\_bytes\_0\_count\_part0\_a*).

## Selectable Event Counter Support

The per-module events (see Statistics Accumulator (SA)) sent to the Selectable Event Counters are enabled by the CMD.trace bit, created by the ISF, in all of the modules. The default mode of operation is for the ISF to have this bit always follow the state of RQE.Trace. When *cr\_isf\_trace\_ctl\_en\_config. sch\_hndl\_rng\_match\_en* is set to a 1, CMD.trace bit can also be set for commands where RQE.scheduler\_handle is within the programmable range in set by *cr\_isf\_trace\_ctl\_limits\_config. sch\_hndl\_hi\_limit* and *isf\_trace\_ctl\_limits\_config. sch\_hndl\_lo\_limit*.

**Note**

There is no CMD.trace bit in the CMD TLV sent to the ISF. The ISF creates this by overwriting CMD.frame\_ctrl.unused .

## Pipeline Status Support

To support the Pipeline Status CSR in the CCEIP and CDDIP support blocks, the ISF provides 3 independent strobe signals that pulse high:

* RQE TLV received at the ISF input
* CQE TLV received at the ISF input
* CQE TLV exits the ISF

## Miscellaneous Debug Support

For debug purposes, the ISF can exert backpressure on the inbound AXI bus by setting *cr\_isf\_debug\_ctl\_config.force\_ib\_bp* to a 1.

## Statistics

Table 10 shows the statistics events the ISF provides to the Selectable Event Counters in the SA.

|  |  |  |
| --- | --- | --- |
| **Statistic Event** | **Statistic**  **Event**  **Number** | **Description** |
| ISF\_INPUT\_COMMANDS | 896 | Number of Commands that the engine receives |
| ISF\_INPUT\_FRAMES | 897 | Number of Frames that the engine receives |
| ISF\_INPUT\_STALL\_TOTAL | 898 | Number of cycles the system is stalling the engine inbound interface |
| ISF\_INPUT\_SYSTEM\_STALL\_TOTAL | 899 | Number of cycles the system is stalling the engine inbound interface while a command is active |
| ISF\_OUTPUT\_BACKPRESSURE\_TOTAL | 900 | Number of cycles the ISF outbound interface is being stalled by the downstream engine pipeline |
| ISF\_AUX\_CMD\_COMPRESS\_CTL\_MATCH\_COMMAND\_0 | 901 | Number of commands where the AUX\_CMD comp\_ctl and crypto\_ctl fields match programmable values. See Note 1. |
| ISF\_AUX\_CMD\_COMPRESS\_CTL\_MATCH\_COMMAND\_1 | 902 | Number of commands where the AUX\_CMD comp\_ctl and crypto\_ctl fields match programmable values. See Note 1. |
| ISF\_AUX\_CMD\_COMPRESS\_CTL\_MATCH\_COMMAND\_2 | 903 | Number of commands where the AUX\_CMD comp\_ctl and crypto\_ctl fields match programmable values. See Note 1. |

|  |  |  |
| --- | --- | --- |
| ISF\_AUX\_CMD\_COMPRESS\_CTL\_MATCH\_COMMAND\_3 | 904 | Number of commands where the AUX\_CMD comp\_ctl and crypto\_ctl fields match programmable values. See Note 1. |

Table : ISF Statistics Events

Note 1: The ISF has 4 sets of programmable match and mask registers to support these events and each set has an associated enable for the event. As an example, the registers for ISF\_AUX\_CMD\_COMPRESS\_CTL\_MATCH\_COMMAND\_0 are:

cr\_isf\_aux\_cmd\_ev\_match\_val\_0\_comp\_config

cr\_isf\_aux\_cmd\_ev\_match\_val\_0\_crypto\_config

cr\_isf\_aux\_cmd\_ev\_mask\_val\_0\_comp\_config

cr\_isf\_aux\_cmd\_ev\_mask\_val\_0\_crypto\_config

cr\_isf\_ctl\_config.aux\_cmd\_match0\_en

## Self-Test

As part of the self-test scheme described in 6.6, the ISF is isolated from the rest of the system by not accepting inbound traffic. To achieve this, the TVALID to the ISF is de-asserted, and TREADY back to the system is asserted. As such, the engine will not accept any inbound traffic, but will not back-pressure the system.

## Interrupts

Table 11 shows the interrupts provided by the ISF.

|  |  |
| --- | --- |
| **Interrupt** | **Description** |
| System Stall | ISF outbound interface stalled beyond a programmable value. See details below. |
| Overflow | ISF FIFO Overflow |
| Protocol Error | See 3.2.2 for details |
| TLV Parser Error | TLVP parser in the ISF is reporting an error. See 11 for details. |
| Uncorrectable ECC Error | Uncorrectable ECC error detected in ISF FIFO RAM |

Table : ISF Interrupts

In order to detect cases where the ISF outbound interface is being stalled for an extended period of time, the ISF will generate a “System Stall” interrupt when the TREADY input on the AXI4S master interface is continually de-asserted for the number of clocks configured in *cr\_isf\_system\_stall\_limit\_config.l*imit. This interrupt is enabled when *cr\_isf\_ctl\_config.sys\_stall\_en* is set to a 1.

## Engine Errors

The only engine error condition reported by the ISF in FOOTER.ErrorCode is the ISF\_PREFIX\_ERR (180) described in 3.2.3.1.

## One-Time Programmable (OTP) Inputs

The OTP inputs to the ISF are described in Table 12.

|  |  |
| --- | --- |
| **OTP Input** | **Description** |
| dbg\_cmd\_disable | When set to a 1, force the entire 32-bits of the CMD.debug\_ctl to all 0’s to prevent the use of any debug commands in the CCEIP/CDDIP |
| xp9\_disable | When set to a 1 for a CCEIP only, force CMD.comp\_ctrl.algorithm to NONE(0x0) if it is received as XPRESS9L6 (0x3). This prevents the use of the XPRESS9L6 compression algorithm. This input has no effect on a CDDIP. |

Table : ISF OTP Inputs

## Interfaces

The non-test ISF interfaces are shown in   
Table 13. With the exception of the Statistic Events, Interrupts, OTP, and Pipeline Status interfaces, which are a collection of single-bit signals, a definition of each interface type is available in Interface Descriptions.

|  |  |  |
| --- | --- | --- |
| Interface  Type | Data  Width | Notes |
| AXI4S Slave | 64 | Inbound Data |
| AXI4S Master | 64 | Outbound Data |
| RBUS In | 32 | CSR Access |
| RBUS Out | 32 | CSR Access |
| Statistic Events | 9 | To SA |
| Interrupts | 5 | To CCEIP/CDDIP support block |
| Pipeline Status | 3 | To CCEIP/CDDIP support block |
| OTP | 2 | xp9\_disable, dbg\_cmd\_disable |

Table : ISF Interfaces

# CRC Generator and Checker (CRCGC)

The CCEIP and CDDIP both require the generation and checking of CRCs. Since the various CRC applications require checking only, generation only, and dynamic switching between generation and checking on a per-frame basis, a common generator and checker block (CRCGC) will be deployed for all applications to meet all of the requirements with a single piece of IP. Tie-offs will be used to configure each CRGC for its application. The descriptions of the various applications of the CRCGC are split into separate generation (CRCG), checking (CRCC), and generation/checking (CRCGC) sections below.

## CRC Generation (CRCG)

One instance of the CRC Generator (CRCG) block is found in both the CCEIP and CDDIP.

### Features

* Calculates a CRC across the frame data in a DATA TLV
  + Supported codes: XP10 CRC64 , CRC16T, CRC64E (see 4.4)
  + Inserts CRC in FOOTER TLV
* Debug support
  + Supports debug commands found in CMD TLV
  + 8-entry CRC history buffer

### Data Flow Description

The locations of the CRCGs are shown in Figure 1 and Figure 2. Figure 8 is a block diagram of the CRCG.

**Application Summary**

* CCEIP CRCG #0 generates an XP10 CRC64 over the encrypted and compressed outbound frame data
* CDDIP CRCG #0 0 generates a configurable CRC over the decrypted and decompressed (I.e. raw) outbound frame data.



Figure : CRCG Block Diagram

**CCEIP CRCG #0**

* This CRCG receives its input from the Encrypt engine. The DATA TLV in this application contains compressed and encrypted data.
* An XP10 CRC64 (see 4.4) is computed over the frame data and inserted into FOOTER.EncCmpDataCkSum.
* The output command/frame is sent to the OSF and Decryption engine with the modifications shown in Figure 9 and Figure 10.



Figure : Simple Command Modifications for Encrypted/Compressed CRC Generation



Figure : Compound Command Modifications for Encrypted/Compressed CRC Generation

**CDDIP CRCG #0**

* This CRCGC receives its input from the Crypto CKMIC Checker. The DATA TLV in this application contains decrypted and decompressed data (i.e. raw data)
* Regardless of the values of CMD.FRMD\_OUT\_TYPE and CMD.MD\_TYPE, all 3 CRCs (XP10 CRC64, CRC64E, and CRC16T) (see 4.4) are computed over the frame data.
* The operation of this CRCG is described in Table 14. It uses the CMD fields shown to determine which CRC gets inserted into which FOOTER field.

|  |  |  |  |
| --- | --- | --- | --- |
| CMD.  FRMD\_OUT\_TYPE | CMD.  MD\_TYPE (Value) | Footer Field  Used for  CRC | CRC  Inserted |
| FRMD\_USER\_NULL | N/A | N/A | None |
| FRMD\_USER\_PI16 | N/A | NVMERawCkSum | CRC16T |
| FRMD\_USER\_PI64 | CRC64(2) | RawDataCkSum | XP10 CRC64 |
| FRMD\_USER\_PI64 | CRC64E(3) | RawDataCkSum | CRC64E |
| FRMD\_USER\_VM | N/A | RawDataCkSum | XP10 CRC64 |

Table : CDDIP CRCG #0 Function Table

* The output command/frame is sent to the CG and OSF via an AXI4S master interface (Table 42) with the modifications shown in Figure 20 and Figure 21 for the FRMD\_USER\_PI64 and FRMD\_USER\_VM TLVs, and Figure 11 and Figure 12 for a FRMD\_USER\_PI16 TLV. No modifications are made for a FRMD\_USER\_NULL TLV.



Figure : Simple Command Modifications for CRC Generation for FRMD\_USER\_PI16



Figure : Compound Command Modifications for CRC Generation for FRMD\_USER\_PI16

### Debug Support

#### Debug Command

The CRCG supports the Data Corruption debug command that is described in 11.2.5.1.

#### CRC History Buffer

The CRCG keeps a buffer of the CRCs for the last 8 data frames. The DATA.SequenceNum, DATA.FrameNum for compound commands, an integrity check error bit (0: no error; 1: error), a check enable bit, and a valid bit are stored along with the CRC. The format of each entry is shown in Table 15. CheckEn is used to identify data frames where the integrity check function was done. Since this is not the case for CRCG applications, the CheckEn and Error fields are always 0. The FrameNum field will be 0 for all simple commands.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Bits** | 91:87 | 86 | 85 | 84 | 83:72 | 71:64 | 63:0 |
| **Field** | Reserved | Valid | Error | CheckEn | FrameNum | SeqNum | CRC |

Table : CRC History Buffer Entry Format

### Statistics

The CRCG statistic events are shown in Table 32 and Table 33 for both the CCEIP and CDDIP.

### Interrupts

The CRCG only provides a TLVP Parser Error interrupt. See 11 for details.

### Engine Errors

The CRCG does not create any engine errors.

### Interfaces

The CRCG interfaces are shown in Table 19.

## CRC Checking (CRCC)

The CRC Checker (CRCC) block is found in the both the CCEIP and CDDIP. Two instances are used in the CCEIP and one instance is used in the CDDIP.

### Features

* Calculates a CRC across the frame data and compares it to a CRC in the FOOTER TLV
  + Supported codes: XP10 CRC64 , CRC16T, CRC64E (see 4.4)
  + The FOOTER CRC(s) to use is configurable for multiple applications via tie-offs, and also depends on the FRMD type and CMD.MD\_TYPE.
  + Insert a CRCC error into FOOTER.ErrorCode if integrity check fails and no prior errors reported.
* Interfaces to the SA
* Debug support
  + Supports debug commands in CMD.CCEIP\_CDDIP\_Debug
  + 8 entry CRC history buffer including an error bit to identify data frames that failed the CRC check

### Data Flow Description

The locations of the CRCCs in the CCEIP and CDDIP are shown in Figure 1 and Figure 2. Figure 13 is a block diagram of the CRCC. Tie-offs are used to configure it for each application.



Figure : CRCC Block Diagram

**Application Summary**

* CCEIP CRCC #0 performs an integrity check on the compressed and encrypted outbound data. The operation of this CRCC is very similar to CDDIP CRCC #0. The only difference is the source and format of the inbound command/compressed data frame.
* CCEIP CRCC #1 performs an integrity check on the decrypted and decompressed data frame (i.e. raw data)
* CDDIP CRCC #0 performs an integrity check on the compressed and encrypted inbound data frame

**CCEIP CRCC #1**

* This CRCC receives its input from the XP10 Decompression engine. The DATA TLV in this application contains the decompressed and decrypted data that is supposed to match the raw data that was originally sent to the CCEIP. Figure 14 and Figure 15 show stack-ups for this CRCC.
* This CRCC computes 3 CRCs over the frame data (XP10 CRC64, CRC64E, and CRC16T). The selection of which CRC(s) is/are checked against the values in the FOOTER is dependent upon the FRMD type associated with the frame and CMD.MD\_TYPE. Table 16 shows all of the possible cases. The XP10 CRC64 and CRC64E are checked against FOOTER.RawDataCkSum, while the CRC16T is checked against FOOTER.NVMERawCkSum.

An error is inserted in FOOTER.ErrorCode if there is a mismatch and this is the “first error”. Regardless of whether it is the first error in the command, the “integrity check error” bit will be set in the Error register in the CRCC and an “integrity check error” event is sent to the SA.

**Note:**

For FRMD\_USER\_PI16 frames, an error is inserted if either of the 2 CRC checks fails.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| FRMD Type | MD\_TYPE | XP10  CRC64 | CRC64E | CRC16T | Comments |
| FRMD\_USER\_NULL | N/A | Y |  |  | No NVME MD |
| FRMD\_USER\_PI16 | N/A | Y |  | Y | Both CRCs checked |
| FRMD\_USER\_PI64 | CRC64 | Y |  |  |  |
| FRMD\_USER\_PI64 | CRC64E |  | Y |  |  |
| FRMD\_USER\_VM | N/A | Y |  |  |  |

Table : CCEIP CRCC #0 Checking Cases

* The output command/frame is sent to the CG engine with the only modification being the reporting of any first errors in FOOTER.ErrorCode and the associated frame number in FOOTER.ErroredFrameNumber.



Figure : Simple Command Raw CRC Checker Stack-up



Figure : Compound Command Raw CRC Checker Stack-up

**CDDIP CRCC #0**

* This CRCC receives its input from the ISF. The DATA TLV in this application contains compressed and encrypted data. For this CRCC, the stack-ups for frames without “user-prefix” data is the same as that shown in Figure 14 and Figure 15, while the stack-ups for frames with “user-prefix” data are shown in Figure 16 and Figure 17.
* An XP10 CRC64 (see 4.4) is computed over the frame data and checked against FOOTER. EncCmpDataCkSum. An error is inserted in FOOTER.ErrorCode if there is a mismatch and this is the “first error”. Regardless of whether it is the first error in the command, the “integrity check error” bit will be set in the Error register in the CRCC and an “integrity check error” event is sent to the SA.
* The output command/frame is sent to the Prefix Attach engine with the only modification being the reporting of any first errors in FOOTER.ErrorCode and the associated frame number in FOOTER.ErroredFrameNumber.



Figure : CDDIP Simple Command Compressed/Encrypted CRC Checker Stack-up with Prefix TLV



Figure : CDDIP Compound Command Compressed/Encrypted CRC Checker Stack-up with Prefix TLV

**CCEIP CRCC #0**

* This CRCC receives its input from CRCG #0. The DATA TLV in this application contains compressed and encrypted data. Figure 18 and Figure 19 show the stack-ups for this CRCC. The operation of this CRCC is similar to that for CDDIP CRCC #0. The output is sent to the Decryption engine.



Figure : CCEIP Simple Command Compressed/Encrypted CRC Checker Stack-up



Figure : CCEIP Compound Command Compressed/Encrypted CRC Checker Stack-up

### Statistics

The CRCC provides an “integrity check error” signal to the SA which pulses each time an integrity check fails.

### Debug Support

#### Debug Command

The CRCC supports the Data Corruption debug command that is described in 11.2.5.1.

#### CRC History Buffer

The CRCC keeps a buffer of the CRCs for the last 8 data frames. This buffer is the same as the one discussed in 4.1.3.2 for the CRCC. The only difference for a CRCC is that the CheckEn bit, used to identify data frames where the integrity check function was done, is always set, and the Error field will be set for any data frames that failed the CRC check.

### Statistics

The CRCC statistic events are shown in Table 32 and Table 33 for both the CCEIP and CDDIP.

### Interrupts

The CRCC only provides a TLVP Parser Error interrupt. See 11 for details.

### Engine Errors

CRC errors are reported by the CRCC. The engine error codes for each instance of the CRCC are shown in Table 17.

|  |  |
| --- | --- |
| Instance | Error Code  (Decimal) |
| CCEIP CRCC #0 | 51 |
| CCEIP CRCC #1 | 52 |
| CDDIP CRCC #0 | 56 |

Table : CRCC Error Codes

### Interfaces

The CRCC interfaces are shown in Table 19.

## CRC Generation and Checking (CRCGC)

The CRCGC has all of the features found in both the CRCC and CRCG and some additional ones. The unique features of this application are:

* Switches between generation and checking on a per-frame basis depending on the FRMD type and CMD.MD\_TYPE, as described below
* Supports the generation of 4 CRC types for the FOOTER.RawDataCkSumProtocol (XP10 CRC64, XP10 CRC32, GZIP CRC32, and Adler-32, as shown in 4.4) used by the Huffman Compress engine. The standalone CRCG and CRCC applications do not generate or check the FOOTER.RawDataCkSumProtocol.

The CRCGC is used only in the CCEIP and the location is shown in Figure 1.

### Data Flow Description

**CCEIP CRCGC #0**

* This CRCGC receives its input from the ISF. The DATA TLV in this application contains raw data.
* Whether the CRCGC is a generator or checker depends on the FRMD type and CMD.MD\_TYPE, as described in Table 18. For all FRMD types except FRMD\_USER\_NULL, FRMD\_USER\_PI64 with CMD.MD\_TYPE=0 or 3, and FRMD\_USER\_VM with CMD.MD\_TYPE=2 or 3, it checks the FOOTER field shown in the table. For FRMD\_USER\_NULL, FRMD\_USER\_PI64 with CMD.MD\_TYPE=0 or 3, and FRMD\_USER\_VM with CMD.MD\_TYPE=2 or 3,it is a generator.

Frames of type FRMD\_USER\_PI16 are a special case. In addition to checking the CRC in FOOTER.NVMERawCkSum, it will always generate and append an XP10 CRC64 to provide the highest quality data integrity check when the frame reaches the CCEIP CRCC #0. The XP10 CRC64 is always inserted in FOOTER.RawDataCkSum.

When this instance is operating as a CRCC, an error is inserted in FOOTER.ErrorCode if there is a mismatch and this is the “first error”. Regardless of whether it is the first error in the command, the “integrity check error” bit will be set in the Error register and an “integrity check error” event is sent to the SA.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FRMD Type | CMD.  MD\_TYPE | Footer Field  Checked | CRC  Checked  For  CRCC | XP10  CRC64  Inserted in  RawDataCkSum |
| FRMD\_USER\_NULL | N/A | None | N/A | Y |
| FRMD\_USER\_PI16 | N/A | NVMERawCkSum | CRC16T | Y |
| FRMD\_USER\_PI64 | 1 (CRC64) | RawDataCkSum | XP10 CRC64 | N |
| FRMD\_USER\_PI64 | 2 (CRC64E) | RawDataCkSum | CRC64E | N |
| FRMD\_USER\_PI64 | 0,3 | None | N/A | Y |
| FRMD\_USER\_VM | 0,1 | RawDataCkSum | XP10 CRC64 | N |
| FRMD\_USER\_VM | 2,3 | None | N/A | Y |

Table : CCEIP CRCGC Function Table

* This CRCGC also generates the compressed data format CRC used by the Huffman Compress Engine and inserts it in FOOTER.RawDataCkSumProtocol. It inserts 1 of 4 possible CRCs based on CMP.CompAlgorithm and CMD.XP10CrcMode, as described in Table 22 . This adds 3 additional CRC generators to this CRCGC (XP10 CRC32, GZIP CRC32, and Adler-32).
* The output command/frame is sent to the Prefix engine via an AXI4S master interface (Table 42) with the modifications shown in Figure 20, Figure 21, Figure 22, Figure 23, Figure 24, and Figure 25. For each of the figures, there is a corresponding stack-up for frames with “user-prefix” data that is not shown. The modifications for CRC checking for non-FRMD\_USER\_PI16 frames is the same as that shown in Figure 14 and Figure 15.



Figure : Simple Command Modifications for FRMD\_USER\_PI64 (CMD.MD\_TYPE {1,2}) and FRMD\_USER\_VM



Figure : Compound Command Modifications for FRMD\_USER\_PI64 (CMD.MD\_TYPE {1,2}) and FRMD\_USER\_VM



Figure : Simple Command Modifications for FRMD\_USER\_NULL and FRMD\_USER\_PI64 (CMD.MD\_TYPE {0,3})



Figure : Compound Command Modifications for FRMD\_USER\_NULL and FRMD\_USER\_PI64 (CMD.MD\_TYPE {0,3})



Figure : Simple Command Modifications for FRMD\_USER\_PI16



Figure : Compound Command Modifications for FRMD\_USER\_PI16

### Statistics

The following table shows the statistics events the CRCGC provides to the Selectable Event Counters in the SA.

|  |  |  |
| --- | --- | --- |
| **Statistic Event** | **Statistic**  **Event**  **Number** | **Description** |
| cts\_raw\_data\_cksum\_good\_se | 64 | Number of raw data checksums that matched |
| cts\_raw\_data\_cksum\_err\_se | 65 | Number of raw data checksums that failed |
| cts\_crc64e\_cksum\_good\_se | 66 | Number of CRC64e checksums that matched |
| cts\_crc64e\_cksum\_err\_se | 67 | Number of CRC64e checksums that failed |
| cts\_enc\_cmp\_data\_cksum\_good\_se | 68 | Number of enccmp checksums that matched |
| cts\_enc\_cmp\_data\_cksum\_err\_se | 69 | Number of enccmp checksums that failed |
| cts\_nvme\_raw\_cksum\_good\_se | 70 | Number of NVME raw data checksums that matched |
| cts\_nvme\_raw\_cksum\_err\_se | 71 | Number of NVME raw data checksums that failed |

### Debug Support

#### Debug Command

The CRCGC supports the Data Corruption debug command that is described in 11.2.5.1.

### Statistics

The CRCGC statistic events are shown in Table 32 and Table 33 for both the CCEIP and CDDIP.

### Interrupts

The CRCGC only provides a TLVP Parser Error interrupt. See 11 for details.

### Engine Errors

CRC errors are reported by the CRCGC using engine error code 54 (decimal).

### Interfaces

The CRCGC interfaces are shown in Table 19. A definition of each interface type is available in Interface Descriptions.

|  |  |  |
| --- | --- | --- |
| Interface  Type | Data  Width | Notes |
| AXI4S Slave | 64 | Inbound Data |
| AXI4S Master | 64 | Outbound Data |
| RBUS In | 32 | CSR Access |
| RBUS Out | 32 | CSR Access |
| Interrupts | 1 | To CCEIP/CDDIP support block |
| Statistic Events | 8 | To SA |

Table : CRCGC Interfaces

## CRC Polynomials

The XP10 CRC64, XP10 CRC32, CRC64-ECMA-182, CRC16T, and GZIP CRC32 polynomials used in the CRCGC, CRCC, and CRCG are described below in Table 20. Since Adler-32 is not a CRC there is no polynomial.

|  |  |
| --- | --- |
| CRC Type | Polynomial |
| XP10 CRC64 | 0x9a6c\_9329\_ac4b\_c9b5 |
| CRC64-ECMA-182 | 0x42F0\_E1EB\_A9EA\_3693 |
| XP10 CRC32 | 0x82f6\_3b78 |
| 16CRCT | 0x8bb7 |
| GZIP CRC32 | 0xedb8\_8320 |

Table : CRC Polynomials

# Completion Generator (CG)

## Features

* Creates the FRMD, STATS and GUID TLVs sent to the OSF
* Creates the “final” CQE TLV sent to the OSF
* Configurable for CDDIP/CCEIP applications via tie-off

## Operational Description

For every command processed by the CCEIP or CDDIP, the Completion Generator (CG) provides the Post-data TLVs (GUID, FRMD, STATS and CQE TLVs) that are sent to the OSF via an outbound AXI4S master interface. Additionally, for each frame that is part of a compound command, the CG provides the FRMD TLV. The locations of the CG in the CCEIP and CDDIP are shown in Figure 1 and Figure 2.

### Command TLV Modifications

In both the CCEIP and CDDIP, the CG modifies the TLV stack-up on the commands it receives from the compress/encrypt or decrypt/decompress pipelines for simple commands such that only the CQE TLV remains, but is modified, and the GUID (optional), FRMD (out type), and STATS are created, as shown in Figure 26. Figure 27 shows the modifications made for compound command frames. The CQE TLV received after the last frame of a compound command is handled the same as a simple command. The GUID TLV is created before the first FRMD of a compound command, just like a simple command.



Figure : CG TLV Modifications for Simple Commands



Figure : CG TLV Modifications for Compound Command Frames

#### GUID TLV

The GUID TLV is created before the first FRMD of the command by adding a TLV Word 0 to FOOTER.GUID. The creation of the GUID TLV is optional. It is only created when CMD. dst\_aux\_frame\_data\_guid\_present is set to a 1.

#### FOOTER TLV Transformation to FRMD TLV

The CG receives a full-size footer that includes all of the fields that are in the mini-footer created by the ISF (see Table 5) and the GUID and IV fields added by the Crypto CKMIC modules in the CCEIP and CDDIP. The format of the full-size FOOTER TLV is shown in Table 21. The CG receives it fully populated and transforms it to a FRMD TLV per FOOTER.GenFRMDOutType (see [1] for a definition of all of the FRMD types). The transformation of the FOOTER TLV to the FRMD TLV primarily involves copying fields from the FOOTER to the FRMD. Some fields can vary in size between the FOOTER and the FRMD to shorten the length of the FRMDs. Table 23 and Table 24 describe how all of the FRMD fields are created for both the CCEIP and CDDIP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | | **Size** | **Word** | **Notes** |
| Type | | 8 | 0 | Type = 6 |
| Length | | 8 | 0 | Length = 28 (Length of mini-FOOTER from ISF) |
| Sequence Number | | 8 | 0 | Increments per RQE/Command |
| Engine ID | | 4 | 0 | 0-3 CCEIP, 4-7 CDDIP |
| Reserved | | 4 | 0 | Reserved |
| Frame Number | | 11 | 0 | 0 – Simple Command  0-2047 for Compound Commands |
| Reserved | | 1 | 0 | Reserved |
| Status | Coding | 2 | 0 | 0: Raw  1: Parsable  2: XP10CFH 4K  3: XP10CFH 8K |
| EncCmpDataMacSize | 2 | 0 | 0: 64 bits valid (upper 64 bits of the field)  1: 128 bits valid (upper 128 bits of the field)  2: 256 bits valid (full size of the field)  3: Reserved |
|  | RawDataMACSize | 2 | 0 | 0: 64 bits valid (upper 64 bits of the field1: 128 bits valid (upper 128 bits of the field)  2: 256 bits vaelid (full size of the field)  3: Reserved |
|  | Reserved | 2 | 0 | Reserved |
| GenFRMDOutType | | 8 | 0 |  |
| Reserved | | 2 |  |  |
| Parity | | 2 | 0 | BIP-2 (Even) |
| GUID | | 256 | 1-4 |  |
| IV | | 128 | 5-6 | Big Endian format with bits [127:64] in Word5 |
| RawDataMAC | | 256 | 7-10 | SHA2, SHA2-HMAC (Big Endian format with bits [255:192] in Word7) |
| RawDataCkSum | | 64 | 11 | CRC-64 |
| RawDataCkSumProtocol | | 64 | 12 | CRC added to the compressed data in the Huffman Encoder for XP10, GZIP, and ZLIB,. CRC can be one of has 4 different types per the protocol specified in CMD. comp\_ctl.algorithm and the CRC specified in CMD. comp\_ctl.xp10\_crc\_mode and described in Table 18. |
| EncCmpDataMac | | 256 | 13-16 | SHA2, SHA2-HMAC, AES-GMAC (Big Endian format with bits [255:192] in Word13) |
| EncCmpDataCkSum | | 64 | 17 | CRC-64 |
| Bytes Out | | 24 | 18 | Bytes Out |
| Bytes In | | 24 | 18 | Bytes In (xfer’d over PCIe) (ISF updates) |
| NVMERawCkSum | | 16 | 18 | CRC-16T |
| Errored Frame Number | | 11 | 19 | 0 for simple commands; otherwise has the first frame in the compound command that generated the error. |
| Reserved | | 1 | 19 | Reserved |
| Error Code | | 8 | 19 | Engine error code (see [1] for a complete list) |
| Compressed Length | | 24 | 19 | Compressed Size (Huffman Encoder builds) |
| Reserved | | 20 | 19 | Reserved |

Table : FOOTER TLV Format

|  |  |  |
| --- | --- | --- |
| **CMD. CompAlgorithm** | **CMD.XP10CrcMode** | **CRC Selected** |
| 0 (Passthrough) | Don’t care | None |
| 1 (ZLIB) | Don’t care | Adler-32 |
| 2 (GZIP) | Don’t care | GZIP CRC32 (see Table 20) |
| 3 (XP9) | Don’t care | None |
| 4 (XP10) | 0 (CRC32) | XP10CRC32 |
| 4 (XP10) | 1 (CRC64) | XP10CRC64 |
| 5 (XP10CHU4K) | 0 (CRC32) | XP10CRC32 |
| 5 (XP10CHU4K) | 1 (CRC64) | XP10CRC64 |
| 6 (XP10CHU8K) | 0 (CRC32) | XP10CRC32 |
| 6 (XP10CHU8K) | 1 (CRC64) | XP10CRC64 |

Table : Function Table for RawDataCkSumProtocol Footer Field

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Source of Field Values for Supported FRMD Types (CCEIP)** | | | | |
|  | **FRMD**  **INT**  **APP** | **FRMD**  **INT**  **SIP** | **FRMD**  **INT**  **LIP** | **FRMD**  **INT**  **VM SHORT** | **FRMD**  **INT**  **VM LONG** |
| Type | 15 | 16 | 17 | 22 | 18 |
| Length | 13 | 7 | 13 | 19 | 25 |
| Engine ID | From FRMD | Same As APP | Same As APP | Same As APP | Same As APP |
| Sequence Num | From FRMD | Same As APP | Same As APP | Same As APP | Same As APP |
| Frame Num | From FRMD | Same As APP | Same As APP | Same As APP | Same As APP |
| Reserved | All 0’s | Same As APP | Same As APP | Same As APP | Same As APP |
| Parity | CG Gen | Same As APP | Same As APP | Same As APP | Same As APP |
| Status | FOOTER.  Status | Same As APP | Same As APP | Same As APP | Same As APP |
| Reserved | All 0’s | All 0’s | All 0’s | All 0’s | All 0’s |
| Compressed  Length | FOOTER.  CompressedLength | Same As APP | Same As APP | Same As APP | Same As APP |
| CmpEncCkSum | FOOTER.  EncCmpDataCkSum | Same As APP | Same As APP | Same As APP | Same As APP |
| CmpEncMAC (Big Endian) | FOOTER.  EncCmpDataMac  [255:128] | N/A | N/A | FOOTER.  EncCmpDataMac  [255:0] | FOOTER.  EncCmpDataMac  [255:0] |
| IVTWEAK (Big Endian) | FOOTER.  IV[127:32]  Note 1 | N/A | N/A | FOOTER.  IV[127:0] | FOOTER.  IV[127:0] |
| RawDataMAC (Big Endian) | N/A | FOOTER.  RawDataMAC  [255:192] | FOOTER.  RawDataMAC  [255:0] | FOOTER.  RawDataMAC  [255:192] | FOOTER.  RawDataMAC  [255:0] |

Table : CCEIP Output FRMD Creation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Source of Field Values for Supported FRMD Types (CDDIP)** | | | |
|  | **FRMD**  **USER**  **NULL** | **FRMD**  **USER**  **PI16** | **FRMD**  **USER**  **PI64** | **FRMD**  **USER**  **VM** |
| Type | 11 | 12 | 13 | 14 |
| Length | 2 | 4 | 4 | 16 |
| Engine ID | From FRMD | Same As Null | Same As Null | Same As Null |
| Sequence Num | From FRMD | Same As Null | Same As Null | Same As Null |
| Frame Num | From FRMD | Same As Null | Same As Null | Same As Null |
| Reserved | All 0’s | Same As Null | Same As Null | Same As Null |
| Parity | CG Gen | Same As Null | Same As Null | Same As Null |
| NVME MetaData CRC16T | N/A | FOOTER.  NVMERawCkSum | N/A | N/A |
| NVME Metadata CRC64 | N/A | N/A | FOOTER.  RawDataCkSum | N/A |
| IVTWEAK | N/A | N/A | N/A | FOOTER.  IV[127:0] |
| RawDataCkSum | N/A | N/A | N/A | FOOTER.  RawDataCkSum |
| RawDataMAC  (Big Endian) | N/A | N/A | N/A | FOOTER.  RawDataMAC  [255:0] |
|  |  |  |  |  |

Table : CDDIP Output FRMD Creation

**Notes**

1. FRMD.IVTWEAK is 96-bits. Populate FOOTER.IV[127:32] with FRMD.IVTWEAK and FOOTER.IV[31:0]=0.

**Error Handling**

The CG has the ability to detect and accommodate TLV misconfigurations where the FOOTER.GenFRMDOutType, which is taken from **CMD. comp\_ctl.** **frame\_ctl.**

**out\_type**, is an undefined value. It flags these cases as errors by inserting a CG\_UNDEF\_FRMD\_OUT (170) in **CQE.engine\_error\_code** if the command did not arrive at the CG with any pre-existing errors. as described in 5.2.1.4. Additionally, and regardless of whether this error will be reported in the CQE, the CG takes the following action:

* CCEIP creates a INT APP FRMD (0xF TLV type)
* CDDIP creates a USER NULL FRMD (0xB TLV Type)

#### STATS TLV

The CG creates the STATS TLV, and the format is shown in Table 25. For simple commands, STATS.bytes\_in comes directly from FOOTER.BytesIn. For compound commands, the CG accumulates FOOTER.BytesIn for each frame and inserts the final sum into STATS.bytes\_in.

The source of STATS.bytes\_out is different for the CCEIP and CDDIP. For the CCEIP, FOOTER.CompressedLength is used, while FOOTER.BytesOut is used for the CDDIP. For simple commands, STATS.bytes\_out comes directly from the FOOTER values. For compound commands, the CG accumulates the FOOTER values for each frame and inserts the final sum into STATS.bytes\_out.

The value of STATS.cmd\_type is determined by the value of CQE.status\_code. If it is 0x0, which indicates that the command has no errors, STATS.cmd\_type is set to 0. Otherwise, is it set to 1, which indicates that the command has an error. The final value of STATS.clocks is added in the OSF (see 6.2.2), so it is just set to 0x0 in the CG.

|  |  |  |  |
| --- | --- | --- | --- |
| **Fields** | **Size** | **Word** | **Notes** |
| Type | 8 | 0 | Value for STATS TLV is 0x08 |
| Length | 8 | 0 | 0x06 |
| Sequence Num | 8 | 0 | Increments per RQE/Command (copied from FOOTER TLV) |
| Engine ID | 4 | 0 | 0-3 CCEIP, 4-7 CDDIP (copied from FOOTER TLV) |
| Reserved | 4 | 0 |  |
| Frame Num | 11 | 0 | Copied from FOOTER TLV  0 – Simple Command  0-2047 for Compound Commands |
| Reserved | 19 | 0 |  |
| Parity | 2 | 0 | BIP-2 (Even) |
| Bytes\_Out | 24 | 1 | Aggregate output payload bytes of the command submitted. |
| Reserved | 8 | 1 |  |
| Bytes\_In | 24 | 1 | Aggregate input payload bytes of the command submitted. This includes User Prefix Data but not Predetermined Prefix Data |
| Reserved | 8 | 1 |  |
| Clocks | 24 | 2 | Number of clock cycles that this command takes to leave the engine. It includes from the first beat of the RQE to the last beat of the FRMD. This is inserted in the STATS TLV in the OSF. The CG inserts 0x0 here. |
| Reserved | 8 | 2 |  |
| CMD\_Type | 1 | 2 | 0: Command has no errors  1: Command had an error |
| Reserved | 31 | 2 |  |

Table : STATS TLV Format

#### CQE TLV

The CG updates the CQE error related fields described in [1]. Commands can arrive at the CCEIP/CDDIP with errors (i.e. CQE.status\_code not 0x0) or the CCEIP/CDDIP pipeline blocks can detect errors during the processing of the commands (i.e. engine errors). If a command arrives with an error, the CG does not update CQE.status\_code and sets CQE.engine\_error\_code and CQE.error\_frame to 0x0, regardless of whether the pipeline blocks report an error. If the CCEIP/CDDIP report an error without an incoming error, the CG updates CQE.status\_code to ENGINE\_ERR (199) and sets CQE.engine\_error\_code and CQE.error\_frame to the values described below (see [1] for the complete list of engine errors and status codes).

When the pipeline blocks detect errors for a frame, they all report them in the FOOTER. The CCEIP and CDDIP use a “First Error” scheme that is described in [1] and results in only the first error (i.e. most upstream error) found in the pipeline being reported in the FOOTER that arrives at the CG. Errors that ripple down from upstream blocks to downstream blocks are not reported.

The CG examines the FOOTER.ErrorCode for each frame and stores it and FOOTER.ErroredFrameNum if the ErrorCode is !=0x0. To accommodate compound commands, once an error occurs, no further errors are stored for subsequent frames that are part of the same compound command. When the CQE TLV arrives at the CG, it inserts these stored values into CQE.engine\_error\_code and CQE.error\_frame.

## Debug Support

There is no debug support provided in the CG. The Post-data TLV FIFO in the OSF serves as a history buffer.

## Statistics

Table 26 shows the statistics events the CG provides to the Selectable Event Counters in the SA.

|  |  |  |
| --- | --- | --- |
| **Statistic Event** | **Statistic**  **Event**  **Number** | **Description** |
| CG\_CQE\_OUTPUT\_COMMAND | 99 | Number of CQE TLVs that the engine outputs |
| CG\_SYSTEM\_ERROR\_COMMAND | 98 | Number of commands with system related errors that the engine outputs |
| CG\_SELECT\_ENGINE\_ERROR\_COMMAND | 97 | Number of commands with programmable engine related error codes that the engine outputs. The error code and associated mask are configured in the CR\_CG\_DEBUG\_CTL\_CONFIG register |
| CG\_ENGINE\_ERROR\_COMMAND | 96 | Number of commands with engine related errors that the engine outputs |

Table : CG Statistics Events

## Interrupts

The CG only provides a TLVP Parser Error interrupt. See 11 for details.

## Engine Errors

The only engine error condition reported by the CG is CG\_UNDEF\_FRMD\_OUT (170) which is described in 5.2.1.2

## Interfaces

The CG interfaces are shown in Table 27. With the exception of the Statistic Events and Interrupts, which are a collection of single-bit signals, a definition of each interface type is available in Interface Descriptions.

|  |  |  |
| --- | --- | --- |
| Interface  Type | Data  Width | Notes |
| AXI4S Slave | 64 | Inbound Data (Shared Interface. See Note 1) |
| AXI4S Master | 64 | Outbound Data |
| RBUS In | 32 | CSR Access |
| RBUS Out | 32 | CSR Access |
| Statistic Events | 4 | To SA |
| Interrupts | 1 | To CCEIP/CDDIP support block |

Table : CG Interfaces

**Notes**

1. In both the CCEIP and CDDIP, the inbound slave interface is also shared with another slave interface. As such, the CG must know the state of the other interface’s TREADY signal to know when it is back pressuring. The CG has logic to de-assert the TVALID signal from the master whenever the other interface’s TREADY signal is de-asserted.

# Outbound Stream FIFO (OSF)

## Features

* Dual data interfaces
  + Encrypted/Compressed data (CCEIP) or Decrypted/Decompressed data(CDDIP)
  + CG (CCEIP/CDDIP)
* Outbound commands modified so only consist of the DATA, FRMD, GUID (optional), STATS and CQE TLVs. All others are discarded.
* Provides 2 output buffers that are multiplexed to create the outbound AXI4S interface
  + 512 x 92 FIFO for RQE and DATA TLVs
  + 256 x 92 FIFO for Post-data TLVs which consist of the GUID (optional), FRMD, STATS, and CQE TLVs
  + Outbound order is RQE-> DATA-> PDT
* Inserts “clocks” field in STATS TLV
* Supports halting of outbound AXI4S interface when CCEIP/CDDIP generates an interrupt
* Provides the self-test control used to isolate the CCEIP/CDDIP form the rest of the system during Crypto self-test
* SA Interface
  + Statistic Events
* Debug Support
  + Debug Counters
  + Debug hooks to support SW control of the data flow out of the CCEIP and CDDIP
    - Single step, etc
  + Interfaces to CCEIP and CDDIP support blocks to provide pipeline status information

## Description

The locations of the OSF in the CCEIP and CDDIP are shown in Figure 1 and Figure 2. A block diagram of the OSF is shown in Figure 28.



Figure : OSF Block Diagram

### Dataflow

The OSF drives both the CCEIP and CDDIP outbound traffic via the primary 64-bit AXI4S master interface. The outbound traffic consists of the RQE and DATA TLVs (i.e. compressed/encrypted data for the CCEIP and decompressed/decrypted data for the CDDIP) and the associated Post-data TLVs, which consist of the GUIID (optional), FRMD, STATS, and CQE TLVs, as shown in Figure 33. The OSF provides separate FIFOs for the RQE/DATA and PDT traffic and multiplexes between them to create the outbound traffic. The outbound order is RQE-> DATA-> PDT for simple commands and RQE-> DATA-> PDT-> DATA-> PDT.... DATA-> PDT for compound commands.

A 512 x 92 FIFO is used to buffer the RQE/DATA TLVs and a 256 x 92 FIFO is used to buffer the Post-data TLVs. The format of each entry is the same as that for the ISF for both FIFOs and is shown in Table 1.

For the CCEIP, there is a significant delay between the arrival of the compressed/encrypted DATA TLV at the OSF for the current command and the arrival of the Post-data TLVs because the frame data has to be decrypted and decompressed in the validator pipeline before these can be provided. This delay is much less in the CDDIP.

### Command TLV Modifications

In both the CCEIP and CDDIP, the OSF modifies the TLV stack-up on the commands it receives from the compress/encrypt or decrypt/decompress pipeline such that only the RQE and DATA TLVs remain, as shown in the 4 figures below. The OSF does not modify the stack-up it receives from the CG for the Post-data TLVs.



Figure : CCEIP OSF TLV Modifications for Simple Commands



Figure : CCEIP OSF TLV Modifications for Compound Command Frames



Figure 31 : CDDIP OSF TLV Modifications for Simple Commands



Figure : CDDIP OSF TLV Modifications for Compound Command Frames

Figure 33 shows the complete OSF output TLV stack-up for simple and compound commands.



Figure : CCEIP/CDDIP OSF Output Stack-Up

In addition to the modifications mentioned above, the OSF also inserts the final value of STATS.clocks. This field is the number of clocks from the SOT of the RQE to the EOT of the FRMD at the output of the OSF. This includes any idle cycles or cycles where the OSF is being back-pressured.

## Debug Support

### Dataflow Control

The OSF has the debug support features described below for both FIFOs that allow SW to control the flow of data out of the CCEIP and CDDIP. The various modes are controlled via *cr\_osf\_data\_fifo\_debug\_ctl\_config.debug\_mode,* and *cr\_osf\_pdt\_fifo\_debug\_ctl\_config.debug\_mode.* The values for the modes described below are included with the heading. For non-debug operation, this field is set to NORMAL (0x0).

**SW Data Entry (RW\_DISABLE 0x1)**

Load OSF via SW and then transfer the entire contents to the system in real time via SW enable.

Both FIFOs must be put back in NORMAL mode to transfer the entire contents of the FIFOs to the system.

**DMA Debug Buffer (RD\_DISABLE 0x2)**

In this mode, the OSF holds the output data for inspection by SW.

Both FIFOs must be put back in NORMAL mode to transfer the entire contents of the FIFOs to the system.

**Single Step (SINGLE\_STEP 0x3)**

Load OSF via SW and single step the transfer of this data to the system via SW. The OSF can also single step the real-time contents of the OSF FIFOs captured while in the DMA Debug Buffer mode described above.

When using this mode, there are ordering considerations to be aware of with respect to the 2 FIFOs. The DATA FIFO is singled stepped first to the end of the DATA TLV and then the PDT FIFO is single stepped next until the CQE for simple commands or until the end of the FRMD for non-last data frames of a compound command. For compound commands, the procedure is repeated until the last frame of a compound command which also includes the STATS and CQE TLVs. At any time in the process, both FIFOs can be put in NORMAL mode to send the rest of the contents to the system.

### Debug Counters

The OSF provides the following 50-bit local debug counters that are accessible via SW. Both are clear-on-read counters.

* Aggregate outbound data bytes (DATA TLV payload not including TLV word 0) for the outbound traffic (*cr\_osf\_ob\_agg\_data\_bytes\_0\_count\_part1\_a* and *cr\_osf\_ob\_agg\_data\_bytes\_0\_count\_part0\_a*)
* Aggregate outbound command count (*cr\_osf\_ob\_agg\_frame\_0\_count\_part1\_a* and *cr\_osf\_ob\_agg\_frame\_0\_count\_part0\_a*)

**Usage Notes**

* Prior to reading the count registers, a read of the associated “global\_read” register (*cr\_osf\_ob\_agg\_data\_bytes\_global\_read.read\_strobe* or *cr\_osf\_ob\_agg\_frame\_global\_read.read\_strobe*) is required to latch the counter values in the registers.
* Since these are clear-on-read counters, the order in which the count registers is important. The upper 18-bits (e.g. *cr\_osf\_ob\_agg\_data\_bytes\_0\_count\_part1\_a)* must be read first and then the lower 32-bits can be read (*cr\_osf\_ob\_agg\_data\_bytes\_0\_count\_part0\_a*).

### Post-data TLV History Buffers

The 256 x 92 FIFO used for Post-data TLVs can double as a history buffer because it is SW accessible. It is deep enough to store all of the Post-data TLVs for multiple simple commands and compound command data frames.

### Pipeline Status Support

To support the Pipeline Status CSR in the CCEIP and CDDIP support blocks, the OSF provides a strobe signal that pulses high when a CQE TLV exits the OSF.

### Miscellaneous Debug Support

For debug purposes, the outbound AXI bus of the OSF can be backpressured by setting *cr\_osf\_debug\_ctl\_config.force\_ob\_bp* to a 1.

## Statistics

Table 28 shows the statistics events the OSF provides to the Selectable Event Counters in the SA.

|  |  |  |
| --- | --- | --- |
| **Statistic Event** | **Statistic**  **Event**  **Number** | **Description** |
| OSF\_DATA\_INPUT\_STALL\_TOTAL | 512 | Number of cycles the OSF is stalling the inbound data path interface |
| OSF\_CG\_INPUT\_STALL\_TOTAL | 513 | Number of cycles the OSF is stalling the inbound completion interface from the CG |
| OSF\_OUTPUT\_BACKPRESSURE\_TOTAL | 514 | Number of cycles the OSF engine outbound interface is being stalled by the system |
| OSF\_OUTPUT\_STALL\_TOTAL | 515 | Number of cycles the OSF is stalling the engine outbound interface |

Table : OSF Statistics Events

## Interrupts

Table 30 shows the interrupts provided by the OSF.

|  |  |
| --- | --- |
| **Interrupt** | **Description** |
| TLV Parser Error | TLVP parser in the OSF is reporting an error. See 11 for details. |
| Uncorrectable ECC Error | Uncorrectable ECC error detected in the OSF DATA or PDT FIFO RAMs |

Table : ISF Interrupts

## Self-Test

The OSF provides the self-test control used to isolate the CCEIP/CDDIP from the rest of the system during Crypto self-test via cr\_osf\_debug\_ctl\_config. eng\_self\_test\_en. During self-test, the CCEIP/CDDIP is isolated by not accepting inbound traffic from the system and not sending outbound traffic to the system. To achieve this on the inbound side, the TVALID to the ISF is de-asserted, and TREADY back to the system is asserted. As such, the engine will not accept any inbound traffic, but will not back-pressure the system. To achieve this on the outbound side, the TVALID to the system from the OSF is de-asserted, and TREADY to the OSF is asserted. As such, the system sees no outbound traffic and cannot back-pressure the OSF.

## Outbound Halting

The OSF receives a halt signal from the CCEIP/CDDIP support blocks (9) whenever an interrupt occurs and is enabled. The OSF immediately halts the outbound traffic on the AXI4S interface when this occurs, and the entire engine must be reset to recover. Since almost all of the interrupts are considered fatal errors, this is a protection mechanism to prevent corrupted traffic from exiting the engine.

## Interfaces

The OSF interfaces are shown in Table 30. With the exception of the Stats/Error Events and Pipeline Status interfaces, which are a collection of single-bit signals, a definition of each interface type is available in Interface Descriptions.

|  |  |  |
| --- | --- | --- |
| Interface  Type | Data  Width | Notes |
| AXI4S Slave | 64 | Inbound Data (Shared Interface. See Note 1) |
| AXI4S Master | 64 | Outbound Data |
| RBUS In | 32 | CSR Access |
| RBUS Out | 32 | CSR Access |
| Statistic Events | 4 | To SA |
| Interrupts | 2 | To CCEIP/CDDIP support block |
| Pipeline Status | 1 | To CCEIP/CDDIP support block |
| Halt | 1 | From CCEIP/CDDIP support block |

Table : OSF Interfaces

**Notes**

1. In both the CCEIP and CDDIP, the inbound slave interface is also shared with another slave interface. As such, the OSF must know the state of the other interface’s TREADY signal to know when it is back pressuring. The OSF has logic to de-assert the TVALID signal from the master whenever the other interface’s TREADY signal is de-asserted.

## Engine Errors

The OSF does not generate any engine errors.

# Statistics Accumulator (SA)

## Features

* 64, 50-bit Selectable Event Counters that monitor events from CCEIP/ CDDIP engines
  + Programmable assignment of events to counters
  + SW controlled “Snapshot” and “Clear Live” support
  + All counters SW accessible (read and clear)

## Operational Description

The SA contains 64, 50-bit Selectable Event Counters that monitor events from the various modules in the CCEIP and CDDIP. Since the CCEIP and CDDIP have a different number of events, there are separate SA modules for each. These events are single-bit signals that are pulsed high for one clock period and cause the counter it is associated with to increment by 1. Each of the 64 counters is assigned an event to monitor via a 10-bit, per-counter, configuration register. Per-module lists of the events that can be monitored are found in Table 31, while all of the events in the CCEIP and CDDIP are found in Table 32 and Table 33. These counters are rollover counters that at 50-bits will only rollover every 22 years. They are not “clear on read” counters.

The events sent to the SA are enabled on a per-command basis in the ISF via the scheme described in 3.3.3. It is the responsibility of each module that generates events to enable/disable them per the state of CMD.trace.

### Controls

**Snapshot View**

SW, via *cr\_cceip\_64\_sa\_global\_ctrl.sa\_snap* and *cr\_cddip\_sa\_global\_ctrl.sa\_snap*, can command that the current “live” values of all of the counters be transferred to a set of snapshot counters in a single clock cycle to provide a coherent and static view of all of the values. These are accessible through the *cr\_cceip\_64\_sa\_snapshot* and *cr\_cddip\_sa\_snapshot* registers. The “live” values are accessible the *cr\_cceip\_64\_sa\_count* and *cr\_cddip\_sa\_count* registers

**Clear Live**

SW, via *cr\_cceip\_64\_sa\_global\_ctrl.sa\_clear\_live* and *cr\_cddip\_sa\_global\_ctrl.sa\_clear\_live* can reset all of the live counters to 0 in one clock cycle.

### Countable Events

The event to monitor for each counter is selected via the event number in the associated *cr\_cceip\_64\_sa\_ctrl* and *cr\_cddip\_sa\_ctrl* register arrays (e.g. CCEIP Event Counter 0 is associated with *cr\_cceip\_64\_sa\_ctrl[0]*). The event numbers are shown in Table 32 and Table 33.

|  |  |
| --- | --- |
| Stat Event Number Range | Stat Event Vector |
| 63 - 0 | crypto\_ckmic\_chk\_stat\_events |
| 127 - 64 | {crcc0\_stat\_events events , crcc1\_stat\_events, crcgc0\_stat\_events cg\_stat\_events} |
| 191 - 128 | spare |
| 255 - 192 | spare |
| 319 - 256 | xp10\_decomp\_hufd\_stat\_events[127:64] |
| 383 - 320 | xp10\_decomp\_hufd\_stat\_events[63:0] |
| 447 - 384 | xp10\_decomp\_lz77d\_stat\_events |
| 511 - 448 | crypto\_decrypt\_stat\_events |
| 575 - 512 | osf\_stat\_events |
| 639 - 576 | crypto\_encrypt\_stat\_events |
| 703 - 640 | huf\_comp\_stat\_events |
| 767 - 704 | lz77\_comp\_stat\_events |
| 831 - 768 | prefix\_stat\_events [127:64] |
| 895 - 832 | prefix\_stat\_events [63:0] |
| 959 - 896 | isf\_stat\_events |
| 1023 - 960 | huf\_comp\_xp10\_decomp\_lz77d\_stat\_events |

Table : Per-Module Statistic Events

The following table lists the stat events by event number:

Note: any Stat Event Number not listed is unused.

|  |  |  |
| --- | --- | --- |
| Stat Event Number | Stat Event Name | Stat Event Description |
| 0 | CKMIC\_IV\_MISMATCH\_FRAME | This should only Trigger in Decrypt Block of CCEIP Engine. This happens when random IV generated by Encrypt Block does not match the IV in Footer.It is mainly due to bit errors. |
| 1 | CKMIC\_ENGINE\_ID\_MISMATCH\_FRAME | Engine ID in CMD TLV does not match Engine ID in Key TLV. |
| 2 | CKMIC\_SEQ\_ID\_MISMATCH\_FRAME | Sequence ID in CMD TLV does not match Engine ID in Key TLV. |
| 3 | CKMIC\_HMAC\_SHA256\_TAG\_FAIL\_FRAME | HMAC-SHA256 Tag was generated and it failed against Footer Tag upon comparison. |
| 4 | CKMIC\_SHA256\_TAG\_FAIL\_FRAME | SHA256 Tag was generated and it failed against Footer Tag upon comparison. |
| 5 | CKMIC\_GMAC\_TAG\_FAIL\_FRAME | GMAC Tag was generated and it failed against Footer Tag upon comparison. |
| 6 | CKMIC\_GCM\_TAG\_FAIL\_FRAME | GCM Tag was generated and it failed against Footer Tag upon comparison. |
| 7 | CKMIC\_AUTH\_NOP\_FRAME | GCM Tag was generated and it failed against Footer Tag upon comparison. |
| 8 | CKMIC\_AUTH\_HMAC\_SHA256\_FRAME | HMAC-SHA256 Authentication Operation was requested by a frame. |
| 9 | CKMIC\_AUTH\_SHA256\_FRAME | SHA256 Authentication Operation was requested by a frame. |
| 10 | CKMIC\_AUTH\_AES\_GMAC\_FRAME | GMAC Operation was requested by a frame. |
| 11 | CKMIC\_CIPH\_NOP\_FRAME | NOP Operation was requested by a frame. |
| 12 | CKMIC\_CIPH\_AES\_XEX\_FRAME | AES-256-XEX Operation was requested by a frame. |
| 13 | CKMIC\_CIPH\_AES\_XTS\_FRAME | AES-256-XTS Operation was requested by a frame. |
| 14 | CKMIC\_CIPH\_AES\_GCM\_FRAME | AES-256-GCM Operation was requested by a frame. |
| 64 | CRCG0\_RAW\_CHSUM\_GOOD\_TOTAL | Total number of raw data chksum good |
| 65 | CRCG0\_RAW\_CHSUM\_ERROR\_TOTAL | Total number of raw data chksum errors |
| 66 | CRCG0\_CRC64E\_CHSUM\_GOOD\_TOTAL | Total number of crc64e data chksum good |
| 67 | CRCG0\_CRC64E\_CHSUM\_ERROR\_TOTAL | Total number of crc64e data chksum errors |
| 68 | CRCG0\_ENC\_CHSUM\_GOOD\_TOTAL | Total number of crc encap data chksum good; |
| 69 | CRCG0\_ENC\_CHSUM\_ERROR\_TOTAL | Total number of crc encap data chksum errors |
| 70 | CRCG0\_NVME\_CHSUM\_GOOD\_TOTAL | Total number of nvme data chksum good; |
| 71 | CRCG0\_NVME\_CHSUM\_ERROR\_TOTAL | Total number of nvme data chksum errors |
| 72 | CRCGC0\_RAW\_CHSUM\_GOOD\_TOTAL | Total number of raw data chksum good |
| 73 | CRCGC0\_RAW\_CHSUM\_ERROR\_TOTAL | Total number of raw data chksum errors |
| 74 | CRCGC0\_CRC64E\_CHSUM\_GOOD\_TOTAL | Total number of crc64e data chksum good |
| 75 | CRCGC0\_CRC64E\_CHSUM\_ERROR\_TOTAL | Total number of crc64e data chksum errors |
| 76 | CRCGC0\_ENC\_CHSUM\_GOOD\_TOTAL | Total number of crc encap data chksum good; |
| 77 | CRCGC0\_ENC\_CHSUM\_ERROR\_TOTAL | Total number of crc encap data chksum errors |
| 78 | CRCGC0\_NVME\_CHSUM\_GOOD\_TOTAL | Total number of nvme data chksum good; |
| 79 | CRCGC0\_NVME\_CHSUM\_ERROR\_TOTAL | Total number of nvme data chksum errors |
| 80 | CRCC1\_RAW\_CHSUM\_GOOD\_TOTAL | Total number of raw data chksum good |
| 81 | CRCC1\_RAW\_CHSUM\_ERROR\_TOTAL | Total number of raw data chksum errors |
| 82 | CRCC1\_CRC64E\_CHSUM\_GOOD\_TOTAL | Total number of crc64e data chksum good |
| 83 | CRCC1\_CRC64E\_CHSUM\_ERROR\_TOTAL | Total number of crc64e data chksum errors |
| 84 | CRCC1\_ENC\_CHSUM\_GOOD\_TOTAL | Total number of crc encap data chksum good; |
| 85 | CRCC1\_ENC\_CHSUM\_ERROR\_TOTAL | Total number of crc encap data chksum errors |
| 86 | CRCC1\_NVME\_CHSUM\_GOOD\_TOTAL | Total number of nvme data chksum good; |
| 87 | CRCC1\_NVME\_CHSUM\_ERROR\_TOTAL | Total number of nvme data chksum errors |
| 88 | CRCC0\_RAW\_CHSUM\_GOOD\_TOTAL | Total number of raw data chksum good |
| 89 | CRCC0\_RAW\_CHSUM\_ERROR\_TOTAL | Total number of raw data chksum errors |
| 90 | CRCC0\_CRC64E\_CHSUM\_GOOD\_TOTAL | Total number of crc64e data chksum good |
| 91 | CRCC0\_CRC64E\_CHSUM\_ERROR\_TOTAL | Total number of crc64e data chksum errors |
| 92 | CRCC0\_ENC\_CHSUM\_GOOD\_TOTAL | Total number of crc encap data chksum good; |
| 93 | CRCC0\_ENC\_CHSUM\_ERROR\_TOTAL | Total number of crc encap data chksum errors |
| 94 | CRCC0\_NVME\_CHSUM\_GOOD\_TOTAL | Total number of nvme data chksum good; |
| 95 | CRCC0\_NVME\_CHSUM\_ERROR\_TOTAL | Total number of nvme data chksum errors |
| 96 | CG\_ENGINE\_ERROR\_COMMAND | Number of commands with engine related errors that the engine outputs |
| 97 | CG\_SELECT\_ENGINE\_ERROR\_COMMAND | Number of commands with programmable engine related error codes that the engine outputs. The error code and associated mask is configured in the CR\_CG\_DEBUG\_CTL\_CONFIG CSR. |
| 98 | CG\_SYSTEM\_ERROR\_COMMAND | Number of commands with system related errors that the engine outputs |
| 99 | CG\_CQE\_OUTPUT\_COMMAND | Number of CQE TLVs that the engine outputs |
| 320 | HUFD\_FE\_XP9\_FRM\_TOTAL | Number of XP9 frames received |
| 321 | HUFD\_FE\_XP9\_BLK\_TOTAL | Number of XP9 Blocks received |
| 322 | HUFD\_FE\_XP9\_RAW\_FRM\_TOTAL | Number of XP9 Raw frames received |
| 323 | HUFD\_FE\_XP10\_FRM\_TOTAL | Number of XP10 Frames received |
| 324 | HUFD\_FE\_XP10\_FRM\_PFX\_TOTAL | Number of Prefix TLVs received |
| 325 | HUFD\_FE\_XP10\_FRM\_PDH\_TOTAL | Number of Predefined Huffman TLVs received |
| 326 | HUFD\_FE\_XP10\_BLK\_TOTAL | Number of XP10 Blocks received |
| 327 | HUFD\_FE\_XP10\_RAW\_BLK\_TOTAL | Number of XP10 Raw Blocks received |
| 328 | HUFD\_FE\_GZIP\_FRM\_TOTAL | Number of GZIP Frames received |
| 329 | HUFD\_FE\_GZIP\_BLK\_TOTAL | Number of GZIP Blocks received |
| 330 | HUFD\_FE\_GZIP\_RAW\_BLK\_TOTAL | Number of GZIP Raw Blocks received |
| 331 | HUFD\_FE\_ZLIB\_FRM\_TOTAL | Number of ZLIB Frames received |
| 332 | HUFD\_FE\_ZLIB\_BLK\_TOTAL | Number of ZLIB Blocks received |
| 333 | HUFD\_FE\_ZLIB\_RAW\_BLK\_TOTAL | Number of ZLIB Raw Blocks received |
| 334 | HUFD\_FE\_CHU4K\_TOTAL | Number of CHU4K frames received |
| 335 | HUFD\_FE\_CHU8K\_TOTAL | Number of CHU8K frames received |
| 336 | HUFD\_FE\_CHU4K\_RAW\_TOTAL | Number of CHU4K Raw frames received |
| 337 | HUFD\_FE\_CHU8K\_RAW\_TOTAL | Number of CHU8K Raw frames received |
| 338 | HUFD\_FE\_PFX\_CRC\_ERR\_TOTAL | Number of Prefix TLVs received with CRC error. |
| 339 | HUFD\_FE\_PHD\_CRC\_ERR\_TOTAL | Number of Pre-determined Huffman TLVs received with CRC error. |
| 340 | HUFD\_FE\_XP9\_CRC\_ERR\_TOTAL | Number of XP9 frames received with header CRC error. |
| 341 | HUFD\_HTF\_XP9\_SIMPLE\_SHORT\_BLK\_TOTAL | Number of XP9 blocks with simple encoding of the short symbols |
| 342 | HUFD\_HTF\_XP9\_RETRO\_SHORT\_BLK\_TOTAL | Number of XP9 blocks with retro encoding of the short symbols |
| 343 | HUFD\_HTF\_XP9\_SIMPLE\_LONG\_BLK\_TOTAL | Number of XP9 blocks with simple encoding of the long symbols |
| 344 | HUFD\_HTF\_XP9\_RETRO\_LONG\_BLK\_TOTAL | Number of XP9 blocks with retro encoding of the long symbols |
| 345 | HUFD\_HTF\_XP10\_SIMPLE\_SHORT\_BLK\_TOTAL | Number of XP10 blocks with simple encoding of the short symbols |
| 346 | HUFD\_HTF\_XP10\_RETRO\_SHORT\_BLK\_TOTAL | Number of XP10 blocks with retro encoding of the short symbols |
| 347 | HUFD\_HTF\_XP10\_PREDEF\_SHORT\_BLK\_TOTAL | Number of XP10 blocks with predef encoding of the short symbols |
| 348 | HUFD\_HTF\_XP10\_SIMPLE\_LONG\_BLK\_TOTAL | Number of XP10 blocks with simple encoding of the long symbols |
| 349 | HUFD\_HTF\_XP10\_RETRO\_LONG\_BLK\_TOTAL | Number of XP10 blocks with retro encoding of the long symbols |
| 350 | HUFD\_HTF\_XP10\_PREDEF\_LONG\_BLK\_TOTAL | Number of XP10 blocks with predef encoding of the long symbols |
| 351 | HUFD\_HTF\_CHU4K\_SIMPLE\_SHORT\_BLK\_TOTAL | Number of CHU4K blocks with simple encoding of the short symbols |
| 352 | HUFD\_HTF\_CHU4K\_RETRO\_SHORT\_BLK\_TOTAL | Number of CHU4K blocks with retro encoding of the short symbols |
| 353 | HUFD\_HTF\_CHU4K\_PREDEF\_SHORT\_BLK\_TOTAL | Number of CHU4K blocks with predef encoding of the short symbols |
| 354 | HUFD\_HTF\_CHU4K\_SIMPLE\_LONG\_BLK\_TOTAL | Number of CHU4K blocks with simple encoding of the long symbols |
| 355 | HUFD\_HTF\_CHU4K\_RETRO\_LONG\_BLK\_TOTAL | Number of CHU4K blocks with retro encoding of the long symbols |
| 356 | HUFD\_HTF\_CHU4K\_PREDEF\_LONG\_BLK\_TOTAL | Number of CHU4K blocks with predef encoding of the long symbols |
| 357 | HUFD\_HTF\_CHU8K\_SIMPLE\_SHORT\_BLK\_TOTAL | Number of CHU8K blocks with simple encoding of the short symbols |
| 358 | HUFD\_HTF\_CHU8K\_RETRO\_SHORT\_BLK\_TOTAL | Number of CHU8K blocks with retro encoding of the short symbols |
| 359 | HUFD\_HTF\_CHU8K\_PREDEF\_SHORT\_BLK\_TOTAL | Number of CHU8K blocks with predef encoding of the short symbols |
| 360 | HUFD\_HTF\_CHU8K\_SIMPLE\_LONG\_BLK\_TOTAL | Number of CHU8K blocks with simple encoding of the long symbols |
| 361 | HUFD\_HTF\_CHU8K\_RETRO\_LONG\_BLK\_TOTAL | Number of CHU8K blocks with retro encoding of the long symbols |
| 362 | HUFD\_HTF\_CHU8K\_PREDEF\_LONG\_BLK\_TOTAL | Number of CHU8K blocks with predef encoding of the long symbols |
| 363 | HUFD\_HTF\_DEFLATE\_DYNAMIC\_BLK\_TOTAL | Number of DEFLATE blocks with dynamic encoding |
| 364 | HUFD\_HTF\_DEFLATE\_FIXED\_BLK\_TOTAL | Number of DEFLATE blocks with fixed encoding |
| 365 | HUFD\_MTF\_0\_TOTAL | Number of MTF[0] occurences |
| 366 | HUFD\_MTF\_1\_TOTAL | Number of MTF[1] occurences |
| 367 | HUFD\_MTF\_2\_TOTAL | Number of MTF[2] occurences |
| 368 | HUFD\_MTF\_3\_TOTAL | Number of MTF[3] occurences |
| 369 | HUFD\_FE\_FHP\_STALL\_TOTAL | Number of stall cycles at the FHP |
| 370 | HUFD\_FE\_LFA\_STALL\_TOTAL | Number of stall cycles at the LFA |
| 371 | HUFD\_HTF\_PREDEF\_STALL\_TOTAL | Number of stall cycles at the HTF predef interface |
| 372 | HUFD\_HTF\_HDR\_DATA\_STALL\_TOTAL | Number of stall cycles at the HTF header data interface |
| 373 | HUFD\_HTF\_HDR\_INFO\_STALL\_TOTAL | Number of stall cycles at the HTF header info interface |
| 374 | HUFD\_SDD\_INPUT\_STALL\_TOTAL | Number of stall cycles at the SDD input interface |
| 375 | HUFD\_SDD\_BUF\_FULL\_STALL\_TOTAL | Number of stall cycles in the SDD due to circular bit buffer full |
| 384 | LZ77D\_PTR\_LEN\_256\_TOTAL | Number of 256 byte length pointers received |
| 385 | LZ77D\_PTR\_LEN\_128\_TOTAL | Number of 128 byte length pointers received |
| 386 | LZ77D\_PTR\_LEN\_64\_TOTAL | Number of 64 byte length pointers received |
| 387 | LZ77D\_PTR\_LEN\_32\_TOTAL | Number of 32 byte length pointers received |
| 388 | LZ77D\_PTR\_LEN\_11\_TOTAL | Number of 11 byte length pointers received |
| 389 | LZ77D\_PTR\_LEN\_10\_TOTAL | Number of 10 byte length pointers received |
| 390 | LZ77D\_PTR\_LEN\_9\_TOTAL | Number of 9 byte length pointers received |
| 391 | LZ77D\_PTR\_LEN\_8\_TOTAL | Number of 8 byte length pointers received |
| 392 | LZ77D\_PTR\_LEN\_7\_TOTAL | Number of 7 byte length pointers received |
| 393 | LZ77D\_PTR\_LEN\_6\_TOTAL | Number of 6 byte length pointers received |
| 394 | LZ77D\_PTR\_LEN\_5\_TOTAL | Number of 5 byte length pointers received |
| 395 | LZ77D\_PTR\_LEN\_4\_TOTAL | Number of 4 byte length pointers received |
| 396 | LZ77D\_PTR\_LEN\_3\_TOTAL | Number of 3 byte length pointers received |
| 397 | LZ77D\_LANE\_1\_LITERALS\_TOTAL | Total count of single literal received per clock |
| 398 | LZ77D\_LANE\_2\_LITERALS\_TOTAL | Total count of 2 literals received per clock |
| 399 | LZ77D\_LANE\_3\_LITERALS\_TOTAL | Total count of 3 literals received per clock |
| 400 | LZ77D\_LANE\_4\_LITERALS\_TOTAL | Total count of 4 literals received per clock |
| 401 | LZ77D\_PTRS\_TOTAL | Number of Pointers received |
| 402 | LZ77D\_FRM\_IN\_TOTAL | Number of Frames received |
| 403 | LZ77D\_FRM\_OUT\_TOTAL | Number of Frames sent |
| 404 | LZ77D\_STALL\_TOTAL | Number of Stall cycles at the LZ77 Decompressor |
| 448 | DECRYPT\_IV\_MISMATCH\_FRAME | This should only Trigger in Decrypt Block of CCEIP Engine. This happens when random IV generated by Encrypt Block does not match the IV in Footer. It is mainly due to bit errors. |
| 449 | DECRYPT\_ENGINE\_ID\_MISMATCH\_FRAME | Engine ID in CMD TLV does not match Engine ID in Key TLV. |
| 450 | DECRYPT\_SEQ\_ID\_MISMATCH\_FRAME | Sequence ID in CMD TLV does not match Engine ID in Key TLV. |
| 451 | DECRYPT\_HMAC\_SHA256\_TAG\_FAIL\_FRAME | HMAC-SHA256 Tag was generated and it failed against Footer Tag upon comparison. |
| 452 | DECRYPT\_SHA256\_TAG\_FAIL\_FRAME | SHA256 Tag was generated and it failed against Footer Tag upon comparison. |
| 453 | DECRYPT\_GMAC\_TAG\_FAIL\_FRAME | GMAC Tag was generated and it failed against Footer Tag upon comparison. |
| 454 | DECRYPT\_GCM\_TAG\_FAIL\_FRAME | GCM Tag was generated and it failed against Footer Tag upon comparison. |
| 455 | DECRYPT\_AUTH\_NOP\_FRAME | GCM Tag was generated and it failed against Footer Tag upon comparison. |
| 456 | DECRYPT\_AUTH\_HMAC\_SHA256\_FRAME | HMAC-SHA256 Authentication Operation was requested by a frame. |
| 457 | DECRYPT\_AUTH\_SHA256\_FRAME | SHA256 Authentication Operation was requested by a frame. |
| 458 | DECRYPT\_AUTH\_AES\_GMAC\_FRAME | GMAC Operation was requested by a frame. |
| 459 | DECRYPT\_CIPH\_NOP\_FRAME | NOP Operation was requested by a frame. |
| 460 | DECRYPT\_CIPH\_AES\_XEX\_FRAME | AES-256-XEX Operation was requested by a frame. |
| 461 | DECRYPT\_CIPH\_AES\_XTS\_FRAME | AES-256-XTS Operation was requested by a frame. |
| 462 | DECRYPT\_CIPH\_AES\_GCM\_FRAME | AES-256-GCM Operation was requested by a frame. |
| 512 | OSF\_DATA\_INPUT\_STALL\_TOTAL | Number of cycles the OSF is stalling the inbound data path interface |
| 513 | OSF\_CG\_INPUT\_STALL\_TOTAL | Number of cycles the OSF is stalling the inbound completion interface from the CG |
| 514 | OSF\_OUTPUT\_BACKPRESSURE\_TOTAL | Number of cycles the OSF engine outbound interface is being stalled by the system |
| 515 | OSF\_OUTPUT\_STALL\_TOTAL | Number of cycles the OSF is stalling the engine outbound interface |
| 576 | ENCRYPT\_IV\_MISMATCH\_FRAME | This should only Trigger in Decrypt Block of CCEIP Engine. This happens when random IV generated by Encrypt Block does not match the IV in Footer. It is mainly due to bit errors. |
| 577 | ENCRYPT\_ENGINE\_ID\_MISMATCH\_FRAME | Engine ID in CMD TLV does not match Engine ID in Key TLV. |
| 578 | ENCRYPT\_SEQ\_ID\_MISMATCH\_FRAME | Sequence ID in CMD TLV does not match Engine ID in Key TLV. |
| 579 | ENCRYPT\_HMAC\_SHA256\_TAG\_FAIL\_FRAME | HMAC-SHA256 Tag was generated and it failed against Footer Tag upon comparison. |
| 580 | ENCRYPT\_SHA256\_TAG\_FAIL\_FRAME | SHA256 Tag was generated and it failed against Footer Tag upon comparison. |
| 581 | ENCRYPT\_GMAC\_TAG\_FAIL\_FRAME | GMAC Tag was generated and it failed against Footer Tag upon comparison. |
| 582 | ENCRYPT\_GCM\_TAG\_FAIL\_FRAME | GCM Tag was generated and it failed against Footer Tag upon comparison. |
| 583 | ENCRYPT\_AUTH\_NOP\_FRAME | GCM Tag was generated and it failed against Footer Tag upon comparison. |
| 584 | ENCRYPT\_AUTH\_HMAC\_SHA256\_FRAME | HMAC-SHA256 Authentication Operation was requested by a frame. |
| 585 | ENCRYPT\_AUTH\_SHA256\_FRAME | SHA256 Authentication Operation was requested by a frame. |
| 586 | ENCRYPT\_AUTH\_AES\_GMAC\_FRAME | GMAC Operation was requested by a frame. |
| 587 | ENCRYPT\_CIPH\_NOP\_FRAME | NOP Operation was requested by a frame. |
| 588 | ENCRYPT\_CIPH\_AES\_XEX\_FRAME | AES-256-XEX Operation was requested by a frame. |
| 589 | ENCRYPT\_CIPH\_AES\_XTS\_FRAME | AES-256-XTS Operation was requested by a frame. |
| 590 | ENCRYPT\_CIPH\_AES\_GCM\_FRAME | AES-256-GCM Operation was requested by a frame. |
| 640 | SHORT\_MAP\_ERR\_TOTAL | Number of short symbol mapping errors detected when the corresponding value is out of range, depending on the protocol. Note, this error is also inserted into the FTR TLV on a per frame basis. |
| 641 | LONG\_MAP\_ERR\_TOTAL | Number of long symbol mapping errors detected when the corresponding value is out of range, depending on the protocol. Note, this error is also inserted into the FTR TLV on a per frame basis. |
| 642 | XP9\_BLK\_COMP\_TOTAL | Total number of XP9 coding blocks detected when frames are selected for compression. |
| 643 | XP9\_FRM\_RAW\_TOTAL | Total number of XP9 frames sent raw (uncompressed).Note, if a frame is sent raw, the frame will not be broken up into coding blocks. |
| 644 | XP9\_FRM\_TOTAL | Total number of XP9 frames sent, compressed and raw. |
| 645 | XP9\_BLK\_SHORT\_SIM\_TOTAL | Total number of XP9 short coding blocks using simple encoding. |
| 646 | XP9\_BLK\_LONG\_SIM\_TOTAL | Total number of XP9 long coding blocks using simple encoding. |
| 647 | XP9\_BLK\_SHORT\_RET\_TOTAL | Total number of XP9 short coding blocks using retrospective encoding. |
| 648 | XP9\_BLK\_LONG\_RET\_TOTAL | Total number of XP9 long coding blocks using retrospective encoding. |
| 649 | XP10\_BLK\_COMP\_TOTAL | Total number of XP10 coding blocks when the coding block is sent compressed. |
| 650 | XP10\_BLK\_RAW\_TOTAL | Total number of XP10 coding blocks when the coding block is sent raw. |
| 651 | XP10\_BLK\_SHORT\_SIM\_TOTAL | Total number of XP10 short coding blocks using simple encoding. |
| 652 | XP10\_BLK\_LONG\_SIM\_TOTAL | Total number of XP10 long coding blocks using simple encoding. |
| 653 | XP10\_BLK\_SHORT\_RET\_TOTAL | Total number of XP10 short coding blocks using retrospective encoding. |
| 654 | XP10\_BLK\_LONG\_RET\_TOTAL | Total number of XP10 long coding blocks using retrospective encoding. |
| 655 | XP10\_BLK\_SHORT\_PRE\_TOTAL | Total number of XP10 short coding blocks using predetermined encoding. |
| 656 | XP10\_BLK\_LONG\_PRE\_TOTAL | Total number of XP10 long coding blocks using predetermined encoding. |
| 657 | XP10\_FRM\_TOTAL | Total number of XP10 frames sent, compressed and raw. |
| 658 | CHU8\_FRM\_RAW\_TOTAL | Total number of CHU8 frames sent raw. |
| 659 | CHU8\_FRM\_COMP\_TOTAL | Total number of CHU8 frames sent compressed. |
| 660 | CHU8\_FRM\_SHORT\_SIM\_TOTAL | Total number of CHU8 short frames using simple encoding. |
| 661 | CHU8\_FRM\_LONG\_SIM\_TOTAL | Total number of CHU8 long frames using simple encoding. |
| 662 | CHU8\_FRM\_SHORT\_RET\_TOTAL | Total number of CHU8 short frames using retrospective encoding. |
| 663 | CHU8\_FRM\_LONG\_RET\_TOTAL | Total number of CHU8 long frames using retrospective encoding. |
| 664 | CHU8\_FRM\_SHORT\_PRE\_TOTAL | Total number of CHU8 short frames using predetermined encoding. |
| 665 | CHU8\_FRM\_LONG\_PRE\_TOTAL | Total number of CHU8 long frames using predetermined encoding. |
| 666 | CHU8\_CMD\_TOTAL | Total number of CHU8 commands. |
| 667 | CHU4\_FRM\_RAW\_TOTAL | Total number of CHU4 frames sent raw. |
| 668 | CHU4\_FRM\_COMP\_TOTAL | Total number of CHU4 frames sent compressed. |
| 669 | CHU4\_FRM\_SHORT\_SIM\_TOTAL | Total number of CHU4 short frames using simple encoding. |
| 670 | CHU4\_FRM\_LONG\_SIM\_TOTAL | Total number of CHU4 long frames using simple encoding. |
| 671 | CHU4\_FRM\_SHORT\_RET\_TOTAL | Total number of CHU4 short frames using retrospective encoding. |
| 672 | CHU4\_FRM\_LONG\_RET\_TOTAL | Total number of CHU4 long frames using retrospective encoding. |
| 673 | CHU4\_FRM\_SHORT\_PRE\_TOTAL | Total number of CHU4 short frames using predetermined encoding. |
| 674 | CHU4\_FRM\_LONG\_PRE\_TOTAL | Total number of CHU4 long frames using predetermined encoding. |
| 675 | CHU4\_CMD\_TOTAL | Total number of CHU4 commands. |
| 676 | DF\_BLK\_COMP\_TOTAL | Total number of deflate coding blocks when the coding block is sent compressed. |
| 677 | DF\_BLK\_RAW\_TOTAL | Total number of deflate coding blocks when the coding block is sent raw. |
| 678 | DF\_BLK\_SHORT\_SIM\_TOTAL | Total number of deflate short coding blocks using simple encoding. |
| 679 | DF\_BLK\_LONG\_SIM\_TOTAL | Total number of deflate long coding blocks using simple encoding. |
| 680 | DF\_BLK\_SHORT\_RET\_TOTAL | Total number of deflate short coding blocks using retrospective encoding. |
| 681 | DF\_BLK\_LONG\_RET\_TOTAL | Total number of deflate long coding blocks using retrospective encoding. |
| 682 | DF\_FRM\_TOTAL | Total number of deflate frames sent, compressed and raw. |
| 683 | PASS\_THRU\_FRM\_TOTAL | Total number of pass through frames sent. |
| 684 | BYTE\_0\_TOTAL | Total number of payload data bytes in lane 0, compressed, raw and passthrough, sent by the Huffman Encoder. Note, software must add the counts for all 8 lanes to get total count. |
| 685 | BYTE\_1\_TOTAL | Total number of payload data bytes in lane 1, compressed, raw and passthrough, sent by the Huffman Encoder. Note, software must add the counts for all 8 lanes to get total count. |
| 686 | BYTE\_2\_TOTAL | Total number of payload data bytes in lane 2, compressed, raw and passthrough, sent by the Huffman Encoder. Note, software must add the counts for all 8 lanes to get total count. |
| 687 | BYTE\_3\_TOTAL | Total number of payload data bytes in lane 3, compressed, raw and passthrough, sent by the Huffman Encoder. Note, software must add the counts for all 8 lanes to get total count. |
| 688 | BYTE\_4\_TOTAL | Total number of payload data bytes in lane 4, compressed, raw and passthrough, sent by the Huffman Encoder. Note, software must add the counts for all 8 lanes to get total count. |
| 689 | BYTE\_5\_TOTAL | Total number of payload data bytes in lane 5, compressed, raw and passthrough, sent by the Huffman Encoder. Note, software must add the counts for all 8 lanes to get total count. |
| 690 | BYTE\_6\_TOTAL | Total number of payload data bytes in lane 6, compressed, raw and passthrough, sent by the Huffman Encoder. Note, software must add the counts for all 8 lanes to get total count. |
| 691 | BYTE\_7\_TOTAL | Total number of payload data bytes in lane 7, compressed, raw and passthrough, sent by the Huffman Encoder. Note, software must add the counts for all 8 lanes to get total count. |
| 692 | ENCRYPT\_STALL\_TOTAL | Total number of instances where backpressure from the downstream encyption block caused the Huffmane Encoder to stall. |
| 693 | LZ77\_STALL\_TOTAL | Total number of instances where an idle input from the upstream LZ77 block caused the Huffmane Encoder to stall. |
| 704 | LZ77C\_eof\_FRAME | End of compressed file. This event will occur once per compressed data TLV, and is used to count the number |
| 705 | LZ77C\_bypass\_FRAME | This event occurs for all frames types for which the TLV Parser is configured as MODIFY, but do not undergo lz77 compression. For the default configuration this includes the TLV types FTR and DATA\_UNK. It also includes the DATA TLV if the lz77 compression algorithm is configured as NONE. |
| 706 | LZ77C\_mtf\_3\_TOTAL | This event occurs every time the fourth entry on the mtf stack is selected as the offset used in an {offset,length} output tuple |
| 707 | LZ77C\_mtf\_2\_TOTAL | This event occurs every time the third entry on the mtf stack is selected as the offset used in an {offset,length} output tuple |
| 708 | LZ77C\_mtf\_1\_TOTAL | This event occurs every time the second entry on the mtf stack is selected as the offset used in an {offset,length} output tuple |
| 709 | LZ77C\_mtf\_0\_TOTAL | This event occurs every time the first entry on the mtf stack is selected as the offset used in an {offset,length} output tuple |
| 710 | LZ77C\_run\_256\_nup\_TOTAL | This event occurs every time the length in an {offset,length}output tuple is, length >= 256. |
| 711 | LZ77C\_run\_128\_255\_TOTAL | This event occurs every time the length in an {offset,length}output tuple is, 128 <= length <= 256 |
| 712 | LZ77C\_run\_64\_127\_TOTAL | This event occurs every time the length in an {offset,length}output tuple is, 64 <= length <= 127 |
| 713 | LZ77C\_run\_32\_63\_TOTAL | This event occurs every time the length in an {offset,length}output tuple is, 32 <= length <= 63 |
| 714 | LZ77C\_run\_11\_31\_TOTAL | This event occurs every time the length in an {offset,length}output tuple is, 11 <= length <= 31 |
| 715 | LZ77C\_run\_10\_TOTAL | This event occurs every time the length in an {offset,length}output tuple is equal to 10. |
| 716 | LZ77C\_run\_9\_TOTAL | This event occurs every time the length in an {offset,length}output tuple is equal to 9. |
| 717 | LZ77C\_run\_8\_TOTAL | This event occurs every time the length in an {offset,length}output tuple is equal to 8. |
| 718 | LZ77C\_run\_7\_TOTAL | This event occurs every time the length in an {offset,length}output tuple is equal to 7. |
| 719 | LZ77C\_run\_6\_TOTAL | This event occurs every time the length in an {offset,length}output tuple is equal to 6. |
| 720 | LZ77C\_run\_5\_TOTAL | This event occurs every time the length in an {offset,length}output tuple is equal to 5. |
| 721 | LZ77C\_run\_4\_TOTAL | This event occurs every time the length in an {offset,length}output tuple is equal to 4. |
| 722 | LZ77C\_run\_3\_TOTAL | Note, length 3 run have been deprecated and this event will never occur. |
| 723 | LZ77C\_mtf\_TOTAL | This event occurs every time any entry on the mtf stack is selected as the offset used in an {offset,length} output tuple. |
| 724 | LZ77C\_ptr\_TOTAL | This event occurs every time the offset used in an {offset,length}output tuple does not come from the mtf stack. That is, the value carried in the offset member is a true offset, rather than an index into the mtf stack. |
| 725 | LZ77C\_four\_lit\_TOTAL | This event occurs when 4 literals are output in a single cycle. |
| 726 | LZ77C\_three\_lit\_TOTAL | This event occurs when only 3 literals are output in a single cycle. |
| 727 | LZ77C\_two\_lit\_TOTAL | This event occurs when only 2 literals are output in a single cycle. |
| 728 | LZ77C\_one\_lit\_TOTAL | This event occurs when only 1 literal is output in a single cycle. |
| 729 | LZ77C\_throttled\_FRAME | Number of frames for which the data TLV and prefix size in bytes divided by 512 is greater than the configured active\_lpo\_thresh. |
| 832 | PREFIX\_NUM\_0\_TOTAL | Total number of times the prefix written was = 0 |
| 833 | PREFIX\_NUM\_1\_TOTAL | Total number of times the prefix written was = 1 |
| 834 | PREFIX\_NUM\_2\_TOTAL | Total number of times the prefix written was = 2 |
| 835 | PREFIX\_NUM\_3\_TOTAL | Total number of times the prefix written was = 3 |
| 836 | PREFIX\_NUM\_4\_TOTAL | Total number of times the prefix written was = 4 |
| 837 | PREFIX\_NUM\_5\_TOTAL | Total number of times the prefix written was = 5 |
| 838 | PREFIX\_NUM\_6\_TOTAL | Total number of times the prefix written was = 6 |
| 839 | PREFIX\_NUM\_7\_TOTAL | Total number of times the prefix written was = 7 |
| 840 | PREFIX\_NUM\_8\_TOTAL | Total number of times the prefix written was = 8 |
| 841 | PREFIX\_NUM\_9\_TOTAL | Total number of times the prefix written was = 9 |
| 842 | PREFIX\_NUM\_10\_TOTAL | Total number of times the prefix written was = 10 |
| 843 | PREFIX\_NUM\_11\_TOTAL | Total number of times the prefix written was = 11 |
| 844 | PREFIX\_NUM\_12\_TOTAL | Total number of times the prefix written was = 12 |
| 845 | PREFIX\_NUM\_13\_TOTAL | Total number of times the prefix written was = 13 |
| 846 | PREFIX\_NUM\_14\_TOTAL | Total number of times the prefix written was = 14 |
| 847 | PREFIX\_NUM\_15\_TOTAL | Total number of times the prefix written was = 15 |
| 848 | PREFIX\_NUM\_16\_TOTAL | Total number of times the prefix written was = 16 |
| 849 | PREFIX\_NUM\_17\_TOTAL | Total number of times the prefix written was = 17 |
| 850 | PREFIX\_NUM\_18\_TOTAL | Total number of times the prefix written was = 18 |
| 851 | PREFIX\_NUM\_19\_TOTAL | Total number of times the prefix written was = 19 |
| 852 | PREFIX\_NUM\_20\_TOTAL | Total number of times the prefix written was = 20 |
| 853 | PREFIX\_NUM\_21\_TOTAL | Total number of times the prefix written was = 21 |
| 854 | PREFIX\_NUM\_22\_TOTAL | Total number of times the prefix written was = 22 |
| 855 | PREFIX\_NUM\_23\_TOTAL | Total number of times the prefix written was = 23 |
| 856 | PREFIX\_NUM\_24\_TOTAL | Total number of times the prefix written was = 24 |
| 857 | PREFIX\_NUM\_25\_TOTAL | Total number of times the prefix written was = 25 |
| 858 | PREFIX\_NUM\_26\_TOTAL | Total number of times the prefix written was = 26 |
| 859 | PREFIX\_NUM\_27\_TOTAL | Total number of times the prefix written was = 27 |
| 860 | PREFIX\_NUM\_28\_TOTAL | Total number of times the prefix written was = 28 |
| 861 | PREFIX\_NUM\_29\_TOTAL | Total number of times the prefix written was = 29 |
| 862 | PREFIX\_NUM\_30\_TOTAL | Total number of times the prefix written was = 30 |
| 863 | PREFIX\_NUM\_31\_TOTAL | Total number of times the prefix written was = 31 |
| 864 | PREFIX\_NUM\_32\_TOTAL | Total number of times the prefix written was = 32 |
| 865 | PREFIX\_NUM\_33\_TOTAL | Total number of times the prefix written was = 33 |
| 866 | PREFIX\_NUM\_34\_TOTAL | Total number of times the prefix written was = 34 |
| 867 | PREFIX\_NUM\_35\_TOTAL | Total number of times the prefix written was = 35 |
| 868 | PREFIX\_NUM\_36\_TOTAL | Total number of times the prefix written was = 36 |
| 869 | PREFIX\_NUM\_37\_TOTAL | Total number of times the prefix written was = 37 |
| 870 | PREFIX\_NUM\_38\_TOTAL | Total number of times the prefix written was = 38 |
| 871 | PREFIX\_NUM\_39\_TOTAL | Total number of times the prefix written was = 39 |
| 872 | PREFIX\_NUM\_40\_TOTAL | Total number of times the prefix written was = 40 |
| 873 | PREFIX\_NUM\_41\_TOTAL | Total number of times the prefix written was = 41 |
| 874 | PREFIX\_NUM\_42\_TOTAL | Total number of times the prefix written was = 42 |
| 875 | PREFIX\_NUM\_43\_TOTAL | Total number of times the prefix written was = 43 |
| 876 | PREFIX\_NUM\_44\_TOTAL | Total number of times the prefix written was = 44 |
| 877 | PREFIX\_NUM\_45\_TOTAL | Total number of times the prefix written was = 45 |
| 878 | PREFIX\_NUM\_46\_TOTAL | Total number of times the prefix written was = 46 |
| 879 | PREFIX\_NUM\_47\_TOTAL | Total number of times the prefix written was = 47 |
| 880 | PREFIX\_NUM\_48\_TOTAL | Total number of times the prefix written was = 48 |
| 881 | PREFIX\_NUM\_49\_TOTAL | Total number of times the prefix written was = 49 |
| 882 | PREFIX\_NUM\_50\_TOTAL | Total number of times the prefix written was = 50 |
| 883 | PREFIX\_NUM\_51\_TOTAL | Total number of times the prefix written was = 51 |
| 884 | PREFIX\_NUM\_52\_TOTAL | Total number of times the prefix written was = 52 |
| 885 | PREFIX\_NUM\_53\_TOTAL | Total number of times the prefix written was = 53 |
| 886 | PREFIX\_NUM\_54\_TOTAL | Total number of times the prefix written was = 54 |
| 887 | PREFIX\_NUM\_55\_TOTAL | Total number of times the prefix written was = 55 |
| 888 | PREFIX\_NUM\_56\_TOTAL | Total number of times the prefix written was = 56 |
| 889 | PREFIX\_NUM\_57\_TOTAL | Total number of times the prefix written was = 57 |
| 890 | PREFIX\_NUM\_58\_TOTAL | Total number of times the prefix written was = 58 |
| 891 | PREFIX\_NUM\_59\_TOTAL | Total number of times the prefix written was = 59 |
| 892 | PREFIX\_NUM\_60\_TOTAL | Total number of times the prefix written was = 60 |
| 893 | PREFIX\_NUM\_61\_TOTAL | Total number of times the prefix written was = 61 |
| 894 | PREFIX\_NUM\_62\_TOTAL | Total number of times the prefix written was = 62 |
| 895 | PREFIX\_NUM\_63\_TOTAL | Total number of times the prefix written was = 63 |
| 896 | ISF\_INPUT\_COMMANDS | Number of Commands that the engine receives |
| 897 | ISF\_INPUT\_FRAMES | Number of Frames that the engine receives |
| 898 | ISF\_INPUT\_STALL\_TOTAL | Number of cycles the ISF is stalling the engine inbound interface |
| 899 | ISF\_INPUT\_SYSTEM\_STALL\_TOTAL | Number of cycles the system is stalling the engine inbound interface while a command is active |
| 900 | ISF\_OUTPUT\_BACKPRESSURE\_TOTAL | Number of cycles the ISF outbound interface is being stalled by the downstream engine pipeline |
| 901 | ISF\_AUX\_CMD\_COMPRESS\_CTL\_MATCH\_COMMAND\_0 | Number of commands where the AUX\_CMD comp\_ctl and crypto\_ctl fields match the masked value(s) programmed into theCR\_ISF\_AUX\_CMD\_MATCH\_VAL\_0 and CR\_ISF\_AUX\_CMD\_MATCH\_MASK\_0 CSRs. |
| 902 | ISF\_AUX\_CMD\_COMPRESS\_CTL\_MATCH\_COMMAND\_1 | Number of commands where the AUX\_CMD comp\_ctl and crypto\_ctl fields match the masked value(s) programmed into theCR\_ISF\_AUX\_CMD\_MATCH\_VAL\_1 and CR\_ISF\_AUX\_CMD\_MATCH\_MASK\_1 CSRs. |
| 903 | ISF\_AUX\_CMD\_COMPRESS\_CTL\_MATCH\_COMMAND\_2 | Number of commands where the AUX\_CMD comp\_ctl and crypto\_ctl fields match the masked value(s) programmed into theCR\_ISF\_AUX\_CMD\_MATCH\_VAL\_2 and CR\_ISF\_AUX\_CMD\_MATCH\_MASK\_2 CSRs. |
| 904 | ISF\_AUX\_CMD\_COMPRESS\_CTL\_MATCH\_COMMAND\_3 | Number of commands where the AUX\_CMD comp\_ctl and crypto\_ctl fields match the masked value(s) programmed into theCR\_ISF\_AUX\_CMD\_MATCH\_VAL\_3 and CR\_ISF\_AUX\_CMD\_MATCH\_MASK\_3 CSRs. |
| 960 | HUF\_COMP\_LZ77D\_PTR\_LEN\_256\_TOTAL | Number of 256 byte length pointers received |
| 961 | HUF\_COMP\_LZ77D\_PTR\_LEN\_128\_TOTAL | Number of 128 byte length pointers received |
| 962 | HUF\_COMP\_LZ77D\_PTR\_LEN\_64\_TOTAL | Number of 64 byte length pointers received |
| 963 | HUF\_COMP\_LZ77D\_PTR\_LEN\_32\_TOTAL | Number of 32 byte length pointers received |
| 964 | HUF\_COMP\_LZ77D\_PTR\_LEN\_11\_TOTAL | Number of 11 byte length pointers received |
| 965 | HUF\_COMP\_LZ77D\_PTR\_LEN\_10\_TOTAL | Number of 10 byte length pointers received |
| 966 | HUF\_COMP\_LZ77D\_PTR\_LEN\_9\_TOTAL | Number of 9 byte length pointers received |
| 967 | HUF\_COMP\_LZ77D\_PTR\_LEN\_8\_TOTAL | Number of 8 byte length pointers received |
| 968 | HUF\_COMP\_LZ77D\_PTR\_LEN\_7\_TOTAL | Number of 7 byte length pointers received |
| 969 | HUF\_COMP\_LZ77D\_PTR\_LEN\_6\_TOTAL | Number of 6 byte length pointers received |
| 970 | HUF\_COMP\_LZ77D\_PTR\_LEN\_5\_TOTAL | Number of 5 byte length pointers received |
| 971 | HUF\_COMP\_LZ77D\_PTR\_LEN\_4\_TOTAL | Number of 4 byte length pointers received |
| 972 | HUF\_COMP\_LZ77D\_PTR\_LEN\_3\_TOTAL | Number of 3 byte length pointers received |
| 973 | HUF\_COMP\_LZ77D\_LANE\_4\_LITERALS\_TOTAL | Total count of 4 literals received per clock |
| 974 | HUF\_COMP\_LZ77D\_LANE\_3\_LITERALS\_TOTAL | Total count of 3 literals received per clock |
| 975 | HUF\_COMP\_LZ77D\_LANE\_2\_LITERALS\_TOTAL | Total count of 2 literals received per clock |
| 976 | HUF\_COMP\_LZ77D\_LANE\_1\_LITERALS\_TOTAL | Total count of single literal received per clock |
| 977 | HUF\_COMP\_LZ77D\_PTRS\_TOTAL | Number of Pointers received |
| 978 | HUF\_COMP\_LZ77D\_FRM\_IN\_TOTAL | Number of Frames received |
| 979 | HUF\_COMP\_LZ77D\_FRM\_OUT\_TOTAL | Number of Frames sent |
| 980 | HUF\_COMP\_LZ77D\_STALL\_STB\_TOTAL | Number of Stall cycles at the LZ77 Decompressor |

Table : CCEIP Statistic Events Description

|  |  |  |
| --- | --- | --- |
| Stat Event Number | Stat Event Name | Stat Event Description |
| 0 | CKMIC\_IV\_MISMATCH\_FRAME | This should only Trigger in Decrypt Block of CCEIP Engine. This happens when random IV generated by Encrypt Block does not match the IV in Footer.It is mainly due to bit errors. |
| 1 | CKMIC\_ENGINE\_ID\_MISMATCH\_FRAME | Engine ID in CMD TLV does not match Engine ID in Key TLV. |
| 2 | CKMIC\_SEQ\_ID\_MISMATCH\_FRAME | Sequence ID in CMD TLV does not match Engine ID in Key TLV. |
| 3 | CKMIC\_HMAC\_SHA256\_TAG\_FAIL\_FRAME | HMAC-SHA256 Tag was generated and it failed against Footer Tag upon comparison. |
| 4 | CKMIC\_SHA256\_TAG\_FAIL\_FRAME | SHA256 Tag was generated and it failed against Footer Tag upon comparison. |
| 5 | CKMIC\_GMAC\_TAG\_FAIL\_FRAME | GMAC Tag was generated and it failed against Footer Tag upon comparison. |
| 6 | CKMIC\_GCM\_TAG\_FAIL\_FRAME | GCM Tag was generated and it failed against Footer Tag upon comparison. |
| 7 | CKMIC\_AUTH\_NOP\_FRAME | GCM Tag was generated and it failed against Footer Tag upon comparison. |
| 8 | CKMIC\_AUTH\_HMAC\_SHA256\_FRAME | HMAC-SHA256 Authentication Operation was requested by a frame. |
| 9 | CKMIC\_AUTH\_SHA256\_FRAME | SHA256 Authentication Operation was requested by a frame. |
| 10 | CKMIC\_AUTH\_AES\_GMAC\_FRAME | GMAC Operation was requested by a frame. |
| 11 | CKMIC\_CIPH\_NOP\_FRAME | NOP Operation was requested by a frame. |
| 12 | CKMIC\_CIPH\_AES\_XEX\_FRAME | AES-256-XEX Operation was requested by a frame. |
| 13 | CKMIC\_CIPH\_AES\_XTS\_FRAME | AES-256-XTS Operation was requested by a frame. |
| 14 | CKMIC\_CIPH\_AES\_GCM\_FRAME | AES-256-GCM Operation was requested by a frame. |
| 64 | CRCG0\_RAW\_CHSUM\_GOOD\_TOTAL | Total number of raw data chksum good |
| 65 | CRCG0\_RAW\_CHSUM\_ERROR\_TOTAL | Total number of raw data chksum errors |
| 66 | CRCG0\_CRC64E\_CHSUM\_GOOD\_TOTAL | Total number of crc64e data chksum good |
| 67 | CRCG0\_CRC64E\_CHSUM\_ERROR\_TOTAL | Total number of crc64e data chksum errors |
| 68 | CRCG0\_ENC\_CHSUM\_GOOD\_TOTAL | Total number of crc encap data chksum good; |
| 69 | CRCG0\_ENC\_CHSUM\_ERROR\_TOTAL | Total number of crc encap data chksum errors |
| 70 | CRCG0\_NVME\_CHSUM\_GOOD\_TOTAL | Total number of nvme data chksum good; |
| 71 | CRCG0\_NVME\_CHSUM\_ERROR\_TOTAL | Total number of nvme data chksum errors |
| 88 | CRCC0\_RAW\_CHSUM\_GOOD\_TOTAL | Total number of raw data chksum good |
| 89 | CRCC0\_RAW\_CHSUM\_ERROR\_TOTAL | Total number of raw data chksum errors |
| 90 | CRCC0\_CRC64E\_CHSUM\_GOOD\_TOTAL | Total number of crc64e data chksum good |
| 91 | CRCC0\_CRC64E\_CHSUM\_ERROR\_TOTAL | Total number of crc64e data chksum errors |
| 92 | CRCC0\_ENC\_CHSUM\_GOOD\_TOTAL | Total number of crc encap data chksum good; |
| 93 | CRCC0\_ENC\_CHSUM\_ERROR\_TOTAL | Total number of crc encap data chksum errors |
| 94 | CRCC0\_NVME\_CHSUM\_GOOD\_TOTAL | Total number of nvme data chksum good; |
| 95 | CRCC0\_NVME\_CHSUM\_ERROR\_TOTAL | Total number of nvme data chksum errors |
| 96 | CG\_ENGINE\_ERROR\_COMMAND | Number of commands with engine related errors that the engine outputs |
| 97 | CG\_SELECT\_ENGINE\_ERROR\_COMMAND | Number of commands with programmable engine related error codes that the engine outputs. The error code and associated mask is configured in the CR\_CG\_DEBUG\_CTL\_CONFIG CSR. |
| 98 | CG\_SYSTEM\_ERROR\_COMMAND | Number of commands with system related errors that the engine outputs |
| 99 | CG\_CQE\_OUTPUT\_COMMAND | Number of CQE TLVs that the engine outputs |
| 320 | HUFD\_FE\_XP9\_FRM\_TOTAL | Number of XP9 frames received |
| 321 | HUFD\_FE\_XP9\_BLK\_TOTAL | Number of XP9 Blocks received |
| 322 | HUFD\_FE\_XP9\_RAW\_FRM\_TOTAL | Number of XP9 Raw frames received |
| 323 | HUFD\_FE\_XP10\_FRM\_TOTAL | Number of XP10 Frames received |
| 324 | HUFD\_FE\_XP10\_FRM\_PFX\_TOTAL | Number of Prefix TLVs received |
| 325 | HUFD\_FE\_XP10\_FRM\_PDH\_TOTAL | Number of Predefined Huffman TLVs received |
| 326 | HUFD\_FE\_XP10\_BLK\_TOTAL | Number of XP10 Blocks received |
| 327 | HUFD\_FE\_XP10\_RAW\_BLK\_TOTAL | Number of XP10 Raw Blocks received |
| 328 | HUFD\_FE\_GZIP\_FRM\_TOTAL | Number of GZIP Frames received |
| 329 | HUFD\_FE\_GZIP\_BLK\_TOTAL | Number of GZIP Blocks received |
| 330 | HUFD\_FE\_GZIP\_RAW\_BLK\_TOTAL | Number of GZIP Raw Blocks received |
| 331 | HUFD\_FE\_ZLIB\_FRM\_TOTAL | Number of ZLIB Frames received |
| 332 | HUFD\_FE\_ZLIB\_BLK\_TOTAL | Number of ZLIB Blocks received |
| 333 | HUFD\_FE\_ZLIB\_RAW\_BLK\_TOTAL | Number of ZLIB Raw Blocks received |
| 334 | HUFD\_FE\_CHU4K\_TOTAL | Number of CHU4K frames received |
| 335 | HUFD\_FE\_CHU8K\_TOTAL | Number of CHU8K frames received |
| 336 | HUFD\_FE\_CHU4K\_RAW\_TOTAL | Number of CHU4K Raw frames received |
| 337 | HUFD\_FE\_CHU8K\_RAW\_TOTAL | Number of CHU8K Raw frames received |
| 338 | HUFD\_FE\_PFX\_CRC\_ERR\_TOTAL | Number of Prefix TLVs received with CRC error. |
| 339 | HUFD\_FE\_PHD\_CRC\_ERR\_TOTAL | Number of Pre-determined Huffman TLVs received with CRC error. |
| 340 | HUFD\_FE\_XP9\_CRC\_ERR\_TOTAL | Number of XP9 frames received with header CRC error. |
| 341 | HUFD\_HTF\_XP9\_SIMPLE\_SHORT\_BLK\_TOTAL | Number of XP9 blocks with simple encoding of the short symbols |
| 342 | HUFD\_HTF\_XP9\_RETRO\_SHORT\_BLK\_TOTAL | Number of XP9 blocks with retro encoding of the short symbols |
| 343 | HUFD\_HTF\_XP9\_SIMPLE\_LONG\_BLK\_TOTAL | Number of XP9 blocks with simple encoding of the long symbols |
| 344 | HUFD\_HTF\_XP9\_RETRO\_LONG\_BLK\_TOTAL | Number of XP9 blocks with retro encoding of the long symbols |
| 345 | HUFD\_HTF\_XP10\_SIMPLE\_SHORT\_BLK\_TOTAL | Number of XP10 blocks with simple encoding of the short symbols |
| 346 | HUFD\_HTF\_XP10\_RETRO\_SHORT\_BLK\_TOTAL | Number of XP10 blocks with retro encoding of the short symbols |
| 347 | HUFD\_HTF\_XP10\_PREDEF\_SHORT\_BLK\_TOTAL | Number of XP10 blocks with predef encoding of the short symbols |
| 348 | HUFD\_HTF\_XP10\_SIMPLE\_LONG\_BLK\_TOTAL | Number of XP10 blocks with simple encoding of the long symbols |
| 349 | HUFD\_HTF\_XP10\_RETRO\_LONG\_BLK\_TOTAL | Number of XP10 blocks with retro encoding of the long symbols |
| 350 | HUFD\_HTF\_XP10\_PREDEF\_LONG\_BLK\_TOTAL | Number of XP10 blocks with predef encoding of the long symbols |
| 351 | HUFD\_HTF\_CHU4K\_SIMPLE\_SHORT\_BLK\_TOTAL | Number of CHU4K blocks with simple encoding of the short symbols |
| 352 | HUFD\_HTF\_CHU4K\_RETRO\_SHORT\_BLK\_TOTAL | Number of CHU4K blocks with retro encoding of the short symbols |
| 353 | HUFD\_HTF\_CHU4K\_PREDEF\_SHORT\_BLK\_TOTAL | Number of CHU4K blocks with predef encoding of the short symbols |
| 354 | HUFD\_HTF\_CHU4K\_SIMPLE\_LONG\_BLK\_TOTAL | Number of CHU4K blocks with simple encoding of the long symbols |
| 355 | HUFD\_HTF\_CHU4K\_RETRO\_LONG\_BLK\_TOTAL | Number of CHU4K blocks with retro encoding of the long symbols |
| 356 | HUFD\_HTF\_CHU4K\_PREDEF\_LONG\_BLK\_TOTAL | Number of CHU4K blocks with predef encoding of the long symbols |
| 357 | HUFD\_HTF\_CHU8K\_SIMPLE\_SHORT\_BLK\_TOTAL | Number of CHU8K blocks with simple encoding of the short symbols |
| 358 | HUFD\_HTF\_CHU8K\_RETRO\_SHORT\_BLK\_TOTAL | Number of CHU8K blocks with retro encoding of the short symbols |
| 359 | HUFD\_HTF\_CHU8K\_PREDEF\_SHORT\_BLK\_TOTAL | Number of CHU8K blocks with predef encoding of the short symbols |
| 360 | HUFD\_HTF\_CHU8K\_SIMPLE\_LONG\_BLK\_TOTAL | Number of CHU8K blocks with simple encoding of the long symbols |
| 361 | HUFD\_HTF\_CHU8K\_RETRO\_LONG\_BLK\_TOTAL | Number of CHU8K blocks with retro encoding of the long symbols |
| 362 | HUFD\_HTF\_CHU8K\_PREDEF\_LONG\_BLK\_TOTAL | Number of CHU8K blocks with predef encoding of the long symbols |
| 363 | HUFD\_HTF\_DEFLATE\_DYNAMIC\_BLK\_TOTAL | Number of DEFLATE blocks with dynamic encoding |
| 364 | HUFD\_HTF\_DEFLATE\_FIXED\_BLK\_TOTAL | Number of DEFLATE blocks with fixed encoding |
| 365 | HUFD\_MTF\_0\_TOTAL | Number of MTF[0] occurences |
| 366 | HUFD\_MTF\_1\_TOTAL | Number of MTF[1] occurences |
| 367 | HUFD\_MTF\_2\_TOTAL | Number of MTF[2] occurences |
| 368 | HUFD\_MTF\_3\_TOTAL | Number of MTF[3] occurences |
| 369 | HUFD\_FE\_FHP\_STALL\_TOTAL | Number of stall cycles at the FHP |
| 370 | HUFD\_FE\_LFA\_STALL\_TOTAL | Number of stall cycles at the LFA |
| 371 | HUFD\_HTF\_PREDEF\_STALL\_TOTAL | Number of stall cycles at the HTF predef interface |
| 372 | HUFD\_HTF\_HDR\_DATA\_STALL\_TOTAL | Number of stall cycles at the HTF header data interface |
| 373 | HUFD\_HTF\_HDR\_INFO\_STALL\_TOTAL | Number of stall cycles at the HTF header info interface |
| 374 | HUFD\_SDD\_INPUT\_STALL\_TOTAL | Number of stall cycles at the SDD input interface |
| 375 | HUFD\_SDD\_BUF\_FULL\_STALL\_TOTAL | Number of stall cycles in the SDD due to circular bit buffer full |
| 384 | LZ77D\_PTR\_LEN\_256\_TOTAL | Number of 256 byte length pointers received |
| 385 | LZ77D\_PTR\_LEN\_128\_TOTAL | Number of 128 byte length pointers received |
| 386 | LZ77D\_PTR\_LEN\_64\_TOTAL | Number of 64 byte length pointers received |
| 387 | LZ77D\_PTR\_LEN\_32\_TOTAL | Number of 32 byte length pointers received |
| 388 | LZ77D\_PTR\_LEN\_11\_TOTAL | Number of 11 byte length pointers received |
| 389 | LZ77D\_PTR\_LEN\_10\_TOTAL | Number of 10 byte length pointers received |
| 390 | LZ77D\_PTR\_LEN\_9\_TOTAL | Number of 9 byte length pointers received |
| 391 | LZ77D\_PTR\_LEN\_8\_TOTAL | Number of 8 byte length pointers received |
| 392 | LZ77D\_PTR\_LEN\_7\_TOTAL | Number of 7 byte length pointers received |
| 393 | LZ77D\_PTR\_LEN\_6\_TOTAL | Number of 6 byte length pointers received |
| 394 | LZ77D\_PTR\_LEN\_5\_TOTAL | Number of 5 byte length pointers received |
| 395 | LZ77D\_PTR\_LEN\_4\_TOTAL | Number of 4 byte length pointers received |
| 396 | LZ77D\_PTR\_LEN\_3\_TOTAL | Number of 3 byte length pointers received |
| 397 | LZ77D\_LANE\_1\_LITERALS\_TOTAL | Total count of single literal received per clock |
| 398 | LZ77D\_LANE\_2\_LITERALS\_TOTAL | Total count of 2 literals received per clock |
| 399 | LZ77D\_LANE\_3\_LITERALS\_TOTAL | Total count of 3 literals received per clock |
| 400 | LZ77D\_LANE\_4\_LITERALS\_TOTAL | Total count of 4 literals received per clock |
| 401 | LZ77D\_PTRS\_TOTAL | Number of Pointers received |
| 402 | LZ77D\_FRM\_IN\_TOTAL | Number of Frames received |
| 403 | LZ77D\_FRM\_OUT\_TOTAL | Number of Frames sent |
| 404 | LZ77D\_STALL\_TOTAL | Number of Stall cycles at the LZ77 Decompressor |
| 448 | DECRYPT\_IV\_MISMATCH\_FRAME | This should only Trigger in Decrypt Block of CCEIP Engine. This happens when random IV generated by Encrypt Block does not match the IV in Footer. It is mainly due to bit errors. |
| 449 | DECRYPT\_ENGINE\_ID\_MISMATCH\_FRAME | Engine ID in CMD TLV does not match Engine ID in Key TLV. |
| 450 | DECRYPT\_SEQ\_ID\_MISMATCH\_FRAME | Sequence ID in CMD TLV does not match Engine ID in Key TLV. |
| 451 | DECRYPT\_HMAC\_SHA256\_TAG\_FAIL\_FRAME | HMAC-SHA256 Tag was generated and it failed against Footer Tag upon comparison. |
| 452 | DECRYPT\_SHA256\_TAG\_FAIL\_FRAME | SHA256 Tag was generated and it failed against Footer Tag upon comparison. |
| 453 | DECRYPT\_GMAC\_TAG\_FAIL\_FRAME | GMAC Tag was generated and it failed against Footer Tag upon comparison. |
| 454 | DECRYPT\_GCM\_TAG\_FAIL\_FRAME | GCM Tag was generated and it failed against Footer Tag upon comparison. |
| 455 | DECRYPT\_AUTH\_NOP\_FRAME | GCM Tag was generated and it failed against Footer Tag upon comparison. |
| 456 | DECRYPT\_AUTH\_HMAC\_SHA256\_FRAME | HMAC-SHA256 Authentication Operation was requested by a frame. |
| 457 | DECRYPT\_AUTH\_SHA256\_FRAME | SHA256 Authentication Operation was requested by a frame. |
| 458 | DECRYPT\_AUTH\_AES\_GMAC\_FRAME | GMAC Operation was requested by a frame. |
| 459 | DECRYPT\_CIPH\_NOP\_FRAME | NOP Operation was requested by a frame. |
| 460 | DECRYPT\_CIPH\_AES\_XEX\_FRAME | AES-256-XEX Operation was requested by a frame. |
| 461 | DECRYPT\_CIPH\_AES\_XTS\_FRAME | AES-256-XTS Operation was requested by a frame. |
| 462 | DECRYPT\_CIPH\_AES\_GCM\_FRAME | AES-256-GCM Operation was requested by a frame. |
| 512 | OSF\_DATA\_INPUT\_STALL\_TOTAL | Number of cycles the OSF is stalling the inbound data path interface |
| 513 | OSF\_CG\_INPUT\_STALL\_TOTAL | Number of cycles the OSF is stalling the inbound completion interface from the CG |
| 514 | OSF\_OUTPUT\_BACKPRESSURE\_TOTAL | Number of cycles the OSF engine outbound interface is being stalled by the system |
| 515 | OSF\_OUTPUT\_STALL\_TOTAL | Number of cycles the OSF is stalling the engine outbound interface |
| 896 | ISF\_INPUT\_COMMANDS | Number of Commands that the engine receives |
| 897 | ISF\_INPUT\_FRAMES | Number of Frames that the engine receives |
| 898 | ISF\_INPUT\_STALL\_TOTAL | Number of cycles the ISF is stalling the engine inbound interface |
| 899 | ISF\_INPUT\_SYSTEM\_STALL\_TOTAL | Number of cycles the system is stalling the engine inbound interface while a command is active |
| 900 | ISF\_OUTPUT\_BACKPRESSURE\_TOTAL | Number of cycles the ISF outbound interface is being stalled by the downstream engine pipeline |
| 901 | ISF\_AUX\_CMD\_COMPRESS\_CTL\_MATCH\_COMMAND\_0 | Number of commands where the AUX\_CMD comp\_ctl and crypto\_ctl fields match the masked value(s) programmed into theCR\_ISF\_AUX\_CMD\_MATCH\_VAL\_0 and CR\_ISF\_AUX\_CMD\_MATCH\_MASK\_0 CSRs. |
| 902 | ISF\_AUX\_CMD\_COMPRESS\_CTL\_MATCH\_COMMAND\_1 | Number of commands where the AUX\_CMD comp\_ctl and crypto\_ctl fields match the masked value(s) programmed into theCR\_ISF\_AUX\_CMD\_MATCH\_VAL\_1 and CR\_ISF\_AUX\_CMD\_MATCH\_MASK\_1 CSRs. |
| 903 | ISF\_AUX\_CMD\_COMPRESS\_CTL\_MATCH\_COMMAND\_2 | Number of commands where the AUX\_CMD comp\_ctl and crypto\_ctl fields match the masked value(s) programmed into theCR\_ISF\_AUX\_CMD\_MATCH\_VAL\_2 and CR\_ISF\_AUX\_CMD\_MATCH\_MASK\_2 CSRs. |
| 904 | ISF\_AUX\_CMD\_COMPRESS\_CTL\_MATCH\_COMMAND\_3 | Number of commands where the AUX\_CMD comp\_ctl and crypto\_ctl fields match the masked value(s) programmed into theCR\_ISF\_AUX\_CMD\_MATCH\_VAL\_3 and CR\_ISF\_AUX\_CMD\_MATCH\_MASK\_3 CSRs. |

Table : CDDIP Statistic Events Description

## Debug Support

All counters can be read and cleared via SW.

## Interfaces

The SA interfaces are shown in Table 34. With the exception of the Events interface, which is just a collection of single-bit signals from many of the modules, a definition of each interface type is available in Interface Descriptions.

|  |  |  |
| --- | --- | --- |
| Interface  Type | Data  Width | Notes |
| Events |  | Event pulses from all of the modules. A complete list is found  in Table 32 and Table 33. |
| RBUS In | 32 | CSR Access |
| RBUS Out | 32 | CSR Access |

Table : SA Interfaces

# Scheduler Update (SU)

## Features

* Creates a per-Huffman Coding block Scheduler Update TLV that is sent to the centralized Scheduler in the system for both compression in the CCEIP and decompression in the CDDIP over a 8-bit AXI4S master interface.
* Debug support
  + 8-entry Scheduler Update TLV history buffer

## Description

The Scheduler Update (SU) block provides a SCH TLV with inflation/deflation information (i.e. bytes\_in and bytes\_out) back to the external (i.e. in the system) centralized scheduler via the Scheduler Update AXI4S interface. In the CCEIP, the Huffman Compression engine provides the raw information used to create the SCH TLV on Huffman Coding block boundaries for frames that it compresses and on 8KB boundaries for raw frames. The XP10 Decompression engine provides the same information on Huffman Coding block boundaries in the CDDIP and a final EOT message. For raw frames, it sends a single message with BYTES\_IN=BYTES\_OUT and the EOT message is suppressed.

The scheme described above results in the centralized Scheduler receiving a SCH TLV for each Huffman coding block when the frame is not raw. It was chosen to provide timely updates to the Scheduler. The Huffman coding blocks are reasonable sized and only providing a single TLV per frame would result in an update latency for large frames that is too long.

The Huffman Compression and XP10 Decompression engines provide the raw information over an internal interface shown in Table 44. The SU stores each single beat message in a 64 entry FIFO and then re-formats the message into the SCH TLV format shown in Table 36, which is compatible with the 8-bit AXI4S master interface of the SU. Table 35 shows the format of the SCH TLV in a per-field format for clarity.

|  |  |  |  |
| --- | --- | --- | --- |
| **Fields** | **Size** | **Word** | **Notes** |
| Type | 8 | 0 | Value for SCH TLV is 0x1a |
| Length | 8 | 0 | 0x05 |
| Sequence Num | 8 | 0 | Increments per RQE |
| Engine ID | 4 | 0 | 0-3 CCEIP, 4-7 CDDIP |
| Unused | 4 | 0 | 0x0 |
| Frame Num | 11 | 0 | 0x000 |
| Scheduler Handle | 16 | 0 | From RQE |
| Unused | 2 | 0 | 0x0 |
| Command Complete | 1 | 0 | 1 for last SU TLV of each command |
| Parity | 2 | 0 | BIP-2 (Even) |
| Bytes Out | 24 | 1 | Size of Huffman Coding block out of  compressor/decompressor |
| Unused | 8 | 1 | 0x00 |
| Bytes In | 24 | 1 | Size of Huffman Coding block into  compressor/decompressor |
| Unused | 8 | 1 | 0x00 |
| Basis | 24 | 2 | Size of Huffman Coding block into  compressor/decompressor without any  additional data such as predetermined prefix |
| Unused | 8 | 2 | 0x00 |

Table : SCH TLV Format (64-bit words)

|  |  |
| --- | --- |
| **8-bit Beat** | **Contents** |
| 0 | type |
| 1 | length |
| 2 | seq |
| 3 | {unused, eng} |
| 4 | frame[7:0] (all 0’s) |
| 5 | {scheduler\_handle[4:0,] frame[10:8]} (frame all 0’s) |
| 6 | scheduler\_handle[12:5] |
| 7 | {bip2, command\_complete,unused,scheduler\_handle[15:13]} |
| 8 | bytes\_out[7:0] |
| 9 | bytes\_out[15:8] |
| 10 | bytes\_out[23:16] |
| 11 | unused |
| 12 | bytes\_in[7:0] |
| 13 | bytes\_in[15:8] |
| 14 | bytes\_in[23:16] |
| 15 | unused |
| 16 | basis[7:0] |
| 17 | basis[15:8] |
| 18 | basis[23:16] |
| 10 | unused |

Table : Serialized SCH TLV Format

## Debug Support

### Debug Counters

The SU provides an aggregate counter of SCH TLVs sent to the scheduler. This is a clear-on-read counter.

### Scheduler Update Message History Buffer

The SU keeps a buffer of the last 8 SCH TLVs sent to the scheduler. The format of each entry is shown in Table 37. The frame number stored is the actual value as opposed to it being forced to all 0’s in the SCH TLV.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Bits** | 107 | 106:96 | 95:80 | 79:72 | 71:48 | 47:24 | 23:0 |
| **Field** | Command  Complete | Frame  Num | Scheduler  Handle | Seq  Num | Basis | Bytes  In | Bytes  Out |

Table : Scheduler Update History Buffer Entry Format

### Miscellaneous Debug Support

For debug purposes, the SU AXI4S AXI can be backpressured by setting *cr\_su\_dbg\_config.force\_ob\_bp* to a 1.

## Self-Test

As part of the self-test scheme described in 6.6, the SU AXI4S master interface is isolated from the rest of the system by de-asserting TVALID and asserting TREADY. As such, the system will not recognize any SCH TLVs and the SU cannot be back-pressured from the system.

## Interrupts

Table 38 shows the interrupt provided by the SU.

|  |  |
| --- | --- |
| **Interrupt** | **Description** |
| Uncorrectable ECC Error | Uncorrectable ECC error detected in the SU Input FIFO RAM |

Table : SU Interrupts

## Interfaces

The SU interfaces are shown in Table 39. A definition of each interface type is available in Interface Descriptions.

|  |  |  |
| --- | --- | --- |
| Interface  Type | Data  Width | Notes |
| AXI4S Master | 8 | Scheduler Update Message |
| RBUS In | 32 | CSR Access |
| RBUS Out | 32 | CSR Access |
| Scheduler Update | 114 | 2 interfaces. One from the Huffman Compression  block in CCEIP and one from XP10  Decompression block in CDDIP |
| Interrupt | 1 | To CCEIP/CDDIP support block |

Table : SU Interfaces

# CCEIP and CDDIP Support Blocks

## Description

The CCEIP and the CDDIP each have an associated support block that has the following:

* General Purpose CSRs
* Pipeline Status CSR
* Interrupt support CSRs
* Dataflow Mux (CCEIP only)
* Engine IDLE interface
* Provides halt signal to OSF when engine generates an interrupt
* Bus sharing logic for KME interfaces

## Idle Interface

Each engine has an Idle interface that is sourced from the support blocks. This is a single bit that is asserted to a “1” whenever the engine pipeline is not processing any traffic. The engine is considered idle whenever all of the following conditions exist:

* Pipeline is not processing any commands (*cr\_cceip\_64\_support\_pipe\_stat.pipe\_cmds* = 0, *cr\_cddip\_64\_support\_pipe\_stat.pipe\_cmds* = 0).
* All of the Crypto blocks in the engine are not calculating any random numbers.

## KME Interface Support

Figure 1 and Figure 2 show the KME interface to each engine that provides the KEY TLVs to the various crypto blocks (CCEIP: encrypt, decrypt, and CKMIC; CDDIP: decrypt and CKMIC). The support blocks provide the logic that allows this AXI4S interface to be shared among the multiple crypto blocks. The interface is shared by allowing each crypto block to know the state of the TREADY signals from the other block(s) to know when external back pressure exists. When this occurs, the crypto block will not accept new transactions. The support blocks also have the logic that ANDs the TREADY signals from each crypto block to create a single TREADY signal to the KME slave interface.

## Debug Support

## Dataflow Mux

Figure 1 shows the CCEIP Dataflow mux (DF Mux). It is used to configure the CCEIP to output the results of the CRCG #0 (normal CCEIP output) or CRCC #1 (decompressed output for debug). This mux is controlled via *cr\_cceip\_64\_support\_df\_mux\_ctrl.df\_mux\_sel*  (0: Normal compressed output; 1: Decompressed debug data).

## Interrupts

The support blocks source the single engine interrupt that each engine provides. The support blocks receive all of the interrupts from each block and have of the necessary interrupt support registers (i.e. interrupt raw status, interrupt masked status, and interrupt mask) to support the CPU that will service the interrupts.

Whenever an engine interrupt occurs, the support blocks also provide a halt signal to the OSF (6.7 ) that results in the outbound traffic on the AXI4S interface being immediately halted. Since almost all of the interrupts are considered fatal errors, this is a protection mechanism to prevent corrupted traffic from exiting the engine.

## Control and Status Registers

Each support block contains the following general purpose control and status registers:

* Block Id / Revision Id
* Spare R/W register to verify that the CCEIP or CDDIP can be accessed by SW
* Summary Inter-Stage Monitor (ISM) Available register that reports the status of each ISM
* ISM Consumed register used to clear the read status of the ISMs
* Dataflow Mux control register (CCEIP only)
* Pipeline status register
* Interrupt support registers

# Inter-Stage Monitor (ISM)

The ISM is a common block that is designed to allow software to monitor the AXI interface signals between blocks. The design will implement a single SPRAM and treat as a ping pong buffer. Hardware will write to one half of the buffer while software reads the other half.

## Features

* The design is treated as a true ping pong buffer, with no pointers to read.
* The AXI side band signals will be written to the buffer to allow software to determine the end of a frame via the last bit.
* The ISM will have 4 modes of operation:
  + Continuous mode, which will back pressure the input side if software cannot keep up with reading it’s half of the ping pong buffer.
  + Start mode, capture an entire buffer (both halves) after reset and do not overwrite.
  + Stop mode, continually overwrite the entire buffer, readable by software when traffic is halted.
  + Disable mode, monitor is disabled. All pointers are reset, can be used to retrigger operation in start and stop modes.

## Description

The ISM block diagram is shown in Figure 34.



Figure : ISM Block Diagram

Hardware will manage and keep the internal “order” of what half of the ping pong buffer is next to be read, so that only 1 AVAILABLE status bit, 0 or 1 will ever be set when software goes to read it. This makes it so software doesn’t need to “remember” which half of the ping pong buffer was last read. It gets this information directly from the hardware.

The AVAILABLE status bit gets set when either one half is filled or when the EOB indicator occurs. Software will read the descriptor from the start (1st entry) up to the first entry whose descriptor’s EOB bit is set.

Software only needs to poll the AVAILABLE status register, and when any AVAILABLE bit is set, it knows what to read and how to clear it. No state needs to be kept by software.

There will be per ISM registers for the AVAILABLE status and CONSUMED control registers.

The AVAILABLE and CONSUMED bits will also be brought to a top level, shared support block where an additional 2 registers will be instantiated- ISM\_GLOBAL\_AVAILABLE and ISM\_GLOBAL\_CONSUMED.

These 2 registers (each 32 bits wide) will allow us to support up to 16 ISMs via single summary status registers. The ISM\_GLOBAL\_CONSUMED bits will drive down into the ISM of each block and be OR’d into the individual ISM version of these same control signals. Thus we can use the top level to run these blocks (which is how we will use it) and they can standalone as future IP’s.

The ISM is constructed from other common, “nx” building blocks, including the 1R1W indirect access controller and FIFO credit manager. This provides a common software interface to access the ping pong buffer.

## Interface Description

|  |  |  |
| --- | --- | --- |
| Name | I/O | Description |
| **Clocks and resets** | | |
| clk | I | 800 MHz clock |
| rst\_n | I | Active low reset |
| **Stream Assembler** | | |
| reg\_addr[n:0] | I | Software access address from RDB, width is block implementation dependent |
| wr\_stb | I | Write strobe from RDB |
| rd\_stb | I | Read strobe from RDB |
| wr\_data[31:0] | I | Write data from RDB |
| cmd\_op[3:0] | I | Software access command operand, part of the standard memory indirect access controller |
| cmd\_addr[7:0] | I | Software access command address, part of the standard memory indirect access controller |
| im\_vld | I | Valid signal for the interface being monitored |
| im\_din[81:0] | I | Interface data bus being monitored, 64 bits of AXI data plus 28 bits of descriptor information |
| im\_comsumed[1:0] | I | From the global ISM registers, used by software to release control of the ping pong buffer back to hardware |
| im\_config[10:0] | I | From RDB block, 2 bits for ISM mode configuration (start, stop, continuous or disabled), 9 bits for the credit manager |
| stat\_code[2:0] | O | Indirect memory access status code to RDB |
| stat\_datawords[4:0] | O | Indirect memory access data to RDB |
| rd\_data[31:0] | O | Read data to RDB |
| im\_rdy | O | Back pressure signal, should be merged into the AXI control path. Asserted when hardware has filled it’s ping pong buffer. Only asserted in continuous mode. |
| im\_available[1:0] | O | Ping pong buffer available status for the Global ISM registers. |
| im\_status[9:0] | O | Status to the local RDB, includes available, overflow and write pointer. Note, the write pointer is not required by software as the EOB and last bits in the descriptor are used to determine the end of the data. |

Table : ISM Interface Signals

# TLV Parser

The TLV Parser is a common block that can be instantiated in any engine block. The TLV Parser is used to split out of the inbound TLV data stream, into TLV’s of interest by the using logic and those which are simply passed through. Additionally, the TLV Parser will reassemble the two data streams back into one outbound AXI stream.

## Features

* Common subblock
* Optional AXI4 Streaming Slave sub-block
* TLV BIP-2 error checking
* Configurable TLV parse action on per Type number basis
* TLV Order Number generated and used internally to reassemble outbound frames
* Additional sideband TLV signals generated and used internally i.e. Start of TLV, End of TLV, etc
* Configurable TLV insertion
* Inbound data stream split into pass-through and User inbound FIFOs
* Debug Support
* Parameter FIFO sizes

## Description

The TLV parser accepts an inbound TLV frame stack up, identifies the individual TLV types and stores the TLV and frame data in either of two input FIFO’s. The TLV parser will accept TLV frames from the User logic block, the pass-through data and reassemble the output AXI data stream.

### AXI4S Slave

An AXI4S Slave block will be available as a separate block which can be instantiated into the User Block. The inbound AXI data frames are written into an inbound FIFO. A programmable watermark exists to apply backpressure via the AXI Streaming interface READY signal. This block will have all inputs and outputs registered*.*

### TLV Disassembler

The TLV Disassembler block is composed of two sub-blocks, described below, plus the passthrough FIFO and User inbound FIFO. This block could be instantiated without the other parts of the TLV parser if the user only wanted the functions of this block without the TLV reassembly function.

#### TLV Identifier

The input interface to the TLV Parser will be the AXI Slave Read side FIFO. If the AXI4S Slave is not used then the user logic must mimic this FIFO interface. The TLV Identifier will include the following:

* Logic to read input data from the FIFO interface into a register stage
* Perform BIP-2 (even parity) checking of the TLV BIP-2 field. An ERROR signal will be output when errors are detected.
* Generate a sideband order number for each TLV type. The order number begins at 1 for the first TLV type received and increments by 1 for each successive TLV type. The order number will reset to 0 after the AXI4S signal tlast is received.
* Generate sideband signals: Start of TLV(SOT) and End of TLV (EOT) for each TLV type number. Note: the inbound AXI user bits are used to generate the SOT/EOT signals.
* Note inbound frame data and axi tid are persevered unmodified.

#### TLV Splitter

The TLV Splitter takes the data from the TLV identification logic and writes it into the Passthrough FIFO and/or the User inbound FIFO. A configuration vector provides the information as to where to store the input TLV. This input must be driven by the User Logic to “personalize” the TLV Parser to the User application. These inputs could be just tied off or driven by RBUS regs.

This 32 bit configuration vector will consist of a 2 bit code for each TLV type number, where the code is defined as follows:

|  |  |
| --- | --- |
|  | TLV Parse Action |
| 00 | Replicate – The TLV will be written into both FIFO’s. The User logic can use this TLV but must not write out a TLV corresponding to this Order #. The TLV from the Passthrough FIFO will be used in the output data stream, unmodified. |
| 01 | Pass-through – The TLV will be written into only the Passthrough FIFO and will not be available to the User Logic |
| 10 | Modify – The TLV will be written into the Usr inbound FIFO only. The User logic can use this TLV and must write out a TLV corresponding to this Order #. This is used for TLV’s the User logic will modify. |
| 11 | Delete – The TLV will be written into the Usr inbound FIFO but no order number is assigned. This action is used when the user logic will remove a TLV from the output bound data stream. |

### TLV Reassembler

The TLV Reassembler will reassemble the outbound data stream from the Pass-through FIFO and the User generated TLV’s using the TLV Order Number.

The TLV Reassembler will work using the following rules to arbitrate the two input interfaces:

1. A new outbound frame always begins with TLV Order number 1.
2. The reassembler will keep track of the next order number expected by incrementing the last order number written into the output FIFO.
3. The reassembler will keep track of the last tlv number written into the output FIFO.
4. If either interface has a TLV available, the interface that matches the next order will be serviced.
5. If both Pass-through and User interfaces have TLV’s available, and neither matches the next order number, then the interface with the lowest Order Number will be serviced first. Note this allows an order number to be skipped, i.e. deleted TLV’s.
6. If the User interface has a TLV available and the insert bit is set, then the user tlv type number field will be used as the “tlv after” number. When the next tlv number equals the user “tlv after” number then insert tlv will be processed. Note the order number is not used in this case. After the tlv is inserted the reassembler will return to normal processing.
7. The TLV Reassembler will use the outbound FIFO full flag to stall reading of the Pass-through and User Data.
8. Only the TLV data will be written into the outbound FIFO. The sideband signals, SOT/EOT, TLV order number etc will be stripped prior to the FIFO.

### AXI4S Master

The AXI4S master reads data from the TLVP output FIFO, registers it, and drives the AXI4 Streaming interface. The input tready is used along with the FIFO flags to create the FIFO read.

#### TLV Insertion

To insert a new TLV into the outbound data stream at least one TLV must have the TLV Parse Action set to Modify. When the reassembler reaches the order number assigned to this TLV it will wait for that order number to be written into the usr\_ob fifo interface.

* To insert a new TLV *after* the TLV marked with Modify, the user logic must first write the Modify TLV with the insert bit set high. Then write the “new” TLV, but with the insert bit set low.
* To insert a new TLV *before* the TLV marked with Modify, the user logic must first write the “new” TLV with the insert bit set high. Then write the existing TLV with the insert bit set low.

Note all TLV’s written in this manner must have the order number set to the order number assigned to the inbound TLV, marked with Modify. This is necessary for the TLV reassembler to maintain proper order for all TLV’s processed after the TLV’s are inserted.

##### Example: insert CMD\_IV TLV before Data\_UNK TLV

The following example shows how to insert a “new” CMD\_IV TLV *before* the existing DATA\_UNK TLV. The tlv parser assigns order numbers as TLV’s are received on the incoming interface. The received TLV’s with TLV Parse Action set to REP, MODIFY or DELETE will be sent to the user logic. In this case the user logic will use the incoming DATA\_UNK to create the “new” CMD\_IV TLV. Note it is important that the incoming ordern be copied to the “new” CMD\_IV TLV, as well as the existing DATA\_UNK TLV.

1. The RQE, CMD, FRMD and KEY TLV’s will all arrive at the tlv parser reassembler via the Pass-through interface. The tlv parser will transmit these TLV’s via the outbound interface, in order, using the order number. TLV’s with a Parse Action set to REP will have a copy sent to the user logic, this copy should not be written into the usr\_ob interface.
2. The tlv parser will then wait for order number 5 to appear at the usr\_ob interface. In this case the “new” CMD\_IV TLV will be transmitted next. When the CMD\_IV TLV is transmitted the insert bit is set and the tlv parser arbitration logic will only look to the next TLV from the usr\_ob interface. This ensures that no other TLV can be transmitted out of order, even if there is some delay before the next TLV is written into the usr\_ob interface.
3. Once the DATA\_UNK TLV is written into the usr\_ob interface, the tlv parser will begin transmitting it to the outbound interface. The insert bit is set low, so after the DATA\_UNK is transmitted; the tlv parser is free to look at both interfaces for the next TLV.
4. Finally, the CQE TLV will be transmitted from the pass-through interface to the outbound interface.

Note that it is critical that the final TLV have the AXI4S tlast bit set to complete an axi transaction. The tlv parser uses the incoming tlast to reset the order number, so the next TLV command will begin with order number = 1 (not 0).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Inbound Frame Stackup** | | **TLV Parse Action** | **usr\_ob interface** | | | **Outbound Frame Stackup** |
| ordern |  |  | insert | ordern | tlv type |  |
| 1 | RQE | PASS | -- | -- | -- | RQE |
| 2 | CMD | REP | -- | -- | -- | CMD |
| 3 | FRMD | REP | -- | -- | -- | FRMD |
| 4 | KEY | PASS | -- | -- | -- | KEY |
| 5 | DATA\_UNK | **MODIFY** | 1 | 5 | CMD\_IV | CMD\_IV |
|  |  |  | 0 | 5 | DATA\_UNK | DATA\_UNK |
| 6 | CQE | PASS | -- | -- | -- | CQE |
|  |  |  |  |  |  |  |

##### Example: insert FTR and STAT after DATA\_UNK

The following example shows how to insert a “new” FTR and STAT TLV *after* the existing DATA\_UNK TLV. In this case the user logic will use the incoming DATA\_UNK to create two “new” TLV’s. Note it is important that the incoming order number be copied to the “new” TLV’s, as well as the existing DATA\_UNK TLV.

1. The RQE, CMD, FRMD and KEY TLV’s will all arrive at the tlv parser reassembler via the Pass-through interface. The tlv parser will transmit these TLV’s via the outbound interface, in order, using the order number. TLV’s with a Parse Action set to REP will have a copy sent to the user logic, this copy should not be written into the usr\_ob interface.
2. The tlv parser will then wait for order number 5 to appear at the usr\_ob interface. In this case the existing DATA\_UNK TLV will be transmitted next. When the DATA\_UNK TLV is transmitted the insert bit is set and the tlv parser arbitration logic will only look for the next TLV from the usr\_ob interface. This ensures that no other TLV can be transmitted out of order, even if there is some delay before the next TLV is written into the usr\_ob interface.
3. Next the user logic writes the “new” FTR TLV. The insert bit is set high, so the tlv parser will only look at the usr\_ob interface for the next TLV.
4. Next the user logic writes the “new” STAT TLV. The insert bit is set low, so after the STAT TLV is transmitted; the tlv parser is free to look at both interfaces for the next TLV.
5. Once the DATA\_UNK TLV is written into the usr\_ob interface, the tlv parser will begin transmitting it to the outbound interface.
6. Finally, the CQE TLV will be transmitted from the pass-through interface to the outbound interface.

Note that it is critical that the final TLV have the AXI4S tlast bit set to complete an axi transaction. The tlv parser uses the incoming tlast to reset the order number, so the next TLV command will begin with order number = 1 (not 0).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Inbound Frame Stackup** | | **TLV Parse Action** | **usr\_ob interface** | | | **Outbound Frame Stackup** |
| ordern |  |  | insert | ordern | tlv type |  |
| 1 | RQE | PASS | -- | -- | -- | RQE |
| 2 | CMD | REP | -- | -- | -- | CMD |
| 3 | FRMD | REP | -- | -- | -- | FRMD |
| 4 | KEY | **MODIFY** | 1 | 4 | -- | KEY |
| 5 | DATA\_UNK | **MODIFY** | 1 | 5 | DATA\_UNK | DATA\_UNK |
|  |  |  | 1 | 5 | FTR | FTR |
|  |  |  | 0 | 5 | STAT | STAT |
| 6 | CQE | PASS | -- | -- | -- | CQE |
|  |  |  |  |  |  |  |

### Debug Support

The TLV Parser supports the following TLV Debug Commands:

* Data Corruption
* Tready Modulation

When the TLV Parser performs any data corruption in the TLVP Id module, so the TLV is corrupted before the TLV is split to either/and the user logic or the pass-through interface. The TLV Parse action is not affected by the debug commands.

CMD TLV Word 1 contains a 32-bit debug field, the format is shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Size** | **Word** | **Notes** |
| tlvp\_corrupt | 1 | 1 | USER=0, user logic will corrupt the TLV  TLVP=1, TLV Parser will corrupt the TLV |
| cmd\_mode | 2 | 1 | SINGLE\_ERR=0,  CONTINUOUS\_ERROR=1,  STOP=2,  RSV=3 |
| module\_id | 5 | 1 | Id number of the module to implement debug command |
| cmd\_type | 1 | 1 | DATAPATH=0  Datapath corrupt. ( use this for XOR corruption).  BACKPRESSURE=1  Create backpressure at the module inbound interface per settings in bits [17:0] |
| tlv\_num | 5 | 1 | TLV number to corrupt |
| byte\_num | 10 | 1 | When cmd\_type=BACKPRESSURE, the byte\_num field is defined as the backpressure assert count  For other modes, this value is the byte number to corrupt. Value can be 0 to 1023. |
| byte\_msk | 8 | 1 | When cmd\_type=BACKPRESSURE, the byte\_msk field is defined as the backpressure de-assert count  For other modes, this is the value that will be XOR with the datapath value to create the error injection. |

#### Data Corruption

If all of the following conditions are true, then the TLV Parser will corrupt the specified TLV.

* tlvp\_id\_cmd\_word1.debug.tlvp\_corrupt==TLVP
* tlvp\_id\_cmd\_word1.debug.module\_id == module\_id (i.e. the current module id)
* tlvp\_id\_cmd\_word1.debug.cmd\_type == DATAPATH\_CORRUPT

The incoming TLV data will be modified by XOR-ing the incoming data with the byte\_msk data for the specified bye\_num only.

If the tlvp\_id\_cmd\_word1.debug.cmd\_mode == SINGLE\_ERR, then the TLV Parser will only corrupt the TLV once. If the tlvp\_id\_cmd\_word1.debug.cmd\_mode == CONTINUOUS\_ERROR, then the TLV Parser will corrupt every instance of the TLV until a new CMD TLV is received with the tlvp\_id\_cmd\_word1.debug.cmd\_mode == STOP.

#### Tready Modulation

The TLV Parser can be configured, via the CMD TLV Word 1 to artificially assert backpressure on the incoming AXI interface. The TLVP normally reads AXI data from the Inbound FIFO when space is available in the outbound interfaces. The TLV Parser contains a counter with a programmable on-time and off-time. When enabled, this counter will gate-off the normal reads during the count off-time and allow reads during the on-time. The counter runs continuously, loading the on\_time count, counting down to zero, then loading the off\_time count, counting down to zero, and repeating.

The counter is programmed, dynamically with the CMD TLV Word 1 debug fields.

If all of the following conditions are true, then the TLV Parser counter will be programmed:

* tlvp\_id\_cmd\_word1.debug.tlvp\_corrupt==TLVP
* tlvp\_id\_cmd\_word1.debug.module\_id == module\_id (i.e. the current module id)
* tlvp\_id\_cmd\_word1.debug.cmd\_type == FUNCTIONAL\_ERROR
* tlvp\_id\_cmd\_word1.debug.byte\_num == on\_time count
* tlvp\_id\_cmd\_word1.debug.byte\_msk == off\_time count

## Interfaces

The TLV Parser interfaces are shown below.

| Name | I/O | Description |
| --- | --- | --- |
| **Clocks and resets** |  |  |
| clk | I | 800MHz clock |
| Rst\_n | I | Active low reset |
| **AXI4S Slave** |  |  |
| tvalid | I | TVALID indicates that the master is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted. |
| tlast | I | TLAST indicates the boundary of a packet. |
| tid | I | TID is the data stream identifier that indicates different streams of data. |
| tkeep[7:0] | I | TKEEP is the byte qualifier that indicates whether the content of the associated byte of TDATA is processed as part of the data stream. Associated bytes that have the TKEEP byte qualifier deasserted are null bytes and can be removed from the data stream. |
| tuser[7:0] | I | TUSER is user defined sideband information that can be transmitted alongside the data stream.  TUSER[0] : Start of TLV  TUSER[1] : End of TLV |
| tdata[63:0] | I | TDATA is the primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes. |
| tready | O | TREADY indicates that the slave can accept a transfer in the current cycle. |
| **AXI4S Master** |  |  |
| tvalid | O | TVALID indicates that the master is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted. |
| tlast | O | TLAST indicates the boundary of a packet. |
| tid | O | TID is the data stream identifier that indicates different streams of data. |
| tkeep[7:0] | O | TKEEP is the byte qualifier that indicates whether the content of the associated byte of TDATA is processed as part of the data stream. Associated bytes that have the TKEEP byte qualifier deasserted are null bytes and can be removed from the data stream. |
| tuser[7:0] | O | TUSER is user defined sideband information that can be transmitted alongside the data stream.  TUSER[0] : Start of TLV  TUSER[1] : End of TLV |
| tdata[63:0] | O | TDATA is the primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes. |
| tready | I | TREADY indicates that the slave can accept a transfer in the current cycle. |
| **Config In** |  |  |
| tlv\_parse\_action[32:0] | I | TLV input parse action |
| **TLV Parser Inbound** |  |  |
| tlvp\_ib\_rd | O | read strobe |
| tlvp\_ib\_empty | I | Inbound FIFO is empty |
| tlvp\_ib\_aempty | I | Inbound FIFO almost empty (parameter AEMPTY\_VAL) |
| tlvp\_ib.tlast | O | AXI4S tlast |
| tlvp\_ib.tid | O | AXI4 tid |
| tlvp\_ib.tuser[7:0] | O | AXI4 tuser  TUSER[0] : Start of TLV  TUSER[1] : End of TLV |
| tlvp\_ib.tdata[63:0] | O | User inbound FIFO read data |
| **TLV Parser Outbound** |  |  |
| tlvp\_ob\_rd | I | read strobe |
| tlvp\_ob\_empty | O | TLVP output FIFO empty flag |
| tlvp\_ob\_aempty | O | TLVP output FIFO almost empty (parameter AEMPTY\_VAL) |
| tlvp\_ob.tlast | O | AXI4S tlast |
| tlvp\_ob.tid | O | AXI4 tid |
| tlvp\_ob.tuser[7:0] | O | AXI4 tuser  TUSER[0] : Start of TLV  TUSER[1] : End of TLV |
| tlvp\_ob.tdata[63:0] | O | User inbound FIFO read data |
| **User inbound** |  |  |
| usr\_ib\_rd | I | User inbound FIFO read strobe |
| usr\_ib\_empty | O | User inbound FIFO empty |
| usr\_ib\_aempty | O | User inbound FIFO almost empty (parameter AEMPTY\_VAL) |
| usr\_ib\_tlv.insert | O | TLV insert (will always be 0 for usr\_ib interface) |
| usr\_ib\_tlv.ordern[7:0] | O | TLV order Number |
| usr\_ib\_tlv.typen[7:0] | O | TLV type number |
| usr\_ib\_tlv.sot | O | Start of Type |
| usr\_ib\_tlv.eot | O | End of Type |
| usr\_ib\_tlv.tlast | O | AXI4S tlast |
| usr\_ib\_tlv.tid | O | AXI4S tid |
| usr\_ib\_tlv.tuser[7:0] | O | AXI4S tuser  TUSER[0] : Start of TLV  TUSER[1] : End of TLV |
| usr\_ib\_tlv.tdata[63:0] | O | User inbound FIFO read data |
| **User outbound** |  |  |
| usr\_ob\_full | O | Reassembler FIFO full flag |
| usr\_obafull | O | Reassembler almost full flag (parameter AFULL\_VAL) |
| usr\_ob\_wr | I | User logic write strobe to write TLV to reassembler |
| usr\_ob\_tlv.insert | I | TLV insert indicates a new TLV will be inserted after the typen |
| usr\_ob\_tlv.ordern[7:0] | I | TLV order Number |
| usr\_ob\_tlv.typen[7:0] | I | TLV type number (tlv insert uses this field as after type number) |
| usr\_ob\_tlv.sot | I | Start of Type |
| usr\_ob\_tlv.eot | I | End of Type |
| usr\_ob\_tlv.tlast | I | AXI4S tlast |
| usr\_ob\_tlv.tid | I | AXI4 tid |
| usr\_ob\_tlv.tuser[7:0] | I | AXI4S tuser  TUSER[0] : Start of TLV  TUSER[1] : End of TLV |
| usr\_ob\_tlv.tdata[63:0] | I | TLV data |

## Implementation Details

The Verilog for the TLV parser is written in a hierarchical manner to allow a user to instantiate only the parts that are needed for a specific application. Parameters are included for each FIFO. Typedef structs are used for the TLV interfaces to connect the various components.

### Verilog Files

|  |  |
| --- | --- |
| cr\_tlvp.v | Top level of the TLV Parser |
| cr\_tlvp\_dsm.v | TLVP Parser disassembler module, this includes the cr\_tlvp\_id, cr\_tlvp\_spl, passtrough FIFO and User inbound FIFO. |
| cr\_tlvp\_rsm.v | TLVP Parser reassemble module, this includes a user input FIFO and the tlvp ouput FIFO |
| cr\_tlvp\_id.v | TLVP identification module |
| cr\_tlvp\_spl.v | TLVP splitter module |
| cr\_axi4s\_slave.v | A general AXI4S slave module that connects to a streaming axi bus as a slave and stores received frames in a FIFO. This module provides a registered input interface |
| cr\_axi4s\_mstr.v | A general AXI4S master module that reads axi commands from an external FIFO and drives the streaming axi bus. Note this module really just uses the input axi tready to read from a FIFO and register the output |
| cr\_FIFO\_wrap1.v | A simple wrapper around the nx\_FIFO to convert used\_slots into aempty and afull flags, with parameters |

### Parameters

The following parameters are used in many of the above files. They are described once below, refer to the actual Verilog files of usage.

|  |  |  |
| --- | --- | --- |
| Parameter | Default | Description |
| N\_PT\_ENTRIES | 16 | Depth of the passthrough FIFO |
| N\_PT\_AFULL\_VAL | 3 | Number of locations from full |
| N\_PT\_AEMPTY\_VAL | 1 | Number of locations from empty |
| N\_TM\_ENTRIES | 16 | Depth of the User inbound FIFO |
| N\_TM\_AFULL\_VAL | 3 | Number of locations from full |
| N\_TM\_AEMPTY\_VAL | 1 | Number of locations from empty |
| N\_OF\_ENTRIES | 16 | Depth of the TLVP output FIFO |
| N\_OF\_AFULL\_VAL | 2 | Number of locations from full |
| N\_OF\_AEMPTY\_VAL | 1 | Number of locations from empty |

# Interface Descriptions

The following tables list all of the interfaces for the blocks described in this document.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **I/O** | **Width** | **Description** |
| TID | I | 9 | Data block identifier |
| TVALID | I | 1 | Data transfer valid  0 : Not valid  1 : Valid |
| TLAST | I | 1 | Last data phase of the data block |
| TDATA | I | 64 | Input data |
| TSTRB | I | 8 | Byte strobes corresponding to:  TSTRB[0] : TDATA[7:0]  TSTRB[1] : TDATA[15:8]  …  TSTRB[7] : TDATA[63:56]  Data Byte Valid:  0 : Not valid  1 : Valid  Only thermometer codes allowed (e.g. 0000\_0001 through 1111\_1111) |
| TUSER | I | 8 | TUSER[0] : Start of TLV (first beat)  TUSER[1] : End of TLV (last beat)  TUSER[7:2] : All 0 |
| TREADY | O | 1 | Slave data acceptance for current cycle:  0 : Not Accepted  1 : Accepted |

Table : 64-bit AXI4S Slave Interface

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **I/O** | **Width** | **Description** |
| TID | O | 9 | Data block identifier |
| TVALID | O | 1 | Data transfer valid  0 : Not valid  1 : Valid |
| TLAST | O | 1 | Last data phase of the data block |
| TDATA | O | 64 | Output data |
| TSTRB | O | 8 | Byte strobes corresponding to:  TSTRB[0] : TDATA[7:0]  TSTRB[1] : TDATA[15:8]  …  TSTRB[7] : TDATA[63:0]  Data Byte Valid:  0 : Not valid  1 : Valid  Only thermometer codes allowed (e.g. 0000\_0001 through 1111\_1111) |
| TUSER | I | 8 | TUSER[0] : Start of TLV (first beat)  TUSER[1] : End of TLV (last beat)  TUSER[7:2] : All 0 |
| TREADY | I | 1 | Slave data acceptance for current cycle:  0 : Not Accepted  1 : Accepted |

Table : 64-bit AXI4S Master Interface

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **I/O** | **Width** | **Description** |
| TVALID | O | 1 | Data transfer valid  0 : Not valid  1 : Valid |
| TLAST | O | 1 | Last data phase of SU TLV |
| TDATA | O | 8 | Output data |
| TUSER | I | 2 | TUSER[0] : Start of TLV (first beat)  TUSER[1] : End of TLV (last beat) |
| TREADY | I | 1 | Slave data acceptance for current cycle:  0 : Not Accepted  1 : Accepted |

Table : 8-bit AXI4S Master Interface to Scheduler

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **I/O** | **Width** | **Description** |
| SU\_READY | I | 1 | Scheduler Update ready. High true. |
| VALID | O | 1 |  |
| SCHED\_HANDLE | O | 16 | Scheduler handle from RQE |
| LAST | O | 1 | Last Scheduler Update message for the command |
| TLV\_FRAME\_NUM | O | 11 | TLV Frame Number |
| TLV\_ENGINE\_ID | O | 4 | TLV Engine ID |
| TLV\_SEQ\_NUM | O | 8 | TLV Sequence Number |
| TLV\_BYTES\_IN | O | 24 | Bytes In |
| TLV\_BYTES\_OUT | O | 24 | Bytes out |
| BASIS | O | 24 | Bytes in before adjustment for “predetermined prefix” |

Table : SU Interface from Huffman Compressor/XP10 Decompressor

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **I/O** | **Width** | **Description** |
| addr | I | 32 | Byte Address: address bits [1:0] are not significant, and are always set to “00.” |
| wr\_stb | I | 1 | Write strobe : indicates the start of a write transaction |
| wr\_data | I | 32 | Write data |
| rd\_stb | I | 1 | Read strobe : indicates the start of a read transaction. |

Table : RBUS Ring Input Interface

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **I/O** | **Width** | **Description** |
| ack | O | 1 | Acknowledge indicates the end of a transaction (read or write). |
| rd\_data | O | 32 | Read data |
| err\_ack | O | 1 | Indicates that the slave aborted the transaction due to error. |

Table : RBUS Ring Output Interface

# References

[1] Project\_Zipline\_Top\_Micro\_Architecture\_Specification(OCP)

1. If CMD.comp\_ctl.algorithm is not one of the specified values, CMD.comp\_ctl.xp10\_prefix\_mode is forced to NONE (0x0). [↑](#footnote-ref-2)