ATLAS Phase-II Upgrade Project

ATLAS FELIX firmware Phase-II Upgrade: Firmware specifications

Abstract

This document describes the firmware specifications of the ATLAS FELIX Phase-II Upgrade Project [**Collaboration:2285584**].

FELIX Phase-II firmware specifications		
ATLAS Doc:	ABC-D-EF-1234	
EDMS Id:	1234567 v.1	
EDMS Url:	https://edms.cern.ch/document/1234567/1	
Version:	0.87	
Created:	January 12, 2021	
Last modified:	May 7, 2021	
Prepared by: The FELIX Team	**Checked by:** The FELIX Team	**Approved by:** The ATLAS review committee

[INTENTIONALLY BLANK PAGE]

0

TEMPORARY ORGANISATIONAL THINGS

This chapter is just for info and shall be removed at the end.

0.1 TEMPLATES

The following lists templates that the ATLASPhaseII document class brings along.

There are definitions (definition 0.1), requirements (requirement 0.1), recommendations (recommendation 0.1) and remarks (remark 0.1).

Definition 0.1: *Definition template*

Define the term definition first before you can define other terms.

Requirement 0.1: *Requirement template*

Requirements define behaviour or properties of something that has been defined.

Recommendation 0.1: *Recommendation template*

Recommendations give "should", "could" or "aught to" examples in addition to dedicated requirements.

Remark 0.1: *Remark template*

Remarks give additional precision in case e.g. a definition or recommendation is not directly clear or needs further explanation.

Revision History

Revision	Date	Author(s)	Description
0.1	2019-12-13	Frans Schreuder	Initial commit of the FELIX phase 2 firmware specifications (mostly just copied from Phase2_FW_Blockdesign and added Atlas Phase2 template)
0.2	2019-12-19	Frans Schreuder	Added some entities as a graphical symbol and waveforms for axi stream
0.3	2019-12-19	Frans Schreuder	Added skeleton for RD53b decoder
0.4	2019-12-20	Frans Schreuder	Added several blocks, entities and moved around some text
0.5	2019-12-20	Frans Schreuder	Added full mode decoder entity
0.6	2020-01-07	Frans Schreuder	Split RD53b decoder and Aurora decoder in separate subsections
0.7	2020-01-08	Frans Schreuder	Described the DecodingGearBox
0.8	2020-01-09	Frans Schreuder	Added block diagram of Decoding Egroups and Epaths for (lp)GBT 8b10b mode
0.9	2020-01-10	Frans Schreuder	Added block diagram for Pixel ToHost e-path
0.10	2020-01-10	Frans Schreuder	Added description of the 8b10b decoder
0.11	2020-01-10	Frans Schreuder	Added progress bars to the different sections (Thank you LASP people for the idea)
0.12	2020-01-13	Jacopo Pinzino	added some informations about endeavour blocks
0.13	2020-01-14	Frans Schreuder	Added some missing encoder (skeleton) tex files
0.14	2020-01-14	Frans Schreuder	Added TTC Emulator
0.15	2020-01-14	Jacopo Pinzino	improving the Endeavour Encoder subsection
0.16	2020-01-16	Frans Schreuder	Added decoding egroup resources
0.17	2020-01-16	Frans Schreuder	regenerated pdf
0.18	2020-01-21	Elena Zhivun	Added description of the LCB protocol
0.19	2020-01-23	Frans Schreuder	Added atlas template
0.20	2020-01-24	Frans Schreuder	Changed information about CRC polynomial in FullMode.pdf
0.21	2020-01-27	Marius Wensing	starting work on the RD53B Decoder
0.22	2020-01-28	Frans Schreuder	Replaced verbatim with lstlisting in LCBEncoder.tex, it gave typesetting errors
0.23	2020-02-04	Marius Wensing	updating RD53B decoder entity and re-generated PDF
0.24	2020-02-04	Marius Wensing	adding entity for FromHost Central Router
0.25	2020-02-04	Jacopo Pinzino	add table
0.26	2020-02-10	Marius Wensing	more work on the RD53B decoder section
0.27	2020-02-10	Jacopo Pinzino	improvement endeavour encoder decoder part
0.28	2020-02-11	Frans Schreuder	Regenerated wupper documentation with rm4.9
0.29	2020-02-17	Marius Wensing	adding example waveform for CRFromHost input
0.30	2020-02-17	Marius Wensing	starting to document the FromHost Central Router
0.31	2020-02-18	Marius Wensing	adding resource usage for RD53B decoder
0.32	2020-02-20	Frans Schreuder	Fixed some issues in the EndeavourDecoder/Encoder documents (figures not found etc) Unified tables throughout the document
0.33	2020-02-20	Jacopo Pinzino	improvement endeavour encoder decoder part
0.34	2020-02-20	Frans Schreuder	regenerated pdf
0.35	2020-02-21	Frans Schreuder	Some minor updates to Endeavour Decoder / Encoder
0.36	2020-03-17	Frans Schreuder	Added section about TTC Encoder
0.37	2020-05-07	jacopo pinzino	added EndeavourDeglitcher in the Endeavour Decoder subsection of the Phase2_FM_specs

0.38	2020-05-07	jacopo pinzino	correct typo in Endeavour Decoder subsection of the Phase2_FM_specs
0.39	2020-05-12	jacopo pinzino	small grammatical corrections
0.40	2020-06-05	Elena Zhivun	Started on updating ITk Strips documentation
0.41	2020-06-05	Elena Zhivun	Started on LCB module documentation
0.42	2020-06-07	Elena Zhivun	Editing the documentation
0.43	2020-06-08	Elena Zhivun	Fixed bit ordering
0.44	2020-06-08	Elena Zhivun	Editing the text
0.45	2020-06-08	Elena Zhivun	Edit documentation
0.46	2020-06-09	Elena Zhivun	Added examples
0.47	2020-06-10	Elena Zhivun	Updated Strips protocol description
0.48	2020-06-16	Frans Schreuder	Built PDF
0.49	2020-06-24	Elena Zhivun	Added remark about BC gating interval
0.50	2020-06-25	Elena Zhivun	Updated remark about BC gating generation
0.51	2020-07-23	Nico Giangiacomi	Modified TTC Encoder table, removed useless TTCOptions
0.52	2020-11-19	Elena Zhivun	Update Strips module documentation
0.53	2021-01-12	Frans Schreuder	Added chapter about testing
0.54	2021-01-12	Frans Schreuder	Added related documents
0.55	2021-01-15	Frans Schreuder	Added section about FULL mode, added detailed toplevel schematic including all toplevel signals
0.56	2021-01-19	Frans Schreuder	Added register-map 5.0 as appendix
0.57	2021-01-21	Frans Schreuder	Added documentation for: * Wupper * Firmware flavours * Minor other modifications
0.58	2021-01-21	Frans Schreuder	Minor modifications in felix toplevel (detailed) drawing
0.59	2021-01-21	Marius Wensing	working on CRFromHost
0.60	2021-01-22	Frans Schreuder	Started section about CRToHost
0.61	2021-01-26	Kai Chen	some texts are added in section 4/6/8, to be continued
0.62	2021-01-27	Marius Wensing	more work on CRFromHost chapter
0.63	2021-01-27	Frans Schreuder	Finished section about CRToHost, added resources for CRFromHost
0.64	2021-01-27	Frans Schreuder	Removed FELIX_Phase2_firmware_specs generated PDF, and instead generated it using Gitlab CI. Need to find a way to publish it somewhere.
0.65	2021-01-27	Frans Schreuder	Fixed capitalization of extension png=>PNG of file name
0.66	2021-01-28	Frans Schreuder	Added makefile for Wupper
0.67	2021-01-28	Frans Schreuder	Worked on Data Formats
0.68	2021-02-02	Kai Chen	add material for GBT/lpGBT in sec 8.6, and sec. 4
0.69	2021-02-02	Frans Schreuder	Updated front page and added glossaries
0.70	2021-02-02	Elena Zhivun	Add resource utilization for Strips links
0.71	2021-02-02	Elena Zhivun	Update the Strips documentation
0.72	2021-02-02	Elena Zhivun	Fix tables
0.73	2021-02-08	Nico Giangiacomi	Added 8b10bEncoder
0.74	2021-02-09	Kai Chen	Changes to the Section 8.6
0.75	2021-02-09	Kai Chen	Changes to the Section 6
0.76	2021-02-09	Kai Chen	Changes to the Section 8.6
0.77	2021-02-15	Frans Schreuder	Some work on Global Description
0.78	2021-02-16	Frans Schreuder	Added a chapter about AXI stream IDs per firmware flavour
0.79	2021-02-18	Frans Schreuder	Added description of HDLC Decoder
0.80	2021-02-18	Kai Chen	add fansink information for FLX-712
0.81	2021-02-19	Frans Schreuder	Described HDLC Encoder

0.82	2021-03-04	Frans Schreuder	Added description of the BUSY ToHost Virtual E-Link
0.83	2021-03-04	Frans Schreuder	Modified makefile to generate History.tex from git log
0.84	2021-03-04	Frans Schreuder	Reverted template/Makefile, now generate History.tex from MakeHistory.sh
0.85	2021-03-04	Frans Schreuder	Automatic version history from GIT with 0.xx numbering, alsu automate the version of the document this way
0.86	2021-03-05	Frans Schreuder	Separated TTC (Legacy) and BUSY sections, added LTI/TTC interface (empty placeholder)
0.87	2021-03-05	Frans Schreuder	Added section about the TTC ToHost Virtual E-Link

Table of Contents

1

Conventions and Glossary

Glossary

ATLAS A Toroidal LHC Apparatus. i

AXI Advanced eXtensible Interface, widely used on Xilinx IP. AXI4-Stream is widely used in the FELIX project first. 24

BC Bunch Crossing, The CERN LHC bunch crossing clock frequency is 40.07897 MHz first. 71

Block Fixed section of memory with a specific formatting, headers and trailers first. B.29

BUSY A condition that can be raised from the FELIX system towards the central trigger processor in case buffers fill up and data aquisition must be halted first. 102, 104

DMA Direct Memory Access first. 113

FELIX Front End LInk eXchange. i

FIFO First In First Out, a type of memory to store data, also used to cross clock domains first. 23, 65

FLX128 Xilinx VCU128 / VU37P Development kit with FELIX firmware. 13

FLX712 FELIX Phase I PCIe card (BNL712) with FELIX firmware. 13

FromHost Direction of data communication, in ATLAS also referred to as Downlink. Data flows from the Host PC towards the FPGA first. 110

GBT VersatileLink GigaBitTransceiver, a protocol and chip (GBTx) with 4.8Gb/s communication and logical links (E-Links) first. 21

lpGBT low power GigaBitTransceiver, a successor of GBT with 9.6Gb/s Uplink, 2.56Gb/s Downlink and logical links (E-Links) first. 21

ToHost Direction of data communication, in ATLAS also referred to as Uplink. Data flows from the FPGA towards the Host PC first. 105

TTC Timing, Trigger and Control, a protocol to distribute timing and trigger information first. 102

Wupper An implementation of a PCIe DMA controller for Xilinx FPGAs first. 113

2

RELATED DOCUMENTS

Remark 2.1: *Instructions for this chapter*

List all documents (other than references) linked to this specification. Include in this list the specification of the actual hardware where the firmware is supposed to be deployed as well as all specifications related to interfaces and standards that the firmware refers to.

Most of the important documents can be found on the Atlas FELIX project website, under User documentation:

- ATLAS FELIX website
  https://atlas-project-felix.web.cern.ch
- FELIX user documentation
  https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/documentation.html

Three important documents are especially worth mentioning in this section:

- FELIX User Manual
  https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/felix-user-manual/versions/Latest/
- BNL712 User manual
  https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/docs/BNL-711_V2P0_manual.pdf
- FELIX software documentation
  https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/felixdoc/

3

Global Description and Specification

Remark 3.1: *Instructions for this chapter*

Give a clear description of the overall firmware and its main functions, providing the general context and where it will be used. (Re-)Introduce global design specifications as imposed by outer constraints (latencies, frequencies) and terms.

While the FELIX firmware for ATLAS Phase-II upgrade will inherit most of the functionalities from the Phase-I firmware, the architecture or structure needs a throughout re-think and re-design. The aim is to improve the generality of the core of the firmware while making it more flexible for developer to incorporate different modules for specific detectors.

This document details a preliminary design of the FELIX firmware for ATLAS Phase-II upgrade. It starts from the top level firmware structure before going into details for each module. The interfaces between the modules are specified.

The aim is to kick-start the discussion among the developers. Comments and ideas are most welcome. As development progresses, the diagrams will be updated continuously, more details will be added.

3.1 Firmware Flavours

The FELIX Phase II firmware is kept as generic as possible. All the firmware flavours that fall within the scope of this document - the officially supported flavours - are built from a single toplevel VHDL file called felix_top.vhd. The firmware flavour is determined at build time by means of a generic: "FIRMWARE_MODE". Based on this generic, the appropriate Link Wrapper will be instantiated, and a set of encoders and decoders is selected.

Flavour	Link Wrapper	Decoders	Encoders	Remarks
0: GBT	GBT	8b10b 8.3.14 HDLC 8.3.15 Direct 8.3.17 TTCToHost 8.3.18 BusyToHost ??	8b10b 8.4.14 HDLC 8.4.15 Direct 8.4.16 TTC 8.4.17	The GBT mode flavour is available in 8 and 24 channel versions, with a complete set of encoders / decoders, and a so called SemiStatic configuration where some decoders/encoders are left out. FELIX aims to provide a 24 channel fully configurable version for FLX712, it has been demonstrated to work but with high resource count (78% LUTs)

1: FULL	ToHost FULL, FromHost GBT	FULL 8.3.16 TTCToHost 8.3.18 BusyToHost **??**	8b10b 8.4.14 HDLC 8.4.15 Direct 8.4.16 TTC 8.4.17	The FULL mode flavour is available in 24 channels for FLX712 and FLX128. The ToHost side/decoding is using 9.6Gb/s 8b10b data without logical links. FromHost/encoding is identical to GBT
2: LTDB	GBT	8b10b 8.3.14 HDLC 8.3.15 Direct 8.3.17 TTCToHost 8.3.18 BusyToHost **??**	8b10b 8.4.14 HDLC 8.4.15 Direct 8.4.16 TTC 8.4.17	LTDB mode is a 48 channel version of GBT mode, but with reduced e-link configurability. This flavour only includes the EC and IC e-links, as well as an AUX e-link (Egroup 4, link 7) with HDLC/8b10b/Direct configuration. Additionally TTC distribution is available on all FromHost/ToFrontend e-links.
4: PIXEL	lpGBT	HDLC (EC/IC) 8.3.15 Aurora 8.3.12 TTCToHost 8.3.18 BusyToHost **??**	RD53A/B 8.4.11 TTC 8.4.17 HDLC (IC/EC) 8.4.15	The Pixel flavour was designed to read out the ITk Pixel detector over lpGBT with Aurora e-links. The encoder uses a custom protocol for RD53 and includes a trigger and command state machine.
5: STRIP	lpGBT	HDLC (IC) 8.3.15 Endeavour (EC) 8.3.11 8b10b 8.3.14, 8.3.10 TTCToHost 8.3.18 BusyToHost **??**	HDLC (EC) 8.4.15 Endeavour (EC) 8.4.10 LCB 8.4.12 R3L1 8.4.13	The Strip flavour was designed to read out the ITk Strip detector over lpGBT with 8b10b e-links. The encoder uses a strip custom protocol with so called trickle merge.
9: LPGBT	lpGBT	HDLC (EC/IC) 8.3.15 8b10b 8.3.14 Direct 8.3.17 TTCToHost 8.3.18 BusyToHost **??**	8b10b 8.4.14 HDLC 8.4.15 Direct 8.4.16 TTC 8.4.17	The lpGBT Flavour is the lpGBT equivalent of the GBT flavour. It involves 8b10b, HDLC and TTC protocols and the aim is to have a fully configurable 24 channel build available.
10: INTERLAKEN	64b67b	Interlaken 8.3.20	Interlaken 8.4.18	The Interlaken Flavour has not yet been specified, the aim is to have at least 8 bidirectional 25Gb/s Interlaken links. Combinations with other protocols (lpGBT/GBT) may be required by subdetectors.

Table 3.1: Firmware Flavours and their configurations.

The following firmware flavours fall outside the scope of this document, and are documented elsewhere.

- 3: FEI4, For internal development only, not an official release.
- 6: FELIG, GBT Front End emulator [1].
- 7: FMEMU, FULL Mode Front End Emulator [2].
- 8: MROD, FELIX_MROD is a special flavour that was developed in case the MRODs fail. [3]

3.1.1 E-Path IDs/ AXIs IDs

At build time, the firmware flavour is defined, and depending on this flavour every physical link is equipped with a number of logical links (E-Links). Every individual encoder or decoder is associated with an AXIs ID, which is used to address the correct encoder / decoder. Addressing is done my means of the header in the FromHost data format (see B.4), and the block header in the ToHost data format (see B.2)

Flavour	ToHost AXIs IDs	FromHost AXIs IDs	Remarks
0: GBT	0-39: 8b10b, HDLC, Direct 40: EC: 8b10b, HDLC, Direct 41: IC: HDLC	0-39: 8b10b, HDLC, Direct, TTC 40: EC: 8b10b, HDLC, Direct 41: IC: HDLC	A semistatic configuration may have a subset of this configuration
1: FULL	0: FULL	0-39: 8b10b, HDLC, Direct, TTC 40: EC: 8b10b, HDLC, Direct 41: IC: HDLC	
2: LTDB	39: AUX: 8b10b, HDLC, Direct 40: EC: 8b10b, HDLC, Direct 41: IC: HDLC	0-38: TTC 39: AUX 8b10b, HDLC, Direct, TTC 40: EC: 8b10b, HDLC, Direct 41: IC: HDLC	

4: PIXEL	0,4,8,12,16,20,24: Aurora 28: EC: 8b10b, HDLC, Direct 29: IC: HDLC	0-15: RD53 16: EC: 8b10b, HDLC, Direct 17: IC: HDLC	1 E-Path per ToHost E-group, 3 AXIs IDs per ToHost E-group are unused.
5: STRIP	0-27: 8b10b 28: EC: Endeavour 29: IC: HDLC	0,5,10,15: lcb config 1,6,11,16: lcb command 2,7,12,17: lcb trickle 3,8,13,18: r3l1 config 4,9,14,19: r3l1 command 20: EC Endeavour 21: IC HDLC	Strip FromHost AXIs IDs are not associated with the E-Link number on the lpGBT frame, but have a dedicated numbering scheme, see also 8.4.12 and 8.4.13
9: LPGBT	0-27: 8b10b 28: EC: 8b10b, HDLC, Direct 29: IC: HDLC	0-15: 8b10b, HDLC, Direct, TTC 16: EC: 8b10b, HDLC, Direct 17: IC: HDLC	
10: INTERLAKEN	0: Interlaken	0:Interlaken	No logical links on top of Interlaken

Table 3.2: E-Link configurations and AXIs IDs for the Firmware Flavours.

Table 3.2 shows the available AXIs IDs which are mapped on the physical links. Every link and its associated E-Links/AXIs IDs are replicated by the number of physical optical links in the build, so the encoder / decoder is not only addressed by the AXIs ID, but also by the GBT ID, which is the number of the physical link starting at 0 from every endpoint. For a typical 24 channel firmware flavour, every PCIe endpoint is associated with 12 GBT IDs (0-11).

In ToHost direction, there is one extra GBT ID for the virtual E-links, associated with axis_aux, the auxiliary AXI Streams. These streams contain the TTCToHost and BUSYToHost virtual E-links.

- TTCToHost: GBT ID: GBT_NUM (12), AXIs ID: 0
- BUSYToHost: GBT ID: GBT_NUM (12), AXIs ID: 1

3.2 Top Level

The design strategy is to keep the top level architecture as general as possible. At all time, the dependencies among the module should be kept as minimal as possible to maintain the amount of change at a minimum when a small change is needed in a feature. Modules with similar functionality shall be grouped together to encourage code reuse.

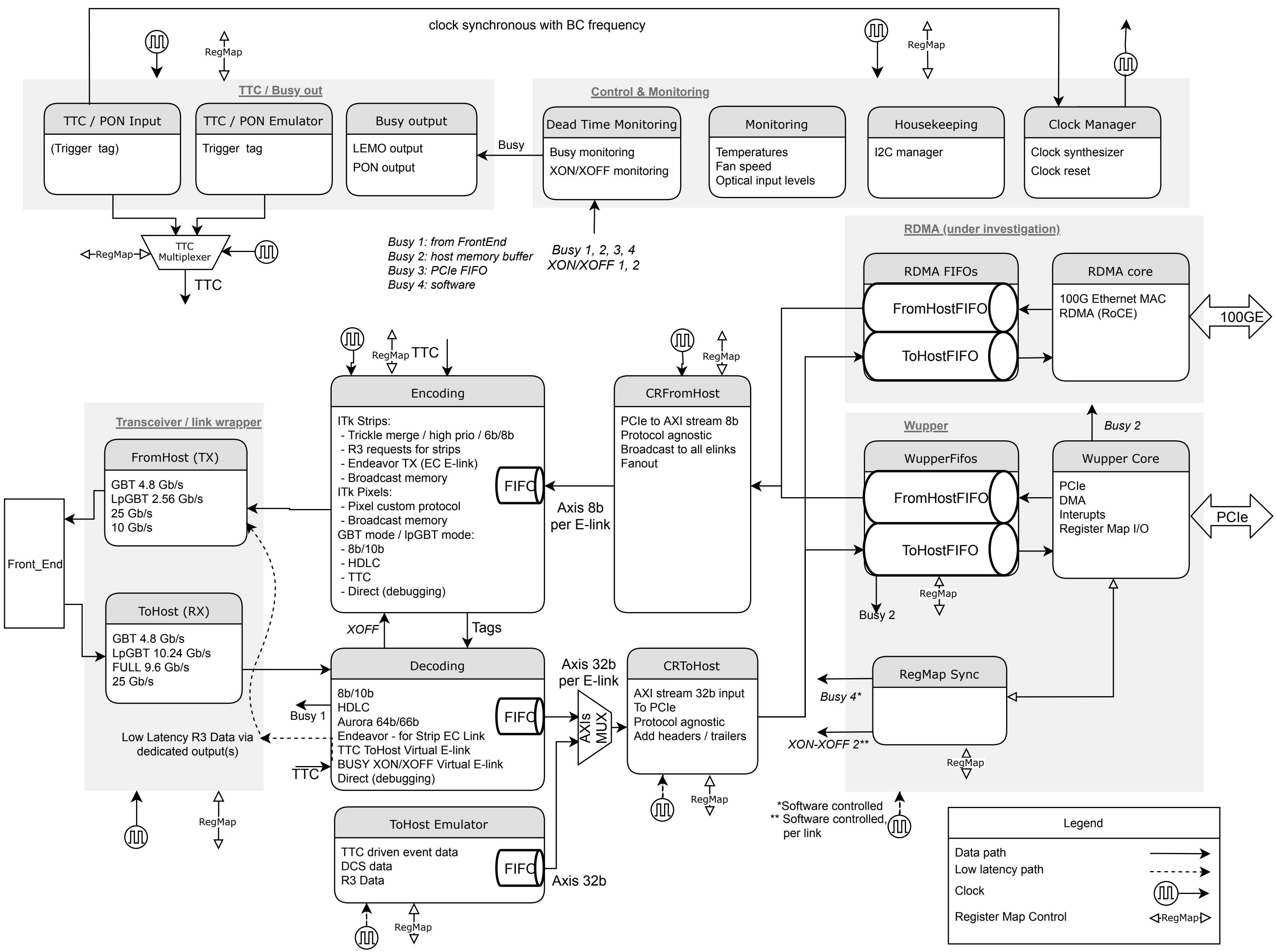


Figure 3.1: The FELIX firmware top level block diagrams..

3.2.1 Transceiver and link wrapper

- Interfaces to electrical to optical and optical to electrical transceivers
- TX: GBT or lpGBT scrambling: input GBT or lpGBT E-Link frames @ BC frequency, 25Gb/s links (Aurora or Interlaken), 10Gb/s links
- RX
  - For GBT or lpGBT descrambling, FEC handling, output GBT or lpGBT E-Link frames @ BC frequency to E-link Decoder
  - For FULL mode output of 8B/10B decoded data stream, with a CharisK indicator. The FULL mode Decoder block will further decode this data stream into 32b AXI stream.
  - for 25Gb/s Interlaken links 64b67b frames will be delivered to the Encoding / Decoding blocks.

A more detailed description of the Transceiver and Link wrapper is given in 8.5

3.2.2 Encoding

- Inputs: Encoding connects to CRFromHost by means of a 2D array of 8-bit AXI4-Stream, the size of this array is the number optical links by the number of logical links (E-Links) on top of every optical link. Data for the 25G Interlaken links will be delivered on a 64-bit AXI4-Stream.
- Outputs: GBT or lpGBT E-link frames @ BC frequency, TTC virtual E-link, 64b67b encoded data for 25G Interlaken links. Depending on the firmware flavour (See section 3.1) a set of the following encoders may be included in some or all E-links:
  - HDLC coding for IC E-link and configurable per E-link for other E-Links
  - Endeavour for the EC e-link of the strip flavour
  - 8B/10B coding for XON-XOFF messages configurable per E-link
  - 6B/8B coding of merged FromHost data and TTC signals (accepts, resets) for strips configurable per E-link
  - Pixel custom coding of merged FromHost data and of TTC signals (accepts, resets) configurable per E-link
  - TTC signals @ BC frequency configurable per E-link
  - Interlaken for 25 Gb/s links
- Broadcast Memory
  - In combination with the TTC emulators, generates a fixed pattern and send them to front ends chips at a programmable frequency in order to act as trigger loops.
  - Broadcast memory will be used in combination with trickle merge, see figure 8.10

A more detailed description of the Encoding block is given in 8.4

3.2.3 Decoding

- Inputs:
  - GBT frames, using E-Links. These E-links can carry multiple protocols such as 8b/10b, HDLC or direct (no encoding) mode.
  - lpGBT frames, using E-Links. These E-Links can carry multiple protocols such as 8b/10b, direct (no encoding), Aurora streams or Endeavour.
  - FULL mode. Links are 8b/10b encoded at 9.6Gb/s and chunks are delimited with special K-characters defined in 8.3.16

    - 25Gb/s links Interlaken links, the raw (scrambled) 67b data is decoded in a submodule of the decoding block. 8.3.20

  GBT or lpGBT E-Link frames @ BC frequency or 8B/10B or Aurora streams via E-links @ rate synchronous with BC frequency

- Outputs: data fragments, to be forwarded via the ToHost Multplexer to the ToHost Router with associated flags signaling start and end of fragments and error conditions, and an output or outputs for BUSY-ON and BUSY-OFF. The data fragments are also called Chunks, these chunks consist of any number of bytes and will later be packed in blocks by the ToHost Central Router (CRToHost) 8.11. The output format of the Decoding block, and all it's internal components is a 2D array of AXI-Stream 32b, the size of the AXI-Stream 32b array is the number of optical links by the number of logical links (E-Links) on top of the optical link. Data from 25G Interlaken links will be carried by 64-bit AXI4-Streams.

- Every E-link on a GBT or lpGBT is encoded depending on the specification of the subdetector / frontend. A firmware flavour (See section 3.1) may have a subset of on or more of the following options to decode the E-links:

    - 8B/10B decoding for E-links transferring 8B/10B coded data. Strip data streams contain event and register data, splitting in software in the host PC. Extraction of BUSY-ON and BUSY-OFF control symbols and forwarding to the Busy output of this block is done by the 8b10b decoder as well.
    - HDLC decoding of the IC E-link data and configurable for other E-links
    - Aurora decoding, single data stream via either 1, 2 or 4 lanes (1 lane per E-link), this single data stream needs to be reconstructed, in the case of 4 lanes two lanes may be associated with two E-links from another physical link than the two other lanes, mapping of lanes on E-links need to be configurable. Register data and event data in same data stream. See also 8.3.12.
    - Endevour decoding is included for the EC E-Link of the strips flavour.
    - FULL Mode: 9.6Gb/s 8b10b encoded links can be decoded. FULL mode does not include E-Links but the encoding happens directly on top of the physical link.
    - Interlaken: The 25G Interlaken decoder will be included as a submodule of the decoding block.

A more detailed description of the Decoding block is given in 8.3

3.2.4 AXIs MUX (ToHost Fanout Selector)

- Forwards data with associated flags signalling start and end of fragments and of error conditions, either from E-Link Decoder, for FULL mode from Link Wrapper RX or from ToHost Emulator to ToHost Router
- Control with configuration register

3.2.5 CRFromHost: CentralRouter in FromHost direction

- Inputs and buffers data packets that contain information on E-link and packet length in data streams from FromHost FIFOs. Packets are buffered, complete packets are output to FromHost Multiplexer

3.2.6 CRToHost: CentralRouter in ToHost direction

- Inputs fragments with associated flags signalling start and end and error conditions
- Inputs fragments from virtual TTC E-Link and from virtual E-Link for BUSY XON/XOFF monitoring (if implemented)
- Forms blocks with headers and filled with chunks or subchunks with appropriate trailers on the basis of the data and the flags received
- Outputs blocks to the FIFO of the ToHost FIFOs with which the block is associated.

- The number of output FIFOs is determined by the number of parallel ToHost DMA paths supported by Wupper (see 8.13).

3.2.7 ToHost Emulator

- Forward either event data, DCS or R3 data to ToHost Switch
- For each E-link there is a separate data stream
- Event data have an embedded L1ID, which is incremented for each fragment
- Event data can be generated on the basis of L0 or of L1 accepts, as received via TTC P2P, or as generated by the TTC emulator
- Random fragment sizes on the basis of arbitrary probability distribution.
- R3 Data can be generated on the basis of L0 accepts as received via TTC P2P, or as generated by the TTC emulator

3.2.8 Wupper

- FromHost FIFOs
  - One FIFO in FromHost direction
- ToHost FIFOs
  - One FIFO per ToHost DMA channel
  - Generates Busy if FIFO(s) becomes too FULL
  - Generates Busy if server PC memory becomes too FULL
- Register map
  - All registers with updates synchronised with BC clock
  - Can generate Busy under software control
  - Can generate XON or XOFF for individual links under software control
- Wupper Core
  - DMA engine
  - PCIe interfacing
  - Interrupt generation and control
  - Register map I/O
  - Generates Busy if output circular buffer(s) in host memory are too full

Control and Monitoring

- Dead Time Monitoring
  - Input of all Busy signals
  - Input of all Xon/XOFF statuses
  - Status available in registers
  - Output of Busy to Busy Output, configurable which Busy inputs contribute
  - Optional virtual E-link output of time stamped messages indicating Busy-On, Busy-Off, XON or XOFF and the E-link or link associated with the condition if relevant

- Monitoring
  - Temperatures
  - Fan Speed
  - Optical input levels
  - Voltages
  - ...
- Housekeeping: i2c control
- Clock Manager
  - Receives clock synchronous with BC frequency from TTC or PON, if present, and jitter cleaning of this clock
  - Can also generate clock without presence of TTC or PON

TTC / Busy out

- TTC P2P Input
  - Input of TTC data patterns from the original TTC system or TTC P2P.
  - Output to E-Link Encoder and ToHost Emulator via TTC Multiplexer
- TTC / TTC P2P Emulator
  - Generation of TTC data patterns and trigger tags as received either from the original TTC system or via PON
  - Output to E-Link Encoder and ToHost Emulator via TTC Multiplexer
- Busy output: receives Busy signal from Dead Time Moritoring and outputs this via LEMO output or via PON

3.2.9 Number instances per FPGA

- TTC / Busy out: 1
- Control and Monitoring: 1
- Link Protocol Wrapper: 1
- CRToHost: 1 per Wupper
- CRFromHost: 1 per Wupper
- Encoding: 1 per Wupper
- Decoding: 1 per Wupper
- Wupper: typically 2, each servicing an 8 lane PCIe interface
- RDMA: typically 2, each servicing one 100Gb Ethernet link

4

External Interfaces (I/O)

This section describes the hardware interfaces (I/O) provided by the cards. The FLX-712 card provides up to 24 or 48 bi-directional optical links via the 12-channel 14-Gbps MiniPOD modules. The supported protocols are listed in Section 4.1. The detailed format are described in Section 8.5. The timing mezzanine card can provide interfaces for TTC and BUSY, which can be connected to the existing ATLAS TTC system. With different assembly, it can also be configured to with an SFP module [4], to interface different types of timing system, for instance TTC-PON or White Rabbit.

For Phase-II card, the optical module will be the Samtec FireFly module, which can support a link speed of up to 28 Gbps. More details will be shown in the hardware documents. The VCU128 used for firmware demonstration provides 4× QSFP28 module, with in total 16× 28 Gbps links, which can be used to verify the proposed 25 Gbps Interlaken and also 100 Gbps Ethernet connection.

4.1 FrontEnd links

The protocols supported by FELIX firmware are listed in Table 4.1. For different protocol, FELIX firmware will configure the on-board jitter cleaner to output clocks with low phase noise for Xilinx transceivers.

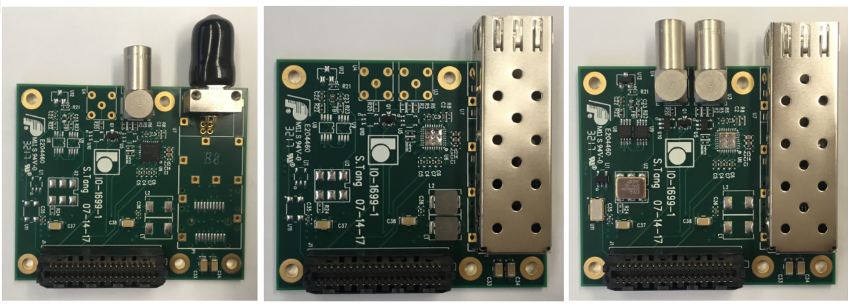


Figure 4.1: The timing mezzanine for FLX-712, with different configuration.

Protocol	FELIX	Front-end
GBT	TX: 4.8G, RX: 4.8G	TX: 4.8G, RX: 4.8G
FULL	TX: 4.8G, RX: 9.6G	TX: 9.6G, RX: 4.8G
lpGBT	TX: 2.56.8G, RX: 10.24G	TX: 10.24G, RX: 2.56G
25G link	TBD	TBD

Table 4.1: Protocols supported by FELIX.

4.2 PCIe

The FELIX Phase II firmware will interface with the FELIX server through a PCIe Gen4x16 interface. This interface will consist of 2 Gen4x8 interfaces in the FELIX FPGA, combined with a PCIe Gen4 bridge on the FELIX card.

During the development phase, FELIX is also built for the Phase I hardware platform - FLX712, which has a PCIe Gen3x16 interface. The firmware will support both Gen3 and Gen4 PCIe interfaces, depending on the hardware platform the link speed will be chosen.

4.3 TTC Interface

The left photo in Figure 4.1 shows the timing mezzanine on FLX-712 card, with the TTC optical receiver and CDR ASIC on it. For Phase-2, the TTC interface with LTI is still under discussion.

4.4 BUSY

The Phase I hardware platform has a dedicated LEMO-00 output (Open drain / pull down) to report BUSY, shown in Figure 4.1. In Phase II the BUSY condition will be communicated to the LTI over the TTC-P2P link.

4.5 100Gb/s Ethernet

The hardware platform that is used to evaluate Phase II link speeds up to 25 Gb/s, will also be equipped with one or more 100 Gb/s links. The FLX128 (Xilinx VCU128) is equipped with 4 QSFP28 transceivers for this purpose. The 100Gb/s Ethernet interface can be used for the RDMA link (see section 8.14) which is currently under investigation as a possible alternative / addition to PCIe DMA.

5

TARGET FPGA

Remark 5.1: *Instructions for this chapter*

Indicate the target FPGA (vendor, part number with speed grade) and reference the data sheet. List the main features (number of pins, fabric speed).
Table 5.1 is part of the instructions.
Extend the table accordingly (e.g. embedded processor, etc.).

The FELIX Phase I prototype card, also called FLX712 FELIX Phase I PCIe card (BNL712) with FELIX firmwares has been used for developing the FELIX firmware for Phase I. Most of the components if the FELIX Phase II firmware will be based on their Phase I counterparts, even though some interfaces will change and some components have to be redesigned while others will have to be created from the start.

The primary development platform for the FELIX Phase II firmware PDR will the FLX712, and all the features that can be demonstrated on that platform will be implemented.

There are some features that are planned for the Phase II upgrade that can not be demonstrated with the FLX712 hardware. These features are:

- PCIe Gen4
- 25 Gb/s links
- 100Gb/s RDMA

In order to demonstrate these features, a second development platform will be used, the Xilinx VCU128, incorporating a XCVU37P-L2FSVH2892EES9837 FPGA. The VCU128 may also be referred to as Xilinx VCU128 / VU37P Development kit with FELIX firmware, meaning a VCU128 running FELIX firmware.

Resource	Availability (FLX712)	Availability (VCU128)	Estimated usage
Logic elements	1451	2852K	0 %
Block ram	75.9 Mb	70.9 Mb	0 %
UltraRAM	-	270 Mb	0 %
HBM DRAM	-	8 GB	0 %
Transceivers (< 25 Gbps)	64	-	0 %
Transceivers (> 25 Gbps)	-	96	81 %

Table 5.1: Estimated resource usage of the FELIX Phase II firmware.

	Phase-I	Phase-II
FPGA	XCKU115	XCVU9P
CLB LUTs	663,360	1,182,000
CLB Flip-Flops	1,326,720	2,364,000
Block RAM [Mb]	75.9	75.9
UltraRAM [Mb]	-	270

Table 5.2: Available FPGA resources [5, 6]..

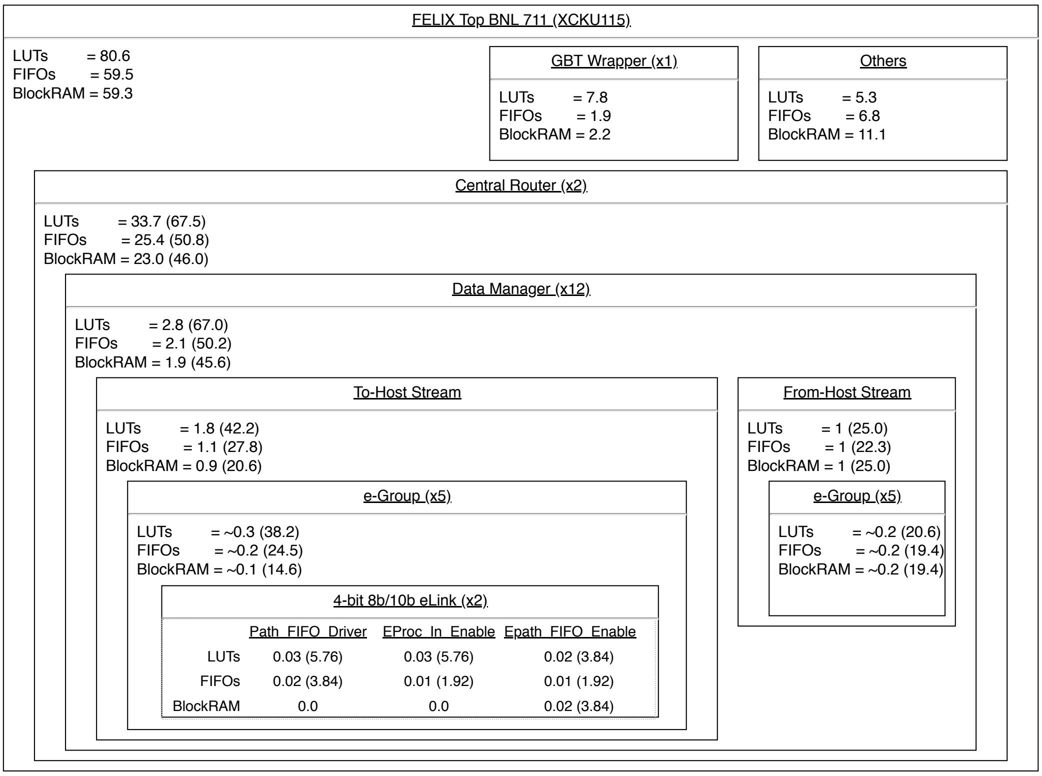


Figure 5.1: The Phase-I FELIX resource utilization in percentage (rounded up) for XCKU115 FPGA. Only the components which consume the most resources are shown. The numbers in each block are part of the numbers in their immediate outer block. For example, of the 81% LUTs utilization, each central router contributes about 34%. The rest of the 13% (not shown) come from other sources. The numbers in parenthesis are the total value. .

6

Power and Cooling

Remark 6.1: *Instructions for this chapter*

List all the voltage and current sources supplied to the FPGA. For each power rail, list the maximum current budgeted and estimated current draw with present firmware design, usually provided by the power estimate tool from FPGA vendor. Cooling scheme applied to FPGA and board which is used to derive the power budget could be described. Some items like current draw may not be known at the time of the first specification review but a good estimate should be included by the time of PDR & FDR with final numbers for PRR.
This section is usually provided by the hardware design team, which will provide important boundary condition for the firmware development. An early interaction between firmware and hardware design teams is crucial.
Table 6.4 is part of the instructions.

For the FLX-712, the power rails are listed in Table 6.2. The maximum current of each power rail and the esitimated (measured) current with Phase-I firmware are also contained in the Table.

(to do: Verify Maximum current for Phase-I cases can be measured, for phase-2, estimation can be got by Xilinx XPE, however this section really relies on the hardware, not sure whether this should be put in firmware, especially for Phase 2.)

Name of Voltage Rails	Max I budgeted for Voltage Rail	Estimated I budgeted for Voltage Rail
SYS5: 5V	18A	very low, only for the TTC receiver module
VCCINT: 0.95V	18A	typical: 6A
PEX0P9V: 0.9V	18A	typical: 6A
MGTAVCC: 1.0V	18A	typical: 8A
MGTAVTT: 1.2V	18A	<4.5A
SYS18: 1.8V	18A	<1.5A
SYS25: 2.5V	18A	<1.5A
SYS33: 3.3V	18A	typical: 3.5A for 48-channel card

Table 6.2: Power Requirements.

The fansink used on FLX-712 is 30-18828-04 shown in Figure 6.1. There are three pins for the fan, which are the 12V power, ground and the pin for tachometer. The fan speed is around 8500 RPM. The mean time to failure (MTTF) at 40 Celsius degree is about 36 years, while after 10 years, 10% of fans are estimated to

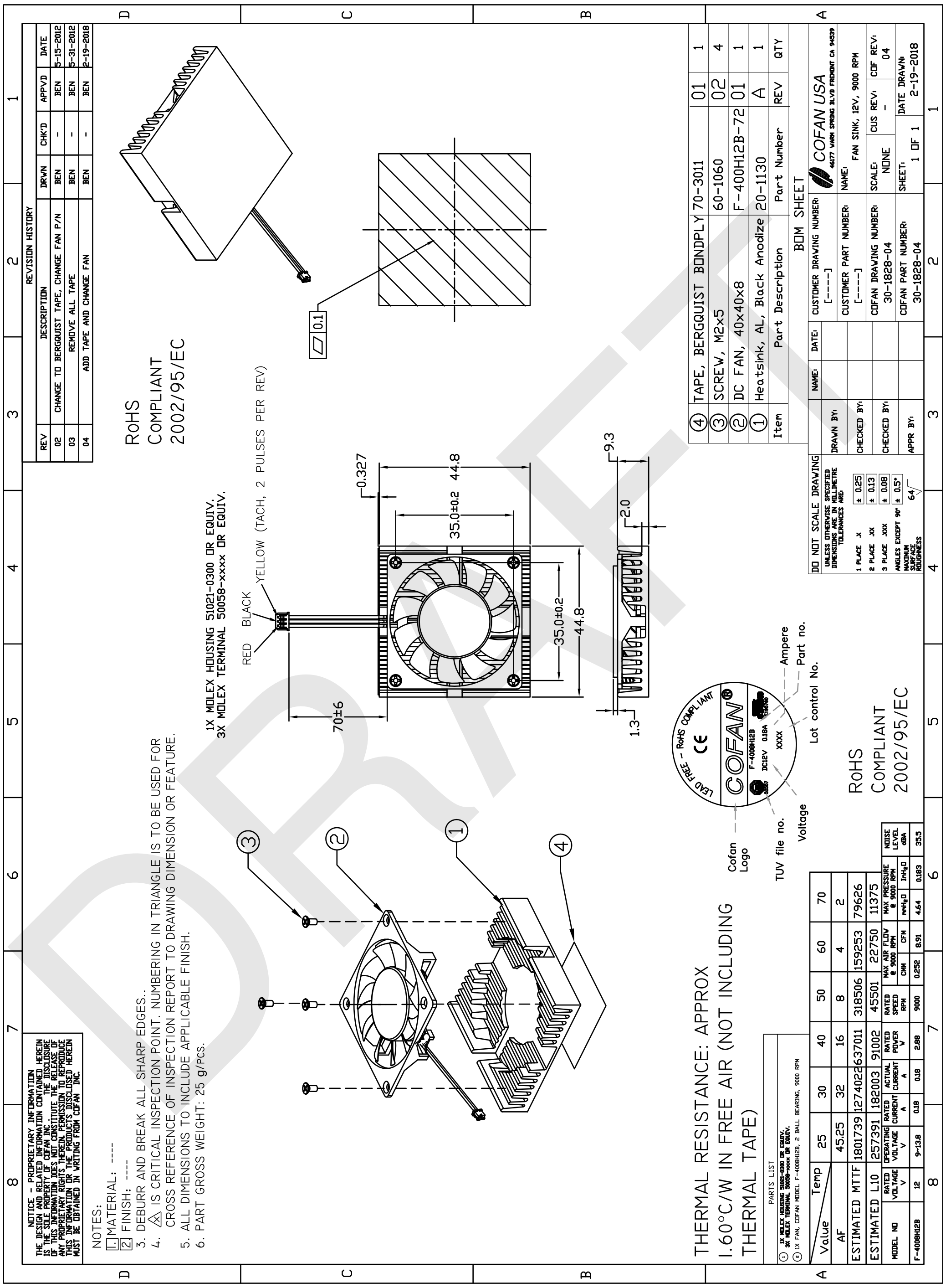

DRAFT
RoHS COMPLIANT 2002/95/EC
THERMAL RESISTANCE: APPROX 1.60°C/W IN FREE AIR (NOT INCLUDING THERMAL TAPE)
NOTES:
1 MATERIAL: ----
2 FINISH: ----
3. DEBURR AND BREAK ALL SHARP EDGES.
4. IS CRITICAL INSPECTION POINT. NUMBERING IN TRIANGLE IS TO BE USED FOR CROSS REFERENCE OF INSPECTION REPORT TO DRAWING DIMENSION OR FEATURE.
5. ALL DIMENSIONS TO INCLUDE APPLICABLE FINISH.
6. PART GROSS WEIGHT: 25 g/PCS.
1X MOLEX HOUSING 51021-0300 OR EQUIV.
3X MOLEX TERMINAL 50058-xxxx OR EQUIV.
RED BLACK YELLOW (TACH, 2 PULSES PER REV)
4 TAPE, BERGQUIST BONDPLY 70-3011 01 1
3 SCREW, M2x5 60-1060 02 4
2 DC FAN, 40x40x8 F-400H12B-72 01 1
1 Heatsink, AL, Black Anodize 20-1130 A 1
Item Part Description Part Number REV QTY
BOM SHEET
COFAN USA
COFAN DRAWING NUMBER: 30-1828-04
COFAN PART NUMBER: 30-1828-04
DATE DRAWN: 2-19-2018
DO NOT SCALE DRAWING
Cofan Logo
TUV file no.
Voltage
Lot control No.
Part no.
Ampere

Figure 6.1

malfunction. The heatsink is stuck on the FPGA. During the production, it was found for some heatsink, it has bad contact with the FPGA. For Phase 2 cards, the maxiGRIP fansink will be used. Phase change thermal interface material (TIM) will be used to attach the heatsink to the FPGA. Screws will be used to assemble the fansink.

Phase-2: to be added by Hongbin: fan selection, control and monitoring. And power estimation.

Name of Voltage Rails	Max I budgeted for Voltage Rail	Estimated I budgeted for Voltage Rail	Other Requirements
1	2	3	4

Table 6.4: Power Requirements.

7

INPUT/OUTPUT

Remark 7.1: *Instructions for this chapter*

The table should match what is planned to be implemented in the hardware design, between firmware port name and hardware pin number. It can be filled out once the hardware design is available by PDR. Eventually this will evolve into a constraint file for the firmware build by PDR/FDR/PRR. Table 7.1 is part of the instructions.

Name	Direction	Type	Description
BUSY_OUT	out	LVCMOS18	std_logic, Busy output (to LEMO) 1 = BUSY
CLK_TTC_N	in	LVDS	std_logic, 160 MHz clock from TTC
CLK_TTC_P	in	LVDS	std_logic, 160 MHz clock from TTC
DATA_TTC_N	in	LVDS	std_logic, Recovered data from TTC
DATA_TTC_P	in	LVDS	std_logic, Recovered data from TTC
I2C_SMB	out	LVCMOS18	std_logic, PEX I2C Enable
I2C_SMBUS_CFG_nEN	out	LVCMOS18	std_logic, PEX SMBus CFG Enable
I2C_nRESET_PCIe	out	LVCMOS18	std_logic, PEX active low reset
LOL_ADN	in	LVCMOS18	std_logic, ADN2814 LOL input
LOS_ADN	in	LVCMOS18	std_logic, ADN2814 LOS input
MGMT_PORT_EN	out	LVCMOS18	std_logic, PEX management port enable
NT_PORTSEL	out	LVCMOS18	std_logic_vector(2 downto 0), PEX port select
PCIE_PERSTn1	out	LVCMOS18	std_logic, PEX PERST
PCIE_PERSTn2	out	LVCMOS18	std_logic, PEX PERST
PEX_PERSTn	out	LVCMOS18	std_logic, PEX PERST
PEX_SCL	out	LVCMOS18	std_logic, PEX I2C
PEX_SDA	inout	LVCMOS18	std_logic, PEX I2C
PORT_GOOD	in	LVCMOS18	std_logic_vector(7 downto 0), PEX port good indicator
Perstn1_open	in	LVCMOS18	std_logic, input pin, leave open
Perstn2_open	in	LVCMOS18	std_logic, input pin, leave open
GTREFCLK_N_IN	in	LVDS	std_logic_vector(5 downto 0), Reference clocks for transceivers
GTREFCLK_P_IN	in	LVDS	std_logic_vector(5 downto 0), Reference clocks for transceivers
RX_N	in	LVDS	std_logic_vector(23 downto 0), To and from Minipods

Table 7.1: IO pins (continued...)

Name	Direction	Type	Description
RX_P	in	LVDS	std_logic_vector(23 downto 0), To and from Minipods
TX_N	out	LVDS	std_logic_vector(23 downto 0), To and from Minipods
TX_P	out	LVDS	std_logic_vector(23 downto 0), To and from Minipods
SCL	inout	LVCMOS18	std_logic, Global board I2C bus
SDA	inout	LVCMOS18	std_logic, Global board I2C bus
SHPC_INT	out	LVCMOS18	std_logic, output, tie to constant '1'
SI5345_A	out	LVCMOS18	std_logic_vector(1 downto 0), Si5345 jitter cleaner configuration
SI5345_INSEL	out	LVCMOS18	std_logic_vector(1 downto 0), Si5345 jitter cleaner configuration
SI5345_OE	out	LVCMOS18	std_logic, Si5345 jitter cleaner configuration
SI5345_RSTN	out	LVCMOS18	std_logic, Si5345 jitter cleaner configuration
SI5345_SEL	out	LVCMOS18	std_logic, Si5345 jitter cleaner configuration
SI5345_nLOL	in	LVCMOS18	std_logic, Si5345 jitter cleaner configuration
STN0_PORTCFG	out	LVCMOS18	std_logic_vector(1 downto 0), Constant output, tie to "0Z"
STN1_PORTCFG	out	LVCMOS18	std_logic_vector(1 downto 0), Constant output, tie to "01"
SmaOut_x3	out	LVCMOS18	std_logic, Optional debug output
SmaOut_x4	out	LVCMOS18	std_logic, Optional debug output
SmaOut_x5	out	LVCMOS18	std_logic, Optional debug output
SmaOut_x6	out	LVCMOS18	std_logic, Optional debug output
TACH	in	LVCMOS18	std_logic, Fan tachometer input
TESTMODE	out	LVCMOS18	std_logic_vector(2 downto 0), Constant output, tie to "000"
UPSTREAM_PORTSEL	out	LVCMOS18	std_logic_vector(2 downto 0), Constant output, tie to "000"
app_clk_in_n	in	LVDS	std_logic, 200 MHz board crystal
app_clk_in_p	in	LVDS	std_logic, 200 MHz board crystal
clk40_ttc_ref_out_n	out	LVDS	std_logic, BC clock Towards Si5345 CLKIN
clk40_ttc_ref_out_p	out	LVDS	std_logic, BC clock Towards Si5345 CLKIN
clk_ttcfx_ref1_in_n	in	LVDS	std_logic, 240.474 MHz from Si5345
clk_ttcfx_ref1_in_p	in	LVDS	std_logic, 240.474 MHz from Si5345
emcclk	in	LVCMOS18	std_logic, High speed JTAG clock
i2cmux_rst	out	LVCMOS18	std_logic, Reset I2C mux
leds	out	LVCMOS18	std_logic_vector(7 downto 0), Board GPIO leds
flash_SEL	out	LVCMOS18	std_logic, Boot flash pins
flash_a	out	LVCMOS18	std_logic_vector(24 downto 0), Boot flash pins
flash_a_msb	inout	LVCMOS18	std_logic_vector(1 downto 0), Boot flash pins
flash_adv	out	LVCMOS18	std_logic, Boot flash pins
flash_cclk	out	LVCMOS18	std_logic, Boot flash pins
flash_ce	out	LVCMOS18	std_logic, Boot flash pins
flash_d	inout	LVCMOS18	std_logic_vector(15 downto 0), Boot flash pins
flash_re	out	LVCMOS18	std_logic, Boot flash pins
flash_we	out	LVCMOS18	std_logic, Boot flash pins
opto_inhibit	out	LVCMOS18	std_logic_vector(OPTO_TRX-1 downto 0), Minipod / FireFly enable / reset

Table 7.1: IO pins (continued...)

Name	Direction	Type	Description
pcie_rxn	in	LVDS	std_logic_vector(15 downto 0), PCIe link lanes
pcie_rxp	in	LVDS	std_logic_vector(15 downto 0), PCIe link lanes
pcie_txn	out	LVDS	std_logic_vector(15 downto 0), PCIe link lanes
pcie_txp	out	LVDS	std_logic_vector(15 downto 0), PCIe link lanes
sys_clk_n	in	LVDS	std_logic_vector(ENDPOINTS-1 downto 0), 100MHz PCIe reference clock
sys_clk_p	in	LVDS	std_logic_vector(ENDPOINTS-1 downto 0), 100MHz PCIe reference clock
sys_reset_n	in	LVCMOS18	std_logic, Active-low system reset from PCIe interface
uC_reset_N	out	LVCMOS18	std_logic, Active-low reset for the AtMega uC

Table 7.1: IO pins.

8

Detailed Functional Description and Specification

8.1 Introduction

The FELIX toplevel design instantiates all the components / blocks that are described in the sections in this chapter. The detailed schematic of the toplevel design can be found in Figure 8.1.

The toplevel design (felix_top.vhd) is designed to work for all firmware flavours (GBT, FULL, Strip, Pixel, lpGBT) as well as all hardware platforms (FLX709, FLX712, FLX128, FLX180).

8.2 Compatibility

FELIX had been tested on the following platforms and tools:

1. Operating systems:
    - Scientific Linux CERN 6, kernel 2.6
    - Scientific Linux 7, kernel 3.10
2. Xilinx Vivado:
    - 2020.1: migrated 11-2020
    - 2018.1: migrated 05-2019
    - 2015.4: migrated 02-2016
    - 2014.4: initial version
3. Xilinx FPGA:
    - Virtex-7 690T
    - Kintex Ultrascale XCKU115
    - Virtex Ultrascale+ VU9P, VU37P
    - Versal Prime VM1802

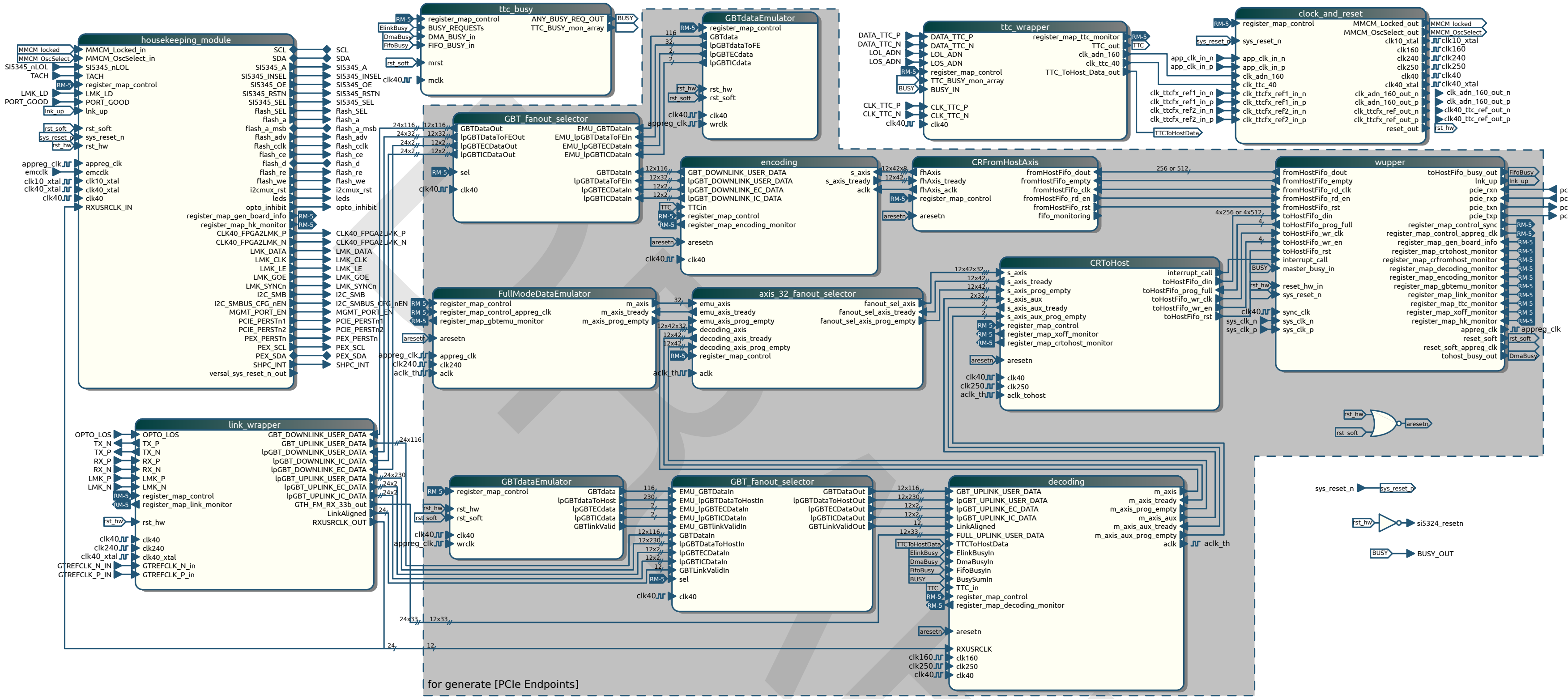


Figure 8.1: The FELIX firmware top level detailed schematic..

8.3 Decoding

8.3.1 Introduction

Decoding is the block in the FELIX firmware which instantiates the subdetector specific, but also Atlas wide protocol handling in the ToHost direction (Upstream).

8.3.2 Interfaces

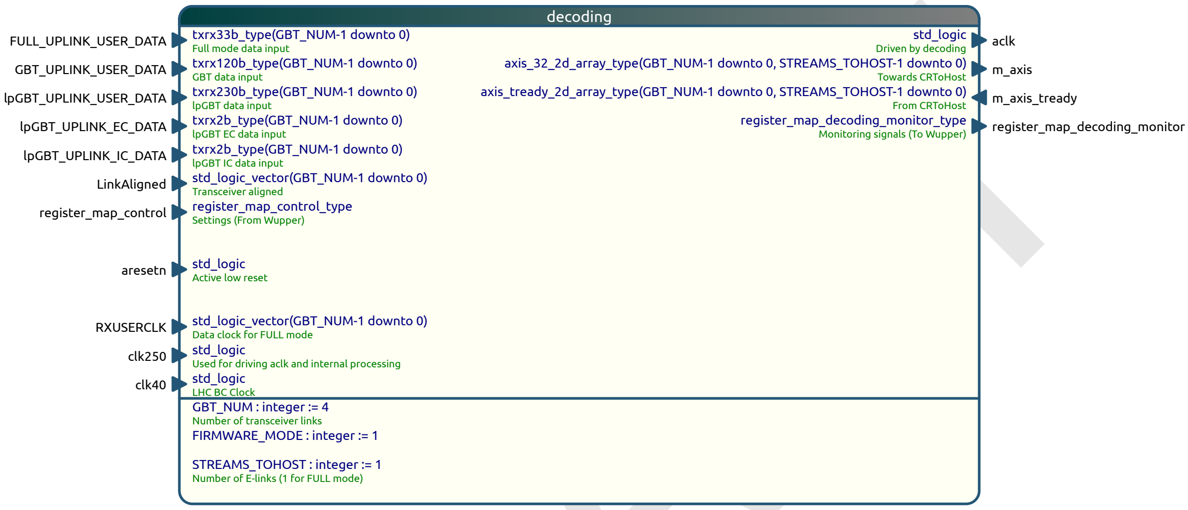


Figure 8.2: The decoding block, instantiating all decoder entities based on FIRMWARE_MODE [7].

8.3.2.1 Overview

The decoder for GBT mode FELIX in phase 2 was derived from the CentralRouter Egroup in phase 1 FELIX. The functionality is the same, but the design will be more modular, and the entities will be more unified among different E-Path / EPROC widths.

Instead of defining a separate entity for every E-link width, as done in phase 1, a configurable and generic gearbox was introduced (see 8.3.9). This gearbox can be configured to support all E-link widths in GBT and lpGBT mode, and output widths for the different protocols (HDLC, 8b10b, Aurora).

The HDLC and 8b10b decoder are very similar to the phase 1 design and can be taken with only slight modification. Finally the GBT mode epath should output the axi stream32 protocol. Therefore the ByteToAxiStream entity was introduced which will take care of the conversion, but also contain the axi stream E-Path FIFO.

8.3.2.1.1 GBT MODE, 8B10B, HDLC

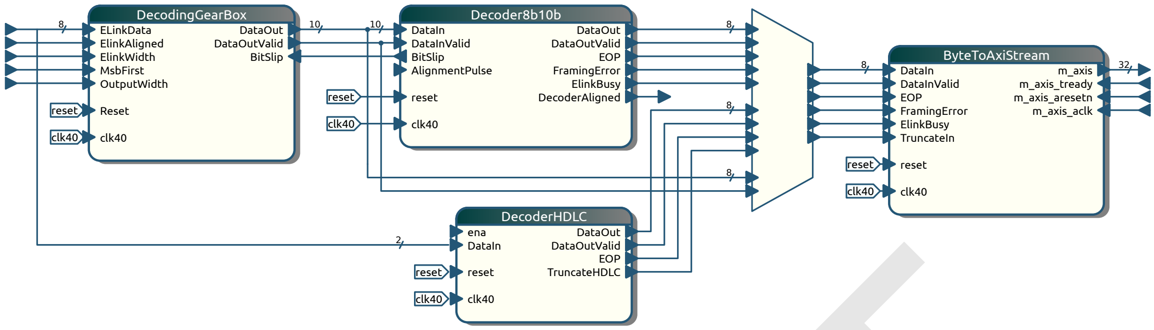


Figure 8.3: Block diagram of a single E-Path decoder in GBT mode.

The E-Path as described in Figure 8.3 can be configured to support 8, 4 and 2 bit E-links, and handle different protocols; 8b10b, HDLC and direct mode (the latter is mainly meant for development purposes, the protocol decoder will be skipped if this mode is selected. There is also no bit alignment)

E-Paths are grouped in an E-group. In phase 2 GBT mode, an e-group has 8 E-paths. This is similar to the behaviour of phase 1, however phase 1 FELIX had 15 E-procs that were fixed in width. Because 4 of the 8 E-paths in phase 2 GBT mode will have a selectable with (2/4/8 or 2/4) only 8 E-paths are needed. The concept of E-proc will be removed in phase 2, only E-path will be used.

Figure 8.4 shows how 8 E-paths are grouped in an E-group, inputting 16 bits of the GBT E-group data. The resulting output data is in the form of an array of AXI Stream 32 buses. Per GBT link this array will have a fixed size of 40 from the Egroups. Additionally 2 AXI stream buses will be added per link for the IC and EC E-link, plus 2 AXI stream buses for the virtual E-links (BUSY/XOFF and TTCToHost). In GBT mode the total number of AXI streams per GBT link will be set to 44.

The GBT mode decoding block will finally handle the data of all the 24 GBT links, outputting a 2 dimensional array of AXI stream 32 buses (24 x 44).

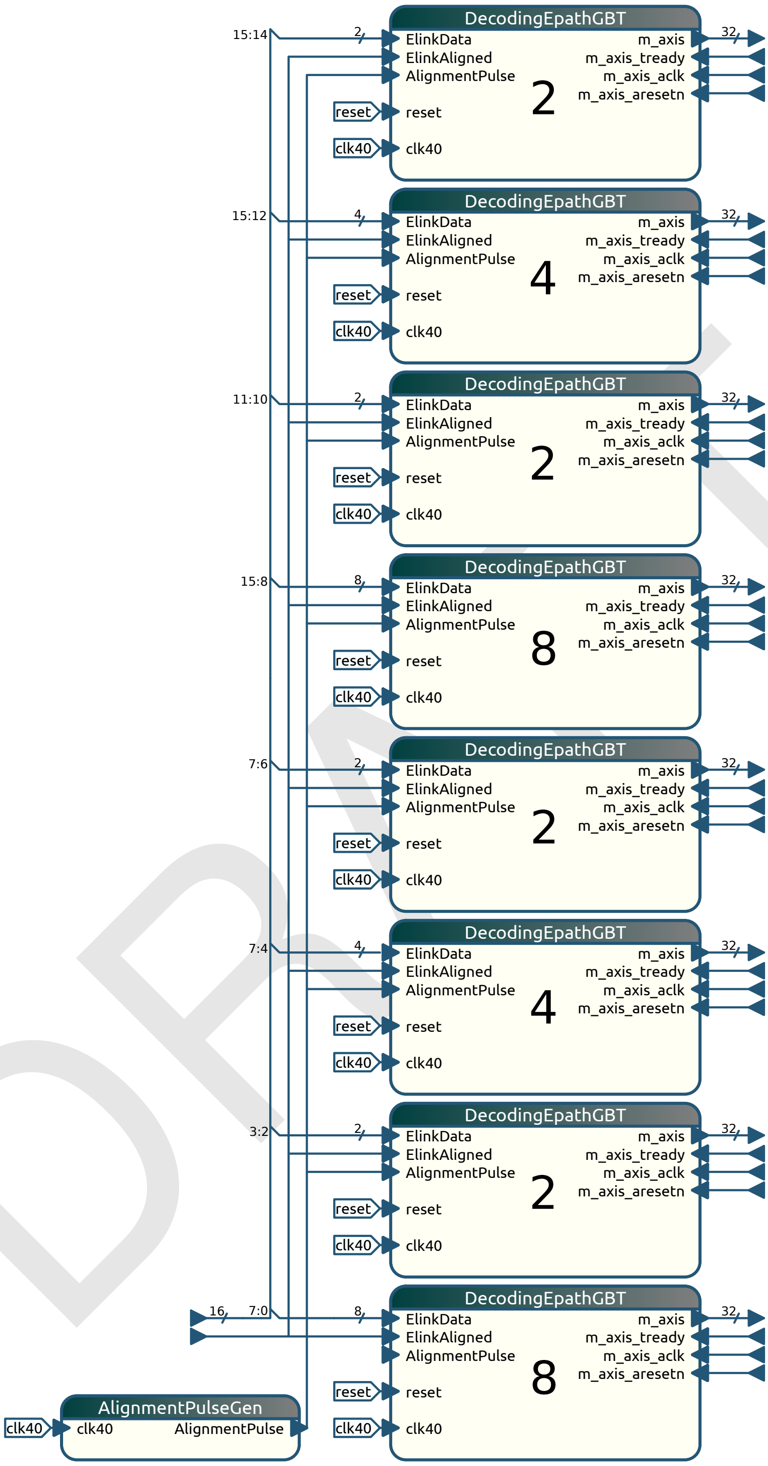


Figure 8.4: Block diagram of an E-Group decoder in GBT mode.

8.3.2.1.2 LPGBT MODE, 8B10B

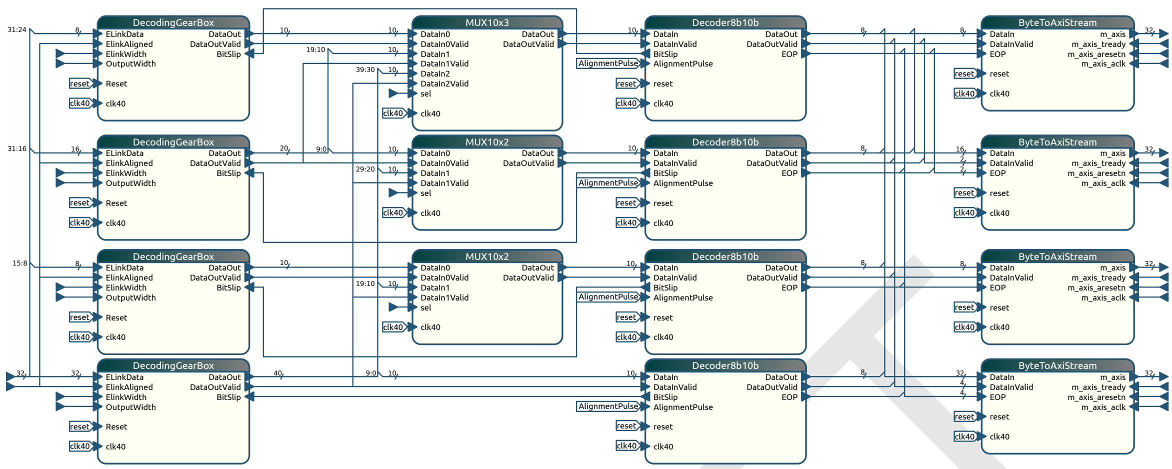


Figure 8.5: Block diagram of an E-Group decoder in lpGBT/8b10b mode.

8.3.2.1.3 LPGBT MODE, PIXEL

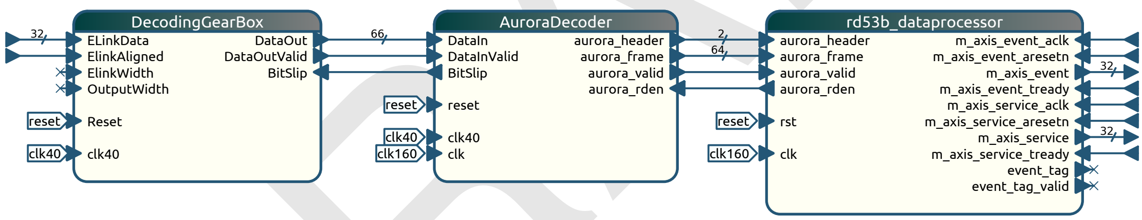


Figure 8.6: Block diagram of a single E-Path decoder in lpGBT / Pixel (RD53b) mode.

8.3.2.2 INTERFACE TO CRTOHOST

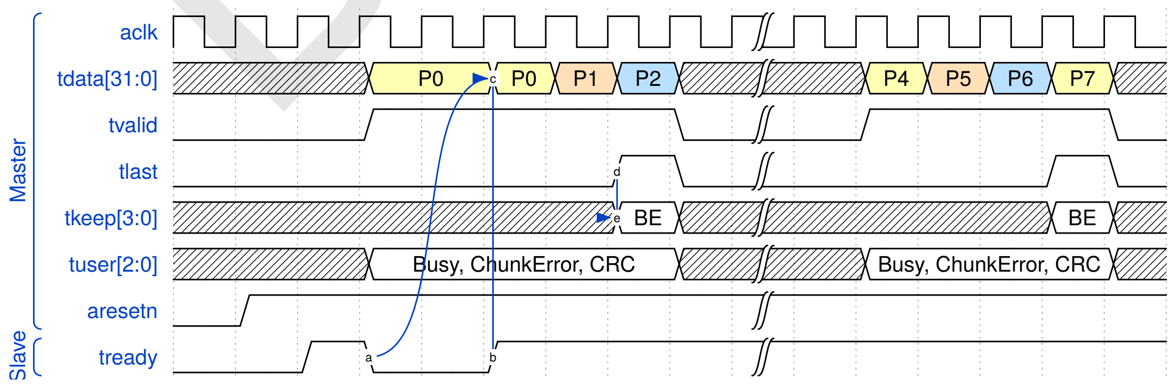


Figure 8.7: Example waveform of a typical AXI stream 32b transfer. [8].

Name	Direction	Type	Remark
clk250	in	std_logic	Used for driving aclk and internal processing
aclk	out	std_logic	Driven by decoding
aresetn	in	std_logic	Active low reset
m_axis	out	axis_32_2d_array_type	Towards CRToHost
m_axis_tready	in	axis_tready_2d_array_type	From CRToHost
m_axis_prog_empty	out	axis_tready_2d_array_type	Towards CRToHost, indicating that 1 block of data is available in the FIFO

Table 8.1: Ports to/from CRToHost..

8.3.2.3 Interface to Link Wrapper

Name	Direction	Type	Remark
FULL_UPLINK_USER_DATA	in	txrx33b_type	Full mode data input
GBT_UPLINK_USER_DATA	in	txrx120b_type	GBT data input
lpGBT_UPLINK_USER_DATA	in	txrx230b_type	lpGBT data input
lpGBT_UPLINK_EC_DATA	in	txrx2b_type	lpGBT EC data input
lpGBT_UPLINK_IC_DATA	in	txrx2b_type	lpGBT IC data input
LinkAligned	in	std_logic_vector	Transceiver aligned
RXUSERCLK	in	std_logic_vector	Data clock for FULL mode
clk40	in	std_logic	LHC BC Clock

Table 8.2: Ports to/from Link Wrapper..

8.3.2.4 Interface to Wupper

Name	Direction	Type	Remark
register_map_control	in	register_map_control_type	Settings
register_map_decoding_monitor	out	register_map_decoding_monitor_type	Monitoring signals

Table 8.3: Ports to/from Wupper..

8.3.3 Functional Description

The decoding block contains no functional logic, it is only used to instantiate the different decoding blocks, depending on the generic FIRMWARE_MODE. Therefore the decoding block contains a set of if/generate and for/generate statements in which the functional protocol decoders are instantiated. Additionally the arrays of buses (AXI stream 32 array, GBT, lpGBT and FULL mode data array) are indexed and routed towards and from the correct decoder.

8.3.4 Configuration

The Wupper registermap will be routed towards the different protocol decoders and virtual E-links. Decoding has no configuration settings itself.

8.3.5 Status Indicators

Status indicators from the various protocol decoders are described in their specific sections.

8.3.6 Latency

Latency of the various protocol decoders is described in their specific sections.

8.3.7 Error Handling

8.3.8 Estimated Resource Usage

Resource	E-Group	GBT link	24 GBT links	% (XKCU115)
LUTs	1348	6740	161760	24.38%
Flip-Flops	1592	7960	191040	14.40%
Block RAM	4	20	480	22.22%

Table 8.4: Resource consumption in GBT mode, fully configurable.

8.3.9 Decoding Gearbox

8.3.9.1 Introduction

for lpGBT and GBT based firmware flavours, the data arrives at E-Link level with for GBT mode 2, 4 or 8 bits per BC clock cycle. for lpGBT mode the data arrives with 8, 16 or 32 bits per BC clock cycle.

The different protocol decoders require different data widths per BC clock cycle, the Decoding Gearbox will deliver these different data widths by means of shift registers to the different decoder blocks. The available widths on in- and output of the gearbox will be partly configurable at runtime and partly at build time.

8.3.9.2 Interfaces

8.3.9.2.1 Overview

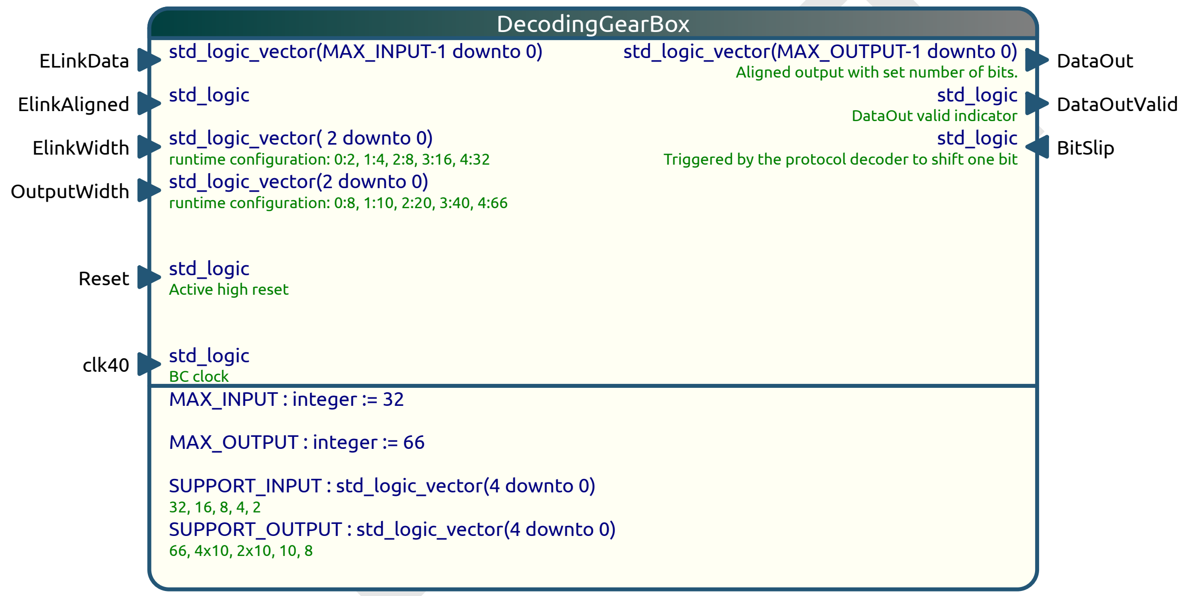


Figure 8.8: The Decoding GearBox entity.

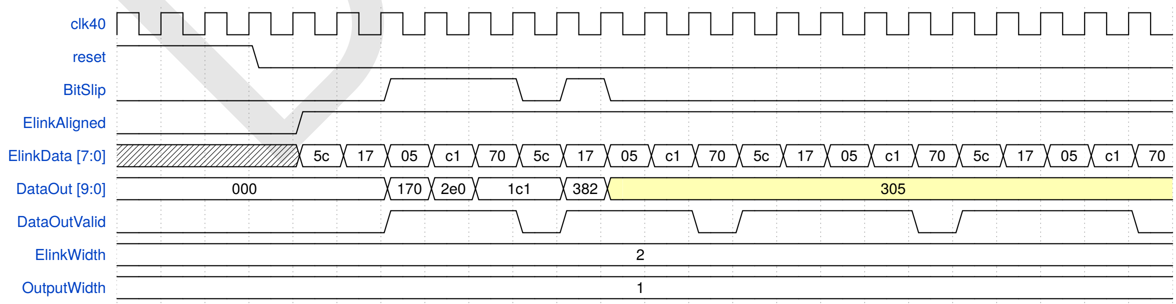


Figure 8.9: DecodingGearBox running with 8 bit input, 10 bit output. The data is constant 0x305 (k28.5+). [8].

8.3.9.2.2 Interface to GBT or lpGBT wrapper

Data from an E-link (lpGBT mode or GBT mode) will be connected to ELinkData. Depending on the maximum required speed of the E-link and also the position of the DecodingGearBox in the E-Group, MAX_INPUT will be set. For instance, a GBT mode E-Group will contain 2 Gearboxes with MAX_INPUT set to 8, 2 Gearboxes with MAX_INPUT set to 4 and 4 Gearboxes with MAX_INPUT set to 2. This way a total of 8 streams of variable bandwidth (80, 160 or 320 Mb/s) can be created.

Apart from ElinkData there is one other connection to the GBT or lpGBT wrapper: ElinkAligned, which will be connected to the GBT or lpGBT aligned flag of the (lp)GBT wrapper.

8.3.9.2.3 Interface to Decoders

3 ports are connected to the different protocol decoders: DataOut, DataOutValid and BitSlip.

DataOut

The input bandwidth / number of bits (MAX_INPUT) should not exceed MAX_OUTPUT. For a 16 bit E-link in 8b10b mode, the OutputWidth has to be set to 20 bits("010"), this way every clock cycle carries 2 8b10b words on DataOut if DataOutValid = '1'. For a 1.28Gb/s E-link in 8b10b the number of 8b10b decoders per DecodingGearBox will be 4.

DataOutValid

DataOutValid indicates that enough bits were shifted into the gearbox, and the correct number of bits were loaded on DataOut. Correct alignment of the 8, 10, 20, 40 or 66 bit word is not guaranteed or indicated in any way. It is the responsibility of the protocol decoder to detect alignment.

BitSlip If the protocol decoder detects a misalignment of DataOut, a pulse of 1 clockcycle can be given on BitSlip. This will shift DataOut by 1 bit.

8.3.9.3 Functional Description

Depending on the configuration, the DecodingGearBox will shift a number of bits of ElinkData (2, 4, 8, 16 or 32) into a shift register every clockcycle. The number of bits in the shift register are counted. Depending on the configured OutputWidth (8, 10, 20, 40 or 66) the data will be loaded on DataOut and the number of output bits will be subtracted from the internal bit counter. When data is available on DataOut, DataOutValid indicates that the data can be loaded into the decoder for further handling.

A pulse one BitSlip will decrement the internal counter by 1, resulting in a bitshift on the output. This can be used for alignment of the data that goes into the decoder.

8.3.9.4 Configuration

Buildtime configuration 4 generics of the DecoderGearBox define its functionality.

- MAX_INPUT: Defines the maximum number of bits that is supported at ElinkData
- MAX_OUPUT: Defines the maximum number of bits that is supported at DataOut
- SUPPORT_INPUT: a 5 bit vector of which every bit configures a supported input width to be configured
  - 0: 2 bit / 80 Mb/s E-Link is supported
  - 1: 4 bit / 160 Mb/s E-Link is supported
  - 2: 8 bit / 320 Mb/s E-Link is supported
  - 3: 16 bit / 640 Mb/s E-Link is supported
  - 4: 32 bit / 1280 Mb/s E-Link is supported
- SUPPORT_OUTPUT: a 5 bit vector of which every bit configures a supported output width to be configured
  - 0: 8 bit output is supported
  - 1: 10 bit output is supported
  - 2: 20 bit (2 x 10 bit) output is supported
  - 3: 40 bit (4 x 10 bit) output is supported
  - 4: 66 bit output (Aurora) is supported

Runtime configuration

The DecodingGearBox can also be configured at runtime, if the option was supported at build time. Two input ports are provided for this purpose:

- ElinkWidth[2:0] can be connected to a register of the Wupper register map to configure the width of the E-Link to be decoded. Possible values are:
  - 0: 2 bit / 80Mb/s Elink connected to ElinkData[1:0]
  - 1: 4 bit / 160Mb/s Elink connected to ElinkData[3:0]
  - 2: 8 bit / 320Mb/s Elink connected to ElinkData[7:0]
  - 3: 16 bit / 640Mb/s Elink connected to ElinkData[15:0]
  - 4: 32 bit / 1280Mb/s Elink connected to ElinkData[31:0]
- OutputWidth[2:0] can be connected to a register of the Wupper register map to configure the width of the path to the decoder. Possible values are:
  - 0: 8 bit for HDLC or no decoding
  - 1: 10 bit for 8b10b decoding
  - 2: 20 bit for 8b10b decoding (2 decoders)
  - 3: 40 bit for 8b10b decoding (4 decoders)
  - 4: 66 bit for Aurora 64b66b decoding

8.3.9.5 Status Indicators

DecodingGearBox has no status indicators. Status of the protocol decoder has to be provided by the decoder itself.

8.3.9.6 Latency

The Decoding Gearbox has a latency for all configurations of 1 clockcycle (40,079 Mhz, 25 ns), that means the output data will be valid 1 clockcycle after the last bits of the E-link data were delivered.

8.3.9.7 Error Handling

DecodingGearBox has no internal error checking. The user / software must make sure that the configuration ports are set up correctly, the protocol decoder should be able to detect and handle protocol errors on the E-link.

8.3.9.8 Estimated Resource Usage

#	In2	In4	In8	In16	In32	Out8	Out10	Out20	Out40	Out66	LUT	FF	Remark
1	✓					✓					33	23	HDLC
2	✓					✓	✓				44	29	HDLC, 8b10b
3	✓	✓				✓	✓				65	37	HDLC, 8b10b
4	✓	✓	✓			✓	✓				93	40	HDLC, 8b10b
5			✓				✓				66	37	8b10b
6			✓	✓			✓	✓			137	71	8b10b
7			✓	✓	✓		✓	✓	✓		400	153	8b10b
8					✓					✓	332	207	Aurora

Table 8.5: Estimated resource consumption for Decoding Gearbox..

In GBT mode firmware we can implement maximum 8 2-bit E-links per E-group, 4 4-bit E-links and 2 8-bit E-links. Assuming a fully configurable 24-channel GBT mode firmware that supports 8b10b, the resources add up as follows for the XKCU115 (Phase I prototype card).

	LUT	FF	LUT(% XKCU115)	FF(% XKCU115)
Egroup	492	270	0.07%	0.02%
Link	2460	1350	0.37%	0.11%
Card (24)	59040	32400	8.90%	2.74%

Table 8.6: Estimated resource consumption for Decoding Gearbox in GBT mode..

The necessary configurations for lpGBT mode are not fully defined yet. It is not clear whether there will for instance be a use case for 8b10b encoding on a 1.28Gb (32 bit) E-link.

Assuming that all the possible 8b10b configurations in lpGBT mode will be implemented, the resources of the XKCU115 (Phase I prototype card) will be as follows.

	LUT	FF	LUT(% XKCU115)	FF(% XKCU115)
Egroup	669	298	0.10%	0.03%
Link	4014	1788	0.61%	0.15%
Card (24)	96336	42912	14.52%	3.63%

Table 8.7: Estimated resource consumption for Decoding Gearbox in lpGBT mode (8b10b)..

In the pixel (RD53b) mode, only Aurora encoding will be used on 32 bit E-links. This will give the following figure on the XKCU115 (Phase I prototype card)

	LUT	FF	LUT(% XKCU115)	FF(% XKCU115)
Egroup	332	207	0.05%	0.02%
Link	1992	1242	0.30%	0.11%
Card (24)	47808	29808	7.21%	2.52%

Table 8.8: Estimated resource consumption for Decoding Gearbox in lpGBT mode (Aurora)..

8.3.10 StripDecoder

8.3.10.1 Introduction

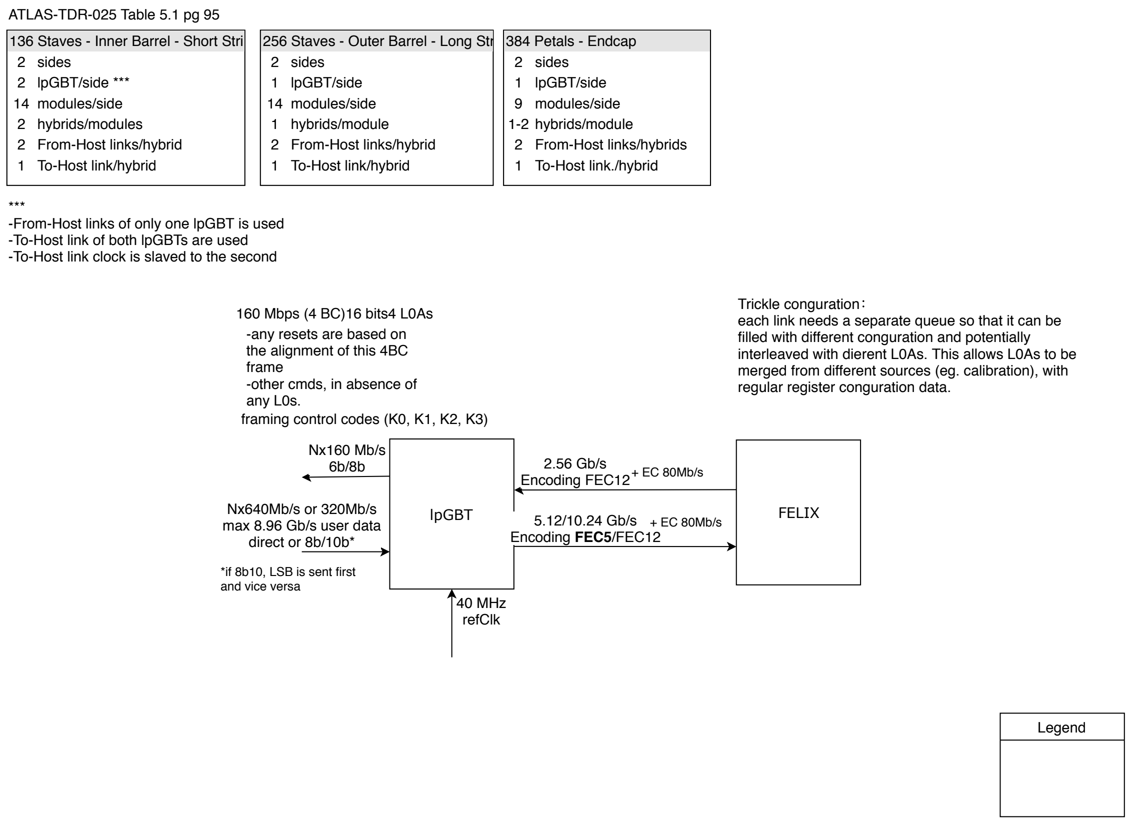


Figure 8.10: The Phase-II ITk Strip data flow and specification..

The decoder for ITk Strips will be the same decoder as described in section 8.3.14. Special K-characters are defined for the strips at build time, but the behaviour is the same.

8.3.11 Endeavour Decoder

8.3.11.1 Introduction

In the lpGBT based firmware flavours there are two blocks designed to communicate with the AMAC ASIC chips: the endeavour decoder and the endeavour encoder. The AMAC is designed to serve monitoring and Low Voltage and High Voltage control functions on the ATLAS ITk strips modules, where it sits on the module power board. The Endeavour is a serial "morse code" protocol used to communicate with the chips and it tolerates $\pm 50\%$ variation with respect to the nominal 40 MHz AMAC ring-oscillator frequency. The Endeavour Decoder is used to decode the information arriving from the AMAC chips. In the following table the bit sequences of the AMAC chip's reply to the commands READ, WRITE and SETID are shown.

Command	Sequence	Length
READ	[amac ID (5b)][Seq num (3b)][Data (32b)]	40 bit
WRITE	[amac ID (5b)][Seq num (3b)]	8 bit
SETID	[amac ID (5b)][Seq num (3b)]	8 bit

Table 8.9: AMAC / Endeavour commands.

"Seq num" is a number that increments by 1 for each valid serial-bus command received by the AMAC.

8.3.11.2 Interfaces

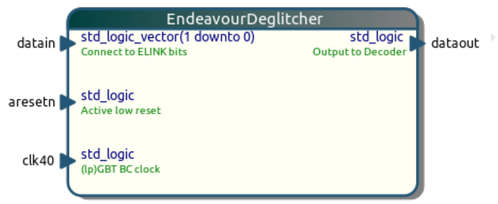


Figure 8.11: The Endeavour deglitcher entity.

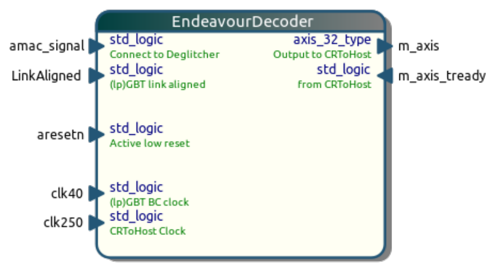


Figure 8.12: The Endeavour decoder entity.

8.3.11.2.1 Overview

The Endeavour Decoder will take input from an E-link (usually the EC e-link of the lpGBT frame) and send its output towards the ToHost Central Router (CRToHost) by means of axi stream 32. Besides the data input and output, there are 2 clock inputs; clk40 which is used for the decoding logic, and m_axis_aclk which serves as a readout clock for the axis fifo. The input port m_axis_aresetn serves as an active-low reset for the decoder and the FIFO. Finally there are two input ports: LinkAligned and Enable; LinkAligned indicates that the lpGBT link is aligned and Enable will come from the Wupper register map. Both signals are a condition for the module to be enabled.

8.3.11.2.2 Interface from E-Link

The Endeavour Decoder will decode the data coming from an EC E-link, connected to the AMAC chip. From the lpGBT wrapper, 2 bits will arrive from the EC E-link on a clock speed of 40 MHz. Since the bandwidth of the serial Endeavour signal is much lower than 40 Mbit/s, and there is a large tolerance on the datarate, both bits will carry the same bit value, unless a transition of the signal just occurred. It is therefore not important which bits will be used from the E-link, in the most simple case, only the LSB of the E-link will be read out in the decoder. Between the EC E-link and the Decoder a block, the Endeavour Deglitcher, takes care of the deglitching of the signal coming from the lpGBT and of the serialization of the 2 bit. In the Endeavour Deglitcher a std logic vector counter increases by 1 whenever the AMAC signal is 1 and decreases by 1 whenver the signal is 0 (the 2 bit of the input line are checked alternately). If the counter reaches 6 the single bit line is set to 1, when the counteres reach 0 the bit line is set to 0. A fifo is used in the Endeavour Deglitcher to absorb the processing time of the deglitcher.

8.3.11.2.3 Interface to CRToHost

The output data is in the form of AXI Stream 32 bus. The Decoding entity will map the output into the right element of the axis_32_2d_array_type, containing axi streams of other E-links.

8.3.11.3 Functional Description

The Endeavour Decoder consists just of a block that decodes the serial information in an 8 or 40 bit word like shown in Table 8.9. The words are decoded using the following protocol.

- The serial line coming from the AMAC chips rests LOW when idle.
- To send a ZERO, the transmitter causes the serial line to go HIGH for $6 < n < 22$ clocks.
- To send a ONE, the transmitter causes the serial line to go HIGH for $29 < n < 124$ clocks.
- A low signal longer than 75 clocks indicates the end of the word.

The words are then sent to an AXI Stream fifo of 32 bit, the 40 bit READ reply is divided in a 32 bit word and an 8 bit word.

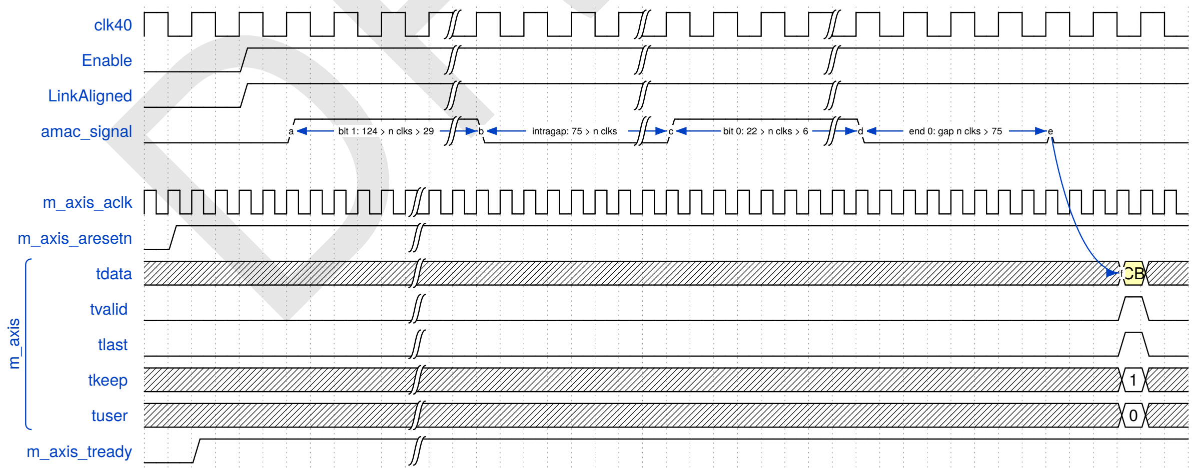


Figure 8.13: example of waveform.

8.3.11.4 Configuration

The Endeavour Decoder can be enabled with an active high Enable input. Also the LinkAligned input has to be 1 in order to enable the Endeavour Decoder.

8.3.11.5 Status Indicators

Other than the error signals in the tuser section of the axi stream signal, the Endeavour Decoder does not indicate any status signals.

8.3.11.6 Latency

8.3.11.7 Error Handling

The chunk error in the tuser word of the axi stream is risen if arrive a sequence of bits that doesn't respect the codification.

8.3.11.8 Estimated Resource Usage

Resource	lpGBT link	24 GBT links	% (XKCU115)
LUTs	237	5688	0.8%
Flip-Flops	177	4248	0.3%
Block RAM	0.5	12	0.5%

Table 8.10: Resource consumption of Endeavour Decoder module.

8.3.12 Aurora Decoder for RD53

8.3.12.1 Introduction

8.3.12.2 Pixel Aurora Decoder

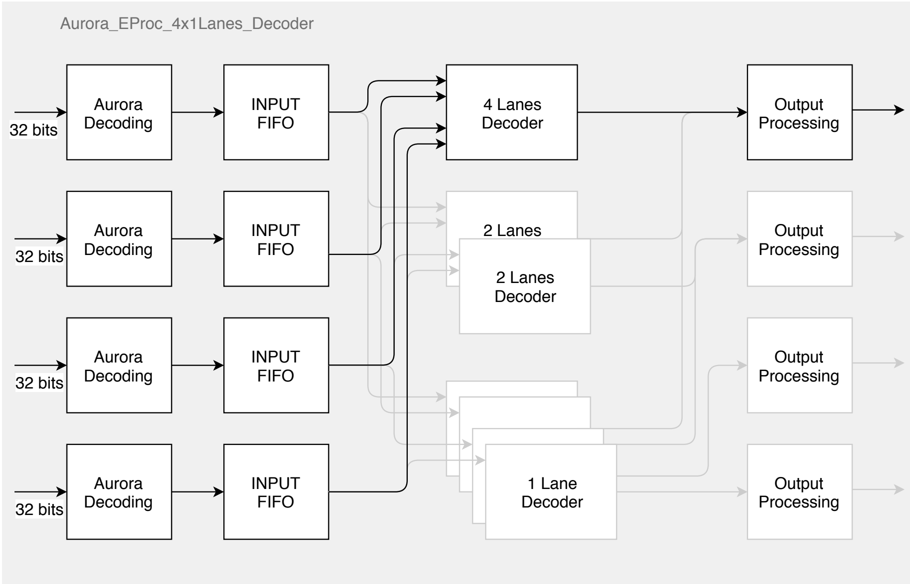
Figure 8.14 shows the block diagram for the ITk Pixel To-Host decoder. The encoding used is Aurora 64b/66b. The 32-bit input utilized 1280 Mb/s e-link. The decoder takes care of aligning and unscrambles the incoming data frames. FIFO is responsible for lane bonding as well as clock domain crossing, hence a 4x clock speed is needed for each FIFO. The output processors filter frames and insert SOP or EOP into each data packet. Four lanes are grouped into one EProc. Three modes are needed, i.e 1x4 lanes, 2x2 lanes and 4x1 lane mode. The number of e-links that are needed depend on each mode.

PCIe Gen4 x16 has a bandwidth in the order of 250 Gb/s. Having 1,28 Gb/s per elink would mean something like 192 elinks per FELIX card should be possible when only taking bandwidth into account. In case of using the lpGBT, ITk Pixel will most probably use 6 elinks per optical fiber, resulting in a maximum of 32 uplink fibers per FELIX card.

In the former case, 48 ITk Pixel EProcs are needed, each serve 4 elinks. We will need 192 Aurora decoders, 192 input FIFOs and up to 192 interfaces between EProcs and Central Router. The 192 input FIFOs can be built out of 192 RAMB36E2 blocks (out of 1080) or 1920 SLICEM. The distributed RAM solution (the SLICEM) will result in a minimal depth of 32 whereas the block RAM has a minimum of 512 words. Therefore, it is proposed to use distributed RAM for the input FIFOs. Those can also be used for CDC.

A single Aurora decoder occupies 430 LUTs and 413 FFs (Table 8.11), which would result in 83k LUTs and 80k FFs for all Aurora Decoders. This is roughly 7% of the LUTs and 3.4% of the FFs in the VU9P. As this is only for the Aurora decoders, it might be too much to include this into a general version of the FELIX FW, but should be fine in case of a ITK Pixel only version.

> **Remark 8.1:** *Update Needed*
>
> The numbers in this estimation cover the Aurora decoder only and are probably out of date, additionally they don't include the RD53B dataprocessor.



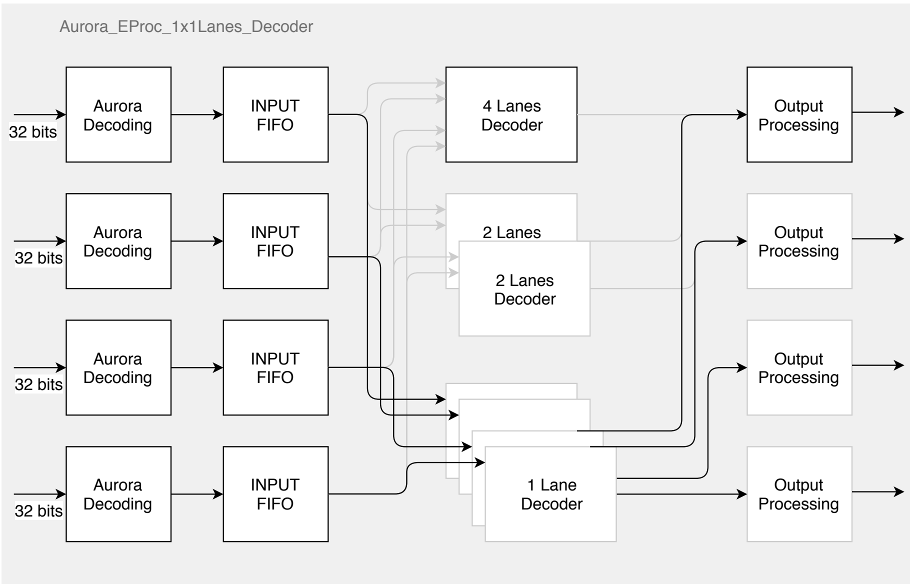


Figure 8.14: The block diagram for the ITk Pixel Aurora To-Host decoder. Two use cases are shown, i.e the 4x1 and the 1x4 lanes. .

8.3.12.3 Interfaces

8.3.12.3.1 Overview

8.3.12.3.2 Interface to component 2

8.3.12.4 Functional Description

8.3.12.5 Configuration

8.3.12.6 Status Indicators

8.3.12.7 Latency

8.3.12.8 Error Handling

8.3.12.9 Estimated Resource Usage

8.3.12.9.1 Aurora decoder

Num. of Aurora Decoder	1	192
LUTs	430	82560
Flip-Flops	413	79296
Block RAM [kb]	TBD	TBD

Table 8.11: Estimated resource consumption for Pixel Aurora decoder in Phase-II..

8.3.13 RD53B Decoder

8.3.13.1 Introduction

The RD53B Decoder is responsible for preprocessing the compressed and encoded data from the ITk Pixel front-end chip. It processes the Aurora-decoded and channel-bonded raw data from the Aurora Decoder (see Section 8.3.12) and splits off event data and service data. The event data is also split into single events and the hit information will be decompressed. Both data streams are sent through the AXI-Stream interface towards the ToHost Central Router (see Section 8.11).

8.3.13.2 Interfaces

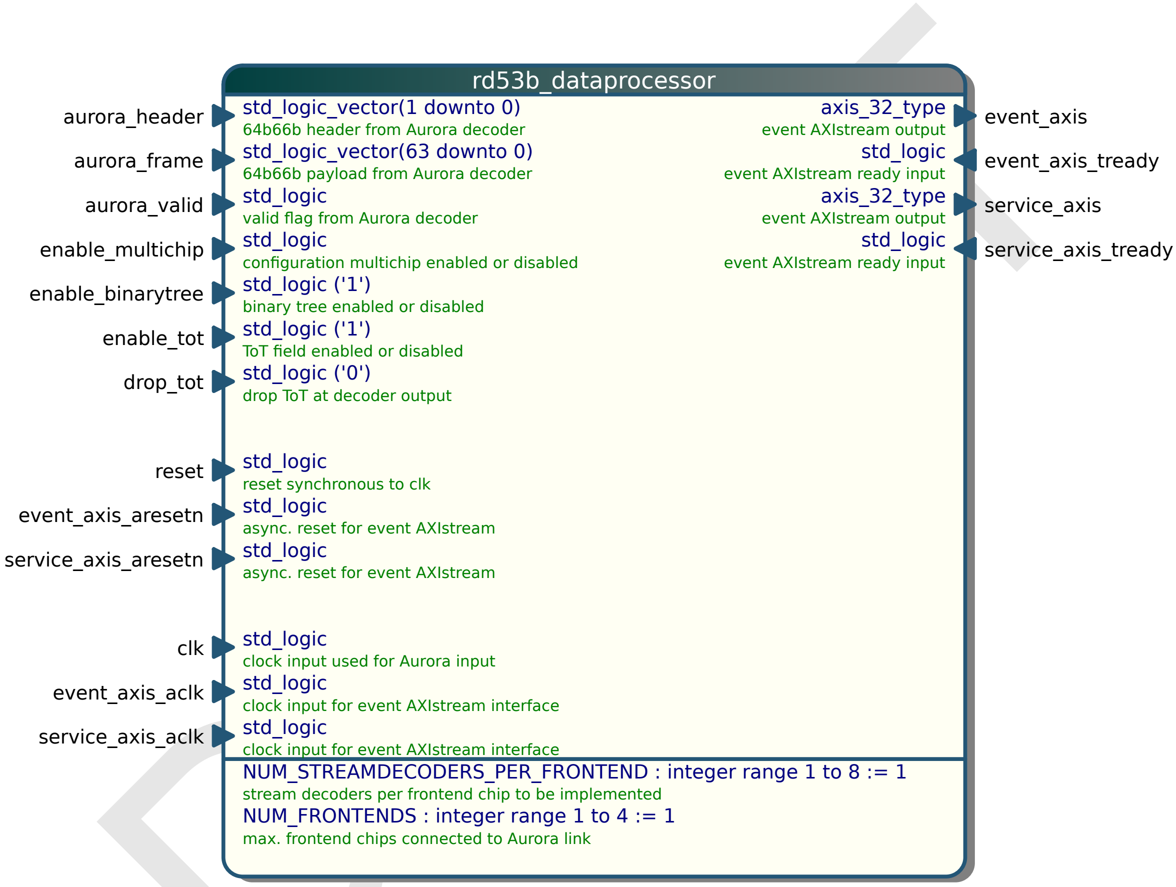


Figure 8.15: The RD53b Dataprocessor entity.

8.3.13.2.1 Overview

The RD53B Decoder has four main interfaces. The incoming data is passed through a simple data bus with data valid signal and no backpressure. All outgoing data is sent through AXI-Stream interfaces to the ToHost Central Router. Additional to the data interfaces, there is also a configuration interface to change settings in the decoder. Figure 8.15 shows the entity of the RD53B Decoder.

8.3.13.2.2 Interface to the Aurora Decoder

The interface to the Aurora decoder is kept as simple as possible. It consists of a 64-bit data bus, taking the data bits of a full Aurora frame. Additional to that a 2-bit data bus carries the header bits of the Aurora frame to be able to distinguish between event and service frames. A data valid bit indicates that both data busses are valid during that clock cycle.

8.3.13.2.3 Interface to the ToHost Central Router

The data outputs for service and event data to the ToHost Central Router are both AXI-Stream 32b busses. An example waveform of the AXI-Stream 32b bus is given in Figure 8.7.

> **Remark 8.2:** *ToDo*
>
> Describe internal data structure of the AXI interface!

8.3.13.3 Functional Description

8.3.13.3.1 Input stage

The input stage will distribute the Aurora frames to the different sub-decoders. First it will distinguish service data from event data by looking at the Aurora header. All frames with header `10` are identified as service frames. If these service frames contain register data (Aurora codes 0xB4, 0x55, 0x99, 0xD2) they are put into the service data FIFO. Frames with header `01` contain event data. They are split by front-ends (if multi-chip readout is enabled) and streams and put into FIFOs in front of every stream decoder. Frames with an invalid Aurora header (`00` or `11`) are dropped.

8.3.13.3.2 Stream decoder

The stream decoders are the central part of the RD53B Decoder. Each stream decoder will process a single event stream from an RD53B front-end chip. Internally, it contains a finite state machine which controls the splitting of the different fields in the event stream. The fields are then re-assembled into an AXI-Stream 32b bus. If a stream contains multiple events, they are split into packets.

8.3.13.3.3 Output multiplexer

The output multiplexer is responsible for merging the event streams from multiple stream decoders into a single AXI-Stream 32b bus. First, all events are collected into packet FIFOs. If a packet is completed it will be forwarded to the output in one piece. The arbitration of this merging is round-robin.

8.3.13.4 Configuration

The configuration of the RD53B Decoder is split into two parts. There is a static synthesis-time configuration, and a dynamic run-time configuration.

The static configuration options are passed as a generic:

- **`NUM_STREAMDECODERS_PER_FRONTEND`**: integer between 1 and 8, will define the maximum number of streams which can be processed in parallel. As each stream decoder has a limited throughput this number should be at least $\lceil \frac{\text{front-end bandwidth}}{\text{stream decoder througput}} \rceil$
- **`NUM_FRONTENDS`**: integer between 1 and 4, will define the number of front-end chips sharing a single Aurora link. Each front-end requires its own stream decoder.

The dynamic configuration is passed through four configuration bits in the port of the RD53B Decoder. These bits can be changed at run-time and have to match the configuration of the front-end chip, otherwise the decoding will not work as expected and produce garbage. Following configuration bits are available:

- **`enable_multichip`**: a logic-1 on this port enables the multi-chip readout mode in the decoder. If the multi-chip mode is enabled, each Aurora frame has to start with a 2-bit chip ID followed by the data for this particular front-end chip. If the multi-chip mode is disabled, all data bits of the Aurora frame will be decoded. **Note:** The front-end chip usually powers-up in the multi-chip mode, even if only a single front-end chip is connected to the Aurora link.

- **enable_binarytree**: a logic-1 on this port configures the decoder to uncompress the binary tree into the uncompressed 16-bit hitmap. For debugging purposes the binary tree can be disabled in the front-end chip. Then the decoder has to be configured accordingly.
- **enable_tot**: If ToT fields in the data stream are present this bit has to be set to logic-1.
- **drop_tot**: To reduce the output bandwidth of the RD53B Decoder the ToT fields can be dropped in the decoder.

8.3.13.5 Status Indicators

Currently, there are no status indicators foreseen. The RD53B protocol does not contain enough redundancy for proper error checking. Also the stream-based encoding allows to recover from decoding errors at the beginning of a new stream.

8.3.13.6 Latency

Latency studies have been made with a data set from the ATLAS simulation group for a specific region in the outer part of the ITk Pixel detector. Under the assumption of a 50 % link occupancy and with five stream decoders per front-end chip, the latency between input and output of the decoder has been measured using a behavioral simulation. In this simulation the latency is defined as the difference between two timestamp. The first timestamp is set, when the event header is provided at the input of the decoder. A second timestamp is created when the event appears in the AXI stream at the output of the decoder. Figure 8.16 shows the latency distributions of all simulated events for different number of events per data stream.

Also the impact of the binary-tree encoding in the event data has been analyzed with respect to latency. Therefore, an example data set with the uncompressed raw 16-bit hitmap has been generated. Figure 8.17 shows the latency distributions for the same number of events per stream as before, but with disabled binary tree.

8.3.13.7 Estimated Resource Usage

For the resource usage the total number of stream decoders is important:
NUM_STREAMDECODERS_TOTAL = **NUM_STREAMDECODERS_PER_FRONTEND** × **NUM_FRONTENDS**

It is now possible to estimate the resource usage for each type of FPGA element:

- LUTs = 3200 × **NUM_STREAMDECODERS_TOTAL**
- Flipflops = 720 × **NUM_STREAMDECODERS_TOTAL** + 264
- BRAMs = 2 × **NUM_STREAMDECODERS_TOTAL** + 1

All numbers have been derived from synthesis results of the pure RD53B Decoder.

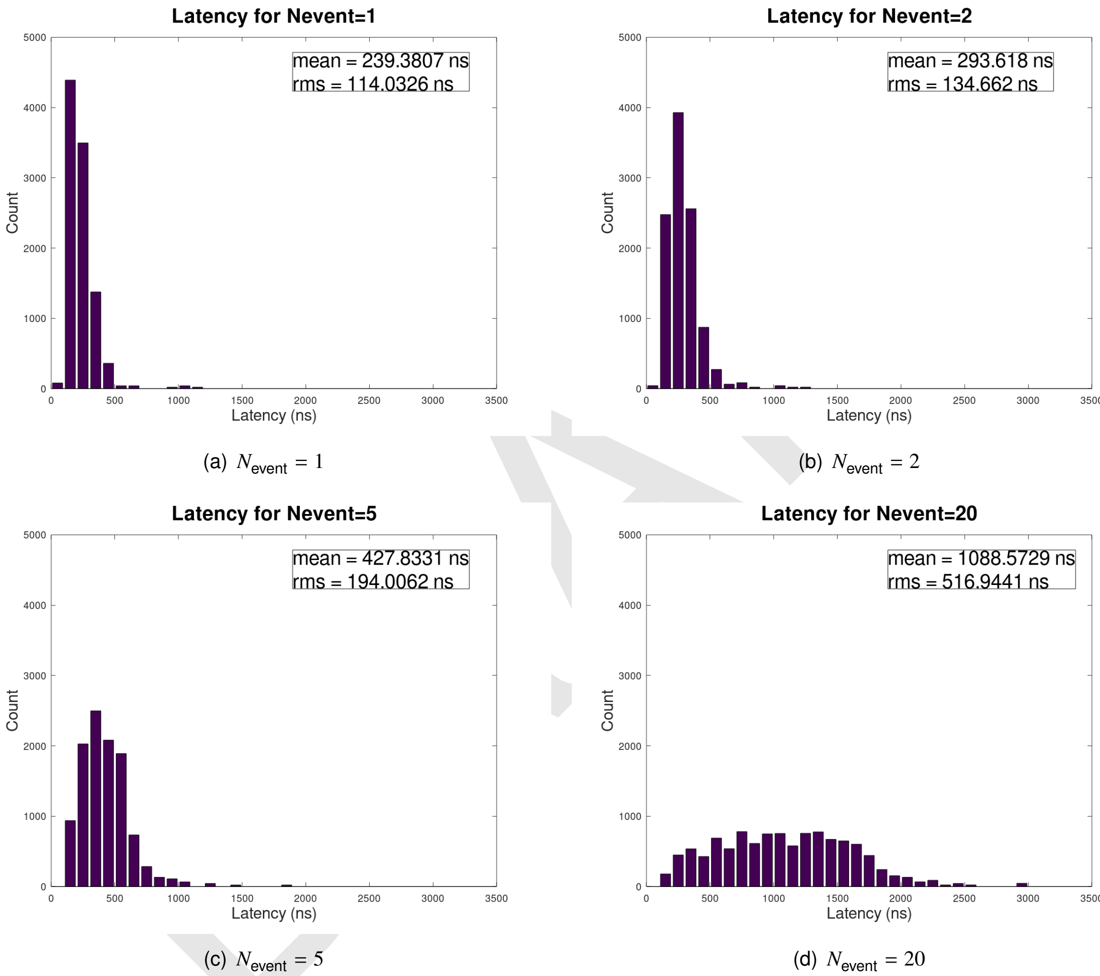
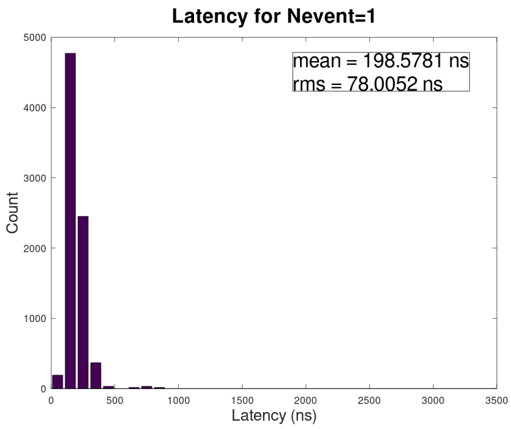
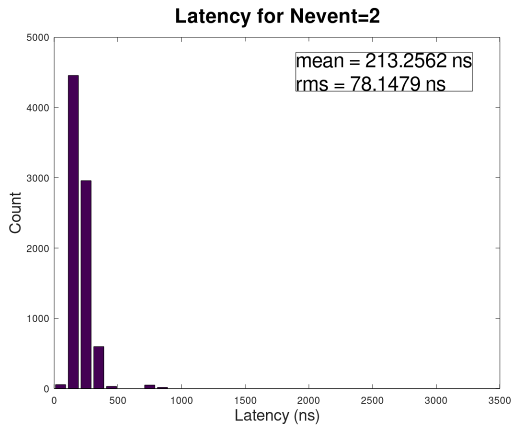


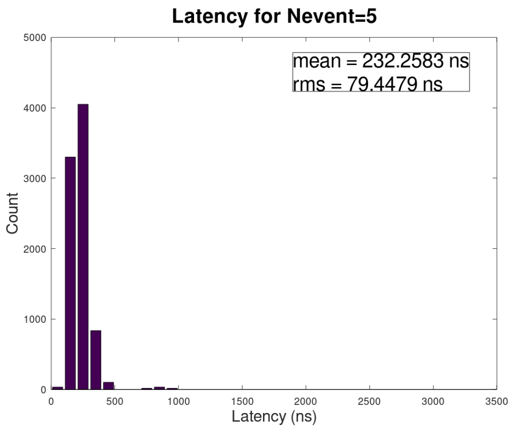
Figure 8.16: RD53B Decoder latency for different number of events per stream (N_{event}) with a binary-tree encoded hitmap.



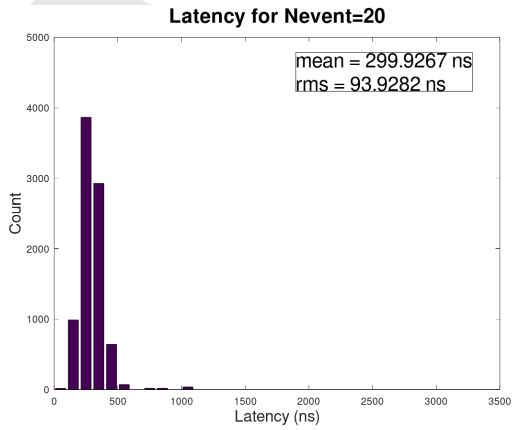
(a) $N_{\text{event}} = 1$



(b) $N_{\text{event}} = 2$



(c) $N_{\text{event}} = 5$



(d) $N_{\text{event}} = 20$

Figure 8.17: RD53B Decoder latency for different number of events per stream (N_{event}) with uncompressed hitmap..

8.3.14 8B10B E-LINK DECODER

8.3.14.1 INTRODUCTION

The 8b10b Decoder has been extensively used in phase 1 FELIX in GBT mode. In Phase II, the 8b10b decoder has been decoupled from the E-proc, and retains in the generic E-Path in GBT and lpGBT mode firmware flavours.

The tasks for the 8b10b decoder are:

- Alignment of the 8b10b words using K28.5 / BitSlip
- Decode the 8b10b stream to 8-bits + CharIsK
- Detect Framing Errors
- Detect E-link BUSY assertion
- Deframing: Convert Decoded byte + CharIsK into DataOut, DataOutValid and EOP

8.3.14.2 INTERFACES

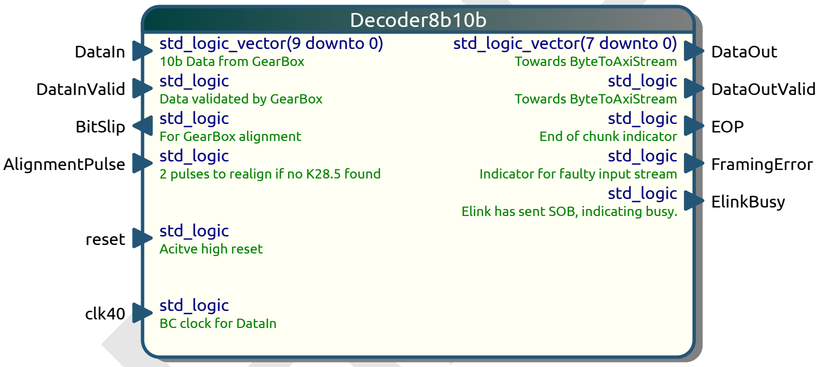


Figure 8.18: The 8b10b Decoder entity.

8.3.14.2.1 INTERFACE TO DECODINGGEARBOX

The 8b10b decoder receives DataIn[9:0] and DataInValid from the DecodingGearBox. If the 10 bit word is misaligned, a pulse can be generated on BitSlip in order to skip one bit in the gearbox.

8.3.14.2.2 INTERFACE TO BYTETOAXISTREAM

All the outputports of the 8b10b decoder will be connected to ByteToAxiStream.

- DataOut[7:0] : Contains payload data. Comma characters are stripped from the data stream
- DataOutValid : Indicates that DataOut contains payload data
- EOP : End of packet (chunk) indicator
- FramingError : EOP or SOP character was missing from the input stream.
- ElinkBusy : FrontEnd has asserted busy (Using SOP K-character.)

8.3.14.3 Functional Description

8.3.14.3.1 Alignment

The 8b10b encoder must perform an alignment sequence on the 10b word on DataIn. When 2 **consecutive** Idle Comma characters are received (K28.5 in GBT mode, K28.1 in Strip or FEI4 mode), the Decoder is in an aligned state. Only in the aligned state, DataOutValid will be asserted. A timer external to the 8b10b decoder generates pulses at a given adjustable interval. The decoder should count 2 pulses. Detection of 2 consecutive Idle comma characters causes the counter to reset to 0. If the counter arrives at the value of 2, the alignment state of the decoder will be deasserted and a pulse will be given on BitSlip. BitSlip will cause the DecodingGearBox to skip one bit and the decoder cat retry the alignment sequence.

8.3.14.3.2 8b10b decoding

Comma characters:

Function	GBT mode	Strip/LCB	FEI4	Meaning
Comma	K28.5	K28.1	K28.1	Idle character
SOP	K28.1	K28.7	K28.7	Start of chunk / packet
EOP	K28.6	K28.5	K28.5	End of chunk / packet
SOB	K28.2	N/A	N/A	Start of busy
EOB	K28.3	N/A	N/A	End of busy

Table 8.12: Comma characters with a special meaning in different firmware flavours.

The functional description of the 8b10b decoder itself, converting a 10b word into 8 bit + CharIsK is well defined in other literature, and the code has been implemented in phase 1 FELIX.

8.3.14.3.3 Framing error detection

A chunk or packet of data coming from the FrontEnd electronics over an E-link should be encapsulated in SOP and EOP characters (see table 8.12). A framing error is asserted if any of the following conditions is violated:

- A payload data byte that is not encapsulated within SOP / EOP
- An SOP occurring before a chunk was ended with EOP
- An EOP occurring without an SOP.

Note that SOB and EOB (Start and End of BUSY) may occur at any moment within or outside a chunk. Also IDLE comma characters may be inserted in the middle of a chunk without assertion of FramingError.

8.3.14.3.4 E-link busy assertion

An FrontEnd may assert BUSY by sending an SOB (Start Of BUSY) character (K28.2, see 8.12) EOB (K28.3) will deassert BUSY. Whether the BUSY LEMO connector will actually be raised on E-link busy can be configured through the register map, see also section **??**

8.3.14.3.5 Deframing

DataOut will contain only the payload data (CharIsK = '0') that is decoded from the 8b10b stream. The last byte of a chunk / packet will be indicated with EOP. For this mechanism an extra pipeline stage after the 8b10b decoder is needed to store the payload data, until the next byte is validated. If the next byte is an EOP comma character, the EOP signal will be asserted with the last payload byte. SOP, Idle, SOB and EOB characters will simply be ignored by the deframer.

8.3.14.4 Configuration

The meaning of the different comma characters in table 8.12 can be configured based on the FIRMWARE_-MODE generic at build time. It is not foreseen at the moment to make a runtime configurable option for the 8b10b decoder.

8.3.14.5 Status Indicators

The 8b10b decoder will output ElinkAligned into the Wupper registermap. Framing errors and busy will be reported through the datastream and will end up in chunk trailers.

8.3.14.6 Latency

The 8b10b decoder has a latency of 1 clock cycle (25 ns). The deframer adds another clock. This will bring the total latency of the 8b10b Decoding block to 2 BC clocks or 50 ns.

8.3.14.7 Error Handling

Misalignment of the 8b10b encoded E-link is reported through the Wupper registermap. Framing error and ElinkBusy will be reported through the data stream.

8.3.14.8 Estimated Resource Usage

The resource usage will be estimated for the complete GBT Egroup and the complete decoding block per firmware mode.

8.3.15 HDLC E-LINK DECODER

8.3.15.1 INTRODUCTION

The HDLC Protocol [9] is used by the GBTx chip, to configure the chip itself through the Internal Control (IC) E-link, and to communicate with the GBT Slow Control Adaptor (GBT-SCA) over the External Control (EC) E-Link or any other 80 Mb/s E-link of the GBT or lpGBT.

The HDLC decoder used in Phase II FELIX was based on the GBT-sc module for FPGA by Julian Mendez [10]. Only the deserializer was used to decode the bytes. All higher level decoding that is covered in the original GBT-sc module was left out in FELIX and instead handled by software, in order to save FPGA resources. Additionally the deserializer was modified to fit FELIX requirements with the following modifications:

- The interface was modified to fit ByteToAxiStream
- A truncation mechanism was added
- The deserializer for IC and EC were merged into a single file.

8.3.15.2 INTERFACES

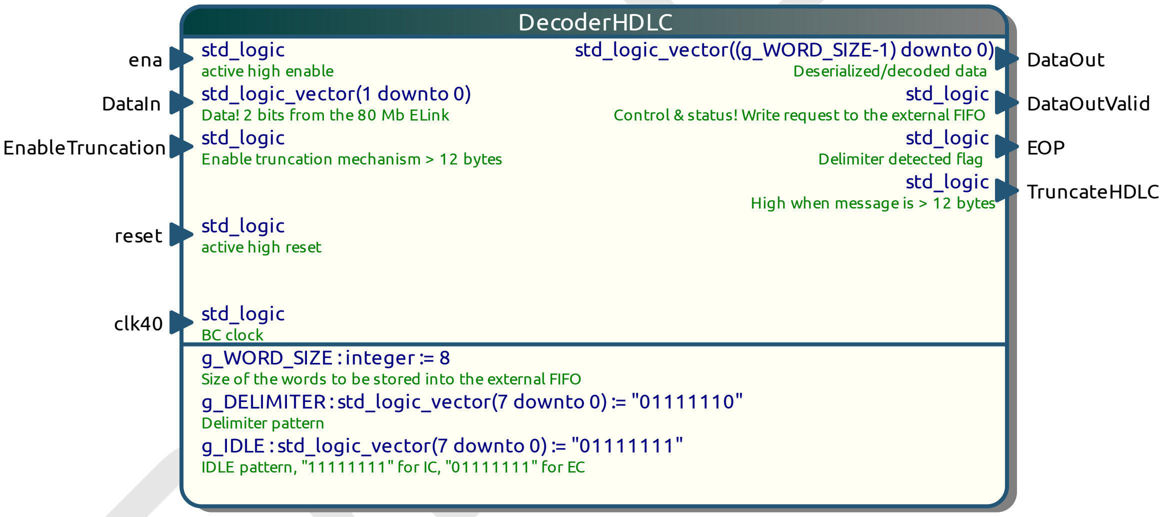


Figure 8.19: The HDLC decoder entity.

8.3.15.2.1 GENERICS

- g_WORD_SIZE: This generic should be set to 8 to be compatible with the FELIX operation.
- g_DELIMITER: The standard delimiter or FLAG is by default set to 0x7E, and should be unchanged.
- g_IDLE: The IDLE pattern, or ERROR FLAG in the HDLC specification is defined differently by the GBTx chip (IC link) and the GBT-SCA (EC link), therefore it can be set to 0xFF for IC and ox7F for EC.

8.3.15.2.2 ELINK INTERFACE

The HDLC decoder does not connect to the DecodingGearbox, since it only connects to 2-bit (80 Mb/s) E-Links. Instead it connects directly to the 2 bits of the E-Link data.

8.3.15.2.3 Interface to ByteToAxiStream

The HDLC decoder is combined with other decoders (8b10b, direct) in one DecodingEpath, and therefore shares its output with these other decoders. The output port consists of:

- DataOut: 8-bit output data
- DataOutValid: Indication that DataOut should be registered this clock cycle
- EOP: End of packet indication
- TruncateHDLC: Indication that the current packet consists of more than 12 bytes, if EnableTruncation is set.

8.3.15.3 Functional Description

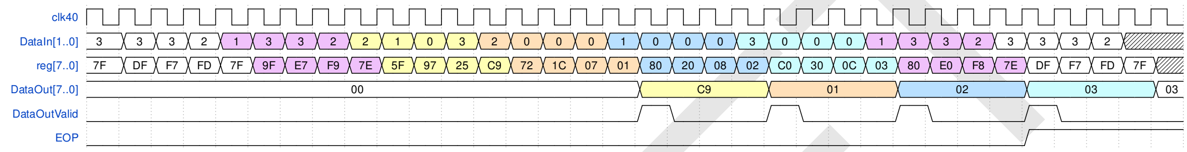


Figure 8.20: The HDLC decoder waveform.

The HDLC decoder is a shift register that shifts in 2 bits at a time. Data arrives LSB first, for the E-Link bits (DataIn) the LSB arrives at bit 1, bit 0 is the second bit. The deserializer process has a bitstuffing detection, if 5 consecutive ones are detected, the next '0' is removed. If this is not the case, a FLAG or IDLE message is marked.

A second process buffers the deserialized byte, and if a FLAG is decoded after the data byte, the byte is marked as EOP (end of packet).

Additionally, a truncation mechanism can be enabled. For this mechanism, a counter counts the number of bytes before a FLAG, if this number exceeds 12, the TruncateHDLC output will be asserted.

8.3.15.4 Configuration

The HDLC decoder has two configuration inputs:

- ena: To enable the decoder. Setting this input to '0' will keep DataOutValid low.
- EnableTruncation: This input enables the truncation mechanism which limits the chunk size to 12 bytes.

8.3.15.5 Status Indicators

The outputs will be handled by the tuser bits of the AXI Stream, and marked as flags in the trailer bits of the chunk trailer by CRToHost.

8.3.15.6 Latency

One byte arrives 2 bit per BC clock cycle and therefore takes 4 clockcycles to clock into DataIn. Once the last bits of the data have arrived, the byte is available in the internal shift register of the decoder called "reg" (see Figure 8.20). In order to make the decoder compatible with AXI Stream, the last byte has to be synchronized with the end of packet indication, therefore the data must be buffered to see if the next byte is a "Flag" to indicate the end of a frame. This mechanism takes a total latency of 5 clock cycles, but 1 additional clock needs to be accounted for if a '0' is stuffed in the data, as described in the HDLC protocol [9]

8.3.15.7 Error Handling

The HDLC Decoder has a truncation mechanism that limits the bandwidth in case of a faulty E-Link which generates random data. It limits the chunk to 12 bytes, any data after that will be ignored by CRToHost.

8.3.15.8 Estimated Resource Usage

The resource usage of the complete GBT E-group, including the HDLC decoder is shown in Table 8.4.

8.3.16 FullModeDecoder

8.3.16.1 Introduction

In Phase I FELIX, two types of link protocols were supported; GBT (4.8Gb/s, divided into E-links) and FULL (9.6Gb/s 8b10b). The FULL mode protocol will be implemented in the Phase II firmware without functional modification.

The protocol specified in this section, called “FULL mode” (The name originates from FULL bandwidth). Full mode is intended for high bandwidth (i.e. 9.6 Gb/s) connections from FPGAs, as opposed to links from the GBTx ASIC [11]. Data is streamed to FELIX without any handshaking.

At this time, the need for Full mode links only in the ToHost direction has been expressed. The opposite, from-FELIX, direction would use standard GBTx protocol. The rest of this section will therefore focus on the ToHost Full mode direction only. Should the need for Full mode in the FromHost direction be needed in future, it can be implemented in a similar manner. Figure 8.21 shows a block diagram of both the FrontEnd and FELIX ends of a Full mode link in the to-FELIX direction. The number of channels supported by single FELIX FPGA is not yet determined. As an upper limit estimation, six channels, each with a maximum payload throughput of 7.68 Gb/s (9.6 Gb/s reduced by 8b/10b encoding) could be transferred within the PCIe Gen3 8-lane bandwidth (maximum 64 Gb/s). FELIX based on the FLX712 FPGA platform has two such PCIe interfaces which may be combined to a single 16-lane interface. A standard FLX712 FULL mode build in the FELIX release has 24 channels, however we recommend to connect only 12 out of the 24 channels, unless the transceiver bandwidth is limited by means of the XOFF mechanism, see also 8.3.16.3.1.

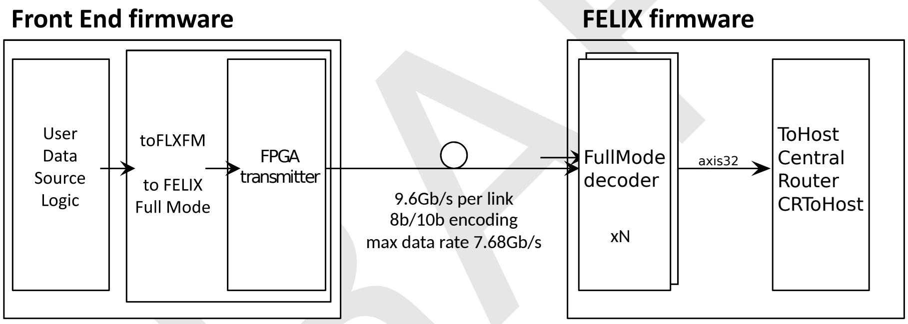


Figure 8.21: Block diagram of both the FrontEnd and FELIX ends of a Full mode link in the ToHost direction.

In summary, the main features of Full mode are:

- Channel line transmission rate of 9.6 Gb/s
- Maximum user payload of 7.68 Gb/s
- 8b/10b encoding
- Logical packets: packets are multiples of 32-bit words, no maximum packet size is specified.
- Option to include a stream id per packet for transmitting different logical data streams on the same physical link. Streams may be routed by FELIX to different network endpoints. When the E-link is configured to have stream ids, they are included as the low byte of the first word of every packet of user data.
- Support for forwarding BUSY from the Front End to the Central Trigger. Policies for asserting BUSY are not determined by FELIX.
- Possibility of flow control with XON, XOFF symbols sent from FELIX on a GBT normal mode E-link.
- A user example design with a FIFO-like interface has been provided.

8.3.16.2 INTERFACES

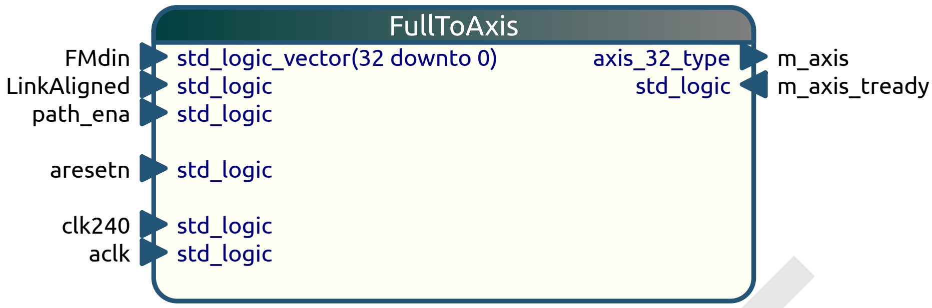


Figure 8.22: The FULL mode decoder entity.

8.3.16.2.1 INTERFACE FROM LINKWRAPPER

The FULL Mode decoder has two ports that connect to the LinkWrapper in FULL mode:

- FMdin: This 33-bit signal carries the 32 data bits (bits [31..0]). The MSB, bit 32 indicates that bits 31..24 carry a K-character (Idle, SoP, EoP, SoB, EoB).
- LinkAligned: This input indicates that the transceiver is properly aligned and able to receive data from the Front End.

8.3.16.2.2 INTERFACE TO CRTOHOST

The interface to the Central Router ToHost (CRToHost) is the same as for other decoders: axi stream 32. In de decoding block the axis32_type outputs from the FullModeDecoder will be combined into a 2D array of axis32_2d_array_type with the first dimension the number of links (GBT_NUM) and the second dimension is set to 1, because every link has only one logical link. In summary, each Full mode connection is essentially a high bandwidth 8b/10b E-link.

The axis32_type is defined in axi_stream_package.vhd. The individual record fields are described in Table 8.13

Table 8.13: 32 bit axi stream interface.

Field	Bits	Description
tdata	[31..0]	Payload data
tvalid	0	Indicates that a data chunk is active
tlast	0	Indicates the last 32 bits of a chunk
tkeep	[3..0]	Byte enable, always "1111" for Full Mode
tuser	3	Truncation, indicates that data was received while a FIFO was full, a part of the chunk was discarded.
tuser	2	Link Busy, Asserted when SOB is received, deasserted when EOB is received
tuser	1	Chunk error, Asserted when data is not correctly embedded within SoP/EoP
tuser	0	CRC20 error, Asserted when the CRC20 calculation over the payload does not match the CRC20 field in th

8.3.16.3 FUNCTIONAL DESCRIPTION

The FULL Mode Decoder (FullToAxis) interprets the K-characters as cescribed in section 8.14, and translates the stream of data into the industry standard AXI4 stream bus, which can be handled by the CRToHost entity.

Table 8.14: K-characters used in FULL Mode.

K-character	8-bit value	Use
K28.1	0x3c	Start-of-Packet, SOP
K28.6	0xdc	End-of-Packet, EOP
K28.5	0xbc	idle
K28.2	0x5c	BUSY-ON
K28.3	0x7c	BUSY-OFF

The idle K-character is the comma character defined for the serializer core that forces 32-bit alignment. The format of the data transmission between the serializer and deserializer of the Full mode wrapper is shown in Figure 8.23. See Section 8.3.16.3.2 for details on the CRC.

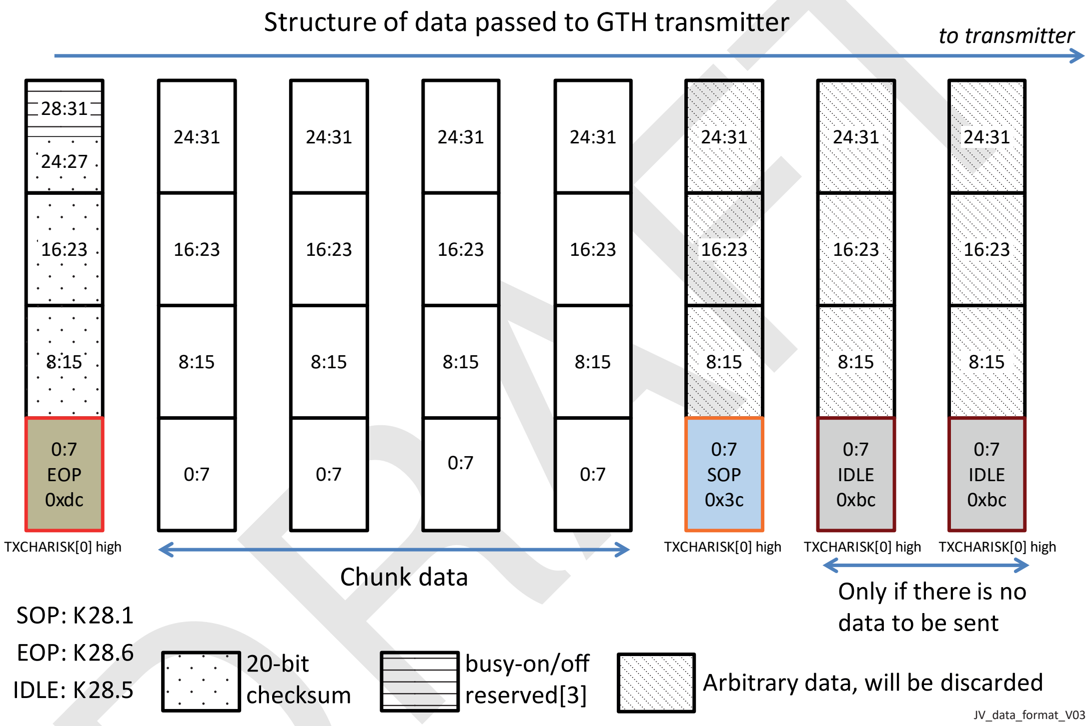


Figure 8.23: The format of the data transmitted between the serializer and deserializer of the Full mode wrapper.

8.3.16.3.1 Flow control

If a Front-end requires FELIX to assert BUSY to CTP it will transmit BUSY-ON K-character via the stream controller interface (defined as rising edge of BUSY line). On receipt of this, FELIX asserts BUSY for a minimum of two 40 MHz clock cycles. While in BUSY state FELIX will fill its input buffers and send out data to host flagged with a BUSY symbol. Once the buffers are full FELIX will reject all subsequent data until BUSY-OFF is received.

Once the condition is cleared the front-end should send a BUSY-OFF K-character (falling edge of busy line), causing FELIX to de-assert BUSY to CTP. Caveat: users should account for one extra word of data being added by FELIX for insertion of BUSY symbol, otherwise buffers will overflow. The K-characters used for BUSY signalling will not appear in the FELIX data stream (but can be flagged to processing code or used to generate interrupts as needed) The EOP word for each packet should contain BUSY state, to allow for recovery if signal on busy line not received.

If the data rate of the 24 FULL mode links exceeds the PCIe bandwidth towards the host server, the XOFF flow control system can be used.

The Xoff signal can be sent through a 2-bit (8b10b configured) GBT E-link on the FromHost link. The e-link used to send out Xoff is Elink 0 / Egroup 0 of every FromHost GBT link.

To assert flow control, FELIX sends an XOFF (K28.2) K-character on this link when firmware detects an internal FIFOs reaching the almost full state (or if it receives a direct software signal). Upon receipt of XOFF, the front-end should halt data transmission and wait for new signal before resuming transfers.

When the condition is cleared (defined as internal FELIX FIFOs reaching almost empty state, or direct software signal), a XON (K28.3) K-character sent by FELIX to front-end, resuming flow of data.

8.3.16.3.2 CRC

The 32-bit EoP word will contain a 20-bit CRC field for the packet / chunk. The CRC will not be part of the payload transmitted over the PCIe bus to the FELIX server. When a CRC error is detected by the Central Router, a flag will be set in the packet trailer sent to the FELIX server.

During the transmission of a K-character, 24 bits are normally unused, except for the EOP (End of Package) K-Char (K28.6). In Figure 8.47 has been defined that bits 27:8 carry a 20-bit CRC checksum. The TX Stream controller (included in the Full Mode example design provided to the Felix users) calculates this 20-bit CRC checksum and adds it to the EOP field. The FELIX Full mode implementation checks the CRC using the same algorithm and reports a CRC error to the software, by setting the CRC error bit in the trailer.

The CRC module has a data width of 32 bit and a checksum width of 20 bits. The polynomal and initial value have been set to the values below.

- **Polynomal:** 0xC1ACF (alternative notation)
- **Polynomial:** 0x8359F (different endianness, see https://its.cern.ch/jira/browse/FLXUSERS-149 for details)
- **Initial value:** 0xFFFFF

The polynomal has been chosen based on research by Philip Koopman https://users.ece.cmu.edu/ koopman/crc/. With this polynomal a Hamming distance of four can be achieved with a maximum message length of 524267 bits.

The VHDL module to calculate the checksum can be found in the FELIX firmware repository, as well as a C example to calculate the same checksum.

The C module can be found here: crc.c

A highly optimized and generated VHDL version of the CRC20 module which is currently used in the FELIX firmware can be found here: crc.vhd

For future reference, a more descriptive module with the same behavior as crc.vhd, but depending on the vendor / version of the synthesis tool with a wors performance can be found here. crc20.vhd

8.3.16.4 Configuration

The only configuration bit of the FullToAxis entity is the "path_ena" input port, which will be connected to the register DECODING_EGROUP_CTRL[LINK][0].EPATH_ENA[0]. This is the same register that would enable Egroup 0 / Epath 0 on a GBT or lpGBT link.

8.3.16.5 Status Indicators

The status indicators for FullToAxis are only the tuser bits in the axi stream interface. The BUSY / Xoff status bits are reflected in dedicated registers, see also section **??**.

8.3.16.6 Latency

TBD

8.3.16.7 Error Handling

Data errors in the FullToAxis module (Framing error, CRC error, Truncation/FIFO full error, as well as the BUSY status) are reflected by the tuser bits in the AXI4 stream interface. The bits will be interpreted by the CRToHost entity and will then be reflected in the FELIX data format before sent to the host PC. The BUSY bits can also be found in the register map.

8.3.16.8 Estimated Resource Usage

A single FullToAxis entity (including the axi stream FIFO) is reflected in the table below. The numbers are for a single channel.

Resource	Count	% (XKCU115)
LUTs	248	0.037%
Flip-Flops	286	0.021%
Block RAM	3	0.13%

Table 8.15: Resource consumption for the FullToAxis entity.

8.3.16.9 User Example design

The user example design transmits data via the so-called "Full mode stream controller" module, which hides the details of the protocol between the user and FELIX FPGAs, as described below. Figure 8.24 shows a block diagram with the user's data source connected to the to-FELIX Full mode stream controller provided by the FELIX project. The transmission channel line rate is 9.6 Gb/s, whereas user data (payload) has a maximum net rate of 7.68 Gb/s as a result of 8b/10b encoding. The effective bandwidth will be further reduced, depending on the packet lengths, by 4-byte packet headers and trailers, as described in Section 8.3.16.3.

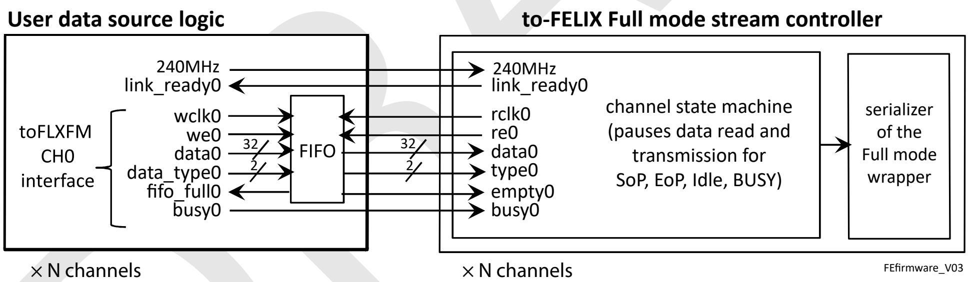


Figure 8.24: block diagram with the user's data source and to-FELIX Full mode stream controller.

In the "to-FELIX Full mode" each link has its independent interface. Each channel in the Full mode stream controller reads data from a dual clock FIFO provided by the user. This allows the user's logic to run with a clock speed different from the 240 MHz required by the transmit logic. The FIFO data width is 32 bits (4 bytes) plus two additional bits (data_type) which qualify the four bytes written. The FIFO implementation (LUT or Block RAM) and depth are chosen by the user. Table 8.16 describes the stream controller's input and output signals.

The first and last word data-type flags result in a word containing a Start-of-Packet (SoP) or End-of-Packet (EoP) K-character to be inserted into the data stream. Refer to section 8.3.16.3 for the K-characters inserted by the stream controller. The FIFO write port is in the user's clock domain, i.e. the write-clock is the user's design clock. Once the channel state machine asserts link_ready, users can directly send data to the FIFOs. Data is written when the WE signal is asserted. For a 240 MHz user clock, if data is written on every clock without pausing between packets, the FIFO will eventually overflow. The user should use the FIFO full signal to prevent this.

The to-FELIX Full mode stream controller will be provided by the FELIX team and integrated into the user's firmware. It is a closed module with the interface described in Table 8.16. The module will be implemented by:

Table 8.16: Description of the stream controller input and output signals.

Signal	Direction	Description
240 MHz clock	from user	clock for the Tx logic; this clock MUST be derived from the BC clock and also used as the GTH reference clock
link_ready	to user	active when link detected and locked
rclk	to user	read clock to read from user's FIFO
re	to user	read enable
data[32]	from user	32-bit wide payload data
data_type[2]	from user	2-bit qualifier for data: 0b01: The word is the first word of a packet. 0b10: The word is the last word of the current packet. 0b00: The word is an intermediate word of the current packet. 0b11: The word is ignored.
fifo_empty	from user	user's FIFO is empty
busy	from user	a level indicating that the user wants FELIX to assert BUSY to the Central Trigger. Minimum duration is two 240 MHz clocks

- a read interface to the user's FIFO running at 240 MHz, reading with maximal data rate of 7.68 Gb/s.
- the serializer part of the Full mode wrapper
- control logic, i.e. a state machine, that inserts defined packet boundary K-characters and busy K-characters into the data stream.

8.3.17 Direct mode E-Link Decoder

8.3.17.1 Introduction

Direct decoding is implemented by omitting the decoder. This is done by connecting ByteToAxiStream directly to the DecodingGearBox, as shown in figure 8.3

Remark 8.3: *Direct mode*

Direct decoding (no decoding) should not be used by any front-end, and is only included for debugging purposes. If no encoding technique is used on top of an E-Link, there is no way for the decoder to distinguish the byte boundary, and where a frame (chunk) starts or ends.

8.3.18 TTCTOHOST VIRTUAL E-LINK

8.3.18.1 INTRODUCTION

The TTC ToHost Virtual E-Link generates a data stream uplon L1A events. This data stream can be subscribed to just like any other E-Link. The data is meant to inform the subscriber about TTC related events and counters, so the event data can be matched to TTC information.

8.3.18.2 INTERFACES

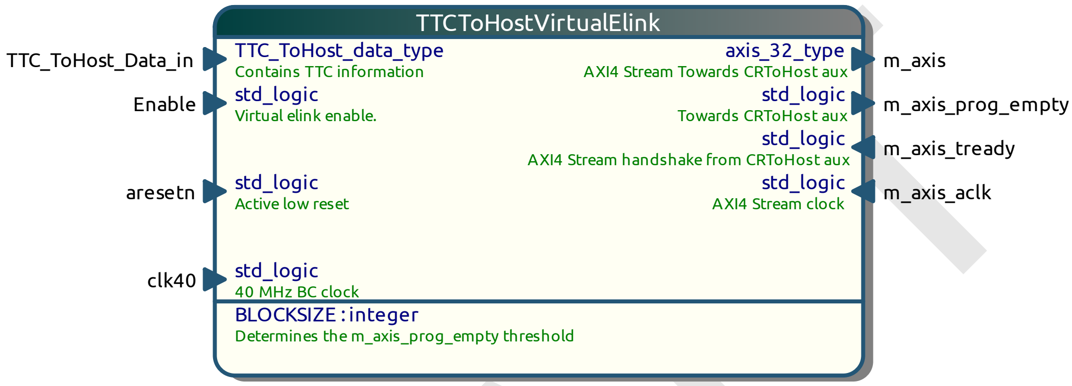


Figure 8.25: The TTC ToHost Virtual E-Link entity.

8.3.18.2.1 GENERICS

- BLOCKSIZE: Used to set the threshold for m_axis_prog_empty to go low if there is at least a block of data in the AXIs FIFO.

8.3.18.2.2 INTERFACE FROM TTC WRAPPER

The TTC Wrapper generates data for the various TTC related signals. On every L1A, the data record as described in Listing 8.1 is generated, and a single pulse on data_rdy is asserted. This record is used by the TTCToHost Virtual E-Link in order to generate a message, to notify a subscriber of the L1A and the corresponding fields.

```
type TTC_ToHost_data_type is record
        FMT           : std_logic_vector(7 downto 0);   --byte0
        LEN           : std_logic_vector(7 downto 0);   --byte1
        reserved0     : std_logic_vector(3 downto 0);   --byte2
        BCID          : std_logic_vector(11 downto 0); --byte2,3
        XL1ID         : std_logic_vector(7 downto 0);   --byte4
        L1ID          : std_logic_vector(23 downto 0); --byte 5,6,7
        orbit         : std_logic_vector(31 downto 0); --byte 8,9,10,11
        trigger_type  : std_logic_vector(15 downto 0); --byte 12,13
        reserved1     : std_logic_vector(15 downto 0); --byte 14,15
        L0ID          : std_logic_vector(31 downto 0); --byte 16,17,18,19
        data_rdy      : std_logic;
end record;
```

Listing 8.1: The TTC_ToHost_data_type as declared in centralRouter_package.vhd.

8.3.18.2.3 Clock, reset and enable

- clk: 40 Mhz bunch crossing clock. It is assumed that all non AXIs related inputs are registered on this clock.
- m_axis_aclk: Clock on which the AXI4 Stream bus is operated towards the CRToHost.
- aresetn: Active low reset.
- Enable: To enable the virtual E-Link. Connected to the Wupper register map.

8.3.18.2.4 Interface to Central Router ToHost

The AXI4 Stream interface consisting of m_axis, m_axis_prog_empty, m_axis_tready and m_axis_aclk holds the data towards CRToHost. CRToHost has a secondary input called s_axis_aux, which will have equal functionality with respect to the regular AXI4 Stream input s_axis, however the dimension is different (Always an array of 2) to connect to the two Virtual E-Links (BusyVirtualElink and TTCToHostVirtualElink).

8.3.18.3 Functional Description

The TTC ToHost Virtual E-Link will be triggered by the data_rdy signal in TTC_ToHost_Data_in input. Upon this trigger, it will create a message containing all the data fields from the input. The last 6 bytes contain a so called L1A counter. This L1A counter will not be reset after an ECR, and can be used as a measure to verify whether any event was lost. Before an ECR, it should hold the same value as L1ID.

The message / chunk is described in Appendix B.2.3.

The length of the message is 26 bytes. When the TTC ToHost virtual e-link is triggered, it immediately constructs the complete message and writes this into a FIFO. This FIFO is read out and the output is converted into AXI4 stream (32b). This dual FIFO mechanism allows the virtual E-Link to be triggered every clock cycle, until the first FIFO is full (Depth=16 messages) without dead time.

8.3.18.4 Configuration

The TTC ToHost virtual E-Link can only be Enabled using the Enable input. No other configuration possibilities are implemented.

8.3.18.5 Status Indicators

This virtual sends data towards CRToHost. No additional status indication is available.

8.3.18.6 Latency

From the first data_rdy input to the end of transmission of the 26-byte AXI4 Stream packet it was measured to take 157 ns. The latency may increase if multiple L1A events are fired shortly after each other and the internal FIFOs fill up.

8.3.18.7 Error Handling

If the FIFOs are full while more busy events occur, the truncation flag in the TUSER bits of the AXI4 stream bus will be asserted.

8.3.18.8 Estimated Resource Usage

Resource	Count	% (XKCU115)
LUTs	329	0.05%
Flip-Flops	651	0.05%

Block RAM	1	0.05%

Table 8.17: TTC ToHost Virtual E-Link Resource utilization.

8.3.19 BUSY virtual E-Link

8.3.19.1 Introduction

The FELIX system knows 4 sources of BUSY:

- E-Link BUSY (BUSY-ON/BUSY-OFF from FrontEnd electronics over E-Links)
- Soft BUSY (Assertion of a register in the register map)
- FIFO busy (The ToHost FIFO in Wupper passed a certain threshold)
- DMA busy (The circular buffer in the server memory is filled beyond a certain threshold)

8.3.19.2 Interfaces

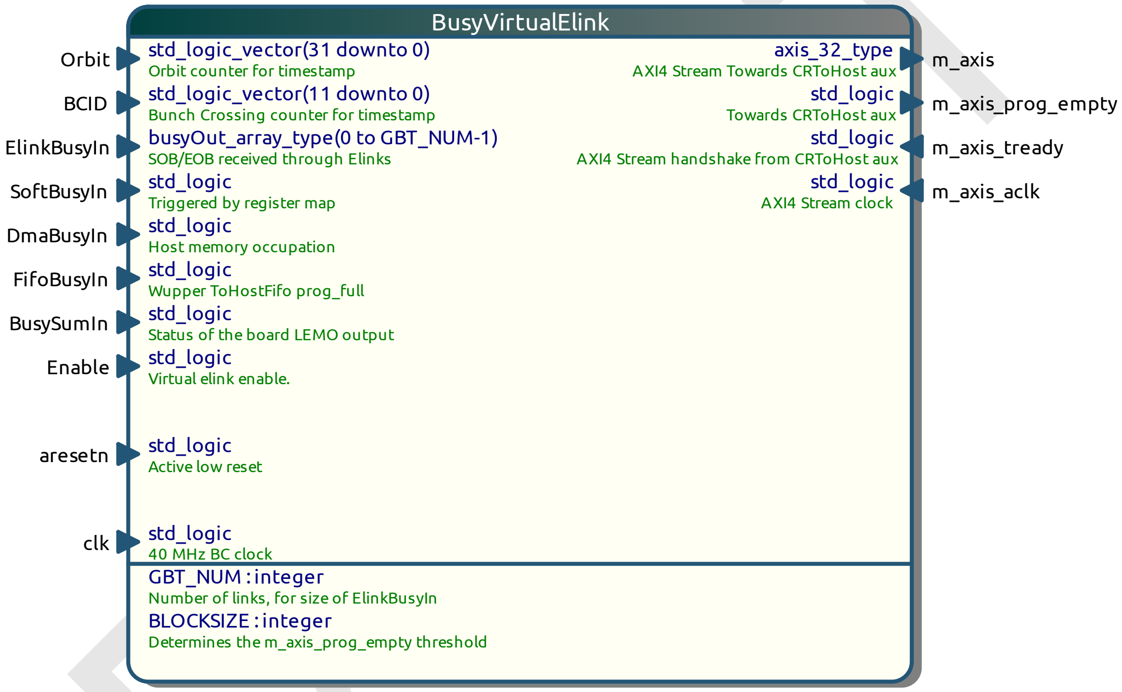


Figure 8.26: The Busy Virtual E-Link entity.

8.3.19.2.1 Generics

- GBT_NUM: Specifies the number of GBT links, to determine the size of the ElinkBusyIn input
- BLOCKSIZE: Used to set the threshold for m_axis_prog_empty to go low if there is at least a block of data in the AXIs FIFO.

8.3.19.2.2 Interface from various BUSY sources

- ElinkBusyIn: A 2-D array of std_logic, each bit representing the BUSY state of the E-Link. The FrontEnd can set this BUSY state by issuing a BUSY-ON/SOB command, and clear it by issuing a BUSY-OFF/EOB command.
- SoftBusyIn: BUSY state triggered by a write to a register

- DmaBusyIn: BUSY asserted because the PC memory (ToHost) was occupied beyond a certain threshold
- FifoBusyIn: FIFO busy asserted, the Wupper ToHost FIFO was occupied beyond a certain threshold

8.3.19.2.3 Timestamp inputs

- Orbit: Orbit counter input from TTC system / Emulator, used as a timestamp in the message.
- BCID: Bunch Crossing counter input from TTC system / Emulator, used as a timestamp in the message.

8.3.19.2.4 Clock, reset and enable

- clk: 40 Mhz bunch crossing clock. It is assumed that all non AXIs related inputs are registered on this clock.
- m_axis_aclk: Clock on which the AXI4 Stream bus is operated towards the CRToHost.
- aresetn: Active low reset.
- Enable: To enable the virtual E-Link. Connected to the Wupper register map.

8.3.19.2.5 Interface to Central Router ToHost

The AXI4 Stream interface consisting of m_axis, m_axis_prog_empty, m_axis_tready and m_axis_aclk holds the data towards CRToHost. CRToHost has a secondary input called s_axis_aux, which will have equal functionality with respect to the regular AXI4 Stream input s_axis, however the dimension is different (Always an array of 2) to connect to the two Virtual E-Links (BusyVirtualElink and TTCToHostVirtualElink).

8.3.19.3 Functional Description

The BUSY Virtual E-Link monitors the status of the 4 sources of busy explained in 8.3.19. Together with the current timestamp (Orbit/BCID) a message will be constructed containing the state of all BUSY sources. This message will be created if BUSY is asserted, but also when it is negated. The message / chunk is described in Appendix B.2.4.

The length of the message is 64 bit. When the BUSY virtual e-link is triggered, it immediately constructs the complete message and writes this into a FIFO. This FIFO is read out and the output is converted into AXI4 stream (32b). This dual FIFO mechanism allows the virtual E-Link to be triggered every clock cycle, until the first FIFO is full (Depth=16 messages) without dead time.

8.3.19.4 Configuration

The BUSY virtual E-Link can only be Enabled using the Enable input. No other configuration possibilities are implemented.

8.3.19.5 Status Indicators

This virtual E-Link is in fact a status indicator of the BUSY system. No additional status indication is available.

8.3.19.6 Latency

From the first busy input to the end of transmission of the 64-bit AXI4 Stream packet it was measured to take 144 ns. The latency may increase if multiple BUSY events are fired shortly after each other and the internal FIFOs fill up.

8.3.19.7 Error Handling

If the FIFOs are full while more busy events occur, the truncation flag in the TUSER bits of the AXI4 stream bus will be asserted.

8.3.19.8 Estimated Resource Usage

Resource	Count	% (XKCU115)
LUTs	313	0.05%
Flip-Flops	436	0.03%
Block RAM	1	0.05%

Table 8.18: Busy Virtual E-Link Resource utilization.

8.3.20 25GbLinksDecoder

8.3.20.1 Introduction

8.3.20.2 Interfaces

8.3.20.2.1 Overview

8.3.20.2.2 Interface to component 2

8.3.20.3 Functional Description

8.3.20.4 Configuration

8.3.20.5 Status Indicators

8.3.20.6 Latency

8.3.20.7 Error Handling

8.3.20.8 Estimated Resource Usage

8.4 Encoding

8.4.1 Introduction

Encoding is the block in the FELIX firmware which instantiates the subdetector specific, but also Atlas wide protocol handling in the FromHost direction (Downstream).

8.4.2 Interfaces

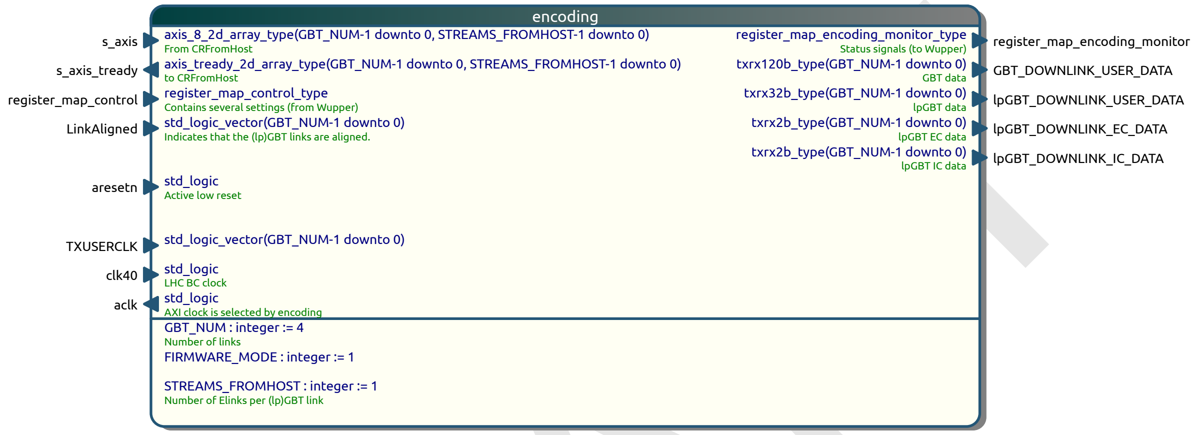


Figure 8.27: The encoding block, instantiating all encoder entities based on FIRMWARE_MODE.

8.4.2.1 Overview

The encoder entity itself does not contain any protocol specific logic, but rather instantiates the protocol specific encoders inside its hierarchy.

The encoder for GBT mode FELIX in phase 2 for instance was derived from the CentralRouter Egroup in phase 1 FELIX. The functionality is the same, but the design will be more modular, and the entities will be more unified among different E-Path / EPROC widths.

Instead of defining a separate entity for every E-link width, as done in phase 1, a configurable and generic gearbox was introduced (see 8.4.9). This gearbox can be configured to support all E-link widths in GBT and lpGBT mode, and output widths for the different protocols (8b10b, direct mode, 6b8b).

The HDLC and 8b10b decoder are very similar to the phase 1 design and can be taken with only slight modification. Finally the GBT mode epath should input the axi stream8 protocol. Therefore the AxiStreamToByte entity was introduced which will take care of the conversion, but also contain the axi stream E-Path FIFO.

8.4.2.2 Interface from CRFromHost

All the protocol encoders that take data from CRFromHost will be equipped with an AXI Stream (8-bit) interface. The encoding entity has an input for a 2-dimensional array of AXI Stream ports, each of them represents a single E-Link. An exception will be made for the 25Gb/s Interlaken links. These Interlaken encoders will need a higher bandwidth which can't be delivered with 8-bit AXI Stream, therefore a 64-bit AXI Stream interface will be used.

8.4.2.3 Interface to LinkWrapper

The outputs towards the optical links are arrays of std_logic_vector, depending on the protocol.

- GBT: GBT_NUM * 120b

- lpGBT: GBT_NUM * (32b(E-Links) + 2b(EC) + 2b(IC))
- Interlaken: GBT_NUM * 76b

8.4.3 Functional Description

Encoding does not contain any functional logic, protocol specific logic is implemented in the instantiated encoders. Depending on the firmware flavour and other generics, a series of if- and for-generate statements determine the content of the encoding block.

8.4.4 Configuration

Configuration registers in register_map_control are routed through encoding into the instantiated encoders.

8.4.5 Status Indicators

Status of the different encoders can be monitored in register_map_encoding_monitor.

8.4.6 Latency

N/A

8.4.7 Error Handling

N/A

8.4.8 Estimated Resource Usage

TBD

8.4.9 Encoding Gearbox

8.4.9.1 Introduction

8.4.9.2 Interfaces

8.4.9.2.1 Overview

8.4.9.2.2 Interface to component 2

8.4.9.3 Functional Description

8.4.9.4 Configuration

8.4.9.5 Status Indicators

8.4.9.6 Latency

8.4.9.7 Error Handling

8.4.9.8 Estimated Resource Usage

8.4.10 Endeavour Encoder

8.4.10.1 Introduction

The Endeavour Encoder is used to encode the commands to send to the AMAC chips: READ register, READ-NEXT register, WRITE register or SETID command (to set the AMAC chip ID). The registers are 32 bit wide and are addressed with an 8 bit number. In the following table are shown the commands bit sequence.

Command	Sequence	Length
READ	"101"[amac ID (5b)][Reg Add (8b)]	16 bit
READNEXT	"100"[amac ID (5b)]	8 bit
WRITE	"111"[amac ID (5b)][Reg Add (8b)][Data (32b)][CRC (8b)]	56 bit
SETID	"110""11111111"[new AMAC ID (5b)]"1111"[efuse ID (20b)]"111"[ID pads (5b)][CRC (8b)]	56 bit

Table 8.20: AMAC commands towards AMAC chip (Encoder).

8.4.10.2 Interfaces

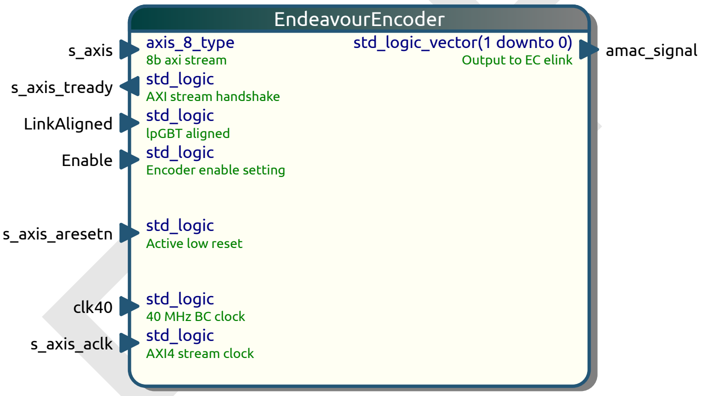


Figure 8.28: The Endeavour encoder entity.

8.4.10.2.1 Overview

The Endeavour Encoder will take input from the FromHost Central Router (CRFromHost) using axi stream 8, and send it's output to an E-Link of the lpGBT frame, usually the EC E-link. Besides the data input and output, there are 2 clock inputs; clk40 which is used for the encoding logic, and s_axis_aclk which serves as a write clock for the axis fifo. The input port s_axis_aresetn serves as an active-low reset for the decoder and the FIFO. Finally there are two input ports: LinkAligned and Enable; LinkAligned indicates that the lpGBT link is aligned and Enable will come from the Wupper register map. Both signals are a condition for the module to be enabled.

8.4.10.2.2 Interface to lpGBT

The two bits of amac_signal are connected to the EC E-link of the lpGBT frame and carry the Endeavour signal. Both bits have the same value.

8.4.10.2.3 Interface to CRFromHost

The input arrive from an AXI Stream 8 bus.

8.4.10.3 Functional Description

The Endeavour Encoder consists of a block that encodes the commands arriving from an AXI Stream fifo of 8 bit using the serial endeavour protocol. The command words are converted using the Endeavour in a serial slow sequence, where the output line is maintained HIGH for a different number of clocks to send 0 or 1 (see table 8.21), sent to a line connected with the AMAC chips. Instead between every bit and after a command the line is maintained LOW. In the following table the number of clocks used for encoding of the words is shown.

Bit	Description
bit 0	N clock HIGH 14
bit 1	N clock HIGH 77
Intra-word bit gap	N clock LOW 43
End of word gap	N clock LOW 100

Table 8.21: Endeavour protocol.

During the encoding it rise up a busy line to prevent the Endeavour Decoder to start the decoding.

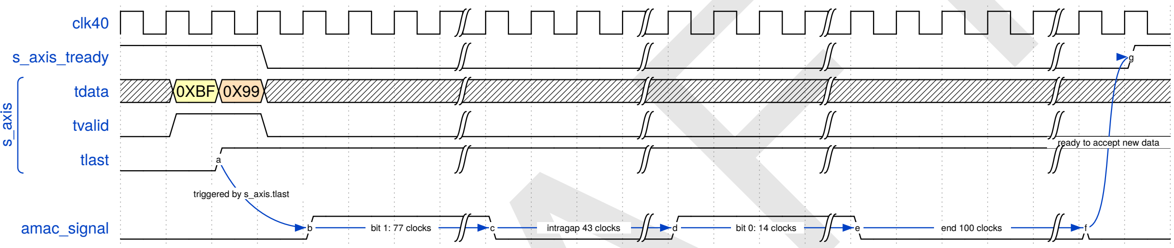


Figure 8.29: example of waveform.

8.4.10.4 Configuration

The Endeavour Encoder can be enabled by setting the Enable input port to '1'. The Encoder will also be disabled if the signal LinkEnabled is low.

8.4.10.5 Status Indicators

There are currently no planned status indicators in the EndeavourEncoder.

8.4.10.6 Latency

8.4.10.7 Error Handling

No errors are implemented.

8.4.10.8 Estimated Resource Usage

Resource	lpGBT link	24 GBT links	% (XKCU115)
LUTs	65	1560	0.2%
Flip-Flops	37	888	0.07%
Block RAM	0.5	12	0.5 %

Table 8.22: Resource consumption of Endeavour Encoder module.

8.4.11 RD53 ENCODER

8.4.11.1 INTRODUCTION

8.4.11.2 INTERFACES

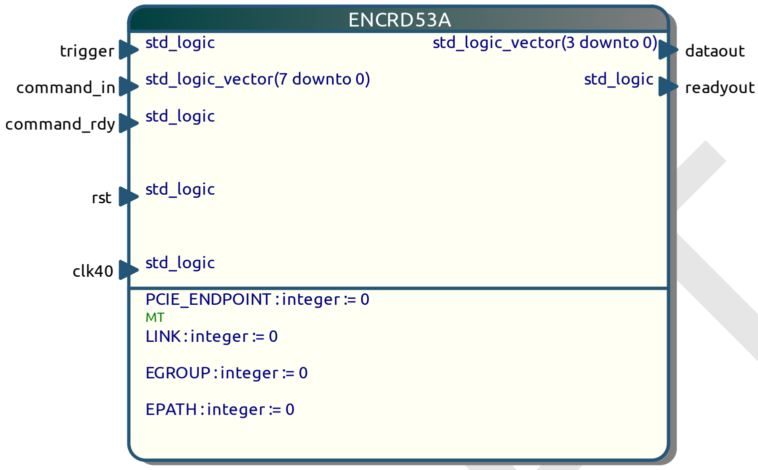


Figure 8.30: The RD53A/B encoder entity.

8.4.11.2.1 OVERVIEW

8.4.11.2.2 INTERFACE TO COMPONENT 2

8.4.11.3 FUNCTIONAL DESCRIPTION

8.4.11.4 CONFIGURATION

8.4.11.5 STATUS INDICATORS

8.4.11.6 LATENCY

8.4.11.7 ERROR HANDLING

8.4.11.8 ESTIMATED RESOURCE USAGE

8.4.12 ITk Strips LCB Encoder

8.4.12.1 Introduction

Strips LCB encoder facilitates control of LCB link of ITk Strips modules. It provides independent control for each Strips link, as well as independent trickle configuration memory storage. The commands are accepted in two formats: compact encoding for efficient storage of trickle configuration, and raw 6b8b user-encoded frames for testing. The LCB encoder merges commands from TTC system, trickle configuration memory and LCB Command elink. Commands originated from TTC system are prioritized and have fixed latency. Strips LCB encoder may be configured to send low-priority commands only within a configurable BC interval, for example during a beam gap.

The functional diagram of LCB encoder module is presented on Figure 8.31. The blocks shown in red are data inputs. The corresponding FromHost elink IDs are listed in Table 8.23).

The LCB encoder inputs data from the TTC system and three FELIX FromHost elinks. The LCB Configuration elink is used to configure the LCB encoder. The data sent to the Trickle Configuration elink is written into the trickle configuration memory. Finally, the Command elink sends commands to the LCB input of a Strips module (shown in green).

Elink hex	Elink dec	Strips Encoder
00	0	LCB#0 configuration
01	1	LCB#0 command
02	2	LCB#0 trickle
03	3	R3L1#0 configuration
04	4	R3L1#0 command
05	5	LCB#1 configuration
06	6	LCB#1 command
07	7	LCB#1 trickle
08	8	R3L1#1 config
09	9	R3L1#1 command
0a	10	LCB#2 config
0b	11	LCB#2 command
0c	12	LCB#2 trickle
0d	13	R3L1#2 config
0e	14	R3L1#2 command
0f	15	LCB#3 configuration
10	16	LCB#3 command
11	17	LCB#3 trickle
12	18	R3L1#3 config
13	19	R3L1#3 command
14	20	EC (AMAC out)
15	21	IC

Table 8.23: Strips ToHost elilnk mapping. In this table, elink mapping of lpGBT optical link 0 is listed. To find elink IDs for encoders of another optical link, add `0x40` * (lpGBT link ID) to the elink IDs listed in the table..

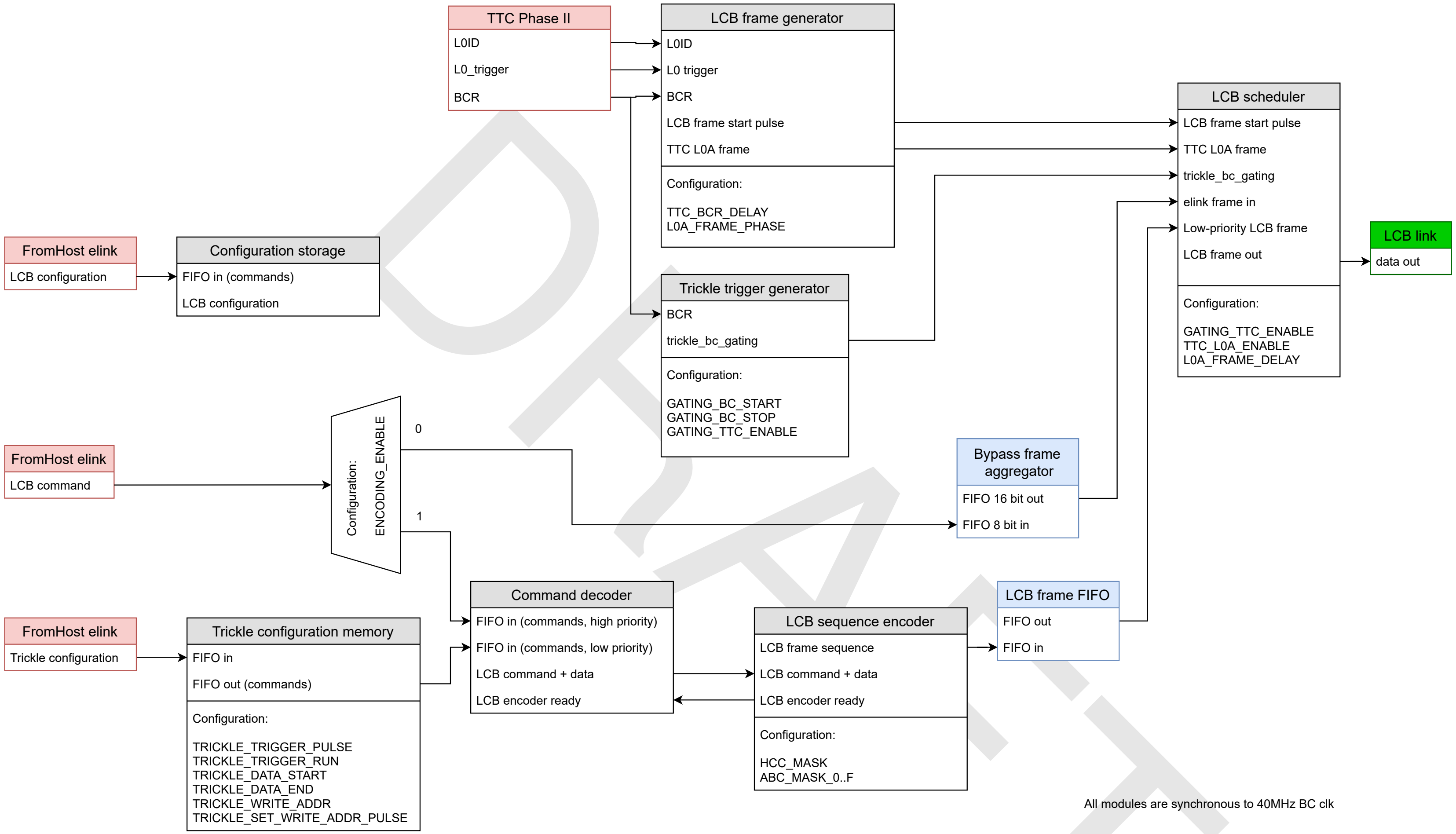


Figure 8.31: Functional diagram of ITk Strips LCB Encoder module.

Byte \ Bit	7	6	5	4	3	2	1	0
0	0x10							
1	Data [15:8]							
2	Data [7:0]							
3	Register address							

Figure 8.32: LCB link configuration command format.

8.4.12.2 Configuration storage submodule

This submodule stores and updates the LCB link configuration registers. Please note that these registers are separate and independent from the FELIX register map. In the default configuration after FELIX power-on all registers are set to zero. The module can be returned to the default configuration at any time by disabling and re-enabling the LCB configuration elink. The configuration registers can be updated by issuing the “configure” command via the LCB configuration elink. This is the only valid command for the configuration elink.

8.4.12.2.1 Configuration command. Configuration commands update LCB configuration registers given the data and the register address. (Fig. 8.32). The configuration registers of the LCB encoder are listed in the Table 8.24. Please note that although the data width in the “configure” command is always 16 bits, many configuration registers only use a few least significant bits (indicated by the #bits column in Table 8.24).

Whenever a configuration register is mentioned in this section, it refers to the local LCB encoder configuration storage register, unless explicitly specified as a FELIX register.

8.4.12.3 LCB frame generator submodule

This submodule determines the phase of LCB frame and generates L0A frames in response to the signals from the TTC module. The exact contents of L0A frame depend on configurable LCB frame phase and the timing of L0 triggers.

The phase of LCB frame with respect to BCR signal is configurable via `L0A_FRAME_PHASE` register. The frame phase is locked to TTC BCR signal in order to facilitate synchronization of all ITk Strips links. Independently of the frame phase, the module can also add configurable delay to the BCR signal via adjusting `TTC_BCR_DELAY` register. This setting only affects L0A frame generation, and does not influence other functions dependent on BCR signal, such as frame phase or trickle triggering. BCR delay must always be smaller than BCR period for the module to function correctly.

Remark 8.4: *Adjusting LCB frame phase*

Adjusting LCB frame phase while the link is active will result in data corruption and decoding errors on the front-end side. LCB frame phase should not be adjusted during active data taking and command transmission.

8.4.12.4 Bypass frame aggregator submodule

Bypass frame aggregator forms 16-bit LCB frames from Command elink data and forwards them to frame scheduler when `ENCODING_ENABLE=0` (default). Since bypass frames are not processed by the encoder logic, it is user’s responsibility to ensure that the frame sequence is complete and encoded in 6b8b. The odd-count elink bytes becomes MSB, and even-count bytes become LSB of the LCB frames. Bypass frames are treated as low-priority frames by the frame scheduler.

Address	Name	#bits	Description
0x00	L0A_FRAME_PHASE	2	Determines LCB frame phase with respect to the TTC BCR signal
0x01	L0A_FRAME_DELAY	4	Determines the overall delay of L0A frame in BC units. Only L0A frames originated from TTC system are delayed. Will affect LCB frame phase when it's not a multiple of 4.
0x02	TTC_L0A_ENABLE	1	Enables generation of L0A frames in response to the TTC signals
0x03	TTC_BCR_DELAY	12	Delay BCR signal from TTC system by this many BC units before issuing L0A
0x04	GATING_TTC_ENABLE	1	When set to 1, the low-priority frames are only allowed during the interval between GATING_BC_START and GATING_BC_STOP (counted from non-delayed TTC BCR signal)
0x05	GATING_BC_START	12	Start of BC gating interval
0x06	GATING_BC_END	12	End of BC gating interval
0x07	TRICKLE_TRIGGER_PULSE	1	Write 1 to issue a single trickle trigger
0x08	TRICKLE_TRIGGER_RUN	1	Write 1 to issue trickle trigger continuously
0x09	TRICKLE_DATA_START	16	Address of the first valid byte in the trickle configuration memory
0x0A	TRICKLE_DATA_END	16	Address of the last valid byte in the trickle configuration memory
0x0B	TRICKLE_WRITE_ADDR	16	Trickle configuration memory write pointer
0x0C	TRICKLE_SET_WR_ADDR_PULSE	1	Write 1 to move the trickle configuration write pointer to the address in TRICKLE_WRITE_ADDR
0x0D	ENCODING_ENABLE	1	When 0, the data sent into LCB command elink is forwarded to LCB line without processing. It is the user's responsibility to ensure the commands are formed correctly and encoded in 6b8b. When 1, the commands are interpreted as described in Section 8.4.12.6.
0x0E	HCC_MASK	16	HCC* command mask. When a bit is set to 1, commands to this HCC* chip (and all connected ABC* chips) will be ignored. Will match broadcasts. LSB corresponds to HCC* address 0, MSB corresponds to HCC* address 0xF.
0x0F	ABC_MASK_0	16	ABC* command mask for chips connected to HCC* with address 0. LSB corresponds to ABC* address 0, MSB corresponds to ABC* address 0xF. When a bit is set to 1, commands to this ABC* chip will be ignored. Will match broadcasts.
0x10	ABC_MASK_1	16	ABC* command mask for chips connected to HCC* with address 1.
...	...	...	...
0x1E	ABC_MASK_F	16	ABC* command mask for chips connected to HCC* with address 0xF.

Table 8.24: LCB link configuration registers.

Byte \ Bit	7	6	5	4	3	2	1	0
0	0x00							

Figure 8.33: No operation command format.

> **Remark 8.5:** *Zero bytes*
>
> Bypass frame aggregator removes all zero bytes from the data stream.

8.4.12.5 Trickle configuration memory

Each LCB link has an independent memory storage for trickle configuration. The memory is byte-addressable and has the total size of 64 kB per LCB encoder. Other data, such as calibration sequence, may also be stored in trickle configuration memory.

Trickle configuration memory can store multiple sequences provided there is sufficient space. The sequence selected for readout is determined by memory pointers defined in the configuration registers `TRICKLE_DATA_START` and `TRICKLE_DATA_END`.

Trickle configuration memory can be triggered from software. To issue a single software trickle trigger, write '1' to register `TRICKLE_TRIG_PULSE`. To send trickle configuration continuously, write '1' to `TRICKLE_TRIG_RUN`. If synchronization between multiple LCB links is required, software trickle trigger pulse can be issued simultaneously for all links by writing '1' to FELIX register `GLOBAL_STRIPS_CONFIG.TRICKLE_TRIG_PULSE`.

Trickle configuration memory can only be written when trickle configuration readout is inactive. This requires that `TRICKLE_TRIG_RUN` is set to '0', and any preceding trickle configuration readout has completed. Before updating the memory, set `TRICKLE_WRITE_ADDR` to the memory address where the configuration is to be stored and write '1' to `TRICKLE_SET_WRITE_ADDR_PULSE` to move the write pointer there. Send the data into the Trickle Configuration elink to write it into the trickle configuration memory. As the data is written into the elink, the memory write pointer will automatically advance. The trickle configuration commands must be in the format compatible with the command decoder (see Section 8.4.12.6 below). No bypass frames may be stored in trickle configuration memory.

For the data taking with hardware triggering, the LCB link can be configured to only send trickle configuration during a beam gap. See the description of Trickle Trigger Generator and LCB Scheduler modules for more detail, and see Section 8.4.12.11 for the setup procedure.

> **Remark 8.6:** *Time-critical command sequences read out from trickle memory*
>
> For certain command sequences, such as L0A followed immediately by fast command, command decoder might be unable to encode the LCB frames in time, and IDLE frames are inserted in between. This disrupts calibration sequences that require predictable timing between command frames. The workaround for sending time-critical command sequences is provided in section 8.4.12.11.

8.4.12.6 Command decoder

This module decodes commands originating from the trickle configuration memory and LCB Command elink (only when `ENCODING_ENABLE=1`). The commands from the two sources are merged into a single low-priority frame queue. Command decoder always processes LCB Command elink commands first when both sources have data. Below is the list of valid commands and their format.

8.4.12.6.1 No operation. This command is ignored by the command decoder. It is added for compatibility with phase1 firmware and to prevent frame generation from uninitialized trickle configuration memory. The format of no operation command is shown on Fig. 8.33.

Byte \ Bit	7	6	5	4	3	2	1	0
0	0x80							

Figure 8.34: IDLE command format.

Byte \ Bit	7	6	5	4	3	2	1	0
0	0x82							
1	0	0	0	BCR	mask			

Figure 8.35: L0A command format.

8.4.12.6.2 IDLE COMMAND. Places a single IDLE frame into the LCB link queue (Fig. 8.34). IDLE can be written into trickle configuration memory as a part of the calibration sequence to add 100 ns delay between commands.

8.4.12.6.3 L0A COMMAND. This issues a user-defined L0A frame to the front-end (Fig. 8.35). At least a single bit in mask or BCR must be set to ’1’ for the command to be valid. Invalid L0A commands are ignored by the command decoder.

8.4.12.6.4 FAST COMMAND. Fast command sends a user-defined fast command to the front-end (Fig. 8.36).

8.4.12.6.5 REGISTER COMMANDS. Register commands issue a read (Fig. 8.37) or write (Fig. 8.38) frame sequence for HCC* or ABC* register.

8.4.12.7 LCB SEQUENCE ENCODER

This module generates single or multiple low-priority LCB frames as requested by the command decoder. Generating register commands addressed to certain chips may be blocked by this module using HCC ID and ABC ID masking. This is achieved by writing configuration registers `HCC_MASK` and `ABC_MASK_X`. When a bit in the mask is set to ’1’, register commands for the corresponding chip will be ignored by the module. This can be used to quickly disable configuration for selected chips without overwriting trickle configuration memory.

8.4.12.8 LCB FRAME FIFO

This FIFO stores contents of low-priority LCB frames, originated from elink or trickle configuration memory. Default FIFO depth is 64 frames.

8.4.12.9 TRICKLE TRIGGER GENERATOR

This module controls timing of sending trickle configuration commands to the front-end during the data taking. When enabled by setting `GATING_TTC_ENABLE` to ’1’, low-priority frames are only allowed during BC gating

Byte \ Bit	7	6	5	4	3	2	1	0
0	0x81							
1	0	0	BC select		Command ID			

Figure 8.36: Fast command format.

Byte \ Bit	7	6	5	4	3	2	1	0
0	0xA0 (ABC*) or 0xA1 (HCC*)							
1	Register address							
2	HCC ID				ABC ID			

Figure 8.37: Register read command format.

Byte \ Bit	7	6	5	4	3	2	1	0
0	0xA2 (ABC*) or 0xA3 (HCC*)							
1	Data [31:24]							
2	Data [23:16]							
3	Data [15:8]							
4	Data [7:0]							
5	Register address							
6	HCC ID				ABC ID			

Figure 8.38: Register write command format.

interval between `GATING_BC_START` and `GATING_BC_STOP`. Low-priority frames are defined as frames that did not originate from the TTC system. Please note that trickle configuration readout must be enabled by setting `TRICKLE_TRIG_RUN` to '1' in addition to setting `GATING_TTC_ENABLE` to '1'.

This module also defines the guard interval for register commands, which begins 64 BC periods before `GATING_BC_STOP`. This ensures that register commands are transmitted completely before the BC gating interval ends. During the guard interval any active register command is allowed to complete, but no new register commands are allowed to begin.

Remark 8.7: *BC gating and stuck elinks*

Please note that when the BC gating is enabled the encoder module may not process LCB Command elink commands unless it receives periodic BCR signal from the TTC system and BC gating interval is correctly configured. BC gating signal will not be generated if `BC_START=BC_STOP`. BC gating signal will be generated incorrectly if `BC_START>BC_STOP`.

Remark 8.8: *BC gating and the guard interval*

Please note that BC gating interval duration must be at least equal to the guard interval size + 5 (69 BC) for the module to function correctly. When this condition is violated, register commands may be either transmitted partially, or not transmitted at all.

8.4.12.10 LCB SCHEDULER

This module prioritized LCB commands according to their source, merges them into a single data stream, and sends them to the front end. The module encodes LCB frames in 6b8b as needed, and may be configured to add a variable overall time delay to the LCB frame, as defined by `L0A_FRAME_DELAY` in BC period units.

> **Remark 8.9:** *Adjusting LCB frame delay*
>
> Adjusting the overall frame delay will result in loss or corruption of LCB frames and decoding errors on the front-end side. LCB frame delay should not be adjusted during active data taking and command transmission. Adding a delay will change the phase of the LCB frame, meaning that frame phases on different links may not match if they are configured with the same phase, but different frame delays.

Overview of the scheduling algorithm:

1. Send TTC L0A frame if available
2. Else send bypass frame if available and no register command from LCB frame FIFO is in progress
3. Else send next frame from LCB frame FIFO
4. Else send an IDLE frame

If BC gating is enabled (`GATING_TTC_EN` is set to 1), low-priority frames will only be sent during the BC gating interval. TTC L0A frames are always sent regardless of the BC gating configuration, provided `TTC_L0A_ENABLE=1`.

8.4.12.11 EXAMPLES

8.4.12.11.1 SENDING BASIC LCB COMMANDS VIA LCB COMMAND ELINK AND COMMAND DECODER (`ENCODING_ENABLE=1`)

- Fast command example (command=6, BC=3): 0x81 0x36
- L0A command example (BCR=1, mask=0x3, tag=0x53): 0x82 0x13 0x53
- ABC* register read example (register 0x12, HCC ID=0xA, ABC ID=F): 0xA0 0x12 0xAF
- HCC* register read example (register 0x42, HCC ID=0x7, ABC ID=0): 0xA1 0x42 0x70
- ABC* register write example (write 0xDEADBEEF to register 0x12, HCC ID=0xA, ABC ID=0xF): 0xA2 0xDE 0xAD 0xBE 0xEF 0x12 0xAF
- HCC* register write example (write 0xBABEABBA to register 0x42, HCC ID=7, ABC ID=0): 0xA3 0xBA 0xBE 0xAB 0xBA 0x42 0x70

8.4.12.11.2 SENDING BASIC LCB COMMANDS VIA LCB COMMAND ELINK AND BYPASS FRAME AGGREGATOR (`ENCODING_ENABLE=0`)

To send the commands directly to Strips LCB input, send commands to the Bypass elink. The bypass commands are not verified or processed in any way. Register commands send through bypass elink are not filtered based on `HCC_MASK` or `ABC_MASK_X`. The bypass register commands can be merged correctly with trickle configuration commands, as long as complete register commands arrive in a single chunk to the Bypass elink.

- Fast command example (command=0xB, BC=3): 0x6A 0x5A
- L0A command example (BCR=1, mask=0xD, tag=0x38): 0x3A 0xB8

- ABC* register read example (register 0x38, HCC ID=0xC, ABC ID=0xD) 0x47 0x3C 0x71 0xB4 0x71 0x74 0x47 0xAC
- HCC* register read example (register 0xD6, HCC ID=0x5, ABC ID=0xB): 0x47 0x95 0x71 0x6C 0x59 0xAC 0x47 0xC5
- ABC* register write example register write example (write 0x1021ABD2 to register 0x5E, HCC ID=0x5, ABC ID=0xC): 0x47 0x35 0x59 0xB1 0x59 0x3C 0x59 0x71 0x59 0x71 0x59 0xC6 0x71 0x17 0x71 0xD2 0x47 0xA5
- HCC* register write example (write 0x8A37DF3C to register 0x2B, HCC ID=0xF, ABC ID=0xB): 0x47 0x5C 0x59 0xAC 0x71 0x96 0x59 0x69 0x71 0xD1 0x71 0x5C 0x59 0x4E 0x59 0x3C 0x47 0x4B

8.4.12.11.3 WRITING TRICKLE CONFIGURATION

1. Set `TRICKLE_WRITE_ADDR`=0
2. Set `TRICKLE_SET_WRITE_ADDR_PULSE`=1
3. Set `TRICKLE_DATA_START`=0, and set `TRICKLE_DATA_END` equal to the length of the trickle configuration in bytes
4. Write trickle configuration to the Trickle Configuration elink. All commands must be in the format described in Section 8.4.12.6

8.4.12.11.4 ISSUING SOFTWARE-GENERATED TRICKLE TRIGGER

8.4.12.11.1 SINGLE LCB ELINK. Issue trickle trigger to a single elink by writing '1' to `TRICKLE_TRIGGER_PULSE` configuration register.

8.4.12.11.2 CONTINUOUS TRICKLE CONFIGURATION. Send trickle configuration continuously by writing '1' to `TRICKLE_TRIGGER_RUN` configuration register.

8.4.12.11.3 ALL LCB ELINKS SIMULTANEOUSLY. Issue trickle trigger to LCB encoder elink by writing '1' to FELIX register `GLOBAL_STRIPS_CONFIG.TRICKLE_TRIG_PULSE`.

8.4.12.11.4 ALL LCB ELINKS SIMULTANEOUSLY WITH PRE-BUFFERING.

1. Write 0 to `GATING_BC_START` and `GATING_BC_STOP` of each elink to be triggered
2. Write '1' to FELIX register `GLOBAL_STRIPS_CONFIG.TTC_GENERATE_GATING_ENABLE`
3. Write '1' to FELIX register `GLOBAL_STRIPS_CONFIG.TRICKLE_TRIG_PULSE`
4. Wait a few milliseconds for the encoded frames to buffer
5. Write '0' to FELIX register `GLOBAL_STRIPS_CONFIG.TTC_GENERATE_GATING_ENABLE` to issue the commands

8.4.12.11.5 TRICKLE TRIGGER DURING SPECIFIED BC INTERVAL

1. Write the first BCID of the allowed interval into `GATING_BC_START`
2. Write the last BCID of the allowed interval into `GATING_BC_STOP`
3. Write '1' to `GATING_TTC_ENABLE` to enable BC gating
4. Write '1' to `TRICKLE_TRIGGER_RUN` to start trickle configuration

8.4.12.12 Latency

- TTC: Fixed 10 BC latency
- Bypass: Fixed TBD BC latency (when not pre-empted by TTC)
- Elink decoder: Variable 16–20 BC latency (when not pre-empted)
- Trickle: ~36 BC after readout enabled (empty LCB frame buffer, `GATING_TTC_ENABLE=0`)

8.4.12.13 Estimated Resource Usage

Resource	E-Group	lpGBT link	24 lpGBT links	% (XKCU115)
LUTs	475	1900	45600	7%
Flip-Flops	837	3348	80352	6%
Block RAM	17.5	70	1680	78%

Table 8.25: Resource consumption of LCB encoder module.

8.4.13 ITk Strips R3L1 Encoder

8.4.13.1 Introduction

Strips R3L1 encoder facilitates control of R3L1 link of ITk Strips modules. The module processes R3 and L1 hardware triggers, and inserts user-encoded R3L1 frames into the data stream.

The functional diagram of R3L1 encoder module is presented on Figure 8.39.

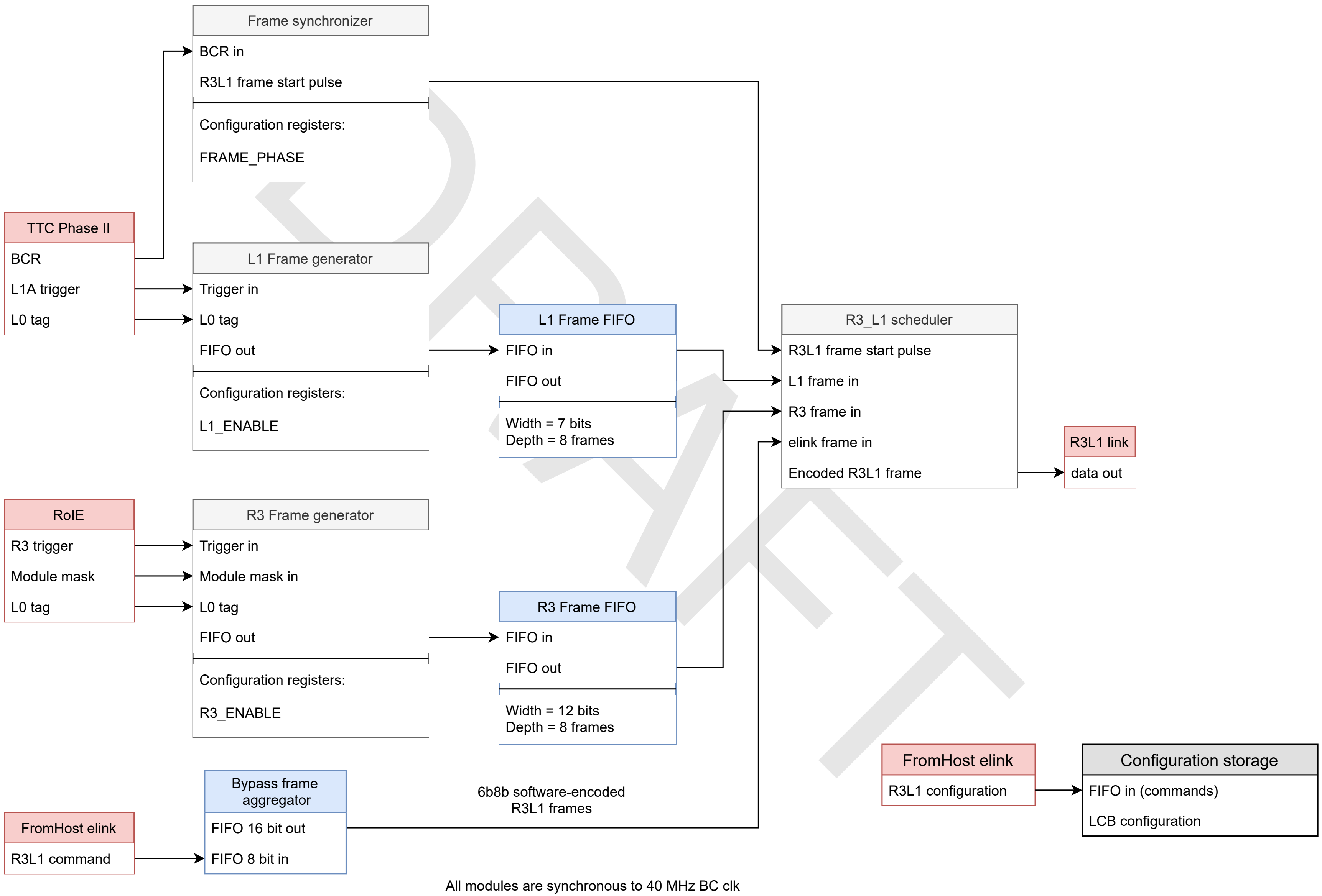


Figure 8.39: Functional diagram of ITk Strips R3L1 Encoder module.

Byte \ Bit	7	6	5	4	3	2	1	0
0	0x10							
1	Data [15:8]							
2	Data [7:0]							
3	Register address							

Figure 8.40: R3L1 link configuration command format.

Address	Name	#bits	Description
0x00	FRAME_PHASE	2	Determines R3L1 frame phase with respect to the TTC BCR signal
0x01	L1_ENABLE	1	Allows processing of L1 signals from the TTC system
0x02	R3_ENABLE	1	Allows processing of R3 signals from the TTC system

Table 8.26: R3L1 link configuration registers.

8.4.13.2 Configuration storage submodule

This submodule stores and updates the R3L1 link configuration registers. Please note that these registers are separate and independent from the FELIX register map. In the default configuration after FELIX power-on all registers are set to zero. The module can be returned to the default configuration at any time by disabling and re-enabling the R3L1 configuration elink. The configuration registers can be updated by issuing the "configure" command via the R3L1 configuration elink. This is the only valid command for the configuration elink.

8.4.13.2.1 Configuration command. Configuration commands update R3L1 configuration registers given the data and the register address. (Fig. 8.40). The corresponding FromHost elink IDs are listed in Table 8.23). The configuration registers of the R3L1 encoder are listed in the Table 8.26. Please note that although the data width in the "configure" command is always 16 bits, many configuration registers only use a few least significant bits (indicated by the #bits column in Table 8.26).

Whenever a configuration register is mentioned in this section, it refers to the local R3L1 encoder configuration storage register, unless explicitly specified as a FELIX register.

8.4.13.3 Frame synchronizer

This submodule determines the phase of R3L1 frame, which is configurable via `FRAME_PHASE` register. The frame phase is locked to TTC BCR signal in order to facilitate synchronization of all ITk Strips links.

8.4.13.4 R3 and L1 Frame generators

R3 and L1 Frame modules generate R3 and L1 frames in response to the corresponding hardware signals. Generation of either frame must be enabled by setting registers `L1_ENABLE` or `R3_ENABLE` to '1'.

8.4.13.5 R3 and L1 Frame FIFOs

These FIFOs stores contents or either R3 or L1 frames. Default FIFO depth is 16 frames.

8.4.13.6 Bypass frame aggregator

Bypass frame aggregator forms 16-bit R3L1 frames from 8-bit elink data and forwards them to frame scheduler. Since bypass frames are not processed by the encoder logic, it is user's responsibility to ensure that the frames are valid 6b8b encoded data. The odd-count elink bytes becomes MSB, and even-count bytes become LSB of R3L1 frames. For the purpose of backwards compatibility with phase1 firmware, bypass frame aggregator removes zero bytes from the data stream.

Remark 8.10: *Zero bytes*

Bypass frame aggregator removes all zero bytes from the data stream.

8.4.13.7 R3L1 Scheduler

This module prioritized R3L1 commands according to their source, merges them into a single data stream, and sends them to the front end. The module encodes R3L1 frames into 6b8b as needed.

Overview of the scheduling algorithm:

1. Send R3 frame if available
2. Else send L1 frame if available
3. Else send bypass frame
4. Else send an IDLE frame

8.4.13.8 Latency

- R3: fixed 13 BC latency
- L1: fixed 13 BC latency (when not pre-empted by R3)
- Bypass: Fixed 10 BC latency (when not pre-empted by R3 or L1)

8.4.13.9 Estimated Resource Usage

Resource	E-Group	lpGBT link	24 GBT links	% (XKCU115)
LUTs	141	564	13536	2%
Flip-Flops	261	1044	25056	2%
Block RAM	1	4	96	4%

Table 8.27: Resource consumption of R3L1 encoder module.

8.4.14 8b10b Encoder

8.4.14.1 Introduction

he 8b10b Encoder has been extensively used in phase 1 FELIX in GBT mode. In Phase II, the 8b10b encoder has been decoupled from the E-proc, and retains in the generic E-Path in GBT and lpGBT mode firmware flavours.

The tasks for the 8b10b encoder are:

- Encode the 8b10b stream to 8-bits + CharIsK
- Assertion of E-link BUSY in case of Xoff
- Framing: Convert DataIn, DataInValid and EOP into Encoded byte + CharIsK

8.4.14.2 Interfaces

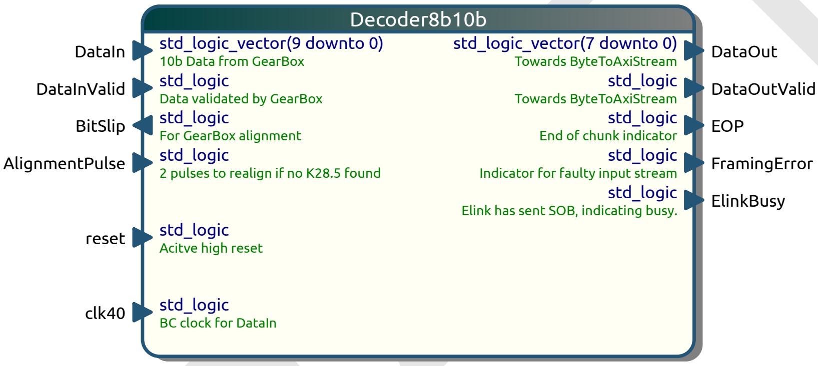


Figure 8.41: The 8b10b Encoder entity.

8.4.14.2.1 Interface to AxiStreamToByte

The 8b10b encoder receives from the AxiStreamToByte:

- DataIn[7:0]: payload data;
- DataInValid: indicates that payload data are valid;
- EOP_in: End Of Packet indicator.

The encoder sends towards the AxiStreamToByte:

- readyOut: flag signaling that the encoder is ready to accept new data; it corresponds to the *m_axis tready* flag of the AxiStreamToByte.

8.4.14.2.2 Interface to EncodingGearBox

The 8b10b encoder receives from the EncodingGearBox:

- readyIn : indicates that the GearBox is ready to accept new data from the encoder

The encoder sends towards the EncodingGearBox

- DataOut[9:0] : 8b/10b encoded data (always valid)

8.4.14.3 Functional Description

8.4.14.3.1 Overview

The 8b/10b encoder encodes idles/payload data into 8b/10b protocol. Payload data are transmitted through packets; each packet starts with a SOP (Start of Packet) comma character and ends with a EOP (End of Packet) comma character (refer to table 8.28). A minimum of two idle comma characters is sent after EOP.

In case of Xoff rising edge, the encoder transmits a SOB (Start of Busy) comma, and a EOB (End Of Busy) comma is sent in case of Xoff falling edge. When special comma characters, such as SOP, EOP, or during Xoff, the encoder stops the AxiStram fifo from sending payload data by asserting a low readyOut signal.

8.4.14.3.2 8b10b encoding

Comma characters:

Function	GBT mode	Strip/LCB	FEI4	Meaning
Comma	K28.5	K28.1	K28.1	Idle character
SOP	K28.1	K28.7	K28.7	Start of chunk / packet
EOP	K28.6	K28.5	K28.5	End of chunk / packet
SOB	K28.2	N/A	N/A	Start of busy
EOB	K28.3	N/A	N/A	End of busy

Table 8.28: Comma characters with a special meaning in different firmware flavours.

The functional description of the 8b10b encoder itself, converting a 10b word into 8 bit + CharIsK is well defined in other literature, and the code has been implemented in phase 1 FELIX.

8.4.14.4 Configuration

The meaning of the different comma characters in table 8.28 can be configured based on the FIRMWARE_-MODE generic at build time. It is not foreseen at the moment to make a runtime configurable option for the 8b10b encoder.

8.4.14.5 Latency

The 8b10b encoder has a latency of 1 or 2 clock cycles (25 ns). However, it must be taken into consideration that the *readyIn* signal from the GearBox plays a crucial role into determining the actual latency of the encoding block.

8.4.14.6 Error Handling

There is no error handling within the 8b/10b Encoder block. All payload data from the AxiStreamToByte are considered valid.

8.4.14.7 Estimated Resource Usage

The resource usage will be estimated for the complete GBT Egroup and the complete encoding block per firmware mode.

8.4.15 HDLC Encoder

8.4.15.1 Introduction

The HDLC Protocol [9] is used by the GBTx chip, to configure the chip itself through the Internal Control (IC) E-link, and to communicate with the GBT Slow Control Adaptor (GBT-SCA) over the External Control (EC) E-Link or any other 80 Mb/s E-link of the GBT or lpGBT.

8.4.15.2 Interfaces

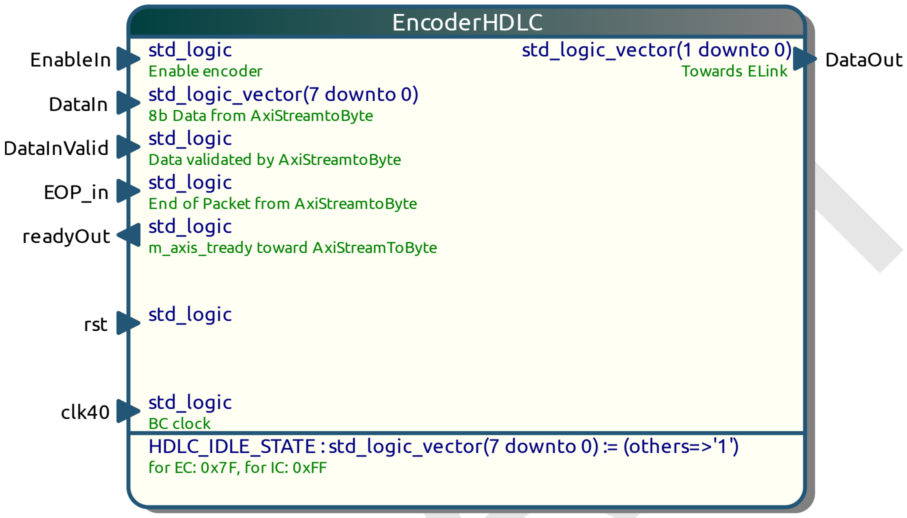


Figure 8.42: The HDLC encoder entity.

8.4.15.2.1 Generics

- HDLC_IDLE_STATE: The byte that is clocked out on IDLE, for IC E-Links this is set to 0xFF, for EC (GBT-SCA) E-Links this should be set to 0x7F.

8.4.15.2.2 Interface from AxiStreamToByte

The signals that connect the HDLCEncoder to AxiStreamToByte directly translate to AXI Stream signals, however multiple encoders (8b10b, direct) are implemented within one E-Path, so there may be connection logic in between ByteToAxiStream and EncoderHDLC.

- DataIn: Carries a data byte. Equivalent to s_axis_tdata.
- DataInValid: Marks that DataIn is valid. Equivalent to s_axis_tvalid.
- EOP_in: Marks the last data byte of a chunk. Equivalent to s_axis_tlast.
- readyOut: Encoder is ready to accept the next data byte. Equivalent to s_axis_tready.

8.4.15.2.3 Interface to GBT/lpGBT E-Link

The 2-bit port DataOut can be directly connected to the 2 bits of an EC or IC E-Link of the GBT or lpGBT frame, it bypasses the EncodingGearBox because only 2-bit E-Links are supported for HDLC. The Encoding Epath may contain additional multiplexing logic depending on the configuration.

8.4.15.3 Functional Description

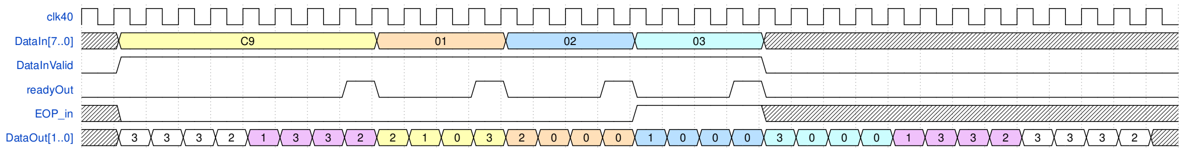


Figure 8.43: The HDLC encoder waveform.

The HDLC decoder is a shift register that shifts out 2 bits at a time. Data is sent out LSB first, for the E-Link bits (DataOut) the LSB is transmitted at bit 1, bit 0 is the second bit. The serializer process has a bitstuffing functionality, if 5 consecutive ones are detected, a '0' inserted into the output data. Before and after a data frame, a FLAG is inserted. On IDLE, the ERROR flag is sent out.

Transmitting the first byte takes two times the time of the next bytes, because the FLAG needs to be sent out first before readyOut can be asserted. See for the timing diagram Figure 8.43.

8.4.15.4 Configuration

The only configuration possible is to enable the entity by setting EnableIn.

8.4.15.5 Status Indicators

The HDLC Encoder has no status indicators.

8.4.15.6 Latency

A byte takes 4 clockcycles to send out. The first FLAG is shipped out the clock cycle after DataInValid is asserted, however it will take 8 clock cycles to clock out this first byte.

8.4.15.7 Error Handling

There is no error handling built into the HDLC Encoder.

8.4.15.8 Estimated Resource Usage

The resource usage of the complete Encoding Epath for GBT will be covered in section 8.4.

8.4.16 Direct mode E-Link Encoder

8.4.16.1 Introduction

Direct encoding is implemented by omitting the encoder. This is done by connecting AxiStreamToByte directly to the EncodingGearBox.

> **Remark 8.11:** *Direct mode*
>
> Direct decoding (no encoding) should not be used by any front-end, and is only included for debugging purposes. If no encoding technique is used on top of an E-Link, there is no way for the decoder to distinguish the byte boundary, and where a frame (chunk) starts or ends.

8.4.17 TTC Encoder

Remark 8.12: *TTC for phase II*

TTC-PON has been mentioned as the replacement for TTC in Phase II. The protocol is not yet final and no functional TTC-PON systems are currently available. Therefore the TTC system as defined in Phase I FELIX will be described in this section.

8.4.17.1 Introduction

For Phase 1, the standard encoded LHC TTC [12] signal will arrive to FELIX via a standard TTC fiber (multi-mode, ST connector) and will be decoded by FPGA firmware that receives the separated clock and data from the TTC FMC card on the FLX-709 (the Mini-FELIX), or by equivalent circuitry on the FLX-711/FLX-712 FPGA card. For Phase 2, the Phase 1 functionality will be implemented as well on lpGBT E-links and extended where needed. TTC data will be stuffed, on each BC clock, with fixed latency, directly into all output E-links to the Front End with the “TTC” attribute.

8.4.17.2 Interfaces

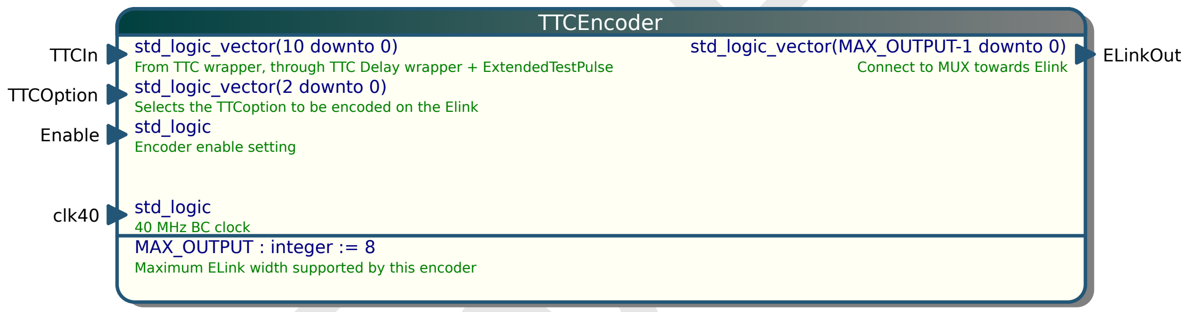


Figure 8.44: The TTC Encoder entity.

Unlike other encoders, the TTC encoder will not have an AXI4-stream interface, and also contains no FIFO. A strict requirement for TTC distribution is that the latency will be fixed. The data to be encoded does not arrive from the usual path as in other encoders, the data encoded arrives on TTCIn and the bits are described in Table 8.29.

8.4.17.3 Functional Description

Each E-link can be configured to choose bits from the possible bits shown in Table 8.29, where Brcst[7:2] are the TTC user-defined broadcast command bits. The number of bits chosen, two, four or eight, must match the width of the TTC E-link.

Table 8.29: Below is the list of bits decoded from the TTC system that can be chosen to be sent on an E-link defined as a TTC E-link..

Brc_t2[1]	Brc_t2[0]	Brc_d4[3]	Brc_d4[2]	Brc_d4[1]	Brc_d4[0]	ECR	BCR	B-chan	L1A

8.4.17.3.1 TTC Delay and Extended testpulse

Inside the Encoding block, at link scope, an optional delay of 0 to 15 BC clocks can be added to the TTC system, before the bits are distributed to the TTCEncoder entity. Additionally one signal is added to the bits to choose from; the Extended testpulse (TP). The Extended testpulse is a copy of Brc_d4[0] which is stretched from 32 40 MHz BC clock cycles.

The result is a delayed version of the bits in Table 8.29 with one extra bit added for the test pulse. The delayed bits are described in Table 8.30

Table 8.30: Below is a copy of the bits found in 8.29 but extended with the external testpulse (TP), and with an adjustable delay (0-15 BC).

TP	Brc_t2[1]	Brc_t2[0]	Brc_d4[3]	Brc_d4[2]	Brc_d4[1]	Brc_d4[0]	ECR	BCR	B-chan	L1A

8.4.17.3.2 TTC Options

Table 8.31 shows the implemented TTC data formats for Front ends. TTC option 5 was a special option implemented in Phase I LTDB mode only, where a BCR would be delayed by 0.5 BC (12.5 ns). This functionality will be implemented as a configuration in Phase II.

Table 8.31: Possible TTC options (Brc_d4[3:0] and Brc_t2[1:0] are the TTC user defined broadcast command bits. Bit 0 is the first bit transmitted out..

E-link option		**bit 7**	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
0	2 bits							B-chan	L1A
1	4 bits					B-chan	ECR	BCR	L1A
2	8 bits	B-chan	Brc_d4[3]	Brc_d4[2]	Brc_d4[1]	Brc_d4[0]	ECR	BCR	L1A
3	8 bits	L1A	Brc_d4[1]	TP	ECR	OCR*	L0A*	Brc_d4[3]	Brc_d4[2]
4	4 bits					BCR	BCR	BCR	BCR
5 (LTDB)	2 bits							BCR	BCRd*

In Table 8.31, 3 bit fields are listed (marked with *) that are not directly input to the TTC Encoder:

- OCR: This bit is set 1 BC clock after BCR, and stretched for a second clock when brc_t2[1] is set.
- L0A: In phase 1 TTC there is no bit for L0A, therefore a copy of L1A is used instead.
- BCRd: A 1 BC clock delayed version of BCR, to allow a 12.5ns shift in time of the BCR distribution (for LAr LTDB mode only)

For Options 0, 1 & 2, the destination must decode the B-channel, one bit per 40 MHz clock. Firmware is available. It may be that 4 or 8 bits of TTC data need to be sent when, due to E-group contraints, only 2 or 4-bit E-links are available. In this case, 2 or 4-bit options it could be defined to send particular TTC bits, so as to build 4 or 8-bit wide data from multiple 2 or 4-bit E-links. Note that:

- The E-link clock can be 40 MHz, but, for example, the 4-bit field can be transferred at 160 Mb/s if the receiver generates a ×4 multiple of the 40 MHz E-link clock.
- Typically, the reverse direction of the event data E-link can be used for TTC.
- Unlike 8b/10b encoding, the TTC options above are not DC-balanced; *TTC E-links must not be AC-coupled.*
- Transparent upgrade to the Phase 2 TTC system will be possible by changing the mezzanine board on the FELIX FPGA PCIe card
- The case of a FELIX with only TTC input and only TTC output, i.e. a TTC distributor, is needed by the LAr LTDB.

As an example, the TTC formats required by the New Small Wheel are described. The first line of Table 8.32 shows the format of the TTC words sent to the NSW Readout Controller on every bunch crossing. It provides 8 bits and requires a 320 Mb/s E-link. NSW uses Option 3 in Table 8.31 and assigns the meanings to the various broadcast bits as shown in Table 8.32. The second line shows the format sent to the NSW ART trigger ASIC on a 160 Mb/s E-link. Only BCR is required; it is repeated four times so that it is present for one complete BC clock.

Table 8.32: Line 1: Format of the 8-bit TTC word sent to the NSW Readout Controller on every bunch crossing. “OCR” is the Orbit Count Reset, “EC0R” is the reset for the Level-0 ID and “reset” is a Readout Controller soft reset. Note that bits 7 and 6 are delivered by the GBTx to the E-link in the bunch crossing **following** the other six bits. See Figure 11 of [11]. EC0R and L0A, are reserved for Phase 2; for Phase 1, FELIX sends ECR and L1A for EC0R and L0A.
Line 2: Format sent to the NSW ART trigger ASIC..

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Use:	*EC0R*	SCA_reset	*L0A*	BCR+OCR	ECR	TestPulse	SoftReset	L1A
					BCR	BCR	BCR	BCR

Note that, to save bits, OCR is encoded on the BCR line: width of 1 BC = BCR, width of 2 BC = OCR. Because BCR does not reset the BCID counter, but rather loads a configurable offset, OCR means reset the orbit counter on the **next rollover** of the BC counter. The double width BCR implies that a BCR is also performed, on the first BC of the pair. A double-width BCR is scheduled by sending Brcst7, otherwise unused by NSW, to indicate that the next BCR should be double-width. Note that EC0R is not needed in a single level trigger scheme. The SCA_reset, bit 6 of the TTC word, allows resetting the SCA via the TTC path.

Compatibility with the legacy TTCrx ASIC: For the TTCrx ASIC, broadcast bits Brcst[1:0] (BCR and ECR) were strobes with one BC duration, whereas Brcst[7:2] were latched until a subsequent broadcast reset them. For FELIX, all broadcast bits are strobes. FELIX will be updated to provide a strobe versus latch option for Brcst[7:2].

8.4.17.4 Configuration

The TTC Encoder has two configuration inputs:

- TTCOption[2:0] which directly translates to the TTC encoding option described in Table 8.31
- Enable to enable the TTC Encoder entity.

8.4.17.5 Status Indicators

The TTC Encoder has no status indicators.

8.4.17.6 Latency

The Latency from TTCIn to ElinkOut is typically 1 BC clock (25ns). OCR and BCRd have one extra BC delay by design. At E-Group level, the E-Path multiplexer adds one additional BC clock of latency.

8.4.17.7 Error Handling

The TTC Encoder has no error handling.

8.4.17.8 Estimated Resource Usage

8.4.18 Encoder for 25 Gb/s links

8.4.18.1 Introduction

8.4.18.2 Interfaces

8.4.18.2.1 Overview

8.4.18.2.2 Interface to component 2

8.4.18.3 Functional Description

8.4.18.4 Configuration

8.4.18.5 Status Indicators

8.4.18.6 Latency

8.4.18.7 Error Handling

8.4.18.8 Estimated Resource Usage

8.5 Link Wrapper

8.5.1 Introduction

As shown in Figure 8.1, the Link Wrapper instantiates the high speed transceivers (Xilinx GTH/GTY) and interfaces with their high speed serial links and reference clocks on one side.

The basic link encoding and decoding is also performed inside the Link Wrapper. The basic protocols that are encoded and decoded inside the link wrapper are:

- GBT: This protocol will be encoded and decoded, data will be (de)scrambled and forward error correction will be performed. Delivered to the the Encoding / Decoding blocks are ready to use Elinks with all their bits clocked at 40 MHz BC Frequency.
- lpGBT: This protocol will be encoded and decoded, data will be (de)scrambled and forward error correction will be performed. Delivered to the the Encoding / Decoding blocks are ready to use Elinks with all their bits clocked at 40 MHz BC Frequency.
- FULL: 9.6Gb/s 8b10b encoded data will be decoded as 32b + CharisK indication.
- 25Gb/s links: Several subdetectors have expressed their interest to interface FELIX with 25Gb/s links. The protocol for this type of link has not been defined yet, but candidates are Aurora and Interlaken. Encoding and Decoding of this link will not happen inside the link wrapper, the link wrapper will deliver either 64b66b or 64b67b encoded frames to the Encoding / Decoding blocks.
- 10Gb/s links: The L1Track group has expressed their interest in 10Gb/s links. The protocol for this link has not yet been defined.

8.5.2 Interfaces

This first two tables should not be in this section (8.6), one tables to be added for GBT mode.

8.5.2.1 Overview

	User Data			IC	EC
Bandwidth [Mb/s]	80	160	320	80	80
Maximum number	16	8	4	1	1
HDLC	TBD	TBD	TBD	✓	✓
8b/10b	TBD	TBD	TBD	TBD	TBD
6b/8b	TBD	✓	TBD	TBD	TBD
TTC Trickle merge	TBD	TBD	TBD	TBD	TBD
Endeavor (Strip)	TBD	TBD	TBD	TBD	✓
Pixel custom protocol	TBD	✓	TBD	TBD	TBD

Table 8.33: From-host eLink Groups..

Bandwidth [Gbps]	5.12						10.24					
FEC coding	FEC5			FEC12			FEC5			FEC12		
Bandwidth [Mbps]	160	320	640	160	320	640	320	640	1280	320	640	1280
Maximum number	28	14	7	24	12	6	28	14	7	24	12	6
HDLC	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD
8b/10b	TBD	TBD	TBD	TBD	TBD	TBD	TBD	✓	TBD	TBD	TBD	TBD
Aurora	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	✓

Table 8.34: To-host eLink Groups..

From-host	
Data Rate [Gb/s]	2.56
Frame [bits]	64
Header [bits]	4
Coded header	Yes
User field [bits]	36
FEC [bits]	24
User Bandwidth [GHz]	1.44
Num. of eLinks groups [8 bit]	4
eLinks bandwidth [MHz]	80/160/320
Num. of eLinks	16/8/4
EC bandwidth [MHz]	80
IC bandwidth [MHz]	80
Corrected [bits]	12
Efficiency (#data/#frame)	56%

Table 8.35: lpGBT From-host specification [13]..

To-host				
Correction Scheme	FEC5		FEC12	
Data Rate [Gb/s]	5.12	10.24	5.12	10.24
Frame [bits]	128	2 x 128	128	2 x 128
Header [bits]	2	2+2	2	2+2
Coded header	no	no	no	no
User field [bits]	116	232	102	204
Code [bits]	10	20	24	48
User Bandwidth [GHz]	4.64	9.28	4.08	8.16
Num. of eLinks groups [16 bit]	7	7	6	6
eLinks bandwidth [MHz]	160/320/640	320/640/1280	160/320/640	320/640/1280
Num. of eLinks	28/14/7	28/14/7	24/12/6	24/12/6
EC bandwidth [MHz]	80	80/160	80	80/160
IC bandwidth [MHz]	80	80/160	80	80/160
Unassigned bits	0	4	2	8
Corrected [bits]	5	2 x 5	12	2 x 12
Efficiency	91%	91%	80%	80%

Table 8.36: lpGBT To-host specification for FEC5 and FEC12 decoding scheme [13]..

2085 ## 8.5.3 FUNCTIONAL DESCRIPTION

2086 ### 8.5.3.1 GBT MODE WRAPPER

2087 A wrapper shown in Figure 8.45 is provided to include the Xilinx transceiver and the GBT encoding and
2088 decoding modules.

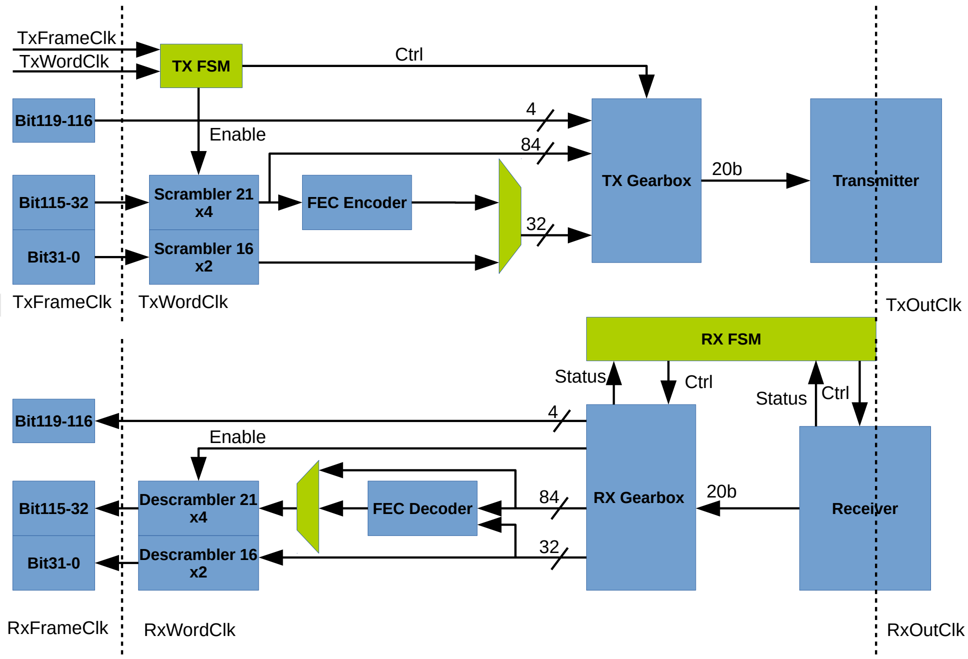


Figure 8.45: Block diagram for the GBT module in the link wrapper.

2089 The transceiver is configured as 4.8 Gbps for both directions. The GTH transceivers can be configured
2090 in unit of one channel, which uses the CPLL in the transceiver, or be configured to in unit of one quad of
2091 four channels, which uses the high quality QPLL in transceiver. The GBT encoding and decoding module are
2092 based on the code from the CERN GBT group. It has the forward error correction (FEC) capability. Data is

scrambler before the FEC encoding in transmiter direction. In the receiver path, data is descrambled after the FEC decoding. Some modifications [14] are done to reduce the latency, and to support the online GBT mode switching between normal mode and wide-bus mode. The interface between GBT wrapper and the Central Router will be 120-bit GBT data frame in the 40 MHz system clock which is recovered from TTC system.

8.5.3.2 LPGBT MODE WRAPPER

A wrapper is provided to include the Xilinx transceiver and the lpGBT encoding and decoding modules. The lpGBT encoding and decoding modules [13], and the lpGBT emulator for the ASIC in front-end side are provided by the CERN GBT group. The PRBS test with 24-ch bidirectional lpGBT links between 2 FLX-712 cards are carried out with different line rates and FEC coding in Table 8.34. The Phase-II firmware with lpGBT wrapper has also been built for FLX-712, and been verified in the system integration between FLX-712 and the ATLAS Phase-2 strip stave 8.46.

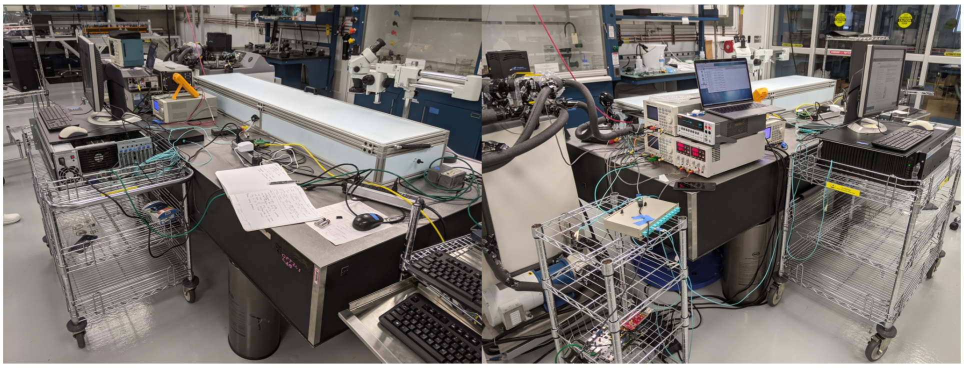


Figure 8.46: Integartion test between FLX-712 and ATLAS Phase-II Strip Stave.

8.5.3.3 FULL MODE WRAPPER

A wrapper for the Xilinx GTH serializer and deserializer cores is provided. Such a wrapper for Altera would have to be provided by someone familiar with Altera FPGAs. The Xilinx GTH transmitter and receiver are configured to operate at a line rate of 9.6 Gb/s. Either QPLL's or CPLL's may be used. For Full mode, the GTH will be operated in simplex mode, i.e. transmission (Tx) or reception (Rx). The GTH reference guide [15] gives details about the serializer for Xilinx 7 series FPGA devices. As shown in Figure 8.47, the GTH transmitter and receiver will be operated at 240 MHz $\times$ 32-bits. The IDLE symbol (K28.5) is defined as the comma character, i.e. the symbol that defines the 32-bit alignment in FELIX MGT receiver. The packet is assembled by the stream controller in multiples of 32-bit words. To insert a K-character (SoP, EoP, Idle, BUSY-ON, BUSY-OFF) in the stream, the low byte is set to the K-character 8-bit code and the lowest of the four Kchar_flag bits is set to 1 (See Figure 8.47). The receiver re-assembles the 32-bit words and flags the K-characters. On the receiver side, at start-up and if alignment is lost, the SoP will be pushed later in the output stream so that it becomes the low byte in the next 32-bit word.

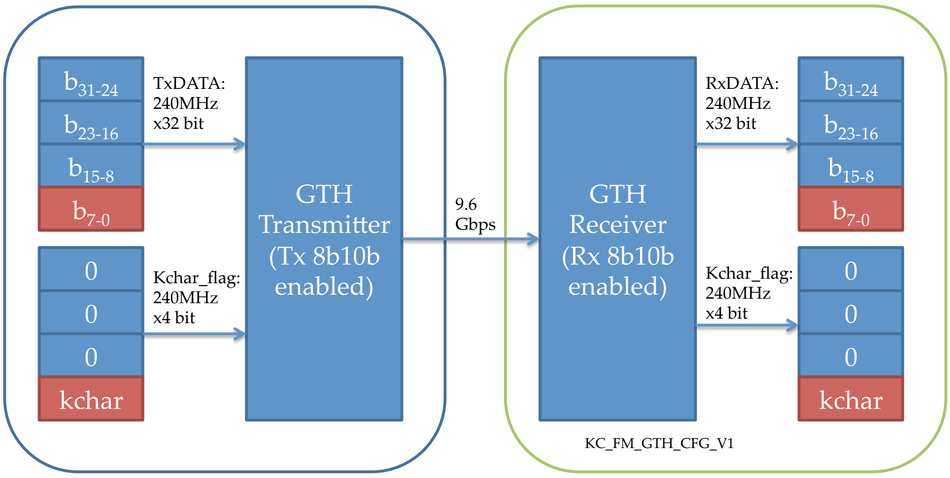


Figure 8.47: Block diagram for the serializer and deserializer modules for Full mode.

8.5.4 Configuration

A unified firmware block Link Wrapper is put in the top level HDL file. For different firmware modes, the building script will configure the Link Wrapper before synthesis. Meanwhile various of reset ports of the Xikinx transceivers and the protocol encoding, decoding modules are connected to FELIX control registers. After the firmware loading, the software can do the online reset and configuration to the Link Wrapper.

8.5.5 Status Indicators

Some status registers from the link wrapper are connected to the FELIX monitoring registers. For example, the TX, RX reset done signals, the PLL, CDR lock status, 8b10b flags of the transceivers, and the link locking flag, error flag and other status signals from the GBT, lpGBT modules. Via the FELIX registers, software will be able to monitor the status of Link Wrapper. The software will be able to monitor the status, reset or reconfigure the Link Wrapper. Meanwhile finite state machine (FSM) inside the Link Wrapper will keep checking the link status and carry out the automatic reset or bit-slip procedures, until the link is locked. There will be two registers for GBT and lpGBT link status. One for the short-term status monitoring, one for the long-term status monitoring. The locking status read out by FELIX software and the FSM. For the latter, once unlock status occurs, the lost of lock bit will be asserted, until manually clearance via software.

8.5.6 Latency

For GBT mode, some optimization was carried out for the GBT encoder. The Link Wrapper contributes about 60-81 ns, or less than 3.25 Bunch Crossings for the toFrontend (downlink) direction. A full chain latency measurement was carried for Phase-I review in the past. For lpGBT mode, the CERN code will be used directly, the latency mainly depends on the lpGBT protocol itself. A full chain latency test will need to be carried out from the fiber from LTI, to the output elink of GBT ASic and lpGBT ASIC.

8.5.7 Estimated Resource Usage

This may be removed form this section, since the resource usage of link wrapper should be much much less than the elink handling in Central Router. A resource usage should be estimated from the top level.

8.6 ToHost Data Emulator

8.6.0.1 Introduction

8.6.0.2 Interfaces

8.6.0.2.1 Overview

8.6.0.2.2 Interface to component 2

8.6.0.3 Functional Description

8.6.0.4 Configuration

8.6.0.5 Status Indicators

8.6.0.6 Latency

8.6.0.7 Error Handling

8.6.0.8 Estimated Resource Usage

8.7 TTC Emulator

8.7.1 Introduction

8.7.2 Interfaces

8.7.2.1 Overview

8.7.2.2 Interface to component 2

8.7.3 Functional Description

8.7.4 Configuration

8.7.5 Status Indicators

8.7.6 Latency

8.7.7 Error Handling

8.7.8 Estimated Resource Usage

8.8 Legacy TTC Wrapper

8.8.1 Introduction

TODO: Describe TTC BUSY wrapper

8.8.2 Interfaces

8.8.2.1 Overview

8.8.2.2 Interface to component 2

8.8.3 Functional Description

8.8.4 Configuration

8.8.5 Status Indicators

8.8.6 Latency

8.8.7 Error Handling

8.8.8 Estimated Resource Usage

8.9 LTI/TTC Interface

8.9.1 Introduction

TODO: Describe LTI TTC_P2P, see also p30, Table 2.2 of [16] interface

8.9.2 Interfaces

8.9.2.1 Overview

8.9.2.2 Interface to component 2

8.9.3 Functional Description

8.9.4 Configuration

8.9.5 Status Indicators

8.9.6 Latency

8.9.7 Error Handling

8.9.8 Estimated Resource Usage

8.10 BUSY SELECTION

8.10.1 INTRODUCTION

TODO: Describe BUSY or / selector

8.10.2 INTERFACES

8.10.2.1 OVERVIEW

8.10.2.2 INTERFACE TO COMPONENT 2

8.10.3 FUNCTIONAL DESCRIPTION

8.10.4 CONFIGURATION

8.10.5 STATUS INDICATORS

8.10.6 LATENCY

8.10.7 ERROR HANDLING

8.10.8 ESTIMATED RESOURCE USAGE

8.11 CRToHost: ToHost or Upstream Central Router

8.11.1 Introduction

CRToHost, or the Upstream / ToHost Central Router is the block that takes AXI stream (axis32) data from the several decoders. This data is formatted into blocks, see Section B.2.1. The AXI stream data enters the CRToHost entity in the form of a two dimensional array of which the first dimension is the number of optical links, the second dimension is the number of streams per link. This is usually the number of E-links on a GBT or lpGBT link. For FULL mode the size of the second dimension is 1.

The data is demultiplexed, buffered and formatted into a 256b or 512b FIFO interface that is acceptable for the Wupper ToHost DMA interface.

8.11.2 Interfaces

8.11.2.1 Overview

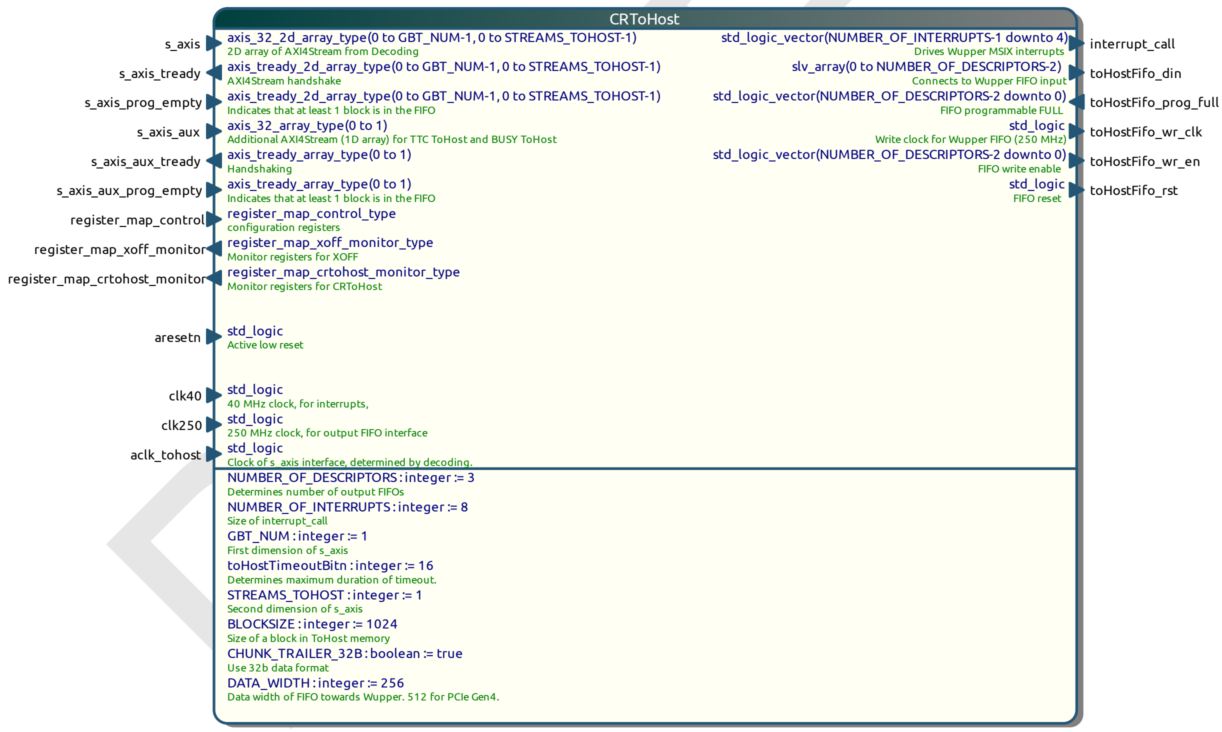


Figure 8.48: CRToHost interface symbol.

8.11.2.2 Interface from decoding

The interface s_axis of the type axis_32_2d_array_type (a 2D array of axis_32_type), see listing 8.2. The input s_axis and the handshake lines s_axis_tready are used to take data from the different protocol decoders. Additionally, s_axis_prog_empty is required. This is a 2D array if std_logic with the same dimensions. It should be connected to the prog_empty outputs of the axis fifo instances in the decoders, to indicate that at least a full block of data is available inside the FIFO. This is used for the selection of the AXIs mux, to assure that a complete block can be sent out at once without stalling the MUX for other AXI stream inputs. The size of s_axis and the corresponding handshake signals is (0 to GBT_NUM-1, 0 to STREAMS_TOHOST-1).

GBT_NUM is the number of optical links connected to the the Decoder in the endpoint. If the FELIX firmware has two PCIe endpoints, this size will be half the total number of optical links available. The second number STREAMS_TOHOST is the number of E-Links per optical link. This depends on the firmware flavour and is defined at build time.

An additional input channel with a different dimension, but otherwise the same functionality is available as s_axis_aux. This link is internally added to the array of s_axis, but is connected to the virtual E-Links: TTCToHost 8.3.18 and BUSYXOFF **??**.

```
type axis_32_type is record
tdata        : std_logic_vector(31 downto 0);   --! Data bus
tvalid       : std_logic;                       --! Valid data when tready is '1'
tlast        : std_logic;                       --! Last cycle of a chunk
tkeep        : std_logic_vector(3 downto 0);    --! Serves as byte enable
tuser        : std_logic_vector(3 downto 0);    --! Meaning of tuser bits:
                                                --!   3: Truncation/FIFO full
                                                --!   2: FrontEnd BUSY
                                                --!   1: Chunk error
                                                --!   0: CRC error
end record;

type axis_32_array_type is array (natural range <>) of axis_32_type;
type axis_32_2d_array_type is array (natural range <>, natural range <>)
                                                  of axis_32_type;
```

Listing 8.2: A snippet from axi_stream_package.vhd showing the 32b axi stream type.

8.11.2.3 Interface to Wupper

The data output of CRToHost is an interface to the input of one or multiple FIFO's. The data width of the interface (256 or 512 bits) is determined by the generic DATA_WIDTH. The number of FIFO interfaces is determined by the generic NUMBER_OF_DESCRIPTORS-1, One is subtracted, because the last of descriptor in Wupper (see also section 8.13) is used for communication in FromHost direction, towards CRFromHost (see 8.12)). The FIFO interface consists of toHostFifo_din, toHostFifo_prog_full, toHostFifo_wr_clk, toHostFifo_wr_en and toHostFifo_rst. The signal names can be connected to the corresponding interface ports of Wupper. Apart from the FIFO interface, CRToHost can also generate MSIX interrupts using interrupt_call.

8.11.3 Functional Description

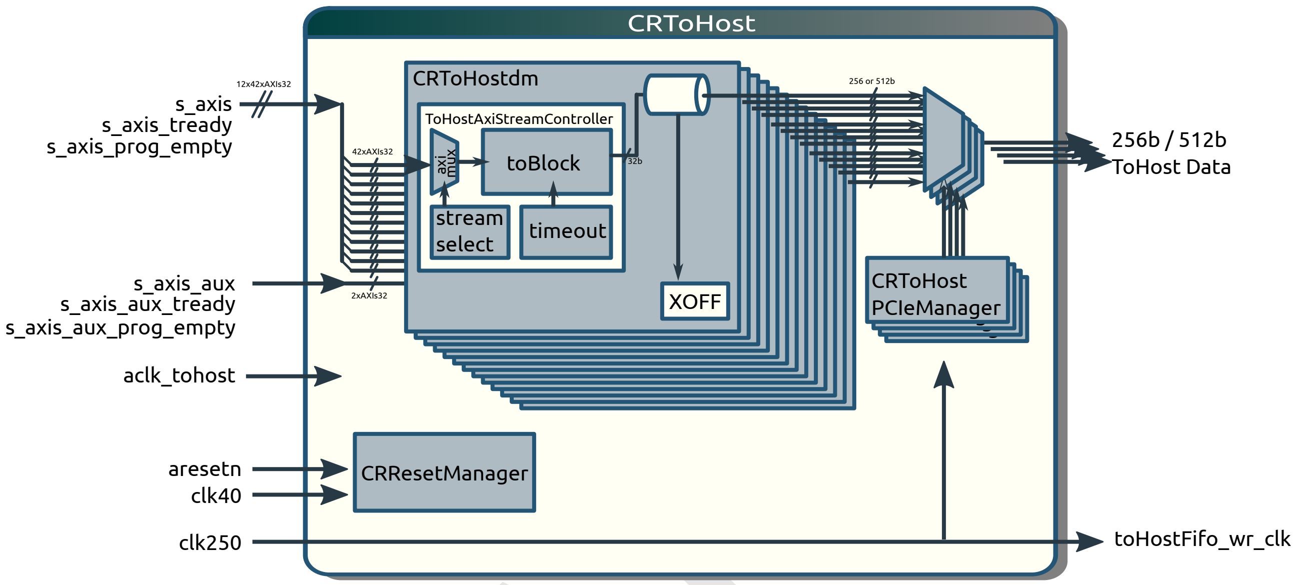


Figure 8.49: CRToHost Block Schematic.

8.11.3.1 CRToHostdm

For every item in the first dimension of s_axis (GBT_NUM), usually the number of optical links, one CRToHostdm is instantiated. This is a wrapper for the ToHostAxiStreamController, the Channel FIFO and the XOFF mechanism.

8.11.3.1.1 ToHostAxiStreamController

The ToHostAxiStreamController Consists of the 4 following processes, that work together to convert AXI streams into the FELIX block format see Appendix B.2.1.

- **Stream Select** looks at the s_axis_prog_empty bits which show whether enough data resides in one of the AXI stream FIFOs, inside the decoders. As a second priority, a stream can be selected that has _tvalid set to '1', after a timeout occurs. This way a partial block can be read out.
- **AXI Stream MUX** is a clocked mux that multiplexes the array of axis_32_type records into a single AXI stream, selected by Stream Select.
- **To Block** takes the seleced AXI stream record and converts this into the FELIX block format (see Appendix B.2.1), 32 bit at a time. It generates a 32b data output, a FIFO write enable and responds to the FIFO full handshake line to pause the operation. Exactly on the beginning of every block, a 32 bit block header will be generated, and at the end of a chunk (s_axis.tlast = '1') a chunk trailer will be added to the data stream. If a chunk is still in the process of being moved out towards the channel FIFO, but the block is at it's end, an intermediate subchunk trailer will be added, indicating the length of the partial chunk and a flag that the chunk is partial. At the end of a block, the AXI mux may select another AXI stream, so the end subchunk will be sent out later, when the corresponding AXI stream is selected again.

- **Timeout Mechanism** Counts up to the value of the register TIMEOUT_CTRL.TIMEOUT and increments a 2-bit counter for every AXI stream if the corresponding tvalid is '1', but prog_empty is also '1' indicating a partial block. A counter value of 2 means that the corresponding AXI stream may be selected by Stream Select and the data copied into the channel FIFO.

8.11.3.1.2 Channel FIFO

The channel FIFO is an assymetric FIFO, matching the 32 bit output of ToHostAxiStreamController to the width of the Wupper input FIFO (256 bit for PCIe Gen3x8, 512 bit for PCIe Gen4x8). The depth of the FIFO in bytes is set to fit exactly to blocks.

8.11.3.1.3 XOFF Mechanism

8.11.3.2 CRToHost PCIeManager

There is one CRToHost PCIeManager generated for every ToHost FIFO / descriptor in Wupper, as well as one CRToHost MUX. The CRToHost PCIe Manager reads the programmable empty flags from the channel FIFOs to determine whether at least one block of data is available inside the FIFO, and whether the Wupper FIFO has empty space to store that block. If these conditions are met, the select signal for the CRToHost MUX is set and the read enable for the channel FIFO, as well as the write enable for the Wupper FIFO will be asserted for exactly the number of cycles needed for one block.

8.11.3.3 CRToHost MUX

The CRToHost MUX is selected by the CRToHost PCIemanager and multiplexes the number of input channels (GBT_NUM+1 for the AUX channel) into a single FIFO data port towards Wupper.

8.11.3.4 CRResetManager

The CRResetManager synchronizes the incoming reset to clk40 with two extended reset pulses:

- **Logic reset**: This reset holds for 15 clocks after the release of the incoming reset (aresetn), this reset is used to reset all logic in the ToHostAxiStreamController, XOFF, CRToHostMUX and CRResetManager.
- **FIFO reset**: This reset holds for 8 clocks after the release of aresetn, and is used to reset the channel FIFO as well as the Wupper FIFO of which the reset is generated from within the CRToHost port. This reset clears earlier, because the FIFOs take a few clock cycles to become active after a reset.

8.11.4 Configuration

CRToHost does not have many runtime configuration options. It assumes that the decoders, feeding data to the AXI stream interfaces can be enabled / disabled through configuration registers. What is left to configure is:

- XOFF_FM_CH_FIFO_THRESH_LOW: The deassertion watermark level of the channel FIFO for which XOFF will be released
- XOFF_FM_CH_FIFO_THRESH_HIGH: The ssertion watermark level of the channel FIFO for which XOFF will be asserted
- TIMEOUT_CTRL.TIMEOUT: Number of BC clock cycles after which a timeout will occur in case a partial block resides in an E-Path FIFO.
- TIMEOUT_CTRL.ENABLE: Enable the timeout mechanism.

8.11.5 Status Indicators

The status of the FIFO can be read through the CRTOHOST_FIFO_STATUS.FULL and CRTOHOST_FIFO_-STATUS.FULL_LATCHED registers in the register map. The XOFF signals are generated from the same FIFO but with a different threshold (see Configuration). The XOFF status can be read through the registers XOFF_FM_HIGH_THRESH.CROSS_LATCHED, XOFF_FM_HIGH_THRESH.CROSSED and XOFF_-FM_LOW_THRESH_CROSSED.

8.11.6 Latency

The latency of CRToHost strongly depends of the number of AXI streams per link. If only one of them contains data, the beginning of a block can start 8 clock cycles after prog_empty goes low. This latency can be neglected as it is much smaller than:

- The time it takes to fill the Decoder FIFO with one block of data
- The PCIe transfer latency towards the host server
- The time it takes to select the AXI mux and / or the CRToHost MUX if other AXI Streams or other channels are in the process of transferring data.

8.11.7 Error Handling

Errors can be generated inside the axi stream in the tuser bits. These bits will be reflected in the (sub)chunk trailers. Also internal data format errors as well as timeout and truncation (caused by a FULL FIFO while data was transferred, so a loss of data) will be reflected in the (sub)chunk trailers.

8.11.8 Estimated Resource Usage

	LUT		**FF**		**BRAM**	
KCU115 / FLX712	10406	1.56%	11713	0.88%	52	2.4%
VU37P / FLX128	7453	0.57%	7643	0.29%	104	5.15%

Table 8.37: CRToHost Resource utilization.

8.12 CRFromHost: FromHost or Downstream Central Router

8.12.1 Introduction

The FromHost or Downstream Central Router (CRFromHost) is the main interface between the Wupper and the encoders towards the detector. It is used to fanout the data from the PCIe interface to the link encoders.

8.12.2 Interfaces

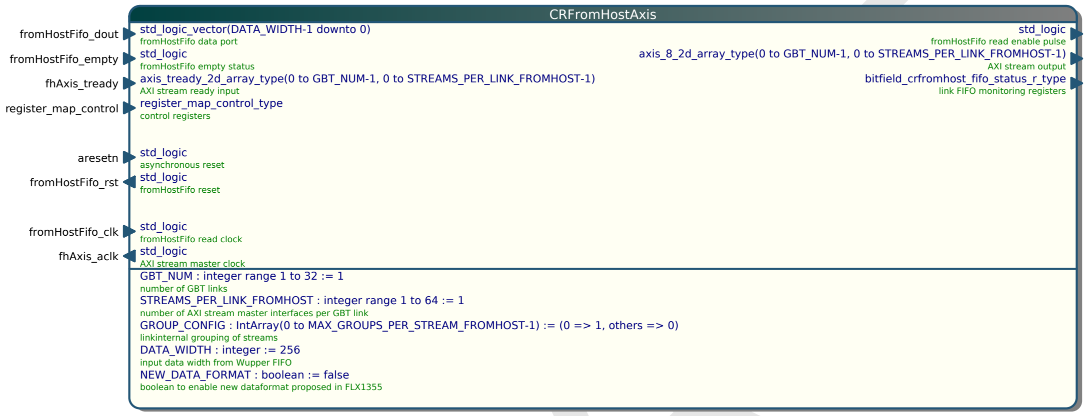


Figure 8.50: The FromHost or Downstream Central Router entity.

8.12.2.1 Interface to Wupper

The interface to Wupper is a 256-bit (for PCIe 3.0) or 512-bit wide (for PCIe 4.0) FIFO interface which can be connected to a standard FIFO. Whenever there is data available (`empty = '0'`) and the internal data forwarding is not stalled, a read-enable pulse is generated. The data has to be valid in the following read-clock-cycle. A separate reset signal can be used to clear the FIFO in case of a reset or flush of the Central Router. Figure 8.51 shows an example waveform of input signals for the CRFromHost.

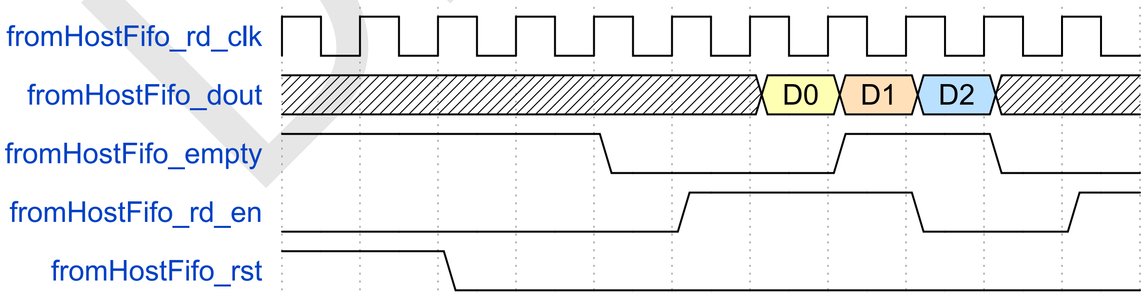


Figure 8.51: Example waveform of a typical FromHost Central Router transfer with its FIFO interface. [8].

Each 256-bit block at the input of the CRFromHost represents a packet. In case of a 512-bit FIFO interface, two packets are sent simultaneously. Each packet consists of a 16 bit header followed by 240 bits of payload. Table B.4 shows how the bits are assigned in that packet. Details of the data format can be found in B.2.2.

8.12.2.2 Interface to the encoders

All encoders are connected to the FromHost Central Router as AXI stream 8b slaves. Therefore, the CRFromHost provides a number of AXI stream 8b master interfaces. Each interface is connected to a single encoding instance. The masters are split into two groups. First all masters are grouped by the corresponding lpGBT or GBTx link they belong to. Inside each lpGBT/GBTx link there is an additional grouping to ease throughput of the Central Router. All AXI stream master of a group have a total maximum bandwidth which cannot be exceeded. An example waveform of a typical AXI stream 8b transfer is shown in Figure 8.52.

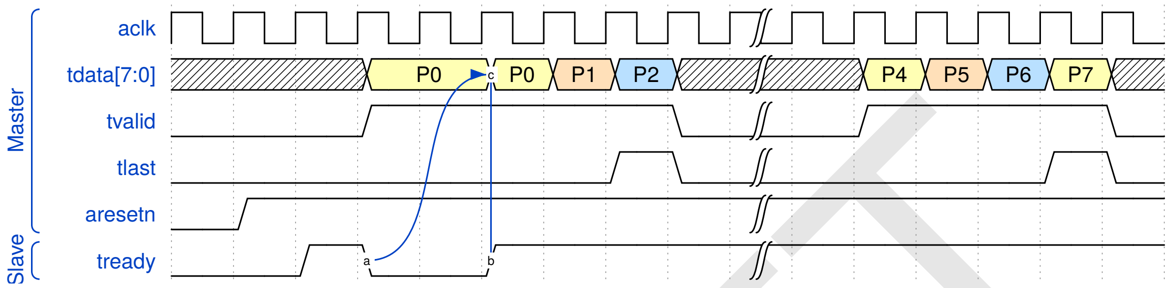


Figure 8.52: Example waveform of a typical AXI stream 8b transfer. [8].

8.12.3 Functional Description

8.12.3.1 CRFromHost top-level

The top-level module provides the instantiation of all sub-modules in the CRFromHost together with logic to monitor the internal status of the CRFromHost and the first distribution level.

The distribution logic first only distributes packets to the different link FIFOs, where each GBT or lpGBT link has its own FIFO. The link ID field in the packet header is used as a address to which link the packet should go. If all bits in the link ID field are set, the packet is treated as a broadcast packet to all links and therefore written to all link FIFOs in parallel.

All link FIFOs are constantly monitored. If a link FIFO is full this is reported through the register bank. A latched version of the full flag is also available in the register bank. Both flags can be found in the `CRFROMHOST_FIFO_STATUS` register.

8.12.3.2 CRFromHost data manager

The data manager contains the next distribution stage in the CRFromHost. Due to bandwidth reasons the streams of a GBT or lpGBT link are split into groups, where each group has a certain maximum bandwidth. This also represents the scheme of e-groups in the GBT chip. The data manager processes the stream ID field in the packet and forwards the packet to the transfer manager handling the group the stream belongs to. If all bits in the stream ID field are set the packet is considered to be a broadcast packet. This broadcast is sent to all group FIFOs in parallel.

8.12.3.3 CRFromHost transfer manager

The transfer manager is the last stage of distribution and handles all streams in a group. Based on the stream ID field it decides which stream will be used to transmit the packet. For this stream a AXI stream transmission is initiated.

8.12.4 Configuration

8.12.4.1 Generics

The configuration of the CRFromHost is mainly accomplished through various generics, which are evaluated during synthesis time of the firmware.

GBT_NUM: This generic defines the total number of GBT or lpGBT links handled by the CRFromHost. It is an integer number between 1 and 31. For each link one data manager is instantiated.

STREAM_PER_LINK_FROMHOST: defines the total number of streams in each GBT or lpGBT link. It is an integer number between 1 and 63.

GROUP_CONFIG: is an array of integers with up to MAX_GROUPS_PER_STREAM_FROMHOST (usually 8) entries. The number of non-zero entries defines the number of groups, while each entry corresponds to the number of streams inside a group. The sum of all entries has to match STREAM_PER_LINK_FROMHOST.

DATA_WIDTH: input width from the PCIe FIFO. Allowed values are 256 for PCIe 3.0 links and 512 for PCIe 4.0 links.

8.12.4.2 Run-time configuration

The run-time configuration of the CRFromHost is performed through the register map of FELIX. During run-time the only configurable part is the enabling or disabling of streams for broadcast transmissions. The BROADCAST_ENABLE_00 to BROADCAST_ENABLE_23 registers allow to include the stream of a specific GBT or lpGBT link to be included in broadcast transmissions.

8.12.5 Status Indicators

The full flag of the link FIFOs is available through the CRFROMHOST_FIFO_STATUS register. Also the latched full flag can be read out there.

8.12.6 Latency

The maximum latency of the CRFromHost depends strongly on the data it has to process. Therefore, no value is given.

The minimal latency was measured in a simulation to be 9 clock cycles of the CRFromHost clock.

8.12.7 Error Handling

8.12.8 Estimated Resource Usage

	LUT		FF		BRAM	
KCU115/FLX712	34113	5.14%	63516	4.78%	48	2.22%
VU37P/FLX128	49736	3.82%	63864	2.45%	48	2.38%

Table 8.38: CRFromHost Resource utilization.

8.13 WUPPER: PCIE DMA CORE AND REGISTER MAP

8.13.1 INTRODUCTION

Wupper[1] is designed for the ATLAS / FELIX project [17], to provide a simple Direct Memory Access (DMA) interface for the Xilinx Virtex-7 PCIe Gen3 hard block and has later been ported to the Kintex Ultrascale, Virtex Ultrascale+ and Versal Prime series. The core is not meant to be flexible among different architectures, but especially designed for the 256 and 512 bit wide AXI4-Stream interface [18] of the Xilinx Virtex-7 and Ultrascale FPGA Gen3 Integrated Block for PCI Express, and the Ultrascale+ and Versal Prime Gen4 Integrated Block for PCI Express (PCIe) [19, 20, 21, 22].

The purpose of Wupper is therefore to provide an interface to a standard FIFO. This FIFO has the same width as the Xilinx AXI4-Stream interface (256 or 512 bits) and runs at 250 MHz. The user application side of the FPGA design can simply read or write to the FIFO; Wupper will handle the transfer into Host PC memory, according to the addresses specified in the DMA descriptors. Several descriptors can be queued, up to a maximum of 8, and they will be processed sequentially one after the other. The number of descriptors (NUMBER_OF_DESCRIPTORS generic) plays an important role, it determines the total number of descriptors, but also the number of FIFO interfaces in the ToHost direction. The last descriptor is always dedicated for FromHost (DMA memory read from the server) transactions, all other descriptors are dedicated for ToHost transfers (Memory writes from the FPGA into the server memory).

Another functionality of Wupper is to manage a set of DMA descriptors, with an $address$, a $read/\overline{write}$ flag, the $transfersize$ (number of 32 bit words) and an $enable$ line. These descriptors are mapped as normal PCIe memory or IO registers. Besides the descriptors and the enable line (one per descriptor), a status register for every descriptor is provided in the register map.

For synthesis and implementation of the Xilinx specific IP cores, it is recommend to use the latest Xilinx Vivado release as listed in section 8.2. The cores (FIFO, clock wizard and PCIe) are provided in the Xilinx .xci format, as well as the constraints file (.xdc) is in the Vivado Format.

For portability reasons, no Xilinx project files will be supplied with the core, but a bundle of TCL scripts has been supplied to create a project and import all necessary files, as well as to do the synthesis and implementation. These scripts will be described later in this document.

[1] The person performing the act of bongelwuppen, the Gronings version of the famous Frisian sport of the Fierljeppen (canal pole vaulting) https://nds-nl.wikipedia.org/wiki/Nedersaksische_sp%C3%B6llegies#Bongelwuppen

2427 ## 8.13.2 INTERFACES

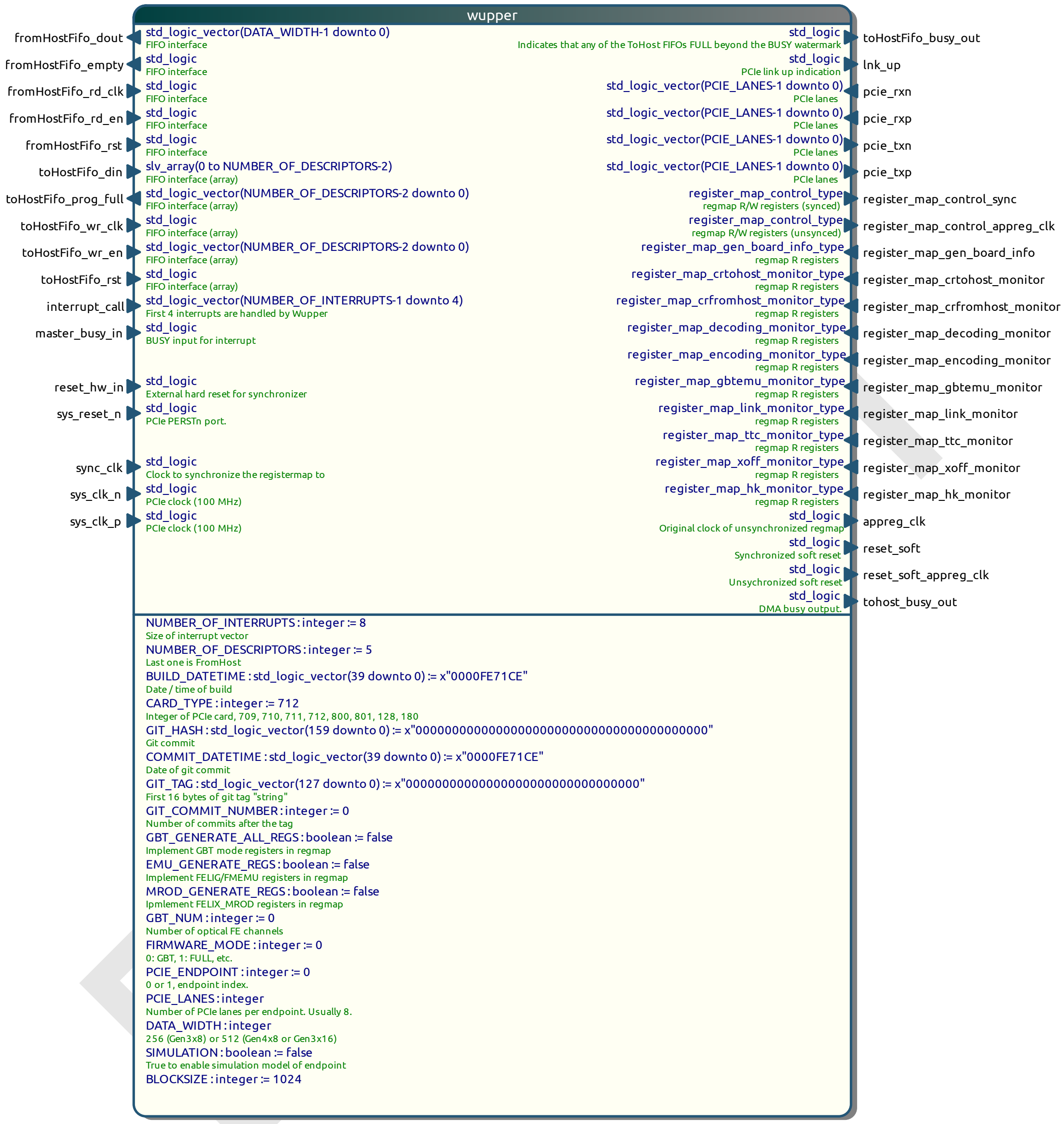


Figure 8.53: Wupper interface symbol.

2428 ### 8.13.2.1 GENERICS

Generic	Type	Default value	Description
NUMBER_OF_INTERRUPTS	integer	8	Number of individual interrupts supported by Wupper. See Section 8.13.6
NUMBER_OF_-DESCRIPTORS	integer	5	Total number of DMA descriptors for From- and ToHost. See 8.13.4
BUILD_DATETIME	std_logic_vector(39 downto 0)	x"0000FE71CE"	Date / time of build shown as BCD/HEX in the form of YYMMDDhhmm
CARD_TYPE	integer	712	Integer representation of the hardware platform:

			709 : VC709 710 : HTG710 711 : BNL711 v1.5 712 : BNL712 800 : Xupp3r VU9P 801 : BNL801 VU9P 128 : VCU128 180 : VMK180
GIT_HASH	std_logic_vector(159 downto 0)	(others => '0')	Git commit
COMMIT_DATETIME	std_logic_vector(39 downto 0)	x"0000FE71CE"	Date of git commit in the same form as BUILD_DATETIME
GIT_TAG	std_logic_vector(127 downto 0)	(ohters => '0')	First 16 bytes of git tag "string"
GIT_COMMIT_NUMBER	integer	0	Number of commits after the tag
GBT_GENERATE_ALL_-REGS	boolean	false	Implement GBT mode registers in regmap
EMU_GENERATE_REGS	boolean	false	Implement FELIG/FMEMU registers in regmap
MROD_GENERATE_REGS	boolean	false	Ipmlement FELIX_MROD registers in regmap
GBT_NUM	integer	0	Number of optical FE channels
FIRMWARE_MODE	integer	0	0: GBT, 1: FULL, etc.
PCIE_ENDPOINT	integer	0	0 or 1, endpoint index.
PCIE_LANES	integer		Number of PCIe lanes per endpoint. Usually 8
DATA_WIDTH	integer		256 (Gen3x8) or 512 (Gen4x8 or Gen3x16)
SIMULATION	boolean	false	True to enable simulation model of endpoint
BLOCKSIZE	integer	1024	FELIX block size to calculate FIFO thresholds

Table 8.39: Wupper Generics.

8.13.2.2 FROMHOSTFIFO

The FromHostFifo interface connects the output of the DMA FIFO in FromHost (Server => FPGA) direction. The FIFO ports are what you would expect from a standard FIFO interface, with a width of 256 bit or 512 bit, depending on the PCIe configuration (Gen3x8 or Gen4x8). In FELIX, the fromHostFifo interface is connected to the FromHost Central Router.

- fromHostFifo_dout : 256 or 512 bit data output of the DMA FromHost FIFO
- fromHostFifo_empty : Asserted if the fifo has no data available
- fromHostFifo_rd_clk : Clock to register fromHostFifo_dout with. Should be close or equal to 250MHz to support the nominal PCIe bandwidth.
- fromHostFifo_rd_en : Assert to read from the FIFO. fromHostFifo_dout will be registered on the next clock cycle.
- fromHostFifo_rst : Assert to reset / flush the FIFO.

8.13.2.3 TOHOSTFIFO

The ToHostFifo interface connects the ToHostFifos input ports (The number of FIFOs is determined by NUMBER_OF_DESCRIPTORS-1, see section 8.13.4) to the ToHost Central Router. Because there are multiple FIFO's in ToHost direction, the ToHostFifo port is also an array.

- toHostFifo_din : Array of 256 or 512 bit data inputs for the DMA ToHost FIFO.
- toHostFifo_prog_full : Programmable FULL indicator, 1 bit per FIFO. The threshold can be programmed through the TOHOST_FULL_THRESH register in BAR0 which has two bitfields named THRESHOLD_-ASSERT and THRESHOLD_NEGATE. See also Table B.1

- toHostFifo_wr_clk : Clock on which toHostFifo_din is registered. Should be close or equal to 250MHz to support the nominal PCIe bandwidth.
- toHostFifo_wr_en : Assert to write into one of the FIFOs. One bit per ToHost FIFO.
- toHostFifo_rst : Assert to reset / flush the FIFO.

8.13.2.4 INTERRUPT_CALL

The input interrupt_call has the size of NUMBER_OF_INTERRUPTS - 4, because the first 4 interrupts are used by Wupper internally. Any of the other bits can be asserted to raise an MSI-X interrupt, see section 8.13.6

8.13.2.5 CLOCKS AND RESETS

- reset_hw_in : this input is used to reset the synchronizer for the register map.
- sys_reset_n : This is input should be connected to the hard reset on the PCIe edge connector (PERSTn).
- reset_soft : This output is a reset that can be triggered using a register, it is synchronized to sync_clk.
- reset_soft_appreg_clk : An unsynchronized version of reset_soft (registered at appreg_clk, 25MHz).
- sync_clk : Clock to synchronize the register map to. In FELIX this is connected to the 40 MHz BC clock.
- appreg_clk : Output of the 25 MHz PCIe slow clock on which the unsynchronized register map is running.
- sys_clk_n / sys_clk_p : 100 MHz PCIe reference clock from the PCIe edge connector.

8.13.2.6 BUSY

- master_busy_in : Used in the interrupt controller, see section 8.13.6
- tohost_busy_out : Used in circular DMA mode, the software pointer is compared to the current_address in the descriptors. If any of them is beyond a set threshold, this BUSY output is raised.
- toHostFifo_busy_out : This busy output is raised when one of the ToHost FIFOs is beyond a set programmable full threshold.

8.13.2.7 PCIE

- pcie_rxn / pcie_rxp : High speed PCIe receiver lanes
- pcie_txn / pcie_txp : High speed PCIe transmitter lanes
- sys_reset_n : This is input should be connected to the hard reset on the PCIe edge connector (PERSTn).
- sys_clk_n / sys_clk_p : 100 MHz PCIe reference clock from the PCIe edge connector.
- lnk_up : Status indication that the PCIe link is aligned.

8.13.2.8 Register Map

Wupper has an internal register map that is generated from a .yaml file. The complete set of registers is available in Appendix B. There are records called register_map_control* that contain all writable registers and self clearing trigger registers. The read only registers are gathered in register_map_monitor which is divided into sub-records of the different monitor sections, so that it is easy to drive each section from an individual functional block in the firmware.

- register_map_control_sync : Synchronized version to sync_clk of the writable/trigger registers in BAR2 of the register map, see Table B.3
- register_map_control_appreg_clk : Unsynchronized version (registered on appreg_clk, 25 MHz) with the same functionality as register_map_control_sync.
- register_map_*_monitor : Input record of the monitor registers as defined by the different monitor sections in Table B.3

8.13.3 Functional Description

Xilinx has introduced the AXI4-Stream interface [18] for the PCIe EndPoint core: a simplified version of the ARM AMBA AXI bus [23]. This interface does not contain any address lines, instead the address and other information are supplied in the header of each PCIe Transaction Layer Packet (TLP). Figure 8.54 shows the structure of the Wupper_core design. The Wupper_core is divided in two parts:

1. **DMA Control:**
   This is the entity in which the Descriptors are parsed and fed to the engine, and where the Status register of every descriptor can be read back through PCIe. Depending on the address range of the descriptor, the pointer of the current address is handled by DMA Control and incremented every time a TLP completes. DMA Control also handles the circular buffer DMA if this is requested by the descriptor (See 8.13.5).

   DMA control contains a register map, with addresses to the descriptors, status registers and external registers for the user space register map.

2. **DMA Read Write:**
   This entity contains two processes:

   - *ToHost / Add Header:* In the first process the descriptors are read and a header according to the descriptor is created. If the descriptor is a ToHost descriptor, the payload data is read from the FIFO and added after the header. This process also takes care of switching to the next active DMA descriptor, which is leading for selecting the MUX on the output ports of the ToHostFifo's.
   - *FromHost / Strip Header:* In the second process the header of the received data is removed and the length is checked; then the payload is shifted into the FIFO.

   Both processes can fire an MSI-X type interrupt by means of the interrupt controller when finished.

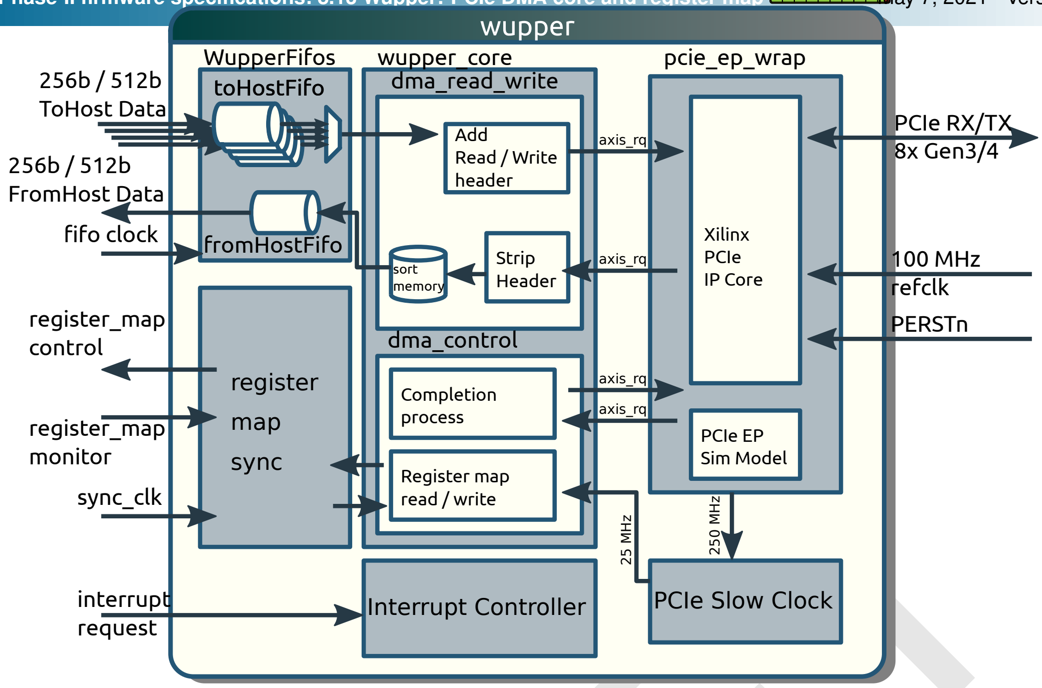


Figure 8.54: Structure of the Felix PCIe Engine.

Figure 8.54 shows a synchronization stage for the IO and external registers, The user space registers are stored and processed in the 25 MHz clock domain in order to relax timing closure of the design. The synchronization stage synchronizes the register map again to the clock used in the application design (sync_clk).

The DMA Control process always responds to a request with a certain *req_type* from the server. It responds only to IO and Memory reads and writes; for all other request types it will send an unknown request reply. If the data in the payload contains more than 128 bits, the process will send a “completion abort” reply and go back to idle state. The maximum register size has been set to 128 bits because this is a useful maximum register size; it is also the maximum payload that fits in one 250 MHz clock cycle of the AXI4-Stream interface.

The add_header process selects the descriptor and sets the ToHostFifo MUX accordingly. Based on the descriptor content, it requests a read or write to/from the server memory. If the descriptor is set to ToHost, it also initiates a FIFO read and adds the data into the payload of the PCIe TLP (Transaction Layer Packet). When the descriptor is set to FromHost this process only creates a header TLP with no payload, to request a certain amount of data from the server memory that fits in one TLP.

The DMA FromHost process checks the size of the payload against the size in the TLP header, the data will be pushed into the FromHost FIFO.

8.13.4 DMA DESCRIPTORS

Each transfer To and From Host is achieved by means of setting up descriptors on the server side, which are then processed by Wupper. The descriptors are set in the BAR0 section of the register map (see Appendix B). An extract of the descriptors and their registers is shown in Table 8.40 below. The register map in BAR0 has space for a maximum of 8 DMA descriptors, but the actual number of descriptors that are implemented is determined by the generic NUMBER_OF_DESCRIPTORS. The last active descriptor is always implemented with READ_WRITE set to 1 (read only) and the descriptors 0 to NUMBER_OF_DESCRIPTORS-2 are implemented as ToHost descriptors. The number of ToHost FIFOs is automatically determined by the same generic, as well as the ToHost FIFO depth. Setting NUMBER_OF_DESCRIPTORS to 5 (default in phase 2 FELIX) will result in 4 ToHost descriptors and FIFOs (descriptor 0..3) and a single FromHost descriptor / FIFO (descriptor 4).

Address	Name/Field	Bits	Type	Description

0x0000	DMA_DESC_0			
	END_ADDRESS	127:64	W	End Address
	START_ADDRESS	63:0	W	Start Address
0x0010	DMA_DESC_0a			
	RD_POINTER	127:64	W	server Read Pointer
	WRAP_AROUND	12	W	Wrap around
	READ_WRITE	11	R	1: FromHost/ 0: ToHost
	NUM_WORDS	10:0	W	Number of 32 bit words
...				
0x0200	DMA_DESC_STATUS_0			
	EVEN_PC	66	R	Even address cycle server
	EVEN_DMA	65	R	Even address cycle DMA
	DESC_DONE	64	R	Descriptor Done
	CURRENT_ADDRESS	63:0	R	Current Address
...				
0x0400	DMA_DESC_ENABLE	7:0	W	Enable descriptors 7:0. One bit per descriptor. Cleared when Descriptor is handled.

Table 8.40: DMA descriptors types.

Every descriptor has a set of registers, with the following specific functions:

- DMA_DESC: the register containing the start (*start_address*) and the end (*end_address*) memory addresses of a DMA transfer; both handled by the server (software API).
- DMA_DESC_a: integrates the information above by adding (i) the status of the read pointer on the server side (*rd_pointer*), (ii) the wrap around functionality enabling (*wrap_around*, see Section 8.13.5 below), (iii) the FromHost ("1") and ToHost ("0") transfer direction bit (*read_write*), and (iv) the number of 32 bits words to be transferred (*num_words*)
- DMA_DESC_STATUS: status of a specific descriptor including (i) wrap around information bits (*even_pc* and *even_dma*), (ii) completion bit (*desc_done*, (iii) DMA pointer current address (*current_address*)
- DMA_DESC_ENABLE: the descriptors enable register (*dma_desc_enable*), one bit per descriptor

8.13.5 Endless DMA with a circular buffer and wrap around

In *single shot* transfer, the DMA ToHost process continues sending data TLPs (Transaction Layer Packets) until the end address (*end_address*) is reached. The server can check the status of a certain DMA transaction by looking at the *desc_done* flag and the *current_address*. Another possible operation mode is the so- called *endless DMA*: the DMA continues its action and starts over (wrap-around) at start address (*start_address*) whenever the end address (*end_address*) is reached. The second mode is enabled by asserting the wrap-around (*wrap_around*) bit. In this mode the server has to provide another address named server pointer (*PC_read_pointer*): indicating where it has last read out the memory. After wrapping around the DMA core will transfer To Host memory until the *PC_read_pointer* is reached. The server read pointer should be updated more often than the wrap-around time of the DMA, however it should not be read too often as that would take up all the bandwidth, limiting the speed of the DMA transfer in progress. A typical rule of thumb to determine what "too often" means is that software should not update the pointer every clock cycle, but rather after processing a block of a few kB of data.

In order to determine whether Wupper is processing an address behind or in front of the server, Wupper keeps track of the number of wrap around occurrences. In the DMA status registers the even_cycle bits displays the status of the wrap-around cycle. In every even cycle (starting from 0), the bits are 0, and every wrap around the status bits will toggle. The *even_pc* bit flags a *PC_read_pointer* wrap-around, the *even_dma* a Wupper wrap-around. By looking at the wrap-around flags the server can also keep track of its own wrap-arounds. Note that while in the *endless DMA* mode (*wrap_around* bit set), the *PC_read_pointer* has to be maintained by the server (software API) and kept within the start and end address range for Wupper to function correctly. Figure 8.55 below shows a diagram of the two pointers racing each other, and the different scenarios in which they can be found with respect to each other.

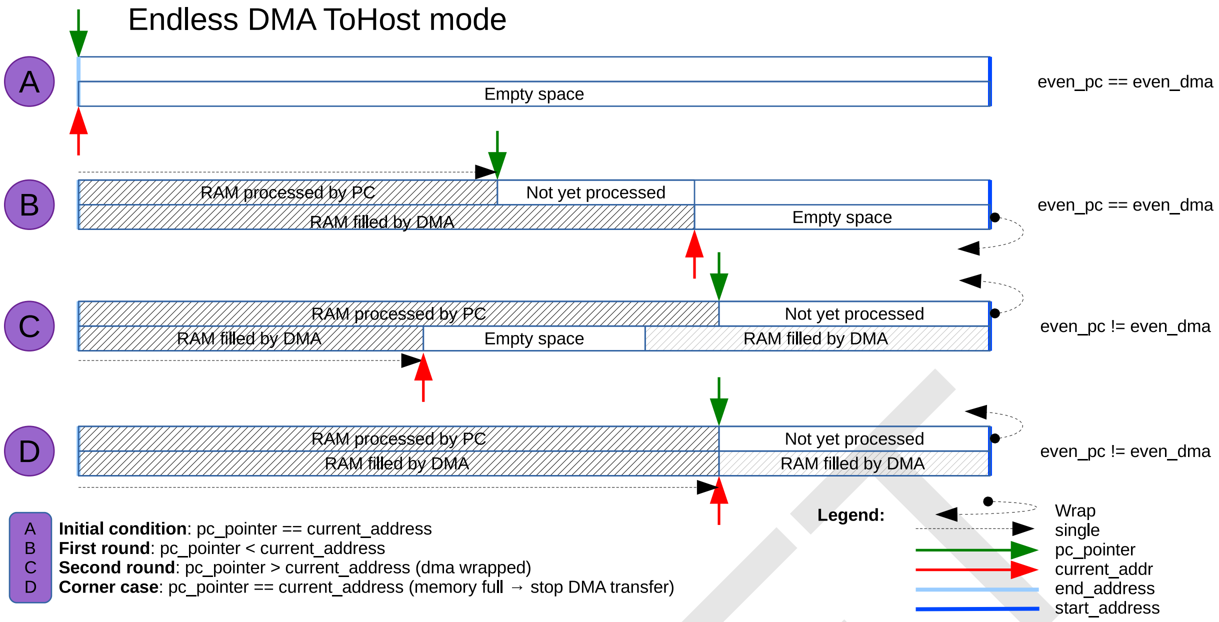


Figure 8.55: Endless DMA buffer and pointers representation diagram in ToHost mode.

Looking at Figure 8.55 above, the following scenarios can be described:

- A : start condition, both the server and the DMA have not started their operation.
- B : normal condition, the PC_read_pointer stays behind the DMA's current_address
- C : normal condition, the DMA's current_address has wrapped around and has to stay behind the PC_read_pointer
- D : the server is reading too slow, the DMA is stalled because the server read pointer is not advancing fast enough, the DMA current_address has to stay behind.

If the DMA descriptor is set to FromHost, the comparison of the even bits is inverted, as the server has to fill the buffer before it is processed in the same cycle. In this mode the *pc_read_pointer* is also maintained by the software API, however it is indicating the address up to where the server has filled the memory. In the first cycle the DMA has to stay behind the read pointer, when the server has wrapped around, the DMA can process memory up to *end_address* until it also wraps around.

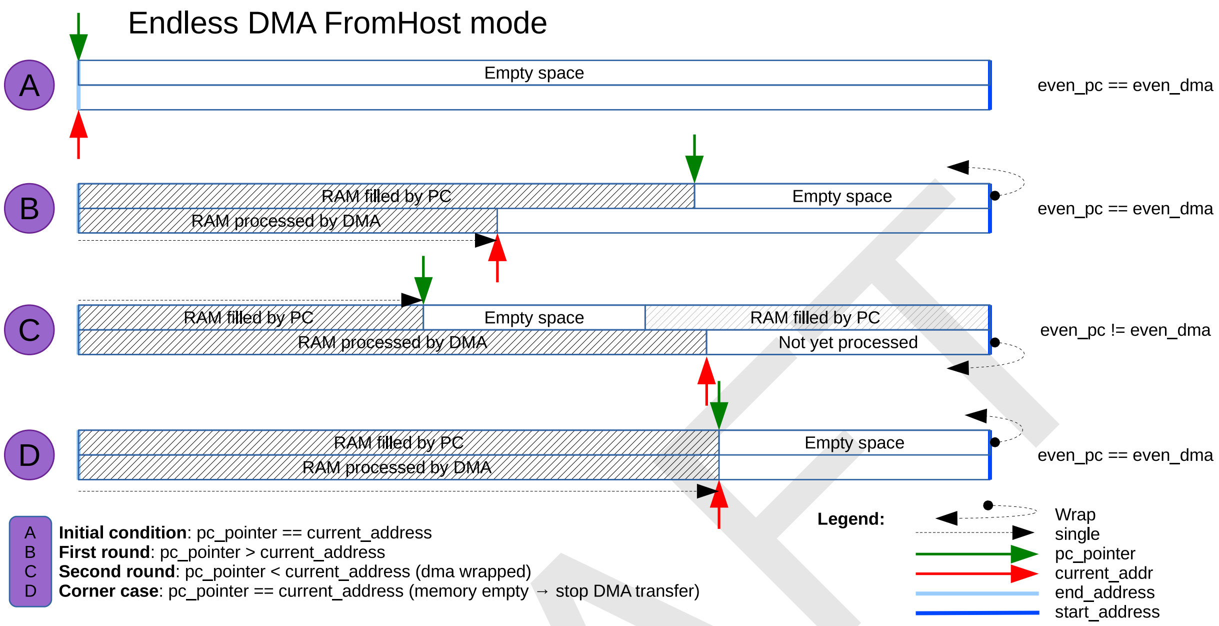


Figure 8.56: Endless DMA buffer and pointers representation diagram in FromHost mode.

Looking at Figure 8.56 above, the following scenarios can be described:

- A : start condition, both the server and the DMA have not started their operation.
- B : normal condition, the DMA's current_address stays behind the PC_read_pointer
- C : normal condition, the PC_read_pointer has wrapped around and has to stay behind the DMA's current_address
- D : the server is writing too slow, the DMA is stalled because the server read pointer is not advancing fast enough, the DMA current_address has to stay behind.

8.13.6 Interrupt controller

Wupper is equipped with an interrupt controller supporting the MSI-X (Message Signaled Interrupt eXtended) as described in "Chapter 17: Interrupt Support" page 812 and onwards of [**PCIe_technology**]. In particular the chapter and tables in "MSI-X Capability Structure".

The MSI-X Interrupt table contains eight interrupts; this number can be extended by a generic parameter in the firmware. All interrupts are mapped to the data_available interrupt of the corresponding ToHost descriptor, formerly known as interrupt number 2. All the other interrupt sources have been removed since multiple ToHost descriptors were introduced. The interrupts are detailed in Table 8.41.

Table 8.41: PCIe interrupts.

Interrupt	Name	Description
0	ToHost 0 Available	Fired when data becomes available in the ToHost FIFO 0 (falling edge of ToHostFifoProgEmpty)
1	ToHost 1 Available	Fired when data becomes available in the ToHost FIFO 1 (falling edge of ToHostFifoProgEmpty)
1	ToHost 2 Available	Fired when data becomes available in the ToHost FIFO 2 (falling edge of ToHostFifoProgEmpty)
3	ToHost 3 Available	Fired when data becomes available in the ToHost FIFO 3 (falling edge of ToHostFifoProgEmpty)
4	reserved	
5	crDownXoff	ToHost combined full flags (CR xoff)
6	BUSY change	Fired when the busy LEMO signal changes
7	ToHost Full	Fired when the ToHost FIFO becomes full

All Interrupts are fired when enough data has arrived in the ToHost fifo to fill at least one TLP of data. Once an interrupt has fired, it will not produce an additional interrupt until the SW_POINTER has been updated by the software.

All the interrupts can also be fired from the register INT_TEST, by setting the bitfield IRQ to the desired interrupt number. This write action will fire a single interrupt.

8.13.7 Xilinx PCIe EndPoint Core

Wupper was built around the interface of the Virtex-7 FPGA Gen3 Integrated Block for PCI Express v4.3 [19], and was later ported to other Xilinx PCIe hard blocks:

- Virtex-7 FPGA Gen3 Integrated Block for PCI Express [19]. Wupper was tested on Virtex7 with the VC709 (FLX709) board and the HTG710 (FLX710) boards using the XC7VX690T FPGA. (PCIe Gen3x8)
- UltraScale Devices Gen3 Integrated Block for PCI Express [20]. Wupper was tested with the BNL711 (FLX711) and BNL712 (FLX712) boards, using the KU115 FPGA. (2x PCIe Gen3x8 with a PCIe x16 switch)
- UltraScale+ Devices Integrated Block for PCI Express [21]. Wupper was tested with the VCU128-es1 (FLX128) (VU37P FPGA), the XUPP3R (VU9P FPGA) (FLX800) and the BNL801 board (FLX801) (VU9P FPGA) 2x PCIe Gen4x8 bifurcated. [2]
- Versal ACAP Integrated Block for PCI Express [22]. Wupper was tested on the VMK180 board (VM1802 ACAP), PCIe Gen4x8

This core is using a PCIe hard block in the Virtex-7 FPGA. The hard block is equipped with an AXI4-Stream interface.

[2]For the VU9P FPGA, PCIe Gen4 is not officially supported, but it was demonstrated to work. It can be enabled only on Vivado 2018.1 using a tcl command or by editing the .xci file

8.13.7.1 Xilinx AXI4-Stream interface

The interface has the advantage that it has two separate bidirectional AXI4-Stream interfaces. The two interfaces are the requester interface, with which the FPGA issues the requests and the PC replies, and the completer interface where the PC takes initiative.

bus	Description	Direction
`axis_rq`	**R**equester re**Q**uest. This interface is used for DMA, the FPGA takes the initiative to write to this AXI4-Stream interface and the PC has to answer.	FPGA → PC
`axis_rc`	**R**equester **C**ompleter. This interface is used for DMA reads (from PC memory to FPGA), this interface also receives a reply message from the PC after a DMA write.	PC → FPGA
`axis_cq`	**C**ompleter re**Q**uest. This interface is used to write the DMA descriptors as well as some other registers.	PC → FPGA
`axis_cc`	**C**ompleter **C**ompleter. This interface is used as a reply inteface for register reads, as well as a reply header for a register write.	FPGA → PC

Table 8.43: AXI4-Stream streams.

8.13.7.2 Configuration of the core

The Xilinx PCIe EndPoint core is configured as a PCI express Gen3 (8.0GT/s) or Gen4 (16.0GT/s) End Point with 8 lanes and the Physical Function (PF0) max payload size is set to 1024 bytes. AXI-ST Frame Straddle is disabled and the client tag is enabled. All other options are set to default, the reference clock frequency is 100MHz and the only option for the AXI4-Stream interface is 256 (512 for Gen4) bit at 250MHz.

8.13.8 Status Indicators

Apart from the lnk_up indicator, indicating that the link is up, all status indicators are described in the register map in B.3

8.13.9 Latency

It is difficult to give a single figure for the latency of the Wupper core, because the DMA latency involves the PCIe operation and is highly dependent on the type of server used.

8.13.10 Error Handling

Error handling is performed through the PCIe standard error messages, as well as status registers in the registermap, see B.3.

8.13.11 Estimated Resource Usage

The estimated resource usage of Wupper, including register map 5.0 can be found in Table 8.44. For cards with two endpoints, the resource count must be multiplied by 2, this applies to both the FLX712 and the FLX128 cards.

	KCU115 / FLX712						VU37P / FLX128					
	LUT		FF		BRAM		LUT		FF		BRAM	
Wupper	30094	4.54%	59706	4.50%	47	2.18%		%		%		%
WupperFifos	3007	0.45%	2275	0.17%	34	1.57%		%		%		%
dma_read_write	1068	0.16%	1788	0.13%	4	0.19%		%		%		%
dma_control	9864	1.49%	27026	2.04%	0	0.00%		%		%		%

pcie_ep_wrap	1606	0.24%	5056	0.38%	9	0.42%		%		%		%
register_map_sync	14221	2.14%	22631	1.71%	0	0.00%		%		%		%
intr_ctrl	319	0.05%	893	0.07%	0	0.00%		%		%		%

Table 8.44: Wupper Resource utilization.

8.13.12 Simulation

The directory *firmware/simulation/Wupper* contains all necessary testbenches (wupper_tb.vhd, pcie_ep_-sim_model.vhd) to run the simulation in Mentor Graphics Modelsim or Questasim [**questasim**].

The directory simulation/UVVMExample contains a file modelsim.ini with some standard information, there is also a script "ci.sh" wich will execute the UVVM based simulation. It assumes that questasim 2019.1 is installed, the Xilinx libraries are compiled in simulation/xilinx_lib and the UVVM library is compiled in simulation/UVVM. The wupper simulation can be started by executing

Listing 8.3: Run the simulation.

```
cd FELIX/firmware/simulation/UVVMExample
./ci.sh Wupper
```

By default the simulation starts in command line mode. If GUI mode is desired (e.g. to view waveforms), the ci.sh script can be edited, and the "-c" parameter from the vsim command can be removed.

8.14 RDMA

8.14.1 Introduction

8.14.2 Interfaces

8.14.2.1 Overview

8.14.2.2 Interface to component 2

8.14.3 Functional Description

8.14.4 Configuration

8.14.5 Status Indicators

8.14.6 Latency

8.14.7 Error Handling

8.14.8 Estimated Resource Usage

8.15 HouseKeeping

8.15.1 Introduction

8.15.2 Interfaces

8.15.2.1 Overview

8.15.2.2 Interface to component 2

8.15.3 Functional Description

8.15.4 Configuration

8.15.5 Status Indicators

8.15.6 Latency

8.15.7 Error Handling

8.15.8 Estimated Resource Usage

9

RADIATION TOLERANCE

Remark 9.1: *Instructions for this chapter*

If some level of radiation tolerance is required, state the minimum total ionizing dose (TID) and non-ionizing fluence after which the component must remain functional and meet all other specifications. Also, describe the allowable single event upset (SEU) rate, possibly by sections of the component if the allowable rate is not the same across the whole component. If there are other special requirements beyond those covered in other sections describe them here. Technique used in firmware to mitigate SEU, e.g. triple mode redundancy or soft error mitigation etc., should be described. If there are no radiation tolerance or other special requirements, then state "Not Applicable" for this section.

Not Applicable

10

Testing, Validation and Commissioning

Firmware is tested in 3 ways within the FELIX project:

- Simulation (In Gitlab CI)
- Automated build (In Gitlab CI)
- On FELIX hardware using nightly tests

10.1 Simulation

The FELIX firmware has many different flavours and configurations. It is unrealistic to create a single simulation testbench to cover the complete picture. There are also parts of the FELIX firmware that are very difficult to simulate. The PCIe interface for instance can be simulated using BFM (Bus Functional Models) if they are available, but the complete behaviour of PCIe operation including the software, PCIe enumeration, register reads / writes, DMA and interrupts would be nearly impossible to simulate. The Xilinx PCIe IP core was therefore modelled by a simulation model that emulates a realistic FELIX operation, indluding register writes and DMA. The high speed interfaces are not modelled, but instead the model is directly emulating the axi4 stream interfaces as documented in the Xilinx IP core documentation [19] [20] [21] [22]

The FELIX team is therefore not trying to simulate the individual blocks, as well chains of blocks exercising different scenarios in the operation of the FELIX firmware. Breaking down the firmware into blocks for simulation sets some constraints on the firmware design:

- The blocks must have a well defined interface, and where possible, industry standard interfaces must be used.
  - For the interface between the different encoders, decoders and both directions of the Central Routers, we have chosen to use AXI4 stream, which can be modeled using existing BFM entities.
  - Between the Central Routers and Wupper (PCIe DMA) a standard 256 or 512 bit wide FIFO interface has been defined, depending on the PCIe speed (Gen3 or Gen4).
  - The interfaces between the Link Wrapper and Encoder / Decoder will be arrays of std_logic_-vector, as these types are already used by the upstream GBT and LpGBT design, and by the transceiver wrapper for FULL mode. An exception is the transceiver for 25G Interlaken, which will communicate through AXI4 stream.

- Bus Functional Models (BFM) must be used to model the interfaces of the different blocks. Where possible the BFM models from standard libraries should be used, but FELIX / ATLAS specific models will have custom BFMs.
- The developer of a block is responsible for a complete coverage of the block by the testbench.

10.1.1 UVVM

Structural testbenches with good coverage are difficult to make. To ease the process, a simulation library can be used. The FELIX team has studied several simulation libraries and as a result we have chosen UVVM. [24]. The UVVM library can be used in different ways. In the most simple way, only the uvvm_utility library is used, which gives access to a set of functions to verify signals, report errors and generated clocks and other types of waveforms. A more advanced utilization of the UVVM library is to use the VVC library, which is a structured and high level way to describe functional models. Both strategies have been used by the several testbenches in the FELIX project, depending on the preferences of the developer of the block and what had previously been implemented before UVVM was introduced in FELIX.

Independent of the used UVVM strategy, the result of the testbench for every block is a simple report that summarizes the simulation results, counts the number of warnings and errors and gives a pass / fail result which can be used in Gitlab CI, see Figure 10.1

```
# UVVM:     ==========================================================================================
# UVVM:     *** FINAL SUMMARY OF ALL ALERTS ***
# UVVM:     ==========================================================================================
# UVVM:                               REGARDED   EXPECTED  IGNORED      Comment?
# UVVM:               NOTE         :     0          0         0         ok
# UVVM:               TB_NOTE      :     0          0         0         ok
# UVVM:               WARNING      :     0          0         0         ok
# UVVM:               TB_WARNING   :     1          0         0         *** TB_WARNING ***
# UVVM:               MANUAL_CHECK :     0          0         0         ok
# UVVM:               ERROR        :     0          0         0         ok
# UVVM:               TB_ERROR     :     0          0         0         ok
# UVVM:               FAILURE      :     0          0         0         ok
# UVVM:               TB_FAILURE   :     0          0         0         ok
# UVVM:     ==========================================================================================
# UVVM:     >> Simulation SUCCESS: No mismatch between counted and expected serious alerts
# UVVM:     ==========================================================================================
# UVVM:
# UVVM:
# UVVM:
# UVVM:
# UVVM: ID_LOG_HDR                        989.0 ns  CRFromHostAxis_tb            SIMULATION COMPLETED
# UVVM: ------------------------------------------------------------------------------------------------
```

Figure 10.1: Results summary of a UVVM successful simulation.

Requirement 10.1: *UVVM Testbenches*

Every functional block inside the FELIX firmware that can be modelled must be covered by at least one UVVM testbench.

10.2 Gitlab CI

The Gitlab CI pipeline for the FELIX Phase II firmware knows 2 stages: Simulation and Build. In the Simulation stage, all the testbenches (UVVM) will be executed in parallel, the transcripts are available as an artefact.

In the Build stage, FPGA bitfiles for all the active firmware flavours will be produced for the FLX712 hardware platform. Currently the following bitfiles will be produced this way:

- FULL mode 24 channels for FLX712

- GBT mode 8 channels for FLX712
- PIXEL/lpGBT mode 24 channels for FLX712

Other firmware flavours will soon be added to the CI build as soon as build scripts are available. A typical pipeline for phase2/firmware CI is shown in Figure 10.2

Requirement 10.2: *CI Simulation*

For every commit, the simulation testbenches as described in Section 10.1 will be executed by Gitlab CI.

Requirement 10.3: *CI Build*

For merge requests and commits to master and phase2/master branches, every active firmware flavour will be built by Gitlab CI to produce a bitfile. A finished CI pipeline is required before a branch can be merged. Additionally an automated test on hardware will be executed as a requirement for a merge request.

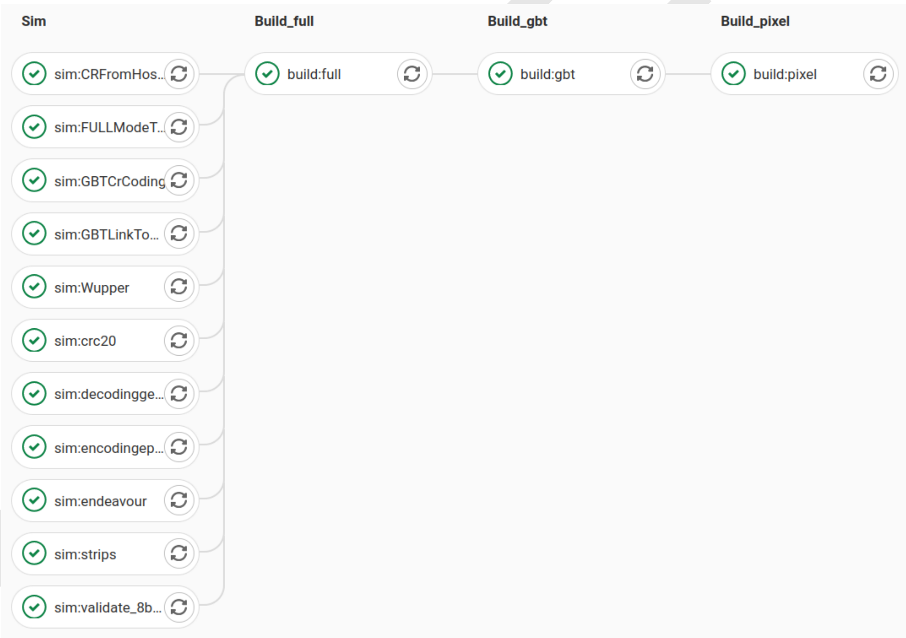


Figure 10.2: Continuous Integration Pipelines as seen in the Gitlab interface.

10.3 Nightly firmware test on hardware

Besides simulation and automated builds, a third way of testing is automatically performed: Nightly firmware tests. The nightly tests are a set of tests that are performed automatically on a FELIX hardware platform (FLX709 or FLX712), and the set of tests depends on the firmware flavour. The nightly firmware tests are not triggered from Gitlab CI, but rather run at night. This way the test system is available at daytime for other developments. The nightly tests involve a frontend emulator, the FELIX PCIe card, the FELIX server and will be extended in the future with a data handler.

The set of tests is available in the following git repository:
https://gitlab.cern.ch/atlas-tdaq-felix/flx-firmware-tester

The results of the nightly tests are published on the following web interface:
https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/nightly/

11

Firmware Management and Reliability Matters

11.1 Firmware Source Management and Release Plan

Remark 11.1: *Instructions for this section*

Describe the firmware source code management plan, track of issues and version control, e.g. git repository etc. Describe the firmware release plan, as required by different end users, e.g. hardware testing and system integration etc.

START YOUR INPUT HERE. REMOVE THE ABOVE.

11.2 Consequences of Failures

Remark 11.2: *Instructions for this section*

Describe the consequences to the detector of a failure of one unit of this component, e.g. x% of the sub-detector channels will be lost. The severity of the consequences will determine the level of reliability required and the level to be validated. This section and following section **??** and section **??** mainly apply to the boards exposed to the radiation.

START YOUR INPUT HERE. REMOVE THE ABOVE.

11.3 Prior Knowledge of Expected Reliability

Remark 11.3: *Instructions for this section*

Based upon industry experience, collaboration experience or personal experience, give an estimate of the reliability of this firmware, e.g. cross section of the SEU rate and expected upset rate during operation.

START YOUR INPUT HERE. REMOVE THE ABOVE.

11.4 Measures Proposed to Ensure Reliability of the Firmware

Remark 11.4: *Instructions for this section*

Include such measures as conservative design techniques (give specific examples), redundancy and possibilities to replace failed part. If failed part could be replaced, estimate the difficulty and time involved for installing replacements.

START YOUR INPUT HERE. REMOVE THE ABOVE.

12

ORGANIZATION OF FIRMWARE DEVELOPMENT

Remark 12.1: *Instructions for this chapter*

Include a list of development teams from different institutes if this applies, and responsibility of different firmware development tasks described in chapter 8 for each team.

This section presents a rough but reasonable estimate of the duration for the first prototyping phase. Again, this is subjected to change depending on available person power and other factors. The major works expected for the first prototyping phase may include evolving the firmware to different hardware platforms, restructuring firmware block, implementing new protocols and so on.

Porting Phase-I firmware to different hardware platforms necessarily involves working with some new types of links (e.g PCIe 3 to PCIe 4). This requires some changes in the Wupper Core. Fortunately this has been implemented for 99% now and it is being verified for the Xilinx VU9P FPGA (BNL DUNE card). For the whole implementation of new protocols (lpGBT, Aurora, 6b/8b etc.) and especially the reorganization of the block diagram, a longer time scale is needed, say about 1-2 years towards the first working firmware.

Table 12.1 shows a rough estimation on the different parts, with time and number of FTE needed.

Table 12.1: The time and FTE estimation for the first firmware prototyping phase..

Task	Time (months)	FTE
Adjusting minimal design to the new hardware	3	2
Fixing Wupper for new hardware	1	1
Implement lpGBT (current hardware)	done	done
Implement lpGBT (new hardware)	1	1
Implement new block diagram framework:		
Toplevel and connections	2	2
Central router ToHost (routing, no decoding / encoding)	3	2
Central router ToHost (decoding):		
Aurora	2	2
8b/10b	1	2
HDLC	1	2
Central router FromHost (routing)	3	2
Central router FromHost (encoding):		
Trickle merge/high priority config.	2	2
8b/10b	1	1
6b/8b	1	1
HDLC	1	1
8b/10b	1	1
TTC	0.5	1
ITk Pixel custom protocol	2	1
TTC / PON	2	2
TTCC emulator	1	1
GBT/lpGBT/FW wrapper	2	2
Internal emulator	2	1

References

[1] Argonne National Lab. *FELIG User Manual*. URL: https://gitlab.cern.ch/atlas-tdaq-felix/felig-tools/-/blob/FELIG_Scripts/FeligFLX712Introduction.pdf (cit. on p. 4).

[2] Nikhef, Radboud University Nijmegen. *FMEMU, Documentation needed*. URL: https://gitlab.cern.ch/atlas-tdaq-felix/firmware/ (cit. on p. 4).

[3] Nikhef, Radboud University Nijmegen. *FELIX_MROD, Documentation needed*. URL: https://gitlab.cern.ch/atlas-tdaq-felix/firmware/ (cit. on p. 4).

[4] K. Chen et al. "A Generic High Bandwidth Data Acquisition Card for Physics Experiments". In: *IEEE Transactions on Instrumentation and Measurement* 69.7 (2020), pp. 4569–4577. DOI: 10.1109/TIM.2019.2947972 (cit. on p. 11).

[5] Xilinx. *UltraScale FPGA Product Tables and Product Selection Guide*. URL: https://www.xilinx.com/support/documentation/selection-guides/ultrascale-fpga-product-selection-guide.pdf (cit. on p. 14).

[6] Xilinx. *UltraScale+ FPGA Product Tables and Product Selection Guide*. URL: https://www.xilinx.com/support/documentation/selection-guides/ultrascale-fpga-product-selection-guide.pdf (cit. on p. 14).

[7] Frans Schreuder. *Tool to create block diagrams from VHDL entities*. URL: https://github.com/fransschreuder/entity-block (cit. on p. 23).

[8] Aliaksei Chapyzhenka. *Tool to create waveforms*. URL: https://wavedrom.com/ (cit. on pp. 26, 29, 110, 111).

[9] Wikipedia. *High-Level Data Link Control*. URL: {https://en.wikipedia.org/wiki/High-Level_Data_Link_Control} (cit. on pp. 48, 49, 88).

[10] Julian Maxime Mendez. *GBT-SC module for FPGA*. URL: {https://gitlab.cern.ch/gbtsc-fpga-support/gbt-sc} (cit. on p. 48).

[11] CERN GBT Project. "The GBTx Manual". In: V0.14 (2016). URL: https://espace.cern.ch/GBT-Project/GBTX/Manuals/gbtxManual.pdf (cit. on pp. 51, 93).

[12] TTC group. "CERN TTC homepage". In: (). URL: http://ttc.web.cern.ch/TTC (cit. on p. 91).

[13] LpGBT-FPGA. 2018. URL: http://lpgbt-fpga.web.cern.ch/doc/html/index.html (cit. on pp. 96–98).

[14] K. Chen et al. "Optimization on fixed low latency implementation of the GBT core in FPGA". In: *Journal of Instrumentation* 12.07 (2017), P07011–P07011. DOI: 10.1088/1748-0221/12/07/p07011. URL: https://doi.org/10.1088/1748-0221/12/07/p07011 (cit. on p. 98).

[15] Xilinx. *7 Series FPGAs GTX/GTH Transceivers User Guide (UG476 v1.11.1)*. 2015. URL: http://www.xilinx.com/support/documentation/user_guides/ug476_7Series_Transceivers.pdf (cit. on p. 98).

[16] P. Farthouat and D. Francis and F. Lanni and T. Pauly. *ATLAS Trigger and DAQ Interfaces with Detector Front-End Systems: Requirement Document for HL-LHC*. URL: {https://edms.cern.ch/ui/file/1563801/1/RequirementsPhaseII_v1.1.0.pdf} (cit. on p. 103).

[17] The FELIX team. *atlas-tdaq-felix website*. URL: https://atlas-project-felix.web.cern.ch/atlas-project-felix/ (cit. on p. 113).

[18] Xilinx. *UG761: Xilinx AXI Bus documentation*. URL: http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf (cit. on pp. 113, 117).

[19] Xilinx. *Virtex-7 FPGA Gen3 Integrated Block for PCI Express v4.3*. URL: https://www.xilinx.com/support/documentation/ip_documentation/pcie3_7x/v4_3/pg023_v7_pcie_gen3.pdf (cit. on pp. 113, 123, 129).

[20] Xilinx. *UltraScale Devices Gen3 Integrated Block for PCI Express v4.4.* URL: https://www.xilinx.com/support/documentation/ip_documentation/pcie3_ultrascale/v4_4/pg156-ultrascale-pcie-gen3.pdf (cit. on pp. 113, 123, 129).

[21] Xilinx. *UltraScale+ Devices Integrated Block for PCI Express v1.3.* URL: https://www.xilinx.com/support/documentation/ip_documentation/pcie4_uscale_plus/v1_3/pg213-pcie4-ultrascale-plus.pdf (cit. on pp. 113, 123, 129).

[22] Xilinx. *Versal ACAP Integrated Block for PCI Express v1.0.* URL: https://www.xilinx.com/support/documentation/ip_documentation/pcie_versal/v1_0/pg343-pcie-versal.pdf (cit. on pp. 113, 123, 129).

[23] ARM. "ARM AMBA AXI bus standard specification page". In: (). URL: http://www.arm.com/products/system-ip/amba/amba-open-specifications.php (cit. on p. 117).

[24] Ali Skaf. *FELIX Standardized Firmware testbench with Gitlab CI.* URL: {https://indico.cern.ch/event/858260/contributions/3613811/attachments/1930907/3198159/ASkaf_QT5r1.pdf} (cit. on p. 130).

[25] FELIX team. *FELIX Data format.* URL: {https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/felix-user-manual/versions/Latest/C_datastructures.html#_13__guide_to_felix_data_structures} (cit. on p. B.29).

Appendix A

Code Management

Appendix B

APPENDIX

B.1 FELIX REGISTER MAP, VERSION 5.0

Starting from the offset address of BAR0, BAR1 and BAR2. BAR0 only contains registers associated with DMA.

Bar0					
DMA_DESC					
0x0000	0,1	DMA_DESC_0			
		END_ADDRESS	127:64	W	End Address
		START_ADDRESS	63:0	W	Start Address
0x0010	0,1	DMA_DESC_0a			
		SW_POINTER	127:64	W	Pointer controlled by the software, indicating read or write status for circular DMA
		WRAP_AROUND	12	W	Wrap around
		FROMHOST	11	R	1: fromHost/ 0: toHost
		NUM_WORDS	10:0	W	Number of 32 bit words
...					
0x00E0	0,1	DMA_DESC_7			
		END_ADDRESS	127:64	W	End Address
		START_ADDRESS	63:0	W	Start Address
0x00F0	0,1	DMA_DESC_7a			
		SW_POINTER	127:64	W	Pointer controlled by the software, indicating read or write status for circular DMA
		WRAP_AROUND	12	W	Wrap around
		FROMHOST	11	R	1: fromHost/ 0: toHost
		NUM_WORDS	10:0	W	Number of 32 bit words
DMA_DESC_STATUS					
0x0200	0,1	DMA_DESC_STATUS_0			
		EVEN_PC	66	R	Even address cycle PC
		EVEN_DMA	65	R	Even address cycle DMA
		DESC_DONE	64	R	Descriptor Done
		FW_POINTER	63:0	R	Pointer controlled by the firmwarre, indicating where the DMA is busy reading or writing
...					
0x0270	0,1	DMA_DESC_STATUS_7			
		EVEN_PC	66	R	Even address cycle PC
		EVEN_DMA	65	R	Even address cycle DMA
		DESC_DONE	64	R	Descriptor Done
		FW_POINTER	63:0	R	Pointer controlled by the firmwarre, indicating where the DMA is busy reading or writing

0x0300	0,1	BAR0_VALUE	31:0	R	Copy of BAR0 offset reg.
0x0310	0,1	BAR1_VALUE	31:0	R	Copy of BAR1 offset reg.
0x0320	0,1	BAR2_VALUE	31:0	R	Copy of BAR2 offset reg.
0x0400	0,1	DMA_DESC_ENABLE	7:0	W	Enable descriptors 7:0. One bit per descriptor. Cleared when Descriptor is handled.
0x0420	0,1	DMA_RESET	any	T	Reset Wupper Core (DMA Controller FSMs)
0x0430	0,1	SOFT_RESET	any	T	Global Software Reset. Any write resets applications, e.g. the Central Router.
0x0440	0,1	REGISTER_RESET	any	T	Resets the register map to default values. Any write triggers this reset.
0x0450	0,1	FROMHOST_FULL_THRESH			
		THRESHOLD_ASSERT	22:16	W	Assert value of the FromHost programmable full flag
		THRESHOLD_NEGATE	6:0	W	Negate value of the FromHost programmalbe full flag
0x0460	0,1	TOHOST_FULL_THRESH			
		THRESHOLD_ASSERT	27:16	W	Assert value of the ToHost programmable full flag
		THRESHOLD_NEGATE	11:0	W	Negate value of the ToHost programmalbe full flag
0x0470	0,1	BUSY_THRESHOLD_ASSERT	63:0	W	Tohost or Fromhost busy will be asserted in circular DMA mode when the server PC buffer gets full (space below ASSERT threshold)..
0x0480	0,1	BUSY_THRESHOLD_NEGATE	63:0	W	Tohost or Fromhost busy will be negated in circular DMA mode when the server PC buffer gets less full (space above NEGATE threshold).
0x0490	0,1	BUSY_STATUS	0	R	A tohost descriptor passed BUSY_THRESHOLD_ASSERT, busy flag set
0x04A0	0,1	PC_PTR_GAP	63:0	W	This is the minimum value that the pc_pointer in a descriptor has to decrease in order to flip the evencycle_pc bit

Table B.1: FELIX register map BAR0.

BAR1 stores registers associated with the Interrupt vector.

Bar1					
INT_VEC					
0x0000	0,1	INT_VEC_0			
		INT_CTRL	127:96	W	Interrupt Control
		INT_DATA	95:64	W	Interrupt Data
		INT_ADDRESS	64:0	W	Interrupt Address
...					
0x00F0	0,1	INT_VEC_15			
		INT_CTRL	127:96	W	Interrupt Control
		INT_DATA	95:64	W	Interrupt Data
		INT_ADDRESS	64:0	W	Interrupt Address
0x0100	0,1	INT_TAB_ENABLE	7:0	W	Interrupt Table enable Selectively enable Interrupts

Table B.2: FELIX register map BAR1.

BAR2 stores registers for the control and monitor of HDL modules inside the FPGA other than Wupper. A portion of this register map's section is dedicated for control and monitor of devices outside the FPGA; as for example simple I2C devices.

Bar2					
Generic Board Information					
0x0000	0,1	REG_MAP_VERSION	15:0	R	Register Map Version, 5.0 formatted as 0x0500
0x0010	0,1	BOARD_ID_TIMESTAMP	39:0	R	Board ID Date / Time in BCD format YYMMDDhhmm
0x0030	0,1	GIT_COMMIT_TIME	39:0	R	Board ID GIT Commit time of current revision, Date / Time in BCD format YYMMDDhhmm
0x0040	0,1	GIT_TAG	63:0	R	String containing the current GIT TAG
0x0050	0,1	GIT_COMMIT_NUMBER	31:0	R	Number of GIT commits after current GIT_TAG
0x0060	0,1	GIT_HASH	31:0	R	Short GIT hash (32 bit)
0x0070	0,1	STATUS_LEDS	7:0	W	Board GPIO Leds
0x0080	0,1	GENERIC_CONSTANTS			
		INTERRUPTS	15:8	R	Number of Interrupts
		DESCRIPTORS	7:0	R	Number of Descriptors
0x0090	0,1	NUM_OF_CHANNELS	7:0	R	Number of GBT or FULL mode Channels
0x00A0	0,1	CARD_TYPE	63:0	R	Card Type: - 709 (0x2c5): FLX709, VC709 - 710 (0x2c6): FLX710, HTG710 - 711 (0x2c7): FLX711, BNL711 - 712 (0x2c8): FLX712, BNL712 - 128 (0x080): FLX128, VCU128
0x00C0	0,1	GENERATE_GBT	0	R	1 when the GBT Wrapper is included in the design
0x00D0	0,1	OPTO_TRX_NUM	7:0	R	Number of optical transceivers in the design
0x00E0	0,1	GENERATE_TTC_EMU	1	R	1 when TTC emulator is generated
INCLUDE_EGROUPS					
0x0100	0,1	INCLUDE_EGROUP_0			
		FROMHOST_02	8	R	FromHost EPROC02 is included in this EGROUP
		FROMHOST_04	7	R	FromHost EPROC04 is included in this EGROUP
		FROMHOST_08	6	R	FromHost EPROC8 is included in this EGROUP
		FROMHOST_HDLC	5	R	FromHost HDLC is included in this EGROUP
		TOHOST_02	4	R	ToHost EPROC02 is included in this EGROUP
		TOHOST_04	3	R	ToHost EPROC04 is included in this EGROUP
		TOHOST_08	2	R	ToHost EPROC08 is included in this EGROUP
		TOHOST_16	1	R	ToHost EPROC16 is included in this EGROUP
		TOHOST_HDLC	0	R	ToHost HDLC is included in this EGROUP
...					
0x0160	0,1	INCLUDE_EGROUP_6			
		FROMHOST_02	8	R	FromHost EPROC02 is included in this EGROUP
		FROMHOST_04	7	R	FromHost EPROC04 is included in this EGROUP
		FROMHOST_08	6	R	FromHost EPROC8 is included in this EGROUP
		FROMHOST_HDLC	5	R	FromHost HDLC is included in this EGROUP
		TOHOST_02	4	R	ToHost EPROC02 is included in this EGROUP
		TOHOST_04	3	R	ToHost EPROC04 is included in this EGROUP
		TOHOST_08	2	R	ToHost EPROC08 is included in this EGROUP
		TOHOST_16	1	R	ToHost EPROC16 is included in this EGROUP
		TOHOST_HDLC	0	R	ToHost HDLC is included in this EGROUP
0x0170	0,1	WIDE_MODE	0	R	GBT is configured in Wide mode

0x0190	0,1	FIRMWARE_MODE	3:0	R	0: GBT mode 1: FULL mode 2: LTDB mode (GBT mode with only IC and TTC links) 3: FEI4 mode 4: ITK Pixel 5: ITK Strip 6: FELIG 7: FULL mode emulator 8: FELIX_MROD mode 9: lpGBT mode
0x01A0	0,1	GTREFCLK_SOURCE	1:0	R	0: Transceiver reference Clock source from Si5345 1: Transceiver reference Clock source from Si5324 2: Transceiver reference Clock from internal BUFG (GREFCLK)
0x01B0	0,1	CR_GENERICS			
		XOFF_INCLUDED	2	R	Xoff bits (usually full mode) can be generated by the FromHost Central Router
		DIRECT_MODE_INCLUDED	1	R	Indicates that the Direct mode functionality was built in the Central Router
		FROM_HOST_INCLUDED	0	R	Indicates that the From Host path of the Central router was included in the design
0x01C0	0,1	BLOCKSIZE	15:0	R	Number of bytes in a block
0x01D0	0,1	PCIE_ENDPOINT	0	R	Indicator of the PCIe endpoint on BNL71x cards with two endpoints. 0 or 1
0x01E0	0,1	CHUNK_TRAILER_32B	0	R	Indicator that the chunk trailer is in the new 32-bit format
0x01F0	0,1	PCIE_ENDPOINTS	1:0	R	Number of PCIe endpoints on the card. The BNL71x cards have 2 endpoints
0x0200	0,1	SUPER_CHUNK_FACTOR	7:0	R	Number of full mode chunks glued together as one chunk
CR To Host Controls And Monitors					
0x0800	0,1	TIMEOUT_CTRL			
		ENABLE	32	W	1 enables the timout trailer generation for ToHost mode
		TIMEOUT	31:0	W	Number of 40 MHz clock cycles after which a timeout occurs.
0x0810	0,1	MAX_TIMEOUT	31:0	R	Maximum allowed timeout value
0x0820	0,1	CRTOHOST_FIFO_STATUS			
		CLEAR	any	T	Any write to this register clears the latched FULL flags
		FULL	47:24	R	Every bit represents the full flag of a channel FIFO
		FULL_LATCHED	23:0	R	like FULL but a latched state, clear by writing to this register
CR From Host Controls And Monitors					
0x1000	0,1	CRFROMHOST_FIFO_STATUS			
		CLEAR	any	T	Any write to this register clears the latched FULL flags
		FULL	47:24	R	Every bit represents the full flag of a channel FIFO
		FULL_LATCHED	23:0	R	like FULL but a latched state, clear by writing to this register
BROADCAST_ENABLE_GEN					
0x1010	0,1	BROADCAST_ENABLE_00	41:0	W	Enable path to be included in a broadcast message.
. . .					
0x1180	0,1	BROADCAST_ENABLE_23	41:0	W	Enable path to be included in a broadcast message.
Decoding Controls And Monitors					
PATH_HAS_STREAM_ID					
0x2000	0,1	LINK_00_HAS_STREAM_ID			
		EGROUP6	55:48	W	EPATH (Wide mode or lpGBT) is associated with a STREAM ID
		EGROUP5	47:40	W	EPATH (Wide mode or lpGBT) is associated with a STREAM ID
		EGROUP4	39:32	W	EPATH is associated with a STREAM ID
		EGROUP3	31:24	W	EPATH is associated with a STREAM ID
		EGROUP2	23:16	W	EPATH is associated with a STREAM ID
		EGROUP1	15:8	W	EPATH is associated with a STREAM ID
		EGROUP0	7:0	W	EPATH is associated with a STREAM ID, use only bit0 for FULL mode.
. . .					

0x2170	0,1	LINK_23_HAS_STREAM_ID			
		EGROUP6	55:48	W	EPATH (Wide mode or lpGBT) is associated with a STREAM ID
		EGROUP5	47:40	W	EPATH (Wide mode or lpGBT) is associated with a STREAM ID
		EGROUP4	39:32	W	EPATH is associated with a STREAM ID
		EGROUP3	31:24	W	EPATH is associated with a STREAM ID
		EGROUP2	23:16	W	EPATH is associated with a STREAM ID
		EGROUP1	15:8	W	EPATH is associated with a STREAM ID
		EGROUP0	7:0	W	EPATH is associated with a STREAM ID, use only bit0 for FULL mode.
DECODING_LINK_STATUS_ARR					
0x2180	0,1	DECODING_LINK_ALIGNED_00	57:0	R	Every bit corresponds to an E-link on one (lp)GBT or FULL-mode frame. For FULL mode only bit 0 is used
. . .					
0x22F0	0,1	DECODING_LINK_ALIGNED_23	57:0	R	Every bit corresponds to an E-link on one (lp)GBT or FULL-mode frame. For FULL mode only bit 0 is used
DECODING_EGROUP_CTRL_GEN					
DECODING_EGROUP					
0x2300	0,1	DECODING_LINK00_EGROUP0_CTRL			
		EPATH_ALMOST_FULL	58:51	R	FIFO full indication
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath
		PATH_ENCODING	42:11	W	Encoding for every EPATH, 4 bits per E-path 0: direct mode 1: 8b10b mode 2: HDLC mode 3: TTC 4: ITk Strips 8b10b 5: ITk Pixel 6: Endeavour 7-15: reserved
		EPATH_WIDTH	10:8	W	Width in bits of all EPATHS in an EGROUP 0:2, 1:4, 2:8, 3:16, 4:32
		EPATH_ENA	7:0	W	Enable bits per EPROC
. . .					
0x2360	0,1	DECODING_LINK00_EGROUP6_CTRL			
		EPATH_ALMOST_FULL	58:51	R	FIFO full indication
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath
		PATH_ENCODING	42:11	W	Encoding for every EPATH, 4 bits per E-path 0: direct mode 1: 8b10b mode 2: HDLC mode 3: TTC 4: ITk Strips 8b10b 5: ITk Pixel 6: Endeavour 7-15: reserved
		EPATH_WIDTH	10:8	W	Width in bits of all EPATHS in an EGROUP 0:2, 1:4, 2:8, 3:16, 4:32
		EPATH_ENA	7:0	W	Enable bits per EPROC
. . .					
DECODING_EGROUP					
0x27D0	0,1	DECODING_LINK11_EGROUP0_CTRL			
		EPATH_ALMOST_FULL	58:51	R	FIFO full indication
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath

		PATH_ENCODING	42:11	W	Encoding for every EPATH, 4 bits per E-path 0: direct mode 1: 8b10b mode 2: HDLC mode 3: TTC 4: ITk Strips 8b10b 5: ITk Pixel 6: Endeavour 7-15: reserved
		EPATH_WIDTH	10:8	W	Width in bits of all EPATHS in an EGROUP 0:2, 1:4, 2:8, 3:16, 4:32
		EPATH_ENA	7:0	W	Enable bits per EPROC
. . .					
0x2830	0,1	DECODING_LINK11_EGROUP6_CTRL			
		EPATH_ALMOST_FULL	58:51	R	FIFO full indication
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath
		PATH_ENCODING	42:11	W	Encoding for every EPATH, 4 bits per E-path 0: direct mode 1: 8b10b mode 2: HDLC mode 3: TTC 4: ITk Strips 8b10b 5: ITk Pixel 6: Endeavour 7-15: reserved
		EPATH_WIDTH	10:8	W	Width in bits of all EPATHS in an EGROUP 0:2, 1:4, 2:8, 3:16, 4:32
		EPATH_ENA	7:0	W	Enable bits per EPROC
MINI_EGROUP_TOHOST_GEN					
0x2840	0,1	MINI_EGROUP_TOHOST_00			
		AUX_ALMOST_FULL	12	R	Indicator that the AUX path FIFO is almost full
		AUX_BIT_SWAPPING	11	W	0: two input bits of IC e-link are as documented, 1: two input bits are swapped
		AUX_ENABLE	10	W	Enables the AUX channel
		IC_ALMOST_FULL	9	R	Indicator that the IC path FIFO is almost full
		IC_BIT_SWAPPING	8	W	0: two input bits of IC e-link are as documented, 1: two input bits are swapped
		IC_ENABLE	7	W	Enables the IC channel
		EC_ALMOST_FULL	6	R	Indicator that the EC path FIFO is almost full
		EC_BIT_SWAPPING	5	W	0: two input bits of EC e-link are as documented, 1: two input bits are swapped
		EC_ENCODING	4:1	W	Configures encoding of the EC channel
		EC_ENABLE	0	W	Enables the EC channel
. . .					
0x29B0	0,1	MINI_EGROUP_TOHOST_23			
		AUX_ALMOST_FULL	12	R	Indicator that the AUX path FIFO is almost full
		AUX_BIT_SWAPPING	11	W	0: two input bits of IC e-link are as documented, 1: two input bits are swapped
		AUX_ENABLE	10	W	Enables the AUX channel
		IC_ALMOST_FULL	9	R	Indicator that the IC path FIFO is almost full
		IC_BIT_SWAPPING	8	W	0: two input bits of IC e-link are as documented, 1: two input bits are swapped
		IC_ENABLE	7	W	Enables the IC channel
		EC_ALMOST_FULL	6	R	Indicator that the EC path FIFO is almost full
		EC_BIT_SWAPPING	5	W	0: two input bits of EC e-link are as documented, 1: two input bits are swapped
		EC_ENCODING	4:1	W	Configures encoding of the EC channel
		EC_ENABLE	0	W	Enables the EC channel
0x29C0	0,1	TTC_TOHOST_ENABLE	0	W	Enables the ToHost Mini Egroup in TTC mode

0x29D0	0,1	DECODING_REVERSE_10B	0	W	Reverse 10-bit word of elink data for 8b10b E-links 1: Receive 10-bit word in ToHost E-Paths, MSB first 0: Receive 10-bit word in ToHost E-Paths, LSB first
RD53 B_PROCESSOR_GEN					
0x29E0	0,1	RD53B_PROCESSOR_00			
		ENABLE_MULTICHIP	3	R	Decoding block
		ENABLE_BINARYTREE	2	R	Decoding block
		ENABLE_TOT	1	R	Decoding block
		DROP_TOT	0	R	Decoding block
. . .					
0x2DD0	0,1	RD53B_PROCESSOR_63			
		ENABLE_MULTICHIP	3	R	Decoding block
		ENABLE_BINARYTREE	2	R	Decoding block
		ENABLE_TOT	1	R	Decoding block
		DROP_TOT	0	R	Decoding block
Encoding Controls And Monitors					
0x3000	0,1	ENCODING_REVERSE_10B	0	W	Reverse 10-bit word of elink data for 8b10b E-links. 1 MSB first, 0 LSB first
ENCODING_EGROUP_CTRL_GEN					
ENCODING_EGROUP					
0x3010	0,1	ENCODING_LINK00_EGROUP0_CTRL			
		TTC_OPTION	62:59	W	Selects TTC bits sent to the E-link
		EPATH_ALMOST_FULL	58:51	R	Indiator that the EPATH FIFO is almost full
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath
		EPATH_WIDTH	42:40	W	Width of the Elinks in the egroup 0: 2 bit 80 Mb/s 1: 4 bit 160 Mb/s 2: 8 bit 320 Mb/s
		PATH_ENCODING	39:8	W	Encoding for every EPATH, 4 bits per E-Path 0: No encoding 1: 8b10b mode 2: HDLC mode 3: ITk Strip LCB 4: ITk Pixel 5: Endeavour 6: reserved 7: reserved greater than 7: TTC mode, see firmware Phase 2 specification doc
		EPATH_ENA	7:0	W	Enable bits per E-PATH
. . .					
0x3050	0,1	ENCODING_LINK00_EGROUP4_CTRL			
		TTC_OPTION	62:59	W	Selects TTC bits sent to the E-link
		EPATH_ALMOST_FULL	58:51	R	Indiator that the EPATH FIFO is almost full
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath
		EPATH_WIDTH	42:40	W	Width of the Elinks in the egroup 0: 2 bit 80 Mb/s 1: 4 bit 160 Mb/s 2: 8 bit 320 Mb/s
		PATH_ENCODING	39:8	W	Encoding for every EPATH, 4 bits per E-Path 0: No encoding 1: 8b10b mode 2: HDLC mode 3: ITk Strip LCB 4: ITk Pixel 5: Endeavour 6: reserved 7: reserved greater than 7: TTC mode, see firmware Phase 2 specification doc

		EPATH_ENA	7:0	W	Enable bits per E-PATH
...					
ENCODING_EGROUP					
0x3380	0,1	ENCODING_LINK11_EGROUP0_CTRL			
		TTC_OPTION	62:59	W	Selects TTC bits sent to the E-link
		EPATH_ALMOST_FULL	58:51	R	Indiator that the EPATH FIFO is almost full
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath
		EPATH_WIDTH	42:40	W	Width of the Elinks in the egroup 0: 2 bit 80 Mb/s 1: 4 bit 160 Mb/s 2: 8 bit 320 Mb/s
		PATH_ENCODING	39:8	W	Encoding for every EPATH, 4 bits per E-Path 0: No encoding 1: 8b10b mode 2: HDLC mode 3: ITk Strip LCB 4: ITk Pixel 5: Endeavour 6: reserved 7: reserved greater than 7: TTC mode, see firmware Phase 2 specification doc
		EPATH_ENA	7:0	W	Enable bits per E-PATH
...					
0x33C0	0,1	ENCODING_LINK11_EGROUP4_CTRL			
		TTC_OPTION	62:59	W	Selects TTC bits sent to the E-link
		EPATH_ALMOST_FULL	58:51	R	Indiator that the EPATH FIFO is almost full
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath
		EPATH_WIDTH	42:40	W	Width of the Elinks in the egroup 0: 2 bit 80 Mb/s 1: 4 bit 160 Mb/s 2: 8 bit 320 Mb/s
		PATH_ENCODING	39:8	W	Encoding for every EPATH, 4 bits per E-Path 0: No encoding 1: 8b10b mode 2: HDLC mode 3: ITk Strip LCB 4: ITk Pixel 5: Endeavour 6: reserved 7: reserved greater than 7: TTC mode, see firmware Phase 2 specification doc
		EPATH_ENA	7:0	W	Enable bits per E-PATH
MINI_EGROUP_FROMHOST_GEN					
0x33D0	0,1	MINI_EGROUP_FROMHOST_00			
		AUX_ALMOST_FULL	12	R	Indicator that the AUX Path FIFO is almost full
		AUX_BIT_SWAPPING	11	W	0: two input bits of AUX e-link are as documented, 1: two input bits are swapped
		AUX_ENABLE	10	W	Enables the AUX channel
		IC_ALMOST_FULL	9	R	Indicator that the IC Path FIFO is almost full
		IC_BIT_SWAPPING	8	W	0: two input bits of IC e-link are as documented, 1: two input bits are swapped
		IC_ENABLE	7	W	Enables the IC channel
		EC_ALMOST_FULL	6	R	Indicator that the EC Path FIFO is almost full
		EC_BIT_SWAPPING	5	W	0: two output bits of EC e-link are as documented, 1: two output bits are swapped
		EC_ENCODING	4:1	W	Configures encoding of the EC channel
		EC_ENABLE	0	W	Configures the FromHost Mini egroup
...					
0x3540	0,1	MINI_EGROUP_FROMHOST_23			

		AUX_ALMOST_FULL	12	R	Indicator that the AUX Path FIFO is almost full
		AUX_BIT_SWAPPING	11	W	0: two input bits of AUX e-link are as documented, 1: two input bits are swapped
		AUX_ENABLE	10	W	Enables the AUX channel
		IC_ALMOST_FULL	9	R	Indicator that the IC Path FIFO is almost full
		IC_BIT_SWAPPING	8	W	0: two input bits of IC e-link are as documented, 1: two input bits are swapped
		IC_ENABLE	7	W	Enables the IC channel
		EC_ALMOST_FULL	6	R	Indicator that the EC Path FIFO is almost full
		EC_BIT_SWAPPING	5	W	0: two output bits of EC e-link are as documented, 1: two output bits are swapped
		EC_ENCODING	4:1	W	Configures encoding of the EC channel
		EC_ENABLE	0	W	Configures the FromHost Mini egroup
ENCODING_EGROUP_CTRL_FEI4_GEN					
ENCODING_EGROUP_FEI4					
0x3550	0,1	ENCODING_LINK00_EGROUP0_FEI4_CTRL			
		PHASE_DELAY1	11:9	W	phase delay of output data, with 320 Bb/s e-link 8 phases per BC
		MANCHESTER_ENABLE1	8	W	enable manchester encoding
		AUTOMATIC_MERGE_DISABLE1	7	W	Disable automatic merging
		TTC_SELECT1	6	W	TTC/FromHost select (if automatic merging is disabled)
		PHASE_DELAY0	5:3	W	phase delay of output data, with 320 Bb/s e-link 8 phases per BC
		MANCHESTER_ENABLE0	2	W	enable manchester encoding
		AUTOMATIC_MERGE_DISABLE0	1	W	Disable automatic merging
		TTC_SELECT0	0	W	TTC/FromHost select (if automatic merging is disabled)
...					
0x3590	0,1	ENCODING_LINK00_EGROUP4_FEI4_CTRL			
		PHASE_DELAY1	11:9	W	phase delay of output data, with 320 Bb/s e-link 8 phases per BC
		MANCHESTER_ENABLE1	8	W	enable manchester encoding
		AUTOMATIC_MERGE_DISABLE1	7	W	Disable automatic merging
		TTC_SELECT1	6	W	TTC/FromHost select (if automatic merging is disabled)
		PHASE_DELAY0	5:3	W	phase delay of output data, with 320 Bb/s e-link 8 phases per BC
		MANCHESTER_ENABLE0	2	W	enable manchester encoding
		AUTOMATIC_MERGE_DISABLE0	1	W	Disable automatic merging
		TTC_SELECT0	0	W	TTC/FromHost select (if automatic merging is disabled)
...					
ENCODING_EGROUP_FEI4					
0x38C0	0,1	ENCODING_LINK11_EGROUP0_FEI4_CTRL			
		PHASE_DELAY1	11:9	W	phase delay of output data, with 320 Bb/s e-link 8 phases per BC
		MANCHESTER_ENABLE1	8	W	enable manchester encoding
		AUTOMATIC_MERGE_DISABLE1	7	W	Disable automatic merging
		TTC_SELECT1	6	W	TTC/FromHost select (if automatic merging is disabled)
		PHASE_DELAY0	5:3	W	phase delay of output data, with 320 Bb/s e-link 8 phases per BC
		MANCHESTER_ENABLE0	2	W	enable manchester encoding
		AUTOMATIC_MERGE_DISABLE0	1	W	Disable automatic merging
		TTC_SELECT0	0	W	TTC/FromHost select (if automatic merging is disabled)
...					
0x3900	0,1	ENCODING_LINK11_EGROUP4_FEI4_CTRL			
		PHASE_DELAY1	11:9	W	phase delay of output data, with 320 Bb/s e-link 8 phases per BC
		MANCHESTER_ENABLE1	8	W	enable manchester encoding
		AUTOMATIC_MERGE_DISABLE1	7	W	Disable automatic merging
		TTC_SELECT1	6	W	TTC/FromHost select (if automatic merging is disabled)
		PHASE_DELAY0	5:3	W	phase delay of output data, with 320 Bb/s e-link 8 phases per BC
		MANCHESTER_ENABLE0	2	W	enable manchester encoding
		AUTOMATIC_MERGE_DISABLE0	1	W	Disable automatic merging
		TTC_SELECT0	0	W	TTC/FromHost select (if automatic merging is disabled)

Frontend Emulator Controls And Monitors					
0x4000	0, 1	FE_EMU_ENA			
		EMU_TOFRONTEND	1	W	Enable GBT dummy emulator ToFrontEnd
		EMU_TOHOST	0	W	Enable GBT dummy emulator ToHost
0x4010	0, 1	FE_EMU_CONFIG			
		WE	54:47	W	write enable array, every bit is one emulator RAM block
		WRADDR	46:33	W	write address bus
		WRDATA	32:0	W	write data bus
0x4020	0, 1	FE_EMU_READ			
		SEL	35:33	W	Select ramblock to read back
		DATA	32:0	R	Read back ramblock at FE_EMU_CONFIG.WRADDR
Link Wrapper Controls					
0x5400	0	GBT_CHANNEL_DISABLE	47:0	W	Disable selected lpGBT, GBT or FULL mode channel
0x5410	0	GBT_GENERAL_CTRL	63:0	W	Alignment chk reset (not self clearing)
0x5420	0	GBT_MODE_CTRL			
		RX_ALIGN_TB_SW	2	W	RX_ALIGN_TB_SW
		RX_ALIGN_SW	1	W	RX_ALIGN_SW
		DESMUX_USE_SW	0	W	DESMUX_USE_SW
0x5480	0	GBT_RXSLIDE_SELECT	47:0	W	RxSlide select [47:0]
0x5490	0	GBT_RXSLIDE_MANUAL	47:0	W	RxSlide select [47:0]
0x54A0	0	GBT_TXUSRRDY	47:0	W	TxUsrRdy [47:0]
0x54B0	0	GBT_RXUSRRDY	47:0	W	RxUsrRdy [47:0]
0x54C0	0	GBT_SOFT_RESET	47:0	W	SOFT_RESET [47:0]
0x54D0	0	GBT_GTTX_RESET	47:0	W	GTTX_RESET [47:0]
0x54E0	0	GBT_GTRX_RESET	47:0	W	GTRX_RESET [47:0]
0x54F0	0	GBT_PLL_RESET			
		QPLL_RESET	59:48	W	QPLL_RESET [11:0]
		CPLL_RESET	47:0	W	CPLL_RESET [47:0]
0x5500	0	GBT_SOFT_TX_RESET			
		RESET_ALL	59:48	W	SOFT_TX_RESET_ALL [11:0]
		RESET_GT	47:0	W	SOFT_TX_RESET_GT [47:0]
0x5510	0	GBT_SOFT_RX_RESET			
		RESET_ALL	59:48	W	SOFT_TX_RESET_ALL [11:0]
		RESET_GT	47:0	W	SOFT_TX_RESET_GT [47:0]
0x5520	0	GBT_ODD_EVEN	47:0	W	OddEven [47:0]
0x5530	0	GBT_TOPBOT	47:0	W	TopBot [47:0]
0x5540	0	GBT_TX_TC_DLY_VALUE1	47:0	W	TX_TC_DLY_VALUE [47:0]
0x5550	0	GBT_TX_TC_DLY_VALUE2	47:0	W	TX_TC_DLY_VALUE [95:48]
0x5560	0	GBT_TX_TC_DLY_VALUE3	47:0	W	TX_TC_DLY_VALUE [143:96]
0x5570	0	GBT_TX_TC_DLY_VALUE4	47:0	W	TX_TC_DLY_VALUE [191:144]
0x5580	0	GBT_DATA_TXFORMAT1	47:0	W	DATA_TXFORMAT [47:0]
0x5590	0	GBT_DATA_TXFORMAT2	47:0	W	DATA_TXFORMAT [95:48]
0x55A0	0	GBT_DATA_RXFORMAT1	47:0	W	DATA_RXFORMAT [47:0]
0x55B0	0	GBT_DATA_RXFORMAT2	47:0	W	DATA_RXFORMAT [95:0]
0x55C0	0	GBT_TX_RESET	47:0	W	TX Logic reset [47:0]
0x55D0	0	GBT_RX_RESET	47:0	W	RX Logic reset [47:0]
0x55E0	0	GBT_TX_TC_METHOD	47:0	W	TX time domain crossing method [47:0]
0x55F0	0	GBT_OUTMUX_SEL	47:0	W	Descrambler output MUX selection [47:0]
0x5600	0	GBT_TC_EDGE	47:0	W	Sampling edge selection for TX domain crossing [47:0]

0x5610	0	GBT_TXPOLARITY	47:0	W	0: default polarity 1: reversed polarity for transmitter of GTH channels
0x5620	0	GBT_RXPOLARITY	47:0	W	0: default polarity 1: reversed polarity for the receiver of the GTH channels
0x5630	0	GTH_LOOPBACK_CONTROL	2:0	W	Controls loopback for loopback: read UG476 for the details. NOTE: the TXBUFFER is disabled, near end PCS loopback is not supported. 000: Normal operation 001: Near-End PCS Loopback 010: Near-End PMA Loopback 011: Reserved 100: Far-End PMA Loopback 101: Reserved 110: Far-End PCS Loopback
0x5700	0	GBT_TOHOST_FANOUT			
		LOCK	48	W	Locks this particular register. If set prevents software from touching it.
		SEL	47:0	W	ToHost FanOut/Selector. Every bitfield is a channel: 1 : GBT_EMU, select GBT Emulator for a specific CentralRouter channel 0 : GBT_WRAP, select real GBT link for a specific CentralRouter channel
0x5710	0	GBT_TOFRONTEND_FANOUT			
		LOCK	48	W	Locks this particular register. If set prevents software from touching it.
		SEL	47:0	W	ToFrontEnd FanOut/Selector. Every bitfield is a channel: 1 : GBT_EMU, select GBT Emulator for a specific GBT link 0 : TTC_DEC, select CentralRouter data (including TTC) for a specific GBT link
Link Wrapper Monitors					
0x6600	0	GBT_VERSION			
		DATE	63:48	R	Date
		GBT_VERSION	47:32	R	GBT Version
		GTH_IP_VERSION	31:16	R	GTH IP Version
		RESERVED	15:3	R	Reserved
		GTHREFCLK_SEL	2	R	GTHREFCLK SEL
		RX_CLK_SEL	1	R	RX CLK SEL
		PLL_SEL	0	R	PLL SEL
0x6680	0	GBT_TXRESET_DONE	47:0	R	TX Reset done [47:0]
0x6690	0	GBT_RXRESET_DONE	47:0	R	RX Reset done [47:0]
0x66A0	0	GBT_TXFSMRESET_DONE	47:0	R	TX FSM Reset done [47:0]
0x66B0	0	GBT_RXFSMRESET_DONE	47:0	R	RX FSM Reset done [47:0]
0x66C0	0	GBT_CPLL_FBCLK_LOST	47:0	R	CPLL FBCLK LOST [47:0]
0x66D0	0	GBT_PLL_LOCK			
		QPLL_LOCK	59:48	R	QPLL LOCK [11:0]
		CPLL_LOCK	47:0	R	CPLL LOCK [47:0]
0x66E0	0	GBT_RXCDR_LOCK	47:0	R	RX CDR LOCK [47:0]
0x66F0	0	GBT_CLK_SAMPLED	47:0	R	clk sampled [47:0]
0x6700	0	GBT_RX_IS_HEADER	47:0	R	RX IS HEADER [47:0]
0x6710	0	GBT_RX_IS_DATA	47:0	R	RX IS DATA [47:0]
0x6720	0	GBT_RX_HEADER_FOUND	47:0	R	RX HEADER FOUND [47:0]
0x6730	0	GBT_ALIGNMENT_DONE	47:0	R	RX ALIGNMENT DONE [47:0]
0x6740	0	GBT_OUT_MUX_STATUS	47:0	R	GBT output mux status [47:0]
0x6750	0	GBT_ERROR	47:0	R	Error flags [47:0]
0x6760	0	GBT_GBT_TOPBOT_C	47:0	R	TopBot_c [47:0]
0x6800	0	GBT_FM_RX_DISP_ERROR1	47:0	R	Rx disparity error [47:0]

0x6810	0	GBT_FM_RX_DISP_ERROR2	47:0	R	Rx disparity error [96:48]
0x6820	0	GBT_FM_RX_NOTINTABLE1	47:0	R	Rx not in table [47:0]
0x6830	0	GBT_FM_RX_NOTINTABLE2	47:0	R	Rx not in table [96:48]
		TTCBUSY Controls And Monitors			
		TTC_DEC_CTRLMON			
0x7000	0	TTC_DEC_CTRL			
		L1A_DELAY	30:27	W	Number of BC to delay the L1A distribution to the frontends
		BCID_ONBCR	26:15	W	BCID is set to this value when BCR arrives
		BUSY_OUTPUT_STATUS	14	R	Actual status of the BUSY LEMO output signal
		ECR_BCR_SWAP	13	W	ECR and BCR signals are swapped at the output of the TTC decoder (needed only for LAr TTC)
		BUSY_OUTPUT_INHIBIT	12	W	forces the Busy LEMO output to BUSY-OFF
		TOHOST_RST	11	W	reset toHost in ttc decoder
		TT_BCH_EN	10	W	trigger type enable / disable for TTC-ToHost
		XL1ID_SW	9:2	W	set XL1ID value, the value to be set by XL1ID_RST signal
		XL1ID_RST	1	W	giving a trigger signal to reset XL1ID value
		MASTER_BUSY	0	W	L1A trigger throttling
0x7010	0	TTC_DEC_MON			
		TH_FF_COUNT	15:5	R	ToHostData Fifo counts
		TH_FF_FULL	4	R	ToHostData Fifo status 1:full 0:not full
		TH_FF_EMPTY	3	R	ToHostData Fifo status 1:empty 0:not empty
		TTC_BIT_ERR	2:0	R	double bit, single bit and comm error in TTC data
		TTC_BUSY_ACCEPTED_G			
0x7020	0,1	TTC_BUSY_ACCEPTED00	56:0	R	busy has been asserted by the given ELINK. Reset by writing to TTC_BUSY_CLEAR
		...			
0x7190	0,1	TTC_BUSY_ACCEPTED23	56:0	R	busy has been asserted by the given ELINK. Reset by writing to TTC_BUSY_CLEAR
0x71A0	0	TTC_EMU			
		FULL	2	R	TTC Emulator memory full indication
		SEL	1	W	Select TTC data source 1 TTC Emu | 0 TTC Decoder
		ENA	0	W	Clear to load into the TTC emulatorâĂŹs memory the required sequence, Set to run the TTC emulator sequence
		TTC_DELAY			
0x71B0	0	TTC_DELAY_00	3:0	W	Controls the TTC Fanout delay values
		...			
0x74A0	0	TTC_DELAY_47	3:0	W	Controls the TTC Fanout delay values
0x74B0	0	TTC_BUSY_TIMING_CTRL			
		PRESCALE	51:32	W	Prescales the 40MHz clock to create an internal slow clock
		BUSY_WIDTH	31:16	W	Minimum number of 40MHz clocks that the busy is asserted
		LIMIT_TIME	15:0	W	Number of prescaled clocks a given busy must be asserted before it is recognized
0x74C0	0	TTC_BUSY_CLEAR	any	T	clears the latching busy bits in TTC_BUSY_ACCEPTED
0x74D0	0	TTC_EMU_CONTROL			
		BROADCAST	32:27	W	Broadcast data
		ECR	26	W	Event counter reset
		BCR	25	W	Bunch counter reset
		L1A	24	W	Level 1 Accept
0x74E0	0	TTC_EMU_L1A_PERIOD	31:0	W	L1A period in BC. 0 means manual L1A with TTC_EMU_CONTROL.L1A
0x74F0	0	TTC_EMU_ECR_PERIOD	31:0	W	ECR period in BC. 0 means manual ECR with TTC_EMU_CONTROL.ECR
0x7500	0	TTC_EMU_BCR_PERIOD	31:0	W	BCR period in BC. 0 means manual BCR with TTC_EMU_CONTROL.BCR
0x7510	0	TTC_EMU_LONG_CHANNEL_-DATA	31:0	W	Long channel data for the TTC emulator

0x7520	0	TTC_EMU_RESET	any	T	Any write to this register resets the TTC Emulator to the default state.
0x7530	0	TTC_L1ID_MONITOR	31:0	R	Monitor L1ID and XL1ID.
0x7540	0	TTC_ECR_MONITOR			
		CLEAR	any	T	Counts the number of ECRs received from the TTC system, any write to this register clears the counter
		VALUE	31:0	R	Counts the number of ECRs received from the TTC system, any write to this register clears the counter
0x7550	0	TTC_TTYPE_MONITOR			
		CLEAR	any	T	Counts the number of TType received from the TTC system, any write to this register clears the counter
		VALUE	31:0	R	Counts the number of TType received from the TTC system, any write to this register clears the counter
0x7560	0	TTC_BCR_PERIODICITY_MONITOR			
		CLEAR	any	T	Counts the number of times the BCR period does not match 3564, any write to this register clears the counter
		VALUE	31:0	R	Counts the number of times the BCR period does not match 3564, any write to this register clears the counter
XOFF_BUSY Controls And Monitors					
0x8000	0, 1	XOFF_FM_CH_FIFO_THRESH_-LOW	3:0	W	Controls the low threshold of the channel fifo in FULL mode on which an Xon will be asserted, bitfields control 4 MSB
0x8010	0, 1	XOFF_FM_CH_FIFO_THRESH_-HIGH	3:0	W	Controls the high threshold of the channel fifo in FULL mode on which an Xoff will be asserted, bitfields control 4 MSB - name: XOFF_FM_LOW_THRESH_CROSSED
0x8020	0, 1	XOFF_FM_LOW_THRESH_-CROSSED	23:0	R	FIFO filled beyond the low threshold, 1 bit per channel
0x8030	0, 1	XOFF_FM_HIGH_THRESH			
		CLEAR_LATCH	any	T	Writing this register will clear all CROSS_LATCHED bits
		CROSS_LATCHED	47:24	R	FIFO filled beyond the high threshold, 1 latch bit per channel
		CROSSED	23:0	R	FIFO filled beyond the high threshold, 1 bit per channel
0x8040	0, 1	XOFF_FM_SOFT_XOFF	23:0	W	Set any bit in this register to assert XOFF for the given channel, clearing bits will assert XON
0x8050	0, 1	XOFF_ENABLE	23:0	W	Enable XOFF assertion (To Frontend) in case the FULL mode CH FIFO gets beyond thresholds. One bit per channel
0x8060	0, 1	DMA_BUSY_STATUS			
		CLEAR_LATCH	any	T	Any write to this register clears TOHOST_BUSY_LATCHED
		ENABLE	4	W	Enable the DMA buffer on the server as a source of busy
		TOHOST_BUSY_LATCHED	3	R	A tohost descriptor has passed BUSY_THRESHOLD_ASSERT in the past, busy flag was set
		TOHOST_BUSY	0	R	A tohost descriptor passed BUSY_THRESHOLD_ASSERT, busy flag set
0x8070	0, 1	FM_BUSY_CHANNEL_STATUS			
		CLEAR_LATCH	any	T	Any write to this register will clear the BUSY_LATCHED bits
		BUSY_LATCHED	47:24	R	one Indicates that the given FULL mode channel has received BUSY-ON
		BUSY	23:0	R	one Indicates that the given FULL mode channel is currently in BUSY state
0x8080	0, 1	BUSY_MAIN_OUTPUT_FIFO_THRESH			
		BUSY_ENABLE	24	W	Enable busy generation if thresholds are crossed
		LOW	23:12	W	Low, Negate threshold of busy generation from main output fifo
		HIGH	11:0	W	High, Assert threshold of busy generation from main output fifo

0x8090	0, 1	BUSY_MAIN_OUTPUT_FIFO_STATUS			
		CLEAR_LATCHED	any	T	Any write to this register will clear the
		HIGH_THRESH_CROSSED_-LATCHED	2	R	Main output fifo has been full beyond HIGH THRESHOLD, write to clear
		HIGH_THRESH_CROSSED	1	R	Main output fifo is full beyond HIGH THRESHOLD
		LOW_THRESH_CROSSED	0	R	Main output fifo is full beyond LOW THRESHOLD
ELINK_BUSY_ENABLE					
0x80A0	0	ELINK_BUSY_ENABLE00	56:0	W	Per elink (and FULL mode link) enable of the busy signal towards the LEMO output
...					
0x8210	0	ELINK_BUSY_ENABLE23	56:0	W	Per elink (and FULL mode link) enable of the busy signal towards the LEMO output
XOFF_STATISTICS					
0x8220	0,1	XOFF_PEAK_DURATION00	63:0	R	Maximum occurred duration of XOFF on the given channel in 25ns bins since reset
0x8230	0,1	XOFF_TOTAL_DURATION00	63:0	R	Total occurred duration of XOFF on the given channel in 25ns bins, divide by number of Xoffs to calculate the average since reset
0x8240	0,1	XOFF_COUNT00	63:0	R	Total number of XOFF events per channel that occurred since a reset.
...					
0x8670	0,1	XOFF_PEAK_DURATION23	63:0	R	Maximum occurred duration of XOFF on the given channel in 25ns bins since reset
0x8680	0,1	XOFF_TOTAL_DURATION23	63:0	R	Total occurred duration of XOFF on the given channel in 25ns bins, divide by number of Xoffs to calculate the average since reset
0x8690	0,1	XOFF_COUNT23	63:0	R	Total number of XOFF events per channel that occurred since a reset.
House Keeping Controls And Monitors					
0x9000	0	HK_CTRL_I2C			
		CONFIG_TRIG	1	W	i2c_config_trig
		CLKFREQ_SEL	0	W	i2c_clkfreq_sel
0x9010	0	HK_CTRL_FMC			
		SI5345_LOL	7	R	Loss of lock pin, only connected on FLX711
		SI5345_INSEL	6:5	W	Selects the input clock source 0 : FPGA (FMC LA01) 1 : FMC OSC (40.079 MHz) 2 : FPGA (FMC LA18)
		SI5345_A	4:3	W	Si5345 I2C address select 2 LSB (0x0:default, dev id 0x68)
		SI5345_OE	2	W	Si5345 active low output enable (0:enable)
		SI5345_RSTN	1	W	Si5345 active low output enable (0:reset)
		SI5345_SEL	0	W	Si5345 programming mode 1 : I2C mode (default) 0 : SPI mode
0x9020	0	HK_MON_FMC			
		SI5345_LOL	1	W	Si5345 Loss Of Lock pin
		SI5345_INTR	0	W	Si5345 Interrupt flagging chip change of status
0x9300	0	MMCM_MAIN			
		LCLK_SEL	3	W	1: LCLK 0: TTC
		MAIN_INPUT	2:1	R	Main MMCM Oscillator Input 2: LCLK fixed 1: TTC fixed 0: selectable
		PLL_LOCK	0	R	Main MMCM PLL Lock Status
0x9310	0	LMK_LOCKED	0	R	LMK Chip on BNL-711 locked

0x9320	0	FPGA_CORE_TEMP	11:0	R	XADC temperature monitor for the FPGA CORE for FLX709, FLX710 temp (C)= ((FPGA_CORE_TEMP* 503.975)/4096)-273.15 for FLX711 temp (C)= ((FPGA_CORE_TEMP* 502.9098)/4096)-273.8195
0x9330	0	FPGA_CORE_VCCINT	11:0	R	XADC voltage measurement VCCINT = (FPGA_CORE_VCCINT *3.0)/4096
0x9340	0	FPGA_CORE_VCCAUX	11:0	R	XADC voltage measurement VCCAUX = (FPGA_CORE_VCCAUX *3.0)/4096
0x9350	0	FPGA_CORE_VCCBRAM	11:0	R	XADC voltage measurement VCCBRAM = (FPGA_CORE_VCCBRAM *3.0)/4096
0x9360	0	FPGA_DNA	63:0	R	Unique identifier of the FPGA
0x9420	0	I2C_WR			
		I2C_WREN	any	T	Any write to this register triggers an I2C read or write sequence
		I2C_FULL	25	R	I2C FIFO full
		WRITE_2BYTES	24	W	Write two bytes
		DATA_BYTE2	23:16	W	Data byte 2
		DATA_BYTE1	15:8	W	Data byte 1
		SLAVE_ADDRESS	7:1	W	Slave address
		READ_NOT_WRITE	0	W	READ/<o>WRITE</o>
0x9430	0	I2C_RD			
		I2C_RDEN	any	T	Any write to this register pops the last I2C data from the FIFO
		I2C_EMPTY	8	R	I2C FIFO Empty
		I2C_DOUT	7:0	R	I2C READ Data
0x9800	0	INT_TEST			
		TRIGGER	any	T	Fire a test MSIx interrupt set in IRQ
		IRQ	3:0	W	Set this field to a value equal to the MSIX interrupt to be fired. The write triggers the interrupt immediately.
0x9810	0	CONFIG_FLASH_WR			
		FAST_WRITE	57	W	Write command only. Only used for fast programming.
		FAST_READ	56	W	Status reading without command writing. Only used for fast programming.
		PAR_CTRL	55	W	Choose use FW or uC to select the Flash partition. 1 FW | 0 uC.
		PAR_WR	54:53	W	Choose Flash partition. Valid when PAR_CTRL is 1.
		FLASH_SEL	52	W	1 takes control over flash, 0 gives JTAG control over flash
		DO_INIT	51	W	Untested feature, don't use it yet.
		DO_READSTATUS	50	W	Reads status from flash
		DO_CLEARSTATUS	49	W	Clears status reading from flash, back to normal flash operation
		DO_ERASEBLOCK	48	W	Erased the current block of the flash, this register has to be cleared by software
		DO_UNLOCK_BLOCK	47	W	Unlock writes to the current block, this register has to be cleared by software
		DO_READ	46	W	Reads the 16 bits from current address, this register has to be cleared by software
		DO_WRITE	45	W	Writes the 16 bits to current address, this register has to be cleared by software
		DO_READDEVICEID	44	W	DIN should return 0x0089, this register has to be cleared by software
		DO_RESET	43	W	Can be used in the future, currently disconnected in firmware
		ADDRESS	42:16	W	Address for read and write operations (25 bits, upper 2 bits are controlled by uC)
		WRITE_DATA	15:0	W	Value of data to write towards flash
0x9820	0	CONFIG_FLASH_RD			
		PAR_RD	19:18	R	Show which Flash partition is selected.
		FLASH_REQ_DONE	17	R	Request done
		FLASH_BUSY	16	R	Flash operation busy
		READ_DATA	15:0	R	Value of data read from flash
0x9830	0	SI5324_STATUS			

		LOL	15:8	R	Loss of Lock Si5324
		LOS	8:0	R	Loss of Signal Si5324
0x9840	0	TACH_CNT	19:0	R	Readout of the Fan tachometer speed of the BNL712 board
0x9850	0	RXUSRCLK_FREQ			
		VALID	38	R	Indicates that the frequency measurement is valid
		CHANNEL	37:32	W	Select the Transceiver channel to measure the clock from.
		VAL	31:0	R	Frequency in Hz of the selected channel
Generators					
0xA000	0	FELIG_L1ID_RESET	any	T	Any write to this register clears the FELIG L1ID
FELIG_DATA_GEN_CONFIG_ARR					
0xA020	0	FELIG_DATA_GEN_CONFIG_00			
		USERDATA	63:48	W	Sets static payload word. When PATTERN_SEL=1.
		CHUNK_LENGTH	47:32	W	FELIG data generator chunk-length in bytes.
		RESET	19:15	W	FELIG data generator reset. One bit per group, 0:normal operation, 1:egroup emulation held in reset.
		SW_BUSY	14:10	W	FELIG elink bus state. One bit per group, 0:normal operation, 1:elink enter busy state.
		DATA_FORMAT	9:5	W	FELIG data generator format. 0:8b10b, 1:direct.
		PATTERN_SEL	4:0	W	FELIG data payload type. One bit per group, 0:byte counter, 1:USERDATA
...					
0xA190	0	FELIG_DATA_GEN_CONFIG_23			
		USERDATA	63:48	W	Sets static payload word. When PATTERN_SEL=1.
		CHUNK_LENGTH	47:32	W	FELIG data generator chunk-length in bytes.
		RESET	19:15	W	FELIG data generator reset. One bit per group, 0:normal operation, 1:egroup emulation held in reset.
		SW_BUSY	14:10	W	FELIG elink bus state. One bit per group, 0:normal operation, 1:elink enter busy state.
		DATA_FORMAT	9:5	W	FELIG data generator format. 0:8b10b, 1:direct.
		PATTERN_SEL	4:0	W	FELIG data payload type. One bit per group, 0:byte counter, 1:USERDATA
FELIG_ELINK_CONFIG_ARR					
0xA1A0	0	FELIG_ELINK_CONFIG_00			
		ENDIAN_MOD	39:35	W	FELIG elink data input endian control. One bit per egroup. 0:little-endian (8b10b), 1:big-endian.
		INPUT_WIDTH	34:30	W	FELIG elink data input width. One bit per egroup. 0:8-bit (direct), 1:10-bit (8b10b).
		OUTPUT_WIDTH	9:0	W	FELIG elink data output width.
...					
0xA310	0	FELIG_ELINK_CONFIG_23			
		ENDIAN_MOD	39:35	W	FELIG elink data input endian control. One bit per egroup. 0:little-endian (8b10b), 1:big-endian.
		INPUT_WIDTH	34:30	W	FELIG elink data input width. One bit per egroup. 0:8-bit (direct), 1:10-bit (8b10b).
		OUTPUT_WIDTH	9:0	W	FELIG elink data output width.
FELIG_ELINK_ENABLE_ARR					
0xA320	0	FELIG_ELINK_ENABLE_00	39:0	W	FELIG elink enable. One bit per elink. 0:disabled, 1:enabled.
...					
0xA490	0	FELIG_ELINK_ENABLE_23	39:0	W	FELIG elink enable. One bit per elink. 0:disabled, 1:enabled.
0xA4A0	0	FELIG_GLOBAL_CONTROL			
		FAKE_L1A_RATE	63:36	W	Sets the internal fake L1 trigger rate. [25ns/LSB]
		PICXO_OFFSET_PPM	35:14	W	When OFFSET_EN is 1, this directly sets the output frequency, within the given adjustment range.
		TRACK_DATA	12:12	W	FELIG GT core control. Must be set to enable normal operation.
		RXUSERRDY	11:11	W	FELIG GT core control. Must be set to enable normal operation.
		TXUSERRDY	10:10	W	FELIG GT core control. Must be set to enable normal operation.
		AUTO_RESET	9:9	W	FELIG GT core control. If set the GT core automatically resets on data error.

		PICXO_RESET	8:8	W	FELIG GT core control. Manual PICXO reset.
		GTTX_RESET	7:7	W	FELIG GT core control. Manual GT TX reset
		CPLL_RESET	6:6	W	FELIG GT core control. Manual CPLL reset.
		X3_X4_OUTPUT_SELECT	5:0	W	X3/X4 SMA output source select.
FELIG_LANE_CONFIG_ARR					
0xA4B0	0	FELIG_LANE_CONFIG_00			
		B_CH_BIT_SEL	63:42	W	When OFFSET_EN is 1. this directly sets the output frequency. within the given adjustment range.
		A_CH_BIT_SEL	41:35	W	Selects the bit from the received FELIX data from which to extract the L1A.
		LB_FIFO_DELAY	34:30	W	When the GTH or GTB loopback is enabled, this controls the loopback latency in clock cycles.
		ELINK_SYNC	7:7	W	When set, synchronizes the elink word boundaries. Must be set back to 0 to resume normal operation.
		PICXO_OFFEST_EN	6:6	W	FELIG TX frequency override. 0:frequency tracking enabled, 1:TX frequency set by PICXO_OFFSET_PPM.
		PI_HOLD	5:5	W	FELIG phase-interpolator hold. 0:frequency tracking enabled, 1:freeze TX frequency.
		GBT_LB_ENABLE	4:4	W	FELIG GBT direct loopback enable. 0:disabled, 1:enabled.
		GBH_LB_ENABLE	3:3	W	FELIG GTH direct loopback enable. 0:disabled, 1:enabled.
		L1A_SOURCE	2:2	W	FELIG L1A data source select. 0:from local counter, 1:from FELIX.
		GBT_EMU_SOURCE	1:1	W	FELIG emulation data source select. 0:state-machine emulator, 1:ram-based emulator.
		FG_SOURCE	0:0	W	FELIG link check data source selection control. 0:normal operation, 1:PRBS link checker (not elink emulation data)
. . .					
0xA620	0	FELIG_LANE_CONFIG_23			
		B_CH_BIT_SEL	63:42	W	When OFFSET_EN is 1. this directly sets the output frequency. within the given adjustment range.
		A_CH_BIT_SEL	41:35	W	Selects the bit from the received FELIX data from which to extract the L1A.
		LB_FIFO_DELAY	34:30	W	When the GTH or GTB loopback is enabled, this controls the loopback latency in clock cycles.
		ELINK_SYNC	7:7	W	When set, synchronizes the elink word boundaries. Must be set back to 0 to resume normal operation.
		PICXO_OFFEST_EN	6:6	W	FELIG TX frequency override. 0:frequency tracking enabled, 1:TX frequency set by PICXO_OFFSET_PPM.
		PI_HOLD	5:5	W	FELIG phase-interpolator hold. 0:frequency tracking enabled, 1:freeze TX frequency.
		GBT_LB_ENABLE	4:4	W	FELIG GBT direct loopback enable. 0:disabled, 1:enabled.
		GBH_LB_ENABLE	3:3	W	FELIG GTH direct loopback enable. 0:disabled, 1:enabled.
		L1A_SOURCE	2:2	W	FELIG L1A data source select. 0:from local counter, 1:from FELIX.
		GBT_EMU_SOURCE	1:1	W	FELIG emulation data source select. 0:state-machine emulator, 1:ram-based emulator.
		FG_SOURCE	0:0	W	FELIG link check data source selection control. 0:normal operation, 1:PRBS link checker (not elink emulation data)
FELIG_MON_TTC_0_ARR					
0xA630	0	FELIG_MON_TTC_0_00			
		L1ID	63:40	R	Live TTC data monitor.
		XL1ID	39:32	R	Live TTC data monitor.
		BCID	31:20	R	Live TTC data monitor.
		RESERVED0	19:16	R	Live TTC data monitor.
		LEN	15:8	R	Live TTC data monitor.
		FMT	7:0	R	Live TTC data monitor.
. . .					
0xA7A0	0	FELIG_MON_TTC_0_23			
		L1ID	63:40	R	Live TTC data monitor.
		XL1ID	39:32	R	Live TTC data monitor.

		BCID	31:20	R	Live TTC data monitor.
		RESERVED0	19:16	R	Live TTC data monitor.
		LEN	15:8	R	Live TTC data monitor.
		FMT	7:0	R	Live TTC data monitor.
		FELIG_MON_TTC_1_ARR			
0xA7B0	0	FELIG_MON_TTC_1_00			
		RESERVED1	63:48	R	Live TTC data monitor.
		TRIGGER_TYPE	47:32	R	Live TTC data monitor.
		ORBIT	31:0	R	Live TTC data monitor.
		...			
0xA920	0	FELIG_MON_TTC_1_23			
		RESERVED1	63:48	R	Live TTC data monitor.
		TRIGGER_TYPE	47:32	R	Live TTC data monitor.
		ORBIT	31:0	R	Live TTC data monitor.
		FELIG_MON_COUNTERS_ARR			
0xA930	0	FELIG_MON_COUNTERS_00			
		SLIDE_COUNT	63:32	R	Counts the number of rx slides commanded by the GBT logic. Should be static once a link is established.
		FC_ERROR_COUNT	31:0	R	When FG_DATA_SELECT is 1, this counter reports the number of detected data errors.
		...			
0xAAA0	0	FELIG_MON_COUNTERS_23			
		SLIDE_COUNT	63:32	R	Counts the number of rx slides commanded by the GBT logic. Should be static once a link is established.
		FC_ERROR_COUNT	31:0	R	When FG_DATA_SELECT is 1, this counter reports the number of detected data errors.
		FELIG_MON_FREQ_ARR			
0xAAB0	0	FELIG_MON_FREQ_00			
		TX	63:32	R	FELIG regenerated TX clock frequency[Hz].
		RX	31:0	R	FELIG recovered RX clock frequency[Hz].
		...			
0xAC20	0	FELIG_MON_FREQ_23			
		TX	63:32	R	FELIG regenerated TX clock frequency[Hz].
		RX	31:0	R	FELIG recovered RX clock frequency[Hz].
0xAC30	0	FELIG_MON_FREQ_GLOBAL			
		XTAL_100MHZ	63:32	W	FELIG local oscillator frequency[Hz].
		CLK_41_667MHZ	31:0	W	FELIG PCIE MGTREFCLK frequency[Hz].
		FELIG_MON_L1 A_ID_ARR			
0xAC40	0	FELIG_MON_L1A_ID_00	31:0	R	FELIG's last L1 ID.
		...			
0xADB0	0	FELIG_MON_L1A_ID_23	31:0	R	FELIG's last L1 ID.
		FELIG_MON_PICXO_ARR			
0xADC0	0	FELIG_MON_PICXO_00			
		VLOT	53:32	R	Value indicates TX clock (recovered RX clock) to RX reference clock frequency offset.
		ERROR	20:0	R	Value indicates RX to TX frequency tracking error.
		...			
0xAF30	0	FELIG_MON_PICXO_23			
		VLOT	53:32	R	Value indicates TX clock (recovered RX clock) to RX reference clock frequency offset.
		ERROR	20:0	R	Value indicates RX to TX frequency tracking error.
0xAF40	0	FELIG_RESET			
		LB_FIFO	63:48	W	One bit per lane. When set to 1, resets all loopback FIFOs.
		FRAMEGEN	47:24	W	One bit per lane. When set to 1, resets all FELIG link checking logic.
		LANE	23:0	W	One bit per lane. When set to 1, resets all FELIG lane logic.

0xAF50	0	FELIG_RX_SLIDE_RESET	23:0	W	One bit per lane. When set to 1, resets the gbt rx slide counter.
FELIG_ITK_STRIPS_DATA_GEN_CONFIG_ARR					
0xAF60	0	FELIG_ITK_STRIPS_DATA_GEN_CONFIG_00			
		ITKS_FIFO_CTL	19:17	W	data fifo control 2:rst 1:rd 0:wr.
		ITKS_FIFO_DATA	16:0	W	itks emu data 16:last word 15-0:data word
...					
0xB0D0	0	FELIG_ITK_STRIPS_DATA_GEN_CONFIG_23			
		ITKS_FIFO_CTL	19:17	W	data fifo control 2:rst 1:rd 0:wr.
		ITKS_FIFO_DATA	16:0	W	itks emu data 16:last word 15-0:data word
FELIG_MON_ITK_STRIPS_ARR					
0xB0E0	0	FELIG_MON_ITK_STRIPS_00	2:0	R	data fifo status 2:write done 1:full 0:empty.
...					
0xB250	0	FELIG_MON_ITK_STRIPS_23	2:0	R	data fifo status 2:write done 1:full 0:empty.
0xB800	0	FMEMU_EVENT_INFO			
		L1ID	63:32	W	32b field to show L1ID
		BCID	31:0	W	32b field to show BCID
0xB810	0	FMEMU_COUNTERS			
		WORD_CNT	63:48	W	Number of 32b words in one chunk
		IDLE_CNT	47:32	W	Minimum number of idles between chunks
		L1A_CNT	31:16	W	Number of chunks to send if not in TTC mode
		BUSY_TH_HIGH	15:8	W	Assert BUSY-ON above this threshold
		BUSY_TH_LOW	7:0	W	De-assert BUSY-ON below this threshold
0xB820	0	FMEMU_CONTROL			
		L1A_BITNR	63:56	W	Bitfield for L1A in TTC frame
		XONXOFF_BITNR	55:48	W	Bitfield for Xon/Xoff in TTC frame
		EMU_START	47:47	W	Start emulator functionality
		TTC_MODE	46:46	W	Control the emulator by TTC input or by RegMap (1/0)
		XONXOFF	45:45	W	Debug Xon/Xoff functionality (1/0)
		INLC_CRC32	44:44	W	0: No checksum 1: Append the data with a CRC32
		BCR	43:43	W	Reset BCID to 0
		ECR	42:42	W	Reset L1ID to 0
		DATA_SRC_SEL	41:41	W	Data source select 0: Data input comes from EMURAM 1: Data input comes from PCIe
		INT_STATUS_EMU	40:32	R	Read internal status emulator
		FFU_FM_EMU_T	31:16	W	For Future Use (trigger registers)
		FFU_FM_EMU_W	15:0	W	For Future Use (write registers)
0xB830	0	FMEMU_RANDOM_RAM_ADDR	9:0	W	Controls the address of the ramblock for the random number generator
0xB840	0	FMEMU_RANDOM_RAM			
		WE	any	T	Any write to this register (DATA) triggers a write to the ramblock
		CHANNEL_SELECT	39:16	W	Enable write enable only for the selected channel
		DATA	15:0	W	DATA field to be written to FMEMU_RANDOM_RAM_ADDR
0xB850	0	FMEMU_RANDOM_CONTROL			
		SELECT_RANDOM	20	W	1 enables the random chunk length, 0 uses a constant chunk length
		SEED	19:10	W	Seed for the random number generator, should not be 0
		POLYNOMIAL	9:0	W	POLYNOMIAL for the random number generator (10b LFSR) Bit9 should always be 1
Wishbone					
0xC000	0	WISHBONE_CONTROL			
		WRITE_NOT_READ	32	W	wishbone write command wishbone read command
		ADDRESS	31:0	W	Slave address for Wishbone bus

0xC010	0	WISHBONE_WRITE			
		WRITE_ENABLE	any	T	Any write to this register triggers a write to the Wupper to Wishbone fifo
		FULL	32	R	Wishbone
		DATA	31:0	W	Wishbone
0xC020	0	WISHBONE_READ			
		READ_ENABLE	any	T	Any write to this register triggers a read from the Wishbone to Wupper fifo
		EMPTY	32	R	Indicates that the Wishbone to Wupper fifo is empty
		DATA	31:0	R	Wishbone read data
0xC030	0	WISHBONE_STATUS			
		INT	4	R	interrupt
		RETRY	3	R	Interface is not ready to accept data cycle should be retried
		STALL	2	R	When pipelined mode slave can't accept additional transactions in its queue
		ACKNOWLEDGE	1	R	Indicates the termination of a normal bus cycle
		ERROR	0	R	Address not mapped by the crossbar
ITK_STRIPS_CTRL					
0xD000	0,1	GLOBAL_STRIPS_CONFIG			
		TEST_MODULE_MASK	15:11	W	(for tests only) contains R3 mask for the simulated trigger data
		TEST_R3L1_TAG	10:4	W	(for tests only) contains R3 or L1 tag for the simulated trigger data
		TTC_GENERATE_GATING_-ENABLE	1	W	Global control for gating signal generation. Enables generating trickle gating signal in response to TTC BCR. TRICKLE_TRIG_RUN must also be enabled for the trickle configuration to work. (See also BC_START, and BC_STOP fields)
0xD010	0,1	GLOBAL_TRICKLE_TRIGGER	any	T	writing to this register issues a single trickle trigger for every LCB link connected to this FELIX device
ITK_STRIPS_GBT					
ITK_STRIPS_LCB_LINKS					
0xD020	0,1	CR_ITK_STRIPS_LCB_LINKS_00_LCB_0			
		L0A_BCR_DELAY	49:38	W	TTC BCR signal will be delayed by this many BCs
		L0A_FRAME_DELAY	37:34	W	By how many BCs to delay an L0A frame. Updating this register may result in brief loss of LCB lock, and some TTC L0A frames may be lost. Don't adjust this parameter while taking data.
		FRAME_PHASE	33:32	W	phase of LCB frame with respect to TTC BCR signal
		TRICKLE_BC_START	31:20	W	Determines the start of the allowed BC interval for low-priority LCB frames
		TRICKLE_BC_STOP	19:8	W	Determines the end of the allowed BC interval for low-priority LCB frames
		LCB_DESTINATION_MUX	5:4	W	Determines where the elink data is sent to: 00: command decoder (use same command encoding format as trickle configuration) 01: trickle memory (see phase2 documentation for command encoding format) 10: directly to LCB link (expecting software-encoded HCC*/ABC* frames) 11: (invalid, don't use)
		TRICKLE_TRIG_RUN	3	W	if enabled, trickle configuration is sent out continuously to the front-end (use together with TTC_GENERATE_GATING_EN for sending trickle configuration continuously during a specified BC range. See also BC_START, and BC_STOP fields.)
		TTC_L0A_ENABLE	2	W	enable generating L0A frames in response to TTC system signals

		TTC_GENERATE_GATING_-ENABLE	0	W	enables generating trickle gating signal in response to TTC BCR. TRICKLE_TRIG_RUN must also be enabled for the trickle configuration to work. (See also BC_START, and BC_STOP fields)
0xD030	0,1	CR_ITK_STRIPS_LCB_LINKS_-00_TRICKLE_TRIGGER_0	any	T	writing to this register issues a single trickle trigger
0xD040	0,1	CR_ITK_STRIPS_LCB_LINKS_00_TRICKLE_MEMORY_CONFIG_0			
		MOVE_WRITE_PTR	any	T	Writing to this register moves trickle configuration memory write pointer to WRITE_PTR address
		WRITE_PTR	47:32	W	Trickle configuration memory write pointer
		VALID_DATA_START	31:16	W	Start address of trickle configuration in trickle memory
		VALID_DATA_END	15:0	W	Stop address of trickle configuration in trickle memory (last valid byte)
0xD050	0,1	CR_ITK_STRIPS_LCB_LINKS_00_MODULE_MASK_F_C_0			
		HCC_MASK	63:48	W	HCC* module mask
		ABC_MASK_HCC_E	47:32	W	Masks register commands with destination hcc_id = 0xE mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_D	31:16	W	Masks register commands with destination hcc_id = 0xD mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_C	15:0	W	Masks register commands with destination hcc_id = 0xC mask(i) <=> (abc_id = i)
0xD060	0,1	CR_ITK_STRIPS_LCB_LINKS_00_ABC_MODULE_MASK_B_8_0			
		ABC_MASK_HCC_B	63:48	W	Masks register commands with destination hcc_id = 0xB mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_A	47:32	W	Masks register commands with destination hcc_id = 0xA mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_9	31:16	W	Masks register commands with destination hcc_id = 0x9 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_8	15:0	W	Masks register commands with destination hcc_id = 0x8 mask(i) <=> (abc_id = i)
0xD070	0,1	CR_ITK_STRIPS_LCB_LINKS_00_ABC_MODULE_MASK_7_4_0			
		ABC_MASK_HCC_7	63:48	W	Masks register commands with destination hcc_id = 0x7 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_6	47:32	W	Masks register commands with destination hcc_id = 0x6 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_5	31:16	W	Masks register commands with destination hcc_id = 0x5 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_4	15:0	W	Masks register commands with destination hcc_id = 0x4 mask(i) <=> (abc_id = i)
0xD080	0,1	CR_ITK_STRIPS_LCB_LINKS_00_ABC_MODULE_MASK_3_0_0			
		ABC_MASK_HCC_3	63:48	W	Masks register commands with destination hcc_id = 0x3 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_2	47:32	W	Masks register commands with destination hcc_id = 0x2 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_1	31:16	W	Masks register commands with destination hcc_id = 0x1 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_0	15:0	W	Masks register commands with destination hcc_id = 0x0 mask(i) <=> (abc_id = i)
		...			

0xD170	0,1	CR_ITK_STRIPS_LCB_LINKS_00_LCB_3			
		L0A_BCR_DELAY	49:38	W	TTC BCR signal will be delayed by this many BCs
		L0A_FRAME_DELAY	37:34	W	By how many BCs to delay an L0A frame. Updating this register may result in brief loss of LCB lock, and some TTC L0A frames may be lost. Don't adjust this parameter while taking data.
		FRAME_PHASE	33:32	W	phase of LCB frame with respect to TTC BCR signal
		TRICKLE_BC_START	31:20	W	Determines the start of the allowed BC interval for low-priority LCB frames
		TRICKLE_BC_STOP	19:8	W	Determines the end of the allowed BC interval for low-priority LCB frames
		LCB_DESTINATION_MUX	5:4	W	Determines where the elink data is sent to: 00: command decoder (use same command encoding format as trickle configuration) 01: trickle memory (see phase2 documentation for command encoding format) 10: directly to LCB link (expecting software-encoded HCC*/ABC* frames) 11: (invalid, don't use)
		TRICKLE_TRIG_RUN	3	W	if enabled, trickle configuration is sent out continuously to the front-end (use together with TTC_GENERATE_GATING_EN for sending trickle configuration continuously during a specified BC range. See also BC_START, and BC_STOP fields.)
		TTC_L0A_ENABLE	2	W	enable generating L0A frames in response to TTC system signals
		TTC_GENERATE_GATING_-ENABLE	0	W	enables generating trickle gating signal in response to TTC BCR. TRICKLE_TRIG_RUN must also be enabled for the trickle configuration to work. (See also BC_START, and BC_STOP fields)
0xD180	0,1	CR_ITK_STRIPS_LCB_LINKS_-00_TRICKLE_TRIGGER_3	any	T	writing to this register issues a single trickle trigger
0xD190	0,1	CR_ITK_STRIPS_LCB_LINKS_00_TRICKLE_MEMORY_CONFIG_3			
		MOVE_WRITE_PTR	any	T	Writing to this register moves trickle configuration memory write pointer to WRITE_PTR address
		WRITE_PTR	47:32	W	Trickle configuration memory write pointer
		VALID_DATA_START	31:16	W	Start address of trickle configuration in trickle memory
		VALID_DATA_END	15:0	W	Stop address of trickle configuration in trickle memory (last valid byte)
0xD1A0	0,1	CR_ITK_STRIPS_LCB_LINKS_00_MODULE_MASK_F_C_3			
		HCC_MASK	63:48	W	HCC* module mask
		ABC_MASK_HCC_E	47:32	W	Masks register commands with destination hcc_id = 0xE mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_D	31:16	W	Masks register commands with destination hcc_id = 0xD mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_C	15:0	W	Masks register commands with destination hcc_id = 0xC mask(i) <=> (abc_id = i)
0xD1B0	0,1	CR_ITK_STRIPS_LCB_LINKS_00_ABC_MODULE_MASK_B_8_3			
		ABC_MASK_HCC_B	63:48	W	Masks register commands with destination hcc_id = 0xB mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_A	47:32	W	Masks register commands with destination hcc_id = 0xA mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_9	31:16	W	Masks register commands with destination hcc_id = 0x9 mask(i) <=> (abc_id = i)

		ABC_MASK_HCC_8	15:0	W	Masks register commands with destination hcc_id = 0x8 mask(i) <=> (abc_id = i)
0xD1C0	0,1	CR_ITK_STRIPS_LCB_LINKS_00_ABC_MODULE_MASK_7_4_3			
		ABC_MASK_HCC_7	63:48	W	Masks register commands with destination hcc_id = 0x7 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_6	47:32	W	Masks register commands with destination hcc_id = 0x6 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_5	31:16	W	Masks register commands with destination hcc_id = 0x5 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_4	15:0	W	Masks register commands with destination hcc_id = 0x4 mask(i) <=> (abc_id = i)
0xD1D0	0,1	CR_ITK_STRIPS_LCB_LINKS_00_ABC_MODULE_MASK_3_0_3			
		ABC_MASK_HCC_3	63:48	W	Masks register commands with destination hcc_id = 0x3 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_2	47:32	W	Masks register commands with destination hcc_id = 0x2 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_1	31:16	W	Masks register commands with destination hcc_id = 0x1 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_0	15:0	W	Masks register commands with destination hcc_id = 0x0 mask(i) <=> (abc_id = i)
ITK_STRIPS_R3 L1_LINKS					
0xD1E0	0,1	CR_ITK_R3L1_LINK_00_R3L1_0			
		FRAME_PHASE	3:2	W	phase of R3L1 frame with respect to TTC BCR signal
		L1_ENABLE	1	W	enables sending TTC L1 signals to the front-end
		R3_ENABLE	0	W	enables sending RoI R3 signals to the front-end
...					
0xD210	0,1	CR_ITK_R3L1_LINK_00_R3L1_3			
		FRAME_PHASE	3:2	W	phase of R3L1 frame with respect to TTC BCR signal
		L1_ENABLE	1	W	enables sending TTC L1 signals to the front-end
		R3_ENABLE	0	W	enables sending RoI R3 signals to the front-end
...					
ITK_STRIPS_LCB_LINKS					
0xD620	0,1	CR_ITK_STRIPS_LCB_LINKS_03_LCB_0			
		L0A_BCR_DELAY	49:38	W	TTC BCR signal will be delayed by this many BCs
		L0A_FRAME_DELAY	37:34	W	By how many BCs to delay an L0A frame. Updating this register may result in brief loss of LCB lock, and some TTC L0A frames may be lost. Don't adjust this parameter while taking data.
		FRAME_PHASE	33:32	W	phase of LCB frame with respect to TTC BCR signal
		TRICKLE_BC_START	31:20	W	Determines the start of the allowed BC interval for low-priority LCB frames
		TRICKLE_BC_STOP	19:8	W	Determines the end of the allowed BC interval for low-priority LCB frames
		LCB_DESTINATION_MUX	5:4	W	Determines where the elink data is sent to: 00: command decoder (use same command encoding format as trickle configuration) 01: trickle memory (see phase2 documentation for command encoding format) 10: directly to LCB link (expecting software-encoded HCC*/ABC* frames) 11: (invalid, don't use)

		TRICKLE_TRIG_RUN	3	W	if enabled, trickle configuration is sent out continuously to the front-end (use together with TTC_GENERATE_GATING_EN for sending trickle configuration continuously during a specified BC range. See also BC_START, and BC_STOP fields.)
		TTC_L0A_ENABLE	2	W	enable generating L0A frames in response to TTC system signals
		TTC_GENERATE_GATING_-ENABLE	0	W	enables generating trickle gating signal in response to TTC BCR. TRICKLE_TRIG_RUN must also be enabled for the trickle configuration to work. (See also BC_START, and BC_STOP fields)
0xD630	0,1	CR_ITK_STRIPS_LCB_LINKS_-03_TRICKLE_TRIGGER_0	any	T	writing to this register issues a single trickle trigger
0xD640	0,1	CR_ITK_STRIPS_LCB_LINKS_03_TRICKLE_MEMORY_CONFIG_0			
		MOVE_WRITE_PTR	any	T	Writing to this register moves trickle configuration memory write pointer to WRITE_PTR address
		WRITE_PTR	47:32	W	Trickle configuration memory write pointer
		VALID_DATA_START	31:16	W	Start address of trickle configuration in trickle memory
		VALID_DATA_END	15:0	W	Stop address of trickle configuration in trickle memory (last valid byte)
0xD650	0,1	CR_ITK_STRIPS_LCB_LINKS_03_MODULE_MASK_F_C_0			
		HCC_MASK	63:48	W	HCC* module mask
		ABC_MASK_HCC_E	47:32	W	Masks register commands with destination hcc_id = 0xE mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_D	31:16	W	Masks register commands with destination hcc_id = 0xD mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_C	15:0	W	Masks register commands with destination hcc_id = 0xC mask(i) <=> (abc_id = i)
0xD660	0,1	CR_ITK_STRIPS_LCB_LINKS_03_ABC_MODULE_MASK_B_8_0			
		ABC_MASK_HCC_B	63:48	W	Masks register commands with destination hcc_id = 0xB mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_A	47:32	W	Masks register commands with destination hcc_id = 0xA mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_9	31:16	W	Masks register commands with destination hcc_id = 0x9 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_8	15:0	W	Masks register commands with destination hcc_id = 0x8 mask(i) <=> (abc_id = i)
0xD670	0,1	CR_ITK_STRIPS_LCB_LINKS_03_ABC_MODULE_MASK_7_4_0			
		ABC_MASK_HCC_7	63:48	W	Masks register commands with destination hcc_id = 0x7 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_6	47:32	W	Masks register commands with destination hcc_id = 0x6 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_5	31:16	W	Masks register commands with destination hcc_id = 0x5 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_4	15:0	W	Masks register commands with destination hcc_id = 0x4 mask(i) <=> (abc_id = i)
0xD680	0,1	CR_ITK_STRIPS_LCB_LINKS_03_ABC_MODULE_MASK_3_0_0			
		ABC_MASK_HCC_3	63:48	W	Masks register commands with destination hcc_id = 0x3 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_2	47:32	W	Masks register commands with destination hcc_id = 0x2 mask(i) <=> (abc_id = i)

		ABC_MASK_HCC_1	31:16	W	Masks register commands with destination hcc_id = 0x1 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_0	15:0	W	Masks register commands with destination hcc_id = 0x0 mask(i) <=> (abc_id = i)
. . .					
0xD770	0,1	CR_ITK_STRIPS_LCB_LINKS_03_LCB_3			
		L0A_BCR_DELAY	49:38	W	TTC BCR signal will be delayed by this many BCs
		L0A_FRAME_DELAY	37:34	W	By how many BCs to delay an L0A frame. Updating this register may result in brief loss of LCB lock, and some TTC L0A frames may be lost. Don't adjust this parameter while taking data.
		FRAME_PHASE	33:32	W	phase of LCB frame with respect to TTC BCR signal
		TRICKLE_BC_START	31:20	W	Determines the start of the allowed BC interval for low-priority LCB frames
		TRICKLE_BC_STOP	19:8	W	Determines the end of the allowed BC interval for low-priority LCB frames
		LCB_DESTINATION_MUX	5:4	W	Determines where the elink data is sent to: 00: command decoder (use same command encoding format as trickle configuration) 01: trickle memory (see phase2 documentation for command encoding format) 10: directly to LCB link (expecting software-encoded HCC*/ABC* frames) 11: (invalid, don't use)
		TRICKLE_TRIG_RUN	3	W	if enabled, trickle configuration is sent out continuously to the front-end (use together with TTC_GENERATE_GATING_EN for sending trickle configuration continuously during a specified BC range. See also BC_START, and BC_STOP fields.)
		TTC_L0A_ENABLE	2	W	enable generating L0A frames in response to TTC system signals
		TTC_GENERATE_GATING_-ENABLE	0	W	enables generating trickle gating signal in response to TTC BCR. TRICKLE_TRIG_RUN must also be enabled for the trickle configuration to work. (See also BC_START, and BC_STOP fields)
0xD780	0,1	CR_ITK_STRIPS_LCB_LINKS_-03_TRICKLE_TRIGGER_3	any	T	writing to this register issues a single trickle trigger
0xD790	0,1	CR_ITK_STRIPS_LCB_LINKS_03_TRICKLE_MEMORY_CONFIG_3			
		MOVE_WRITE_PTR	any	T	Writing to this register moves trickle configuration memory write pointer to WRITE_PTR address
		WRITE_PTR	47:32	W	Trickle configuration memory write pointer
		VALID_DATA_START	31:16	W	Start address of trickle configuration in trickle memory
		VALID_DATA_END	15:0	W	Stop address of trickle configuration in trickle memory (last valid byte)
0xD7A0	0,1	CR_ITK_STRIPS_LCB_LINKS_03_MODULE_MASK_F_C_3			
		HCC_MASK	63:48	W	HCC* module mask
		ABC_MASK_HCC_E	47:32	W	Masks register commands with destination hcc_id = 0xE mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_D	31:16	W	Masks register commands with destination hcc_id = 0xD mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_C	15:0	W	Masks register commands with destination hcc_id = 0xC mask(i) <=> (abc_id = i)
0xD7B0	0,1	CR_ITK_STRIPS_LCB_LINKS_03_ABC_MODULE_MASK_B_8_3			
		ABC_MASK_HCC_B	63:48	W	Masks register commands with destination hcc_id = 0xB mask(i) <=> (abc_id = i)

		ABC_MASK_HCC_A	47:32	W	Masks register commands with destination hcc_id = 0xA mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_9	31:16	W	Masks register commands with destination hcc_id = 0x9 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_8	15:0	W	Masks register commands with destination hcc_id = 0x8 mask(i) <=> (abc_id = i)
0xD7C0	0,1	CR_ITK_STRIPS_LCB_LINKS_03_ABC_MODULE_MASK_7_4_3			
		ABC_MASK_HCC_7	63:48	W	Masks register commands with destination hcc_id = 0x7 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_6	47:32	W	Masks register commands with destination hcc_id = 0x6 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_5	31:16	W	Masks register commands with destination hcc_id = 0x5 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_4	15:0	W	Masks register commands with destination hcc_id = 0x4 mask(i) <=> (abc_id = i)
0xD7D0	0,1	CR_ITK_STRIPS_LCB_LINKS_03_ABC_MODULE_MASK_3_0_3			
		ABC_MASK_HCC_3	63:48	W	Masks register commands with destination hcc_id = 0x3 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_2	47:32	W	Masks register commands with destination hcc_id = 0x2 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_1	31:16	W	Masks register commands with destination hcc_id = 0x1 mask(i) <=> (abc_id = i)
		ABC_MASK_HCC_0	15:0	W	Masks register commands with destination hcc_id = 0x0 mask(i) <=> (abc_id = i)
ITK_STRIPS_R3 L1_LINKS					
0xD7E0	0,1	CR_ITK_R3L1_LINK_03_R3L1_0			
		FRAME_PHASE	3:2	W	phase of R3L1 frame with respect to TTC BCR signal
		L1_ENABLE	1	W	enables sending TTC L1 signals to the front-end
		R3_ENABLE	0	W	enables sending RoI R3 signals to the front-end
. . .					
0xD810	0,1	CR_ITK_R3L1_LINK_03_R3L1_3			
		FRAME_PHASE	3:2	W	phase of R3L1 frame with respect to TTC BCR signal
		L1_ENABLE	1	W	enables sending TTC L1 signals to the front-end
		R3_ENABLE	0	W	enables sending RoI R3 signals to the front-end
0xD820	0,1	STRIPS_R3_TRIGGER	any	T	(for tests only) simulate R3 trigger (issues 4-5 sequential triggers)
0xD830	0,1	STRIPS_L1_TRIGGER	any	T	(for tests only) simulate L1 trigger (issues 4-5 sequential triggers)
0xD840	0,1	STRIPS_R3L1_TRIGGER	any	T	(for tests only) simulate simultaneous R3 and L1 trigger (issues 4-5 sequential triggers)
MRO Dregisters					
0xF000	0	MROD_CTRL			
		OPTIONS	15:4	W	Extra options for MROD
		GOLTESTMODE	3:0	W	GOL Test Mode (emulate CSM): 0: Run Data Emulator when 1; 0: stop, load emulator fifo 1: Enable Circulate when 1; 0: send fifo data only once 2: Enable Triggered Mode when 1; 0: run continueously (no TTC) 3: Enable pattern generator when 1; 0: off
0xF010	0	MROD_EP0_CSMENABLE	23:0	W	EP0 CSM Data Enable channel 23-0
0xF020	0	MROD_EP0_EMPTYSUPPR	23:0	W	EP0 Set Empty Suppression channel 23-0
0xF030	0	MROD_EP0_HPTDCMODE	23:0	W	EP0 Set HPTDC Mode channel 23-0
0xF040	0	MROD_EP0_CLRFIFOS	23:0	W	EP0 Clear FIFOs channel 23-0

0xF050	0	MROD_EP0_EMULOADENA	23:0	W	EP0 Emulator Load Enable channel 23-0
0xF060	0	MROD_EP0_TRXLOOPBACK	23:0	W	EP0 Transceiver Loopback Enable channel 23-0
0xF070	0	MROD_EP0_TXCVRRESET	23:0	W	EP0 Transceiver Reset all channel 23-0
0xF080	0	MROD_EP0_RXRESET	23:0	W	EP0 Receiver Reset channel 23-0
0xF090	0	MROD_EP0_TXRESET	23:0	W	EP0 Transmitter Reset channel 23-0
0xF0A0	0	MROD_EP1_CSMENABLE	23:0	W	EP1 CSM Data Enable channel 23-0
0xF0B0	0	MROD_EP1_EMPTYSUPPR	23:0	W	EP1 Set Empty Suppression channel 23-0
0xF0C0	0	MROD_EP1_HPTDCMODE	23:0	W	EP1 Set HPTDC Mode channel 23-0
0xF0D0	0	MROD_EP1_CLRFIFOS	23:0	W	EP1 Clear FIFOs channel 23-0
0xF0E0	0	MROD_EP1_EMULOADENA	23:0	W	EP1 Emulator Load Enable channel 23-0
0xF0F0	0	MROD_EP1_TRXLOOPBACK	23:0	W	EP1 Transceiver Loopback Enable channel 23-0
0xF100	0	MROD_EP1_TXCVRRESET	23:0	W	EP1 Transceiver Reset all channel 23-0
0xF110	0	MROD_EP1_RXRESET	23:0	W	EP1 Receiver Reset channel 23-0
0xF120	0	MROD_EP1_TXRESET	23:0	W	EP1 Transmitter Reset channel 23-0
MROD Monitors					
0xF800	0	MROD_EP0_CSMH_EMPTY	23:0	R	CSM Handler FIFO Empty 23-0
0xF810	0	MROD_EP0_CSMH_FULL	23:0	R	CSM Handler FIFO Full 23-0
0xF820	0	MROD_EP0_RXLOCKED	23:0	R	EP0 Receiver Locked monitor 23-0
0xF830	0	MROD_EP0_TXLOCKED	23:0	R	EP0 Transmitter Locked monitor 23-0
0xF840	0	MROD_EP1_CSMH_EMPTY	23:0	R	CSM Handler FIFO Empty 23-0
0xF850	0	MROD_EP1_CSMH_FULL	23:0	R	CSM Handler FIFO Full 23-0
0xF860	0	MROD_EP1_RXLOCKED	23:0	R	EP1 Receiver Locked monitor 23-0
0xF870	0	MROD_EP1_TXLOCKED	23:0	R	EP1 Transmitter Locked monitor 23-0

Table B.3: FELIX register map BAR2.

B.2 Data Formats

B.2.1 CRToHost Block format

In Phase I FELIX, the ToHost Block format was defined in [25]. For Phase II, the blocksize is variable, and a multiple of 1024 bytes, and the chunk trailer is set to 32 bits. The block header format has changed to include the block size, as well as an indication that the trailer is 32 bit. The blocks are transferred by Wupper over DMA into a contiguous memory area, reserved by the cmem_rcc driver. Event fragments or other types of data arriving via the FrontEnd links or virtual E-Links are referred to as "chunks" and can have an arbitrary size.

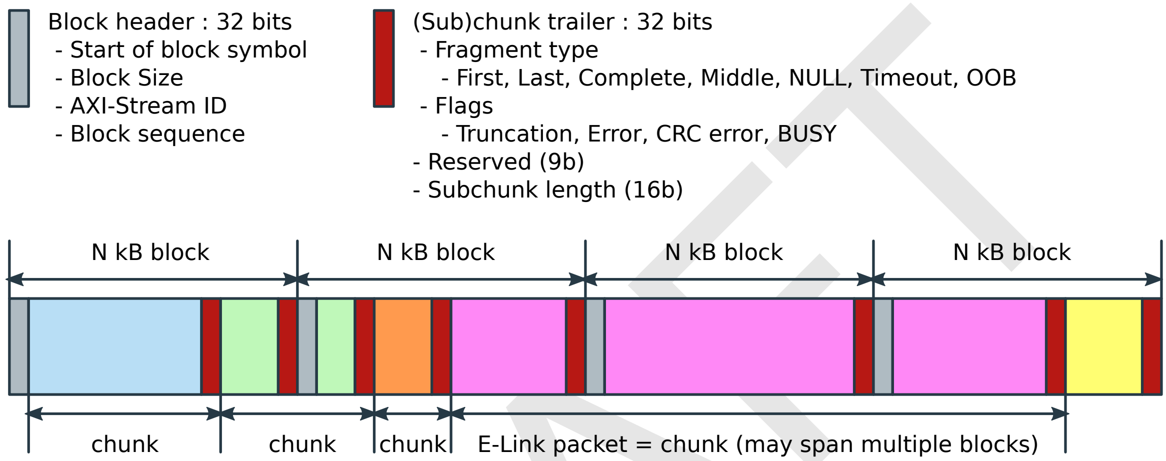


Figure B.1: FELIX ToHost Block format.

31–28	27–24	23–16	15–11	10–6	5–0
0xC	Block Size - 1	0xCE	Block Sequence	GBT ID	AXIs ID

Figure B.2: Block Header Format.

- **0xC** 4b, Header identifier
- **BlockSize - 1**: 4b, Block size in kB-1, 0: 1kB, 3: 4kB etc.
- **0xCE** 8b, Header identifier
- **Block Sequence** 5b, Incremental number per E-Link
- **GBT ID** 5b, Link index starting at 0 for every PCIe endpoint. For a 24 channel firmware with two PCIe endpoints, Link 12 will generate a GBT ID 0 in endpoint 1.
- **AXIs ID** 6b, Index of the E-Link on the AXI-Stream array. For GBT and lpGBT this number is equal to the Egroup * 8 + the Epath ID within the E-Group.

31–29	28	27	26	25	24–16	15–0
Type	T	E	C	B	reserved	(sub)chunk length in bytes

Figure B.3: Chunk Trailer Format.

- **Type** 3b:
  - 0: NULL Trailer, padding
  - 1: First part of a chunk consisting of more than one part
  - 2: Last part of a chunk consisting of more than one part
  - 3: Chunk consists of one part
  - 4: Middle part of a chunk, consisting of more than two parts
  - 5: Timeout trailer
  - 6: Reserved
  - 7: Out of band (OOB)
- **T** Truncation flag, indicating that a decoder truncated the data to a maximum length, or because the FIFO was full.
- **E** Framing error, Front-End data does not comply with the specified data format. For instance a missing SOP, EOP, or payload data not within SOP/EOP.
- **C** CRC error, if implemented by the decoder.
- **B** E-Link BUSY indication
- **reserved** 9b, reserved for future use.
- **Length** 16b, Length in bytes of the chunk of subchunk. If the chunk spans multiple blocks, only the sub-chunk length is given.

B.2.2 CRFromHost Data format

Each 256-bit block at the input of the CRFromHost represents a packet. In case of a 512-bit FIFO interface, two packets are sent simultaneously. Each packet consists of a 16 bit header followed by 240 bits of payload. Table B.4 shows how the bits are assigned in that packet.

Table B.4: Overview of the CRFromHost input data format..

type	header			payload		
bit number	255:251	250:245	244:240	239:232	...	7:0
length	5	6	5	8	...	8
name	link ID	AXIs ID	packet length	payload byte 0	...	payload byte 29

The fields in Table B.4 contain the following information:

- **link ID**: Contains the index of the link number, starting at 0 in every endpoint. If a firmware is built with 24 optical links and two PCIe endpoints, optical link 12 can be accessed through endpoint 1, link ID 0.
- **AXIs ID**: Corresponds with the E-link number in on the GBT or lpGBT frame, multiplied by the e-group. For GBT frames, the maximum number is 41, for lpGBT the maximum number is 17. If no E-links are available on the link, the AXI-Stream ID should be 0.
- **packet length**: The number of valid bytes in this 32-byte block. After the header, 30 payload byte positions are available, when the packet is longer, the header is repeated. If the message is shorter than 31 bytes, this field contains the length in bytes. If this block contains the beginning of a message that will be extended in the next block, the packet length field contains the value 31 (0x1F).

B.2.3 TTC ToHost Data format

Figure B.4 is a table version of the chunk format produced by the TTCToHost Virtual E-Link, containing information about each Level-1 Accept. Like any other ToHost data, the TTCToHost data format is packed as a chunk inside a block, see section B.2.1.

	31 … 24	23 … 16	15 … 8	7 … 0
	⇐ Byte 3 ⇒	⇐ Byte 2 ⇒	⇐ Byte 1 ⇒	⇐ Byte 0 ⇒
0	*reserved* / BCID	BCID	Length (26)	FMT (2)
1	XL1ID	L1ID	L1ID	L1ID
2	Orbit			
3	*reserved*		Trigger Type	
4	L0ID			
5	L1A Counter[31..0]			
6			L1A Counter[47.32]	

Figure B.4: TTC ToHost data format.

The contents of the packet can be described by a C/C++ struct type as a number of bitfields as shown below. Such a 'TTC-to-host' packet in memory can be cast directly to this type:

```
typedef struct {
        unsigned int format        : 8;
        unsigned int length        : 8;
        unsigned int bcid          : 12;
        unsigned int reserved0     : 4;
        union{
                unsigned int full_l1id : 32;
                struct {
                        unsigned int l1id  : 24;
                        unsigned int xl1id : 8;
                };
        };
        unsigned int orbit         : 32;
        unsigned int trigger_type  : 16;
        unsigned int reserved1     : 16;
        unsigned int l0id          : 32;
        unsigned long l1a_counter  : 48;
} __attribute__((packed)) TtcToHost_packet_t;
```

Listing B.1: TTC ToHost Data format as C struct.

B.2.4 BUSY ToHost Data format

The BUSY ToHost Virtual E-Link (see 8.3.19) produces a chunk of data on any change of BUSY.

	31 ... 24	23 ... 16	15 ... 8	7 ... 0
	⇐ Byte 3 ⇒	⇐ Byte 2 ⇒	⇐ Byte 1 ⇒	⇐ Byte 0 ⇒

Word	BCID	Trig Link	Trig AXIs_ID	Trig Val	*reserved*	E-Link	Soft	DMA	FIFO	Busy
0	BCID	Trig Link	Trig AXIs_ID	Trig Val	*reserved*	E-Link	Soft	DMA	FIFO	Busy
1	Orbit									

Figure B.5: BUSY ToHost data format.

Explanation of the bitfields:

- BCID: The Bunch Crossing ID at which the virtual E-Link was triggered. This functions as a timestamp (together with the Orbit counter) to match the BUSY event, to other events.
- Trig Link: Showing the physical link of the BUSY source, triggering this message.
  - If BUSY was triggered by an E-Link (BUSY-ON/BUSY-OFF) the physical link is inserted here.
  - If the source was different (soft, DMA or FIFO), these 5 bits will all be "11111", or decimal 31.
- Trig Axis_ID: E-link identification of the BUSY source:
  - In case of an E-Link (BUSY-ON) source, this field (6-bits) identifies the E-Link in the GBT or lpGBT frame which triggered BUSY. If BUSY was issued by a FULL mode link, this field is 0.
  - In case of another source (soft, DMA or FIFO), this field identifies the source (with Trig Link is 0x1F/31):
    * 0: DMA busy was asserted or deasserted.
    * 1: FIFO busy was asserted or deasserted.
    * 2: Soft busy was asserted or negated.
- Trig Val: Identify whether this message was triggered by assertion or negation of the BUSY source:
  - 0: BUSY was negated
  - 1: BUSY was asserted
- E-Link: Indication of any E-Link currently in BUSY state.
- Soft: Indication of SOFT busy assertion
- DMA: Indication of DMA busy assertion.
- FIFO: Indication of FIFO busy assertion.
- BUSY: Indication of the output of the BUSY signal
- Orbit: Orbit counter while this message was triggered. This functions as a timestamp (together with BCID) to match the BUSY event to other events in the data stream.

B.2.5 Default emulator chunk payload

The internal RAM based emulator on the FLX card can be filled with arbitrary chunk data. The format that can be understood by low level tools (fcheck) for data verification can be used to check the decoder, CRToHost, Wupper and the PCIe link. The data format shown below represents the default payload as bytes, read from the memory as *uint8_t*. This data format can be stored in the emulator ram by means of .COE files at build time, or at runtime by tools like elinkconfig and feconf.

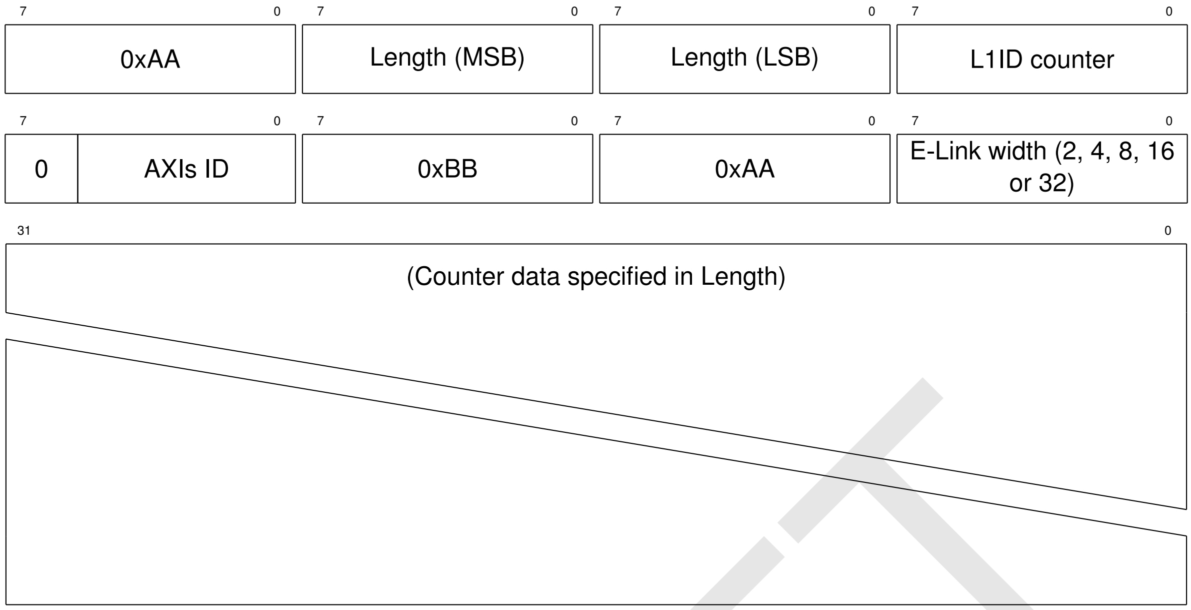


Figure B.6: Default Emulator payload.

Appendix C

Terms, Definitions and Glossary

List of Definitions

List of Requirements

List of Recommendations

List of Remarks

List of Tables

LIST OF FIGURES

C.1 GLOSSARY

ATLAS A Toroidal LHC Apparatus. i

AXI Advanced eXtensible Interface, widely used on Xilinx IP. AXI4-Stream is widely used in the FELIX project first. 24

BC Bunch Crossing, The CERN LHC bunch crossing clock frequency is 40.07897 MHz first. 71

Block Fixed section of memory with a specific formatting, headers and trailers first. B.29

BUSY A condition that can be raised from the FELIX system towards the central trigger processor in case buffers fill up and data aquisition must be halted first. 102, 104

DMA Direct Memory Access first. 113

FELIX Front End LInk eXchange. i

FIFO First In First Out, a type of memory to store data, also used to cross clock domains first. 23, 65

FLX128 Xilinx VCU128 / VU37P Development kit with FELIX firmware. 13

FLX712 FELIX Phase I PCIe card (BNL712) with FELIX firmware. 13

FromHost Direction of data communication, in ATLAS also referred to as Downlink. Data flows from the Host PC towards the FPGA first. 110

GBT VersatileLink GigaBitTransceiver, a protocol and chip (GBTx) with 4.8Gb/s communication and logical links (E-Links) first. 21

lpGBT low power GigaBitTransceiver, a successor of GBT with 9.6Gb/s Uplink, 2.56Gb/s Downlink and logical links (E-Links) first. 21

ToHost Direction of data communication, in ATLAS also referred to as Uplink. Data flows from the FPGA towards the Host PC first. 105

TTC Timing, Trigger and Control, a protocol to distribute timing and trigger information first. 102

Wupper An implementation of a PCIe DMA controller for Xilinx FPGAs first. 113