



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:	AT2-DQ-ES-0006	
EDMS Id:	2681548 v.1	
EDMS Url:	<a href="https://edms.cern.ch/document/2681548/1">https://edms.cern.ch/document/2681548/1</a>	
Version:	1.037	
Created:	January 12, 2021	
Last modified:	June 28, 2024	
<b>Prepared by:</b>	<b>Checked by:</b>	<b>Approved by:</b>
The FELIX Team	The FELIX Team	The ATLAS review committee

© 2024 CERN for the benefit of the ATLAS Collaboration.

Reproduction of this article or parts of it is allowed As specified in the CC-BY-4.0 license.

[INTENTIONALLY BLANK PAGE]

## REVISION HISTORY

Revision	Date	Author(s)	Description
0.001	2019-12-19	Frans Schreuder	Added some entities as a graphical symbol and wave-forms for axi stream
0.002	2019-12-19	Frans Schreuder	Added skeleton for RD53b decoder
0.003	2019-12-20	Frans Schreuder	Added several blocks, entities and moved around some text
0.004	2019-12-20	Frans Schreuder	Added full mode decoder entity
0.005	2020-01-07	Frans Schreuder	Split RD53b decoder and Aurora decoder in separate subsections
0.006	2020-01-08	Frans Schreuder	Described the DecodingGearBox
0.007	2020-01-09	Frans Schreuder	Added block diagram of Decoding Egroups and Epaths for (lp)GBT 8b10b mode
0.008	2020-01-10	Frans Schreuder	Added block diagram for Pixel ToHost e-path
0.009	2020-01-10	Frans Schreuder	Added description of the 8b10b decoder
0.010	2020-01-10	Frans Schreuder	Added progress bars to the different sections (Thank you LASP people for the idea)
0.011	2020-01-13	Jacopo Pinzino	added some informations about endeavour blocks
0.012	2020-01-14	Frans Schreuder	Added some missing encoder (skeleton) tex files
0.013	2020-01-14	Frans Schreuder	Added TTC Emulator
0.014	2020-01-14	Jacopo Pinzino	improving the Endeavour Encoder subsection
0.015	2020-01-16	Frans Schreuder	Added decoding egroup resources
0.016	2020-01-16	Frans Schreuder	regenerated pdf
0.017	2020-01-21	Elena Zhivun	Added description of the LCB protocol
0.018	2020-01-23	Frans Schreuder	Added atlas template
0.019	2020-01-24	Frans Schreuder	Changed information about CRC polynomial in FullMode.pdf
0.020	2020-01-27	Marius Wensing	starting work on the RD53B Decoder
0.021	2020-01-28	Frans Schreuder	Replaced verbatim with lstlisting in LCBEncoder.tex, it gave typesetting errors
0.022	2020-02-04	Marius Wensing	updating RD53B decoder entity and re-generated PDF
0.023	2020-02-04	Marius Wensing	adding entity for FromHost Central Router
0.024	2020-02-04	Jacopo Pinzino	add table
0.025	2020-02-10	Marius Wensing	more work on the RD53B decoder section
0.026	2020-02-10	Jacopo Pinzino	improvement endeavour encoder decoder part
0.027	2020-02-11	Frans Schreuder	Regenerated wupper documentation with rm4.9
0.028	2020-02-17	Marius Wensing	adding example waveform for CRFromHost input
0.029	2020-02-17	Marius Wensing	starting to document the FromHost Central Router
0.030	2020-02-18	Marius Wensing	adding resource usage for RD53B decoder
0.031	2020-02-20	Frans Schreuder	Fixed some issues in the EndeavourDecoder/Encoder documents (figures not found etc) Unified tables throughout the document
0.032	2020-02-20	Jacopo Pinzino	improvement endeavour encoder decoder part
0.033	2020-02-20	Frans Schreuder	regenerated pdf
0.034	2020-02-21	Frans Schreuder	Some minor updates to Endeavour Decoder / Encoder
0.035	2020-03-17	Frans Schreuder	Added section about TTC Encoder
0.036	2020-05-07	jacopo pinzino	added EndeavourDeglitcher in the Endeavour Decoder subsection of the Phase2_FM_specs
0.037	2020-05-07	jacopo pinzino	correct typo in Endeavour Decoder subsection of the Phase2_FM_specs
0.038	2020-05-12	jacopo pinzino	small grammatical corrections

0.039	2020-06-05	Elena Zhivun	Started on updating ITk Strips documentation
0.040	2020-06-05	Elena Zhivun	Started on LCB module documentation
0.041	2020-06-07	Elena Zhivun	Editing the documentation
0.042	2020-06-08	Elena Zhivun	Fixed bit ordering
0.043	2020-06-08	Elena Zhivun	Editing the text
0.044	2020-06-08	Elena Zhivun	Edit documentation
0.045	2020-06-09	Elena Zhivun	Added examples
0.046	2020-06-10	Elena Zhivun	Updated Strips protocol description
0.047	2020-06-16	Frans Schreuder	Built PDF
0.048	2020-06-24	Elena Zhivun	Added remark about BC gating interval
0.049	2020-06-25	Elena Zhivun	Updated remark about BC gating generation
0.050	2020-07-23	Nico Giangiacomi	Modified TTC Encoder table, removed useless TTCOptions
0.051	2020-11-19	Elena Zhivun	Update Strips module documentation
0.052	2021-01-12	Frans Schreuder	Added chapter about testing
0.053	2021-01-12	Frans Schreuder	Added related documents
0.054	2021-01-15	Frans Schreuder	Added section about FULL mode, added detailed toplevel schematic including all toplevel signals
0.055	2021-01-19	Frans Schreuder	Added register-map 5.0 as appendix
0.056	2021-01-21	Frans Schreuder	Added documentation for: * Wupper * Firmware flavours * Minor other modifications
0.057	2021-01-21	Frans Schreuder	Minor modifications in felix toplevel (detailed) drawing
0.058	2021-01-21	Marius Wensing	working on CRFromHost
0.059	2021-01-22	Frans Schreuder	Started section about CRTToHost
0.060	2021-01-26	Kai Chen	some texts are added in section 4/6/8, to be continued
0.061	2021-01-27	Marius Wensing	more work on CRFromHost chapter
0.062	2021-01-27	Frans Schreuder	Finished section about CRTToHost, added resources for CRFromHost
0.063	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.
0.064	2021-01-27	Frans Schreuder	Fixed capitalization of extension png=>PNG of file name
0.065	2021-01-28	Frans Schreuder	Added makefile for Wupper
0.066	2021-01-28	Frans Schreuder	Worked on Data Formats
0.067	2021-02-02	Kai Chen	add material for GBT/lpGBT in sec 8.6, and sec. 4
0.068	2021-02-02	Frans Schreuder	Updated front page and added glossaries
0.069	2021-02-02	Elena Zhivun	Add resource utilization for Strips links
0.070	2021-02-02	Elena Zhivun	Update the Strips documentation
0.071	2021-02-02	Elena Zhivun	Fix tables
0.072	2021-02-08	Nico Giangiacomi	Added 8b10bEncoder
0.073	2021-02-09	Kai Chen	Changes to the Section 8.6
0.074	2021-02-09	Kai Chen	Changes to the Section 6
0.075	2021-02-09	Kai Chen	Changes to the Section 8.6
0.076	2021-02-15	Frans Schreuder	Some work on Global Description
0.077	2021-02-16	Frans Schreuder	Added a chapter about AXI stream IDs per firmware flavour
0.078	2021-02-18	Frans Schreuder	Added description of HDLC Decoder
0.079	2021-02-18	Kai Chen	add fansink information for FLX-712
0.080	2021-02-19	Frans Schreuder	Described HDLC Encoder
0.081	2021-03-04	Frans Schreuder	Added description of the BUSY ToHost Virtual E-Link
0.082	2021-03-04	Frans Schreuder	Modified makefile to generate History.tex from git log



0.083	2021-03-04	Frans Schreuder	Reverted template/Makefile, now generate History.tex from MakeHistory.sh
0.084	2021-03-04	Frans Schreuder	Automatic version history from GIT with 0.xx numbering, alsu automate the version of the document this way
0.085	2021-03-05	Frans Schreuder	Separated TTC (Legacy) and BUSY sections, added LTI/TTC interface (empty placeholder)
0.086	2021-03-05	Frans Schreuder	Added section about the TTC ToHost Virtual E-Link
0.087	2021-03-15	Frans Schreuder	Described encoding entity
0.088	2021-05-07	Elena Zhivun	Added strips ToHost elink mapping
0.089	2021-05-07	Elena Zhivun	Place table at bottom of the page
0.090	2021-05-07	Elena Zhivun	Update LCBEncoder.tex
0.091	2021-08-12	Frans Schreuder	Publish document in User files
0.092	2021-08-19	Nayib Boukadida	Added Interlaken documentation
0.093	2021-08-25	Alexander Paramonov	legacy ttc
0.094	2021-08-27	Alexander Paramonov	legacy ttc
0.095	2021-08-27	Alexander Paramonov	legacy ttc
0.096	2021-08-27	Alexander Paramonov	legacy ttc
0.097	2021-08-27	Alexander Paramonov	legacy ttc
0.098	2021-08-27	Alexander Paramonov	legacy ttc
0.099	2021-08-27	Alexander Paramonov	legacy ttc
0.100	2021-08-27	Alexander Paramonov	legacy ttc
0.101	2021-08-30	Alexander Paramonov	legacy ttc
0.102	2021-08-31	Elena Zhivun	Update documentation
0.103	2021-08-31	Alexander Paramonov	legacy ttc
0.104	2021-08-31	Alexander Paramonov	legacy ttc
0.105	2021-08-31	Alexander Paramonov	legacy ttc
0.106	2021-09-01	Elena Zhivun	Update Endeavour documentation
0.107	2021-09-01	Alexander Paramonov	legacy ttc
0.108	2021-09-01	Alexander Paramonov	legacy ttc
0.109	2021-09-01	Elena Zhivun	Update strips documentation
0.110	2021-09-01	Elena Zhivun	Fix typo
0.111	2021-09-01	Elena Zhivun	Rename file
0.112	2021-09-01	Elena Zhivun	Add missing file
0.113	2021-09-01	Elena Zhivun	Update module diagram for Endeavour
0.114	2021-09-01	Elena Zhivun	Replace files
0.115	2021-09-01	Elena Zhivun	Fix typos
0.116	2021-09-07	Alexander Paramonov	legacy ttc

0.117	2021-09-08	Alexander monov	Para-	legacy ttc
0.118	2021-09-08	Alexander monov	Para-	legacy ttc
0.119	2021-09-13	Alexander monov	Para-	legacy ttc
0.120	2021-09-13	Alexander monov	Para-	legacy ttc
0.121	2021-09-13	Alexander monov	Para-	legacy ttc
0.122	2021-09-14	Alexander monov	Para-	legacy ttc
0.123	2021-09-14	Alexander monov	Para-	legacy ttc
0.124	2021-09-16	Alexander monov	Para-	64b66b decoder
0.125	2021-09-17	Alexander monov	Para-	64b66b decoder
0.126	2021-09-17	Alexander monov	Para-	64b66b decoder
0.127	2021-09-20	Alexander monov	Para-	64b66b decoder
0.128	2021-09-27	Alexander monov	Para-	64b66b decoder
0.129	2021-09-27	Alexander monov	Para-	64b66b decoder
0.130	2021-09-27	Alexander monov	Para-	64b66b decoder
0.131	2021-09-28	Frans Schreuder		fixed typo in wupper_structure diagram
0.132	2021-10-08	Alexander monov	Para-	64b66b decoder
0.133	2021-10-08	Alexander monov	Para-	64b66b decoder
0.134	2021-10-08	Alexander monov	Para-	64b66b decoder
0.135	2021-10-11	Alexander monov	Para-	64b66b decoder
0.136	2021-10-11	Alexander monov	Para-	64b66b decoder
0.137	2021-10-12	Alexander monov	Para-	64b66b decoder
0.138	2021-10-12	Alexander monov	Para-	64b66b decoder
0.139	2021-10-13	Alexander monov	Para-	64b66b decoder
0.140	2021-10-13	Alexander monov	Para-	64b66b decoder
0.141	2021-10-13	Alexander monov	Para-	64b66b decoder
0.142	2021-10-13	Alexander monov	Para-	64b66b decoder
0.143	2021-10-13	Alexander monov	Para-	64b66b decoder

0.144	2021-10-13	Alexander monov	Para-	64b66b decoder
0.145	2021-10-15	Alexander monov	Para-	ItkPix encoder
0.146	2021-10-15	Alexander monov	Para-	ItkPix encoder
0.147	2021-10-15	Alexander monov	Para-	ItkPix encoder
0.148	2021-10-18	Alexander monov	Para-	ItkPix encoder
0.149	2021-10-20	Alexander monov	Para-	ITkPix encoder
0.150	2021-10-21	Alexander monov	Para-	ITkPix encoder
0.151	2021-10-22	Alexander monov	Para-	comments from Will
0.152	2021-10-25	Alexander monov	Para-	LTI TTC
0.153	2021-10-26	Alexander monov	Para-	LTI TTC
0.154	2021-11-16	Frans Schreuder		History
0.155	2021-11-16	Frans Schreuder		Added GBT, IpGBT and FULL mode data emulator section
0.156	2021-11-16	Frans Schreuder		Added Busy/Xoff/Xon chapter
0.157	2021-11-17	Frans Schreuder		Created new toplevel block diagram, updated references
0.158	2021-11-17	Frans Schreuder		Added description of EncodingGearbox, removed RDMA functional description (now a citation to TWEPP21)
0.159	2021-11-19	Nayib Boukadida		Added interlaken documentation with figures
0.160	2021-11-22	Frans Schreuder		Removed 25GbLinksEncoder / Decoder (replaced by Interlaken)
0.161	2022-01-03	Ali Skaf		Update Phase2_FW_specs/text/TTCEmulator.tex
0.162	2022-01-03	Ali Skaf		Included in TTCEmulator.tex
0.163	2022-01-05	Alexander monov	Para-	LTI TTC
0.164	2022-01-05	Alexander monov	Para-	LTI TTC
0.165	2022-01-06	Alexander monov	Para-	fixed compilation
0.166	2022-01-06	Alexander monov	Para-	fixed compilation
0.167	2022-01-06	Alexander monov	Para-	ITk Pix Encoder
0.168	2022-01-10	Frans Schreuder		Updated resources on target FPGA, based on values estimated in FLX-1769
0.169	2022-01-10	Frans Schreuder		Added some comments Management and Reliability, removed some progress bars on sections that are already done
0.170	2022-01-11	Frans Schreuder		Added Organisation and ManagementAndReliability
0.171	2022-01-11	Frans Schreuder		Updated document history
0.172	2022-01-11	Frans Schreuder		Fixed organisation layout
0.173	2022-01-12	Frans Schreuder		Added section about Dynamic DMA channel selection in CRTtoHost

0.174	2022-01-13	William Panduro Vazquez	Update Firmware_Specs-metadata.tex
0.175	2022-01-13	Frans Schreuder	Added description of Housekeeping and clock and reset
0.176	2022-01-14	Frans Schreuder	Some corrections after Jos' comments
0.177	2022-01-14	Frans Schreuder	Added small versiond of the toplevel block diagram with a highlight where it can be found
0.178	2022-01-14	Frans Schreuder	Move figure to another tex file to fix the Wupper standalone document
0.179	2022-01-17	Frans Schreuder	Added clocking scheme
0.180	2022-01-17	Frans Schreuder	Merge branch 'master' of ssh://gitlab.cern.ch:7999/atlas-tdaq-felix/documents
0.181	2022-01-17	Frans Schreuder	Removed draft, pushed version to 1.0
1.000	2022-01-18	Frans Schreuder	Edited decoding figures to explain clock domains, added TC Link and TX Phase alignment to Link Wrapper
1.001	2022-01-18	Frans Schreuder	Updated LTI data formats from lti spec v1.1 (July 2021)
1.002	2022-01-18	Frans Schreuder	Updated screenshot of the CI pipelines
1.003	2022-01-19	Frans Schreuder	Removed "Instructions for this chapter" remarks
1.004	2022-01-20	Jose Guillermo Panduro Vazquez	Suggested pre-review changes.
1.005	2022-01-21	Frans Schreuder	Added requirement for 48-channel hardware option (LTDB), removed empty section headers
1.006	2022-01-25	Nayib Boukadida	Added Interlaken resource utilization
1.007	2022-03-25	Frans Schreuder	Added TTC options 6 and 7 to document. Option 7 includes xoff, option 6 was already there.
1.008	2022-08-11	Elena Zhivun	Fix the error in LOA command format
1.009	2022-09-28	Nayib	Updated (missing) Interlaken documentation
1.010	2022-09-28	Nayib	Small improvements in Interlaken documentation
1.011	2022-11-16	Frans Schreuder	Added new firmware flavours 12, 13, 14
1.012	2022-11-16	Frans Schreuder	Forgot to escape underscore
1.013	2023-01-24	Elena Zhivun	Fix the error in the figure
1.014	2023-01-27	Frans Schreuder	Added interrupt 4 as ToHost available for descriptor 4
1.015	2023-03-02	Frans Schreuder	Added section about LTI encoder, and added LTI encoder to the FULL mode (phase2) and Interlaken flavours
1.016	2023-03-29	Frans Schreuder	Updated description of interrupt masking
1.017	2023-05-09	Frans Schreuder	Added firmware flavour FULL-LTI (14) and renamed FULL to FULL-GBT (1)
1.018	2023-05-10	Frans Schreuder	Changed firmware flavour FULL-LTI to number 15, 14 was already taken by FELIG_PIXEL
1.019	2023-05-17	Frans Schreuder	Removed FULL_LTI flavour from GlobalDescription
1.020	2023-06-08	Frans Schreuder	Added description about Trickle descriptor in Wupper and updated register map 5.0
1.021	2023-06-20	Frans Schreuder	Added TTC option 8, HGTD FastCMD
1.022	2023-06-20	Frans Schreuder	Removed Phase1 TTC options from legacy TTC Wrapper
1.023	2023-07-07	Nayib	Updated info on the Interlaken Receiver
1.024	2023-07-07	Frans Schreuder	Updated register map 5
1.025	2023-09-25	Frans Schreuder	Updated the CRFromHost data format specification, as described in FLX-2294
1.026	2023-10-16	Carlo Alberto Gottardo	Add .gitlab-ci.yml file
1.027	2023-10-16	Carlo Alberto Gottardo	Add regmap submodule, CI pulls last version

1.028	2023-10-16	Carlo Alberto Gottardo	use relative path to regmap submodule
1.029	2023-10-16	Carlo Alberto Gottardo	Update .gitlab-ci.yml
1.030	2023-11-22	Frans Schreuder	Clarified AXIs ID for IpGBT and pixel flavour in Appendix B2
1.031	2024-03-19	Frans Schreuder	FLX-2369: Added extra TTC option with LFSR bit
1.032	2024-05-27	Frans Schreuder	Fixed some funny characters in the TTC emulator section
1.033	2024-05-27	Mark Donszelmann	Update .gitlab-ci.yml file
1.034	2024-05-27	Mark Donszelmann	Update .gitlab-ci.yml file
1.035	2024-06-10	Frans Schreuder	Upded regmap to 5.1
1.036	2024-06-28	Jochem Leijenhorst	Add gitignore
1.037	2024-06-28	Jochem Leijenhorst	Add description of HIFIFO and describe chunk header format


# TABLE OF CONTENTS

<b>Revision History</b>	<b>iii</b>
<b>Table of Contents</b>	<b>x</b>
<b>1 Related Documents</b>	<b>1</b>
<b>2 Global Description and Specification</b>	<b>2</b>
<b>2.1 Firmware Flavours</b>	<b>2</b>
2.1.1 E-Path IDs/ AXIs IDs	3
<b>2.2 Top level</b>	<b>4</b>
2.2.1 Transceiver and link wrapper	5
2.2.2 Encoding	5
2.2.3 Decoding	6
2.2.4 AXIs MUX (ToHost Fanout Selector)	7
2.2.5 CRFromHost: CentralRouter in FromHost direction	7
2.2.6 CRTToHost: CentralRouter in ToHost direction	7
2.2.7 ToHost Emulator	7
2.2.8 Wupper	8
2.2.9 Number instances per FPGA	9
<b>3 Busy Xon/Xoff specification</b>	<b>10</b>
<b>3.1 Overview</b>	<b>10</b>
<b>3.2 Reference Note: K-Characters in 8B/10B encoded links</b>	<b>10</b>
<b>3.3 Flow control (XOFF/XON) for FULL mode links</b>	<b>11</b>
3.3.1 Conditions leading to the assertion of flow control	11
3.3.2 Control and monitoring of XON and XOFF signal generation	11
3.3.2.1 Busy information in the datastream	13
<b>3.4 Propagation and management of BUSY conditions in GBT and FULL mode firmware</b>	<b>13</b>
3.4.1 Generation of BUSY at the request of a front-end data source	13
3.4.1.1 Inclusion of BUSY-ON/BUSY-OFF symbols in FULL mode packets	14
3.4.2 Generation of a BUSY condition on the basis of the state of the FELIX firmware	14
3.4.2.1 BUSY due to host memory saturation	14
3.4.2.2 BUSY PCIe FIFO saturation	14
3.4.2.2.1 NSW MicroMegas	15
3.4.2.2.2 NSW sTGC	15
3.4.3 Control and monitoring of the generation of BUSY conditions and the BUSY signal	17
3.4.3.1 FULL mode specific monitoring and control	17
3.4.3.2 GBT mode specific monitoring and control	17
3.4.3.3 Interrupt-based BUSY reporting	19
3.4.3.4 Virtual E-links for BUSY monitoring	19
3.4.3.4.1 FULL mode virtual E-link configuration	19
3.4.3.4.2 GBT mode virtual E-link configuration	20

<b>3.5</b>	<b>Propagation and management of BUSY and flow control (XOFF) in FELIX software</b>	<b>20</b>
3.5.1	GBT Mode	20
3.5.2	FULL mode	20
3.5.3	Software BUSY and XOFF Monitoring	20
3.5.4	Flow control from SW ROD to FELIX	21
3.5.5	BUSY handling and DCS	21
<b>4</b>	<b>External Interfaces (I/O)</b>	<b>23</b>
4.1	FrontEnd links	23
4.2	PCIe	23
4.3	TTC Interface	24
4.4	BUSY	24
4.5	100Gb/s Ethernet	24
<b>5</b>	<b>Target FPGA</b>	<b>25</b>
<b>6</b>	<b>Power and Cooling</b>	<b>27</b>
<b>7</b>	<b>Input/Output</b>	<b>29</b>
<b>8</b>	<b>Detailed Functional Description and Specification</b>	<b>32</b>
8.1	Introduction	32
8.2	Compatibility	32
8.3	Clocking scheme	33
8.4	Decoding	35
8.4.1	Introduction	35
8.4.2	Interfaces	35
8.4.2.1	Overview	35
8.4.2.1.1	GBT mode, 8b10b, HDLC	36
8.4.2.1.2	lpGBT mode, 8b10b	38
8.4.2.1.3	lpGBT mode, Pixel	38
8.4.2.2	Interface to CRTToHost	38
8.4.2.3	Interface to Link Wrapper	39
8.4.2.4	Interface to Wupper	39
8.4.3	Functional Description	39
8.4.4	Configuration	39
8.4.5	Status Indicators	39
8.4.6	Latency	40
8.4.7	Estimated Resource Usage	40
8.4.8	Decoding Gearbox	41
8.4.8.1	Introduction	41
8.4.8.2	Interfaces	41
8.4.8.2.1	Overview	41
8.4.8.2.2	Interface to GBT or lpGBT wrapper	41
8.4.8.2.3	Interface to Decoders	42
8.4.8.3	Functional Description	42
8.4.8.4	Configuration	42




8.4.8.5	Status Indicators . . . . .	43
8.4.8.6	Latency . . . . .	43
8.4.8.7	Error Handling . . . . .	43
8.4.8.8	Estimated Resource Usage . . . . .	43
8.4.9	StripDecoder . . . . .	45
8.4.10	Endeavour Decoder . . . . .	46
8.4.10.1	Introduction . . . . .	46
8.4.10.2	Interfaces . . . . .	46
8.4.10.3	Functional Description . . . . .	47
8.4.10.4	Error handling . . . . .	47
8.4.10.5	Estimated Resource Usage . . . . .	47
8.4.11	Aurora 64b/66b Decoder for ITkPix . . . . .	48
8.4.12	RD53B Decoder . . . . .	51
8.4.12.1	Introduction . . . . .	51
8.4.12.2	Interfaces . . . . .	51
8.4.12.2.1	Overview . . . . .	51
8.4.12.2.2	Interface to the Aurora Decoder . . . . .	51
8.4.12.2.3	Interface to the ToHost Central Router . . . . .	52
8.4.12.3	Functional Description . . . . .	52
8.4.12.3.1	Input stage . . . . .	52
8.4.12.3.2	Stream decoder . . . . .	52
8.4.12.3.3	Output multiplexer . . . . .	52
8.4.12.4	Configuration . . . . .	52
8.4.12.5	Status Indicators . . . . .	53
8.4.12.6	Latency . . . . .	53
8.4.12.7	Estimated Resource Usage . . . . .	53
8.4.13	8b10b E-Link decoder . . . . .	56
8.4.13.1	Introduction . . . . .	56
8.4.13.2	Interfaces . . . . .	56
8.4.13.2.1	Interface to DecodingGearBox . . . . .	56
8.4.13.2.2	Interface to ByteToAxisStream . . . . .	56
8.4.13.3	Functional Description . . . . .	57
8.4.13.3.1	Alignment . . . . .	57
8.4.13.3.2	8b10b decoding . . . . .	57
8.4.13.3.3	Framing error detection . . . . .	57
8.4.13.3.4	E-link busy assertion . . . . .	57
8.4.13.3.5	Deframing . . . . .	57
8.4.13.4	Configuration . . . . .	58
8.4.13.5	Status Indicators . . . . .	58
8.4.13.6	Latency . . . . .	58
8.4.13.7	Error Handling . . . . .	58
8.4.13.8	Estimated Resource Usage . . . . .	58
8.4.14	HDLC E-Link decoder . . . . .	59
8.4.14.1	Introduction . . . . .	59
8.4.14.2	Interfaces . . . . .	59
8.4.14.2.1	Generics . . . . .	59
8.4.14.2.2	Elink interface . . . . .	59
8.4.14.2.3	Interface to ByteToAxisStream . . . . .	60
8.4.14.3	Functional Description . . . . .	60
8.4.14.4	Configuration . . . . .	60
8.4.14.5	Status Indicators . . . . .	60
8.4.14.6	Latency . . . . .	60
8.4.14.7	Error Handling . . . . .	61
8.4.14.8	Estimated Resource Usage . . . . .	61



8.4.15	FULLModeDecoder	62
8.4.15.1	Introduction	62
8.4.15.2	Interfaces	63
8.4.15.2.1	Interface from LinkWrapper	63
8.4.15.2.2	Interface to CRTToHost	63
8.4.15.3	Functional Description	63
8.4.15.3.1	Flow control	64
8.4.15.3.2	CRC	65
8.4.15.4	Configuration	65
8.4.15.5	Status Indicators	65
8.4.15.6	Error Handling	65
8.4.15.7	Estimated Resource Usage	66
8.4.15.8	User Example design	66
8.4.16	Direct mode E-Link Decoder	68
8.4.16.1	Introduction	68
8.4.17	TTCToHost virtual E-Link	69
8.4.17.1	Introduction	69
8.4.17.2	Interfaces	69
8.4.17.2.1	Generics	69
8.4.17.2.2	Interface from TTC Wrapper	69
8.4.17.2.3	clock, reset and enable	70
8.4.17.2.4	Interface to Central Router ToHost	70
8.4.17.3	Functional Description	70
8.4.17.4	Configuration	70
8.4.17.5	Status Indicators	70
8.4.17.6	Latency	70
8.4.17.7	Error Handling	70
8.4.17.8	Estimated Resource Usage	70
8.4.18	BUSY virtual E-Link	72
8.4.18.1	Introduction	72
8.4.18.2	Interfaces	72
8.4.18.2.1	Generics	72
8.4.18.2.2	Interface from various BUSY sources	72
8.4.18.2.3	Timestamp inputs	73
8.4.18.2.4	clock, reset and enable	73
8.4.18.2.5	Interface to Central Router ToHost	73
8.4.18.3	Functional Description	73
8.4.18.4	Configuration	73
8.4.18.5	Status Indicators	73
8.4.18.6	Latency	73
8.4.18.7	Error Handling	74
8.4.18.8	Estimated Resource Usage	74
8.4.19	25 Gb/s Interlaken 	75
8.4.19.1	Interfaces	75
8.4.19.1.1	User Interface	75
8.4.19.1.2	Clock signals	76
8.4.19.2	Functionality	76
8.4.19.2.1	Burst frames	76
8.4.19.2.2	Meta frames	77
8.4.19.2.3	Encoder/Decoder	78
8.4.19.2.4	(De)Scrambler	78
8.4.19.3	Configuration	79
8.4.19.4	Latency	79
8.4.19.5	Status Indicators	79
8.4.19.6	Error Handling	80

8.4.19.7	Estimated Resource Usage . . . . .	80
<b>8.5</b>	<b>Encoding . . . . .</b>	<b>81</b>
8.5.1	Introduction . . . . .	81
8.5.2	Interfaces . . . . .	81
8.5.2.1	Overview . . . . .	81
8.5.2.2	Interface from CRFromHost . . . . .	82
8.5.2.3	Interface to LinkWrapper . . . . .	82
8.5.3	Functional Description . . . . .	82
8.5.4	Configuration . . . . .	82
8.5.5	Status Indicators . . . . .	82
8.5.6	Encoding Gearbox . . . . .	83
8.5.6.1	Introduction . . . . .	83
8.5.6.2	Interfaces . . . . .	83
8.5.6.2.1	Overview . . . . .	83
8.5.6.2.2	Interface to GBT or IpGBT wrapper . . . . .	83
8.5.6.2.3	Interface from Encoders . . . . .	84
8.5.6.3	Functional Description . . . . .	84
8.5.6.4	Configuration . . . . .	84
8.5.6.5	Status Indicators . . . . .	85
8.5.6.6	Latency . . . . .	85
8.5.6.7	Error Handling . . . . .	85
8.5.6.8	Estimated Resource Usage . . . . .	85
8.5.7	Endeavour Encoder . . . . .	86
8.5.7.1	Introduction . . . . .	86
8.5.7.2	Interfaces . . . . .	86
8.5.7.3	Functional Description . . . . .	87
8.5.7.4	Estimated Resource Usage . . . . .	87
8.5.8	ITkPix Encoder . . . . .	88
8.5.9	ITk Strips LCB Encoder . . . . .	90
8.5.9.1	Introduction . . . . .	90
8.5.9.2	Configuration storage submodule . . . . .	93
8.5.9.2.1	Configuration command. . . . .	93
8.5.9.3	LCB frame generator submodule . . . . .	93
8.5.9.4	Bypass frame aggregator submodule . . . . .	93
8.5.9.5	Trickle configuration memory . . . . .	95
8.5.9.6	Command decoder . . . . .	95
8.5.9.6.1	No operation. . . . .	95
8.5.9.6.2	IDLE command. . . . .	96
8.5.9.6.3	LOA command. . . . .	96
8.5.9.6.4	Fast command. . . . .	96
8.5.9.6.5	Register commands. . . . .	96
8.5.9.6.6	Block commands. . . . .	96
8.5.9.7	LCB sequence encoder . . . . .	96
8.5.9.8	LCB frame FIFO . . . . .	98
8.5.9.9	Trickle trigger generator . . . . .	98
8.5.9.10	LCB scheduler . . . . .	98
8.5.9.11	Examples . . . . .	99
8.5.9.11.1	Sending basic LCB commands via LCB Command elink and Command Decoder (ENCODING_ENABLE=1) . . . . .	99
8.5.9.11.2	Sending basic LCB commands via LCB Command elink and Bypass Frame Aggregator (ENCODING_ENABLE=0) . . . . .	99
8.5.9.11.3	Writing trickle configuration . . . . .	99
8.5.9.11.4	Issuing software-generated trickle trigger . . . . .	99
8.5.9.11.1	Single LCB elink. . . . .	99

8.5.9.11.2	Continuous trickle configuration.	100
8.5.9.11.3	All LCB elinks simultaneously.	100
8.5.9.11.4	All LCB elinks simultaneously with pre-buffering.	100
8.5.9.11.5	Trickle trigger during specified BC interval	100
8.5.9.12	Latency	100
8.5.9.13	Estimated Resource Usage	100
8.5.10	ITk Strips R3L1 Encoder	101
8.5.10.1	Introduction	101
8.5.10.2	Configuration storage submodule	103
8.5.10.2.1	Configuration command.	103
8.5.10.3	Frame synchronizer	103
8.5.10.4	R3 and L1 Frame generators	103
8.5.10.5	R3 and L1 Frame FIFOs	103
8.5.10.6	Bypass frame aggregator	104
8.5.10.7	R3L1 Scheduler	104
8.5.10.8	Latency	104
8.5.10.9	Estimated Resource Usage	104
8.5.11	8b10b Encoder	105
8.5.11.1	Introduction	105
8.5.11.2	Interfaces	105
8.5.11.2.1	Interface to AxiStreamToByte	105
8.5.11.2.2	Interface to EncodingGearBox	105
8.5.11.3	Functional Description	106
8.5.11.3.1	Overview	106
8.5.11.3.2	8b10b encoding	106
8.5.11.4	Configuration	106
8.5.11.5	Latency	106
8.5.11.6	Error Handling	106
8.5.11.7	Estimated Resource Usage	106
8.5.12	HDLC Encoder	107
8.5.12.1	Introduction	107
8.5.12.2	Interfaces	107
8.5.12.2.1	Generics	107
8.5.12.2.2	Interface from AxiStreamToByte	107
8.5.12.2.3	Interface to GBT/lpGBT E-Link	107
8.5.12.3	Functional Description	108
8.5.12.4	Configuration	108
8.5.12.5	Status Indicators	108
8.5.12.6	Latency	108
8.5.12.7	Error Handling	108
8.5.12.8	Estimated Resource Usage	108
8.5.13	Direct mode E-Link Encoder	109
8.5.13.1	Introduction	109
8.5.14	TTC Encoder	110
8.5.14.1	Introduction	110
8.5.14.2	Interfaces	110
8.5.14.3	Functional Description	110
8.5.14.3.1	TTC Delay and Extended testpulse	110
8.5.14.3.2	TTC Options	111
8.5.14.4	Configuration	112
8.5.14.5	Status Indicators	113
8.5.14.6	Latency	113
8.5.14.7	Error Handling	113

<b>8.6</b>	<b>LTI Encoder</b> 	<b>114</b>
<b>8.7</b>	<b>Link Wrapper</b> 	<b>115</b>
8.7.1	Introduction	115
8.7.2	Functional Description	115
8.7.2.1	GBT mode wrapper	115
8.7.2.2	IpGBT mode wrapper	116
8.7.2.2.1	TC Link and TX Phase alignment	117
8.7.2.3	Full mode wrapper	117
8.7.2.4	64b67b Link wrapper for 25G Interlaken	118
8.7.3	Configuration	118
8.7.4	Status Indicators	118
8.7.5	Latency	119
<b>8.8</b>	<b>GBT, IpGBT and AXI4 Stream Data Emulator</b>	<b>120</b>
8.8.0.1	Introduction	120
8.8.0.2	Interfaces	120
8.8.0.3	Functional Description	121
8.8.0.4	Configuration	121
8.8.0.5	Estimated Resource Usage	121
<b>8.9</b>	<b>TTC Emulator</b>	<b>123</b>
8.9.1	Introduction	123
8.9.2	Interfaces	123
8.9.3	Functional Description	123
8.9.4	Configuration	124
8.9.5	Status Indicators	124
8.9.6	Error Handling	124
8.9.7	Estimated Resource Usage	125
<b>8.10</b>	<b>Legacy TTC decoder</b>	<b>126</b>
<b>8.11</b>	<b>LTI/TTC Interface</b> 	<b>128</b>
<b>8.12</b>	<b>CRTtoHost: ToHost or Upstream Central Router</b>	<b>130</b>
8.12.1	Introduction	130
8.12.2	Interfaces	131
8.12.2.1	Overview	131
8.12.2.2	Interface from decoding	131
8.12.2.3	Interface to Wupper	132
8.12.3	Functional Description	132
8.12.3.1	CRTtoHostdm	132
8.12.3.1.1	ToHostAxisStreamController	133
8.12.3.1.2	Channel FIFO	133
8.12.3.2	CRTtoHost PCIeManager	135
8.12.3.2.1	PCIe DMA channel selection	135
8.12.3.3	CRTtoHost MUX	136
8.12.3.4	CRResetManager	136
8.12.4	Configuration	136
8.12.5	Status Indicators	136
8.12.6	Latency	136
8.12.7	Error Handling	137
8.12.8	Estimated Resource Usage	137

<b>8.13</b>	<b>CRFromHost: FromHost or Downstream Central Router</b>	<b>138</b>
8.13.1	Introduction	138
8.13.2	Interfaces	138
8.13.2.1	Interface to Wupper	138
8.13.2.2	Interface to the encoders	139
8.13.3	Functional Description	139
8.13.3.1	CRFromHost top-level	139
8.13.3.2	CRFromHost data manager	139
8.13.3.3	CRFromHost transfer manager	140
8.13.4	Configuration	140
8.13.4.1	Generics	140
8.13.4.2	Run-time configuration	140
8.13.5	Status Indicators	140
8.13.6	Latency	140
8.13.7	Estimated Resource Usage	140
<b>8.14</b>	<b>Wupper: PCIe DMA core and register map</b>	<b>141</b>
8.14.1	Introduction	141
8.14.2	Interfaces	142
8.14.2.1	Generics	142
8.14.2.2	fromHostFifo	143
8.14.2.3	toHostFifo	143
8.14.2.4	interrupt_call	144
8.14.2.5	Clocks and Resets	144
8.14.2.6	BUSY	144
8.14.2.7	PCIe	144
8.14.2.8	Register Map	145
8.14.3	Functional Description	145
8.14.4	DMA descriptors	146
8.14.5	Endless DMA with a circular buffer and wrap around	147
8.14.6	Trickle descriptor	150
8.14.7	Interrupt controller	151
8.14.8	Xilinx PCIe EndPoint Core	151
8.14.8.1	Xilinx AXI4-Stream interface	152
8.14.8.2	Configuration of the core	152
8.14.9	Status Indicators	152
8.14.10	Latency	152
8.14.11	Error Handling	152
8.14.12	Estimated Resource Usage	152
8.14.13	Simulation	153
<b>8.15</b>	<b>HouseKeeping</b>	<b>154</b>
8.15.1	Introduction	154
8.15.2	Interfaces	155
8.15.3	Functional Description	155
8.15.3.1	I2C interface	155
8.15.3.2	GenericConstantsToRegs	155
8.15.3.3	xadc_drp	156
8.15.3.4	dna	156
8.15.3.5	flash_wrapper	156
8.15.3.6	LMK03200_wrapper	156
8.15.3.7	pex_init	156
8.15.3.8	gc_multichannel_frequency_meter	156
8.15.3.9	Tachometer	156

<b>8.16</b>	<b>Clock And Reset</b>	<b>157</b>
8.16.1	Introduction	157
8.16.2	Interfaces	157
8.16.3	Functional Description	158
<b>9</b>	<b>Testing, Validation and Commissioning</b>	<b>159</b>
<b>9.1</b>	<b>Simulation</b>	<b>159</b>
9.1.1	UVVM	160
<b>9.2</b>	<b>Gitlab CI</b>	<b>160</b>
<b>9.3</b>	<b>Nightly firmware test on hardware</b>	<b>161</b>
<b>10</b>	<b>Firmware Management and Reliability Matters</b>	<b>162</b>
<b>10.1</b>	<b>Firmware Source Management and Release Plan</b>	<b>162</b>
10.1.1	Version numbers and releases	162
10.1.2	File name of a firmware build	162
<b>10.2</b>	<b>Consequences of Failures</b>	<b>163</b>
<b>10.3</b>	<b>Reliability measures in the FELIX firmware</b>	<b>164</b>
10.3.1	Redundant DMA channels and separation of DCS data	164
10.3.2	BUSY and XOFF mechanism	164
10.3.3	(E-)Link realignment and truncation	164
<b>11</b>	<b>Organization of Firmware Development</b>	<b>165</b>
11.0.1	Institutes contributing to FELIX firmware	166
11.0.2	Developers and their roles in the FELIX firmware	167
	<b>References</b>	<b>171</b>
	<b>Appendix A: Code Management</b>	<b>A.1</b>
	<b>Appendix B: Appendix</b>	<b>B.1</b>
<b>B.1</b>	<b>FELIX register map, version 5.1</b>	<b>B.2</b>
<b>B.2</b>	<b>Data Formats</b>	<b>B.40</b>
B.2.1	CRToHost Block format	B.40
B.2.2	CRFromHost Data format	B.42
B.2.3	TTC ToHost Data format	B.43
B.2.4	BUSY ToHost Data format	B.44
B.2.5	Default emulator chunk payload	B.45
	<b>Appendix C: Terms, Definitions and Glossary</b>	<b>C.1</b>
<b>C.1</b>	<b>Glossary</b>	<b>C.6</b>

---

# 1

---

## RELATED DOCUMENTS

The following documents are most relevant to the firmware to be described here:

- [FELIX Phase-II Readout Requirements Document](https://edms.cern.ch/document/2166531/1)  
*<https://edms.cern.ch/document/2166531/1>*
- [FELIX Phase-II System Specification Document](https://edms.cern.ch/document/2218837/1)  
*<https://edms.cern.ch/document/2218837/1>*
- [FELIX Phase-II Software Specification Document](https://edms.cern.ch/document/2681892/1)  
*<https://edms.cern.ch/document/2681892/1>*

Other useful links can be found below:

- [FELIX User Manual](https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/felix-user-manual/versions/Latest/)  
*<https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/felix-user-manual/versions/Latest/>*
- [FELIX Developer Manual](https://atlas-project-felix.web.cern.ch/atlas-project-felix/dev/felix-developer-manual.html)  
*<https://atlas-project-felix.web.cern.ch/atlas-project-felix/dev/felix-developer-manual.html>*
- [FLX-712 Hardware User manual](https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/docs/BNL-711_V2P0_manual.pdf)  
*[https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/docs/BNL-711\\_V2P0\\_manual.pdf](https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/docs/BNL-711_V2P0_manual.pdf)*
- [ATLAS FELIX website](https://atlas-project-felix.web.cern.ch)  
*<https://atlas-project-felix.web.cern.ch>*
- [General user documentation](https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/documentation.html)  
*<https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/documentation.html>*

# 2

## GLOBAL DESCRIPTION AND SPECIFICATION

While the FELIX firmware for ATLAS Phase-II upgrade will inherit most of the functionalities from the Phase-I firmware, the architecture has undergone as significant 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.

### 2.1 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 <a href="#">8.4.13</a> HDLC <a href="#">8.4.14</a> Direct <a href="#">8.4.16</a> TTCToHost <a href="#">8.4.17</a> BusyToHost <a href="#">8.4.18</a>	8b10b <a href="#">8.5.11</a> HDLC <a href="#">8.5.12</a> Direct <a href="#">8.5.13</a> TTC <a href="#">8.5.14</a>	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 or LTI	FULL <a href="#">8.4.15</a> TTCToHost <a href="#">8.4.17</a> BusyToHost <a href="#">8.4.18</a>	8b10b <a href="#">8.5.11</a> HDLC <a href="#">8.5.12</a> Direct <a href="#">8.5.13</a> TTC <a href="#">8.5.14</a> LTI-tx <a href="#">8.6</a>	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, with an option to transmit a copy of the LTI-TTC link data at 9.6Gb 8b10b with additional fields for XOFF
2: LTDB	GBT	8b10b <a href="#">8.4.13</a> HDLC <a href="#">8.4.14</a> Direct <a href="#">8.4.16</a> TTCToHost <a href="#">8.4.17</a> BusyToHost <a href="#">8.4.18</a>	8b10b <a href="#">8.5.11</a> HDLC <a href="#">8.5.12</a> Direct <a href="#">8.5.13</a> TTC <a href="#">8.5.14</a>	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	IpGBT	HDLC (EC/IC) <a href="#">8.4.14</a> Aurora <a href="#">8.4.11</a> TTCToHost <a href="#">8.4.17</a> BusyToHost <a href="#">8.4.18</a>	RD53A/B <a href="#">8.5.8</a> TTC <a href="#">8.5.14</a> HDLC (IC/EC) <a href="#">8.5.12</a>	The Pixel flavour was designed to read out the ITk Pixel detector over IpGBT 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) <a href="#">8.4.14</a> Endeavour (EC) <a href="#">8.4.10</a> 8b10b <a href="#">8.4.13</a> , <a href="#">8.4.9</a> TTCtoHost <a href="#">8.4.17</a> BusyToHost <a href="#">8.4.18</a>	HDLC (EC) <a href="#">8.5.12</a> Endeavour (EC) <a href="#">8.5.7</a> LCB <a href="#">8.5.9</a> R3L1 <a href="#">8.5.10</a>	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) <a href="#">8.4.14</a> 8b10b <a href="#">8.4.13</a> Direct <a href="#">8.4.16</a> TTCtoHost <a href="#">8.4.17</a> BusyToHost <a href="#">8.4.18</a>	8b10b <a href="#">8.5.11</a> HDLC <a href="#">8.5.12</a> Direct <a href="#">8.5.13</a> TTC <a href="#">8.5.14</a>	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. The LPGBT flavour will include encoding and decoding schemes for the HGTD
10: INTERLAKEN	64b67b	ToHost Interlaken, FromHost LTI <a href="#">8.4.19</a>	LTI-tx <a href="#">8.6</a>	The Interlaken Flavour has 24x 25.78125 Gb/s Interlaken links in ToHost direction. Note that no more than 12 links can be fully occupied as otherwise the PCIe Gen4 bandwidth will be saturated. As encoders, the Interlaken flavour implements the TTC-LTI encoder, a copy of the received LTI frame but with additional XOFF bits.
12: HGTD_LUMI	lpGBT	6b8b		
13: BCMPRIME	lpGBT			
14: FELIG-PIXEL	lpGBT-FE			

**Table 2.1:** 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 legacy MRODs fail during Run 3. [\[3\]](#)
- 11: FELIG LPGBT, [\[1\]](#) Front-End emulator for lpGBT 8b10b operation
- 14: FELIG Pixel, [\[1\]](#) Front-End emulator for lpGBT / ItkPix operation

## 2.1.1 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 by means of the header in the FromHost data format (see [??](#)), and the block header in the ToHost data format (see [B.3](#))

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 0: LTI-tx	
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 <a href="#">8.5.9</a> and <a href="#">8.5.10</a>
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: LTI-tx	No logical links on top of Interlaken

**Table 2.2:** E-Link configurations and AXIs IDs for the Firmware Flavours.

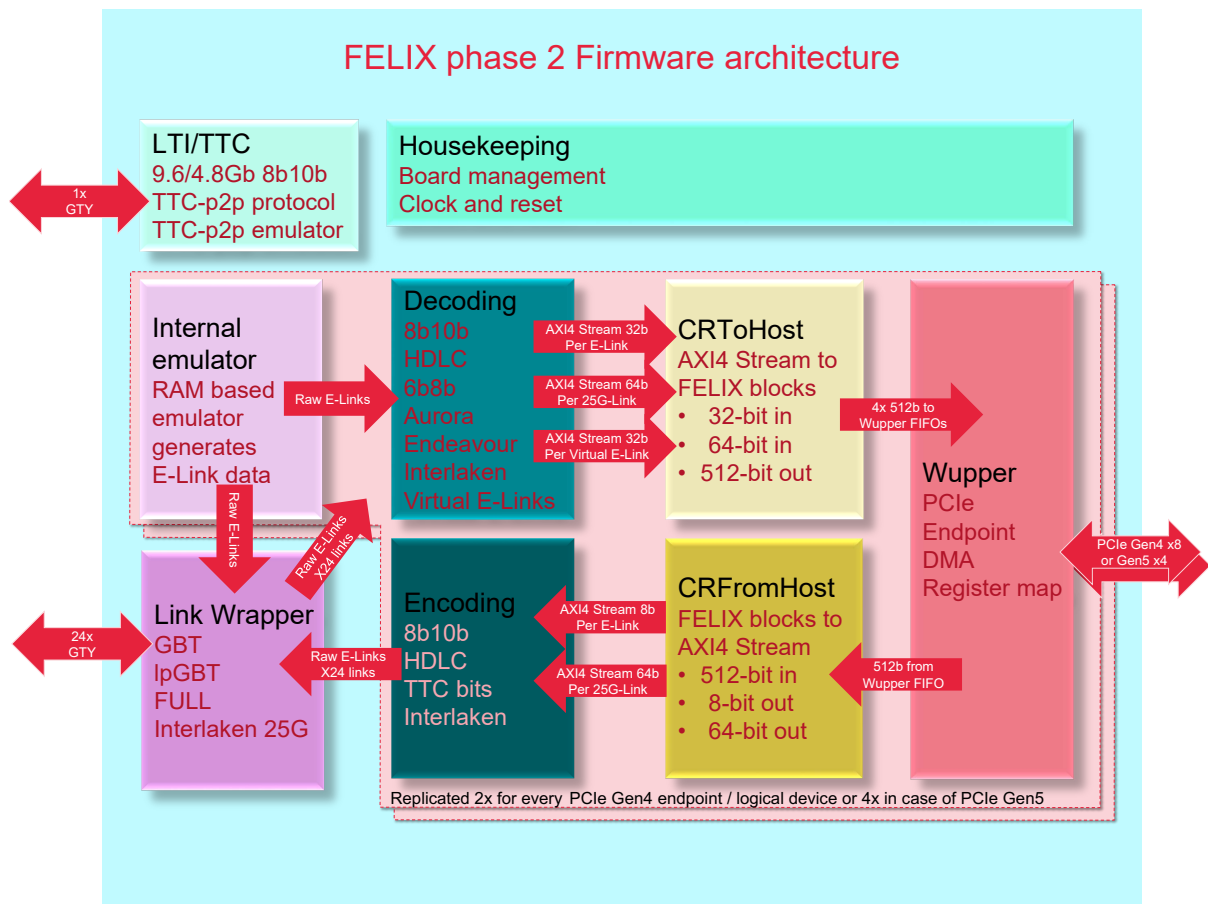
Table [2.2](#) shows the available AXIs IDs which are mapped on the physical links. Every link and its associated E-Links/AXIs IDs are replicated by the number of physical optical links in the build, so the encoder / decoder is not only addressed by the AXIs ID, but also by the GBT ID, which is the number 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

## 2.2 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 2.1:** The FELIX firmware top level block diagram using PCIe Gen4. On Gen5 capable hardware, the diagram will have 4 endpoints, each with a PCIe Gen5x4 link.

## 2.2.1 TRANSCEIVER AND LINK WRAPPER

- Interfaces to electrical to optical and optical to electrical transceivers
- TX: GBT or IpGBT scrambling: input GBT or IpGBT E-Link frames @ BC frequency, 25Gb/s links (Aurora or Interlaken), 10Gb/s links
- RX
  - For GBT or IpGBT descrambling, FEC handling, output GBT or IpGBT 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 25.78125 Gb/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 [8.7](#)

## 2.2.2 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 of 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 IpGBT E-link frames @ BC frequency, TTC virtual E-link, 64b67b encoded data for 25G Interlaken links. Depending on the firmware flavour (See section 2.1) 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 25.78125 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

A more detailed description of the Encoding block is given in 8.5

## 2.2.3 DECODING

- Inputs:
  - GBT frames, using E-Links. These E-links can carry multiple protocols such as 8b/10b, HDLC or direct (no encoding) mode.
  - IpGBT 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 8.4.15
  - 25Gb/s links Interlaken links, the raw (scrambled) 67b data is decoded in a submodule of the decoding block. 8.4.19

GBT or IpGBT 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 Multiplexer 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) 8.12. 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 IpGBT is encoded depending on the specification of the subdetector / frontend. A firmware flavour (See section 2.1) 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 [8.4.11](#).
- Endeavour 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 [8.4](#)

## 2.2.4 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

## 2.2.5 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

## 2.2.6 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 [8.14](#)).

## 2.2.7 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

## 2.2.8 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 TTC-P2P, if present, and jitter cleaning of this clock
  - Can also generate clock without presence of TTC or TTC-P2P

### 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 Monitoring and outputs this via LEMO output or via PON

## 2.2.9 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 Gen4 interface or, for PCIe Gen5, 4, each servicing a 4 lane PCIe interface

# 3

## BUSY XON/XOFF SPECIFICATION

### 3.1 OVERVIEW

This chapter contains a functional specification for the FELIX firmware and software with respect to BUSY propagation and flow control. The majority of the section will deal with the firmware implementation, with the final section detailing the proposed integration with FELIX software and connected clients.

In the context of this document, flow control specifically refers to XON and XOFF<sup>1</sup> signals used to throttle the transmission of data from the front-end to FELIX to prevent buffer overflow.

The BUSY signal, which is either asserted by FELIX over its output LEMO connector in case of the legacy (phase 1) TTC system or through a bit in the TTC/LTI interface, see Fig. 8.68, will be asserted if at least one "BUSY condition" (i.e. a firmware condition that should give rise to assertion of the BUSY signal) has been raised. The signal will be de-asserted once there is no BUSY condition satisfied. Possible sources of BUSY include BUSY-ON or BUSY-OFF commands received from front-ends via input links and the internal state of the firmware and software. BUSY assertion expresses an emergency situation with impending buffer overflow, resulting in all ATLAS data taking being paused, and is not intended (and should not be used) for normal flow control.

The implementation of BUSY handling is required in GBT, FULL, lpGBT, Interlaken, PIXEL and STRIP mode firmware, while XON/XOFF is only required for FULL mode and Interlaken firmware. This is because (lp)GBT front-ends are not expected to have sufficient buffering capacity to be able to implement any meaningful flow control. For FULL mode firmware it is expected that the BUSY conditions based on the state of the firmware will not be active in normal use, with XOFF the preferred method of stopping dataflow. However, the mechanism will still be implemented and retained as an option.

Note: most descriptions in this chapter assume a FLX712 board (i.e. with two Wupper cores and associated link counts). However, the overall implementation is unchanged for the FLX709, FLX128 or FLX181. The main differences are the single Wupper core and reduced link count. The FLX181 and FLX128 boards do not have a LEMO output for BUSY propagation, but it will be assumed that the LTI/TTC interface will be used instead.

### 3.2 REFERENCE NOTE: K-CHARACTERS IN 8B/10B ENCODED LINKS

The the 8B/10B encoding standard implemented by FELIX provides a reference for control symbols to be exchanged with front-end systems. These symbols, implemented as K-characters, are used to indicate: the

<sup>1</sup> XOFF: "transmission off": stop sending data, XON: "transmission on": resume sending data



start of a data packet (SOP); the end of a data packet (EOP); a request from the front-end to assert the BUSY signal (BUSY-ON); and removal of the BUSY condition caused by a previous BUSY-ON request (BUSY-OFF)<sup>2</sup>. Table 8.13 lists the K-characters associated with these control symbols<sup>3</sup>.

**Table 3.1:** K-characters used for 8B/10B coded links. BUSY-ON/OFF arrive from the front-end in both FULL mode and GBT mode cases..

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 (from front-end), XOFF (to front-end for FULL mode)
K28.3	0x7c	BUSY-OFF (from front-end), XON (to front-end for FULL mode)

### 3.3 FLOW CONTROL (XOFF/XON) FOR FULL MODE LINKS

FULL mode links (from front-end) operate at 9.6 Gb/s, with 8B/10B coded data. Flow control is implemented via dedicated GBT links from FELIX to the front-end, using the XOFF/XON K-characters specified in Table 3.1 as control symbols. The FULL mode firmware supports up to 24 FULL mode input links from front-end and, per set of 12 links, at least one GBT output link towards the front-end. The GBT link is clocked with the clock derived from the LHC bunch crossing clock and supplies this clock to the front-end. It also transfers bit patterns associated with TTC information, such as L1A, BCR, ECR (typically via an 8-bit E-link, with fixed latency).

The GBT links will be used to transfer XON and XOFF signals to the front-ends by means of 24 2-bit E-links<sup>4</sup>, i.e. one per incoming FULL mode link from the front-end. The same K-characters are used as XOFF/XON control symbols as for the BUSY-ON and BUSY-OFF requests sent from the front-end to FELIX, as indicated in Table 8.13. The proposed connectivity schema for the links described above is presented in Figure 3.1.

#### 3.3.1 CONDITIONS LEADING TO THE ASSERTION OF FLOW CONTROL

The data arriving on each FULL mode link flow into a 16 kB deep FIFO. Each FIFO has configurable high and low level watermarks. If the FIFO contains more data than indicated by the high watermark, FELIX will exert backpressure by forwarding an XOFF control symbol toward the front-end via an E-link in a GBT link associated with the FIFO. An XON control symbol will be sent via the same E-link once the amount of data has decreased below that indicated by the low watermark. The different high and low watermarks will introduce a form of hysteresis, meaning that there is minimal risk of oscillation of XON/XOFF signals.

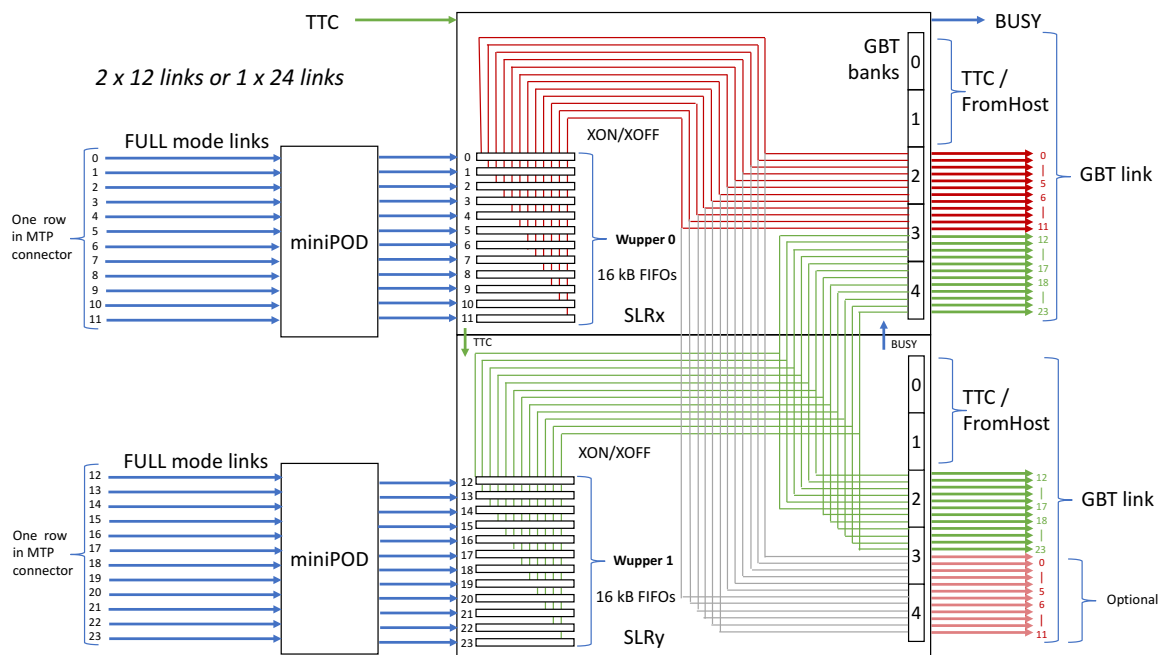
#### 3.3.2 CONTROL AND MONITORING OF XON AND XOFF SIGNAL GENERATION

As explained in the previous section, the generation of XON and XOFF signals in FULL mode is controlled by the high and low watermarks of the 16 kByte FIFOs at the input of each FULL mode link. The values of these watermarks should be configurable for each input FIFO, with a resolution of 1 kByte. Therefore, for each Wupper engine the registers need to contain 12x4-bit fields to set the low watermark values and 12x4-bit fields to set the high watermarks. The fill status of these FIFOs will be monitored via a pair of 12 bit fields,

<sup>2</sup>SOB, "Start-Of-BUSY" and EOB, "End-Of-BUSY" have been, but should no longer be, used as alternatives for BUSY-ON and BUSY-OFF.

<sup>3</sup>The SOP K-character is the comma character defined for the serializer core that forces 32-bit alignment.

<sup>4</sup>All E-links not used for forwarding TTC bit patterns can be used to send messages to the front-end, with the restriction that E-links used for XON-XOFF signalling need to use 8B/10B coding (i.e. use of HDLC or Direct mode is not possible).



**Figure 3.1:** Connections of 24 FULL mode links from the front-end and E-link assignment for GBT links to the front-end, using two fibre assemblies that each connect one row in a MTP48 feedthrough to a single miniPOD. The upper GBT link can be used for XON-XOFF signalling for all 24 links. With the optional E-links implemented the lower GBT link can also be used for this purpose, but with the E-link assignment adapted to use the lower set of links. It is also possible to use the two GBT links in conjunction with two independent sets of 12 FULL mode links. The E-links specified are 80Mb/s (2-bit) wide. The numbers in the rectangular blocks are GBT bank numbers..

indicating whether the fill level is above the high or low watermarks. The register bit values will latch when above the high watermark, and remain until reset by writing to the register.

To facilitate testing of XOFF and XON handling by front-ends, XOFF and XON generation under software control should be possible by means of setting bits in bit fields available for this purpose. Two sets of 12-bit fields are needed for this purpose per Wupper engine. In order to test the ability to generate XOFF and XON on the basis of the amount of data in the input FIFOs it should be possible to halt and restart the Wupper engine under software control. This is possible if tests use "single-shot" DMA transfers. An overview of all bit fields needed in the configuration registers for one Wupper engine is presented in Table 3.2.

**Table 3.2:** Bit fields in the configuration registers of a single Wupper engine (i.e. two per FELIX card) for control and monitoring of the generation of XON and XOFF signals..

Type of contents	Number of bits	Type of access
Low watermark	12 x 4 bits	Write / Read
High watermark	12 x 4 bits	Write / Read
Filling level higher than low / high watermark	2 x 12 bits	Read
High watermark has been crossed upward (latched)	12 bits	Read / Write (for reset)
Generation of XOFF/XON	12 bits	Write only
XOFF Enable	12 bits	Write / Read

### 3.3.2.1 BUSY INFORMATION IN THE DATASTREAM

The busy state of the front end is included in the data stream towards the FELIX host. If the front end has sent start of busy (SOB), the firmware will indicate this by adding an out of band trailer (0xE05C) after every regular trailer. In the 32 bit trailer format, the out of band trailer has been replaced with a 'B' or Busy bitfield in the trailer.

In the felixcore application this character can be interpreted and translated into a virtual elink that can be subscribed to.

## 3.4 PROPAGATION AND MANAGEMENT OF BUSY CONDITIONS IN GBT AND FULL MODE FIRMWARE

FELIX may assert BUSY due either to a request from the front-end (BUSY-ON) or due to FELIX buffers/FIFOs filling up, meaning dataflow cannot continue. In this section both paths will be specified for both GBT and FULL mode.

### 3.4.1 GENERATION OF BUSY AT THE REQUEST OF A FRONT-END DATA SOURCE

The receipt of a BUSY-ON control symbol via a FULL mode link or GBT mode E-link will cause a BUSY condition. Once the BUSY condition for a link has been raised the first BUSY-OFF control symbol received will remove it. *Front-end systems should implement protection such that there is a minimum 6 clock cycles (at 240 MHz) between transitions in state, corresponding to 1 bunch crossing.*

If at least one BUSY condition exists the BUSY signal will be asserted by FELIX through its LEMO connector on the FLX712 card<sup>5</sup> or signalled via the LTI/TTC interface. Once all BUSY conditions have been removed the BUSY signal will be de-asserted. In Table 3.3 the bit fields in the configuration registers associated with BUSY generation are listed. The fields are included in the configuration register specification for software control and monitoring of BUSY conditions in Section 3.4.3 (FULL mode) and Section 3.4.3.2 (GBT mode).

<sup>5</sup>The BUSY output is an open collector output, it is pulled to ground upon assertion.

**Table 3.3:** Bit fields in the configuration registers for control of assertion and de-assertion of the BUSY signal..

Type of contents	Number of bits	Type of access
Generate BUSY condition	1 bit	Write / Read
BUSY condition asserted (each bit is associated with an E-link)	24 x 57 bits	Read
Reset BUSY conditions	1 bit	Write

#### 3.4.1.1 INCLUSION OF BUSY-ON/BUSY-OFF SYMBOLS IN FULL MODE PACKETS

A potentially useful feature for extra robustness is that BUSY-ON and BUSY-OFF control symbols can be sent by FULL mode front-ends to FELIX at any time, and can therefore also be inserted into data packets themselves. This can be exploited in the FULL mode case as each data packet includes in the trailer a BUSY-ON or a BUSY-OFF control symbol, which should reflect whether a BUSY condition request is active. This control symbol can then be handled by the firmware like normal BUSY-ON or a BUSY-OFF control symbols, meaning that the loss of a single BUSY-ON or BUSY-OFF control symbol can be tolerated. Note, the control symbols are stripped out by the firmware and won't be visible to external clients. A similar mechanism has not been implemented in the GBT case, as it was understood it could not be supported by front-ends.

### 3.4.2 GENERATION OF A BUSY CONDITION ON THE BASIS OF THE STATE OF THE FELIX FIRMWARE

The transfer of data from FELIX hardware to the host server is routed through a PCIe FIFO. This is emptied continuously via DMA into a circular buffer in host memory as long as there is space for transfers to be enacted. Generation of a BUSY condition from this chain is possible in two locations: the amount of host memory remaining (indicated by the value of the circular buffer's read/write pointers); and the fill state of the PCIe FIFO itself. For GBT mode this is considered a required feature, but for FULL mode it is expected that the XOFF mechanism will avoid the need to assert BUSY from this source. That said, the BUSY chain will still be implemented in FULL mode and it will be retained as an configurable option.

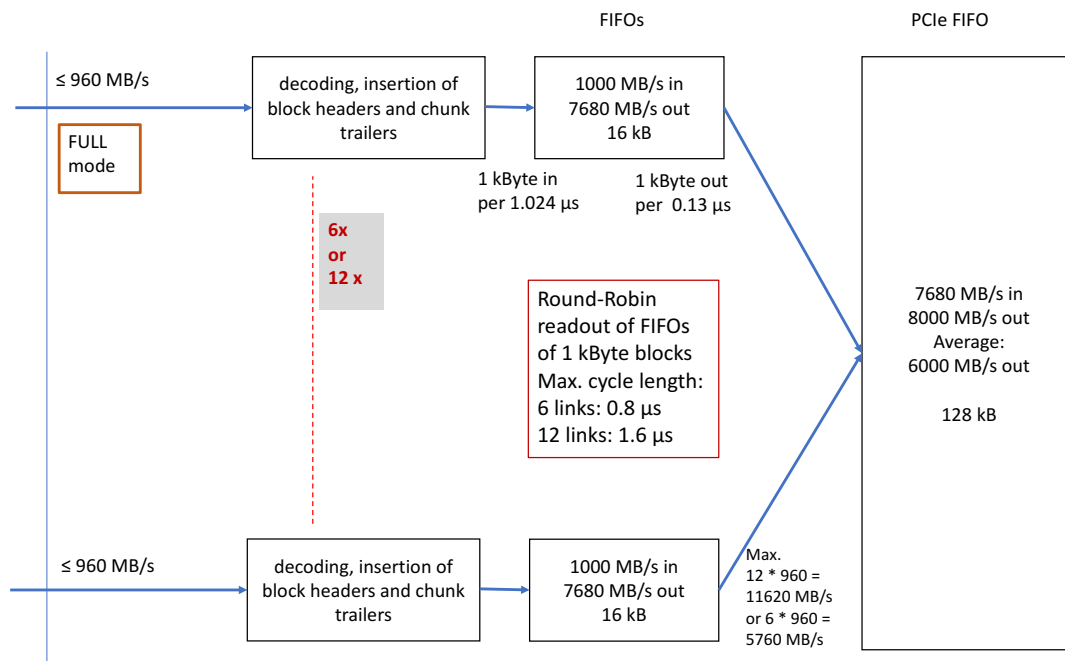
#### 3.4.2.1 BUSY DUE TO HOST MEMORY SATURATION

As mentioned above, the amount of free memory in the host is defined by the values of the read and write pointers. If difference between these two values becomes smaller than a configurable high watermark FELIX will assert BUSY. Given the amount of available memory in modern server PC's this value can be configured generously, with up to 1 GB of space potentially able to be left spare. This will be tuned to allow enough run-off space for any data arriving after BUSY assertion to be safely transferred to the host without overruns. Once the difference in pointers grows to such that occupancy is below the low watermark, BUSY will be de-asserted. The different high and low watermarks will introduce a form of hysteresis, meaning that there is minimal risk of oscillation of BUSY assertion/de-assertion.

#### 3.4.2.2 BUSY PCIE FIFO SATURATION

The 128 kB PCIe FIFO is large enough to handle temporary PCIe transfer stalls, but will nevertheless fill up if its input bandwidth is saturated long enough when a temporary stall occurs. It is therefore safest to implement a configurable low and high watermark for the FIFO, where upward crossings of the high watermark cause a BUSY condition. After the assertion of the BUSY the data flow into the FPGA should halt fast enough to prevent FIFO overflow. Once again, the different high and low watermarks will introduce a form of hysteresis, meaning that there is minimal risk of oscillation of BUSY assertion/de-assertion.

For the GBT mode case, given no XOFF is possible, the generation of BUSY must be finely tuned to avoid FIFO overruns. Such a study has yet to be completed for each use case, but we present some indicative scenarios below. In all cases, downward crossing of the low watermark will cause removal of the BUSY condition.



**Figure 3.2:** Bandwidths of internal data paths for FULL mode firmware..

In the FULL mode case it should not be necessary to rely on BUSY as it is expected that XOFF should result in a fast halt of the data flow (e.g. if there is still 4 kByte buffer space in a 16 kB FIFO and the data arrives at 960 MB/s, the data flow should stop within about 4  $\mu$ s). Figure 3.2 presents an overview of the data paths and associated bandwidths for one Wupper engine in this case.

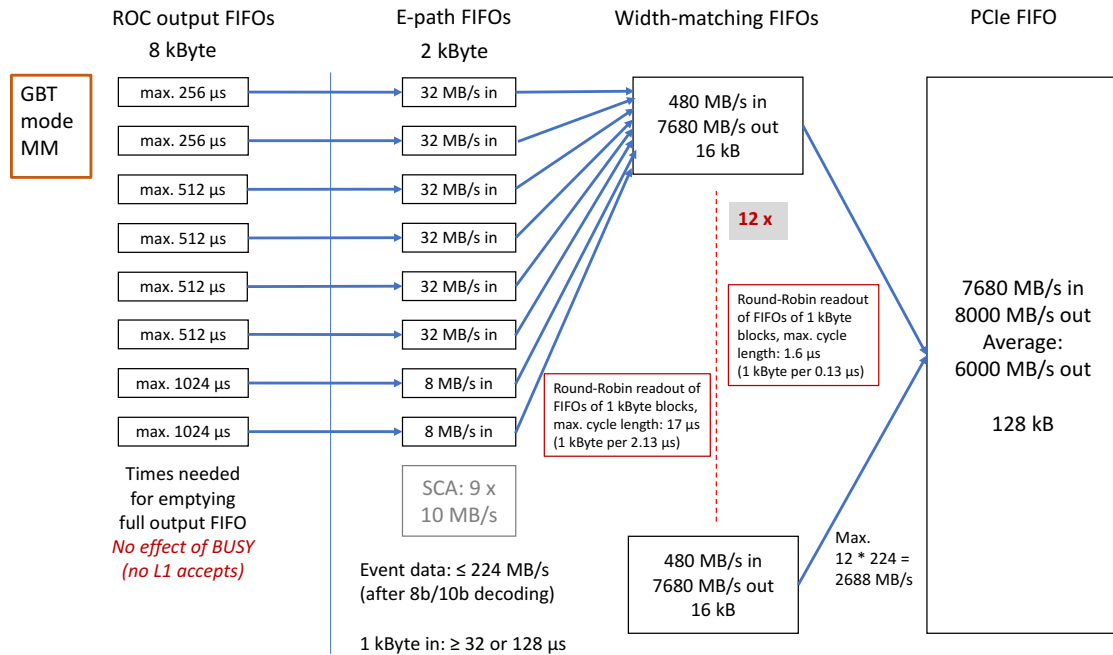
#### 3.4.2.2.1 NSW MICROMEGAS

For the case of NSW MicroMegas detectors, the input bandwidth of the PCIe FIFO at maximum is one third of the available bandwidth (see Figure 3.3 for an overview). After a temporary PCIe transfer stall the FIFO should therefore be emptied considerably faster than it fills. Furthermore, the NSW Readout Controllers (ROCs) have 8 kB output buffers, in which data associated with L1As is stored. Should FELIX assert BUSY due to PCIe FIFO saturation these buffers can be holding at most 64 kByte of extra data to transfer to FELIX if full. The E-path FIFOs and the width-matching FIFOs are emptied fast in comparison with the the transfer of incoming data, so at best there is only 32 kByte of storage available. Therefore, generation of a BUSY condition on the basis of the filling of the PCIe FIFO may in principle cause data loss, although this is unlikely in view of the low predicted E-link utilisation (< 30%).

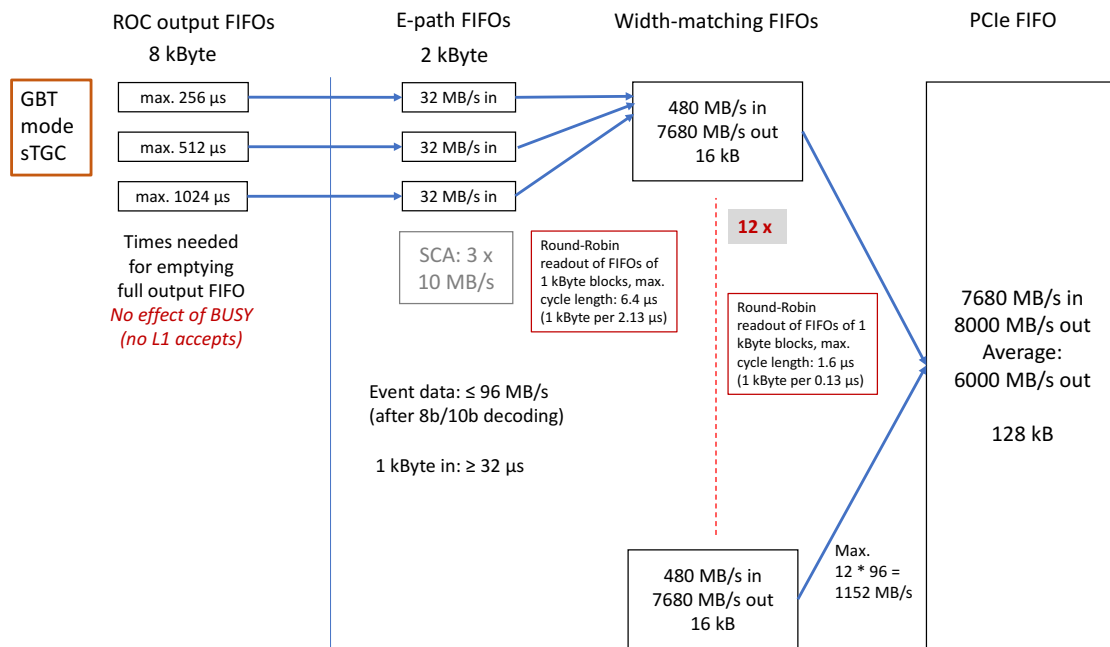
Given the potential amount of data to be stored it is not possible to simply set the high watermark of the PCIe FIFO low enough to accommodate the extra data. One mitigation option which could be considered would be to enlarge the width-matching FIFOs to 64 kByte. Thus far it has been concluded that this is unnecessary if a BUSY condition generation based on the host memory saturation is implemented as specified above.

#### 3.4.2.2.2 NSW sTGC

For the NSW sTGC detectors the problem of insufficient buffer space in the FELIX firmware if the ROC output buffers are full and a BUSY is generated also may occur. In this case it would be possible to configure the high watermark of the PCIe FIFO to generate BUSY early enough to leave sufficient space in the FIFO to safely store all of the data from the ROCs. An overview of the data paths in this case can be seen in Figure 3.4.



**Figure 3.3:** Bandwidths of internal data paths for GBT mode firmware configured for NSW MicroMegas detectors..



**Figure 3.4:** Bandwidths of internal data paths for GBT mode firmware configured for NSW sTGC detectors..

### 3.4.3 CONTROL AND MONITORING OF THE GENERATION OF BUSY CONDITIONS AND THE BUSY SIGNAL

As has been discussed, FELIX supports multiple sources of BUSY condition assertion. These sources are the same in both the FULL and GBT mode cases. In the following section common control and monitoring aspects will be discussed, followed by dedicated discussion of specific differences for each mode.

Alongside BUSY-ON requests received from front-end sources via the input links, there are two additional internal sources of BUSY per Wupper engine: upward crossing of the high watermark of the PCIe FIFO; and saturation of the cyclic buffer in the main memory of the host PC, as indicated by the difference between the read and write pointers. The two Wupper-based BUSY sources need additional configuration functionality. The low watermark and the high watermark of the PCIe FIFO will be configurable in steps of 1 kB, and similarly a low watermark as well as a high watermark for the amount of free buffer memory will be configurable (in steps of 1 MB up to a maximum of 4 GB). A bit in the configuration registers will indicate whether a BUSY condition is associated with one of the two Wupper-based sources. The different BUSY conditions will also be resettable, which per Wupper engine means 1 bit per condition is needed.

It should be possible to assert or de-assert BUSY by writing to a dedicated bit in the configuration registers. In principle only one bit is needed, but for symmetry reasons there will be one bit per Wupper engine for this purpose. To test the generation and removal of the BUSY condition associated with the PCIe FIFO upon upward crossing of the high watermark, it should also be possible to halt or restart the Wupper engine. This can be achieved by testing with "single-shot" DMA transfers.

It should be possible to assert and de-assert the BUSY signal under software control, both taking into account existing BUSY conditions and ignoring them. To achieve this, two bits are needed in the configuration registers. For symmetry reasons it is proposed to implement these for both Wupper engines. With one bit a BUSY condition can be raised or removed. With the other bit the BUSY output is forced to de-assert the BUSY signal irrespective of any existing BUSY condition<sup>6</sup>. Once the BUSY control bit is de-asserted by software the BUSY state will return to that which is dictated by the standard BUSY sources.

An overview of all common bit fields needed in the configuration registers (across all firmware flavours) for one Wupper engine is presented in Table 3.4.

#### 3.4.3.1 FULL MODE SPECIFIC MONITORING AND CONTROL

Were BUSY used in FULL mode (rather than XOFF), the 24 input links could each cause a BUSY condition. A 12-bit field in the configuration registers for each Wupper engine will indicate whether a BUSY condition is associated with each link. Individual BUSY conditions will be resettable, for which a 12-bit field is needed. An overview of all extra bit fields needed in the configuration registers for one Wupper engine in FULL mode is presented in Table 3.5.

#### 3.4.3.2 GBT MODE SPECIFIC MONITORING AND CONTROL

For GBT mode a configurable number of E-links, up to 57 per GBT link (for a 2-bit only E-link configuration without forward error correction), may receive BUSY-ON and BUSY-OFF requests. For the standard 24 GBT link configuration, as used for NSW detectors, there are therefore up to 24 x 57 possible sources of BUSY, as specified in Table 3.3. To avoid resource availability problems it is proposed to monitor the status of the BUSY conditions on a per GBT link basis in the same way as for FULL mode firmware. In addition, for each GBT link a 57-bit field is provided in which each bit is associated with an E-link. A bit is set if a BUSY condition has been generated on that E-link. This should make it simple to determine which E-link(s) caused a BUSY condition.

An overview of all extra bit fields needed in the configuration registers for one Wupper engine in GBT mode is presented in Table 3.6.

<sup>6</sup>Raising a BUSY condition will result in assertion of the BUSY signal if it is not already asserted.



**Table 3.4:** Common bit fields in the configuration registers of a single Wupper engine (i.e. two per FELIX card) for control and monitoring of the generation of BUSY conditions for both GBT and FULL mode firmware. Each firmware flavour adds extra fields on top of this common base, as shown in Tables 3.5 and 3.6..

Type of contents	Number of bits	Type of access
Enable BUSY	1 bit	Write / Read
PCIe FIFO low watermark	11 bits	Write / Read
PCIe FIFO high watermark	11 bits	Write / Read
PCIe FIFO filling level higher than low watermark	1 bit	Read
PCIe FIFO filling level higher than high watermark	1 bit	Read
PCIe FIFO high watermark has been crossed upward (latched)	1 bit	Read / Write (for reset)
Reset PCIe FIFO BUSY condition	1 bit	Write
Free memory low watermark (steps of 1 MB)	12 bits	Write / Read
Free memory high watermark (steps of 1 MB)	12 bits	Write / Read
Free memory filling level higher than low watermark	1 bit	Read
Free memory filling level higher than high watermark	1 bit	Read
Free memory high watermark has been crossed upward (latched)	1 bit	Read / Write (for reset)
Raise / remove BUSY condition	1 bit	Write / Read
Force BUSY de-assertion	1 bit	Write / Read
Enable BUSY from free memory high watermark	1 bit	Write / Read
Enable BUSY from PCIe FIFO high watermark	1 bit	Write / Read
Enable BUSY from E-link (BUSY-ON from FE)	12 x 57 bits	Write / Read

**Table 3.5:** Extra bit fields in the configuration registers of a single Wupper engine (i.e. two per FELIX card) for control and monitoring of the generation of BUSY conditions for FULL mode firmware..

Type of contents	Number of bits	Type of access
Input link BUSY condition status (latched)	12 bits	Read / Write (for reset)

**Table 3.6:** Extra bit fields in the configuration registers of a single Wupper engine (i.e. two per FELIX card) for control and monitoring of the generation of BUSY conditions for GBT mode firmware..

Type of contents	Number of bits	Type of access
Input link BUSY condition state	12 bits	Read / Write
Input link BUSY condition generated	12 bits	Read / Write (for reset)
BUSY condition generated, one bit per E-link	12 x 57 bits	Read
Reset BUSY condition generated (reset of 57 bits)	12 bits	Write
Reset BUSY conditions, one bit per GBT link	12 bits	Write



### 3.4.3.3 INTERRUPT-BASED BUSY REPORTING

In order to notify any client application of a change to the busy state, the FELIX firmware will send an interrupt on a designated line to indicate any assertion or de-assertion of BUSY, whatever the underlying cause. Full details are given in the Wupper engine documentation [wupper].

### 3.4.3.4 VIRTUAL E-LINKS FOR BUSY MONITORING

In order to monitor the fraction of time that a BUSY condition exists is it is proposed (but not concluded) that the use of “virtual E-links” introduced in Section 3.3.2 should be extended to also convey messages indicating the start or end of a BUSY condition. It is also proposed to add a “virtual E-link” per Wupper engine for messages indicating the start or end of a BUSY condition due to the high or low watermark of the PCIe FIFO or host memory ring buffer having been crossed. Each message inserted in these “virtual E-links” should have a header indicating the type of condition. A proposal for the message format is specified in Table 3.7.

**Table 3.7:** “Virtual E-link” message format..

Field	Number of bits
Type of message (example lists in this Section)	4 bits
BCID	12 bits
Last extended L1ID	32 bits
Time stamp derived extracted from BCR (1 MHz clock)	32 bits

The message types transmitted by the “Virtual E-links” will depend on which firmware flavour is in use. However, many elements will be common to both FULL mode and GBT. An proposed list of these is presented below:

- BUSY condition start due to upward crossing of PCIe FIFO high watermark
- BUSY condition stop due to downward crossing of PCIe FIFO low watermark
- BUSY condition start due to upward crossing of free memory high watermark
- BUSY condition stop due to downward crossing of free memory low watermark
- BUSY output asserted
- BUSY output de-asserted

#### 3.4.3.4.1 FULL MODE VIRTUAL E-LINK CONFIGURATION

For FULL mode firmware it will be possible to have one “virtual E-link” per incoming link. The use of a single “virtual E-link” per Wupper engine could also be considered. However, given that information on XON and XOFF signalling has also to be transmitted on a potentially more frequent basis, it is likely that the extra flexibility of having a “virtual E-link” per FULL mode link would be useful. Furthermore, with a “virtual E-link” being associated with a specific physical link there would be no need to include link ids in the messages conveyed. The list of common message types above should be extended in the FULL mode case to include the following extra elements:

- XOFF generated
- XON generated
- BUSY condition start due to receipt of BUSY-ON
- BUSY condition stop due to receipt of BUSY-OFF

#### 3.4.3.4.2 GBT MODE VIRTUAL E-LINK CONFIGURATION

For GBT mode, implementing a “virtual E-link” per E-link will be problematic with respect to resource usage. Furthermore, it is expected that BUSY-ON and BUSY-OFF requests will have a low probability of occurring. Therefore, a single “virtual E-link” per Wupper engine is proposed. Messages are to be transmitted via this link upon BUSY conditions being generated or being removed.

The message format specified in Table 3.7 is suitable for this type of “virtual E-link”. BUSY conditions associated with individual E-links could also be configured to give rise to messages, but this would require that E-link as well as GBT link identification be included in the messages. This is not proposed in view of the low probability for generation of BUSY-ONs and BUSY-OFFs, especially when once considers that implementation may not be straightforward. The list of common message types above should be extended in the GBT mode case to include the following extra elements:

- Start of the OR-ed BUSY condition associated with the receipt of BUSY-ON requests, due to receipt of a BUSY-ON via any E-link
- Removal of the OR-ed BUSY condition associated with the receipt of BUSY-ON requests

## 3.5 PROPAGATION AND MANAGEMENT OF BUSY AND FLOW CONTROL (XOFF) IN FELIX SOFTWARE

This section describes the proposed actions of FELIX software in response to conditions which require either the assertion of a BUSY signal or XOFF. In this context a discussion is also presented on the interactions with the SW ROD and DCS systems.

### 3.5.1 GBT MODE

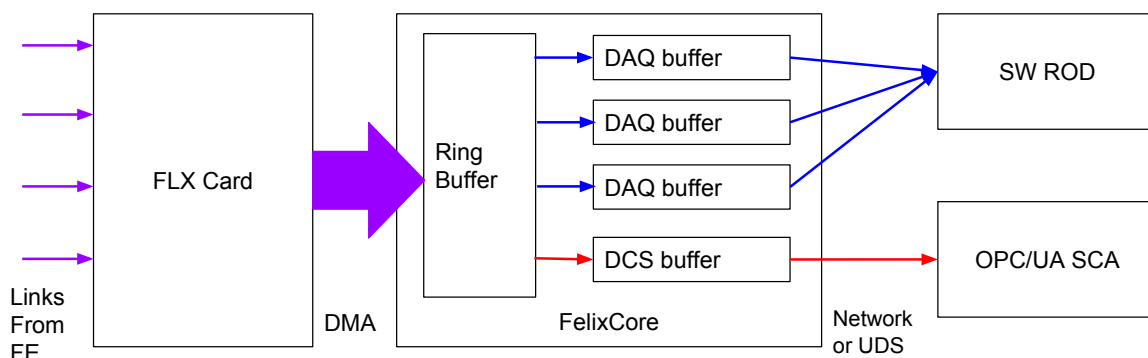
The FelixCore application will maintain a set of DAQ buffers (configurable in terms of number, size and link assignment) through which data are routed on their way to network end points. An example of the layout is given in Figure 3.5. Each buffer will have a configurable high and low watermark. Should the fill state of any one of these buffers exceed the high watermark, FelixCore will immediately request that the FELIX firmware assert a BUSY signal by writing to the ‘Generate BUSY’ control bit in the register map specified in Table 3.3. Once the fill state of the buffer has reduced to below the low watermark FelixCore will request de-assertion of the BUSY using the same register bit.

### 3.5.2 FULL MODE

As for the GBT case, FelixCore will maintain buffers with configurable high and low watermarks. Should a buffer fill beyond the high watermark, FelixCore will request assertion of XOFF for the links in question by writing to the XOFF/XON control bit for the FULL mode link in question as per Table 3.2. The option will be retained to assert BUSY instead of XOFF in this case should requirements evolve such that this is needed. In this case BUSY would be asserted by writing to the ‘Generate BUSY’ control bit described in Table 3.3.

### 3.5.3 SOFTWARE BUSY AND XOFF MONITORING

The FelixCore buffer high and low watermarks, as well as their fill state, can be read and published by monitoring applications for GBT and FULL mode alike. The exact nature of these applications, and their associated clients outside FELIX, has yet to be fully defined. It is currently not proposed to include this data in the potential “virtual E-link” implementations discussed earlier in this document.



**Figure 3.5:** FelixCore buffer schematic in from-front-end direction. In reality DCS will also have a to-front-end path, but this has no bearing on the BUSY logic..

### 3.5.4 FLOW CONTROL FROM SW ROD TO FELIX

SW ROD applications can send flow control (XOFF/XON) signals to FelixCore via a dedicated data subscription at configurable granularity (e.g. specific E-links/FULL mode links or groups of links). Upon receipt of such a signal, FelixCore will stop sending data to the relevant network end point. As the same time, FelixCore will either immediately request XOFF or BUSY from the FELIX firmware, or allow its buffers to fill and eventually trigger the conditions described earlier in this section. The reason one might want to assert XOFF/BUSY immediately is to avoid oscillations caused by the SW ROD de-asserting XOFF after emptying its buffers and immediately becoming saturated once again as FelixCore rapidly empties its own filled buffers into the SW ROD. The final decision can be made based on test experience facilitated by the configurable watermarks available for each FelixCore buffer <sup>7</sup>.

### 3.5.5 BUSY HANDLING AND DCS

In the case of DCS there is a requirement that data remain flowing uninhibited in the case of BUSY. For this purpose FelixCore will maintain a dedicated buffer for DCS data, through which no DAQ data will be routed (as shown in Figure 3.5). It is then proposed that the FelixCore DAQ buffer high watermarks be configured such that sufficient capacity remains to permit them to absorb all regular DAQ data which may arrive between the assertion of BUSY and the halting of L1 Accepts by the CTP. This means that the DMA ring buffer will not be allowed to fill up, thus reserving space at all times for the transfer of DCS data. FelixCore will continue routing DCS data to subscribed clients irrespective of the DAQ BUSY state.

While DCS data itself should be protected from BUSY, it is possible that DCS data itself could cause the high watermark of the PCIe FIFO within the FELIX firmware to be exceeded, with as consequence the assertion of BUSY for all DAQ data. However, the amount of DCS data flowing into FELIX is very small compared to DAQ. Therefore, if such a condition were to arise it is likely that the DAQ data would already be close to causing a BUSY condition. Therefore, this scenario is not considered to require specific mitigation.

However, should DCS clients stop reading data from FELIX, or do so slowly enough that the dedicated DCS buffer fills up, a mechanism is needed to prevent propagation of backpressure into the FELIX firmware such that DAQ dataflow is affected. As such, it is proposed that the DCS buffer also has configurable high and low watermarks. Should the fill state exceed the high watermark, FELIX will discard any DCS data packets received from that point until the fill state is below the low watermark. It is to be discussed how such a condition

<sup>7</sup> There is an open question on how BUSY conditions look to the P1 expert system, and how the source of BUSY is reported for the purposes of stopless removal/recovery. This could either be in the form of: the ROD BUSY module receiving the BUSY from FELIX and triggering a message to the expert system via an application on its master SBC; or via the SW ROD itself sending a message the given links are BUSY and need to be removed. This is an area of discussion with the C&C group, but is beyond the scope of this document

will be communicated to connected DCS clients. Two possible options are: the insertion of a dedicated error message into the DCS data path; or the creation of a dedicated subscription for the propagation of such messages. A final option would be for FELIX to drop the DCS subscription itself, thus generating an error on the client side and activating client-side recovery mechanisms.

# 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 8.7. 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.

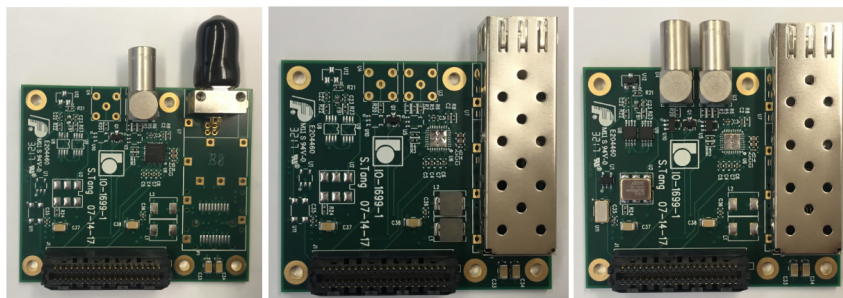
For Phase-II card, the optical module will be the Samtec FireFly module, which can support a link speed of up to 28 Gbps. More details will be shown in the hardware documents. The VCU128 used for firmware demonstration provides 4× QSFP28 module, with in total 16× 28 Gbps links, which can be used to verify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

### 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.



**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	TX: GBT 4.8G, RX: 25.78125 GB Interlaken	TX: 25.78125 GB Interlaken, RX: GBT 4.8G

**Table 4.1:** Protocols supported by FELIX.

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 will be replaced by the TTC-LTI interface.

## 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 capable 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 which is currently under investigation as a possible alternative / addition to PCIe DMA [5].

# 5

## TARGET FPGA

The FELIX Phase I 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 of 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 be the FLX712 card (The Phase I FELIX card), and all the features that can be demonstrated on that platform will be implemented.

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

- PCIe Gen4 or Gen5
- 25 Gb/s Interlaken links
- 100Gb/s RDMA (feasibility study)

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. The next generation of Xilinx FPGA's - the Versal Prime and Versal Premium families will also be investigated. A prototype with the Xilinx Versal Prime VM1802 FPGA has been developed within the FELIX project - known as the BNL181 card or FLX181, and the Xilinx Versal Premium VP1552 is also being investigated as a possible option for the Phase II FELIX card.

Resource	FLX712 / KU115	FLX128 / VU37P	FLX181 / VM1802	VP1552
LUTs	663,360	1,303,680	899,840	1,753,984
FlipFlops	1,326,720	2,607,360	1,799,680	3,508,560
BlockRAM 36kb	2160	2,016	967	2,541
UltraRAM 288kb	-	960	463	1,301
HBM DRAM	-	8 GB	-	-
Transceivers GTH < 16.3 Gbps	64	-	-	-
Transceivers GTY < 32.75 Gbps	-	96	44	68
Transceivers GTM 16 –58 Gbps	-	-	-	20
PCIe interface	Gen3	Gen4	Gen4	Gen5

**Table 5.1:** Available resources on the different development platforms for FELIX Phase II.

		KU115	VU37P	VM1802	VP1552
GBT 24 channel	LUT	80.65%	48.04%	69.60%	35.71%
	FF	77.03%	35.16%	50.94%	26.13%
	BRAM	70.00%	42.91%	89.45%	34.04%
	URAM		30.00%	62.20%	22.14%
FULL 24 channel	LUT	52.59%	30.61%	44.35%	22.75%
	FF	38.40%	22.92%	33.21%	17.03%
	BRAM	40.46%	10.07%	20.99%	7.99%
	URAM		30.00%	62.20%	22.14%
LPGBT 24 channel	LUT	112.51%	57.25%	82.94%	42.55%
	FF	52.39%	26.66%	38.62%	19.81%
	BRAM	68.94%	38.14%	79.52%	30.26%
	URAM		30.00%	62.20%	22.14%
PIXEL 24 channel	LUT	82.40%	41.93%	60.75%	31.17%
	FF	62.04%	31.57%	45.74%	23.46%
	BRAM	61.20%	29.86%	62.25%	23.69%
	URAM		30.00%	62.20%	22.14%
STRIP 24 channel	LUT	67.04%	34.11%	49.42%	25.35%
	FF	49.94%	25.41%	36.81%	18.88%
	BRAM	121.43%	50.10%	104.45%	39.75%
	URAM		70.00%	145.14%	51.65%
INTERLAKEN 8 channel	LUT		6.31%	9.15%	4.69%
	FF		5.44%	7.89%	4.05%
	BRAM		19.39%	40.43%	15.39%
	URAM		0.00%	0.00%	0.00%

**Table 5.2:** Resource utilization for all firmware flavours estimated for the different hardware platforms. The numbers for FM1802 and VP1552, and also for KU115 for the LPGBT, PIXEL and STRIP flavours are estimations based on the build for VU37P.

The numbers shown in Table 5.2 provide a good picture on the requirements for the FELIX Phase II hardware. As a rule of thumb, the LUT and FF utilization should not exceed 70% in order to have a good chance of meeting timing. For Versal Prime devices however this rule of thumb seems to be different, and the tools are unable to properly route the design if the LUT utilization exceeds 50%. The KU115 (FLX712) must only be seen as a development platform for Phase II, and can be used to exercise lower channel counts for all flavours. The FLX181 with the Versal Prime VM1802 FPGA can be seen as a development platform with Gen4 support, and as a learning platform before the Versal Premium devices are available. Both VU37P and VP1552 comfortably fit the resources for all the FELIX flavours with 24 channels, while the VP1552 supports PCIe Gen5 which could double the throughput. With the VP1552 it will be possible to double the channel count to 48 channels for some of the flavours.

#### Requirement 5.1:

A hardware platform with 48 transceiver channels must be available for the LTDB.



# 6

## POWER AND COOLING

For the FLX-712, the power rails are listed in Table 6.2. The maximum current of each power rail and the estimated (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 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.

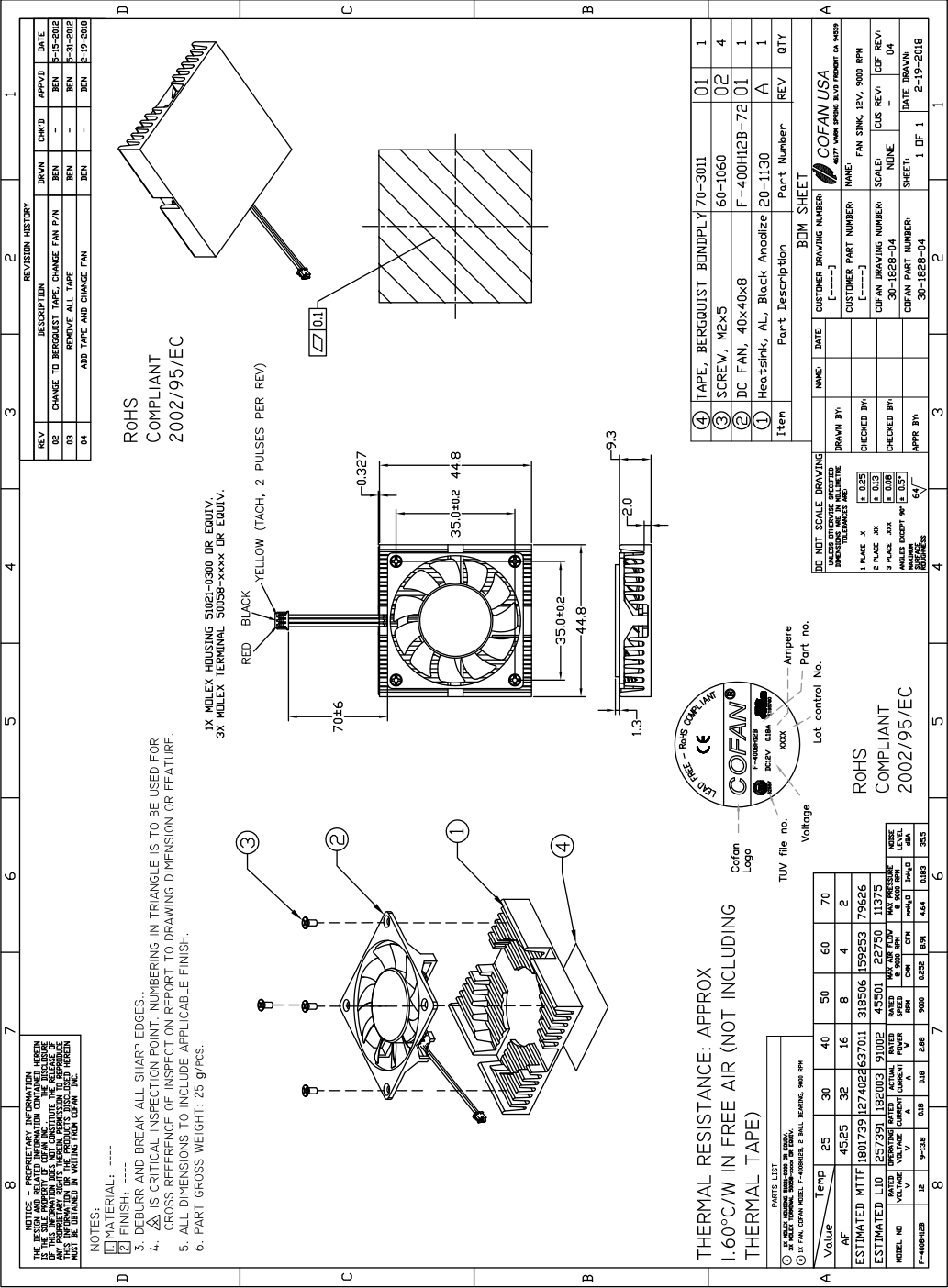


Figure 6.1

# 7

## INPUT/OUTPUT

Name	Direction	Type	Description
BUSY_OUT	out	LVC MOS18	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	LVC MOS18	std_logic, PEX I2C Enable
I2C_SMBUS_CFG_nEN	out	LVC MOS18	std_logic, PEX SMBus CFG Enable
I2C_nRESET_PCIe	out	LVC MOS18	std_logic, PEX active low reset
LOL_ADN	in	LVC MOS18	std_logic, ADN2814 LOL input
LOS_ADN	in	LVC MOS18	std_logic, ADN2814 LOS input
MGMT_PORT_EN	out	LVC MOS18	std_logic, PEX management port enable
NT_PORTSEL	out	LVC MOS18	std_logic_vector(2 downto 0), PEX port select
PCIE_PERSTn1	out	LVC MOS18	std_logic, PEX PERST
PCIE_PERSTn2	out	LVC MOS18	std_logic, PEX PERST
PEX_PERSTn	out	LVC MOS18	std_logic, PEX PERST
PEX_SCL	out	LVC MOS18	std_logic, PEX I2C
PEX_SDA	inout	LVC MOS18	std_logic, PEX I2C
PORT_GOOD	in	LVC MOS18	std_logic_vector(7 downto 0), PEX port good indicator
Perstn1_open	in	LVC MOS18	std_logic, input pin, leave open
Perstn2_open	in	LVC MOS18	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
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	LVC MOS18	std_logic, Global board I2C bus

**Table 7.1:** IO pins (continued...)

Name	Direction	Type	Description
SDA	inout	LVC MOS18	std_logic, Global board I2C bus
SHPC_INT	out	LVC MOS18	std_logic, output, tie to constant '1'
SI5345_A	out	LVC MOS18	std_logic_vector(1 downto 0), Si5345 jitter cleaner configuration
SI5345_INSEL	out	LVC MOS18	std_logic_vector(1 downto 0), Si5345 jitter cleaner configuration
SI5345_OE	out	LVC MOS18	std_logic, Si5345 jitter cleaner configuration
SI5345_RSTN	out	LVC MOS18	std_logic, Si5345 jitter cleaner configuration
SI5345_SEL	out	LVC MOS18	std_logic, Si5345 jitter cleaner configuration
SI5345_nL0L	in	LVC MOS18	std_logic, Si5345 jitter cleaner configuration
STN0_PORTCFG	out	LVC MOS18	std_logic_vector(1 downto 0), Constant output, tie to "0Z"
STN1_PORTCFG	out	LVC MOS18	std_logic_vector(1 downto 0), Constant output, tie to "01"
SmaOut_x3	out	LVC MOS18	std_logic, Optional debug output
SmaOut_x4	out	LVC MOS18	std_logic, Optional debug output
SmaOut_x5	out	LVC MOS18	std_logic, Optional debug output
SmaOut_x6	out	LVC MOS18	std_logic, Optional debug output
TACH	in	LVC MOS18	std_logic, Fan tachometer input
TESTMODE	out	LVC MOS18	std_logic_vector(2 downto 0), Constant output, tie to "000"
UPSTREAM_PORTSEL	out	LVC MOS18	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	LVC MOS18	std_logic, High speed JTAG clock
i2cmux_rst	out	LVC MOS18	std_logic, Reset I2C mux
leds	out	LVC MOS18	std_logic_vector(7 downto 0), Board GPIO leds
flash_SEL	out	LVC MOS18	std_logic, Boot flash pins
flash_a	out	LVC MOS18	std_logic_vector(24 downto 0), Boot flash pins
flash_a_msb	inout	LVC MOS18	std_logic_vector(1 downto 0), Boot flash pins
flash_adv	out	LVC MOS18	std_logic, Boot flash pins
flash_cclk	out	LVC MOS18	std_logic, Boot flash pins
flash_ce	out	LVC MOS18	std_logic, Boot flash pins
flash_d	inout	LVC MOS18	std_logic_vector(15 downto 0), Boot flash pins
flash_re	out	LVC MOS18	std_logic, Boot flash pins
flash_we	out	LVC MOS18	std_logic, Boot flash pins
opto_inhibit	out	LVC MOS18	std_logic_vector(OPTO_TRX-1 downto 0), Minipod / FireFly enable / reset
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

Table 7.1: IO pins (continued...)

Name	Direction	Type	Description
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	LVC MOS18	std_logic, Active-low system reset from PCIe interface
uC_reset_N	out	LVC MOS18	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 8.2.

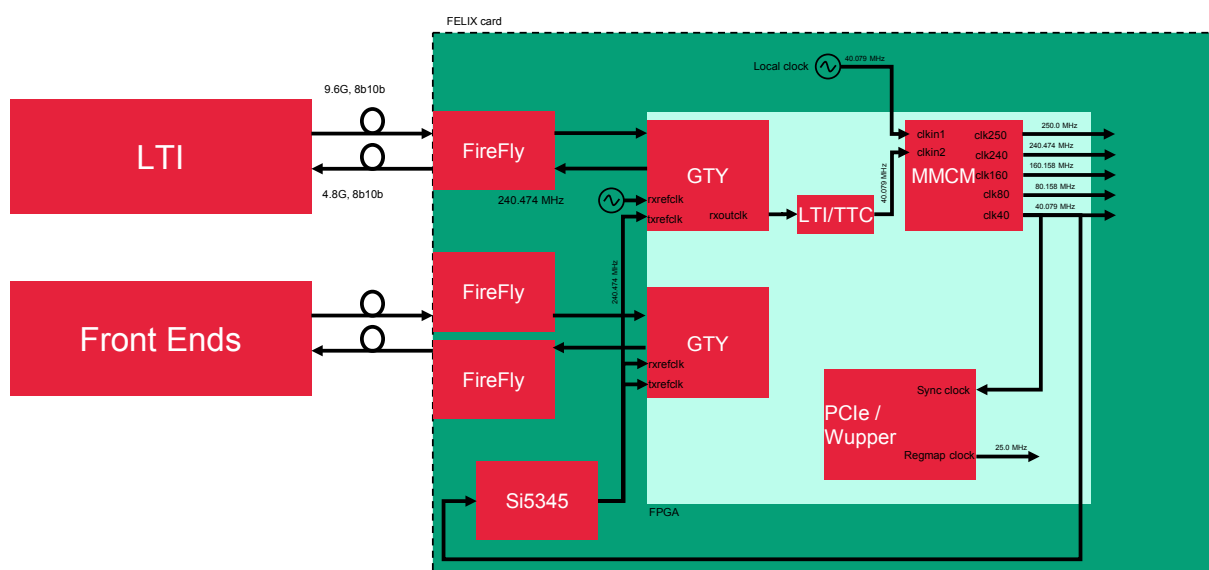
The toplevel design (`felix_top.vhd`) is designed to work for all firmware flavours (GBT, FULL, Strip, Pixel, lpGBT) as well as all hardware platforms (FLX709, FLX712, FLX128, FLX180).

### 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

## 8.3 CLOCKING SCHEME



**Figure 8.1:** Clocking scheme for the FELIX Phase II firmware..

FELIX should be capable of receiving the CERN LHC clock from the LTI and distribute it to the Front End electronics. A large part of the FELIX firmware will also operate on (a multiple of) the LHC clock of 40.079 MHz.

- The LTI transmits TTC information (8b10b encoded) in a fixed data frame of 6 240.474 MHz clock cycles.
- The 240.474 MHz rxoutclk is recovered from the LTI data by the GTY in the FELIX FPGA. The GTY transceiver is using an independent 240.474 MHz clock source as an rxrefclk.
- The LTI/TTC interface recovers the 40.079 MHz and feeds it to the main MMCM.
- The main MMCM can alternatively run from a local 40.079 MHz clock source if the LTI is not available.
- The main MMCM generates 40.079 MHz (clk40) and multiples of that frequency to be used in the FPGA fabric
- clk40 is used to synchronize the Wupper register map to.
- clk40 is fed to the Si5345 jitter cleaner on the board which will create a clean 240.474 MHz clock, used as a reference clock (tx/rx) for all the transceivers, as well as the tx reference clock for the TTC/LTI transceiver.

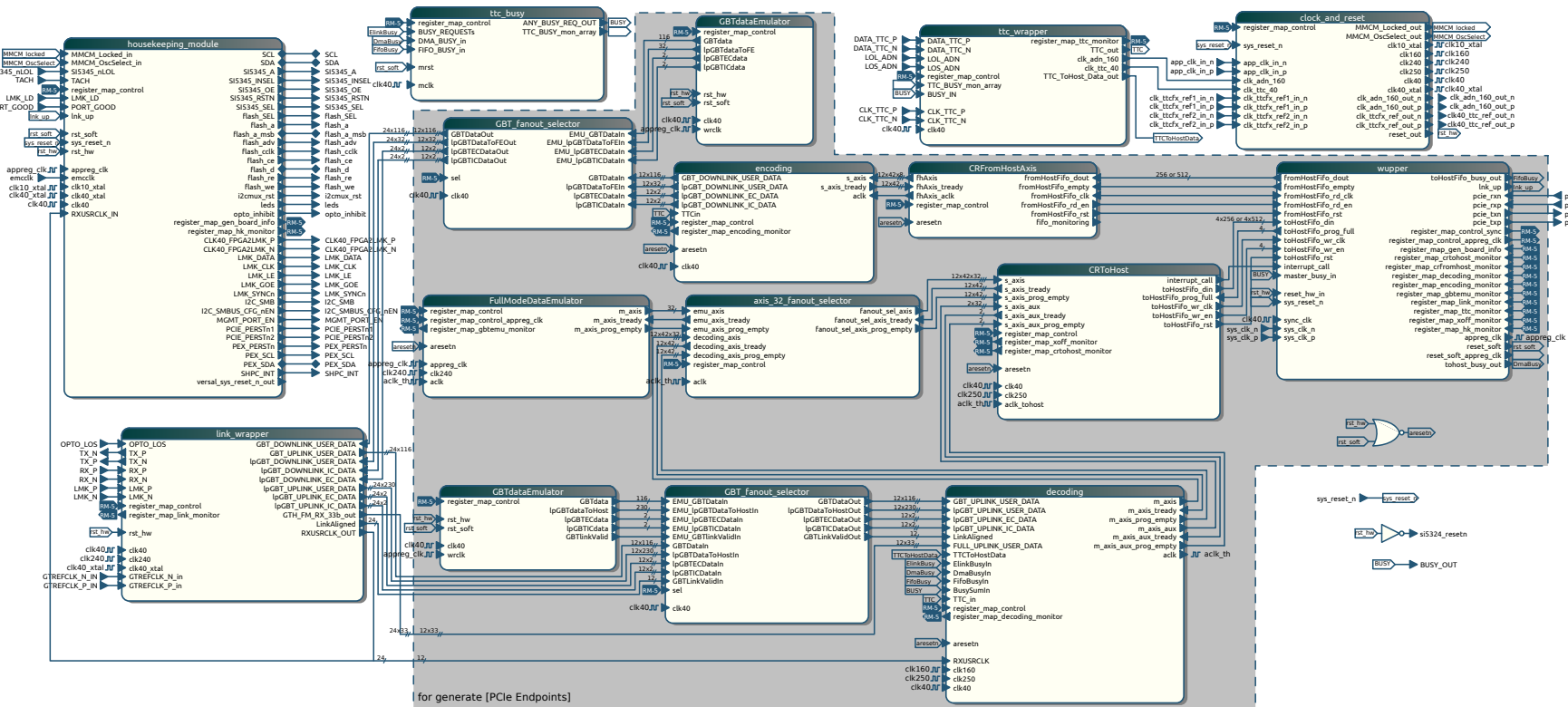


Figure 8.2: The FELIX firmware top level detailed schematic..



## 8.4 DECODING

### 8.4.1 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).

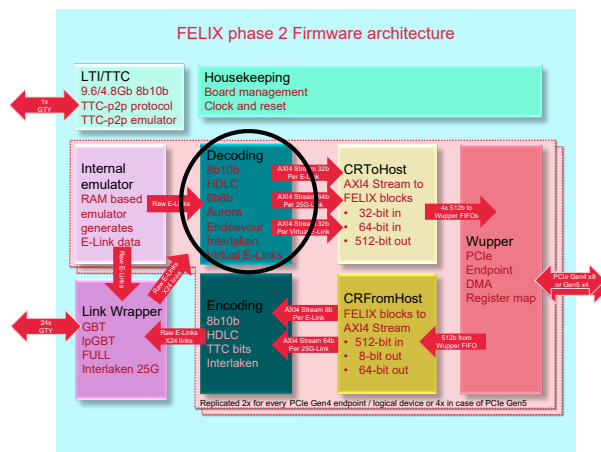


Figure 8.3: The decoding block in the toplevel diagram.

### 8.4.2 INTERFACES

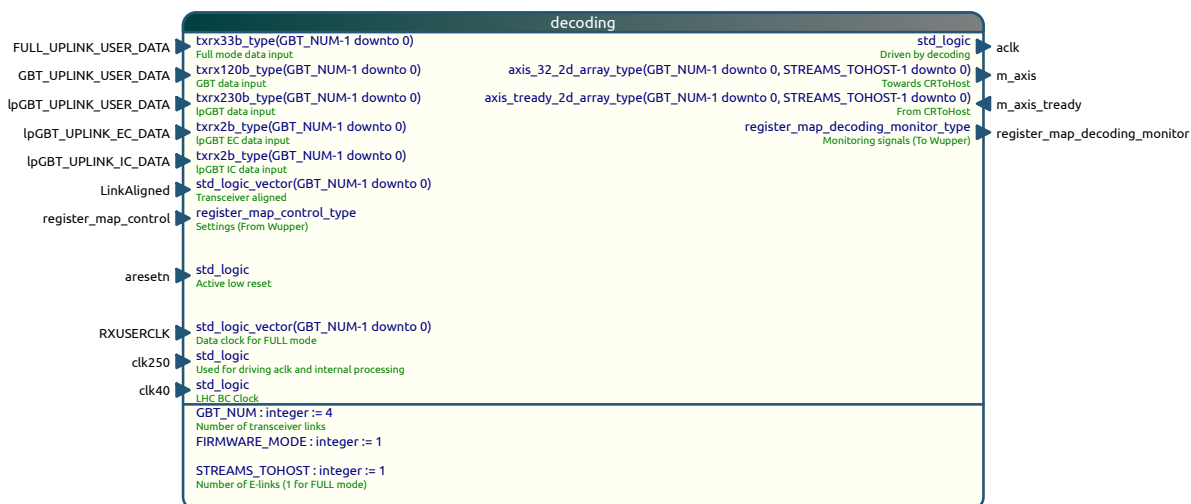


Figure 8.4: The decoding block, instantiating all decoder entities based on FIRMWARE\_MODE [6].

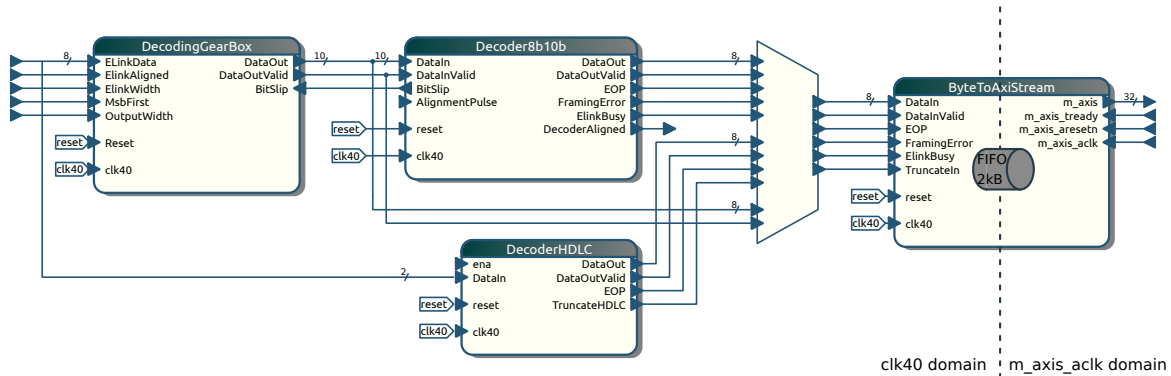
#### 8.4.2.1 OVERVIEW

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 8.4.8). 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.

#### 8.4.2.1.1 GBT MODE, 8B10B, HDLC



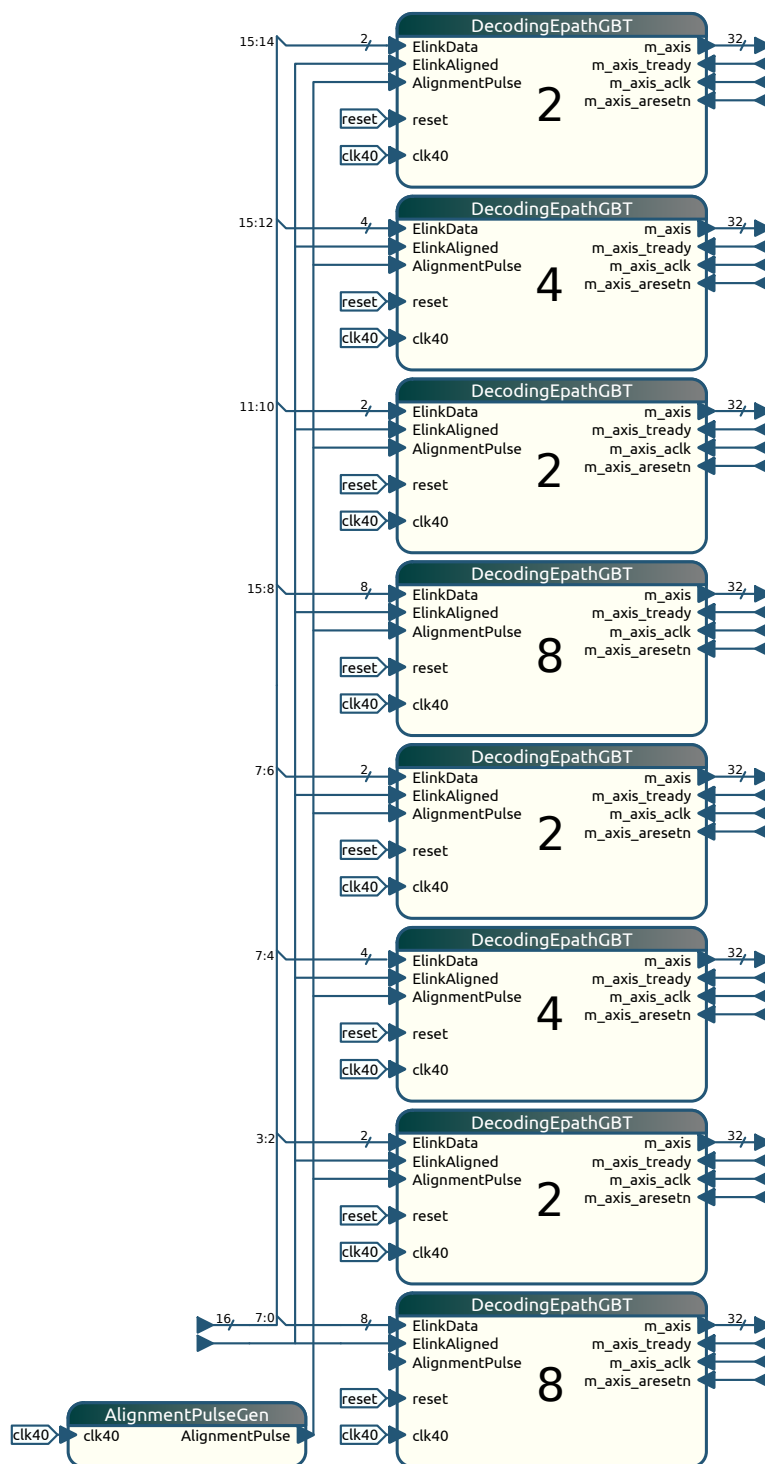
**Figure 8.5:** Block diagram of a single E-Path decoder in GBT mode.

The E-Path as described in Figure 8.5 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 8.6 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 8.6:** Block diagram of an E-Group decoder in GBT mode.

## 8.4.2.1.2 LPGBT MODE, 8B10B

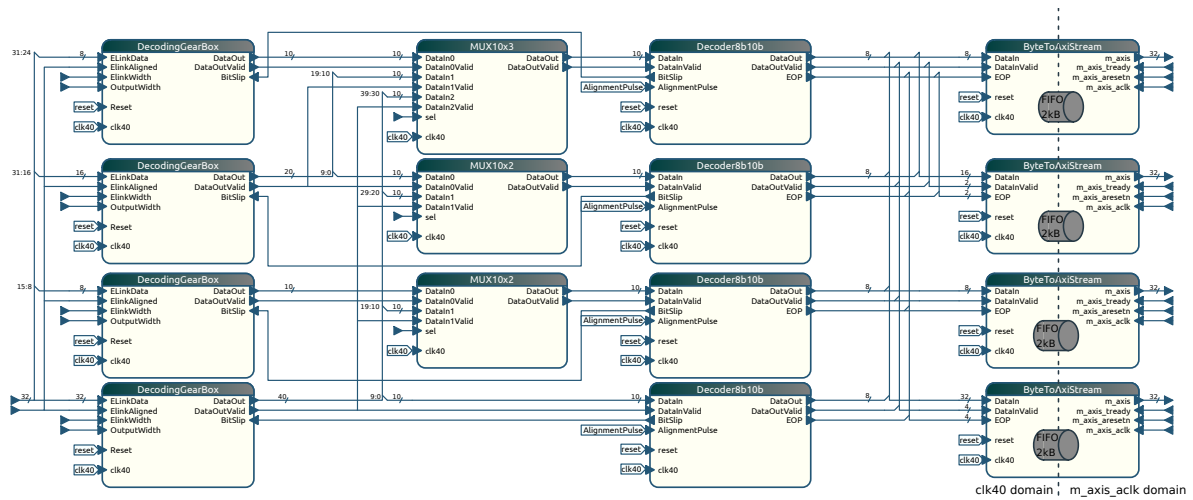


Figure 8.7: Block diagram of an E-Group decoder in lpGBT/8b10b mode.

## 8.4.2.1.3 LPGBT MODE, PIXEL

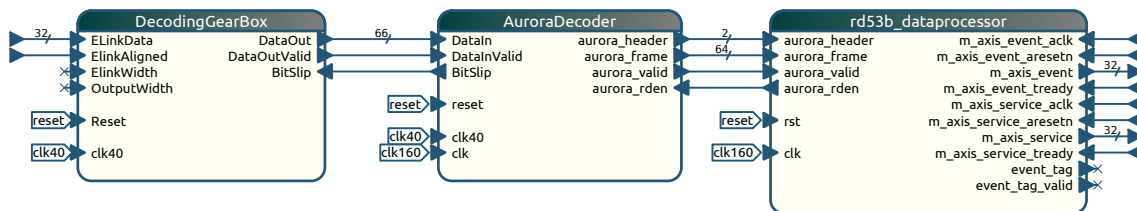


Figure 8.8: Block diagram of a single E-Path decoder in lpGBT / Pixel (RD53b) mode.

## 8.4.2.2 INTERFACE TO CRTOHOST

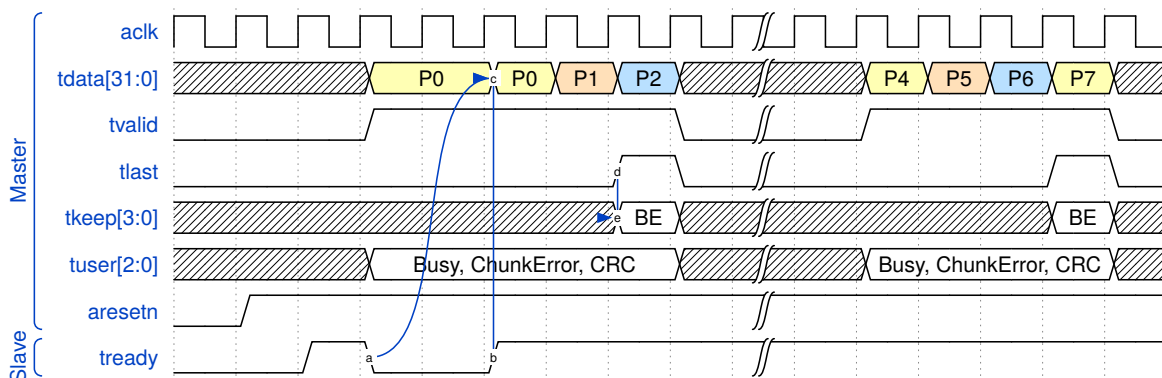


Figure 8.9: Example waveform of a typical AXI stream 32b transfer. [7].

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 CRTToHost
m_axis_tready	in	axis_tready_2d_array_type	From CRTToHost
m_axis_prog_empty	out	axis_tready_2d_array_type	Towards CRTToHost, indicating that 1 block of data is available in the FIFO

**Table 8.1:** Ports to/from CRTToHost..

### 8.4.2.3 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..

### 8.4.2.4 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..

## 8.4.3 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.

## 8.4.4 CONFIGURATION

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

## 8.4.5 STATUS INDICATORS

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

### 8.4.6 LATENCY

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

### 8.4.7 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.

## 8.4.8 DECODING GEARBOX

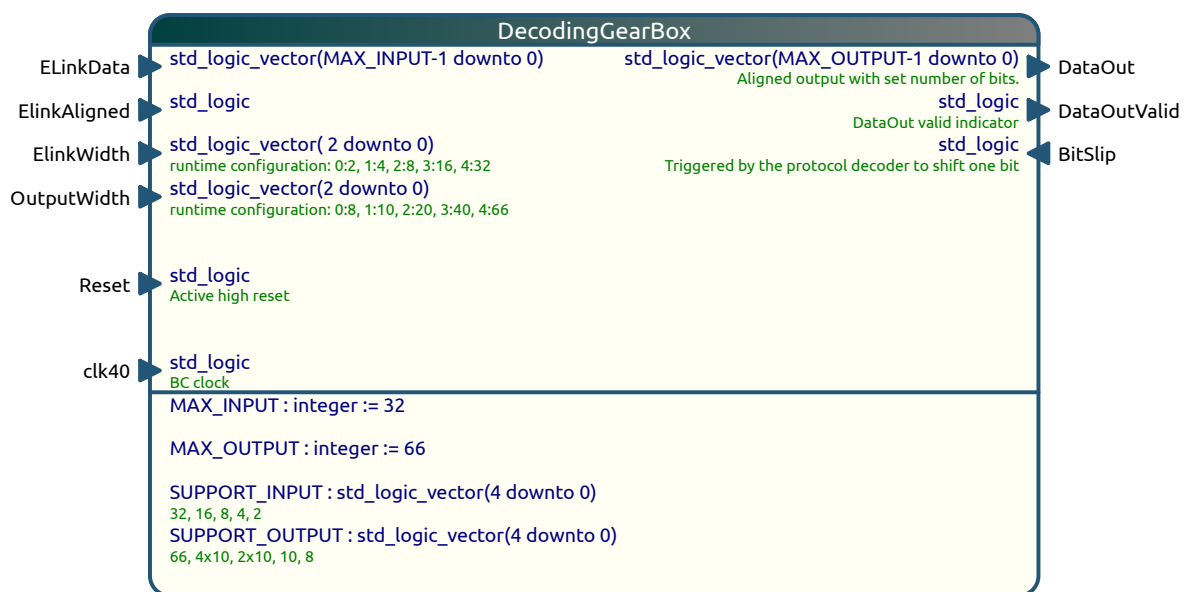
### 8.4.8.1 INTRODUCTION

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.

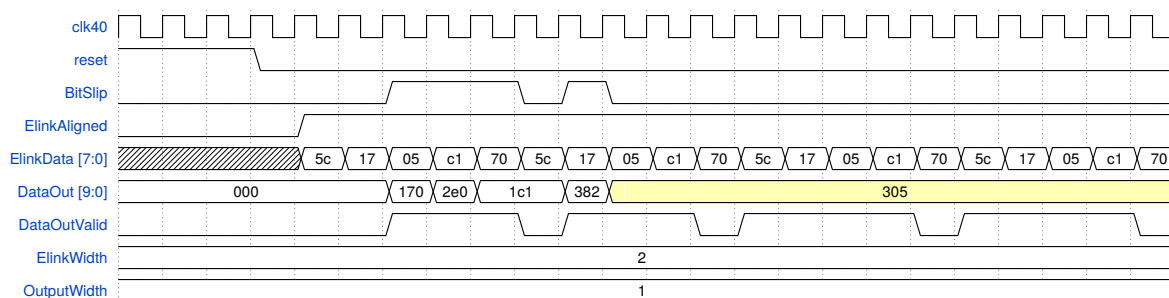
The different protocol decoders require different data widths per BC clock cycle, the Decoding Gearbox will deliver these different data widths by means of shift registers to the different decoder blocks. The available widths on in- and output of the gearbox will be partly configurable at runtime and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

### 8.4.8.2 INTERFACES

#### 8.4.8.2.1 OVERVIEW



**Figure 8.10:** The Decoding GearBox entity.



**Figure 8.11:** DecodingGearBox running with 8 bit input, 10 bit output. The data is constant 0x305 (k28.5+). [7].

#### 8.4.8.2.2 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.

#### 8.4.8.2.3 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.

#### 8.4.8.3 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.

#### 8.4.8.4 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

#### 8.4.8.5 STATUS INDICATORS

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

#### 8.4.8.6 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.

#### 8.4.8.7 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.

#### 8.4.8.8 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)..

### 8.4.9 STRIPDECODER

The decoder for ITk Strips will be the same decoder as described in section [8.4.13](#). Special K-characters are defined for the strips at build time, but the behaviour is the same.

## 8.4.10 ENDEAVOUR DECODER

### 8.4.10.1 INTRODUCTION

Strips firmware has blocks for communicating 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. The Endeavour is a serial “Morse code” protocol, which tolerates  $\pm 50\%$  variation with respect to the nominal 40 MHz AMAC ring-oscillator frequency.

The Endeavour Decoder decodes the data arriving from an AMAC chip. Polarity of the AMAC Decoder serial line can be configured by setting bitfield `INVERT_AMAC_IN` of register `GLOBAL_STRIPS_CONFIG`.

### 8.4.10.2 INTERFACES

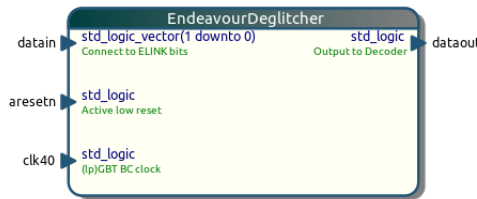


Figure 8.12: The Endeavour deglitcher entity.

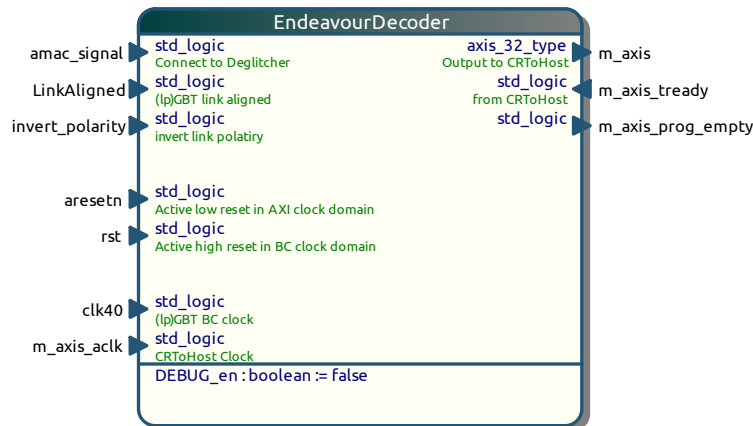


Figure 8.13: The Endeavour decoder entity.

The Endeavour Decoder decodes data from an E-link and send its output towards the ToHost Central Router (CRTToHost) via 32-bit AXI stream interface. In the Strips firmware, the data input is connected to the EC elink of lpGBT frame.

Module ports are listed below. Unless otherwise indicated, the input signals are sampled in `clk40` domain.

- `clk40` - **BC** clock driving the decoder logic
- `m_axis_aclk` - clock for communication with the Central Router
- `amac_signal` - “Morse code” signal from the AMAC chip
- `LinkAligned` (active HIGH) - indicates that the lpGBT link is aligned and decoding may be enabled
- `aresetn` - asynchronous reset for the AXI stream FIFO. Sampled in `m_axis_aclk` domain.
- `rst` - synchronous reset for the main logic

- `m_axis` - output AXI Stream
- `invert_polarity` - inverts polarity of `amac_signal` before decoding
- `m_axis_tready` - indicates that `m_axis` is ready to accept the data. Sampled in `m_axis_aclk` domain.
- `m_axis_prog_empty` - output indicates that the output FIFO is close to being empty. Sampled in `m_axis_aclk` domain.

### 8.4.10.3 FUNCTIONAL DESCRIPTION

Endeavour Decoder de-serializes AMAC data according to these rules:

- serial line is LOW when idle
- HIGH pulses  $6 < n < 22$  BC clock periods long are decoded as ZERO
- HIGH pulses  $29 < n < 124$  BC clock periods long are decoded as ONE
- LOW signal longer than 75 clocks following a pulse is decoded as end-of-word

Decoded words are sent to the host via 32-bit AXI Stream interface, with individual words sent as separate chunks.

### 8.4.10.4 ERROR HANDLING

Chunk error is asserted if the timing of the received waveform does not confirm to the AMAC specification, for example:

- bit pulse is longer than the maximum duration of ONE pulse (bit is truncated)
- bit pulse is shorter than the minimum duration of ZERO pulse (bit is decoded as ZERO)
- bit pulse is longer than the maximum duration of ZERO pulse, but shorter than the minimum duration of ONE pulse (bit is decoded as ONE)
- bit gap is shorter than the minimum duration of a bit gap

If the number of received bits is not divisible by 8, the last byte will have zero bits prepended to the MSB side. There is no indication of whether this situation occurred.

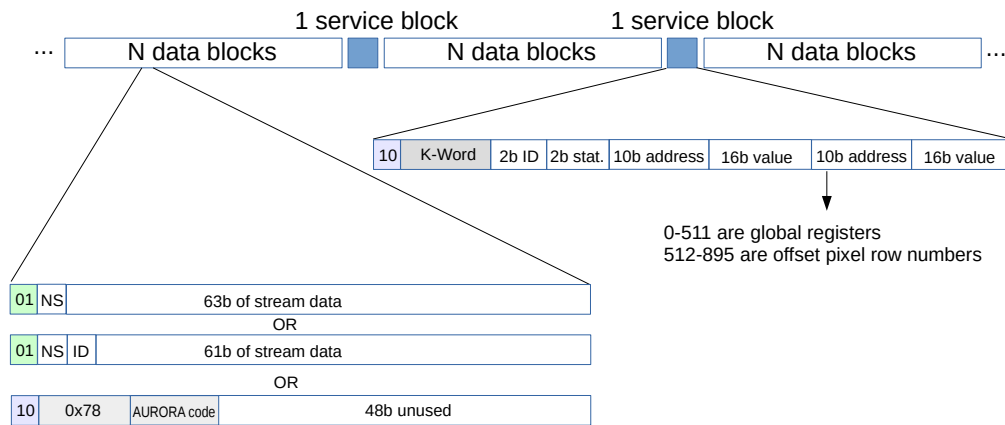
### 8.4.10.5 ESTIMATED RESOURCE USAGE

Resource	lpGBT link	24 GBT links	% (XKCU115)
LUTs	149	3576	<0.1%
Flip-Flops	185	4440	<0.1%
Block RAM	0.5	12	0.5%

**Table 8.9:** Resource consumption of Endeavour Decoder module.

### 8.4.11 AURORA 64B/66B DECODER FOR ITkPix

An ITkPix IC outputs data serially via 1 to 4 lanes at 1.28 Gb/s per lane [8]. The data are encoded with 64b/66b line code as per IEEE 802.3ae-2002 amendment. Data are transferred in 66-bit blocks where the first two bits are sync header and the other 64 bits are payload. Two or more lanes can be aggregated into a transmission channel with the Aurora 64b/66b protocol from Xilinx [9]. The output data can be pixel hit data, service data, or idle blocks (see Fig. 8.14). Frames on all output lanes of an ITkPix are strictly aligned; their data block types are the same. The pixel hit data are transmitted as variable-length streams. Each stream may contain data from multiple ITkPix ICs when on-chip data merging is used. Streams use a sophisticated data format and their processing is discussed in Section 8.4.12. The link level-firmware, i.e. the Aurora 64b/66b decoder, does not process the streams.

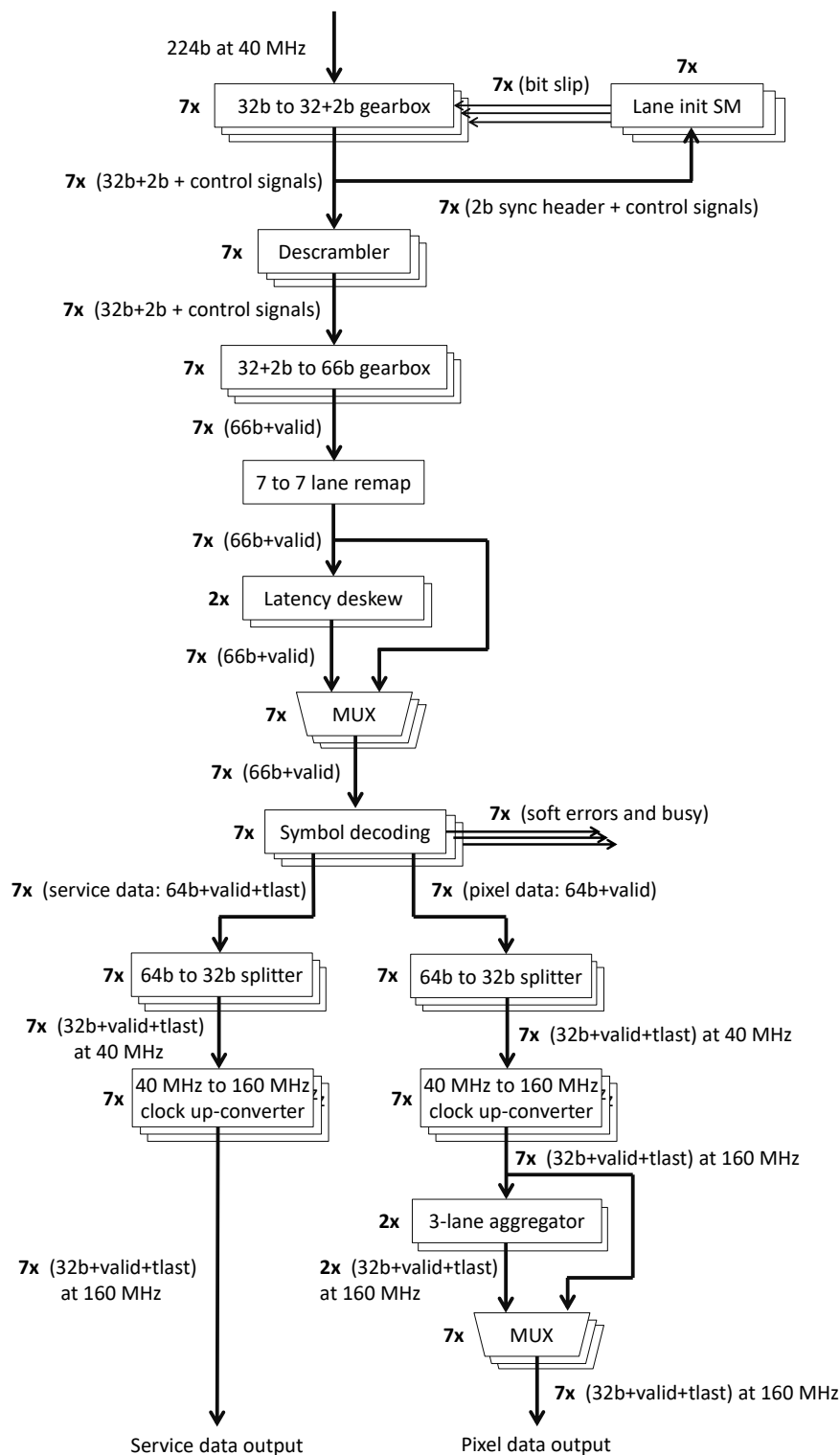


**Figure 8.14:** ITkPix output data consists of data or idle blocks interrupted by periodic service blocks. The content of each block is shown before scrambling. NS stands for New Stream bit and ID is the two least significant bits of chip ID.

An instance of the Aurora 64b/66b decoder receives seven 32-bit words every clock cycle from the IpGBT decoder with one ITkPix lane as a 32-bit word (e-link) as shown in Fig. 8.15. Each 32-bit word is passed to a gearbox that forms 66-bit blocks; the gearbox outputs 32+2 data bits and 2 control bits each clock cycle. Phase alignment of a gearbox is done with a lane initialization state machine (aka block lock) as described in IEEE 802.3ae (2002) Figure 49-12. The state machine uses sync header bits. Only "01" and "10" are valid sync headers. To reach the locked state the state machine needs to receive 6000 valid headers sequentially. Alignment is lost if the state machine detects 16 invalid headers among 5999 or fewer headers. Alignment of lanes can be found in registers `DECODING_LINK_ALIGNED_00` - `DECODING_LINK_ALIGNED_23`.

While the sync headers are transmitted unscrambled, the payload bits are scrambled with a self-synchronizing scrambler polynomial  $G(x) = 1 + x^{39} + x^{58}$  as per IEEE 802.3ae. The decoder descrambles the payload data and performs time deskew of selected lanes using idle blocks with channel bonding bits. The lane ordering and selection is done with `DECODING_LINK_CB(LINK).CBOPT` registers. After completing the time deskew the transmission latencies of these lanes are equal; lanes with the smaller latencies are delayed. The lane with the highest latency is passed through (it is not delayed at all). Time deskew is needed to aggregate the selected lanes (aka channel bond) into a single channel. The channel bonding idles are transmitted on all lanes at the same time at regular intervals. The low-latency deskew is achieved by using shift registers. Selection of lanes for time-deskew can be done via `DECODING_LINK_CB(LINK).CBOPT` registers. A deskew block uses a state machine to determine a delay for each transmission lane. The status of the time deskew state machines is available in registers `AAABBBCCC`.

The link payload can be inspected after descrambling with `YARR_DEBUG_ALLEGROUP_TOHOST(LINK).REF_PACKET(63 downto 32)` and `YARR_DEBUG_ALLEGROUP_TOHOST(LINK).CNT_RX_PACKET` registers. The



**Figure 8.15:** Dataflow in the 64b/66b decoder.

counter is for all the lanes for a given IpGBT link. It increments if there is a data block containing the reference word.

The symbol decoding block separates the idle blocks, pixel hit data, and service data. ItkPix transmits Service data as user K-blocks. Blocks with an incorrect data format (IDLE and user K-blocks) are counted as soft errors. The soft error counters are in AAA\_DECODING\_SOFT\_ERROR register. The errors are detected separately for each lane. The user K-blocks from each lane are output directly to the host (user). The BUSY bits are also extracted from the K-blocks and output as separate signals. The K-blocks can be masked via the DECODING\_MASK64B66BKBLOCK register. Pixel hit data are transmitted as data blocks while separator and separator-7 blocks are not used by ITkPix. Pixel hit data from the selected lanes can be aggregated into channels as needed.

Input to the decoder and the majority of processing is done with the 40 MHz clock. The data is output with a 160 MHz clock that is also used for the lane aggregation blocks (channel bonding). The decoder is reset if the corresponding IpGBT link loses lock. Each lane can be disabled individually with the DECODING\_DISEGROUP register.

The decoder was designed for low-latency data processing. The cumulative latency of the decoder is under 200 ns. About three quarters of the latency is attributed to the gearboxes.

A single decoder takes 7.9k LUTs and 10.2k 28 FFs (Table 8.10), which result in 190k LUTs and 245k FFs for 24 decoders (see Tab. 8.10). For comparison, a VM1802 FPGA offers about 900k LUTs and 27M FFs.

Number of decoders	1	24
LUTs	7896	189.5k
Flip-Flops	10257	246.2k
RAMB36	28	672
RAMB18	7	168

**Table 8.10:** FPGA resource consumption of 64b/66b Aurora decoders for ITkPix. There is one decoder per an IpGBT link.

The FELIX internal data emulator can also be used with a 64b66b bitstream to debug the decoder when ITkPix is not available. The emulator can be enabled with GBT\_TOHOST\_FANOUT.SEL(LINK) register.

The firmware has been extensively tested and validated with simulations, calibration scans and testbeam.

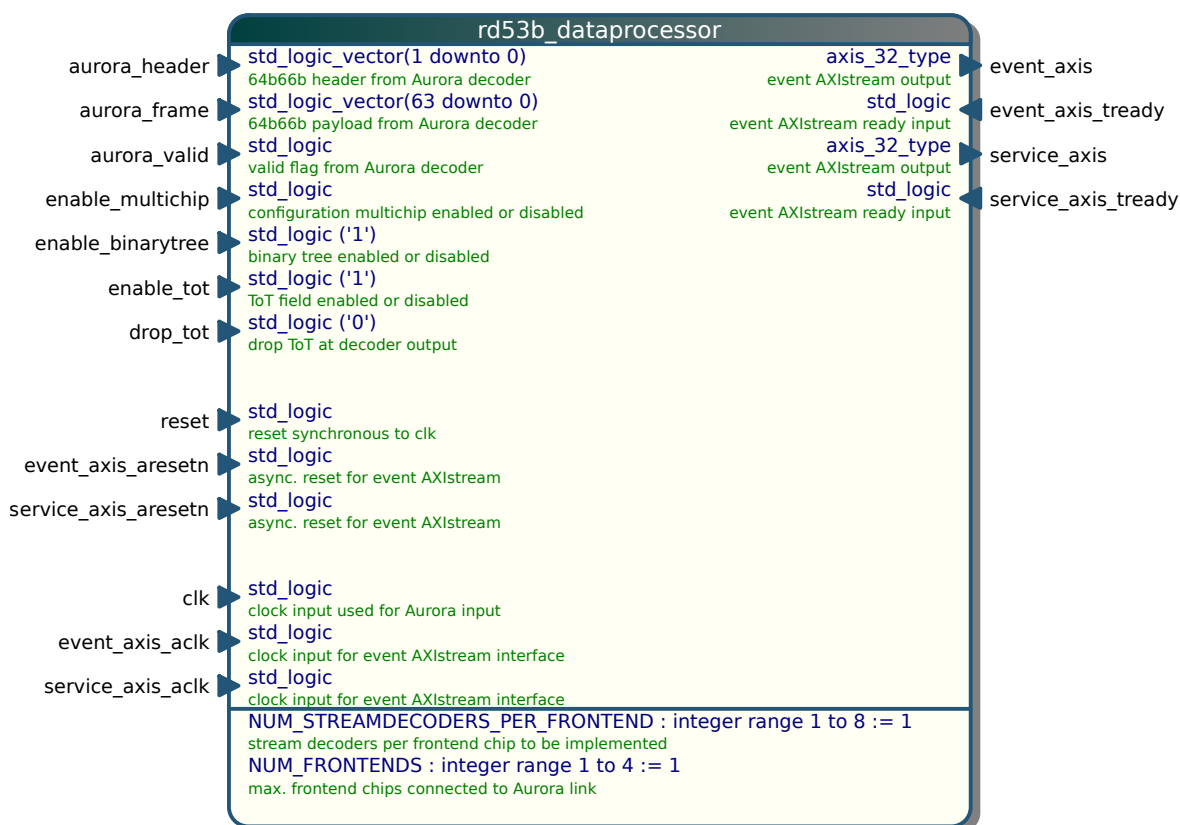


## 8.4.12 RD53B DECODER

### 8.4.12.1 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 8.4.11) 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 8.12).

### 8.4.12.2 INTERFACES



**Figure 8.16:** The RD53b Dataprocessor entity.

#### 8.4.12.2.1 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 8.16 shows the entity of the RD53B Decoder.

#### 8.4.12.2.2 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.

### 8.4.12.2.3 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 8.9.

#### Remark 8.1: *ToDo*

Describe internal data structure of the AXI interface!

### 8.4.12.3 FUNCTIONAL DESCRIPTION

#### 8.4.12.3.1 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.

#### 8.4.12.3.2 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.

#### 8.4.12.3.3 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.

### 8.4.12.4 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  $\lceil \frac{\text{front-end bandwidth}}{\text{stream decoder throughput}} \rceil$
- **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.

#### 8.4.12.5 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.

#### 8.4.12.6 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 8.17 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 8.18 shows the latency distributions for the same number of events per stream as before, but with disabled binary tree.

#### 8.4.12.7 ESTIMATED RESOURCE USAGE

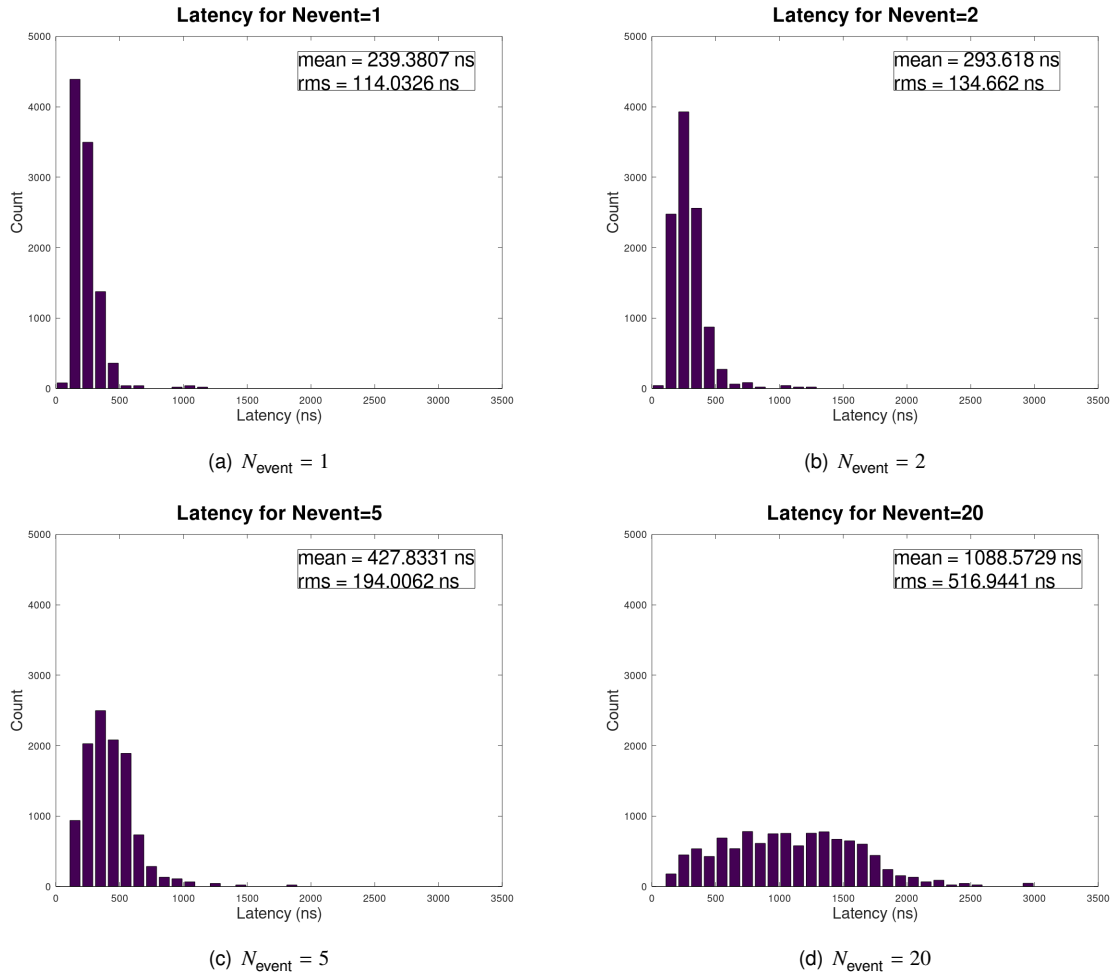
For the resource usage the total number of stream decoders is important:

$$\text{NUM\_STREAMDECODERS\_TOTAL} = \text{NUM\_STREAMDECODERS\_PER\_FRONTEND} \times \text{NUM\_FRONTENDS}$$

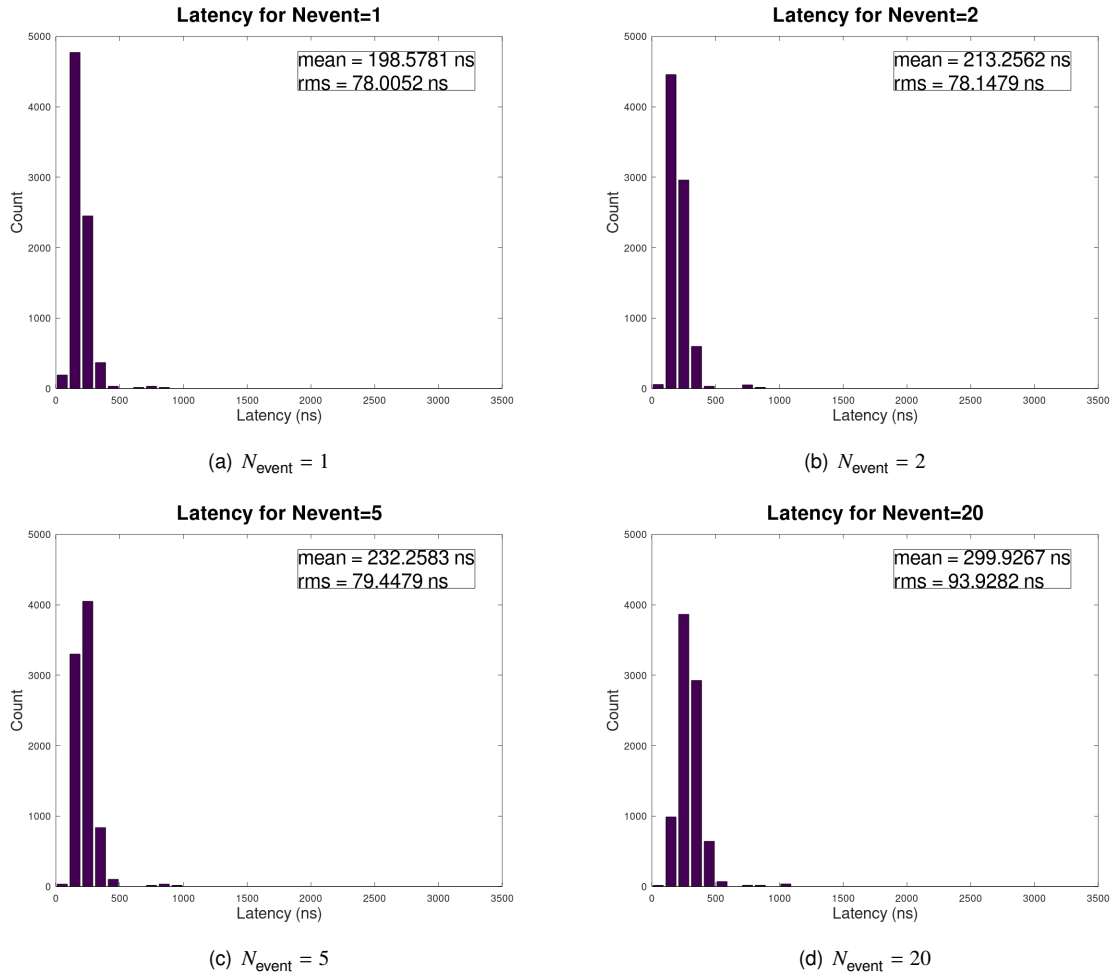
It is now possible to estimate the resource usage for each type of FPGA element:

- LUTs =  $3200 \times \text{NUM\_STREAMDECODERS\_TOTAL}$
- Flipflops =  $720 \times \text{NUM\_STREAMDECODERS\_TOTAL} + 264$
- BRAMs =  $2 \times \text{NUM\_STREAMDECODERS\_TOTAL} + 1$

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



**Figure 8.17:** RD53B Decoder latency for different number of events per stream ( $N_{\text{event}}$ ) with a binary-tree encoded hitmap.



**Figure 8.18:** RD53B Decoder latency for different number of events per stream ( $N_{\text{event}}$ ) with uncompressed hitmap..

### 8.4.13 8B10B E-LINK DECODER

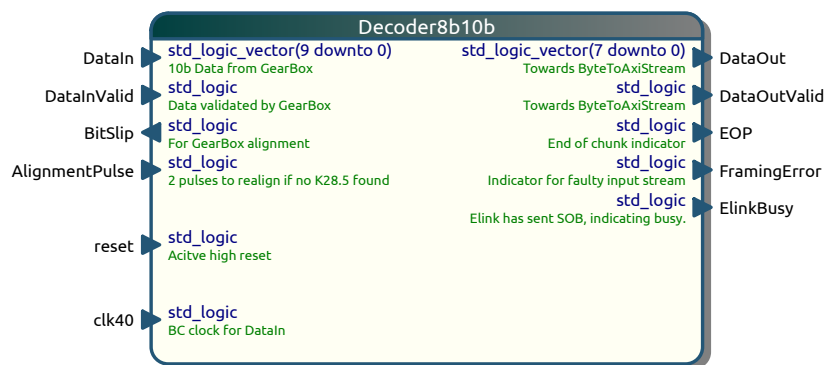
#### 8.4.13.1 INTRODUCTION

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 IpGBT 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 + CharlsK
- Detect Framing Errors
- Detect E-link BUSY assertion
- Deframing: Convert Decoded byte + CharlsK into DataOut, DataOutValid and EOP

#### 8.4.13.2 INTERFACES



**Figure 8.19:** The 8b10b Decoder entity.

##### 8.4.13.2.1 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.

##### 8.4.13.2.2 INTERFACE TO BYTETOAXISTREAM

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

- 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.)

### 8.4.13.3 FUNCTIONAL DESCRIPTION

#### 8.4.13.3.1 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.

#### 8.4.13.3.2 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 8.11:** 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 + CharlsK is well defined in other literature, and the code has been implemented in phase 1 FELIX.

#### 8.4.13.3.3 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 8.11). 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.

#### 8.4.13.3.4 E-LINK BUSY ASSERTION

An FrontEnd may assert BUSY by sending an SOB (Start Of BUSY) character (K28.2, see 8.11) 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 3

#### 8.4.13.3.5 DEFRAMING

DataOut will contain only the payload data (CharlsK = '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.

#### 8.4.13.4 CONFIGURATION

The meaning of the different comma characters in table 8.11 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.

#### 8.4.13.5 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.

#### 8.4.13.6 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.

#### 8.4.13.7 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.

#### 8.4.13.8 ESTIMATED RESOURCE USAGE

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



## 8.4.14 HDLC E-LINK DECODER

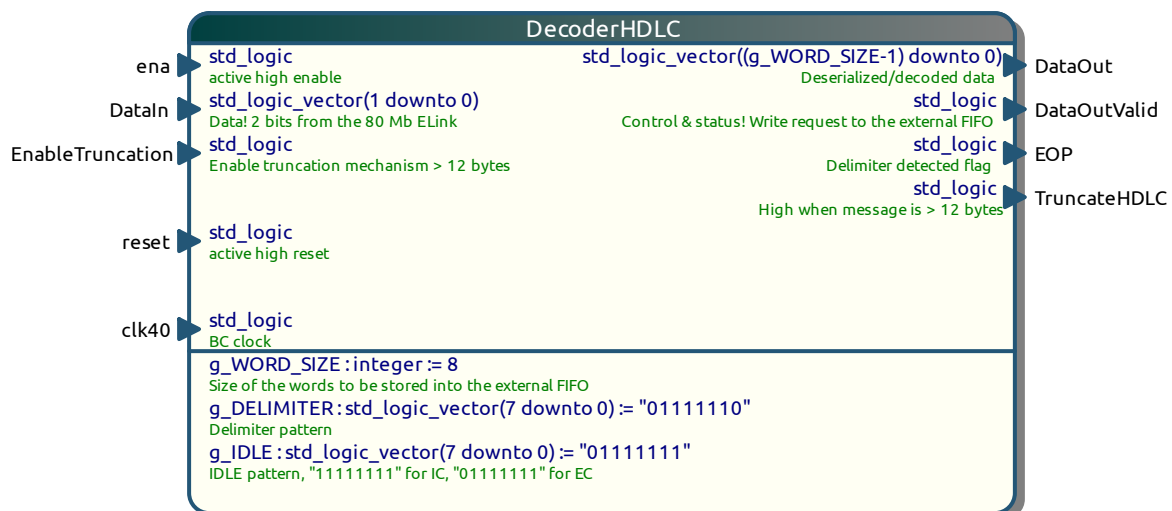
### 8.4.14.1 INTRODUCTION

The HDLC Protocol [10] 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 IpGBT.

The HDLC decoder used in Phase II FELIX was based on the GBT-sc module for FPGA by Julian Mendez [11]. 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 ByteToAxisStream
- A truncation mechanism was added
- The deserializer for IC and EC were merged into a single file.

### 8.4.14.2 INTERFACES



**Figure 8.20:** The HDLC decoder entity.

#### 8.4.14.2.1 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 0x7F for EC.

#### 8.4.14.2.2 ELINK INTERFACE

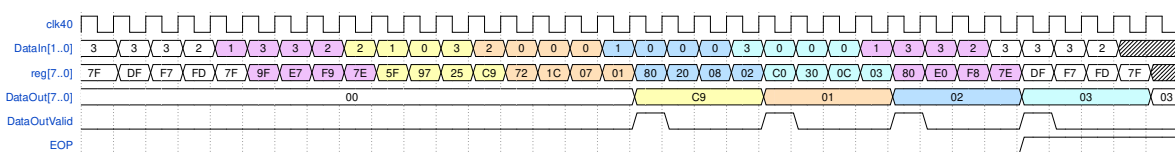
The HDLC decoder does not connect to the DecodingGearbox, since it only connects to 2-bit (80 Mb/s) E-Links. Instead it connects directly to the 2 bits of the E-Link data.

#### 8.4.14.2.3 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.

#### 8.4.14.3 FUNCTIONAL DESCRIPTION



**Figure 8.21:** 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.

#### 8.4.14.4 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.

#### 8.4.14.5 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 CRTToHost.

#### 8.4.14.6 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 8.21). 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 [10]

#### 8.4.14.7 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 CRTToHost.

#### 8.4.14.8 ESTIMATED RESOURCE USAGE

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

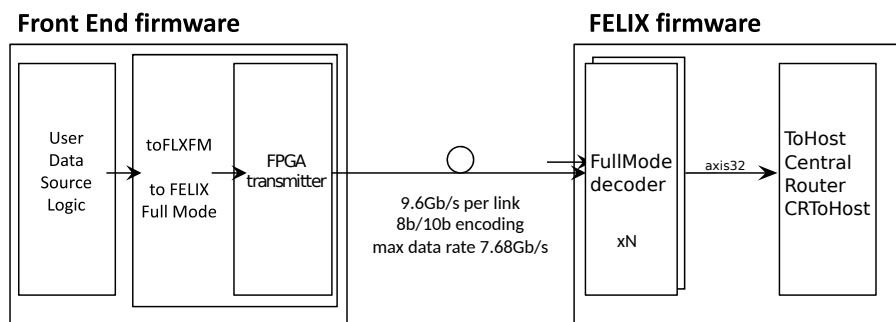
## 8.4.15 FULLMODEDECODER

### 8.4.15.1 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[12]. 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 8.22 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 8.4.15.3.1.

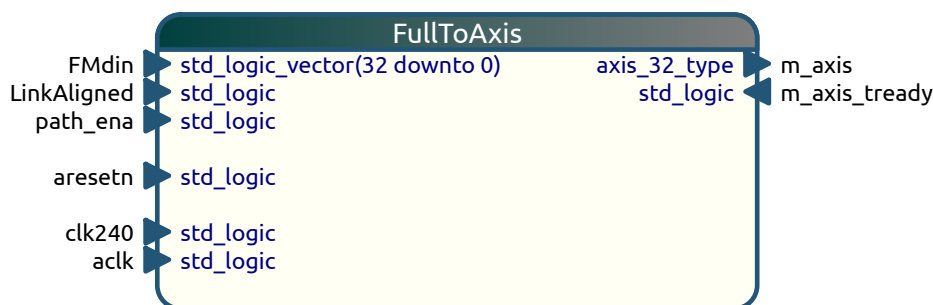


**Figure 8.22:** Block diagram of both the FrontEnd and FELIX ends of a Full mode link in the ToHost direction.

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.

### 8.4.15.2 INTERFACES



**Figure 8.23:** The FULL mode decoder entity.

#### 8.4.15.2.1 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.

#### 8.4.15.2.2 INTERFACE TO CRTOHOST

The interface to the Central Router ToHost (CRToHost) is the same as for other decoders: axi stream 32. In the 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 8.12

**Table 8.12:** 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 the

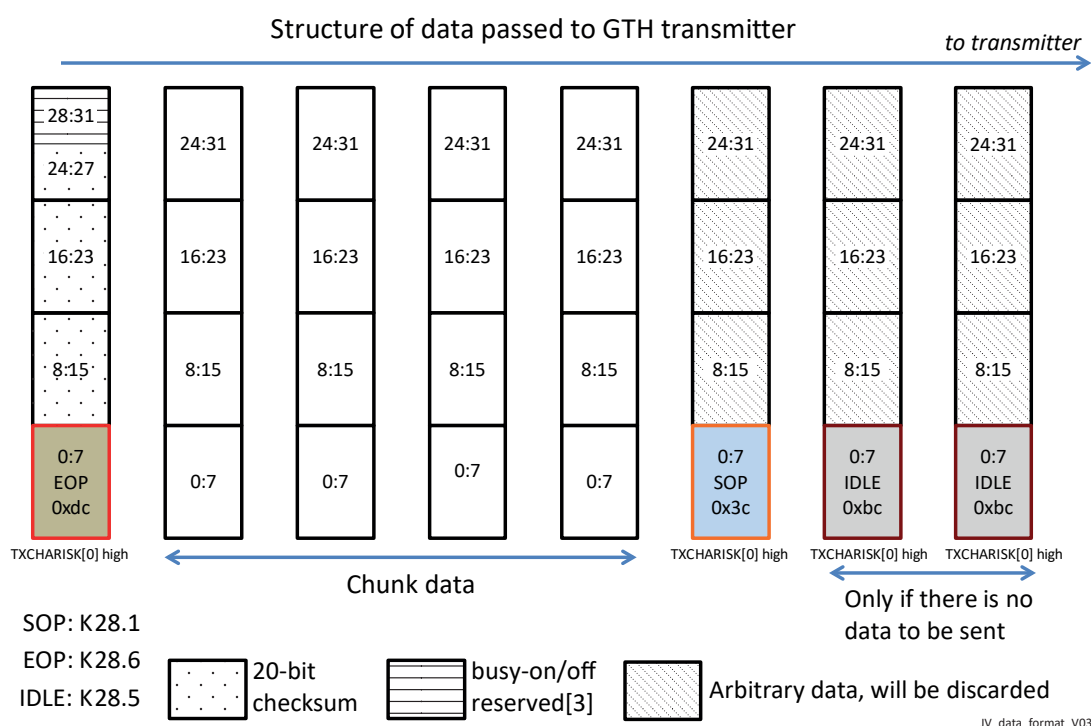
#### 8.4.15.3 FUNCTIONAL DESCRIPTION

The FULL Mode Decoder (FullToAxis) interprets the K-characters as described in section 8.13, and translates the stream of data into the industry standard AXI4 stream bus, which can be handled by the CRToHost entity.

**Table 8.13:** 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 8.24. See Section 8.4.15.3.2 for details on the CRC.



**Figure 8.24:** The format of the data transmitted between the serializer and deserializer of the Full mode wrapper.

#### 8.4.15.3.1 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.

#### 8.4.15.3.2 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 8.60 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 polynomial and initial value have been set to the values below.

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

The polynomial has been chosen based on research by Philip Koopman <https://users.ece.cmu.edu/~koopman/crc/>. With this polynomial 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 worse performance can be found here. [crc20.vhd](#)

#### 8.4.15.4 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.

#### 8.4.15.5 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 3.

#### 8.4.15.6 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 CRTToHost 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.

### 8.4.15.7 ESTIMATED RESOURCE USAGE

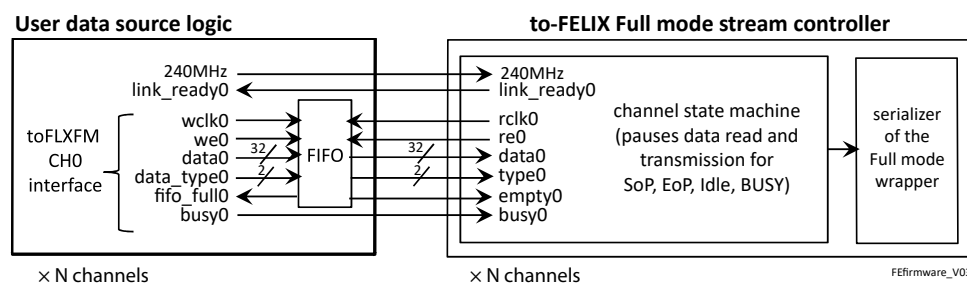
A single FullToAxis entity (including the axi stream FIFO) is reflected in the table below. The numbers are for a single channel.

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

**Table 8.14:** Resource consumption for the FullToAxis entity.

### 8.4.15.8 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 8.25 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 8.4.15.3.



**Figure 8.25:** 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 8.15 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 8.4.15.3 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 8.15. The module will be implemented by:

- 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.



**Table 8.15:** 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
rdclk	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

## 8.4.16 DIRECT MODE E-LINK DECODER

### 8.4.16.1 INTRODUCTION

Direct decoding is implemented by omitting the decoder. This is done by connecting ByteToAxisStream directly to the DecodingGearBox, as shown in figure [8.5](#)

**Remark 8.2:** *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.

## 8.4.17 TTCToHost VIRTUAL E-LINK

### 8.4.17.1 INTRODUCTION

The TTC ToHost Virtual E-Link generates a data stream upon 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.

### 8.4.17.2 INTERFACES

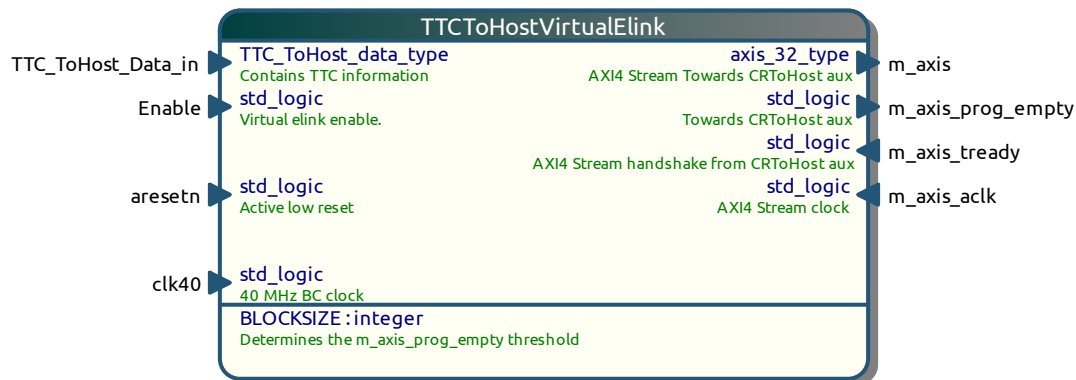


Figure 8.26: The TTC ToHost Virtual E-Link entity.

#### 8.4.17.2.1 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.

#### 8.4.17.2.2 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.

#### 8.4.17.2.3 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 CRTToHost.
- aresetn: Active low reset.
- Enable: To enable the virtual E-Link. Connected to the Wupper register map.

#### 8.4.17.2.4 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 CRTToHost. CRTToHost 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).

#### 8.4.17.3 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.

#### 8.4.17.4 CONFIGURATION

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

#### 8.4.17.5 STATUS INDICATORS

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

#### 8.4.17.6 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.

#### 8.4.17.7 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.

#### 8.4.17.8 ESTIMATED RESOURCE USAGE

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

Block RAM	1	0.05%
-----------	---	-------

**Table 8.16:** TTC ToHost Virtual E-Link Resource utilization.

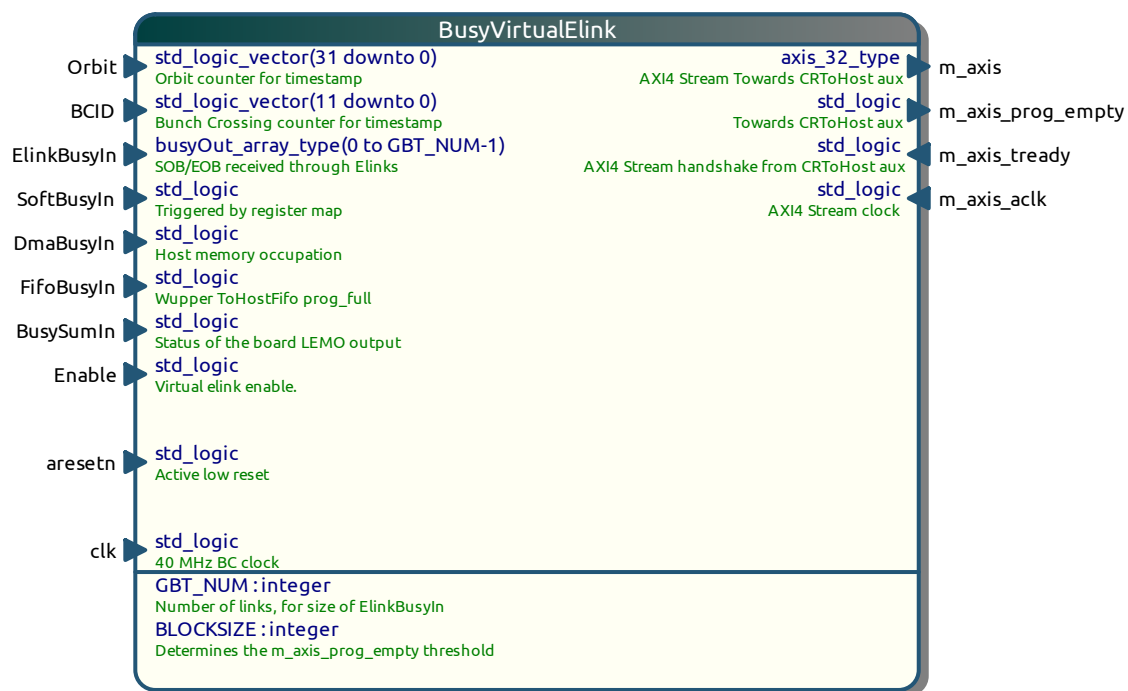
## 8.4.18 BUSY VIRTUAL E-LINK

### 8.4.18.1 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)

### 8.4.18.2 INTERFACES



**Figure 8.27:** The Busy Virtual E-Link entity.

#### 8.4.18.2.1 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.

#### 8.4.18.2.2 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

#### 8.4.18.2.3 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.

#### 8.4.18.2.4 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\_ack: 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.

#### 8.4.18.2.5 INTERFACE TO CENTRAL ROUTER TOHOST

The AXI4 Stream interface consisting of m\_axis, m\_axis\_prog\_empty, m\_axis\_tready and m\_axis\_ack 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).

#### 8.4.18.3 FUNCTIONAL DESCRIPTION

The BUSY Virtual E-Link monitors the status of the 4 sources of busy explained in 8.4.18. 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.

#### 8.4.18.4 CONFIGURATION

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

#### 8.4.18.5 STATUS INDICATORS

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

#### 8.4.18.6 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.

#### 8.4.18.7 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.

#### 8.4.18.8 ESTIMATED RESOURCE USAGE

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

**Table 8.17:** Busy Virtual E-Link Resource utilization.



## 8.4.19 25 GB/S INTERLAKEN

The Interlaken protocol would be an excellent candidate to use for 25 Gb/s data links. It is a simple to use high bandwidth protocol which is also royalty free. An open source FPGA core (Core1990) has already been developed and earlier work has proven that this is compliant with the Interlaken protocol standard.

The core carries provides the following features:

- In band and out of band flow control (simple Xon/Xoff)
- 64b67b line encoding and scrambling
- Performance that scales with lanes
- Error detection implementing both CRC-24 and CRC-32

Core1990 is not included in the Felix firmware repository but is included as a submodule.<sup>8</sup> Felix will only receive Interlaken data and this is why solely the Interlaken receiver will be specified in this document.

### 8.4.19.1 INTERFACES

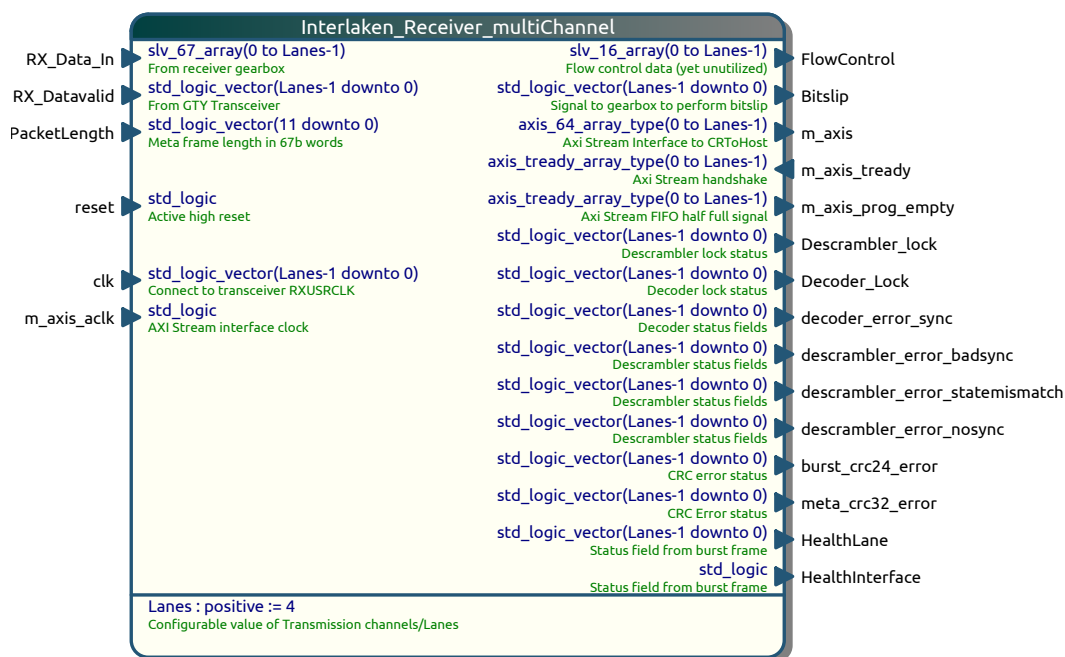


Figure 8.28: The Interlaken receiver entity.

#### 8.4.19.1.1 USER INTERFACE

Interfacing with the Interlaken receiver consists of two parts. The first part is connecting the receiver to an active Interlaken data stream (e.g. a transceiver). This data will be processed by the receiver entity.

The second part consists of received data words that are ready to be processed by the user. The originally transmitted data will be accessible through an interface that's using the AXI-Stream protocol. This mainly consists of three signals: `tdata`, `tvalid` and `tlast`. Data words will arrive at the 64-bit `tdata` bus, while `tvalid` will

<sup>8</sup>[https://gitlab.cern.ch/atlas-tdaq-felix/core1990\\_interlaken](https://gitlab.cern.ch/atlas-tdaq-felix/core1990_interlaken)

indicate that this data is valid and meant to be further processed by the user. Tlast will indicate the end of an AXI-Stream packet. The core will only provide data when the tready handshake signal is set. When this is not the case data will be put on hold. At the receiving side this means stalling the data and potential data loss in case a RX FIFO overflow (this can mitigated by the use of flow control).

Signal	Direction	Width	Description
RX_data_in	In	67	64-bit data word with 3-bit header
RX_data_valid	In	1	RX_data_in is valid
Bitslip	Out	1	Bitslip Interlaken data for alignment (header 64b67b decoding)
m_axis	Out	1	Data output in AXI-Stream format
m_axis_tready	In	1	Handshake to make new data arrive at the output
m_axis_prog_empty	Out	1	AXI-Stream FIFO in the receiver is half full

**Table 8.18:** Interlaken receiver user interface.

Three other signals are also provided in the AXI-Stream interface. This contains a tkeep signal to indicate which bytes of the tdata word are valid. Another signal, tuser, will be used to provide the user status/error indication. The tid signal is used to indicate which Interlaken channel the data is from.

Signal	Width	Description
m_axis.tdata	64	User data word
m_axis.tvalid	1	User data word is valid
m_axis.tlast	1	Last data word of stream / End of packet
m_axis.tkeep	8	Valid bytes
m_axis.tuser	4	Status/error bits
m_axis.tid	8	Channel number

**Table 8.19:** Interlaken receiver AXI-Stream signals.

#### 8.4.19.1.2 CLOCK SIGNALS

The Interlaken receiver requires two clocks. The main clock has to be synchronous to the received data. All Interlaken logic will use this clock. The outputted data towards the user through the AXI-S interface will be synchronous to the m\_axis\_aclk.

Clock signal	Description	Frequency @ 25 Gbps
clk	Synchronous to RX_Data_In, drives RX logic	402.83 MHz
m_axis_aclk	m_axis data read input clock	User defined

**Table 8.20:** Interlaken receiver clock signals.

#### 8.4.19.2 FUNCTIONALITY

The interlaken core uses an AXI-Stream interface to transfer data. This is on a per-lane basis so each AXIS interface will be connected to a lane. The user can configure the amount of lanes required to be implemented and each of these lanes will have it's own AXI-Stream interface.

##### 8.4.19.2.1 BURST FRAMES

User data will be packed in bursts. Such a burst will carry a specific amount of consecutive data words, start with a SOP (Start Of Packet) indication and end with a EOP (End Of Packet) indication. These SOP and EOP

settings will be indicated by transmitting a Burst Control Word with the specific bit set. The minimum and maximum length of these bursts are configurable and can be controlled by the user. However if the payload contains less words than required to achieve the minimum burst length, the burst will contain additional Idle Control words to ensure the minimum burst length is reached. (This minimum has been set to reduce the burden on the transmitter and receiver)

Each Burst and Idle Control word will contain a CRC-24 (poly: 0x328B63) word to ensure data integrity. A Burst Control word containing a EOP will contain a CRC that covers the preceding payload including the EOP itself (So the SOP is not included).

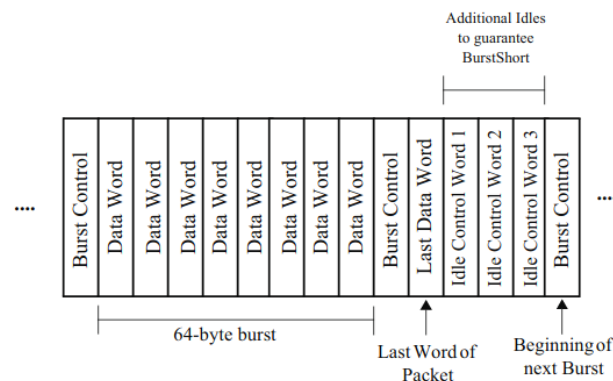


Figure 8.29: Interlaken Burst.

#### 8.4.19.2.2 META FRAMES

The framed burst data (control and data) will be packed in Meta Frames. This is a set of four control words to align lanes (Synchronization), synchronize the scrambler (Scrambler State), clock compensation (Skip) and diagnostic information (Diag). A complete Meta Frame consists of these four control words appended with the payload. The length of the meta frame can be defined by the user.

Each Diagnostic word contains a CRC-32 (poly: 0x1EDC6F41) word that covers the entire meta frame, however the Scrambler State and CRC-32 fields are treated as zeroes. This is done before any encoding/scrambling and covers 64 bits of the words, so without any added headers.

The length of a complete meta frame can be defined by the user but should be configured the same length for both the transmitting and receiving sides.

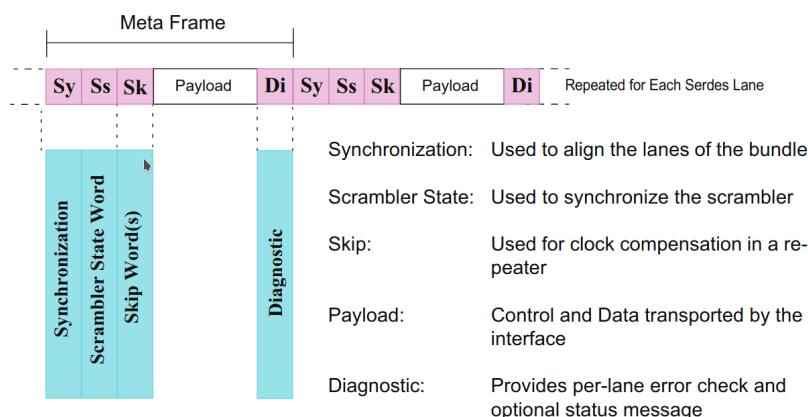


Figure 8.30: Interlaken Metaframe.

### 8.4.19.2.3 ENCODER/DECODER

The Interlaken Protocol defines the usage of 64b67b encoding. The encoder will be provided with 64-bit (63:0) words accompanied by a 2-bit header input. These bits (65:64) will be used to determine whether the provided 64-bit word is a data or control block. This is comparable to 64b66b encoding.

However there will be 67 bits at the output. The additional 67th bit is a data inversion bit to ensure better DC-balancing. This is done by constantly monitoring the running disparity in data packets. A '1' will increase the disparity and a '0' will decrease the disparity. So a disparity value of lower or higher than 32 will indicate that a word contains more ones or zeroes.

Every time a new word enters the encoder, the disparity will be compared against that of the current running disparity. If both these words contain a higher amount of the same sign, the new word will be inverted and the inversion bit will be set high. This will ensure the running disparity will stay with a bound of about 96-bit and will result in better line stability/lower BER.

At the receiver node data will be recovered by detecting for valid encoding bit transitions and aligning the data words correctly so that the data/control words is located at bits 63:0. When the inversion bit is set high, data should be inverted again to retrieve the original data.

Bits(66:64)	Interpretation bits (63:0)
001	Message contains a data word (63:0 non-inverted)
010	Message contains a control word (63:0 non-inverted)
101	Message contains a data word (63:0 inverted)
110	Message contains a control word (63:0 inverted)
others	Illegal word

**Table 8.21:** 64b67b interpretation (Interlaken).

### 8.4.19.2.4 (DE)SCRAMBLER

The 64b67b encoder always has to be accompanied by a scrambler. While the encoder applies inversion on the data/control word, the scrambler will randomize the data by using a polynomial ( $X^{58} + X^{38} + 1$ ). This is done to prevent long sequences of the same sign in data/control words which could caused undesired effects on the communication line (BER, EMI).

The Interlaken Protocol defines the usage of an independent synchronous scrambler. This means that the scrambler state has to be transmitted at pre-defined intervals to ensure the receiver descrambler is still correctly synchronized on the transmitter scrambler. This will cause additional overhead (dependent on the chosen interval) but ensures that the descrambler is always synchronous with the scrambler and can recover after processing faulty data.

Scrambling should be done before the data inversion of the encoder and only with the 64-bit word. However the synchronization and scrambler state control words are exceptions and should not be scrambled.



**Figure 8.31:** Encoding overview.

### 8.4.19.3 CONFIGURATION

By default the Interlaken receiver will be configured with parameters recommended by the Interlaken Protocol Definition. However it is possible to change some of these configurations according to desires. The amount of lanes desired by the user can be configured. However it should be noted that this is intended for channel bonding. When it's desired to implement independent lanes, it's better to instantiate the Interlaken receiver component multiple times (equal to the amount of desired lanes). Another configuration is setting the PacketLength. This is the expected length of received metaframes. Data arriving from the transmitter should contain the same metaframe length as the configured value here, otherwise this will result in an error and thus no data output. For now it should be noted that

Configuration	Default	Range/Description
Lanes	4	Amount of lanes introduced in the core
PacketLength	2048	Length of transmitted/expected metaframes

**Table 8.22:** Interlaken receiver configurations.

### 8.4.19.4 LATENCY

Several tests have been performed to determine the latency of the Interlaken core. Latency of the complete core has been measured real world on a Xilinx VCU128-ES1 (VU37P) and using a QSFP loopback module. However for the receiver side and it's individual components, simulation results have been used. Table 8.23 provides more details on the latency for each RX lane component.

RX latency	Delay cycles	25 Gbps (402.83 MHz)	10 Gbps (156.25 MHz)
RX lane	8 cycles	19.84 ns	51.2 ns
Decoder	1 cycle	2.48 ns	6.4 ns
Descrambler	4 cycles	9.92 ns	25.6 ns
Meta deframer	1 cycle	2.48 ns	6.4 ns
Burst deframer	2 cycles	4.96 ns	12.8 ns

**Table 8.23:** Interlaken RX lane latencies.

### 8.4.19.5 STATUS INDICATORS

The Interlaken receiver has several status signals. Important matters such as the status of the rx fifo or whether the decoder and descrambler are in lock will be notified through the use of a dedicated status signal.

Status signal	Description
Descrambler_Lock	Descrambler of lane is in lock
Decoder_Lock	Decoder of lane is in lock
HealthLane	Lane is ready to receive user data and has no errors
HealthInterface	Same as HealthLane but for all lanes in an Interlaken instantiation
FlowControl	Flow control status received from other end of the connection

**Table 8.24:** Interlaken receiver status signals.

Besides the described status signals, the AXI-Stream interface also contains some information in the m\_ - axis.user field. This provides the most important status signals in four bits, which makes it easy for an existing AXI-Stream entity to read the current status of the core.

Status signal	Description
m_axis.tuser(3)	RX FIFO not accepting data
m_axis.tuser(2)	Flowcontrol status (not correctly utilized yet)
m_axis.tuser(1)	Combined error signal from Meta/Burst deframing if any
m_axis.tuser(0)	Combined error signal from CRC24 and CRC32

**Table 8.25:** Interlaken m\_axis.tuser status signals.

#### 8.4.19.6 ERROR HANDLING

Several signals have been added to notify the user of an error condition in the core and also what is the cause of the error. The core features several error signals to notify the user of problems. Two error signals are provided which indicate whether a CRC24 or CRC32 error has occurred. This should invalidate the data.

Error signal	Description
decoder_error_sync	Decoder cannot synchronize on preamble bits
descrambler_error_badsync	Descrambler received three bad sync word while being in lock (can be caused by wrong Meta length)
descrambler_error_statemismatch	Descrambler received three wrong Scrambler State words while being in lock
descrambler_error_nosync	No Synchronization words detected in data stream
burst_crc24_error	Burst packet contains faulty CRC24
meta_crc32_error	Meta frame contains faulty CRC32

**Table 8.26:** Interlaken core error signals.

#### 8.4.19.7 ESTIMATED RESOURCE USAGE

The Interlaken Core has been implemented on a Xilinx VCU128-ES1 (VU37P) to determine the required resources. Since this only concerns the receiving side, the estimate amount of resources utilized by the Interlaken receiver will be described. Table 8.27 depicts the resources consumed by a Interlaken receiver implementation on a per lane basis. This has been a design with four unbonded channels, so they function independently from each other. This also means that all lanes consume nearly an equal amount of resources. In this case the single lane mentioned represents lane 0 in the design.

RX logic	LUT	FF	BRAM	LUT%	FF%	BRAM%
Single RX lane (incl fifo)	759 (895)	933 (1111)	0 (5.5)	0.06 (0.07)	0.04 (0.04)	0.00 (0.27)
Burst deframer	40	203	0	<0.01	<0.01	0
Meta deframer	344	298	0	0.03	0.01	0
Descrambler	322	345	0	0.02	0.01	0
Decoder	54	87	0	<0.01	<0.01	0

**Table 8.27:** Interlaken resource utilization RX logic.

## 8.5 ENCODING

### 8.5.1 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).

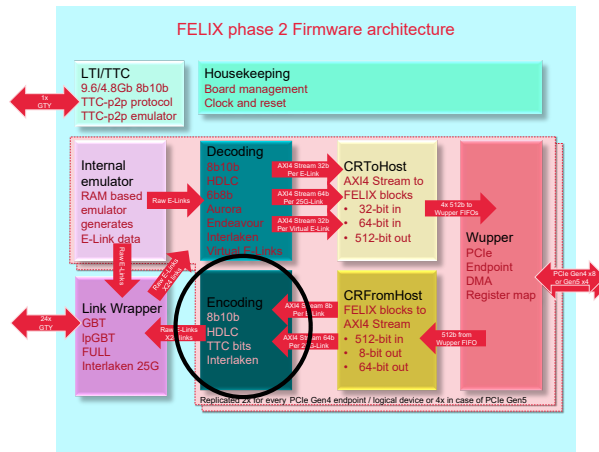


Figure 8.32: The encoding block in the toplevel diagram.

### 8.5.2 INTERFACES

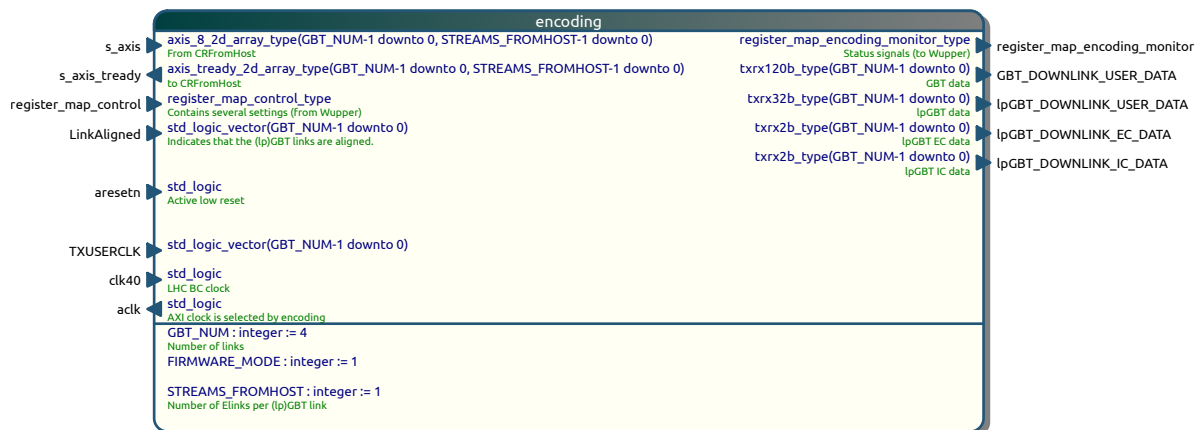


Figure 8.33: The encoding block, instantiating all encoder entities based on FIRMWARE\_MODE.

#### 8.5.2.1 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 8.5.6). This gearbox can be configured to support all E-link widths in GBT and IpGBT 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 AxiStream-ToByte entity was introduced which will take care of the conversion, but also contain the axi stream E-Path **FIFO**.

#### 8.5.2.2 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.

#### 8.5.2.3 INTERFACE TO LINKWRAPPER

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

- GBT:  $\text{GBT\_NUM} * 120\text{b}$
- lpGBT:  $\text{GBT\_NUM} * (32\text{b}(\text{E-Links}) + 2\text{b}(\text{EC}) + 2\text{b}(\text{IC}))$
- Interlaken:  $\text{GBT\_NUM} * 76\text{b}$

### 8.5.3 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.

### 8.5.4 CONFIGURATION

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

### 8.5.5 STATUS INDICATORS

Status of the different encoders can be monitored in `register_map_encoding_monitor`.



## 8.5.6 ENCODING GEARBOX

### 8.5.6.1 INTRODUCTION

for lpGBT and GBT based firmware flavours, the data is first encoded in either HDLC, 8b10b or 6b8b or as parallel TTC bits, and needs to be converted to the width of the E-Link (2, 4 or 8 bits) per BC clock cycle.

The different protocol encoders require different data widths per BC clock cycle, the Encoding Gearbox will read these different data widths by means of shift registers and convert it to the E-Link width. The available widths on in- and output of the gearbox will be partly configurable at runtime and partly at build time.

### 8.5.6.2 INTERFACES

#### 8.5.6.2.1 OVERVIEW

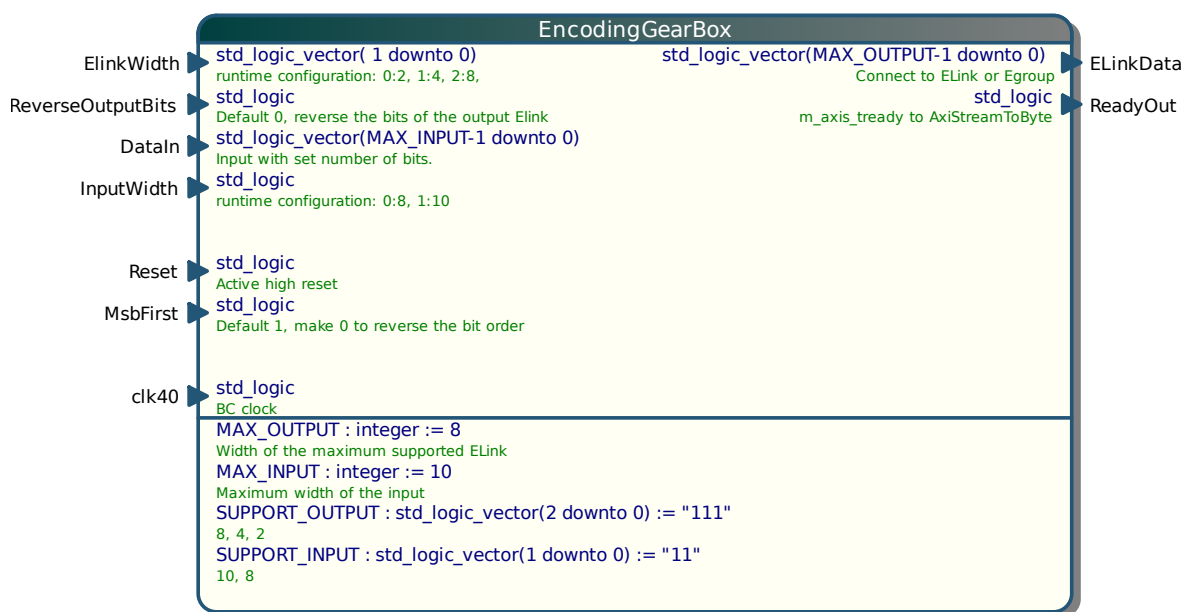


Figure 8.34: The Encoding GearBox entity.

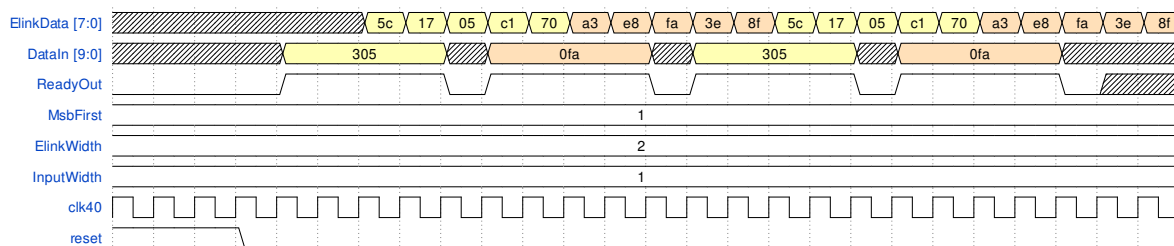


Figure 8.35: EncodingGearBox running with 8 bit output, 10 bit input. The data is alternating 0x305/0x0FA (k28.5+). [7].

#### 8.5.6.2.2 INTERFACE TO GBT OR LPGBT WRAPPER

Data to 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\_OUTPUT will be set. For instance, a GBT mode E-Group will contain 2 Gearboxes with MAX\_OUTPUT set to 8, 2

Gearboxes with MAX\_OUTPUT set to 4 and 4 Gearboxes with MAX\_OUTPUT set to 2. This way a total of 8 streams of variable bandwidth (80, 160 or 320 Mb/s) can be created.

#### 8.5.6.2.3 INTERFACE FROM ENCODERS

2 ports are connected to the different protocol decoders: DataIn and ReadyOut.

##### **DataIn**

The output bandwidth / number of bits should not exceed MAX\_OUTPUT. For an 8 bit E-link in 8b10b mode, the InputWidth has to be set to 10 bits("010"), this way every clock cycle carries 1 8b10b word on InOut if ReadyOut = '1'.

##### **ReadyOut**

ReadyOut indicates that the gearbox is ready to accept new data from the encoder, and the correct number of bits should be available on DataIn.

#### 8.5.6.3 FUNCTIONAL DESCRIPTION

Depending on the configuration, the EncodingGearBox will shift a number of bits of DataIn into a shift register every clockcycle when ReadyOut = '1'. ElinkData is always valid and carries the shifted number of bits, ready to be transmitted over IpGBT or GBT.

#### 8.5.6.4 CONFIGURATION

**Buildtime configuration** 4 generics of the DecoderGearBox define its functionality.

- MAX\_OUTPUT: Defines the maximum number of bits that is supported at ElinkData
- MAX\_INPUT: Defines the maximum number of bits that is supported at DataIn
- SUPPORT\_OUTPUT: a 3 bit vector of which every bit configures a supported output 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
- SUPPORT\_INPUT: a 2 bit vector of which every bit configures a supported input width to be configured
  - 0: 8 bit output is supported
  - 1: 10 bit output is supported

##### **Runtime configuration**

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

- ElinkWidth[1:0] can be connected to a register of the Wupper register map to configure the width of the E-Link to be encoded. 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]
- InputWidth[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

### 8.5.6.5 STATUS INDICATORS

EncodingGearBox has no status indicators. Status of the protocol encoder has to be provided by the encoder itself.

### 8.5.6.6 LATENCY

The Encoding Gearbox has a latency for all configurations of 2 clockcycles (40,079 Mhz, 25 ns), that means the output data will be valid 2 clockcycles after the data was shifted into the shift register.

### 8.5.6.7 ERROR HANDLING

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

### 8.5.6.8 ESTIMATED RESOURCE USAGE

#	Out2	Out4	Out8	In8	In10	LUT	FF	Remark
1	✓				✓	15	24	8b10b
2	✓			✓	✓	15	24	Direct, 8b10b
3	✓	✓			✓	28	30	8b10b
4	✓	✓		✓	✓	33	30	Direct, 8b10b
5	✓	✓	✓		✓	58	41	8b10b
6	✓	✓	✓	✓	✓	63	41	Direct, 8b10b

**Table 8.28:** Estimated resource consumption for Encoding Gearbox, depending on different build-time configurations.

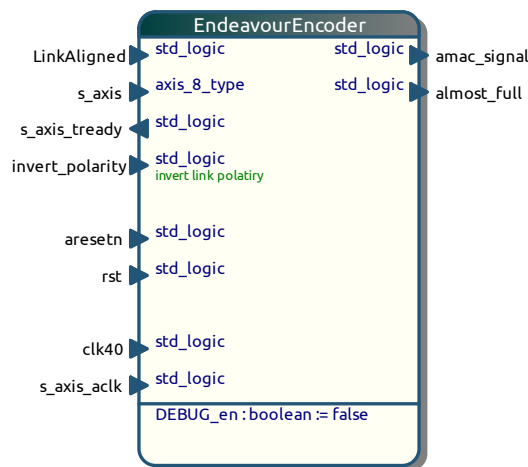
## 8.5.7 ENDEAVOUR ENCODER

### 8.5.7.1 INTRODUCTION

Strips firmware has blocks for communicating 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. The Endeavour is a serial “Morse code” protocol, which tolerates  $\pm 50\%$  variation with respect to the nominal 40 MHz AMAC ring-oscillator frequency.

Endeavour Encoder serializes data for sending it to AMAC chips. Polarity of the encoder can be configured by setting bitfield `INVERT_AMAC_OUT` of register `GLOBAL_STRIPS_CONFIG`.

### 8.5.7.2 INTERFACES



**Figure 8.36:** The Endeavour encoder entity.

The Endeavour Encoder inputs data from the FromHost Central Router (CRFromHost) via 8-bit AXI Stream interface, and outputs the generated pulse sequence to an E-Link. In the Strips firmware, the data input is connected to the EC elink of IpGBT frame. Both bits in EC field are set to the same value, reducing the effective sampling frequency to 40 MHz.

Module ports are listed below. Unless otherwise indicated, the input signals are sampled in `clk40` domain.

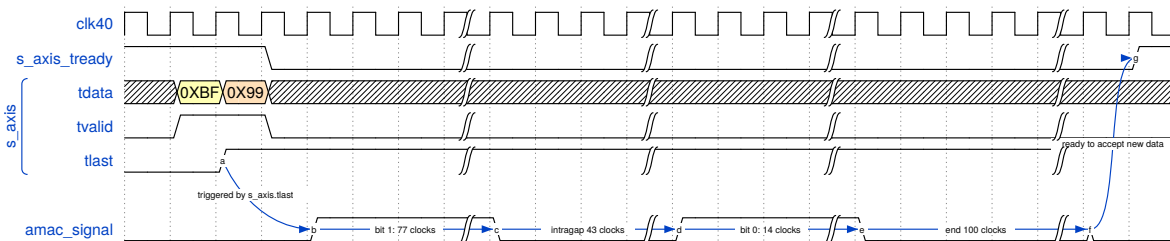
- `clk40` - **BC** clock driving the encoder logic
- `s_axis_aclk` - clock for communication with the Central Router
- `amac_signal` - output “Morse code” signal to the AMAC chip
- `LinkAligned` (active HIGH) - indicates that the IpGBT link is aligned and encoding may be enabled
- `aresetn` - asynchronous reset for the AXI stream FIFO. Sampled in `s_axis_aclk` domain.
- `rst` - synchronous reset for the main logic
- `s_axis` - input AXI Stream
- `invert_polarity` - inverts polarity of `amac_signal`
- `s_axis_tready` - indicates that the module is ready to accept more data. Sampled in `s_axis_aclk` domain.

### 8.5.7.3 FUNCTIONAL DESCRIPTION

The Endeavour Encoder converts data arriving from CRFromHost to a series of pulses, compliant with the serial Endeavour protocol. Table 8.29 lists the timing of the pulse waveform.

Waveform feature	Width in BC periods
ZERO pulse	14
ONE pulse	77
Bit gap	43
Word gap	100

**Table 8.29:** Endeavour protocol.



**Figure 8.37:** example of waveform.

### 8.5.7.4 ESTIMATED RESOURCE USAGE

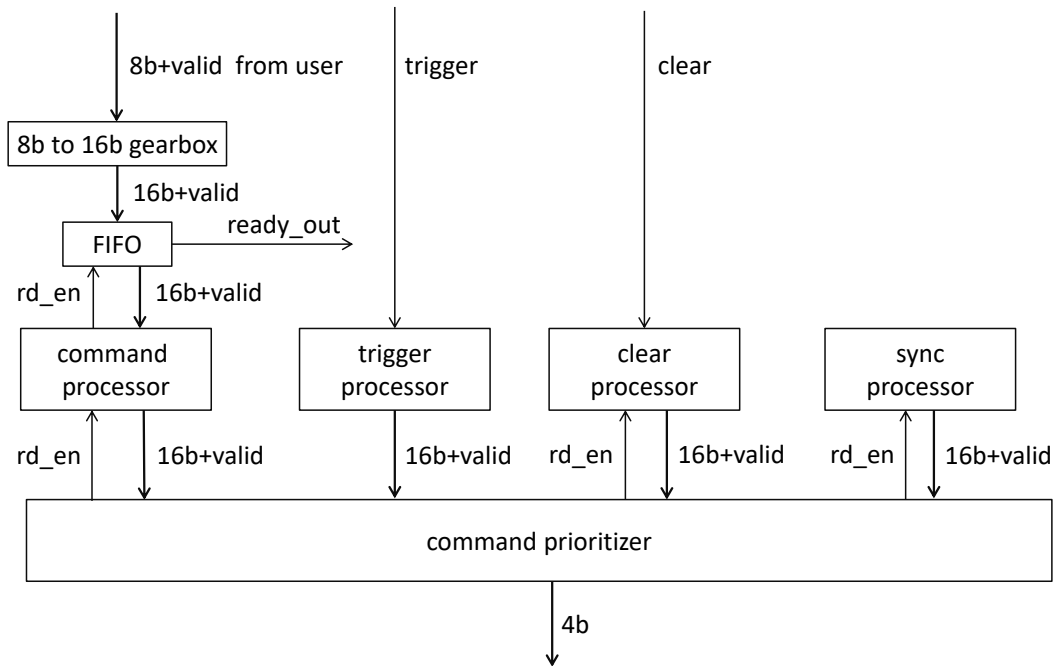
Resource	lpGBT link	24 GBT links	% (XKCU115)
LUTs	44	1056	<0.1%
Flip-Flops	29	696	<0.1%
Block RAM	0.5	12	0.5%

**Table 8.30:** Resource consumption of Endeavour Encoder module.

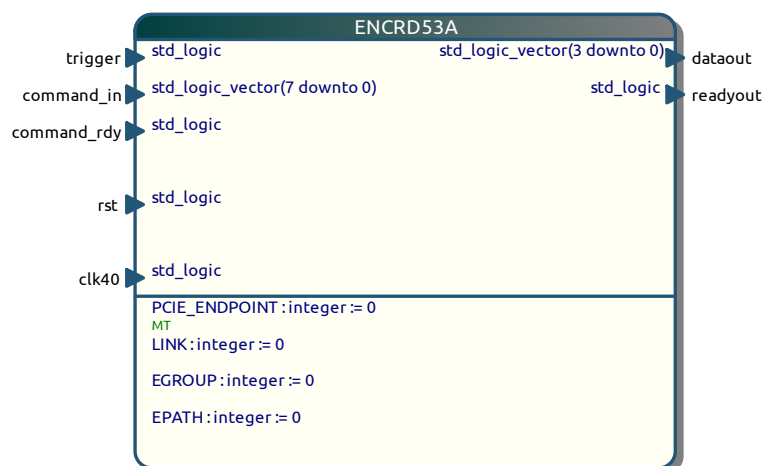
### 8.5.8 ITkPix ENCODER

ITkPix needs to receive a 160 Mbps bitstream composed of 16-bit frames for control and configuration as described in [8]. ITkPix also uses the bitstream to recover an internal clock synchronous to the 40 MHz LHC clock. Trigger, write register, read register, and other commands are composed of one or more frames. Single-frame commands such as trigger and global pulse need to be transmitted synchronously (with fixed latency) with the LHC beam or with each other. The multi-frame commands such as read and write register can be transmitted asynchronously.

An ITkPix encoder receives 40 MHz clock, a trigger and trigger tag signals, an 8-bit command with a valid signal, a clear signal, and a reset signal as shown in Figs. 8.38 and 8.39. The reset and 40 MHz clock signals go to all components of the encoder and they are not shown in the block diagram. The trigger and clear signals come from the LTI-TTC decoder and are identical for all the ITkPix encoders.



**Figure 8.38:** Dataflow in the ITkPix encoder.



**Figure 8.39:** The RD53A/B encoder entity.

The trigger and clear signals are translated into ITkPix commands by the trigger processor and clear processor blocks. The 8-bit input bus is connected to a gearbox to form 16-bit frames. The 16-bit frames from the gearbox are stored in a FIFO to prevent data losses since the input data rate to the encoder exceeds the output data rate. The command processor reads from the FIFO to verify and buffer the incoming commands. The verification and buffering are used for debugging. The processor can also generate sequences of calibration pulse and trigger commands for ITkPix calibrations. A sequence is initiated with a special 16b command. The binary data format for the special ccommand is 111 + number of iteration(7b) + period (6b). The number of iterations is how many times the sequence is sent to ITkPix and the period is the time between sending two sequences in units of 1.6  $\mu$ s. The effective frequency of sequence transmission can vary from 10 kHz to 625 kHz. Sequence data are fully programmable. A typical sequence includes one global pulse and 16 triggers. Therefore, the trigger frequency varies from 160 kHz to 10 MHz.

The command prioritizer reads 16 bit frames from one of the four sources depending on the source priority. The trigger commands have the highest priority and they are read as soon as they are available. The next highest in priority is the clear command followed by the sync command. A sync command is available once every 32 frames. Frames from the command processor are the lowest in priority. If no input data are available, the prioritizer samples NOOP commands. The prioritizer samples 16-bit frames into 4-bit words with adjustable phase.

An encoder takes 481 LUTs, 404 FFs, 1 RMB18 and 1 DSP. There are 8 encoders per IpGBT link and there are 24 IpGBT links per FELIX. Therefore, encoders for an IpGBT link require 3848 LUTs, 3232 FFs, 8 RAMB18, and 8 DSPs. Encoders for 24 IpGBT links require 92352 LUTs, 77568 FFs, 192 RAMB18, and 192 DSPs.

Trickle config of ITkPix is expected to be done by transmitting commands and data from from the host server (via PCIe) since the target FPGAs do not have enough memory to store all the configuration data.

## 8.5.9 ITk STRIPS LCB ENCODER

### 8.5.9.1 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 8.40. The blocks shown in red are data inputs. The corresponding FromHost elink IDs are listed in Table 8.31).

Polarity of the encoder can be adjusted by setting bitfield `INVERT_LCB_OUT` of FELIX register `GLOBAL_STRIPS_CONFIG`.

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 8.31:** Strips ToHost elilnk mapping. In this table, elink mapping of IpGBT optical link 0 is listed. To find elink IDs for encoders of another optical link, add  $0x40 \times (\text{IpGBT link ID})$  to the elink IDs listed in the table..

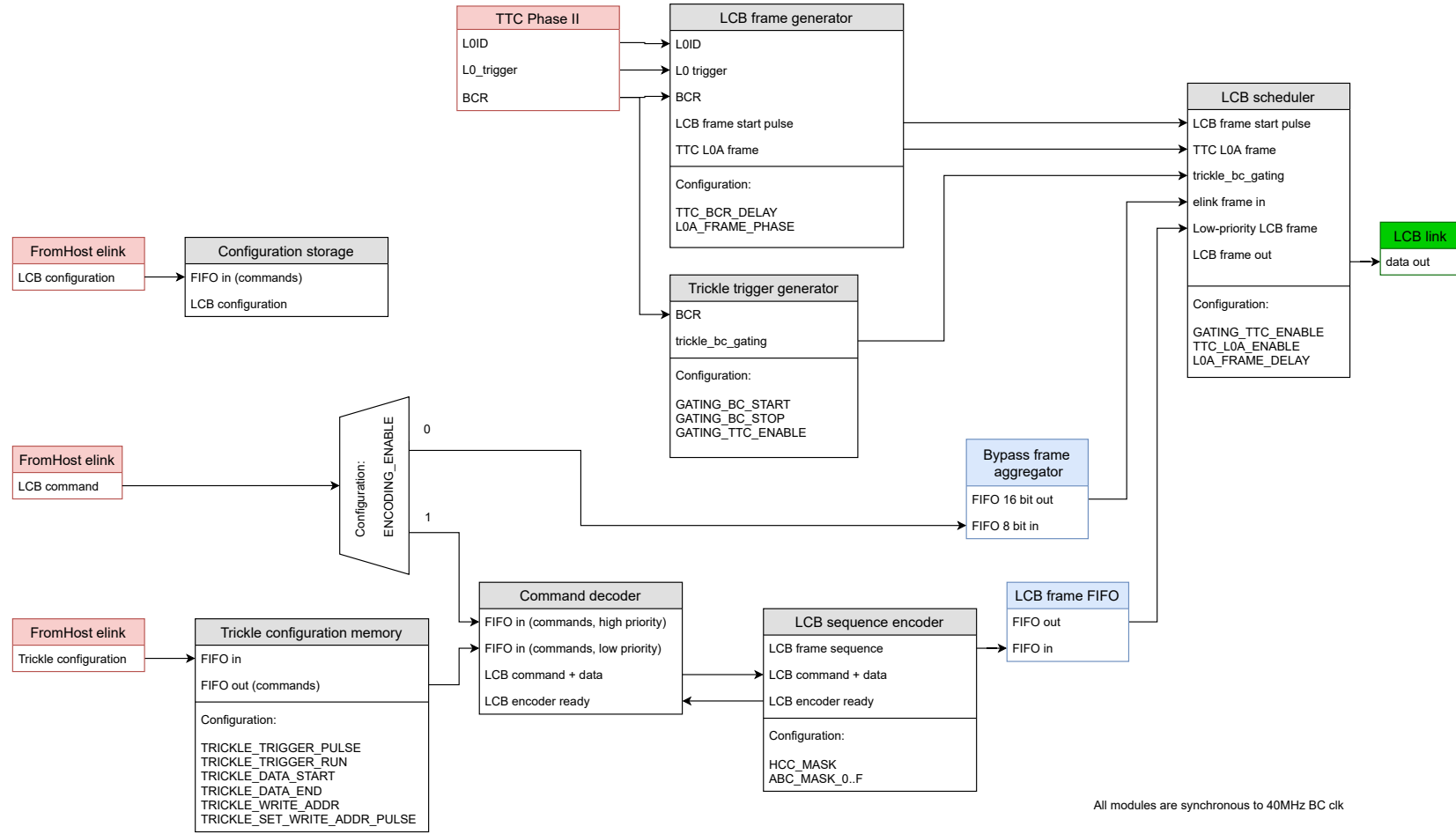


Figure 8.40: Functional diagram of ITk Strips LCB Encoder module.

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

**Figure 8.41:** LCB link configuration command format.

### 8.5.9.2 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.

**8.5.9.2.1 CONFIGURATION COMMAND.** Configuration commands update LCB configuration registers given the data and the register address. (Fig. 8.41). The configuration registers of the LCB encoder are listed in the Table 8.32. 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 8.32). Each command sequence must be completed within 100 ms. Incomplete command sequences are discarded by the encoder.

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.

### 8.5.9.3 LCB FRAME GENERATOR SUBMODULE

This submodule determines the phase of LCB frame and generates LOA frames in response to the signals from the TTC module. The exact contents of LOA 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 LOA 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 8.3:** *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.

### 8.5.9.4 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. Each two-byte frame

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	14	Address of the first valid byte in the trickle configuration memory
0x0A	TRICKLE_DATA_END	14	Address of the last valid byte in the trickle configuration memory
0x0B	TRICKLE_WRITE_ADDR	14	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 8.5.9.6.
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 8.32:** LCB link configuration registers.

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

**Figure 8.42:** No operation command format.

sequence must be completed within 100 ms. Incomplete frame sequences are discarded by the aggregator. Bypass frames are treated as low-priority frames by the frame scheduler.

#### 8.5.9.5 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 16 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 8.5.9.6 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 8.5.9.11 for the setup procedure.

#### **Remark 8.4:** *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 8.5.9.11.

#### 8.5.9.6 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. Each command sequence must be completed within 100 ms. Incomplete command sequences are discarded by the encoder. Below is the list of valid commands and their format.

**8.5.9.6.1 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. 8.42.

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

**Figure 8.43:** IDLE command format.

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

**Figure 8.44:** LOA command format.

**8.5.9.6.2 IDLE COMMAND.** Places a single IDLE frame into the LCB link queue (Fig. 8.43). IDLE can be written into trickle configuration memory as a part of the calibration sequence to add 100 ns delay between commands.

**8.5.9.6.3 LOA COMMAND.** This issues a user-defined LOA frame to the front-end (Fig. 8.44). At least a single bit in mask or BCR must be set to '1' for the command to be valid. Invalid LOA commands are ignored by the command decoder.

**8.5.9.6.4 FAST COMMAND.** Fast command sends a user-defined fast command to the front-end (Fig. 8.45).

**8.5.9.6.5 REGISTER COMMANDS.** Register commands issue a read (Fig. 8.46) or write (Fig. 8.47) frame sequence for HCC\* or ABC\* register.

**8.5.9.6.6 BLOCK COMMANDS.** Block write command (Fig. 8.48) allows writing contiguous blocks of registers with a single command, reducing the command memory overhead. Upon receiving a block write command, the command decoder will issue a sequence of ABC\* or HCC\* register write commands to the LCB link. The first register data word is sent to the specified register address, which is then incremented for each subsequent data word. Byte #3 specifies the number of the following data words in the block sequence, where zero corresponds to a single data word.

### 8.5.9.7 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.

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

**Figure 8.45:** 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 8.46:** 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 8.47:** Register write command format.

Byte \ Bit	7	6	5	4	3	2	1	0
0	0xB2 (ABC*) or 0xB3 (HCC*)							
1	Register address							
2	HCC ID				ABC ID			
3	Number of data words - 1							
4	Register #0 data [31:24]							
5	Register #0 data [23:16]							
6	Register #0 data [15:8]							
7	Register #0 data [7:0]							
...	...							
	Register #N data [15:8]							
	Register #N data [7:0]							

**Figure 8.48:** Block write command format.

### 8.5.9.8 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.

### 8.5.9.9 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 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 8.5: 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 8.6: 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.

### 8.5.9.10 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 8.7: 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`.



### 8.5.9.11 EXAMPLES

#### 8.5.9.11.1 SENDING BASIC LCB COMMANDS VIA LCB COMMAND ELINK AND COMMAND DECODER (ENCODING\_ENABLE=1)

- Fast command example (command=6, BC=3): 0x81 0x36
- LOA 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

#### 8.5.9.11.2 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
- LOA 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

#### 8.5.9.11.3 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 [8.5.9.6](#)

#### 8.5.9.11.4 ISSUING SOFTWARE-GENERATED TRICKLE TRIGGER

8.5.9.11.1 SINGLE LCB ELINK. Issue trickle trigger to a single elink by writing '1' to TRICKLE\_TRIGGER\_PULSE configuration register.

8.5.9.11.2 CONTINUOUS TRICKLE CONFIGURATION. Send trickle configuration continuously by writing '1' to TRICKLE\_TRIGGER\_RUN configuration register.

8.5.9.11.3 ALL LCB ELINKS SIMULTANEOUSLY. Issue trickle trigger to LCB encoder elink by writing '1' to FELIX register GLOBAL\_STRIPS\_CONFIG.TRICKLE\_TRIG\_PULSE.

8.5.9.11.4 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

8.5.9.11.5 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

8.5.9.12 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)

8.5.9.13 ESTIMATED RESOURCE USAGE

Resource	E-Group	lpGBT link	24 lpGBT links	% (XKCU115)
LUTs	661	2644	63456	10%
Flip-Flops	899	3596	86304	7%
Block RAM	5.5	22	528	25%

**Table 8.33:** Resource consumption of LCB encoder module for XKCU115.

## 8.5.10 ITk STRIPS R3L1 ENCODER

### 8.5.10.1 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 [8.49](#).

Polarity of the encoder can be adjusted by setting bitfield `INVERT_R3L1_OUT` of FELIX register `GLOBAL_STRIPS_CONFIG`.

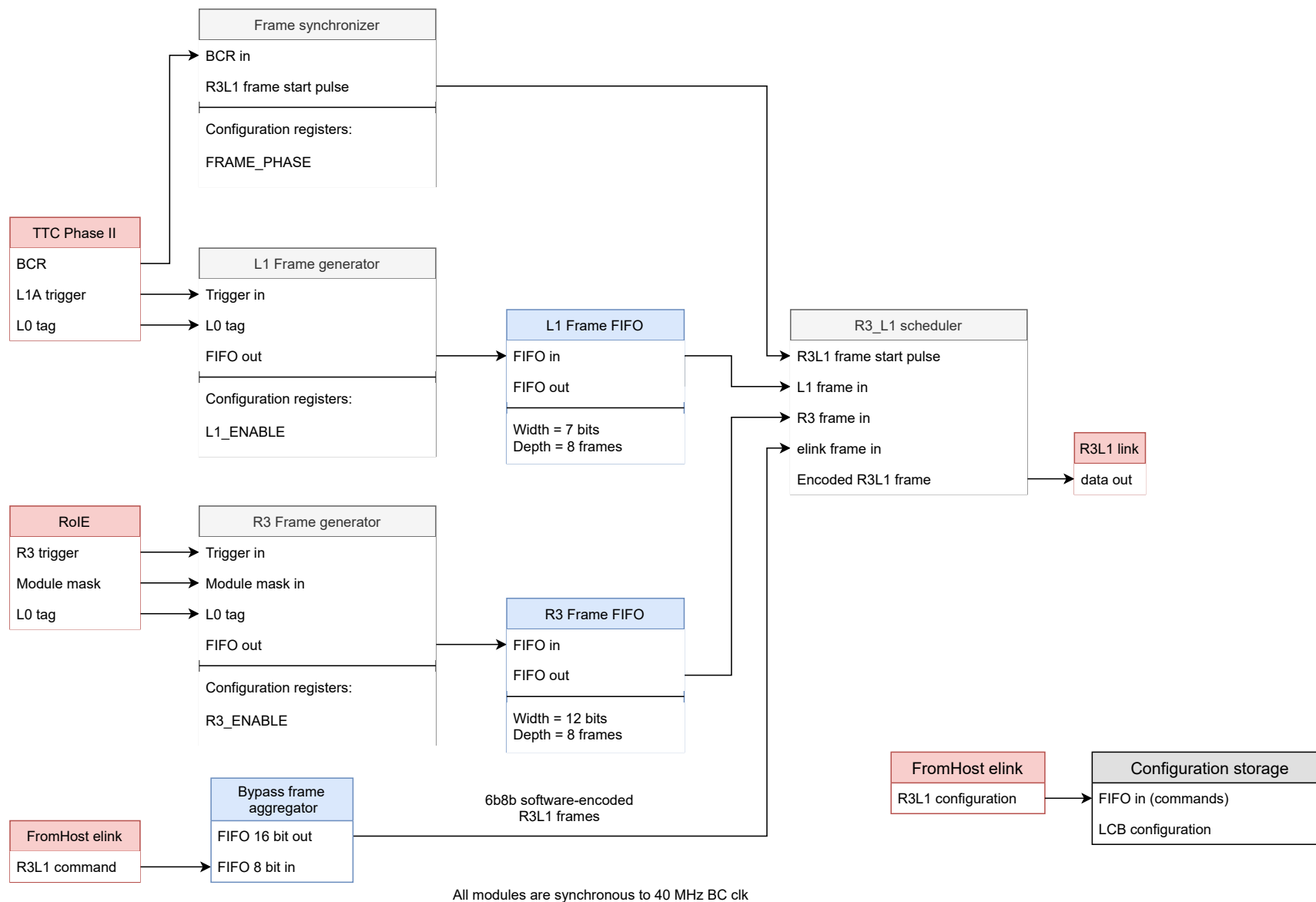


Figure 8.49: Functional diagram of ITk Strips R3L1 Encoder module.

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

**Figure 8.50:** 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 8.34:** R3L1 link configuration registers.

### 8.5.10.2 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.

**8.5.10.2.1 CONFIGURATION COMMAND.** Configuration commands update R3L1 configuration registers given the data and the register address. (Fig. 8.50). The corresponding FromHost elink IDs are listed in Table 8.31). The configuration registers of the R3L1 encoder are listed in the Table 8.34. 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 8.34). Each command sequence must be completed within 100 ms. Incomplete command sequences are discarded by the encoder.

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.

### 8.5.10.3 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.

### 8.5.10.4 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’.

### 8.5.10.5 R3 AND L1 FRAME FIFOs

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

#### 8.5.10.6 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. Each two-byte frame sequence must be completed within 100 ms. Incomplete frame sequences are discarded by the aggregator.

#### 8.5.10.7 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

#### 8.5.10.8 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)

#### 8.5.10.9 ESTIMATED RESOURCE USAGE

Resource	E-Group	lpGBT link	24 GBT links	% (XKCU115)
LUTs	205	820	19680	3%
Flip-Flops	292	1168	28032	2%
Block RAM	1	4	96	4%

**Table 8.35:** Resource consumption of R3L1 encoder module.

## 8.5.11 8B10B ENCODER

### 8.5.11.1 INTRODUCTION

The 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 IpGBT mode firmware flavours.

The tasks for the 8b10b encoder are:

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

### 8.5.11.2 INTERFACES

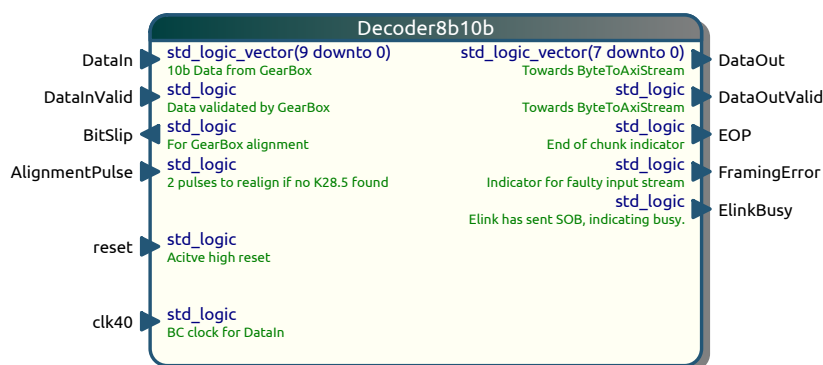


Figure 8.51: The 8b10b Encoder entity.

#### 8.5.11.2.1 INTERFACE TO AXISTREAMTOBYTE

The 8b10b encoder receives from the AxisStreamToByte:

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

The encoder sends towards the AxisStreamToByte:

- readyOut: flag signaling that the encoder is ready to accept new data; it corresponds to the *m\_axis\_tready* flag of the AxisStreamToByte.

#### 8.5.11.2.2 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)

### 8.5.11.3 FUNCTIONAL DESCRIPTION

#### 8.5.11.3.1 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 8.36). 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.

#### 8.5.11.3.2 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 8.36:** 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 + CharlsK is well defined in other literature, and the code has been implemented in phase 1 FELIX.

#### 8.5.11.4 CONFIGURATION

The meaning of the different comma characters in table 8.36 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.

#### 8.5.11.5 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.

#### 8.5.11.6 ERROR HANDLING

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

#### 8.5.11.7 ESTIMATED RESOURCE USAGE

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

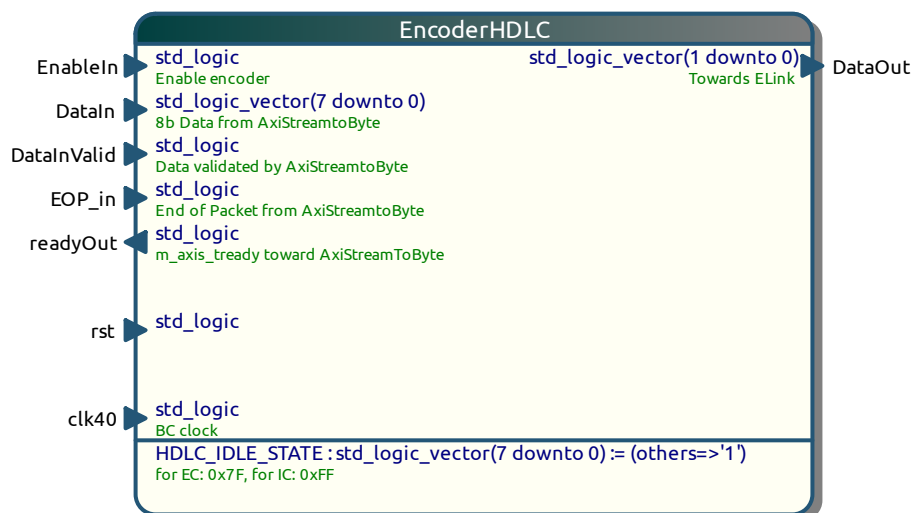


## 8.5.12 HDLC ENCODER

### 8.5.12.1 INTRODUCTION

The HDLC Protocol [10] 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 IpGBT.

### 8.5.12.2 INTERFACES



**Figure 8.52:** The HDLC encoder entity.

#### 8.5.12.2.1 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.

#### 8.5.12.2.2 INTERFACE FROM AXISTREAMTOBYTE

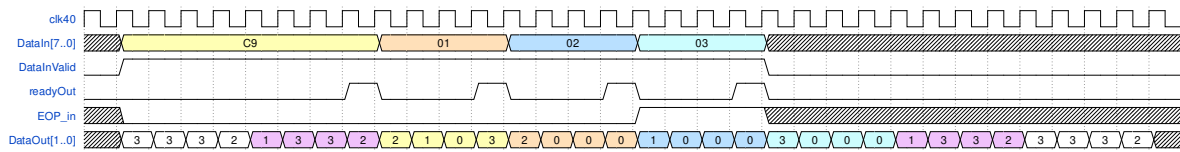
The signals that connect the HDLC Encoder to AxiStreamToByte directly translate to AXI Stream signals, however multiple encoders (8b10b, direct) are implemented within one E-Path, so there may be connection logic in between ByteToAxiStream and EncoderHDLC.

- 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.

#### 8.5.12.2.3 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 IpGBT 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.

### 8.5.12.3 FUNCTIONAL DESCRIPTION



**Figure 8.53:** 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 8.53.

### 8.5.12.4 CONFIGURATION

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

### 8.5.12.5 STATUS INDICATORS

The HDLC Encoder has no status indicators.

### 8.5.12.6 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.

### 8.5.12.7 ERROR HANDLING

There is no error handling built into the HDLC Encoder.

### 8.5.12.8 ESTIMATED RESOURCE USAGE

The resource usage of the complete Encoding Epath for GBT will be covered in section 8.5.

## 8.5.13 DIRECT MODE E-LINK ENCODER

### 8.5.13.1 INTRODUCTION

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

#### **Remark 8.8:** *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.

### 8.5.14 TTC ENCODER

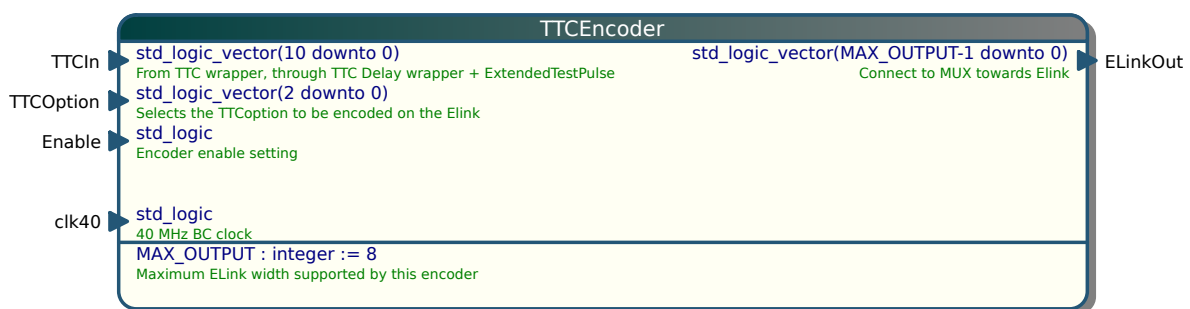
#### Remark 8.9: TTC for phase II

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.

#### 8.5.14.1 INTRODUCTION

For Phase 1, the standard encoded LHC TTC [13] 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 IpGBT 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.

#### 8.5.14.2 INTERFACES



**Figure 8.54:** 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 8.37.

#### 8.5.14.3 FUNCTIONAL DESCRIPTION

Each E-link can be configured to choose bits from the possible bits shown in Table 8.37, 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 8.37:** 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
-----------	-----------	-----------	-----------	-----------	-----------	-----	-----	--------	-----

##### 8.5.14.3.1 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 8.37 with one extra bit added for the test pulse. The delayed bits are described in Table 8.38

**Table 8.38:** Below is a copy of the bits found in 8.37 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
----	-----------	-----------	-----------	-----------	-----------	-----------	-----	-----	--------	-----

### 8.5.14.3.2 TTC OPTIONS

Table 8.39 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 8.39:** 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						BCR	L1A
2	8 bits	B-chan	Brc_d4[3]	Brc_d4[2]	Brc_d4[1]	Brc_d4[0]	ECR	L1A
3	8 bits	L1A	Brc_d4[1]	TP	ECR	OCR*	L0A*	LFSR*
4	4 bits					BCR	BCR	BCR
5 (LTDB)	2 bits						BCR	BCRd*
6	8 bits	L1A	Brc_d4[1]**	TP	ECR	OCR*	L0A*	Brc_d4[3] Brc_d4[2]
7	4 bits					ECR	BCR	L1A
8	8 bits							HGTD Fast Command, 6b8b encoded
9	8 bits	L1A	Brc_d4[1]	TP	ECR	OCR*	L0A*	Brc_d4[3] Brc_d4[2]

In Table 8.39, 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)
- LFSR: To mitigate a deadlock in the GBTx descrambler, we can toggle some bits with pseudo random numbers. This bit is generated with Listing 8.2

```

lfsr_proc : process (clk40)
begin
    if rising_edge (clk40) then
        if (enable = '0') then
            LFSRstate <= "1010101010";
        else
            LFSRstate <= LFSRstate(9) xor LFSRstate(7) xor LFSRstate(6) xor
                LFSRstate(1) & LFSRstate(9 downto 1);
        end if;
    end if;
end if;

```

end process ;

**Listing 8.2:** LFSR Pseudo random generator for TTC option 3 bit 0.

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 constraints, 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.

\*\* For option 6, the Brc\_d4[1] bit is a latched version. The other bits are equal to TTC option 3.

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  $\times 4$  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 8.40 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 8.39 and assigns the meanings to the various broadcast bits as shown in Table 8.40. 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 8.40:** 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, “ECOR” 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 [12]. ECOR and L0A, are reserved for Phase 2; for Phase 1, FELIX sends ECR and L1A for ECOR 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:	ECOR	SCA_reset	L0A	BCR+OCR	ECR BCR	TestPulse BCR	SoftReset BCR	L1A 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 ECOR 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].

#### 8.5.14.4 CONFIGURATION

The TTC Encoder has two configuration inputs:

- TTCOption[2:0] which directly translates to the TTC encoding option described in Table [8.39](#)
- Enable to enable the TTC Encoder entity.

#### 8.5.14.5 STATUS INDICATORS

The TTC Encoder has no status indicators.

#### 8.5.14.6 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.

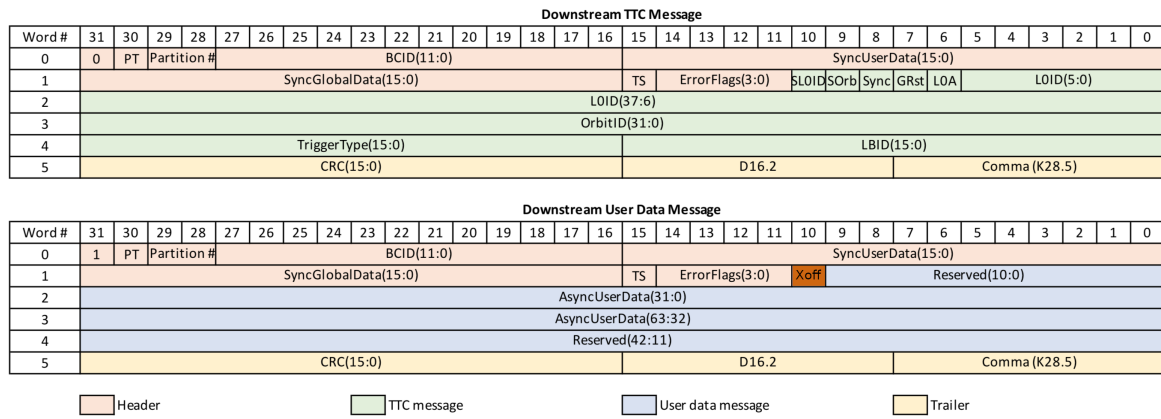
#### 8.5.14.7 ERROR HANDLING

The TTC Encoder has no error handling.

## 8.6 LTI ENCODER

Local Trigger Interface (LTI) modules [14] will distribute Trigger, Timing, and Control (TTC) signals to FELIX during the HL-LHC operations (see table 2.2 of [15]). In the FULL Mode (Phase2 variant) and INTERLAKEN flavours, FELIX will transmit a copy of the received LTI downlink data. The LTI encoded downlink (from FELIX to frontends) is 8b/10b encoded and it operates at 9.6 Gb/s. (see Fig. 8.55.

the downlinks provides user or trigger signals every cycle of the 40 MHz clock as shown in Fig. 8.55. The BCID, L0ID, OrbitID, TriggerType, and LBID are output to host upon reception of Level-0 Accept (L0A). All these signals are associated with the level-0 accept decision. The BCR signal for the front-end system is derived using the Turn Signal (TS) which has fixed latency in respect to the LHC bunch structure. The phase of the Turn Signal in respect to the bunch structure is independent of the trigger latency. The BCID counter in FELIX is set to a configurable value upon reception of the TS pulse. The BCR pulse is generated when the counter reaches zero. The BCID counter is used for incrementing the local OrbitID. Global Reset (GRst), L0A, Set L0ID, sSet OrbitID, and other signal will also be passed to the front-end systems. The LTI protocol offers 16 bits for synchronous signals (e.g. to run calibrations) and 64 bits for various asynchronous signals. This provides flexibility to FELIX firmware to meet the requirements of the front-end systems.



**Figure 8.55:** The TTC message sent from the FELIX to Frontend (32 bytes) presented as six 32-bit words.

Encoding and decoding of the 8b/10b signals is handled in the multi-gigabit transceiver (MGT). The firmware assembles the downlink messages from 32-bit words from the MGT by identifying D16.2 and K28.5 characters. It also checks the data for bit-errors with the 16-bit CRC field. The uplink TTC messages are sampled into 32-bit words to match the MGT interface. The clock frequency is 240.474 MHz. The FPGA resource utilization is insignificant.

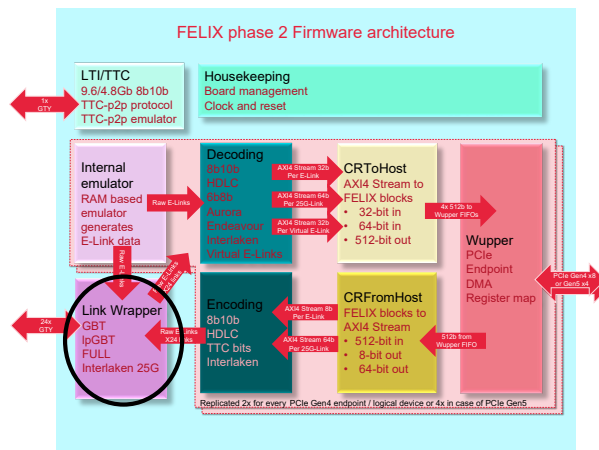
The LTI Encoder format differs from the TTC-LTI format by one bit: In the Downstream user data message, one of the reserved bits has been associated with Xoff functionality.



## 8.7 LINK WRAPPER

### 8.7.1 INTRODUCTION

As shown in Figure 8.2, 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.



**Figure 8.56:** The link wrapper in the toplevel diagram.

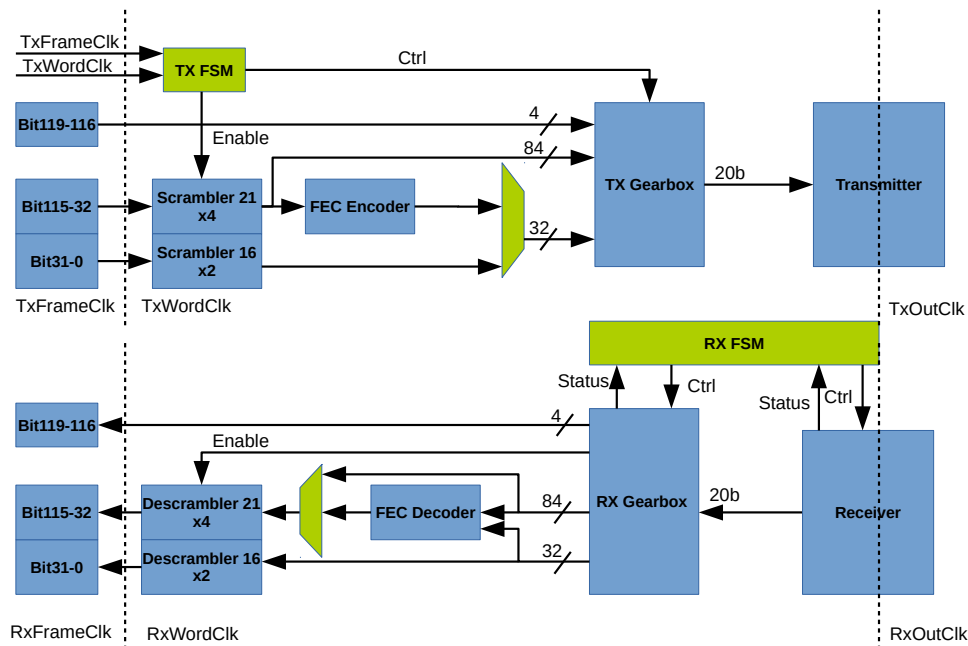
The basic link encoding and decoding is also performed inside the Link Wrapper. The basic protocols that are encoded and decoded inside the link wrapper are:

- 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.
- IpGBT: 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.

### 8.7.2 FUNCTIONAL DESCRIPTION

#### 8.7.2.1 GBT MODE WRAPPER

A wrapper shown in Figure 8.57 is provided to include the Xilinx transceiver and the GBT encoding and decoding modules.



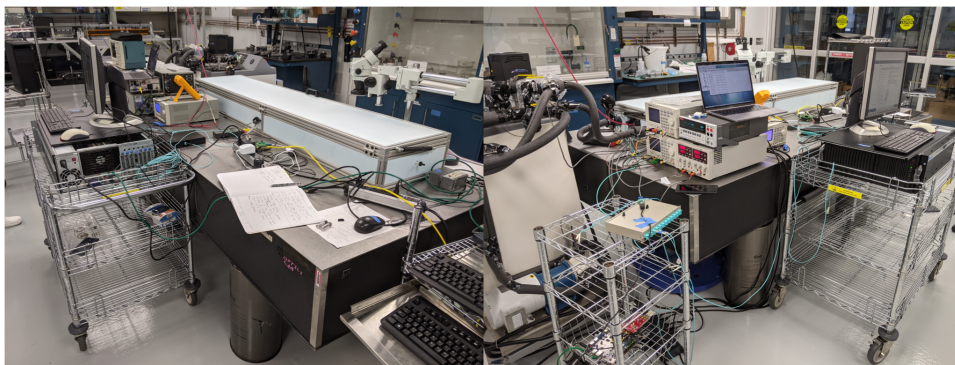
**Figure 8.57:** Block diagram for the GBT module in the link wrapper.

The transceiver is configured as 4.8 Gbps for both directions. The GTH transceivers can be configured in unit of one channel, which uses the CPLL in the transceiver, or be configured to in unit of one quad of four channels, which uses the high quality QPLL in transceiver. The GBT encoding and decoding module are based on the code from the CERN GBT group. It has the forward error correction (FEC) capability. Data is scrambled before the FEC encoding in transmitter direction. In the receiver path, data is descrambled after the FEC decoding. Some modifications [16] 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.

TCLink and TX Phase alignment may be implemented in the GBT wrapper if required, see section 8.7.2.2.1

### 8.7.2.2 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 [17], 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 ???. 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 8.58.

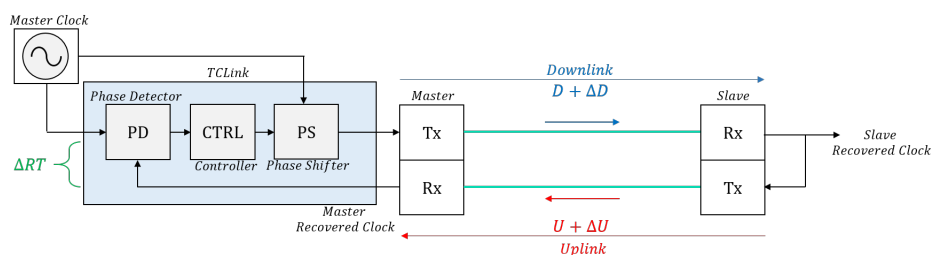


**Figure 8.58:** Integration test between FLX-712 and ATLAS Phase-II Strip Stave.

#### 8.7.2.2.1 TC LINK AND TX PHASE ALIGNMENT

The [https://gitlab.cern.ch/HPTD/tx\\_phase\\_aligner](https://gitlab.cern.ch/HPTD/tx_phase_aligner) will be added in the firmware of the LpGBT TX path in FELIX, to guarantee 1-2 ps fixed latency for the TX side.

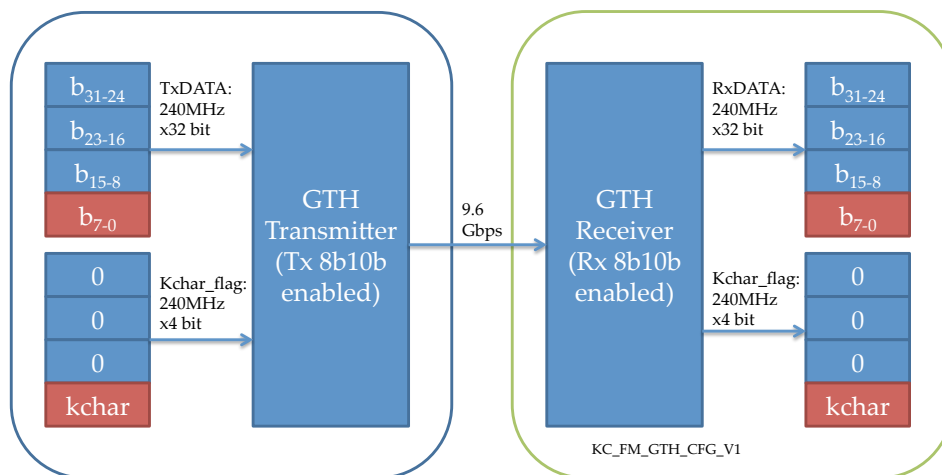
Additionally, TCLink (<https://gitlab.cern.ch/HPTD/tclink>) will be implemented in the LpGBT link wrapper. It will be used to calibrate latency drift (PVT) when communicating with LpGBTx ASIC.



**Figure 8.59:** Simplified block diagram of TCLink.

#### 8.7.2.3 FULL MODE WRAPPER

A wrapper for the Xilinx GTH/GTY 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/GTY will be operated in simplex mode, i.e. transmission (Tx) or reception (Rx). The GTH reference guide [18] gives details about the serializer for Xilinx 7 series FPGA devices. As shown in Figure 8.60, the GTH transmitter and receiver will be operated at  $240\text{ MHz} \times 32\text{-bits}$ . 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 8.60). 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 8.60:** Block diagram for the serializer and deserializer modules for Full mode.

#### 8.7.2.4 64B67B LINK WRAPPER FOR 25G INTERLAKEN

The GTY implementation for 25.78125 Gb/s Interlaken is implemented inside the link wrapper. The FELIX RX link (ToHost / Uplink) is implemented with the following properties:

- RX Data rate 25.78125 Gb/s
- Reference clock 156.25 MHz
- Asynchronous gearbox with bit-slip input
- TX: GBT, equal to the implementation in section 8.7.2.3
- As an option to test FELIX Interlaken in loopback mode, a 25.78125 Gb/s TX link can be implemented with an Interlaken transmitter, see section 8.4.19

### 8.7.3 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 Xilinx 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.

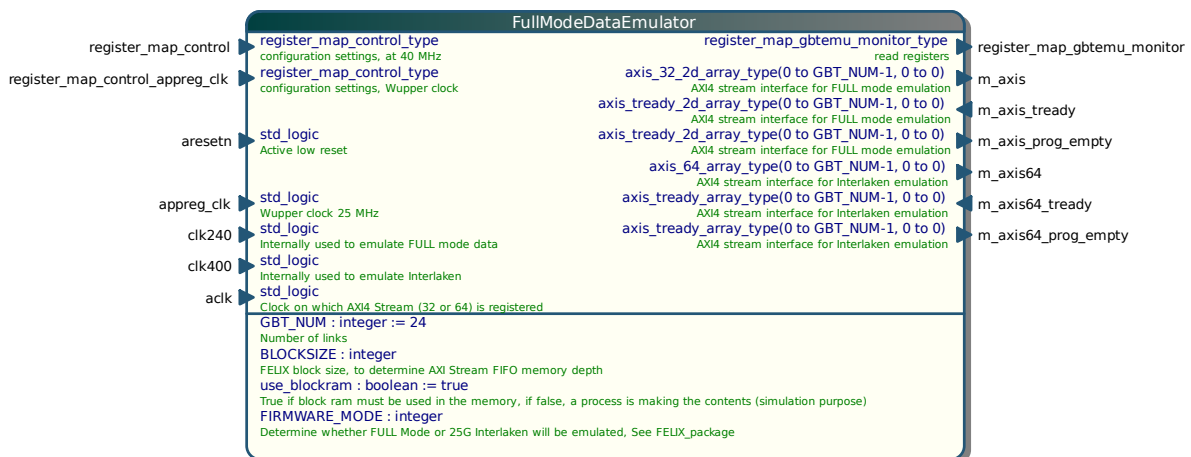
### 8.7.4 STATUS INDICATORS

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.

### 8.7.5 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.





**Figure 8.63:** The FULL Mode and Interlaken data emulator[6].

### 8.8.0.3 FUNCTIONAL DESCRIPTION

The memory blocks in the emulator can be filled at build time using .mem files (generated by elinkconfig) or at runtime through elinkconfig, through the register map. For GBT, LPGBT, PIXEL and STRIP, the contents of the memory are the raw (encoded) data words as seen in a complete E-group. For FULL Mode, the unencoded (32-bit + charisK indication) are loaded into the 33 bits of the memory. For Interlaken the memory is 65 bit wide and contains the AXI4 stream payload + tlast.

A process with a simple counter counts over the address space of the memory blocks and outputs the content on the output. In the FullModeDataEmulator the data is then pushed into AXI4 stream FIFOs, one for every link.

In PIXEL mode, the lpGBT emulator functions similar to the other GBT and lpGBT based modes, but the address counter ends at certain address that makes it possible for the aurora decoder to stay in lock. An extra GBTlinkValid indication is used to tell the decoder when the emulator wraps the address counter, to keep the decoder aligned in emulation mode.

### 8.8.0.4 CONFIGURATION

Register	Description
FE_EMU_ENA.EMU_TOHOST	Enable emulation in ToHost direction.
FE_EMU_ENA.EMU_TOFRONTEND	Enable emulation in ToFrontEnd direction
FE_EMU_CONFIG.WE	7 bits Write enable for the different memory blocks, one per e-group.
FE_EMU_CONFIG.WRADDR	Address of ram block to write
FE_EMU_CONFIG.WRDATA	8 (lpGBT toFE), 16 (GBT), 32 (lpGBT ToHost) or 33 (FULL) bit interface to the memory blocks
SUPER_CHUNK_FACTOR_LINK	For FULL mode only: Concatenate chunks of data together to form superchunks.
GBT_TOHOST_FANOUT.SEL	One bit per link to set the fanout selector into Link mode (0) or Emulator mode (1)
GBT_TOFRONTEND_FANOUT.SEL	One bit per link to set the fanout selector into Link mode (0) or Emulator mode (1)

**Table 8.41:** Configuration registers associated with the GBT, lpGBT and FULL Mode data emulators.

### 8.8.0.5 ESTIMATED RESOURCE USAGE

Resource	Count	% (XKCU115)
----------	-------	-------------

LUTs	555	0.08%
Flip-Flops	62	0.004%
Block RAM	37.5	1.73%

**Table 8.42:** GBT Emulator resources.

Resource	Count	% (XKCU115)
LUTs	677	0.1%
Flip-Flops	70	0.005%
Block RAM	101.5	4.70%

**Table 8.43:** IpGBT ToHost Emulator resources.

Resource	Count	% (XKCU115)
LUTs	107	0.01%
Flip-Flops	43	0.003%
Block RAM	16	0.74%

**Table 8.44:** IpGBT ToFrontEnd Emulator resources.

Resource	Count	% (XKCU115)
LUTs	2912	0.43%
Flip-Flops	4554	0.34%
Block RAM	26	1.20%

**Table 8.45:** FULL Mode Emulator resources.



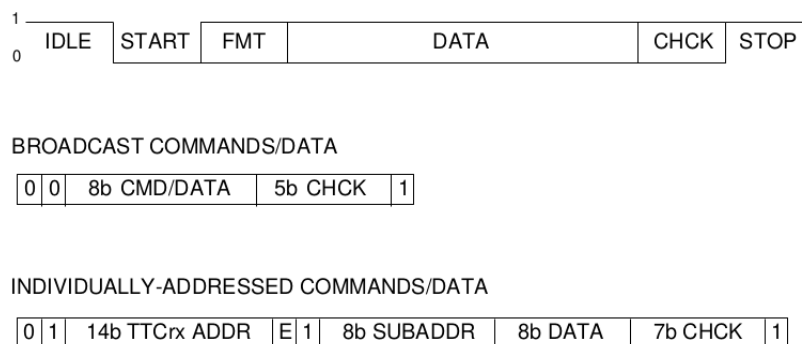
## 8.9 TTC EMULATOR

### 8.9.1 INTRODUCTION

The TTC Emulator block mimics the behavior of the actual TTCrx IC. When enabled, it can generate the level 1 trigger accept signal (L1A), the event counter reset (ECR) and bunch counter reset (BCR) signals, respecting their different modes and delays. The support for short-B and long-B channel serial signal generation is also included, in addition to an extra orbit counter reset (OCR) signal, added to the TTCrx IC functionalities.

### 8.9.2 INTERFACES

Two Time Division Multiplexed (TDM) channels are used. The first, channel A, is exclusively dedicated to broadcast the first-level trigger-accept (L1A) decisions, delivering a one-bit decision for every bunch crossing. The other, channel B, is used to broadcast data to all or specific system destinations. The used format is illustrated in Figure 8.64. Each frame is identified by a header bit (FMT) that indicates its type. Start (logical



**Figure 8.64:** Transmission Frame Format.

"0") and stop (logical "1") bits are always included at the beginning and end of the frame transmission to facilitate correct synchronisation.

The check part (CHCK) implements standard Hamming code on with one additional even parity bit to detect double bit errors. This makes 5 bits and 7 bits for the broadcast and individually-addressed command frames.

### 8.9.3 FUNCTIONAL DESCRIPTION

To process the broadcast command frames, the emulator verifies the value of the incoming frame CHCK and then executes the operation requested in the data part of the frame (8 bits denoted user and system bits: uu ssssss). The table below summarizes the implemented broadcast commands.

Command	Format uu ssssss	Function
NOP	uu sss000	Do nothing
BCRST	uu sss001	Bunch counter reset
ECRST	uu sss010	Event counter reset
BECRST	uu sss011	Reset event and bunch counters
OCR	uu sss100	Orbit counter reset
OBCRST	uu sss101	Reset orbit and bunch counters
OECRST	uu sss110	Reset orbit and event counters
OBECRST	uu sss111	Reset orbit and event and bunch counters

The different reset command signals are generated upon the detection of the rising edge in the incoming broadcast frame corresponding bit. while OCR is generated only once, BCR and ECR can be repeated with

delays determined by the value of bits<3:0> of the coarse delay register. Nevertheless, single signals shall be generated if the corresponding control period is zero (control registers TTC\_EMU\_ECR\_PERIOD and TTC\_EMU\_BCR\_PERIOD).

Individually-Addressed Commands (IAC) are sent to specific chips with identification number (ID). The net data contained in the IAC packet amounts to 16 bits. It is divided into an 8-bit DATA byte, and an 8-bit SUBADDR byte. IACs can be used to write internal registers of the TTCrx and execute internal commands. One bit in the IAC data frame (the  $\hat{\text{AIJE}}$  bit in Figure 1) signals if the command is internal or external. The Emulator completes the IAC frame, by adding the Hamming code and transmits it through the B-channel serial signal output.

Receiving L1Accept signal on channel-A, (in TTC\_EMU\_CONTROL.L1A), the emulator activates the l1\_ - accept output after a delay specified by the lower four bits of the Coarse Delay Register. If TTC\_EMU\_ - L1A\_PERIOD is different from zero, a periodic trigger L1A signal is generated giving an adjustable frequency trigger signal.

## 8.9.4 CONFIGURATION

The emulator can be enabled and disabled on the fly. IN fact, TTC\_EMU\_SEL selects the TTC Source: When set to '0', the TTC data comes from the decoder, when set to '1', the TTC data comes from the TTC emulator. TTC\_EMU\_ENA starts the emulator. When set to '0' the emulator does not produce any data. When set to '1' the emulator is running. The variables TTC\_EMU\_SEL and TTC\_EMU\_ENA are both controlled with the command 'fttcemu -e' (setting both parameters to '1') and 'fttcemu -n'(setting both parameters to '0').

As the TTC emulator is able to generate periodic L1A, ECR and BCR signals, TTC\_EMU\_L1A\_PERIOD is the L1A period in units of LHC clock period (25 ns) set by the user as a frequency using option -e. TTC\_EMU\_ - BCR\_PERIOD is the BCR period in units of LHC clocks and by default has a value 3564 which is the default in the LHC experiments (representing a period of roughly 89.1 microsecond). TT\_EMU\_ECR\_PERIOD is the ECR period in units LHC clocks, but note that the 'fttcemu' tool sets the ECR period in units of milliseconds. Here are few examples:

Set an L1A frequency of 1000 Hz an ECR period of 1 second:

```
fttcemu -f 1000 -E 1000
```

Generate a single ECR and a BCR:

```
fttcemu -E 0 -B 0
```

Generate a single ECR, followed by 10 L1A triggers at 10 Hz, then switch to 1000 Hz L1A:

```
fttcemu -E 0 -L 10 -t 100000 -f 1000
```

## 8.9.5 STATUS INDICATORS

The status of the TTC emulator is shown running the command ./fttcemu, which displays the values of the various TTC emulator parameters. This is an example of what is displayed:

```

$ fttcemu
Status:
TTC_EMU_SEL=0, TTC_EMU_ENA=0
TTC_EMU_BCR_PERIOD=3564
TTC_EMU_ECR_PERIOD=0
TTC_EMU_L1A_PERIOD=0

```

## 8.9.6 ERROR HANDLING

In broadcast command frames, error correction and detection is made on these eight data bits. The emulator computes the Hamming bits corresponding to the value of TTC-EMU-LONG-CHANNEL-DATA register in order to complete the individually addressed frame that shall be written in a fifo and then transmitted through the B channel serial signal output.

## 8.9.7 ESTIMATED RESOURCE USAGE

The FPGA resource utilization for an Ultrascale FPGA is reported in Table 8.47.

**Table 8.46:** Post-synthesis TTC emulator resources for FLX712. .

Entity	Total LUTs	Logic LUTs	LUT Regs	FFs	Carry8	Muxes
TTC emulator	533	510	23	707	26	5

## 8.10 LEGACY TTC DECODER

The TTC system prior to the Run 4 transmitted 2 bits each bunch crossing (LHC) clock. The first bit is the Level-1 accept decision (aka A-channel) and the second bit (B-channel) is interpreted as short and long commands transmitted serially. The 40.08 MHz LHC clock is also recovered from the TTC bitstream. The TTC signal is transmitted via multi-mode optical fiber with ST connectors.

Decoding the the TTC serial stream as the L1A and other signals is done in FELIX firmware. The firmware receives a 160 Mb/s bitstream accompanied by a 160 MHz clock from a clock-data recovery IC, ADN2814. The TTC bitstream is biphas mask encoded so A- and B- channel are sampled as two bits each. The A- and B-channelss can be destinguished without ambiguities because the B-channel can not have eleven consequent 0's while A-channel is mostly 0 (the Level-1 trigger rate is 100 kHz). The FPGA is also receiving LOL, loss off lock, and LOS, loss is signal, from ADN2814. LOS is issued if ADN2814 is not receiving proper signal from the photo-diode. LOL is issued if the IC can not recover the 160 MHz clock. The entire TTC decoding firmware can be reset with TTC\_DEC\_CTRL.TOHOST\_RST register.

The B-channel is a serial data stream with three types of commands:

- idle is 111111111111.
- short, broadcast, command is 16-bit long: 00TTDDDDDEBHHHHH1 ( D=Brcst[7-2], 6 bits. E=Event Counter Reset, 1 bit. B=Bunch Counter Reset, 1 bit. H=Hamming Code, 5 bits).
- long, addressed, command is 42-bit long: 01AAAAAAAAAAAAAAE1SSSSSSSSDDDDDDDDHHHHHHH1 ( A=Address, 14 bits. E=Internal(0)/External(1), 1 bit. S=SubAddress, 8 bits. D=Data, 8 bits. H=Hamming Code, 7 bits).

Transmission of the short commands happens in sync with the LHC beam structure. Errors from the Hamming code are in TTC\_DEC\_MON.TTC\_BIT\_ERR register.

Every clock cycle the TTC firmware in FELIX outputs L1A, B-channel, BCR (Bunch Counter Reset), ECR (Event Counter Reset), and Brcst[2-7] signals to the TTC downlinks (2-, 4-, and 8- bit e-links) with fixed latency. The e-link data formats are in Table 8.39, part of Section 8.5.14, (see also Table 8.37). All these beam-synchronous signals are dedecoded from the B-channel bitstream. All these B-channel-reled signals can be delayed. The delay value in LHC clock cycles can be configured with TTC\_DEC\_CTRL.B\_CHAN\_DELAY register. The ECR and BCR signals can be swapped via TTC\_DEC\_CTRL.ECR\_BCR\_SWAP register; this is needed for LAr calorimeter systems.

The TTC decoder also outputs 27-byte messages to host for every level-1 accept. The messages include BCID(12), XL1ID(8), L1ID(24), orbit(32), Trigger Type(16), and L0ID(32). The message format is shown in Fig. 8.65 (see also Table B.9 and Sec. 8.4.17). Status of of the the FIFO with the to-host data waiting for the

0	FMT(8)	Len(8) = 20	<i>reserved</i>	BCID(12)
1	XL1ID(8)	L1ID(24)		
2	orbit(32)			
3	Trigger Type (16)		<i>reserved(16)</i>	
4	L0ID(32)			

L1Ainfo\_v01

**Figure 8.65:** The TTC message sent to the Back end software (20 bytes) presented as five 32-bit words.

trigger type is accessible with TTC\_DEC\_MON.TH\_FF\_FULL, TTC\_DEC\_MON.TH\_FF\_EMPTY, and TTC\_DEC\_MON.TH\_FF\_COUNT registers.

BCID is a 12-bit bunch crossing counter. It is incremented every clock cycle of the LHC clock. It is reset on arrival of the BCR pulse to a programmable offset. The offset is configured via TTC\_DEC\_CTRL.BCID\_ONBCR

register. The counter values range from 0 to 3563. The BCR signal is expected to arrive with fixed latency in respect to the LHC bunch structure about once an orbit.

The BCR periodicity is checked whether BCR period is 3564 BCs. Every mismatch of the BCR period is recorded with `TTC_BCR_PERIODICITY_MONITOR.VALUE` 32-bit counter register. The counter can be reset with `TTC_BCR_PERIODICITY_MONITOR.CLEAR` register.

The 32-bit orbit counter increments when the BCID counter reaches 3563. In order to reset Orbit you have to send `Brcst[7]` and `BCR`. `Brcst[7]` has to be in the same short word or before the `BCR`.

The 24-bit L1ID counter is incremented on every L1A and reset with `ECR`.

The 8-bit XL1ID counts ECRs. It is set to the `TTC_DEC_CTRL.XL1ID_SW` register on the raising edge of `TTC_DEC_CTRL.XL1ID_RST`. The ECRs are also counted in `TTC_ECR_MONITOR.VALUE` 32-bit register.

The L0ID is a copy of XL1ID and L1ID. The current value of L0ID is stored in `TTC_L1ID_MONITOR` register. The counter is reset with `TTC_ECR_MONITOR.CLEAR`.

The firmware can be configured to read trigger types from long b-channel commands by setting `TTC_DEC_CTRL.TT_Bch_En` register to 1. Then a trigger type frame will be expected for every L1A. Trigger type is set to 0x0000 if reading of the b-channel is disabled or if a trigger types frame is not transmitted shortly after a L1A (within 25\*500 ns). Long commands with trigger types are counted with `TTC_TTYPE_MONITOR.VALUE` 32-bit register. The counter is reset with `TTC_TTYPE_MONITOR.CLEAR`.

FELIX outputs a **BUSY** signal via the Lemo connector with open collector output. Assertion of a BUSY signal tells the Central Trigger Processor to throttle the Level-1 trigger to stop the data flow from the front-end systems to FELIX. The front-end systems can set "BUSY-ON" and "BUSY-OFF" requests to FELIX. Also, BUSY requests can come from host via `TTC_DEC_CTRL.MASTER_BUSY` register. The BUSY output signal is a logical OR of BUSY signals from host and individual (lp)GBT links. A busy signal for an (lp)GBT link is also a logical OR of busy signals from the individual e-links. BUSY signals from individual e-links can be suppressed via `ELINK_BUSY_ENABLE` register. The BUSY status of the FELIX board can be read via `TTC_DEC_CTRL.BUSY_OUTPUT_STATUS`. BUSY status of a GBT/lpGBT link can be monitored as a bit in `TTC_BUSY_ACCEPTED` register. BUSY from DMA can be ignored via `DMA_BUSY_STATUS.ENABLE`. `TTC_BUSY_TIMING_CTRL.LIMIT_TIME` sets minimum time interval for a BUSY signal from an (lp)GBT link to produce global BUSY. `TTC_BUSY_TIMING_CTRL.BUSYWIDTH` extends the output BUSY pulse. `TTC_BUSY_CLEAR` resets BUSY condition on all links.

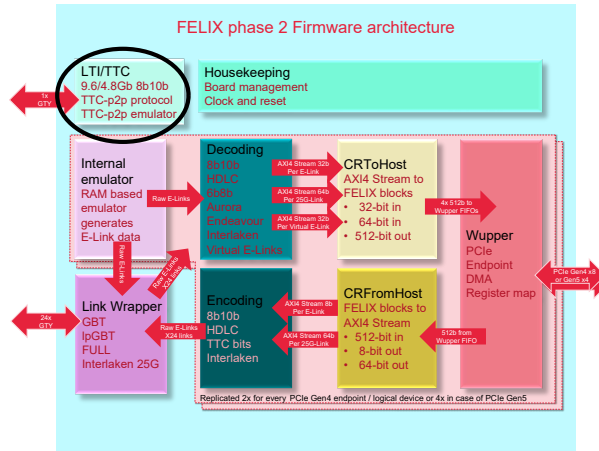
FELIX can control data flow from fullmode links by issuing XOFF and XON commands to the transmitting systems. An XOFF is transmitted when a buffer in FELIX is about to overflow. After transmitting an XOFF FELIX will transmit an XON to resume the dataflow.

The FPGA resource utilization for an Ultrascale FPGA is reported in Table 8.47.

**Table 8.47:** The TTC resources are post-implementation (place and rout) for FLX712. We expect similar resource utilization in Versal FPGAs..

Entity	Total LUTs	Logic LUTs	LUTRAMs	SRLs	FFs	RAMB36	RAMB18
ttc decoder	1005	981	0	24	1529	7	2
busy	1652	1652	0	0	1363	0	0

## 8.11 LTI/TTC INTERFACE



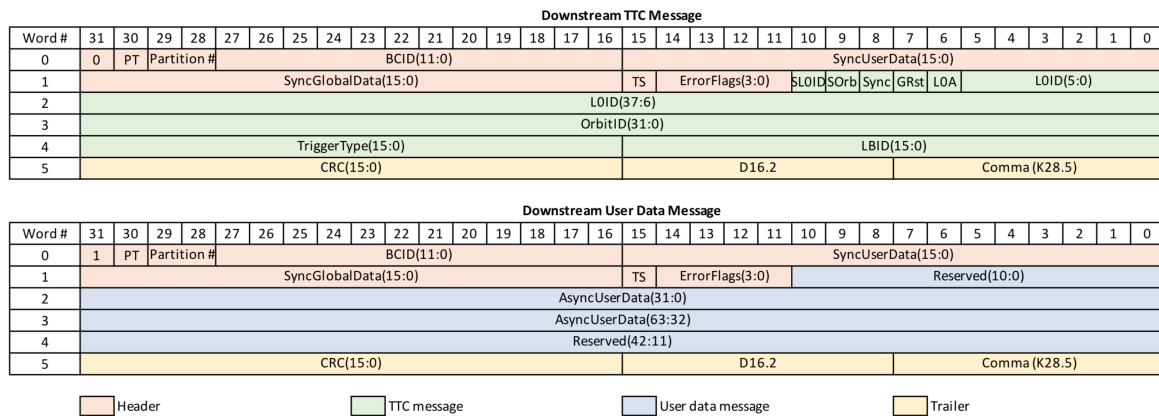
**Figure 8.66:** The LTI-TTC interface in the toplevel diagram.

Local Trigger Interface (LTI) modules [14] will distribute Trigger, Timing, and Control (TTC) signals to FELIX during the HL-LHC operations (see table 2.2 of [15]). FELIX will output BUSY and other signals to LTIs. The TTC system will be upgraded to use the LTI boards to handle the Level-0 trigger rate of 1 MHz. The low bandwidth of the legacy TTC system (40 Mbps for the B-channel) does not allow it to transmit trigger types at 1 MHz rate. LTI boards use point to point connections to FELIX for low and deterministic latency of the uplink (e.g. busy) signals. The downlink (from LTI to FELIX) is 8b/10b encoded and it operates at 9.6 Gb/s. The uplink is also 8b/10b encoded and it runs at 4.8 Gb/s. Both the uplink and downlink data blocks are synchronous with the LHC clock (40 MHz) (see Figs. 8.67 and 8.68).

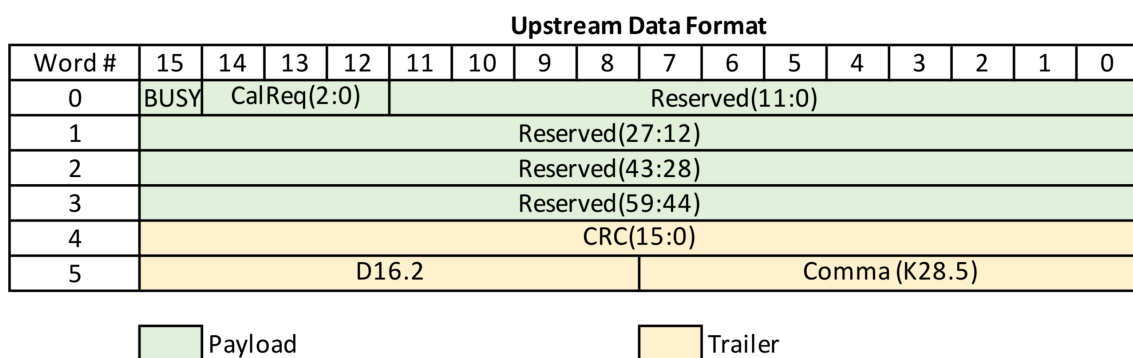
the downlinks provides user or trigger signals every cycle of the 40 MHz clock as shown in Fig. 8.67. The BCID, L0ID, OrbitID, TriggerType, and LBID are output to host upon reception of Level-0 Accept (LOA). All these signals are associated with the level-0 accept decision. The BCR signal for the front-end system is derived using the Turn Signal (TS) which has fixed latency in respect to the LHC bunch structure. The phase of the Turn Signal in respect to the bunch structure is independent of the trigger latency. The BCID counter in FELIX is set to a configurable value upon reception of the TS pulse. The BCR pulse is generated when the counter reaches zero. The BCID counter is used for incrementing the local OrbitID. Global Reser (GRst), LOA, Set L0ID, sSet OrbitID, and other signal will also be passed to the front-end systems. The LTI protocol offers 16 bits for synchronous signals (e.g. to run calibrations) and 64 bits for various asynchronous signals. This provides flexibility to FELIX firmware to meet the requirements of the front-end systems.

The uplink informs LTI of busy signals from FELIX as shown in Fig. 8.68. The uplink will transmit the global BUSY status and Calibration Requests (CalRec) from FELIX to LTIs. The data format has 60 reserved bits in case the front-end system requires additional signals.

Encoding and decoding of the 8b/10b signals is handled in the multi-gigabit transceiver (MGT). The firmware assembles the downlink messages from 32-bit words from the MGT by identifying D16.2 and K28.5 characters. It also checks the data for bit-errors with the 16-bit CRC field. The uplink TTC messages are sampled into 32-bit words to match the MGT interface. The clock frequencies are 200 MHz or lower. The FPGA resource utilization is insignificant.



**Figure 8.67:** The TTC message sent from the LTI to FELIX (32 bytes) presented as six 32-bit words.



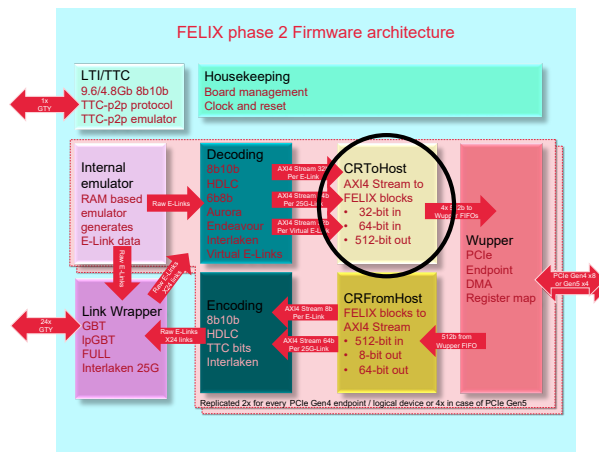
**Figure 8.68:** The TTC message sent from FELIX to the LTI (12 bytes) presented as six 16-bit words.

## 8.12 CRToHost: ToHost OR UPSTREAM CENTRAL ROUTER

### 8.12.1 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 IpGBT link. For FULL mode the size of the second dimension is 1.

The data is demultiplexed, buffered and formatted into a 256b, 512b or 1024b FIFO interface that is acceptable for the Wupper ToHost DMA interface.



**Figure 8.69:** The ToHost Central Router (CRToHost) in the toplevel diagram.



## 8.12.2 INTERFACES

### 8.12.2.1 OVERVIEW

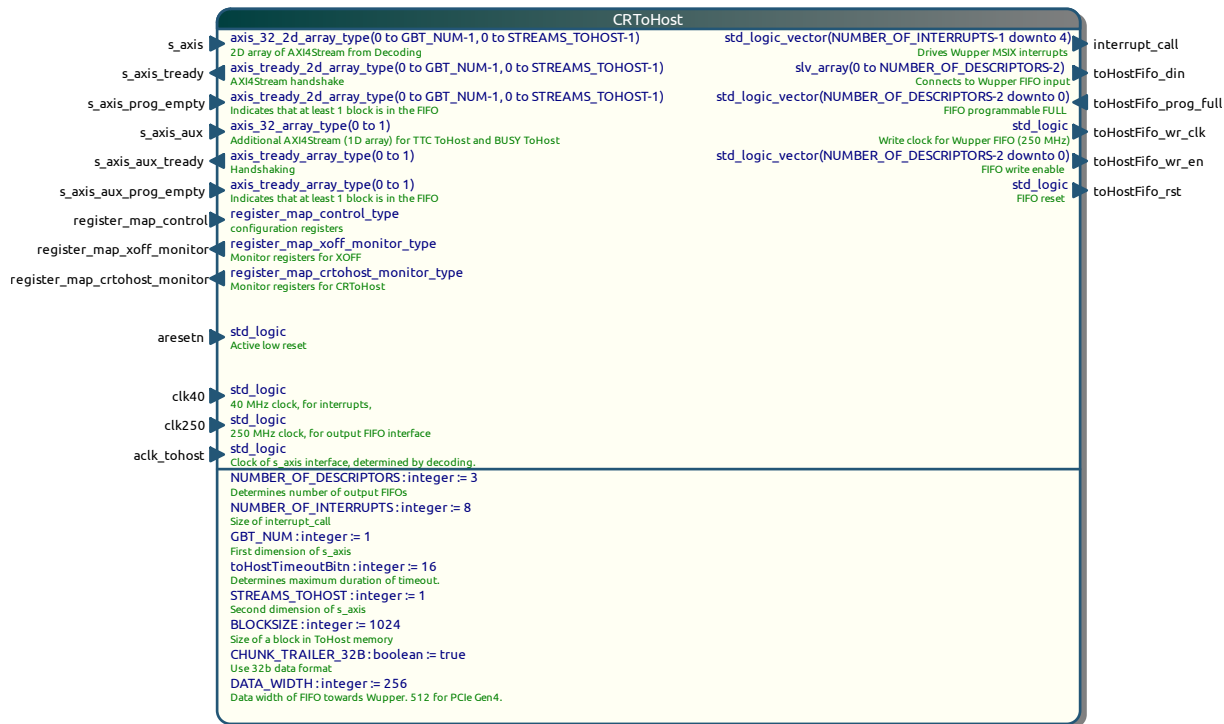


Figure 8.70: CRToHost interface symbol.

#### 8.12.2.2 INTERFACE FROM DECODING

The interface `s_axis` of the type `axis_32_2d_array_type` (a 2D array of `axis_32_type`), see listing 8.3. 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` 8.4.17 and `BUSYXOFF` 8.4.18.

```

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 :

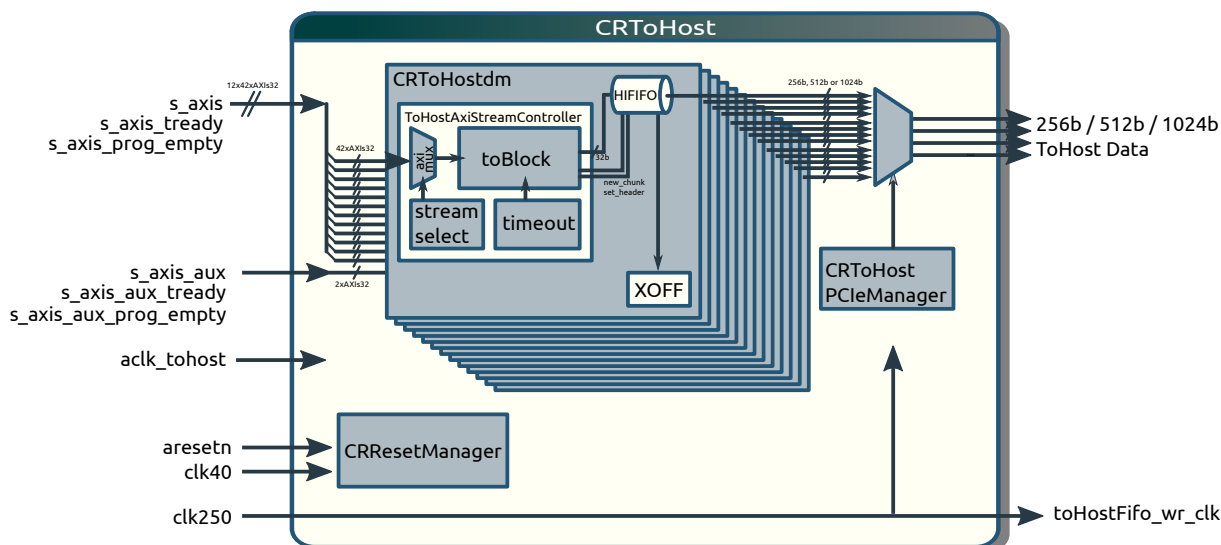
3: Truncation/FIFO full

```

```
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 <>)
```

### 8.12.2.3 INTERFACE TO WUPPER

### 8.12.3 FUNCTIONAL DESCRIPTION



**Figure 8.71: CRTToHost Block Schematic.**

#### 8.12.3.1 CRTToHostTDM

For every item in the first dimension of `s_axis` (GBT\_NUM), usually the number of optical links, one `CRTToHostdm` is instantiated. This is a wrapper for the `ToHostAxisStreamController`, the `Channel FIFO` and the `XOFF` mechanism.

### 8.12.3.1.1 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.

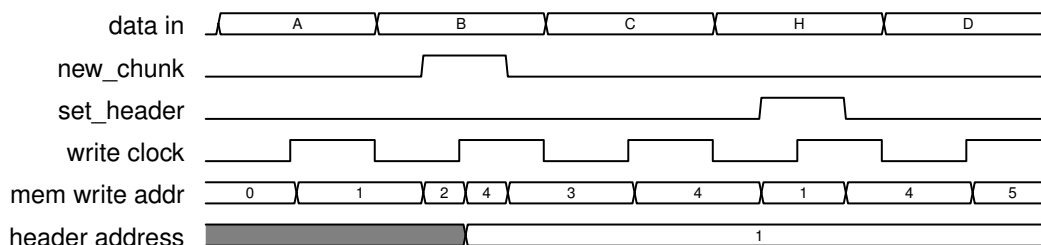
- **Stream Select** looks at the s\_axis\_prog\_empty bits which show whether enough data resides in one of the AXI stream FIFOs, inside the decoders. As a second priority, a stream can be selected that has tvalid set to '1', after a timeout occurs. This way a partial block can be read out.
- **AXI Stream 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 selected 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.

### 8.12.3.1.2 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.

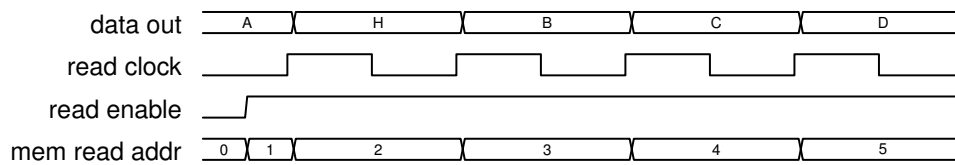
This FIFO is replaced by a Header-Inserting FIFO (HIFIFO) from firmware version 5.2. This is a special FIFO with two extra inputs that can be used to insert headers into its memory. This is necessary to be able to generate chunks with headers, because the length of the chunk is not yet known when the start of the chunk needs to be inserted into the FIFO.

The HIFIFO has two additional inputs: new\_chunk and set\_header. When new\_chunk is asserted, the FIFO will leave a gap in its memory which can be written to at a later time. The word written to the FIFO when new\_chunk is high is placed *after* the reserved gap. This gap can be written to by asserting set\_header. When set\_header is asserted, the word on the input will be written to the gap that was reserved when new\_chunk was high. Figures 8.72 and 8.73 show a write and read sequence to and from the HIFIFO. The FIFO is first-word fall-through, which is why the first word is already at the output when the read enable pin is asserted in fig. 8.73.

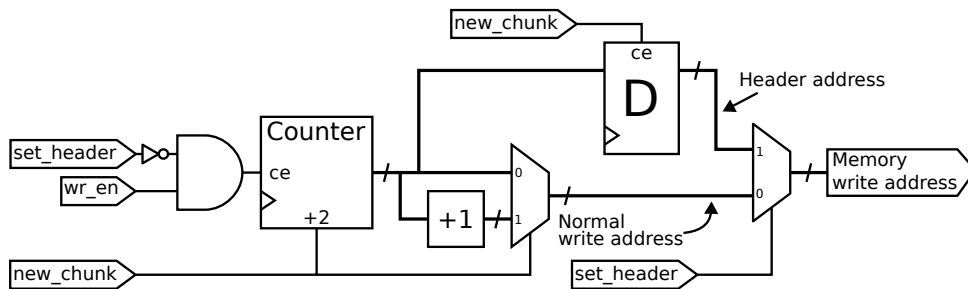


**Figure 8.72:** The process of writing data into the HIFIFO.

The logic that handles the reservation and writing of the header word is shown in fig. 8.74

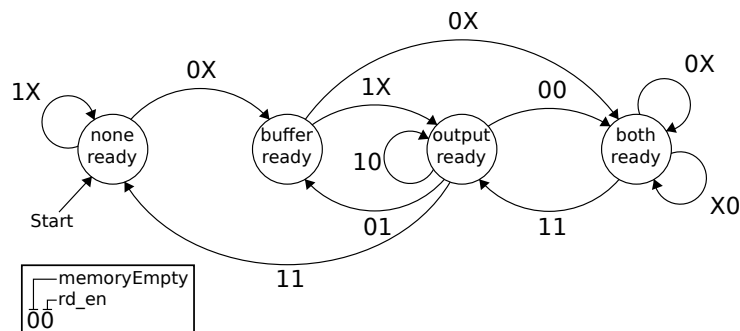


**Figure 8.73:** The process of reading data from the HIFIFO that was written in fig. 8.72.

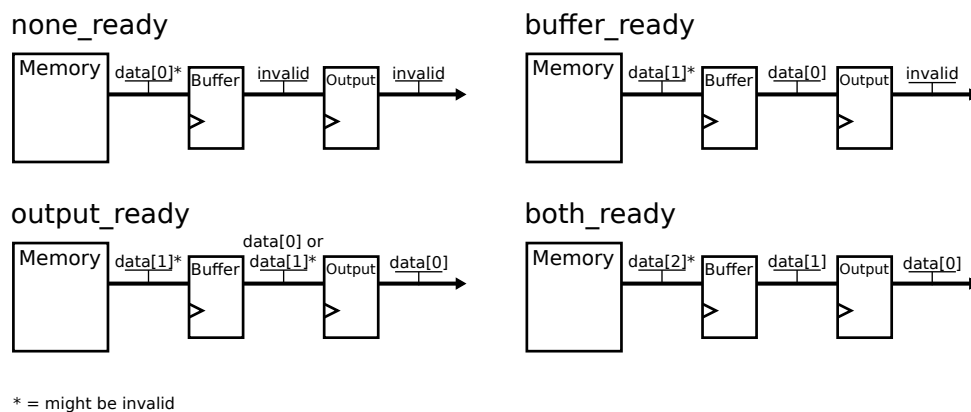


**Figure 8.74:** A slightly simplified version of the writing logic of the HIFIFO.

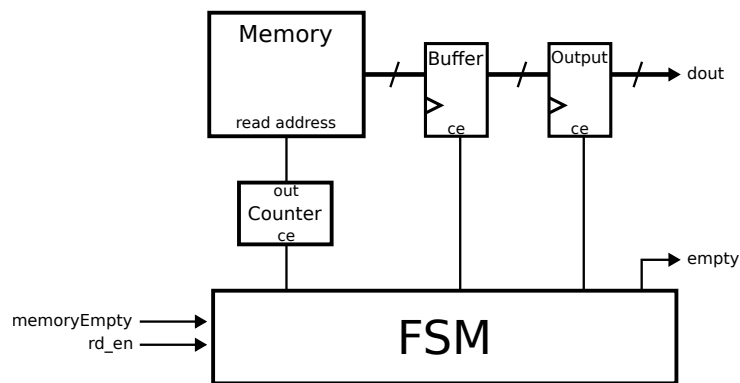
To conform to timing requirements, the HIFIFO has a state machine on its output. This state machine controls the two output registers of the HIFIFO's memory block. It prepares the first two output words into these registers when they are ready. The state machine diagram is shown in fig. 8.75. As shown in fig. 8.77, the state machine controls four signals: the empty output, the clock enable of the first and second registers (separately), and the clock enable of the HIFIFO's read pointer counter.



**Figure 8.75:** The state machine diagram for the state machine used by the HIFIFO.



**Figure 8.76:** An explanation of every state of the state machine at the output of the HIFIFO.



**Figure 8.77:** The signals the state machine controls.

More details about the exact workings of the HIFIFO can be found in its firmware code. A report was written on the design of the HIFIFO, which can be found in [19].

### 8.12.3.2 CRTToHost PCIEMANAGER

The CRTToHost 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. Additionally, it reads the AXI Stream ID from the first cycle of the data block, and associates the AXI stream ID with one of the output descriptors / DMA channels towards Wupper. If the target Wupper FIFO has empty space to store that block, the thch\_sel and output\_select signals for the CRTToHost MUX are set and the read enable for the channel FIFO is asserted, as well as the write enable for the Wupper FIFO will be asserted for exactly the number of cycles needed for one block.

#### 8.12.3.2.1 PCIe DMA CHANNEL SELECTION

The CRTToHost PCIeManager and the CRTToHost MUX work together to write the correct data, coming from a certain E-Link (identified by an AXI Stream ID) to a user selectable DMA channel.

1. When data is available in the Channel FIFO (first-word-fall-through mode), the 11 bits of the AXI-Stream ID are read out.
2. A block memory is addressed with the AXI-Stream ID as the address. For every ID, a 3-bit target DMA channel (descriptor) can be programmed by means of the registers mentioned in the Configuration subsection (8.12.4). By default, all data will be forwarded to descriptor 0, standard builds have 4 ToHost descriptors per PCIe Endpoint.
3. When the AXI Stream ID has been associated with a DMA descriptor, the CRTToHost input and output selection will be set up to connect the selected input channel with the associated Wupper FIFO.

The mechanism described above benefits reliability in two ways:

- **CPU load:** The data load can be separated over multiple DMA buffers in a configurable way per E-Link. This means that when one link is expected to produce more data than the other, this can be accounted for in the DMA channel assignment in the firmware. This way the load of the CPU cores can be balanced.
- **Isolation of (DCS) data streams:** If high link occupancy is likely to cause buffer overload in the server memory, certain (DCS) E-Links may be assigned to a separate DMA channel / descriptor. This way a separate process will be available to handle important data independent of other data acquisition processes.

### 8.12.3.3 CRToHost MUX

The CRToHost MUX is selected by the CRToHost PCIe manager and multiplexes the number of input channels (GBT\_NUM+1 for the AUX channel) into one of the FIFO data ports towards Wupper.

### 8.12.3.4 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 ToHostAxisStreamController, 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.

## 8.12.4 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 assertion 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.
- CRTOHOST\_DMA\_DESCRIPTOR\_2.AXIS\_ID: 11 bit AXI Stream ID of the E-Link to be associated with a DMA stream
- CRTOHOST\_DMA\_DESCRIPTOR\_1.DESCR: DMA channel (descriptor) 0-3 to be associated with the AXI stream ID
- CRTOHOST\_DMA\_DESCRIPTOR\_2.DESCR\_READ: Register to read back the DMA channel (descriptor) associated with the AXI Stream ID.

## 8.12.5 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.

## 8.12.6 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 CRTToHost MUX if other AXI Streams or other channels are in the process of transferring data.

## 8.12.7 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.

## 8.12.8 ESTIMATED RESOURCE USAGE

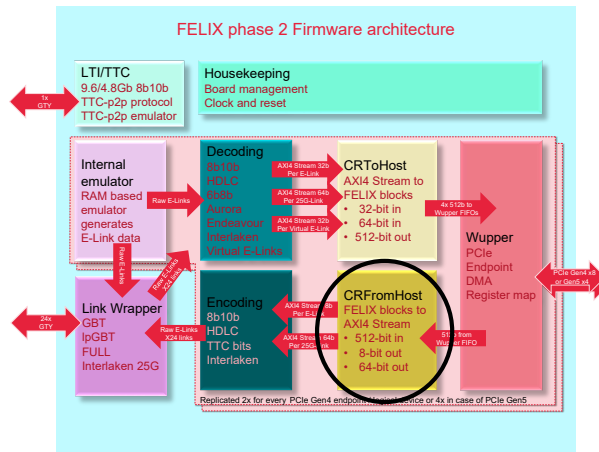
	LUT		FF		BRAM	
KCU115 / FLX712	10406	1.56%	11713	0.88%	52	2.4%
VU37P / FLX128	7453	0.57%	7643	0.29%	104	5.15%

**Table 8.48:** CRTToHost Resource utilization.

## 8.13 CRFROMHOST: FROMHOST OR DOWNSTREAM CENTRAL ROUTER

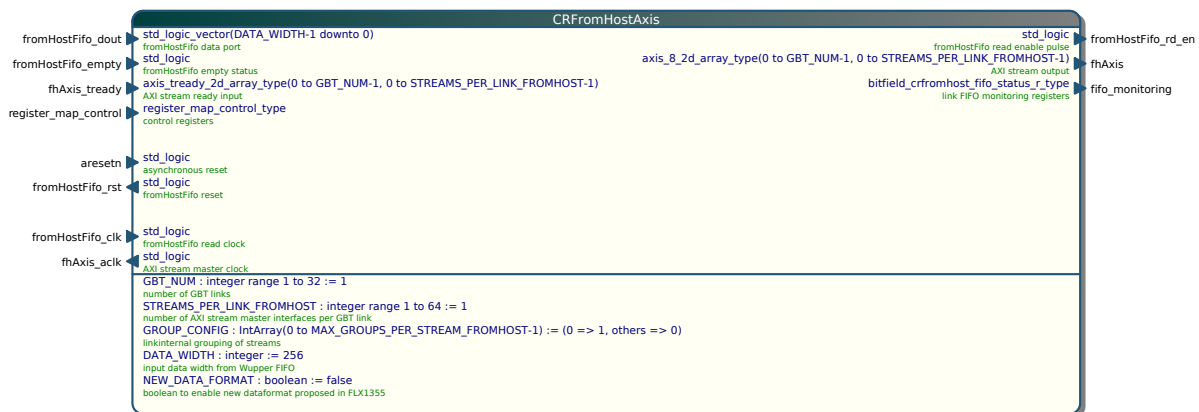
### 8.13.1 INTRODUCTION

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.



**Figure 8.78:** The FromHost Central Router (CRFromHost) in the toplevel diagram.

### 8.13.2 INTERFACES

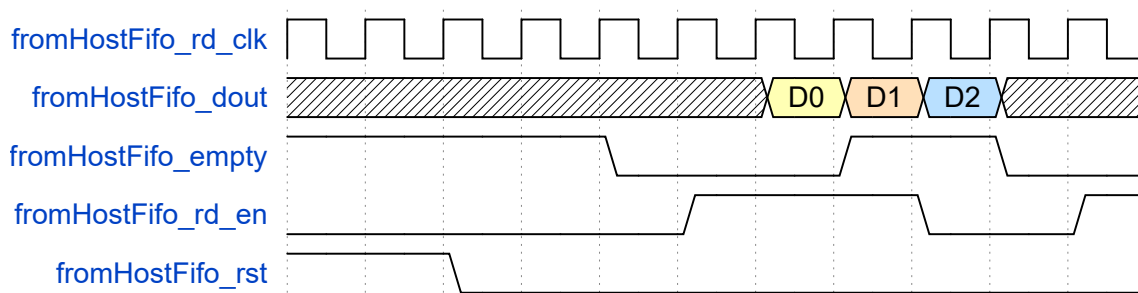


**Figure 8.79:** The FromHost or Downstream Central Router entity.

#### 8.13.2.1 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 8.80 shows an example waveform of input signals for the CRFromHost.



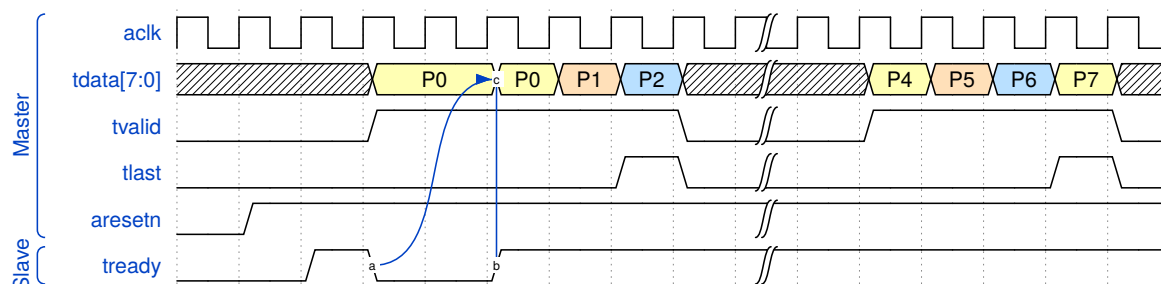


**Figure 8.80:** Example waveform of a typical FromHost Central Router transfer with its FIFO interface. [7].

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 ?? shows how the bits are assigned in that packet. Details of the data format can be found in B.2.2.

### 8.13.2.2 INTERFACE TO THE ENCODERS

All encoders are connected to the FromHost Central Router as AXI stream 8b slaves. Therefore, the CR-FromHost 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 8.81.



**Figure 8.81:** Example waveform of a typical AXI stream 8b transfer. [7].

## 8.13.3 FUNCTIONAL DESCRIPTION

### 8.13.3.1 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.

### 8.13.3.2 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.

### 8.13.3.3 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.

## 8.13.4 CONFIGURATION

### 8.13.4.1 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.

### 8.13.4.2 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.

## 8.13.5 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.

## 8.13.6 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.

## 8.13.7 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 8.49:** CRFromHost Resource utilization.

## 8.14 WUPPER: PCIe DMA CORE AND REGISTER MAP

### 8.14.1 INTRODUCTION

**Wupper**<sup>9</sup> is designed for the ATLAS / FELIX project [20], 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 [21] 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) [22, 23, 24, 25].

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\_DESCRIPTOR 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/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.

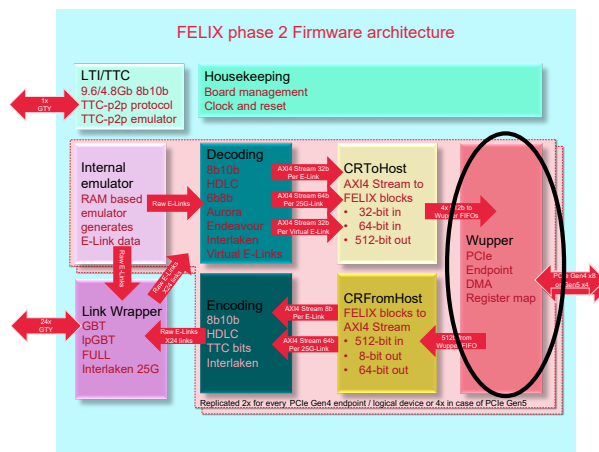


Figure 8.82: Wupper in the toplevel diagram.

<sup>9</sup>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%B6llleges#Bongelwuppen](https://nds-nl.wikipedia.org/wiki/Nedersaksische_sp%C3%B6llleges#Bongelwuppen)



			709 : VC709 710 : HTG710 711 : BNL711 v1.5 712 : BNL712 800 : Xupp3r VU9P 801 : BNL801 VU9P 128 : VCU128 155 : FLX155 180 : VMK180 181 : FLX181 182 : FLX182
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)	(others => '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	Implement 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 8.50:** Wupper Generics.

### 8.14.2.2 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.

### 8.14.2.3 TOHOSTFIFO

The ToHostFifo interface connects the ToHostFifos input ports (The number of FIFOs is determined by NUMBER\_OF\_DESCRIPTOR-1, see section 8.14.4) 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.

#### 8.14.2.4 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 8.14.7

#### 8.14.2.5 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.

#### 8.14.2.6 BUSY

- master\_busy\_in : Used in the interrupt controller, see section 8.14.7
- 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.

#### 8.14.2.7 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.

### 8.14.2.8 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

### 8.14.3 FUNCTIONAL DESCRIPTION

Xilinx has introduced the AXI4-Stream interface [21] for the PCIe EndPoint core: a simplified version of the ARM AMBA AXI bus [26]. 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 8.84 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 8.14.5).

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

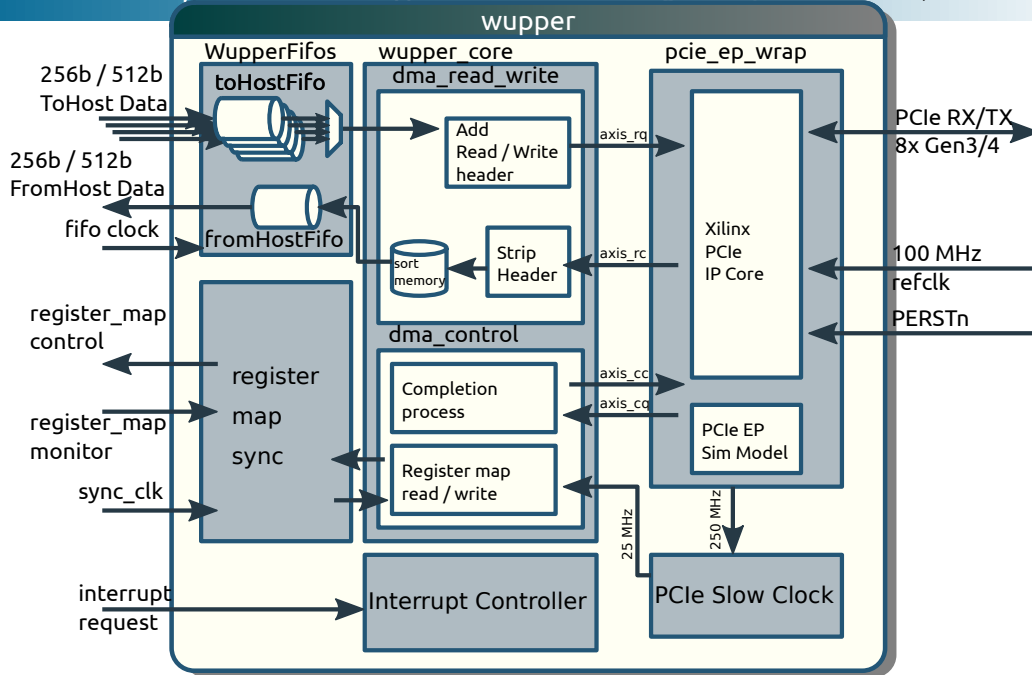
#### 2. DMA Read Write:

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 8.84:** Structure of the Felix PCIe Engine.

Figure 8.84 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.

#### 8.14.4 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 8.51 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_DESCRIPTOR`s. The descriptor at `NUMBER_OF_DESCRIPTOR`s-1 is the FromHost descriptor which always has the `READ_WRITE` bitfield set to 1 (FROMHOST) and the descriptors 0 to `NUMBER_OF_DESCRIPTOR`s-2 are implemented as ToHost descriptors. An additional special FromHost descriptor is implemented at `NUMBER_OF_DESCRIPTOR`s, this is the so called trickle descriptor (see ??) which is similar to the other FromHost descriptor, but it ignores the `pc_pointer`. The number of ToHost FIFOs is automatically determined by the same generic, as well as the ToHost FIFO depth. Setting `NUMBER_OF_DESCRIPTOR`s to 6 (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			
	PC_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 8.51:** DMA descriptors types.

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

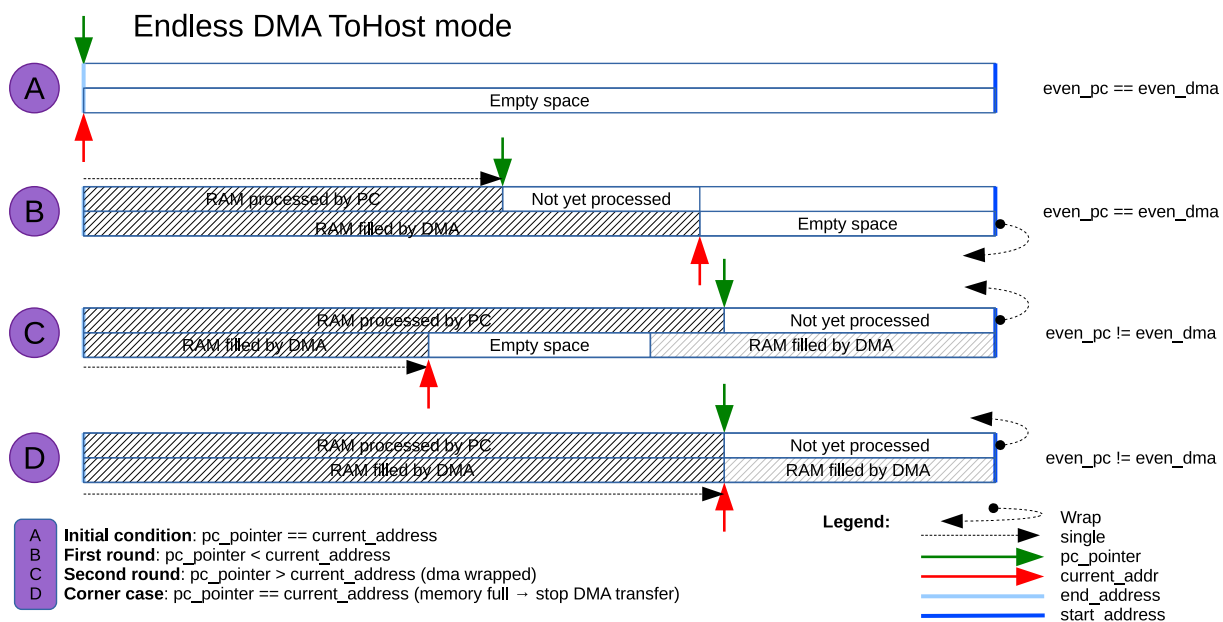
- **DMA\_DESC:** the register containing the start (*start\_address*) and the end (*end\_address*) memory addresses of a DMA transfer; both handled by the server (software API).
- **DMA\_DESC\_a:** integrates the information above by adding (i) the status of the read pointer on the server side (*pc\_pointer*), (ii) the wrap around functionality enabling (*wrap\_around*, see Section 8.14.5 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

### 8.14.5 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 8.85 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.

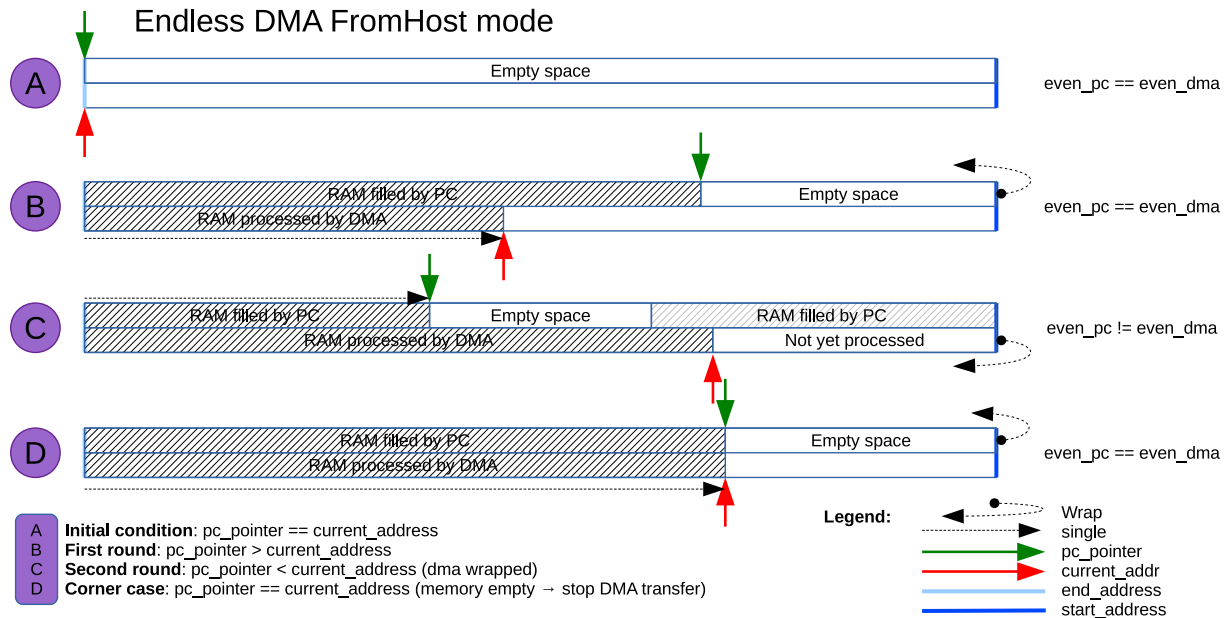


**Figure 8.85:** Endless DMA buffer and pointers representation diagram in ToHost mode.

Looking at Figure 8.85 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 8.86:** Endless DMA buffer and pointers representation diagram in FromHost mode.

Looking at Figure 8.86 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.

### 8.14.6 TRICKLE DESCRIPTOR

The trickle descriptor is a special FromHost descriptor which is implemented at *NUMBER\_OF\_DESCRIPTOR*. This descriptor works exactly as the other FromHost descriptor if *WRAP\_AROUND* is '0' (single shot DMA), but with *WRAP\_AROUND* set to '1', it ignores the *PC\_POINTER* so the throughput is not throttled by the software. Instead the throughput is limited by the target E-Link as defined in the contents of the *cmem\_rcc* buffer containing the trickle commands. If all the data in that buffer is targeted to one 80 Mb/s E-Link the DMA throughput is also limited to 80 Mb/s automatically. Wupper will keep playing back the trickle memory in the host PC until the *DMA\_DESC\_ENABLE* bit is cleared by the software.

## 8.14.7 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 [PCle\_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 in phase1 (rm-4.x) firmware. All the other interrupt sources have been removed since multiple ToHost descriptors were introduced in rm-5.x. The interrupts are detailed in Table 8.52.

**Table 8.52:** 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	ToHost 4 Available	Fired when data becomes available in the ToHost FIFO 4
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 any write occurs to a register in BAR0. The idea is that this write occurs when 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.

## 8.14.8 XILINX PCIE ENDPOINT CORE

Wupper was built around the interface of the Virtex-7 FPGA Gen3 Integrated Block for PCI Express v4.3 [22], and was later ported to other Xilinx PCIe hard blocks:

- Virtex-7 FPGA Gen3 Integrated Block for PCI Express [22]. 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 [23]. 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 [24]. 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. <sup>10</sup>
- Versal ACAP Integrated Block for PCI Express [25]. 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.

<sup>10</sup>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

### 8.14.8.1 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	<b>Requester reQ</b> 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	<b>Requester C</b> 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	<b>Completer reQ</b> uest. This interface is used to write the DMA descriptors as well as some other registers.	PC → FPGA
axis_cc	<b>Completer C</b> 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 8.54:** AXI4-Stream streams.

### 8.14.8.2 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.

### 8.14.9 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](#)

### 8.14.10 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.

### 8.14.11 ERROR HANDLING

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

### 8.14.12 ESTIMATED RESOURCE USAGE

The estimated resource usage of Wupper, including register map 5.0 can be found in Table [8.55](#). 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 8.55:** Wupper Resource utilization.

### 8.14.13 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" which 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 8.4:** Run the simulation.

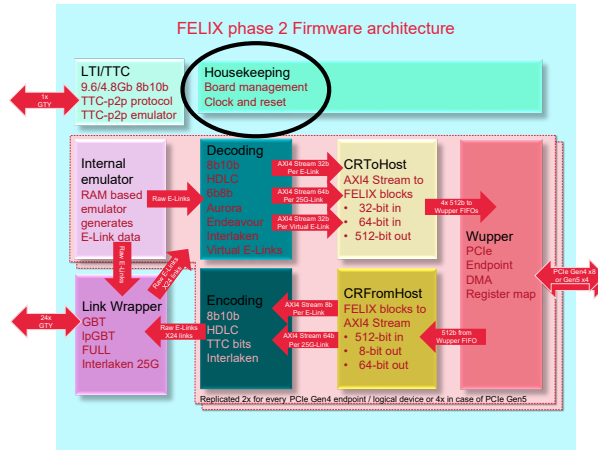
```
cd FELIX/firmware/simulation/UVVMExample
./ci.sh Wupper
```

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.

## 8.15 HOUSEKEEPING

### 8.15.1 INTRODUCTION

Housekeeping is an entity that gathers a set of components to manage and set up the board, as well as assigning some values to a set of registers that represent values of global / toplevel generics.



**Figure 8.87:** The housekeeping interface in the toplevel diagram.



## 8.15.2 INTERFACES



Figure 8.88: Housekeeping interface symbol.

## 8.15.3 FUNCTIONAL DESCRIPTION

### 8.15.3.1 I2C INTERFACE

The I2C interface controls the I2C pins (SCL/SDA) to control several chips on the board. The I2C interface from [i2c on OpenCores](#) has been used.

### 8.15.3.2 GENERICCONSTANTSTOREGS

This entity assigns a set of toplevel generics to the register map section `register_map_gen_board_info`

#### 8.15.3.3 XADC\_DRP

An entity to measure the temperature and FPGA voltages using the XADC or system\_management\_wizard IP cores.

#### 8.15.3.4 DNA

An entity to read out the FPGA DNA (Unique ID) from a register in the FPGA

#### 8.15.3.5 FLASH\_WRAPPER

An entity to read and write the BPI flash on the FLX712 card. This entity may have to be expanded to support SPI flash on the Phase II board.

#### 8.15.3.6 LMK03200\_WRAPPER

Initializes the LMK03200 chip on the FLX712 card to 320.632 MHz for the lpGBT core. The default is to use the Si5345 jitter cleaner and 240.474 MHz.

#### 8.15.3.7 PEX\_INIT

Initializes the PEX PCIe bridge on the FLX712 card. The FLX181 card has a PEX chip that is programmed only once at production time.

#### 8.15.3.8 GC\_MULTICHANNEL\_FREQUENCY\_METER

A frequency meter from [general-cores on OHWR](#) to measure the recovered clock of the links.

#### 8.15.3.9 TACHOMETER

Process to measure the fan speed on the board.

## 8.16 CLOCK AND RESET

### 8.16.1 INTRODUCTION

The entity `clock_and_reset` provides most the system clocks for the FPGA that are synchronous to the LHC clock or a multiple of that, as well as a reset that is synchronous to the 40.079 MHz LHC clock.

### 8.16.2 INTERFACES

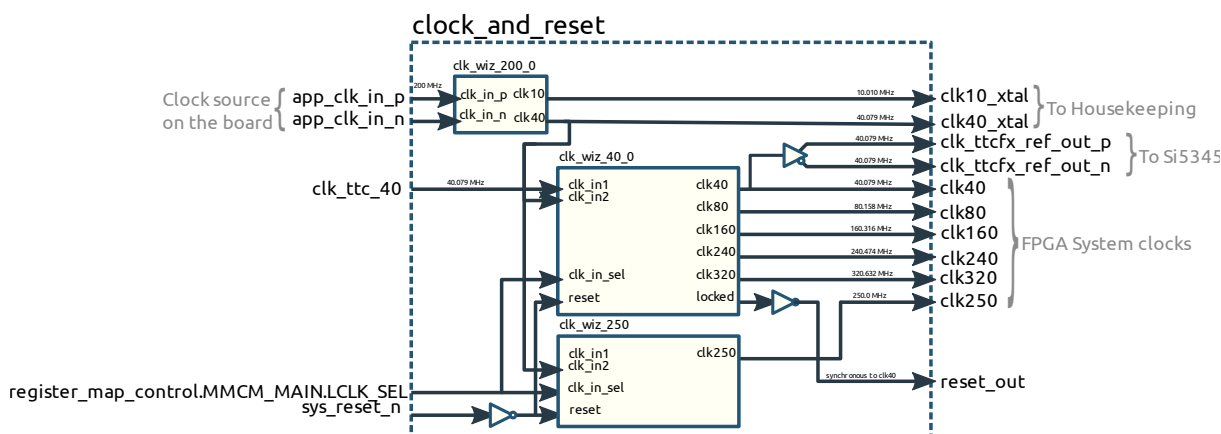


Figure 8.89: Clock and reset block diagram.

- `MMCM_Locked_out` : towards Housekeeping for monitoring
- `MMCM_OscSelect_out` : towards Housekeeping for monitoring
- `clk_ttc_40` : 40.079 MHz clock from TTC or TTC-LTI wrapper
- `app_clk_in_n` : Local clock oscillator on the board (200 MHz)
- `app_clk_in_p` : Local clock oscillator on the board (200 MHz)
- `clk10_xtal` : 10.01 MHz clock derived from local clock source
- `clk40_xtal` : 40.079 MHz clock derived from local clock source
- `clk40` : 40.079 MHz clock for the FPGA logic, synchronous to either TTC or the local clock
- `clk80` : 80.158 MHz clock for the FPGA logic, synchronous to either TTC or the local clock
- `clk160` : 160.316 MHz clock for the FPGA logic, synchronous to either TTC or the local clock
- `clk240` : 240.474 MHz clock for the FPGA logic, synchronous to either TTC or the local clock
- `clk250` : 250.0 MHz clock for the FPGA logic, derived from either TTC or the local clock
- `clk320` : 320.632 MHz clock for the FPGA logic, synchronous to either TTC or the local clock
- `clk_ttcfx_ref_out_n` : Differential copy of `clk40`, to Si5345 jitter cleaner clock input
- `clk_ttcfx_ref_out_p` : Differential copy of `clk40`, to Si5345 jitter cleaner clock input
- `register_map_control` : Contains register `MMCM_MAIN.LCLK_SEL` to switch from local clock to ttc clock
- `reset_out` : System reset (hard reset) synchronous to `clk40`
- `sys_reset_n` : Active-low reset input, connected to PCIe PERSTn pin on the PCIe edge connector

### 8.16.3 FUNCTIONAL DESCRIPTION

clock\_and\_reset contains 3 MMCM components. The first one is to turn the 200 MHz clock source on the board into a local 40 MHz clock. The other 2 MMCM components have 2 clock inputs and can be switched between the local 40 MHz clock and the TTC clock by means of the MMCM\_MAIN.LCLK\_SEL register. The 250 MHz clock is created by an additional MMCM because clk\_wiz\_40\_0 can not create a frequency close to 250 MHz with the other frequencies already set.

clk40 is buffered through an OBUFDS buffer to provide a clock for the Si5345 jitter cleaner. This jitter cleaner will be set up through I2C commands (software controlled) to create the 240.474 MHz reference clocks for the transceivers.

# 9

## 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

### 9.1 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, including 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 [\[22\]](#) [\[23\]](#) [\[24\]](#) [\[25\]](#)

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 BFM.
- The developer of a block is responsible for a complete coverage of the block by the testbench.

### 9.1.1 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. [27]. 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 9.1

```

1558 # UVVM: =====
1559 # UVVM: *** FINAL SUMMARY OF ALL ALERTS ***
1560 # UVVM: =====
1561 # UVVM:
1562 # UVVM:          REGARDED   EXPECTED   IGNORED   Comment?
1563 # UVVM:          :         0         0         0         ok
1564 # UVVM:          :         0         0         0         ok
1565 # UVVM:          :         0         0         0         ok
1566 # UVVM:          :         1         0         0         *** TB_WARNING ***
1567 # UVVM:          :         0         0         0         ok
1568 # UVVM:          :         0         0         0         ok
1569 # UVVM:          :         0         0         0         ok
1570 # UVVM:          :         0         0         0         ok
1571 # UVVM: =====
1572 # UVVM: >> Simulation SUCCESS: No mismatch between counted and expected serious alerts
1573 # UVVM: =====
1574 # UVVM:
1575 # UVVM:
1576 # UVVM:
1577 # UVVM:
1578 # UVVM: ID_LOG_HDR          989.0 ns CRFromHostAxis_tb          SIMULATION COMPLETED
1579 # UVVM: -----

```

Figure 9.1: Results summary of a UVVM successful simulation.

#### Requirement 9.1: UVVM Testbenches

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

## 9.2 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 9.2

#### Requirement 9.2: CI Simulation

For every commit, the simulation testbenches as described in Section 9.1 will be executed by Gitlab CI.

#### Requirement 9.3: 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.

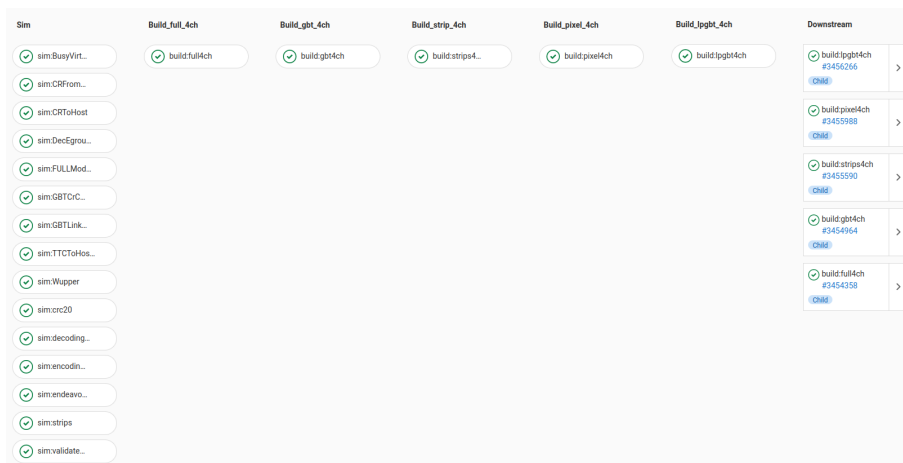


Figure 9.2: Continuous Integration Pipelines as seen in the Gitlab interface.

## 9.3 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/>

# 10

## FIRMWARE MANAGEMENT AND RELIABILITY MATTERS

### 10.1 FIRMWARE SOURCE MANAGEMENT AND RELEASE PLAN

The source code management plan is described in the [FELIX developer manual](#). The firmware can be found in the FELIX firmware repository: <https://gitlab.cern.ch/atlas-tdaq-felix/firmware/>. All issues are tracked in [JIRA](#).

The firmware repository holds code for phase1 (master branch) and phase2 (phase2/master). All branches related to phase2 development are prefixed with phase2/. Both master and phase2/master are protected and merge requests can be completed by the firmware coordinator.

#### 10.1.1 VERSION NUMBERS AND RELEASES

Releases targeted to phase2 start with 5.0-0. The version number is closely related to the version number of the register map. The first official release will be 5.0-xxx where xxx is the number of GiT commits after the rm-5.0 tag. On the release the v5.0 tag will be created, after which the rm-5.1 will mark the beginning of a new release cycle.

A taste of what will be included in the firmware releases for Phase II is shown here. For more details see the [FELIX JIRA issue tracker](#).

- **5.0:** Q1 2022 - Initial release build to demonstrate functionality of the new Phase II firmware ecosystem. Reduced channel count builds for FLX712 will be available for all flavours, as well as FLX128, FLX181 and FLX709 builds.
- **5.1:** Q3 2022 - Added support for Interlaken 25Gb, support for newly added FE protocols such as HGTD Altiroc and lumi.
- **5.2:** Q1 2023 - Nearing feature completeness, transition to bug fix releases.

#### 10.1.2 FILE NAME OF A FIRMWARE BUILD

An example of a firmware build is shown here:

FLX128\_FULLMODE\_24CH\_CLKSELECT\_GIT\_phase2-FLX-1769\_AddGBTForVCU128\_rm-5.0\_301\_211221\_-09\_14.tar.gz

The .tar.gz archive contains the following files:



- **<filename>.bit**: Image that can be written into the FPGA memory using JTAG
- **<filename>.mcs**: Image that can be written to the active flash partition either through JTAG or using the fflashprog tool
- **<filename>.prm**: Information about the .mcs file, required when programming through Vivado
- **<filename>.xlsx**: This file is only included if the build was executed in a graphical Vivado session, it contains a resource report of the build.
- **<filename>\_generics\_timing.txt**: Contains a report of the toplevel generics set at build time, plus a timing and utilization report.
- **<filename>\_debug\_nets.ltx**: Only included if the build includes debug probes. Required to debug the internals of the FPGA with ILA or VIO probes over JTAG in the Vivado GUI.

The file name of a build is build from the following strings:

- **FLX128**: The target board, could be FLX709, FLX712, FLX128, FLX181 or other future target boards.
- **FULLMODE**: The firmware flavour for which the firmware was built, could be GBT, FULLMODE, LPGBT, STRIPS, PIXEL, INTERLAKEN.
- **CLKSELECT**: Indication that the internal FPGA clock is selectable between a clock source on the FELIX card and the external (TTC) clock.
- **GIT\_phase2-master**: Branch from which the build was created. In this case phase2/master. Note that the "/" in the name is replaced with "-".
- **rm-5.0\_301**: Indicating the version number (5.0-301) consisting of major.minor-<number of commits after the rm-5.x tag>.
- **211221\_09\_14**: Timestamp of the build time at which the build was initiated. The format is YYMMDD\_hh\_mm.

## 10.2 CONSEQUENCES OF FAILURES

Several factors may induce failures of the FELIX system, these factors can be internal to the firmware, internal to the FELIX hardware, internal to the FELIX software, external instability of the (LTI/TTC) clocking system or instability of the Front-End links, or a combination of these factors. Failure of the FELIX system can have minor or major consequences for ATLAS data taking:

- Incorrect reconstruction of Front End data if data bytes are changed within FELIX or wrongly received / decoded.
- Loss of data from one or more Front End links, temporarily or permanent
- Loss of data of all links connected to a FELIX card or FELIX system
- In case of obstruction (high link load) loss of DCS data
- Loss of control over the Front Ends.

## 10.3 RELIABILITY MEASURES IN THE FELIX FIRMWARE

### 10.3.1 REDUNDANT DMA CHANNELS AND SEPARATION OF DCS DATA

The FELIX Phase II firmware (from release 5.0 and later) will have multiple ToHost DMA channels enabled by default. The number of ToHost DMA channels (descriptors) can be chosen at build time; the default number of DMA channels for Phase II is set to 4 ToHost channels + 1 FromHost channel per Wupper 8.14 endpoint. This means that 8 independent threads of felix-star can process the load of one FELIX card hosting 2 DMA endpoints.

The ToHost Central Router (CRToHost (8.12) is capable of selecting each block of data, based on its E-Link ID, and assign the data to one of the DMA buffers in Wupper.

The mechanism described above benefits reliability in two ways:

- **CPU load:** The data load can be separated over multiple DMA buffers in a configurable way per E-Link. This means that when one link is expected to produce more data than the other, this can be accounted for in the DMA channel assignment in the firmware. This way the load of the CPU cores can be balanced.
- **Isolation of (DCS) data streams:** If high link occupancy is likely to cause buffer overload in the server memory, certain (DCS) E-Links may be assigned to a separate DMA channel / descriptor. This way a separate process will be available to handle important data independent of other data acquisition processes.

### 10.3.2 BUSY AND XOFF MECHANISM

Several sources of BUSY are available to handle exceptions in case of buffer overloads in several sections of the FELIX system. Details about this mechanism are described in 3.4. Additionally the XOFF mechanism may be used if the total link budget exceeds the PCIe bandwidth. This way the buffers in the Front End electronics may be used to reduce data loss in case of data bursts.

### 10.3.3 (E-)LINK REALIGNMENT AND TRUNCATION

8b10b decoders in FELIX are equipped with an automatic realignment mechanism. When an illegal K-character is received (not-in-table) the E-Link will automatically de-align, and the alignment sequence will be initiated until a valid IDLE character or sequence is received again. This mechanism will help mitigating an overload of the system in case of broken links that will produce random data. A similar mechanism is available for FULL mode links. HDLC links to GBT-SCA devices are enabled with an optional truncation mechanism to mitigate random data, and data messages (chunks) will optionally be truncated at a set length.

# 11

## ORGANIZATION OF FIRMWARE DEVELOPMENT

This section presents a rough but reasonable estimate of the duration for the first prototyping phase, to realize release 5.0. 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.

A large part of the FELIX functionality has been implemented in FELIX Phase I, especially FULL mode and GBT mode. In Phase I however, the most complex part of the FELIX firmware - the Central Router - was implemented differently for each firmware flavour. In Phase II, the Central Router has been divided into 4 independent parts: CRTToHost, CRFromHost, Encoding and Decoding. CRTToHost and CRFromHost are agnostic to the protocol specific data formats, but rather handle a standardized AXI stream format which is translated into or from the FELIX specific block format in the PCIe DMA buffers. Protocol specific encoding and decoding can be enabled at build time, and is handled in the encoding and decoding blocks. This change in design philosophy increases the flexibility of the firmware design and makes it easier to test and simulate certain parts of the firmware separately using standardized design methodologies.

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 required some changes in the Wupper Core. PCIe Gen4 functionality has been implemented in Wupper and was verified on the FLX128 (Virtex Ultrascale+) card as well as the FLX181 card (Versal Prime). It will be a minor change to upgrade Wupper to PCIe Gen5 for Versal Premium Xilinx devices. New protocols such as IpGBT and Interlaken 25G have been implemented and verified as well, and are integrated in the phase2 design.

For the FELIX Phase I design, a system of responsibilities was established, where certain institutes were responsible for certain functional components of the firmware, the benefit of that methodology is that it is very clear who can be held responsible for the implementation and maintenance of a certain part of the firmware, but a drawback is that the work load on certain developers may be high while others can be idle because a certain component may require less attention.

In Phase II, the roles are slightly different. While certain people play expert roles for certain parts, the different developers may be assigned different smaller tasks within the development cycle depending on the need, availability and personal preference. This way it is also guaranteed that knowledge is spread among the different collaborating institutes and will be maintained as developers join or leave the project.

## 11.0.1 INSTITUTES CONTRIBUTING TO FELIX FIRMWARE

Argonne National Lab (ANL), Lemont, Illinois, USA	
Bergische Universität Wuppertal, Germany	
Brookhaven National Lab (BNL), Upton, New York, USA	
CERN, Switzerland	
IFIN-HH Bucharest, Romania	
INFN - Istituto Nazionale di Fisica Nucleare, Italy	
Nikhef - National Institute for Subatomic Physics Amsterdam, The Netherlands	
Technion - Israel Institute of Technology - Haifa, Israel	
Transilvania University of Brasov, Romania	
University of Bologna, Italy	
University of British Columbia (UBC), Canada	
University of Copenhagen, Denmark	
University of Göttingen, Germany	
U-Politechnica Bucharest, Romania	
Weizmann Institute of Science, Rehovot, Israel.	

**Table 11.1:** Institutes contributing to FELIX Firmware.

## 11.0.2 DEVELOPERS AND THEIR ROLES IN THE FELIX FIRMWARE

- Alessandra Camplani <alessandra.camplani@cern.ch>
  - TTC Emulator
- Alessandro Palombi <alessandro.palombi@cern.ch>
  - DUNE
- Alessandro Thea <alessandro.thea@cern.ch>
  - DUNE
  - Optimization of the register map
- Alexander Paramonov <alexander.paramonov@cern.ch>
  - TTC, LTI-TTC
- Ali Skaf <askaf@lab34.ph2.physik.uni-goettingen.de>
  - UVVM simulation
  - TTC Emulator
- Anamika Aggarwal <anamika.aggarwal@cern.ch>
  - GBT front end emulation (GBT sniffer)
- Andrea Borga <andrea.borga@cern.ch>
  - Previous firmware coordinator
  - Toplevel design
  - Wupper interrupt controller
  - Housekeeping
- Carsten D ijlsen <carsten.dulsen@cern.ch>
  - ITk Pixel
  - IpGBT E-Link decoding
- Dimitrios Matakias <dimitrios.matakias@cern.ch>
  - IpGBT core
- Dylan Green <dylan.green@alumni.ubc.ca>
  - ITk Strips
- Elena Zhivun <elena.zhivun@cern.ch>
  - ITk Strips
- Enrico Gamberini <enrico.gamberini@cern.ch>
  - DUNE
- Fabrizio Alfonsi <falfonsi@bo.infn.it>
  - E-Link encoding (GBT)
- Filiberto Bonini <filiberto.bonini@cern.ch>
  - DUNE

- Optimization of the register map
  - Core1990 / Interlaken
- Frans Schreuder <f.schreuder@nikhef.nl>
  - Firmware coordinator
  - Wupper
  - CRTtoHost
  - Decoding
  - Housekeeping
- Hao Xu <haoxu@bnl.gov>
  - Hardware support
- Hongbin Liu <hongbin.liu@cern.ch>
  - Hardware support
- Israel Grayzman <israel.grayzman@weizmann.ac.il>
  - Central Router (Phase I)
- Jacopo Pinzino <jacopo.pinzino@cern.ch>
  - Endeavour endoder / decoder
- Julia Narevicius <julia.narevicius@weizmann.ac.il>
  - Central Router (Phase I)
- Kai Chen <kai@cern.ch>
  - GBT, IpGBT and FULL mode transceiver wrappers.
- Kazuki Todome <ktodome@cern.ch>
  - E-Link encoding (GBT)
- Marco Trovato <mtrovato@felix01.hep.anl.gov>
  - ITK Pixel decoding
  - ITK Pixel encoding
  - IpGBT wrapper
  - FELIG
  - Various contributions in several blocks
- Marius Wensing <wensing@uni-wuppertal.de>
  - ITK Pixel / RD53B decoding
  - CRFromHost
  - Various contributions in encoding and decoding
- Mark Donszelmann <mark.donszelmann@cern.ch>
  - Software coordinator
  - WupperCodeGen
- Mesfin Gebyehu <m.gebyehu@nikhef.nl>

- FMEmu
- External TTC Emulator
- XOFF / BUSY implementation
- Nayib Boukadida <n.boukadida@nikhef.nl>
  - 100Gb/s RDMA core
  - Core1990/Interlaken
- Nico Giangiacomi <nico.giangiacomi@cern.ch>
  - DUNE
  - TTC
  - Endeavour encoder / decoder
  - HDLC encoder / decoder
  - E-Link encoding (GBT)
- Ohad Shaked <ohad.shaked@weizmann.ac.il>
  - Central Router (Phase I)
- Radu Mihai Coliban <coliban.radu@unitbv.ro>
  - NSW compatibility for FELIG
- Rene Habraken <r.habraken@science.ru.nl>
  - FMEmu
- Ricardo Luz <rluz@felix02.hep.anl.gov>
  - FELIG
  - TTC Encoder
- Ryan Quinn <rquinn@cern.ch>
  - ITk Strips emulator in FELIG
- Shaochun Tang <shaochun.tang@cern.ch>
  - Hardware support
- Shelfali Saxena <ssaxena@felix01.hep.anl.gov>
  - FELIG
  - GBT Wrapper
- Simone Ponzio <simone.ponzio@cern.ch>
  - Endeavour encoder / decoder
- Soo Ryu <soo.ryu@cern.ch>
  - TTC
  - BUSY
- Thei Wijnen <t.wijnen@hef.ru.nl>
  - FELIX MROD
- Ton Fleuren <t.fleuren@hef.ru.nl>

- HGTD encoder / decoder
- Tong Xu <xut@felix02.hep.anl.gov>
  - Core 1990/Interlaken
  - Versal compatibility
- Weihao Wu <weihaowu@bnl.gov>
  - GBT, lpGBT and FULL mode wrappers
- William Wulff <william.wulff@cern.ch>
  - DUNE



## REFERENCES

- [1] Argonne National Lab. *FELIG User Manual*. URL: [https://gitlab.cern.ch/atlas-tdaq-felix/felig-tools/-/blob/FELIG\\_Scripts/FeligFLX712Introduction.pdf](https://gitlab.cern.ch/atlas-tdaq-felix/felig-tools/-/blob/FELIG_Scripts/FeligFLX712Introduction.pdf) (cit. on p. 3).
- [2] Nikhef, Radboud University Nijmegen. *FMEMU, Documentation needed*. URL: <https://gitlab.cern.ch/atlas-tdaq-felix/firmware/> (cit. on p. 3).
- [3] Nikhef, Radboud University Nijmegen. *FELIX\_MROD, Documentation needed*. URL: <https://gitlab.cern.ch/atlas-tdaq-felix/firmware/> (cit. on p. 3).
- [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](https://doi.org/10.1109/TIM.2019.2947972) (cit. on p. 23).
- [5] *FPGA implementation of RDMA for ATLAS Readout with FELIX at High Luminosity LHC*. URL: <https://indico.cern.ch/event/1019078/contributions/4444212/> (cit. on p. 24).
- [6] Frans Schreuder. *Tool to create block diagrams from VHDL entities*. URL: <https://github.com/fransschreuder/entity-block> (cit. on pp. 35, 120, 121).
- [7] Aliaksei Chapyzenka. *Tool to create waveforms*. URL: <https://wavedrom.com/> (cit. on pp. 38, 41, 83, 139).
- [8] *RD53B users guide*. Tech. rep. Geneva: CERN, 2020. URL: <https://cds.cern.ch/record/2754251> (cit. on pp. 48, 88).
- [9] XILINX. *Aurora 64B/66B Protocol Specification*. URL: [https://www.xilinx.com/support/documentation/ip\\_documentation/aurora\\_64b66b\\_protocol\\_spec\\_sp011.pdf](https://www.xilinx.com/support/documentation/ip_documentation/aurora_64b66b_protocol_spec_sp011.pdf) (cit. on p. 48).
- [10] Wikipedia. *High-Level Data Link Control*. URL: [https://en.wikipedia.org/wiki/High-Level\\_Data\\_Link\\_Control](https://en.wikipedia.org/wiki/High-Level_Data_Link_Control) (cit. on pp. 59, 60, 107).
- [11] Julian Maxime Mendez. *GBT-SC module for FPGA*. URL: <https://gitlab.cern.ch/gbtsc-fpga-support/gbt-sc> (cit. on p. 59).
- [12] CERN GBT Project. "The GBTx Manual". In: V0.14 (2016). URL: <https://espace.cern.ch/GBT-Project/GBTX/Manuals/gbtManual.pdf> (cit. on pp. 62, 112).
- [13] TTC group. "CERN TTC homepage". In: (). URL: <http://ttc.web.cern.ch/TTC> (cit. on p. 110).
- [14] *Phase-II Local Trigger Interface (LTI) Specification*. Tech. rep. Geneva: CERN, 2021. URL: <https://edms.cern.ch/ui/#!/master/navigator/document?P:1223963387:100640118:subDocs> (cit. on pp. 114, 128).
- [15] 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](https://edms.cern.ch/ui/file/1563801/1/RequirementsPhaseII_v1.1.0.pdf) (cit. on pp. 114, 128).
- [16] 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](https://doi.org/10.1088/1748-0221/12/07/p07011). URL: <https://doi.org/10.1088/1748-0221/12/07/p07011> (cit. on p. 116).
- [17] LpGBT-FPGA. 2018. URL: <http://lpGBT-fpga.web.cern.ch/doc/html/index.html> (cit. on p. 116).
- [18] 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](http://www.xilinx.com/support/documentation/user_guides/ug476_7Series_Transceivers.pdf) (cit. on p. 117).
- [19] Jochem Leijenhorst. URL: [https://atlas-project-felix.web.cern.ch/atlas-project-felix/dev/docs/Bachelor\\_thesis\\_chunk\\_headers\\_Jochem\\_Leijenhorst.pdf](https://atlas-project-felix.web.cern.ch/atlas-project-felix/dev/docs/Bachelor_thesis_chunk_headers_Jochem_Leijenhorst.pdf) (cit. on p. 135).
- [20] The FELIX team. *atlas-tdaq-felix website*. URL: <https://atlas-project-felix.web.cern.ch/atlas-project-felix/> (cit. on p. 141).
- [21] Xilinx. *UG761: Xilinx AXI Bus documentation*. URL: [http://www.xilinx.com/support/documentation/ip\\_documentation/axi\\_ref\\_guide/latest/ug761\\_axi\\_reference\\_guide.pdf](http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf) (cit. on pp. 141, 145).

- [22] 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](https://www.xilinx.com/support/documentation/ip/_documentation/pcie3/_7x/v4/_3/pg023_v7_pcie_gen3.pdf) (cit. on pp. 141, 151, 159).
- [23] 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](https://www.xilinx.com/support/documentation/ip/_documentation/pcie3/_ultrascale/v4/_4/pg156-ultrascale-pcie-gen3.pdf) (cit. on pp. 141, 151, 159).
- [24] 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](https://www.xilinx.com/support/documentation/ip/_documentation/pcie4/_uscale/_plus/v1/_3/pg213-pcie4-ultrascale-plus.pdf) (cit. on pp. 141, 151, 159).
- [25] 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](https://www.xilinx.com/support/documentation/ip/_documentation/pcie/_versal/v1/_0/pg343-pcie-versal.pdf) (cit. on pp. 141, 151, 159).
- [26] 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. 145).
- [27] Ali Skaf. *FELIX Standardized Firmware testbench with Gitlab CI*. URL: [https://indico.cern.ch/event/858260/contributions/3613811/attachments/1930907/3198159/ASkaf\\_QT5r1.pdf](https://indico.cern.ch/event/858260/contributions/3613811/attachments/1930907/3198159/ASkaf_QT5r1.pdf) (cit. on p. 160).
- [28] 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](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.40).

---

# Appendix A

---

## CODE MANAGEMENT

---

Everything related to FELIX firmware code management has been described in the [FELIX developer manual](#)

---

# Appendix B

---

## APPENDIX

## B.1 FELIX REGISTER MAP, VERSION 5.1

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	0x0000000000000000
		START_ADDRESS	63:0	W	Start Address	0x0000000000000000
0x0010	0,1	DMA_DESC_0a				
		SW_POINTER	127:64	W	Pointer controlled by the software, indicating read or write status for circular DMA	0x0000000000000000
		WRAP_AROUND	12	W	Wrap around	0x0
		FROMHOST	11	R	1: fromHost/ 0: toHost	0x0
		NUM_WORDS	10:0	W	Number of 32 bit words	0x00
		...				
0x00E0	0,1	DMA_DESC_7				
		END_ADDRESS	127:64	W	End Address	0x0000000000000000
		START_ADDRESS	63:0	W	Start Address	0x0000000000000000
0x00F0	0,1	DMA_DESC_7a				
		SW_POINTER	127:64	W	Pointer controlled by the software, indicating read or write status for circular DMA	0x0000000000000000
		WRAP_AROUND	12	W	Wrap around	0x0
		FROMHOST	11	R	1: fromHost/ 0: toHost	0x0
		NUM_WORDS	10:0	W	Number of 32 bit words	0x00
		DMA_DESC_STATUS				
0x0200	0,1	DMA_DESC_STATUS_0				
		EVEN_PC	66	R	Even address cycle PC	0x0
		EVEN_DMA	65	R	Even address cycle DMA	0x0
		DESC_DONE	64	R	Descriptor Done	0x0
		FW_POINTER	63:0	R	Pointer controlled by the firmware, indicating where the DMA is busy reading or writing	0x0000000000000000
...						
0x0270	0,1	DMA_DESC_STATUS_7				
		EVEN_PC	66	R	Even address cycle PC	0x0
		EVEN_DMA	65	R	Even address cycle DMA	0x0
		DESC_DONE	64	R	Descriptor Done	0x0
		FW_POINTER	63:0	R	Pointer controlled by the firmware, indicating where the DMA is busy reading or writing	0x0000000000000000
0x0300	0,1	BAR0_VALUE	31:0	R	Copy of BAR0 offset reg.	0x00000000
0x0310	0,1	BAR1_VALUE	31:0	R	Copy of BAR1 offset reg.	0x00000000

0x0320	0,1	BAR2_VALUE	31:0	R	Copy of BAR2 offset reg.	0x00000000
0x0400	0,1	DMA_DESC_ENABLE	7:0	W	Enable descriptors 7:0. One bit per descriptor. Cleared when Descriptor is handled.	0x00
0x0420	0,1	DMA_RESET	any	T	Reset Wupper Core (DMA Controller FSMs)	0x0
0x0430	0,1	SOFT_RESET	any	T	Global Software Reset. Any write resets applications, e.g. the Central Router.	0x0
0x0440	0,1	REGISTER_RESET	any	T	Resets the register map to default values. Any write triggers this reset.	0x0
0x0450	0,1	FROMHOST_FULL_THRESH				
		THRESHOLD_ASSERT	22:16	W	Assert value of the FromHost programmable full flag	0x0
		THRESHOLD_NEGATE	6:0	W	Negate value of the FromHost programmalbe full flag	0x0
0x0460	0,1	TOHOST_FULL_THRESH				
		THRESHOLD_ASSERT	27:16	W	Assert value of the ToHost programmable full flag	0x000
		THRESHOLD_NEGATE	11:0	W	Negate value of the ToHost programmalbe full flag	0x000
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)..	0x0000000006400000
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).	0x0000000006E00000
0x0490	0,1	BUSY_STATUS	0	R	A tohost descriptor passed BUSY_THRESHOLD_ASSERT, busy flag set	0x0
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	0x0000000001000000
0x04B0	0,1	TOHOSTFIFO_EMPTY	3:0	R	Empty flags of the ToHost FIFOs in Wupper	0x0
0x04C0	0,1	TOHOSTFIFO_PEMPTY	3:0	R	Programmable empty flags of the ToHost FIFOs in Wupper	0x0
0x04D0	0,1	FROMHOSTFIFO_FULL	0	R	Full flag of the FromHost FIFO in Wupper	0x0
0x04E0	0,1	FROMHOSTFIFO_PFULL	0	R	Programmable full flag of the FromHost FIFO in Wupper	0x0

**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	0x00000000
		INT_DATA	95:64	W	Interrupt Data	0x00000000
		INT_ADDRESS	64:0	W	Interrupt Address	0x0000000000000000
...						
0x00F0	0,1	INT_VEC_15				
		INT_CTRL	127:96	W	Interrupt Control	0x00000000
		INT_DATA	95:64	W	Interrupt Data	0x00000000
		INT_ADDRESS	64:0	W	Interrupt Address	0x0000000000000000
0x0100	0,1	INT_TAB_ENABLE	7:0	W	Interrupt Table enable Selectively enable Interrupts	0x00

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, 5.1 formatted as 0x0501	0x0000
0x0010	0,1	BOARD_ID_TIMESTAMP	39:0	R	Board ID Date / Time in BCD format YYMMDDhhmm	0x0000000000
0x0030	0,1	GIT_COMMIT_TIME	39:0	R	Board ID GIT Commit time of current revision, Date / Time in BCD format YYMMDDhhmm	0x0000000000
0x0040	0,1	GIT_TAG	63:0	R	String containing the current GIT TAG	0x0000000000000000
0x0050	0,1	GIT_COMMIT_NUMBER	31:0	R	Number of GIT commits after current GIT_TAG	0x00000000
0x0060	0,1	GIT_HASH	31:0	R	Short GIT hash (32 bit)	0x00000000
0x0070	0,1	STATUS_LEDS	7:0	W	Board GPIO Leds	0xAB
0x0080	0,1	GENERIC_CONSTANTS				
		TRICKLE_DESCRIPTOR_INDEX	35:32	R	Index of the (first if more than one) Trickle descriptor	0x0
		FROMHOST_DESCRIPTOR_INDEX	31:28	R	Index of the (first if more than one) FromHost descriptor	0x0
		TRICKLE_DESCRIPTOR	27:24	R	Number of Trickle descriptors	0x0
		FROMHOST_DESCRIPTOR	23:20	R	Number of FromHost descriptors	0x0
		TOHOST_DESCRIPTOR	19:16	R	Number of ToHost descriptors	0x0
		INTERRUPTS	15:8	R	Number of Interrupts	0x00
0x0090	0,1	DESCRIPTORS	7:0	R	Number of Descriptors Tohost + FromHost excluding trickle descriptor	0x00
		NUM_OF_CHANNELS	7:0	R	Number of GBT or FULL mode Channels	0x00
		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 - 180 (0x0B4): FLX180, VMK180 - 181 (0x0B5): FLX181, BNL181 - 182 (0x0B6): FLX182, BNL182	0x0000000000000000
		GENERATE_GBT	0	R	1 when the GBT Wrapper is included in the design	0x0
		OPTO_TRX_NUM	7:0	R	Number of optical transceivers in the design	0x00
		GENERATE_TTC_EMU	1	R	1 when TTC emulator is generated	0x0
		INCLUDE_EGROUPS				
0x0100	0,1	INCLUDE_EGROUP_0				
		TOHOST_32	9	R	ToHost EPATH32 is included in this EGROUP	0x0
		FROMHOST_02	8	R	FromHost EPATH02 is included in this EGROUP	0x0
		FROMHOST_04	7	R	FromHost EPATH04 is included in this EGROUP	0x0
		FROMHOST_08	6	R	FromHost EPATH08 is included in this EGROUP	0x0
		FROMHOST_HDLC	5	R	FromHost HDLC is included in this EGROUP	0x0
		TOHOST_02	4	R	ToHost EPATH02 is included in this EGROUP	0x0



		TOHOST_04	3	R	ToHost EPATH04 is included in this EGROUP	0x0
		TOHOST_08	2	R	ToHost EPATH08 is included in this EGROUP	0x0
		TOHOST_16	1	R	ToHost EPATH16 is included in this EGROUP	0x0
		TOHOST_HDLC	0	R	ToHost HDLC is included in this EGROUP	0x0
...						
0x0160	0,1	INCLUDE_EGROUP_6				
		TOHOST_32	9	R	ToHost EPATH32 is included in this EGROUP	0x0
		FROMHOST_02	8	R	FromHost EPATH02 is included in this EGROUP	0x0
		FROMHOST_04	7	R	FromHost EPATH04 is included in this EGROUP	0x0
		FROMHOST_08	6	R	FromHost EPATH8 is included in this EGROUP	0x0
		FROMHOST_HDLC	5	R	FromHost HDLC is included in this EGROUP	0x0
		TOHOST_02	4	R	ToHost EPATH02 is included in this EGROUP	0x0
		TOHOST_04	3	R	ToHost EPATH04 is included in this EGROUP	0x0
		TOHOST_08	2	R	ToHost EPATH08 is included in this EGROUP	0x0
		TOHOST_16	1	R	ToHost EPATH16 is included in this EGROUP	0x0
		TOHOST_HDLC	0	R	ToHost HDLC is included in this EGROUP	0x0
0x0170	0,1	WIDE_MODE	0	R	GBT is configured in Wide mode	0x0
0x0190	0,1	FIRMWARE_MODE	4:0	R	0: GBT mode 1: FULL-GBT 2: LTDB mode (GBT mode with only IC and TTC links) 3: FEI4 mode 4: ITK Pixel 5: ITK Strip 6: FELIG GBT 7: FULL mode emulator 8: FELIX_MROD mode 9: IpGBT mode 10: 25G Interlaken 11: FELIG LPGBT 12: HGTD_LUMI 13: BCMPRIME 14: FELIG_PIXEL 15: FELIG_STRIP	0x0
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)	0x0
0x01B0	0,1	CR_GENERIC				
		XOFF_INCLUDED	2	R	Xoff bits (usually full mode) can be generated by the FromHost Central Router	0x0
		DIRECT_MODE_INCLUDED	1	R	Indicates that the Direct mode functionality was built in the Central Router	0x0
		FROM_HOST_INCLUDED	0	R	Indicates that the From Host path of the Central router was included in the design	0x0
0x01C0	0,1	BLOCKSIZE	15:0	R	Number of bytes in a block	0x0000
0x01D0	0,1	PCIE_ENDPOINT	0	R	Indicator of the PCIe endpoint on BNL71x cards with two endpoints. 0 or 1	0x0
0x01E0	0,1	CHUNK_TRAILER_32B	0	R	Indicator that the chunk trailer is in the new 32-bit format	0x0
0x01F0	0,1	NUMBER_OF_PCIE_ENDPOINTS	1:0	R	Number of PCIe endpoints on the card. The BNL71x cards have 2 endpoints	0x0

0x0200	0,1	AXI_STREAMS_TOHOST				
		IC_INDEX	23:16	R	The AXIs ID (EPath-ID) of the ToHost IC E-Link	0x00
		EC_INDEX	15:8	R	The AXIs ID (EPath-ID) of the ToHost EC E-Link	0x00
		NUMBER_OF_STREAMS	7:0	R	Total number of AXIs IDs (EPath-IDs) per physical link ToHost	0x00
0x0210	0,1	AXI_STREAMS_FROMHOST				
		IC_INDEX	23:16	R	The AXIs ID (EPath-ID) of the FromHost IC E-Link	0x00
		EC_INDEX	15:8	R	The AXIs ID (EPath-ID) of the FromHost EC E-Link	0x00
		NUMBER_OF_STREAMS	7:0	R	Total number of AXIs IDs (EPath-IDs) per physical link FromHost	0x00
0x0220	0,1	FROMHOST_DATA_FORMAT	2:0	R	0: The data format is as it was in phase1, supporting only multiples of 2 bytes 1: FromHost header uses a 5-bit length field as described in FLX-1355 2: FromHost header is 32-bit and the packet length is 256-bit (32 bytes) including the header FLX-1601 3: FromHost header is 32-bit and the packet length is 512-bit (64 bytes) including the header FLX-1601 4: FromHost header is 32-bit and the packet length is 256-bit (32 bytes) including the header FLX-2294. All header bitfields are 8 bit 5: FromHost header is 32-bit and the packet length is 512-bit (64 bytes) including the header FLX-2294. All header bitfields are 8 bit 6: FromHost header is 32-bit and the packet length is 1024-bit (128 bytes) including the header FLX-2294. All header bitfields are 8 bit	0x0
0x0230	0,1	FULLMODE_HALFRATE	0	R	If set to 1 the FULL mode firmware is running at 4.8Gb instead of the default 9.6Gb	0x0
0x0240	0,1	SUPPORT_HDLC_DELAY	0	R	The HDLC encoders can offload a 1us delay as described in FLX-1826	0x0
0x0250	0,1	TOHOST_DATA_FORMAT	1:0	R	0: Use subchunk trailer format 1: Use subchunk header format 2: Use blockless header format	0x0
CR To Host Controls And Monitors						
0x0800	0,1	TIMEOUT_CTRL				
		ENABLE	32	W	1 enables the timeout trailer generation for ToHost mode	0x1
		TIMEOUT	31:0	W	Number of 40 MHz clock cycles after which a timeout occurs.	0xFFFFFFFF
0x0810	0,1	MAX_TIMEOUT	31:0	R	Maximum allowed timeout value	0x00000000
0x0820	0,1	CRTOHOST_FIFO_STATUS				
		CLEAR	any	T	Any write to this register clears the latched FULL flags	0x0
		FULL	47:24	R	Every bit represents the full flag of a channel FIFO	0x0000000
		FULL_LATCHED	23:0	R	like FULL but a latched state, clear by writing to this register	0x0000000
0x0830	0,1	CRTOHOST_DMA_DESCRIPTOR_1				
		WR_EN	any	T	Any write to this register assigns the DMA ID to the AXIS_ID set in CRTOHOST_DMA_DESCRIPTOR_2.AXIS_ID	0x0
		DESCR	2:0	W	Target descriptor	0x0
0x0840	0,1	CRTOHOST_DMA_DESCRIPTOR_2				
		DESCR_READ	13:11	R	Read back the value of the descriptor assigned to AXIS_ID	0x0
		AXIS_ID	10:0	W	ID of the AXI stream (E-Path ID) to associate with CRTOHOST_DMA_DESCRIPTOR_1.DESCR	0x00
CRTOHOST_INSTANT_TIMEOUT_ENA_GEN						
0x0850	0,1	CRTOHOST_INSTANT_TIMEOUT_ENA_00	41:0	W	Enable instant timeout after the first data arrives in CRToHost.	0x0000000000

...						
0x09C0	0,1	CRTOHOST_INSTANT_TIMEOUT_ENA_23	41:0	W	Enable instant timeout after the first data arrives in CRToHost.	0x0000000000
0x09D0	0,1	DISCARD_DATA_FOR_DESCR				
		FIFO_FULL	15:8	W	Discard data for a given DMA channel when Wupper FIFO is full, even if DMA is enabled	0x00
		DMA_DISABLED	7:0	W	Discard data for a given DMA channel when Wupper FIFO is full, and the descriptor is not enabled	0xFF
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	0x0
		FULL	47:24	R	Every bit represents the full flag of a channel FIFO	0x000000
		FULL_LATCHED	23:0	R	like FULL but a latched state, clear by writing to this register	0x000000
BROADCAST_ENABLE_GEN						
0x1010	0,1	BROADCAST_ENABLE_00	41:0	W	Enable path to be included in a broadcast message.	0x0000000000
...						
0x1180	0,1	BROADCAST_ENABLE_23	41:0	W	Enable path to be included in a broadcast message.	0x0000000000
0x1190	0,1	CRFROMHOST_RESET	any	T	Central Router FromHost Controls and Monitors	0x0
Decoding Controls And Monitors						
0x1800	0,1	ELINK_REALIGNMENT				
		CLEAR_REALIGNMENT_STATUS	any	T	Clears the ELINK Realignement event flags	0x0
		ENABLE	0	W	Enable realignment mechanism in 8b10b E-Links after illegal character reception.	0x1
ELINK_REALIGNMENT_STATUS_GEN						
0x1810	0,1	ELINK_REALIGNMENT_STATUS_00	41:0	R	A realignment event due to an illegal 8b10b symbol has occurred. 1 bit per Epath. Clear status by writing to ELINK_REALIGNMENT.CLEAR_REALIGNMENT_STATUS	0x0000000000
...						
0x18C0	0,1	ELINK_REALIGNMENT_STATUS_11	41:0	R	A realignment event due to an illegal 8b10b symbol has occurred. 1 bit per Epath. Clear status by writing to ELINK_REALIGNMENT.CLEAR_REALIGNMENT_STATUS	0x0000000000
ELINK_REALIGNMENT_COUNT_GEN						
0x18D0	0,1	ELINK_REALIGNMENT_COUNT_00	31:0	R	A realignment event due to an illegal 8b10b symbol on any E-Link in the link increments the counter. Clear status by writing to ELINK_REALIGNMENT.CLEAR_REALIGNMENT_STATUS	0x00000000
...						
0x1980	0,1	ELINK_REALIGNMENT_COUNT_11	31:0	R	A realignment event due to an illegal 8b10b symbol on any E-Link in the link increments the counter. Clear status by writing to ELINK_REALIGNMENT.CLEAR_REALIGNMENT_STATUS	0x00000000
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	0x00
		EGROUP5	47:40	W	EPATH (Wide mode or lpGBT) is associated with a STREAM ID	0x00

		EGROUP4	39:32	W	EPATH is associated with a STREAM ID	0x00
		EGROUP3	31:24	W	EPATH is associated with a STREAM ID	0x00
		EGROUP2	23:16	W	EPATH is associated with a STREAM ID	0x00
		EGROUP1	15:8	W	EPATH is associated with a STREAM ID	0x00
		EGROUP0	7:0	W	EPATH is associated with a STREAM ID, use only bit0 for FULL mode.	0x00
...						
0x2170	0,1	LINK_23_HAS_STREAM_ID				
		EGROUP6	55:48	W	EPATH (Wide mode or lpGBT) is associated with a STREAM ID	0x00
		EGROUP5	47:40	W	EPATH (Wide mode or lpGBT) is associated with a STREAM ID	0x00
		EGROUP4	39:32	W	EPATH is associated with a STREAM ID	0x00
		EGROUP3	31:24	W	EPATH is associated with a STREAM ID	0x00
		EGROUP2	23:16	W	EPATH is associated with a STREAM ID	0x00
		EGROUP1	15:8	W	EPATH is associated with a STREAM ID	0x00
		EGROUP0	7:0	W	EPATH is associated with a STREAM ID, use only bit0 for FULL mode.	0x00
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	0x0000000000000000
...						
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	0x0000000000000000
DECODING_EGROUP_CTRL_GEN						
DECODING_EGROUP						
0x2300	0,1	DECODING_LINK00_EGROUP0_CTRL				
		ENABLE_TRUNCATION	59	W	Enable truncation mechanism in HDLC decoder for chunks > 12 bytes	0x0
		EPATH_ALMOST_FULL	58:51	R	FIFO full indication	0x00
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath	0x00
		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	0x11111111
		EPATH_WIDTH	10:8	W	Width in bits of all EPATHS in an EGROUP 0:2, 1:4, 2:8, 3:16, 4:32	0x0
		EPATH_ENA	7:0	W	Enable bits per EPATH	0x00
...						
0x2360	0,1	DECODING_LINK00_EGROUP6_CTRL				
		ENABLE_TRUNCATION	59	W	Enable truncation mechanism in HDLC decoder for chunks > 12 bytes	0x0
		EPATH_ALMOST_FULL	58:51	R	FIFO full indication	0x00
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath	0x00

		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	0x11111111
		EPATH_WIDTH	10:8	W	Width in bits of all EPATHS in an EGROU 0:2, 1:4, 2:8, 3:16, 4:32	0x0
		EPATH_ENA	7:0	W	Enable bits per EPATH	0x00
...						
DECODING_EGROUP						
0x27D0	0,1	DECODING_LINK11_EGROUP0_CTRL				
		ENABLE_TRUNCATION	59	W	Enable truncation mechanism in HDLC decoder for chunks > 12 bytes	0x0
		EPATH_ALMOST_FULL	58:51	R	FIFO full indication	0x00
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath	0x00
		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	0x11111111
		EPATH_WIDTH	10:8	W	Width in bits of all EPATHS in an EGROU 0:2, 1:4, 2:8, 3:16, 4:32	0x0
		EPATH_ENA	7:0	W	Enable bits per EPATH	0x00
...						
0x2830	0,1	DECODING_LINK11_EGROUP6_CTRL				
		ENABLE_TRUNCATION	59	W	Enable truncation mechanism in HDLC decoder for chunks > 12 bytes	0x0
		EPATH_ALMOST_FULL	58:51	R	FIFO full indication	0x00
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath	0x00
		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	0x11111111
		EPATH_WIDTH	10:8	W	Width in bits of all EPATHS in an EGROU 0:2, 1:4, 2:8, 3:16, 4:32	0x0
		EPATH_ENA	7:0	W	Enable bits per EPATH	0x00
MINI_EGROUP_TOHOST_GEN						
0x2840	0,1	MINI_EGROUP_TOHOST_00				
		ENABLE_AUX_TRUNCATION	15	W	Enable truncation mechanism in HDLC decoder for chunks > 12 bytes	0x0
		ENABLE_IC_TRUNCATION	14	W	Enable truncation mechanism in HDLC decoder for chunks > 12 bytes	0x0

		ENABLE_EC_TRUNCATION	13	W	Enable truncation mechanism in HDLC decoder for chunks > 12 bytes	0x0
		AUX_ALMOST_FULL	12	R	Indicator that the AUX path FIFO is almost full	0x0
		AUX_BIT_SWAPPING	11	W	0: two input bits of IC e-link are as documented, 1: two input bits are swapped	0x1
		AUX_ENABLE	10	W	Enables the AUX channel	0x1
		IC_ALMOST_FULL	9	R	Indicator that the IC path FIFO is almost full	0x0
		IC_BIT_SWAPPING	8	W	0: two input bits of IC e-link are as documented, 1: two input bits are swapped	0x0
		IC_ENABLE	7	W	Enables the IC channel	0x1
		EC_ALMOST_FULL	6	R	Indicator that the EC path FIFO is almost full	0x0
		EC_BIT_SWAPPING	5	W	0: two input bits of EC e-link are as documented, 1: two input bits are swapped	0x0
		EC_ENCODING	4:1	W	Configures encoding of the EC channel	0x2
		EC_ENABLE	0	W	Enables the EC channel	0x1
...						
0x29B0	0,1	MINI_EGROUP_TOHOST_23				
		ENABLE_AUX_TRUNCATION	15	W	Enable truncation mechanism in HDLC decoder for chunks > 12 bytes	0x0
		ENABLE_IC_TRUNCATION	14	W	Enable truncation mechanism in HDLC decoder for chunks > 12 bytes	0x0
		ENABLE_EC_TRUNCATION	13	W	Enable truncation mechanism in HDLC decoder for chunks > 12 bytes	0x0
		AUX_ALMOST_FULL	12	R	Indicator that the AUX path FIFO is almost full	0x0
		AUX_BIT_SWAPPING	11	W	0: two input bits of IC e-link are as documented, 1: two input bits are swapped	0x1
		AUX_ENABLE	10	W	Enables the AUX channel	0x1
		IC_ALMOST_FULL	9	R	Indicator that the IC path FIFO is almost full	0x0
		IC_BIT_SWAPPING	8	W	0: two input bits of IC e-link are as documented, 1: two input bits are swapped	0x0
		IC_ENABLE	7	W	Enables the IC channel	0x1
		EC_ALMOST_FULL	6	R	Indicator that the EC path FIFO is almost full	0x0
		EC_BIT_SWAPPING	5	W	0: two input bits of EC e-link are as documented, 1: two input bits are swapped	0x0
		EC_ENCODING	4:1	W	Configures encoding of the EC channel	0x2
		EC_ENABLE	0	W	Enables the EC channel	0x1
0x29C0	0,1	TTC_TOHOST_ENABLE	0	W	Enables the ToHost Mini Egroup in TTC mode	0x1
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	0x1
0x29E0	0,1	DECODING_ENDIANNESS_FULL_MODE	0	W	Specify the byte order in FULL mode 1: Big-endian 0: Little-endian	0x0
YARR_DEBUG_ALLEGROUP_TOHOST_GEN						
0x29F0	0,1	YARR_DEBUG_ALLEGROUP_TOHOST_00				
		REF_PACKET	63:32	W	Reference packet to be matched	0x02000000
		CNT_RX_PACKET	31:0	R	Count packets of a given value	0x00000000
...						

0x2AA0	0,1	YARR_DEBUG_ALLEGROUP_TOHOST_11				
		REF_PACKET	63:32	W	Reference packet to be matched	0x02000000
		CNT_RX_PACKET	31:0	R	Count packets of a given value	0x00000000
PATH_ERRORS						
0x2AB0	0,1	LINK_00_ERRORS				
		CLEAR_COUNTERS	35	W	Set to 1 to clear all counter values for all egroups in the link. Set to 0 to start counting errors.	0x0
		EGROUP_SELECT	34:32	W	Errors for Egroup1	0x0
		COUNT	31:0	R	Errors for the selected egroup	0x00000000
...						
0x2C20	0,1	LINK_23_ERRORS				
		CLEAR_COUNTERS	35	W	Set to 1 to clear all counter values for all egroups in the link. Set to 0 to start counting errors.	0x0
		EGROUP_SELECT	34:32	W	Errors for Egroup1	0x0
		COUNT	31:0	R	Errors for the selected egroup	0x00000000
0x2C30	0,1	INTERLAKEN_CONTROL				
		HEALTH_INTERFACE	12	R	Automatically detect the lane number in the interlaken descrambler	0x1
		PACKET_LENGTH	11:0	W	Lenth of an interlaken metaframe	0x7E8
INTERLAKEN_STATUS_GEN						
0x2C40	0,1	INTERLAKEN_LANE_00_STATUS				
		CLEAR_STATUS	any	T	Decoding block	0x0
		DECODER_ERROR_SYNC	8	R	Sticky error bit, clear with CLEAR_STATUS	0x0
		DESCRAMBLER_ERROR_BADSYNC	7	R	Sticky error bit, clear with CLEAR_STATUS	0x0
		DESCRAMBLER_ERROR_STATEMISMATCH	6	R	Sticky error bit, clear with CLEAR_STATUS	0x0
		DESCRAMBLER_ERROR_NOSYNC	5	R	Sticky error bit, clear with CLEAR_STATUS	0x0
		BURST_CRC24_ERROR	4	R	Sticky CRC error bit, clear with CLEAR_STATUS	0x0
		META_CRC32_ERROR	3	R	Sticky CRC error bit, clear with CLEAR_STATUS	0x0
		HEALTH_LANE	2	R	Health bit for this lane	0x0
		DESCRAMBLER_ALIGNED	1	R	This channels descrambler is aligned	0x0
		DECODER_ALIGNED	0	R	This channels decoder is aligned	0x0
...						
0x2CF0	0,1	INTERLAKEN_LANE_11_STATUS				
		CLEAR_STATUS	any	T	Decoding block	0x0
		DECODER_ERROR_SYNC	8	R	Sticky error bit, clear with CLEAR_STATUS	0x0
		DESCRAMBLER_ERROR_BADSYNC	7	R	Sticky error bit, clear with CLEAR_STATUS	0x0
		DESCRAMBLER_ERROR_STATEMISMATCH	6	R	Sticky error bit, clear with CLEAR_STATUS	0x0
		DESCRAMBLER_ERROR_NOSYNC	5	R	Sticky error bit, clear with CLEAR_STATUS	0x0
		BURST_CRC24_ERROR	4	R	Sticky CRC error bit, clear with CLEAR_STATUS	0x0
		META_CRC32_ERROR	3	R	Sticky CRC error bit, clear with CLEAR_STATUS	0x0

		HEALTH_LANE	2	R	Health bit for this lane	0x0
		DESCRAMBLER_ALIGNED	1	R	This channels descrambler is aligned	0x0
		DECODER_ALIGNED	0	R	This channels decoder is aligned	0x0
SUPER_CHUNK_FACTOR_GEN						
0x2DE0	0,1	SUPER_CHUNK_FACTOR_LINK_00	7:0	W	number of chunks glued together	0x01
...						
0x2E90	0,1	SUPER_CHUNK_FACTOR_LINK_11	7:0	W	number of chunks glued together	0x01
DECODING_LINK_CB_GEN						
0x2EA0	0,1	DECODING_LINK_00_CB				
		DESKEWED	61:4	R	Every bit corresponds to an E-link on one (lp)GBT frame. Register indicates whether the E-link has been de-skewed in the channel. E-link are grouped in a channel according to CBOPT	0x00000000000000
		CBOPT	3:0	W	Channel bonding option 0: no bonding 3: Bonding 0/1/2 3/4/5 other values: reserved	0x0
...						
0x2F50	0,1	DECODING_LINK_11_CB				
		DESKEWED	61:4	R	Every bit corresponds to an E-link on one (lp)GBT frame. Register indicates whether the E-link has been de-skewed in the channel. E-link are grouped in a channel according to CBOPT	0x00000000000000
		CBOPT	3:0	W	Channel bonding option 0: no bonding 3: Bonding 0/1/2 3/4/5 other values: reserved	0x0
0x2F60	0,1	DECODING_MASK64B66BKBLOCK	3:0	W	Mask User K-Block based on its block number (see sp011)	0xA
0x2F70	0,1	DECODING_DISEGROUP	6:0	W	Disable egroups for debugging purposes	0x0
0x2F80	0,1	FULLMODE_32B_SOP	0	W	When set to 1, use 32-bit 0x0000003C as start of chunk, otherwise only 8-bit 0x3C (FULL mode only)	0x0
0x2F90	0,1	DECODING_HGTD_ALTIROC	0	W	Set to 1 to use HGTD Altiroc K characters in the 8b10b decoders (LPGBT firmware mode)	0x0
0x2FA0	0,1	DECODING_HGTD_LUMI_CONF				
		DEBUG_DATASOURCE	36	W	enable local data source for debugging	0x0
		RAW_MODE	35	W	enable RAW mode (just forwarding 6b8b data)	0x0
		LHC_TURNS	34:25	W	number of LHC turns to aggregate	0x64
		TRIG_LAT	24:16	W	trigger latency for per-event luminosity	0x00
		SYNC_WORD	15:0	W	sync word for luminosity stream	0x4778
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	0x1
ENCODING_EGROUP_CTRL_GEN						
ENCODING_EGROUP						
0x3010	0,1	ENCODING_LINK00_EGROUP0_CTRL				



		ENABLE_DELAY	63	W	Enable inter-packet delay generation in HDLC encoder	0x0
		TTC_OPTION	62:59	W	Selects TTC bits sent to the E-link	0x0
		EPATH_ALMOST_FULL	58:51	R	Indiator that the EPATH FIFO is almost full	0x00
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath	0x00
		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	0x0
		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	0x11111111
		EPATH_ENA	7:0	W	Enable bits per E-PATH	0x00
...						
0x3050	0,1	ENCODING_LINK00_EGROUP4_CTRL				
		ENABLE_DELAY	63	W	Enable inter-packet delay generation in HDLC encoder	0x0
		TTC_OPTION	62:59	W	Selects TTC bits sent to the E-link	0x0
		EPATH_ALMOST_FULL	58:51	R	Indiator that the EPATH FIFO is almost full	0x00
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath	0x00
		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	0x0
		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	0x11111111
		EPATH_ENA	7:0	W	Enable bits per E-PATH	0x00
...						
ENCODING_EGROUP						
0x3380	0,1	ENCODING_LINK11_EGROUP0_CTRL				
		ENABLE_DELAY	63	W	Enable inter-packet delay generation in HDLC encoder	0x0
		TTC_OPTION	62:59	W	Selects TTC bits sent to the E-link	0x0
		EPATH_ALMOST_FULL	58:51	R	Indiator that the EPATH FIFO is almost full	0x00
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath	0x00

		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	0x0
		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	0x11111111
		EPATH_ENA	7:0	W	Enable bits per E-PATH	0x00
...						
0x33C0	0,1	ENCODING_LINK11_EGROUP4_CTRL				
		ENABLE_DELAY	63	W	Enable inter-packet delay generation in HDLC encoder	0x0
		TTC_OPTION	62:59	W	Selects TTC bits sent to the E-link	0x0
		EPATH_ALMOST_FULL	58:51	R	Indicator that the EPATH FIFO is almost full	0x00
		REVERSE_ELINKS	50:43	W	enables bit reversing for the elink in the given epath	0x00
		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	0x0
		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	0x11111111
		EPATH_ENA	7:0	W	Enable bits per E-PATH	0x00
MINI_EGROUP_FROMHOST_GEN						
0x33D0	0,1	MINI_EGROUP_FROMHOST_00				
		AUX_ENCODING	17:14	W	Configures encoding of the AUX channel	0x2
		ENABLE_DELAY	13	W	Enable inter-packet delay generation in HDLC encoder	0x0
		AUX_ALMOST_FULL	12	R	Indicator that the AUX Path FIFO is almost full	0x0
		AUX_BIT_SWAPPING	11	W	0: two input bits of AUX e-link are as documented, 1: two input bits are swapped	0x1
		AUX_ENABLE	10	W	Enables the AUX channel	0x1
		IC_ALMOST_FULL	9	R	Indicator that the IC Path FIFO is almost full	0x0
		IC_BIT_SWAPPING	8	W	0: two input bits of IC e-link are as documented, 1: two input bits are swapped	0x0
		IC_ENABLE	7	W	Enables the IC channel	0x1
		EC_ALMOST_FULL	6	R	Indicator that the EC Path FIFO is almost full	0x0

		EC_BIT_SWAPPING	5	W	0: two output bits of EC e-link are as documented, 1: two output bits are swapped	0x0
		EC_ENCODING	4:1	W	Configures encoding of the EC channel	0x2
		EC_ENABLE	0	W	Configures the FromHost Mini egroup	0x1
...						
0x3540	0,1	MINI_EGROUP_FROMHOST_23				
		AUX_ENCODING	17:14	W	Configures encoding of the AUX channel	0x2
		ENABLE_DELAY	13	W	Enable inter-packet delay generation in HDLC encoder	0x0
		AUX_ALMOST_FULL	12	R	Indicator that the AUX Path FIFO is almost full	0x0
		AUX_BIT_SWAPPING	11	W	0: two input bits of AUX e-link are as documented, 1: two input bits are swapped	0x1
		AUX_ENABLE	10	W	Enables the AUX channel	0x1
		IC_ALMOST_FULL	9	R	Indicator that the IC Path FIFO is almost full	0x0
		IC_BIT_SWAPPING	8	W	0: two input bits of IC e-link are as documented, 1: two input bits are swapped	0x0
		IC_ENABLE	7	W	Enables the IC channel	0x1
		EC_ALMOST_FULL	6	R	Indicator that the EC Path FIFO is almost full	0x0
		EC_BIT_SWAPPING	5	W	0: two output bits of EC e-link are as documented, 1: two output bits are swapped	0x0
		EC_ENCODING	4:1	W	Configures encoding of the EC channel	0x2
		EC_ENABLE	0	W	Configures the FromHost Mini egroup	0x1
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	0x0
		MANCHESTER_ENABLE1	8	W	enable manchester encoding	0x0
		AUTOMATIC_MERGE_DISABLE1	7	W	Disable automatic merging	0x0
		TTC_SELECT1	6	W	TTC/FromHost select (if automatic merging is disabled)	0x0
		PHASE_DELAY0	5:3	W	phase delay of output data, with 320 Bb/s e-link 8 phases per BC	0x0
		MANCHESTER_ENABLE0	2	W	enable manchester encoding	0x0
		AUTOMATIC_MERGE_DISABLE0	1	W	Disable automatic merging	0x0
		TTC_SELECT0	0	W	TTC/FromHost select (if automatic merging is disabled)	0x0
...						
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	0x0
		MANCHESTER_ENABLE1	8	W	enable manchester encoding	0x0
		AUTOMATIC_MERGE_DISABLE1	7	W	Disable automatic merging	0x0
		TTC_SELECT1	6	W	TTC/FromHost select (if automatic merging is disabled)	0x0
		PHASE_DELAY0	5:3	W	phase delay of output data, with 320 Bb/s e-link 8 phases per BC	0x0
		MANCHESTER_ENABLE0	2	W	enable manchester encoding	0x0
		AUTOMATIC_MERGE_DISABLE0	1	W	Disable automatic merging	0x0

		TTC_SELECT0	0	W	TTC/FromHost select (if automatic merging is disabled)	0x0
...						
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	0x0
		MANCHESTER_ENABLE1	8	W	enable manchester encoding	0x0
		AUTOMATIC_MERGE_DISABLE1	7	W	Disable automatic merging	0x0
		TTC_SELECT1	6	W	TTC/FromHost select (if automatic merging is disabled)	0x0
		PHASE_DELAY0	5:3	W	phase delay of output data, with 320 Bb/s e-link 8 phases per BC	0x0
		MANCHESTER_ENABLE0	2	W	enable manchester encoding	0x0
		AUTOMATIC_MERGE_DISABLE0	1	W	Disable automatic merging	0x0
		TTC_SELECT0	0	W	TTC/FromHost select (if automatic merging is disabled)	0x0
...						
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	0x0
		MANCHESTER_ENABLE1	8	W	enable manchester encoding	0x0
		AUTOMATIC_MERGE_DISABLE1	7	W	Disable automatic merging	0x0
		TTC_SELECT1	6	W	TTC/FromHost select (if automatic merging is disabled)	0x0
		PHASE_DELAY0	5:3	W	phase delay of output data, with 320 Bb/s e-link 8 phases per BC	0x0
		MANCHESTER_ENABLE0	2	W	enable manchester encoding	0x0
		AUTOMATIC_MERGE_DISABLE0	1	W	Disable automatic merging	0x0
		TTC_SELECT0	0	W	TTC/FromHost select (if automatic merging is disabled)	0x0
YARR_DEBUG_ALLEGROUP_FROMHOST_GEN						
0x3910	0,1	YARR_DEBUG_ALLEGROUP_FROMHOST1_00				
		RD53A_AZ_EN	48	W	Auto zeroing module enable	0x0
		CNT_TRIG_CMD	47:16	R	Number of issued triggers via cmd	0x00000000
		ERR_GENCALTRIG_DLY	15:8	R	Number of mismatches between CNT_GENCALTRIG_DLY and REF_DLY_GENCALTRIG	0x00
		REF_DLY_GENCALTRIG	7:0	W	Reference distance between GenCal and First Trigger	0x0F
0x3920	0,1	YARR_DEBUG_ALLEGROUP_FROMHOST2_00				
		CNT_CMD	47:16	R	Number of issued commands	0x00000000
		REF_CMD	15:0	W	Cmd type to be counted. See RD53 Manual for list of allowed commands	0x6666
...						
0x3A70	0,1	YARR_DEBUG_ALLEGROUP_FROMHOST1_11				
		RD53A_AZ_EN	48	W	Auto zeroing module enable	0x0
		CNT_TRIG_CMD	47:16	R	Number of issued triggers via cmd	0x00000000
		ERR_GENCALTRIG_DLY	15:8	R	Number of mismatches between CNT_GENCALTRIG_DLY and REF_DLY_GENCALTRIG	0x00
		REF_DLY_GENCALTRIG	7:0	W	Reference distance between GenCal and First Trigger	0x0F

0x3A80	0,1	YARR_DEBUG_ALLEGROUP_FROMHOST2_11				
		CNT_CMD	47:16	R	Number of issued commands	0x00000000
		REF_CMD	15:0	W	Cmd type to be counted. See RD53 Manual for list of allowed commands	0x6666
0x3A90	0,1	YARR_FROMHOST_CALTRIGSEQ_WE	0	W	enable to store CalPulse+Trigger Sequence into memory	0x0
0x3AA0	0,1	YARR_FROMHOST_CALTRIGSEQ_WRDATA	15:0	W	CalPulse+Trigger Sequence to be stored in memory	0x0000
0x3AB0	0,1	YARR_FROMHOST_CALTRIGSEQ_WRADDR	4:0	W	memory address to store CalPulse+Trigger Sequence	0x0
0x3AC0	0,1	HGTD_ALTIROC_FASTCMD				
		ALTIROC3_IDLE	14	W	0 for ALTIROC2 10101100, 1 for ALTIROC3 11110000	0x0
		USE_CAL	13	W	When set to 1, CAL will be sent on L1A, then after TRIG_DELAY BC clocks a TRIGGER. When 0, TRIGGER will be sent on L1A.	0x1
		SYNCLUMI	12	W	Set to 1 to trigger a SYNCLUMI command, rising edge of this bit. Clear in software	0x0
		GBRST	11	W	Set to 1 to trigger a GBRST command, rising edge of this bit. Clear in software	0x0
		TRIG_DELAY	10:0	W	Number of BC clocks between CAL and TRIGGER command if USE_CAL is set to 1	0x05
0x3AD0	0,1	ITKSTRIP_LCB_R3L1_ELINK_SWAP	47:0	W	Setting a bit, moves the LCB E-Link to the odd E-Link position and R3L1 to the even one on the IpGBT downlink. 4 bits per IpGBT link	0x000000000000
0x3AE0	0,1	ENCODING_ITKPIX_TRIGGER_GENERATOR				
		NO_INJECT	28	W	Controls the trigger generator for ItkPix	0x0
		EDGE_MODE	27	W	Controls the trigger generator for ItkPix	0x1
		EDGE_DELAY	26:22	W	Controls the trigger generator for ItkPix	0x0
		EDGE_DURATION	21:14	W	Controls the trigger generator for ItkPix	0x14
		TRIG_DELAY	13:6	W	Controls the trigger generator for ItkPix	0x3A
		TRIG_MULTIPLIER	5:0	W	Controls the trigger generator for ItkPix	0x10
0x3AF0	0,1	LTI_FE_OUTPUT_SELECTOR	1:0	W	0: Low latency LTI-FE distribution 1: 40 MHz sync LTI-FE distribution	0x0
Frontend Emulator Controls And Monitors						
0x4000	0,1	FE_EMU_ENA				
		EMU_TOFRONTEND	1	W	Enable GBT dummy emulator ToFrontEnd	0x0
		EMU_TOHOST	0	W	Enable GBT dummy emulator ToHost	0x0
0x4010	0,1	FE_EMU_CONFIG				
		WE	54:47	W	write enable array, every bit is one emulator RAM block	0x00
		WRADDR	46:33	W	write address bus	0x000
		WRDATA	32:0	W	write data bus	0x00000000
0x4020	0,1	FE_EMU_READ				
		SEL	35:33	W	Select ramblock to read back	0x0

		DATA	32:0	R	Read back ramblock at FE_EMU_CONFIG.WRADDR	0x00000000
0x4030	0,1	FE_EMU_LOGIC				
		L1A_TRIGGERED	33	W	1 Send a chunk on every L1A, 0 use the IDLES to determine the rate	0x0
		ENA	32	W	Enable logic based FrontEnd emulator, instead of RAM based.	0x0
		IDLES	31:16	W	Number of IDLE bytes between chunks.	0x0000
		CHUNK_LENGTH	15:0	W	Chunk length in bytes	0x0000
DECODING_BCM_PRIME_L1_A_CONTROLS_GEN						
0x4200	0,1	DECODING_BCM_PRIME_LINK_00_L1A				
		DELAY	9:5	W	The data in fiber is delayed N clock cycles to match with TTC L1A	0x5
		WINDOW	4:0	W	The L1A signal is extended to cover multiple BCID's	0x5
...						
0x4370	0,1	DECODING_BCM_PRIME_LINK_23_L1A				
		DELAY	9:5	W	The data in fiber is delayed N clock cycles to match with TTC L1A	0x5
		WINDOW	4:0	W	The L1A signal is extended to cover multiple BCID's	0x5
0x4380	0,1	DECODING_BCM_PRIME_ONLY_L1A	0	W	If enabled, the BCM_PRIME firmware, will only readout data when an L1A is sent.	0x0
0x4390	0,1	DECODING_BCM_PRIME_EMU_BCID	0	W	If enabled, the BCM_PRIME firmware will use internally generated BCIDs instead of the TTC one.	0x0
0x43A0	0,1	DECODING_BCM_PRIME_PUBLISH_ZEROS	0	W	If enabled, the BCM_PRIME firmware publish empty data-events if they are matched with L1A	0x0
Link Wrapper Controls						
0x5000	0	LINK_FULLMODE_LTI	23:0	W	Set to 1 to enable LTI format TTC distribution (8b10b at 9.6Gb) in the FULLMODE flavour, one bit per channel. Set to 0 for 4.8Gb GBT distribution	0x000000
0x5400	0	GBT_CHANNEL_DISABLE	47:0	W	Disable selected lpGBT, GBT or FULL mode channel	0x000000000000
0x5410	0	GBT_GENERAL_CTRL	63:0	W	Alignment chk reset (not self clearing)	0x0000000000000000
0x5420	0	GBT_MODE_CTRL				
		RX_ALIGN_TB_SW	2	W	RX_ALIGN_TB_SW	0x0
		RX_ALIGN_SW	1	W	RX_ALIGN_SW	0x0
		DESMUX_USE_SW	0	W	DESMUX_USE_SW	0x0
0x5480	0	GBT_RXSLIDE_SELECT	47:0	W	RxSlide select [47:0]	0x000000000000
0x5490	0	GBT_RXSLIDE_MANUAL	47:0	W	RxSlide select [47:0]	0x000000000000
0x54A0	0	GBT_TXUSRDRDY	47:0	W	TxUsrRdy [47:0]	0xFFFFFFFFFFFF
0x54B0	0	GBT_RXUSRDRDY	47:0	W	RxUsrRdy [47:0]	0xFFFFFFFFFFFF
0x54C0	0	GBT_SOFT_RESET	47:0	W	SOFT_RESET [47:0]	0x000000000000

0x54D0	0	GBT_GTTX_RESET	47:0	W	GTTX_RESET [47:0]	0x000000000000
0x54E0	0	GBT_GTRX_RESET	47:0	W	GTRX_RESET [47:0]	0x000000000000
0x54F0	0	GBT_PLL_RESET				
		QPLL_RESET	59:48	W	QPLL_RESET [11:0]	0x000
		CPLL_RESET	47:0	W	CPLL_RESET [47:0]	0x000000000000
0x5500	0	GBT_SOFT_TX_RESET				
		RESET_ALL	59:48	W	SOFT_TX_RESET_ALL [11:0]	0x000
		RESET_GT	47:0	W	SOFT_TX_RESET_GT [47:0]	0x000000000000
0x5510	0	GBT_SOFT_RX_RESET				
		RESET_ALL	59:48	W	SOFT_TX_RESET_ALL [11:0]	0x000
		RESET_GT	47:0	W	SOFT_TX_RESET_GT [47:0]	0x000000000000
0x5520	0	GBT_ODD_EVEN	47:0	W	OddEven [47:0]	0x000000000000
0x5530	0	GBT_TOPBOT	47:0	W	TopBot [47:0]	0x000000000000
0x5540	0	GBT_TX_TC_DLY_VALUE1	47:0	W	TX_TC_DLY_VALUE [47:0]	0x333333333333
0x5550	0	GBT_TX_TC_DLY_VALUE2	47:0	W	TX_TC_DLY_VALUE [95:48]	0x333333333333
0x5560	0	GBT_TX_TC_DLY_VALUE3	47:0	W	TX_TC_DLY_VALUE [143:96]	0x333333333333
0x5570	0	GBT_TX_TC_DLY_VALUE4	47:0	W	TX_TC_DLY_VALUE [191:144]	0x333333333333
0x5580	0	GBT_DATA_TXFORMAT1	47:0	W	DATA_TXFORMAT [47:0]	0x000000000000
0x5590	0	GBT_DATA_TXFORMAT2	47:0	W	DATA_TXFORMAT [95:48]	0x000000000000
0x55A0	0	GBT_DATA_RXFORMAT1	47:0	W	DATA_RXFORMAT [47:0]	0x000000000000
0x55B0	0	GBT_DATA_RXFORMAT2	47:0	W	DATA_RXFORMAT [95:0]	0x000000000000
0x55C0	0	GBT_TX_RESET	47:0	W	TX Logic reset [47:0]	0x000000000000
0x55D0	0	GBT_RX_RESET	47:0	W	RX Logic reset [47:0]	0x000000000000
0x55E0	0	GBT_TX_TC_METHOD	47:0	W	TX time domain crossing method [47:0]	0x000000000000
0x55F0	0	GBT_OUTMUX_SEL	47:0	W	Descrambler output MUX selection [47:0]	0x000000000000
0x5600	0	GBT_TC_EDGE	47:0	W	Sampling edge selection for TX domain crossing [47:0]	0x000000000000
0x5610	0	GBT_TXPOLARITY	47:0	W	0: default polarity 1: reversed polarity for transmitter of GTH channels	0x000000000000
0x5620	0	GBT_RXPOLARITY	47:0	W	0: default polarity 1: reversed polarity for the receiver of the GTH channels	0x000000000000
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	0x0

0x5640	0	LPGBT_FEC	47:0	W	0: FEC5 1: FEC12	0x000000000000
0x5650	0	LPGBT_DATARATE	47:0	W	0: 10.24 Gbps 1: 5.12 Gbps	0x000000000000
0x5700	0	GBT_TOHOST_FANOUT				
		LOCK	48	W	Locks this particular register. If set prevents software from touching it.	0x0
		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	0x000000000000
0x5710	0	GBT_TOFRONTEND_FANOUT				
		LOCK	48	W	Locks this particular register. If set prevents software from touching it.	0x0
		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	0x000000000000
0x5720	0	FULLMODE_AUTO_RX_RESET				
		ENABLE	32	W	Enable the Automatic RX Reset mechanism	0x1
		TIMEOUT	31:0	W	Number of 40 MHz clock cycles until an unaligned link results in a reset pulse	0x00100000
TCLINK_CNTRL_GEN						
0x5730	0	TCLINK_CONTROL_00				
		OFFSET_ERROR	63:16	W	Error-offset for phase-control Recommended to freeze with an initial value read	0x000000000000
		CLOSE_LOOP	15	W	Close TCLink loop (enables compensation)	0x0
		TX_PI_PHASE_CALIB	14:8	W	UI alignment Tx PI calibrated phase	0x0
		TX_UI_ALIGN_CALIB	7	W	UI alignment Tx PI activate	0x0
		TX_FINE_REALIGN	6	W	Repeats fine alignment procedure	0x0
		PS_STROBE	5	W	Shifts phase of transmitter serial data	0x0
		PS_INC_NDEC	4	W	Shifts phase of transmitter serial data	0x0
		MASTER_MGT_RX_READY	3	W	MGT rx is ready (used as reset)	0x0
		...				
0x58A0	0	TCLINK_CONTROL_23				
		OFFSET_ERROR	63:16	W	Error-offset for phase-control Recommended to freeze with an initial value read	0x000000000000
		CLOSE_LOOP	15	W	Close TCLink loop (enables compensation)	0x0
		TX_PI_PHASE_CALIB	14:8	W	UI alignment Tx PI calibrated phase	0x0
		TX_UI_ALIGN_CALIB	7	W	UI alignment Tx PI activate	0x0
		TX_FINE_REALIGN	6	W	Repeats fine alignment procedure	0x0
		PS_STROBE	5	W	Shifts phase of transmitter serial data	0x0
		PS_INC_NDEC	4	W	Shifts phase of transmitter serial data	0x0
		MASTER_MGT_RX_READY	3	W	MGT rx is ready (used as reset)	0x0



Link Wrapper Monitors						
0x6600	0	GBT_VERSION				
		DATE	63:48	R	Date	0x0000
		GBT_VERSION	47:32	R	GBT Version	0x0000
		GTH_IP_VERSION	31:16	R	GTH IP Version	0x0000
		RESERVED	15:3	R	Reserved	0x000
		GTHREFCLK_SEL	2	R	GTHREFCLK SEL	0x0
		RX_CLK_SEL	1	R	RX CLK SEL	0x0
		PLL_SEL	0	R	PLL SEL	0x0
0x6680	0	GBT_TXRESET_DONE	47:0	R	TX Reset done [47:0]	0x000000000000
0x6690	0	GBT_RXRESET_DONE	47:0	R	RX Reset done [47:0]	0x000000000000
0x66A0	0	GBT_TXFSMRESET_DONE	47:0	R	TX FSM Reset done [47:0]	0x000000000000
0x66B0	0	GBT_RXFSMRESET_DONE	47:0	R	RX FSM Reset done [47:0]	0x000000000000
0x66C0	0	GBT_CPLL_FBCLK_LOST	47:0	R	CPLL FBCLK LOST [47:0]	0x000000000000
0x66D0	0	GBT_PLL_LOCK				
		QPLL_LOCK	59:48	R	QPLL LOCK [11:0]	0x000
		CPLL_LOCK	47:0	R	CPLL LOCK [47:0]	0x000000000000
0x66E0	0	GBT_RXCDR_LOCK	47:0	R	RX CDR LOCK [47:0]	0x000000000000
0x66F0	0	GBT_CLK_SAMPLED	47:0	R	clk sampled [47:0]	0x000000000000
0x6700	0	GBT_RX_IS_HEADER	47:0	R	RX IS HEADER [47:0]	0x000000000000
0x6710	0	GBT_RX_IS_DATA	47:0	R	RX IS DATA [47:0]	0x000000000000
0x6720	0	GBT_RX_HEADER_FOUND	47:0	R	RX HEADER FOUND [47:0]	0x000000000000
0x6730	0	GBT_ALIGNMENT_DONE	47:0	R	RX ALIGNMENT DONE [47:0]	0x000000000000
0x6740	0	GBT_OUT_MUX_STATUS	47:0	R	GBT output mux status [47:0]	0x000000000000
0x6750	0	GBT_ERROR	47:0	R	Error flags [47:0]	0x000000000000
0x6760	0	GBT_GBT_TOPBOT_C	47:0	R	TopBot_c [47:0]	0x000000000000
0x6800	0	GBT_FM_RX_DISP_ERROR1	47:0	R	Rx disparity error [47:0]	0x000000000000
0x6810	0	GBT_FM_RX_DISP_ERROR2	47:0	R	Rx disparity error [96:48]	0x000000000000
0x6820	0	GBT_FM_RX_NOTINTABLE1	47:0	R	Rx not in table [47:0]	0x000000000000
0x6830	0	GBT_FM_RX_NOTINTABLE2	47:0	R	Rx not in table [96:48]	0x000000000000
GT_FEC_ERR_CNT_GEN						
0x6840	0	GT_FEC_ERR_CNT_00	31:0	R	Counts the number of FEC errors in the given channel.	0x00000000
...						
0x69B0	0	GT_FEC_ERR_CNT_23	31:0	R	Counts the number of FEC errors in the given channel.	0x00000000
GT_AUTO_RX_RESET_CNT_GEN						
0x69C0	0	GT_AUTO_RX_RESET_CNT_00				

		CLEAR VALUE	any 31:0	T R	Any write to this register clears the counter value Counts the number of AUTO RX RESET events that happend on the FULLMODE, GBT or lpGBT link	0x0 0x00000000
...						
0x6B30	0	GT_AUTO_RX_RESET_CNT_23				
		CLEAR VALUE	any 31:0	T R	Any write to this register clears the counter value Counts the number of AUTO RX RESET events that happend on the FULLMODE, GBT or lpGBT link	0x0 0x00000000
TCLINK_MON_GEN						
0x6B40	0	TCLINK_MONITOR_1_00				
		ERROR_CONTROLLER	62:15	R	Error-signal for controller Signed complement 2 number.	0x000000000000
		LOOP_CLOSED	14	R	TCLink loop is closed (compensation is enabled)	0x0
		TX_ALIGNED	13	R	Transmitter alignment procedure finished Use as reset for transmitter user logic	0x0
		PS_DONE	12	R	Phase shift is done	0x0
		TX_PI_PHASE	11:5	R	Tx PI phase after alignment	0x0
0x6B50	0	TCLINK_MONITOR_2_00				
		PHASE_DETECTOR	63:32	R	Phase detector response	0x00000000
		TX_FIFO_FILL_PD	31:0	R	Phase detector current value	0x00000000
0x6B60	0	TCLINK_MONITOR_3_00				
		LOOP_NOT_CLOSED_REASON	58:54	R	Reason why the TCLink loop is not closed	0x0
		PHASE_ACC	53:38	R	phase accumulated output (integrated output)	0x0000
		OPERATION_ERROR	37	R	error output indicating that a clk_en_i pulse has arrived before the done_i signal arrived from the previous strobe_o request	0x0
		DEBUG_TESTER_ADDR_READ	36:27	W	read address for reading stocked TCLink phase accumulated results	0x00
		DEBUG_TESTER_DATA_READ	26:11	R	data of stocked TCLink phase accumulated results	0x0000
		PS_PHASE_STEP	10:7	R	number of units to shift the phase of the receiver clock	0x0
...						
0x6F90	0	TCLINK_MONITOR_1_23				
		ERROR_CONTROLLER	62:15	R	Error-signal for controller Signed complement 2 number.	0x000000000000
		LOOP_CLOSED	14	R	TCLink loop is closed (compensation is enabled)	0x0
		TX_ALIGNED	13	R	Transmitter alignment procedure finished Use as reset for transmitter user logic	0x0
		PS_DONE	12	R	Phase shift is done	0x0
		TX_PI_PHASE	11:5	R	Tx PI phase after alignment	0x0
0x6FA0	0	TCLINK_MONITOR_2_23				
		PHASE_DETECTOR	63:32	R	Phase detector response	0x00000000
		TX_FIFO_FILL_PD	31:0	R	Phase detector current value	0x00000000
0x6FB0	0	TCLINK_MONITOR_3_23				
		LOOP_NOT_CLOSED_REASON	58:54	R	Reason why the TCLink loop is not closed	0x0
		PHASE_ACC	53:38	R	phase accumulated output (integrated output)	0x0000

		OPERATION_ERROR	37	R	error output indicating that a clk_en_i pulse has arrived before the done_i signal arrived from the previous strobe_o request	0x0
		DEBUG_TESTER_ADDR_READ	36:27	W	read address for reading stocked TCLink phase accumulated results	0x00
		DEBUG_TESTER_DATA_READ	26:11	R	data of stocked TCLink phase accumulated results	0x0000
		PS_PHASE_STEP	10:7	R	number of units to shift the phase of the receiver clock	0x0
0x6FC0	0	GBT_PLL_LOL_LATCHED				
		CLEAR	any	T	Any write to this bitfield clears the latched LOL bits	0x0
		QPLL_LOL_LATCHED	59:48	R	Asserted when CPLL lock is lost, clear by writing to CLEAR	0x000
		CPLL_LOL_LATCHED	47:0	R	Asserted when CPLL lock is lost, clear by writing to CLEAR	0x000000000000
0x6FD0	0	GBT_ALIGNMENT_LOST				
		CLEAR	any	T	Any write to this bitfield clears the latched ALIGNMENT_LOST bits	0x0
		ALIGNMENT_LOST	47:0	R	Asserted when GBT_ALIGNMENT_DONE bit is 0, clear by writing to CLEAR	0x000000000000
TTCBUSY Controls And Monitors						
TTC_DEC_CTRLMON						
0x7000	0	TTC_DEC_CTRL				
		B_CHAN_DELAY	30:27	W	Number of BC to delay the L1A distribution to the frontends	0x0
		BCID_ONBCR	26:15	W	BCID is set to this value when BCR arrives	0x000
		BUSY_OUTPUT_STATUS	14	R	Actual status of the BUSY LEMO output signal	0x0
		ECR_BCR_SWAP	13	W	ECR and BCR signals are swapped at the output of the TTC decoder (needed only for LAr TTC)	0x0
		BUSY_OUTPUT_INHIBIT	12	W	forces the Busy LEMO output to BUSY-OFF	0x0
		TOHOST_RST	11	W	reset toHost in ttc decoder	0x0
		TT_BCH_EN	10	W	trigger type enable / disable for TTC-ToHost	0x0
		XL1ID_SW	9:2	W	set XL1ID value, the value to be set by XL1ID_RST signal	0x00
		XL1ID_RST	1	W	giving a trigger signal to reset XL1ID value	0x0
		MASTER_BUSY	0	W	L1A trigger throttling	0x0
0x7010	0	TTC_DEC_MON				
		TH_FF_COUNT	15:5	R	ToHostData Fifo counts	0x00
		TH_FF_FULL	4	R	ToHostData Fifo status 1:full 0:not full	0x0
		TH_FF_EMPTY	3	R	ToHostData Fifo status 1:empty 0:not empty	0x0
		TTC_BIT_ERR	2:0	R	double bit, single bit and comm error in TTC data	0x0
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	0x00000000000000
...						
0x7190	0,1	TTC_BUSY_ACCEPTED23	56:0	R	busy has been asserted by the given ELINK. Reset by writing to TTC_BUSY_CLEAR	0x00000000000000
0x71A0	0	TTC_EMU				
		FULL	2	R	TTC Emulator memory full indication	0x0
		SEL	1	W	Select TTC data source 1 TTC Emu   0 TTC Decoder	0x0

		ENA	0	W	Clear to load into the TTC emulator's memory the required sequence, Set to run the TTC emulator sequence	0x0
0x71B0	0	TTC_DELAY	3:0	W	Controls the TTC Fanout delay value, in 25ns (1BC) units	0x0
0x74B0	0	TTC_BUSY_TIMING_CTRL				
		PRESCALE	51:32	W	Prescales the 40MHz clock to create an internal slow clock	0x0000F
		BUSY_WIDTH	31:16	W	Minimum number of 40MHz clocks that the busy is asserted	0x000F
		LIMIT_TIME	15:0	W	Number of prescaled clocks a given busy must be asserted before it is recognized	0x000F
0x74C0	0	TTC_BUSY_CLEAR	any	T	clears the latching busy bits in TTC_BUSY_ACCEPTED	0x0
0x74D0	0	TTC_EMU_CONTROL				
		BUSY_IN_ENABLE	33	W	Enable internal BUSY input to stop L1A on BUSY	0x1
		BROADCAST	32:27	W	Broadcast data	0x0
		ECR	26	W	Event counter reset	0x0
		BCR	25	W	Bunch counter reset	0x0
		L1A	24	W	Level 1 Accept	0x0
0x74E0	0	TTC_EMU_L1A_PERIOD	31:0	W	L1A period in BC. 0 means manual L1A with TTC_EMU_CONTROL.L1A	0x00000000
0x74F0	0	TTC_EMU_ECR_PERIOD	31:0	W	ECR period in BC. 0 means manual ECR with TTC_EMU_CONTROL.ECR	0x00000000
0x7500	0	TTC_EMU_BCR_PERIOD	31:0	W	BCR period in BC. 0 means manual BCR with TTC_EMU_CONTROL.BCR	0x00000DEC
0x7510	0	TTC_EMU_LONG_CHANNEL_DATA	31:0	W	Long channel data for the TTC emulator	0x00000000
0x7520	0	TTC_EMU_RESET	any	T	Any write to this register resets the TTC Emulator to the default state.	0x0
0x7530	0	TTC_L1ID_MONITOR	31:0	R	Monitor L1ID and XL1ID.	0x00000000
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	0x0
		VALUE	31:0	R	Counts the number of ECRs received from the TTC system, any write to this register clears the counter	0x00000000
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	0x0
		VALUE	31:0	R	Counts the number of TType received from the TTC system, any write to this register clears the counter	0x00000000
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	0x0
		VALUE	31:0	R	Counts the number of times the BCR period does not match 3564, any write to this register clears the counter	0x00000000
0x7570	0	TTC_BCR_COUNTER				
		CLEAR	any	T	Counts the number of times BCR is issued, any write to this register clears the counter	0x0
		VALUE	31:0	R	Counts the number of times BCR is issued, any write to this register clears the counter	0x00000000
0x7580	0	TTC_EMU_TP_DELAY	31:0	W	Number of BC that the testpulse should be sent before the L1A, 0 means no test pulse is sent	0x00000040
0x7590	0	TTC_L1A_DELAY	5:0	W	In Phase1 the L0A bit is generated from L1A, but with a variable delay between 0 and 63 BC cycles from L0A to L1A	0x0
0x75A0	0	TTC_CDRLOCK_MONITOR				
		CLEAR	any	T	Clears the latching cdrlock, LOL and LOS bitfields	0x0

		CDRLOCK_LOST	5	R	asserted when CDRLOCKED has been 0, Clear by writing to CLEAR bitfield	0x0
		CDRLOCKED	4	R	Set to 1 if the clock can be successfully recovered from the TTC signal	0x0
		ADN_LOL_LATCHED	3	R	Latched Loss of lock from ADN2814, Clear by writing to CLEAR bitfield	0x0
		ADN_LOS_LATCHED	2	R	Latched Loss of signal from ADN2814, Clear by writing to CLEAR bitfield	0x0
		ADN_LOL	1	R	Loss of lock from ADN2814	0x0
		ADN_LOS	0	R	Loss of signal from ADN2814	0x0
0x75B0	0	TTC_ASYNCUSERDATA				
		WR_EN	any	T	Any write to this registers triggers a FIFO write into AsyncUserData	0x0
		DATA	63:0	W	Write AsyncUserData to the LTI-FE link if legacy TTC or TTC Emulator are selected	0x0000000000000000
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	0xB
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	0xB
0x8020	0, 1	XOFF_FM_LOW_THRESH_CROSSED	23:0	R	FIFO filled beyond the low threshold, 1 bit per channel	0x000000
0x8030	0, 1	XOFF_FM_HIGH_THRESH				
		CLEAR_LATCH	any	T	Writing this register will clear all CROSS_LATCHED bits	0x0
		CROSS_LATCHED	47:24	R	FIFO filled beyond the high threshold, 1 latch bit per channel	0x000000
		CROSSED	23:0	R	FIFO filled beyond the high threshold, 1 bit per channel	0x000000
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	0x000000
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	0x000000
0x8060	0, 1	DMA_BUSY_STATUS				
		CLEAR_LATCH	any	T	Any write to this register clears TOHOST_BUSY_LATCHED	0x0
		ENABLE	4	W	Enable the DMA buffer on the server as a source of busy	0x0
		TOHOST_BUSY_LATCHED	3	R	A tohost descriptor has passed BUSY_THRESHOLD_ASSERT in the past, busy flag was set	0x0
		TOHOST_BUSY	0	R	A tohost descriptor passed BUSY_THRESHOLD_ASSERT, busy flag set	0x0
0x8070	0, 1	FM_BUSY_CHANNEL_STATUS				
		CLEAR_LATCH	any	T	Any write to this register will clear the BUSY_LATCHED bits	0x0
		BUSY_LATCHED	47:24	R	one Indicates that the given FULL mode channel has received BUSY-ON	0x000000
		BUSY	23:0	R	one Indicates that the given FULL mode channel is currently in BUSY state	0x000000

0x8080	0,1	BUSY_MAIN_OUTPUT_FIFO_THRESH				
		BUSY_ENABLE	24	W	Enable busy generation if thresholds are crossed	0x0
		LOW	23:12	W	Low, Negate threshold of busy generation from main output fifo	0x3FF
		HIGH	11:0	W	High, Assert threshold of busy generation from main output fifo	0x4FF
0x8090	0,1	BUSY_MAIN_OUTPUT_FIFO_STATUS				
		CLEAR_LATCHED	any	T	Any write to this register will clear the	0x0
		HIGH_THRESH_CROSSED_LATCHED	2	R	Main output fifo has been full beyond HIGH THRESHOLD, write to clear	0x0
		HIGH_THRESH_CROSSED	1	R	Main output fifo is full beyond HIGH THRESHOLD	0x0
		LOW_THRESH_CROSSED	0	R	Main output fifo is full beyond LOW THRESHOLD	0x0
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	0x0000000000000000
...						
0x8210	0	ELINK_BUSY_ENABLE23	56:0	W	Per elink (and FULL mode link) enable of the busy signal towards the LEMO output	0x0000000000000000
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	0x0000000000000000
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	0x0000000000000000
0x8240	0,1	XOFF_COUNT00	63:0	R	Total number of XOFF events per channel that occurred since a reset.	0x0000000000000000
...						
0x8670	0,1	XOFF_PEAK_DURATION23	63:0	R	Maximum occurred duration of XOFF on the given channel in 25ns bins since reset	0x0000000000000000
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	0x0000000000000000
0x8690	0,1	XOFF_COUNT23	63:0	R	Total number of XOFF events per channel that occurred since a reset.	0x0000000000000000
0x86A0	0,1	BUSY_TOHOST_ENABLE	0	W	Enable the busy ToHost Virtual Elink	0x0
LTITTCBUSY Controls And Monitors						
0x8800	0	LTITTC_ALIGNMENT_DONE	0:0	R	RX ALIGNMENT DONE	0x0
0x8810	0	LTITTC_CPLL_FBCLK_LOST	0:0	R	CPLL FBCLK LOST	0x0
0x8820	0	LTITTC_PLL_LOCK				
		QPLL_LOCK	1:1	R	QPLL LOCK	0x0
		CPLL_LOCK	0:0	R	CPLL LOCK	0x0
0x8830	0	LTITTC_RXCDR_LOCK	0:0	R	RX CDR LOCK	0x0
0x8840	0	LTITTC_RXRESET_DONE	0:0	R	RX Reset done	0x0
0x8850	0	LTITTC_RX_BYTEISALIGNED	0:0	R	LTITTC link not aligned	0x0
0x8860	0	LTITTC_RX_DISP_ERROR	3:0	R	Rx disp error in byte 3,2,1,0 of LTITTC link	0x0

0x8870	0	LTITTC_RX_NOTINTABLE	3:0	R	Character in byte 3,2,1,0 of LTITTC link not in 8b10b table	0x0
LTITTC_CTRLMON						
0x8880	0	LTITTC_CTRL				
		LTITTC_GTH_LOOPBACK_CONTROL	11:9	W	GTH_LOOPBACK_CONTROL for LTITTC Link	0x0
		LTITTC_SOFT_RESET	8	W	SOFT_RESET	0x0
		LTITTC_QPLL_RESET	7	W	QPLL_RESET	0x0
		LTITTC_CPLL_RESET	6	W	CPLL_RESET	0x0
		LTITTC_SOFT_TX_RESET	5	W	SOFT_TX_RESET_ALL	0x0
		LTITTC_SOFT_RX_RESET	4	W	SOFT_RX_RESET_ALL	0x0
		LTITTC_GENERAL_CTRL	3:2	W	Alignment chk reset (not self clearing)	0x0
		LTITTC_CHANNEL_DISABLE	1	W	clear toHostData	0x0
		TOHOST_RST	0	W	clear toHostData	0x0
0x8890	0	LTITTC_MON				
		BUSY_OUTPUT_STATUS	3	R	Actual status of the BUSY LEMO output signal	0x0
		LTITTC_BIT_ERR	2:0	R	Alignment comma not received correctly. Place holder	0x0
LTITTC_BUSY_ACCEPTED_G						
0x88A0	0,1	LTITTC_BUSY_ACCEPTED00	56:0	R	busy has been asserted by the given ELINK. Reset by writing to TTC_BUSY_CLEAR	0x0000000000000000
...						
0x8A10	0,1	LTITTC_BUSY_ACCEPTED23	56:0	R	busy has been asserted by the given ELINK. Reset by writing to TTC_BUSY_CLEAR	0x0000000000000000
0x8A20	0	LTITTC_SL0ID_MONITOR				
		CLEAR	any	T	Counts Set L0ID input bits	0x0
		VALUE	31:0	R	Counts Set L0ID input bits	0x00000000
0x8A30	0	LTITTC_SORB_MONITOR				
		CLEAR	any	T	Counts SetOrbit input bits	0x0
		VALUE	31:0	R	Counts SetOrbit input bits	0x00000000
0x8A40	0	LTITTC_GRST_MONITOR				
		CLEAR	any	T	Counts GRST input bits	0x0
		VALUE	31:0	R	Counts GRST input bits	0x00000000
0x8A50	0	LTITTC_SYNC_MONITOR				
		CLEAR	any	T	Counts the Sync input bits	0x0
		VALUE	31:0	R	Counts the Sync input bits	0x00000000
0x8A60	0	LTITTC_TTYPE_MONITOR				
		CLEAR	any	T	Counts the number of TType received from the LTITTC system, any write to this register clears the counter	0x0
		VALUE	46:15	R	Counts the number of TType received from the LTITTC system, any write to this register clears the counter	0x00000000
		REFVALUE	15:0	W	Counts the number of TType received from the LTITTC system, any write to this register clears the counter	0x0000

0x8A70	0	LTITTC_L0ID_ERR_MONITOR				
		CLEAR VALUE	any 31:0	T R	Counts the number of times the internal I0id /= input L0ID  Counts the number of times the internal I0id /= input L0ID	0x0  0x00000000
0x8A80	0	LTITTC_BCR_ERR_MONITOR				
		CLEAR VALUE	any 31:0	T R	Counts the number of times the BCR period does not match 3564, any write to this register clears the counter  Counts the number of times the BCR period does not match 3564, any write to this register clears the counter	0x0  0x00000000
0x8A90	0	LTITTC_CRC_ERR_MONITOR				
		CLEAR VALUE	any 31:0	T R	Counts the number of time the internally computed crc /= input CRC  Counts the number of time the internally computed crc /= input CRC	0x0  0x00000000
House Keeping Controls And Monitors						
0x9000	0	HK_CTRL_I2C				
		CONFIG_TRIG CLKFREQ_SEL	1 0	W W	i2c_config_trig  i2c_clkfreq_sel	0x0  0x0
0x9010	0	HK_CTRL_FMC				
		CLEAR	any	T	Write to this bitfield clears the latched SI5345_LOL status, SI5345_LOL_LATCHED	0x0
		SI5345_LOL_LATCHED	14	R	Latched version of SI5345_LOL, clear by writing to CLEAR bitfield	0x0
		SI5345_INTR_B	13:12	R	Connects to SI5345_INTR_B pins	0x0
		SI5345_FINC_B	11:10	W	Connects to FINC_B pins of SI5345	0x1
		SI5345_FDEC_B	9:8	W	Connects to FDEC_B pins of SI5345	0x1
		SI5345_LOL	7	R	Loss of lock pin, not connected on VC709	0x0
		SI5345_INSEL	6:5	W	Selects the input clock source 0 : FPGA (FMC LA01) 1 : FMC OSC (40.079 MHz) 2 : FPGA (FMC LA18)	0x0
		SI5345_A	4:3	W	SI5345 I2C address select 2 LSB (0x0:default, dev id 0x68)	0x0
		SI5345_OE	2	W	SI5345 active low output enable (0:enable)	0x1
		SI5345_RSTN	1	W	SI5345 active low reset (0:reset)	0x1
		SI5345_SEL	0	W	SI5345 programming mode 1 : I2C mode (default) 0 : SPI mode	0x1
0x9300	0	MMCM_MAIN				
		CLEAR LOL_LATCHED LCLK_SEL	any 4 3	T R W	Clears the LOL_LATCHED status  Main MMCM has lost lock, clear by writing to the CLEAR bitfield  1: LCLK 0: TTC	0x0  0x0  0x1



		MAIN_INPUT	2:1	R	Main MMCM Oscillator Input 2: LCLK fixed 1: TTC fixed 0: selectable	0x0
		PLL_LOCK	0	R	Main MMCM PLL Lock Status	0x0
0x9310	0	LMK_LOCKED	0	R	LMK Chip on BNL-711 locked	0x0
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	0x000
0x9330	0	FPGA_CORE_VCCINT	11:0	R	XADC voltage measurement VCCINT = (FPGA_CORE_VCCINT *3.0)/4096	0x000
0x9340	0	FPGA_CORE_VCCAUX	11:0	R	XADC voltage measurement VCCAUX = (FPGA_CORE_VCCAUX *3.0)/4096	0x000
0x9350	0	FPGA_CORE_VCCBRAM	11:0	R	XADC voltage measurement VCCBRAM = (FPGA_CORE_VCCBRAM *3.0)/4096	0x000
0x9360	0	FPGA_DNA	63:0	R	Unique identifier of the FPGA	0x0000000000000000
0x9420	0	I2C_WR				
		I2C_WREN	any	T	Any write to this register triggers an I2C read or write sequence	0x0
		DATA_BYTE3	34:27	W	Data byte 3 used when RW16BIT is set	0x00
		RW16BIT	26	W	Set to 1 to Write 3 bytes (ADDR + 16 data bits) or read 16 data bits.	0x0
		I2C_FULL	25	R	I2C FIFO full	0x0
		WRITE_2BYTES	24	W	Write two bytes	0x0
		DATA_BYTE2	23:16	W	Data byte 2	0x00
		DATA_BYTE1	15:8	W	Data byte 1	0x00
		SLAVE_ADDRESS	7:1	W	Slave address	0x0
		READ_NOT_WRITE	0	W	READ/<0>WRITE/<0>	0x0
0x9430	0	I2C_RD				
		I2C_RDEN	any	T	Any write to this register pops the last I2C data from the FIFO	0x0
		I2C_EMPTY	8	R	I2C FIFO Empty	0x0
		I2C_DOUT	7:0	R	I2C READ Data	0x00
0x9800	0	INT_TEST				
		TRIGGER	any	T	Fire a test MSIx interrupt set in IRQ	0x0
		IRQ	3:0	W	Set this field to a value equal to the MSIX interrupt to be fired. The write triggers the interrupt immediately.	0x0
0x9810	0	CONFIG_FLASH_WR				
		FAST_WRITE	57	W	Write command only. Only used for fast programming.	0x0
		FAST_READ	56	W	Status reading without command writing. Only used for fast programming.	0x0
		PAR_CTRL	55	W	Choose use FW or uC to select the Flash partition. 1 FW   0 uC.	0x0
		PAR_WR	54:53	W	Choose Flash partition. Valid when PAR_CTRL is 1.	0x0
		FLASH_SEL	52	W	1 takes control over flash, 0 gives JTAG control over flash	0x0
		DO_INIT	51	W	Untested feature, don't use it yet.	0x0

		DO_READSTATUS	50	W	Reads status from flash	0x0
		DO_CLEARSTATUS	49	W	Clears status reading from flash, back to normal flash operation	0x0
		DO_ERASEBLOCK	48	W	Erased the current block of the flash, this register has to be cleared by software	0x0
		DO_UNLOCK_BLOCK	47	W	Unlock writes to the current block, this register has to be cleared by software	0x0
		DO_READ	46	W	Reads the 16 bits from current address, this register has to be cleared by software	0x0
		DO_WRITE	45	W	Writes the 16 bits to current address, this register has to be cleared by software	0x0
		DO_READDEVICEID	44	W	DIN should return 0x0089, this register has to be cleared by software	0x0
		DO_RESET	43	W	Can be used in the future, currently disconnected in firmware	0x0
		ADDRESS	42:16	W	Address for read and write operations (25 bits, upper 2 bits are controlled by uC)	0x000000
		WRITE_DATA	15:0	W	Value of data to write towards flash	0x0000
0x9820	0	CONFIG_FLASH_RD				
		PAR_RD	19:18	R	Show which Flash partition is selected.	0x0
		FLASH_REQ_DONE	17	R	Request done	0x0
		FLASH_BUSY	16	R	Flash operation busy	0x0
		READ_DATA	15:0	R	Value of data read from flash	0x0000
0x9830	0	SI5324_STATUS				
		LOL	15:8	R	Loss of Lock SI5324	0x00
		LOS	8:0	R	Loss of Signal SI5324	0x00
0x9840	0	TACH_CNT	19:0	R	Readout of the Fan tachometer speed of the BNL712 board	0x00000
0x9850	0	RXUSRCLK_FREQ				
		VALID	38	R	Indicates that the frequency measurement is valid	0x0
		CHANNEL	37:32	W	Select the Transceiver channel to measure the clock from.	0x0
		VAL	31:0	R	Frequency in Hz of the selected channel	0x00000000
Generators						
0xA000	0	FELIG_L1ID_RESET	any	T	Any write to this register clears the FELIG L1ID	0x0
FELIG_DATA_GEN_CONFIG_ARR						
0xA020	0	FELIG_DATA_GEN_CONFIG_00				
		CHUNK_LENGTH	50:35	W	FELIG data generator chunk-length in bytes.	0x0000
		RESET	34:28	W	FELIG data generator reset. One bit per group, 0:normal operation, 1:egroup emulation held in reset.	0x0
		SW_BUSY	27:21	W	FELIG elink busy state. One bit per group, 0:normal operation, 1:elink enter busy state.	0x0
		DATA_FORMAT	20:7	W	FELIG data generator format, 2 bits per e-group. 00 8b10b, 01 direct, 10 Aurora	0x000
		PATTERN_SEL	6:0	W	FELIG data payload type. One bit per group, 0:byte counter, 1:USERDATA	0x0
...						
0xA190	0	FELIG_DATA_GEN_CONFIG_23				
		CHUNK_LENGTH	50:35	W	FELIG data generator chunk-length in bytes.	0x0000
		RESET	34:28	W	FELIG data generator reset. One bit per group, 0:normal operation, 1:egroup emulation held in reset.	0x0
		SW_BUSY	27:21	W	FELIG elink busy state. One bit per group, 0:normal operation, 1:elink enter busy state.	0x0

		DATA_FORMAT	20:7	W	FELIG data generator format, 2 bits per e-group. 00 8b10b, 01 direct, 10 Aurora	0x000
		PATTERN_SEL	6:0	W	FELIG data payload type. One bit per group, 0:byte counter, 1:USERDATA	0x0
FELIG_ELINK_CONFIG_ARR						
0xA1A0	0	FELIG_ELINK_CONFIG_00				
		ENDIAN_MOD	34:28	W	FELIG elink data input endian control. One bit per egroup. 0:little-endian (8b10b), 1:big-endian.	0x0
		INPUT_WIDTH	27:21	W	FELIG elink data input width. One bit per egroup. 0:8-bit (direct), 1:10-bit (8b10b).	0x0
		OUTPUT_WIDTH	20:0	W	FELIG elink data output width. 3 bits per egroup. 0:2b, 1:4b, 2:8b, 3:16b, 4:32b	0x00000
...						
0xA310	0	FELIG_ELINK_CONFIG_23				
		ENDIAN_MOD	34:28	W	FELIG elink data input endian control. One bit per egroup. 0:little-endian (8b10b), 1:big-endian.	0x0
		INPUT_WIDTH	27:21	W	FELIG elink data input width. One bit per egroup. 0:8-bit (direct), 1:10-bit (8b10b).	0x0
		OUTPUT_WIDTH	20:0	W	FELIG elink data output width. 3 bits per egroup. 0:2b, 1:4b, 2:8b, 3:16b, 4:32b	0x00000
FELIG_ELINK_ENABLE_ARR						
0xA320	0	FELIG_ELINK_ENABLE_00	39:0	W	FELIG elink enable. One bit per elink. 0:disabled, 1:enabled.	0x0000000000
...						
0xA490	0	FELIG_ELINK_ENABLE_23	39:0	W	FELIG elink enable. One bit per elink. 0:disabled, 1:enabled.	0x0000000000
0xA4A0	0	FELIG_GLOBAL_CONTROL				
		FAKE_L1A_RATE	63:36	W	Sets the internal fake L1 trigger rate. [25ns/LSB]	0x0000000
		PICXO_OFFSET_PPM	35:14	W	When OFFSET_EN is 1, this directly sets the output frequency, within the given adjustment range.	0x00000
		TRACK_DATA	12:12	W	FELIG GT core control. Must be set to enable normal operation.	0x0
		RXUSERRDY	11:11	W	FELIG GT core control. Must be set to enable normal operation.	0x0
		TXUSERRDY	10:10	W	FELIG GT core control. Must be set to enable normal operation.	0x0
		AUTO_RESET	9:9	W	FELIG GT core control. If set the GT core automatically resets on data error.	0x0
		PICXO_RESET	8:8	W	FELIG GT core control. Manual PICXO reset.	0x0
		GTTX_RESET	7:7	W	FELIG GT core control. Manual GT TX reset	0x0
		CPLL_RESET	6:6	W	FELIG GT core control. Manual CPLL reset.	0x0
X3_X4_OUTPUT_SELECT	5:0	W	X3/X4 SMA output source select.	0x0		
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.	0x00000
		A_CH_BIT_SEL	41:35	W	Selects the bit from the received FELIX data from which to extract the L1A.	0x0
		LB_FIFO_DELAY	34:30	W	When the GTH or GTB loopback is enabled, this controls the loopback latency in clock cycles.	0x0
		ELINK_SYNC	7:7	W	When set, synchronizes the elink word boundaries. Must be set back to 0 to resume normal operation.	0x0
		PICXO_OFFEST_EN	6:6	W	FELIG TX frequency override. 0:frequency tracking enabled, 1:TX frequency set by PICXO_OFFSET_PPM.	0x0
		PI_HOLD	5:5	W	FELIG phase-interpolator hold. 0:frequency tracking enabled, 1:freeze TX frequency.	0x0
		GBT_LB_ENABLE	4:4	W	FELIG GBT direct loopback enable. 0:disabled, 1:enabled.	0x0
		GBH_LB_ENABLE	3:3	W	FELIG GTH direct loopback enable. 0:disabled, 1:enabled.	0x0

		L1A_SOURCE	2:2	W	FELIG L1A data source select. 0:from local counter, 1:from FELIX.	0x0
		GBT_EMU_SOURCE	1:1	W	FELIG emulation data source select. 0:state-machine emulator, 1:ram-based emulator.	0x0
		FG_SOURCE	0:0	W	FELIG link check data source selection control. 0:normal operation, 1:PRBS link checker (not elink emulation data)	0x0
...						
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.	0x00000
		A_CH_BIT_SEL	41:35	W	Selects the bit from the received FELIX data from which to extract the L1A.	0x0
		LB_FIFO_DELAY	34:30	W	When the GTH or GTB loopback is enabled, this controls the loopback latency in clock cycles.	0x0
		ELINK_SYNC	7:7	W	When set, synchronizes the elink word boundaries. Must be set back to 0 to resume normal operation.	0x0
		PICXO_OFFSET_EN	6:6	W	FELIG TX frequency override. 0:frequency tracking enabled, 1:TX frequency set by PICXO_OFFSET_PPM.	0x0
		PI_HOLD	5:5	W	FELIG phase-interpolator hold. 0:frequency tracking enabled, 1:freeze TX frequency.	0x0
		GBT_LB_ENABLE	4:4	W	FELIG GBT direct loopback enable. 0:disabled, 1:enabled.	0x0
		GBH_LB_ENABLE	3:3	W	FELIG GTH direct loopback enable. 0:disabled, 1:enabled.	0x0
		L1A_SOURCE	2:2	W	FELIG L1A data source select. 0:from local counter, 1:from FELIX.	0x0
		GBT_EMU_SOURCE	1:1	W	FELIG emulation data source select. 0:state-machine emulator, 1:ram-based emulator.	0x0
		FG_SOURCE	0:0	W	FELIG link check data source selection control. 0:normal operation, 1:PRBS link checker (not elink emulation data)	0x0
FELIG_MON_TTC_0_ARR						
0xA630	0	FELIG_MON_TTC_0_00				
		L1ID	63:40	R	Live TTC data monitor.	0x000000
		XL1ID	39:32	R	Live TTC data monitor.	0x00
		BCID	31:20	R	Live TTC data monitor.	0x000
		RESERVED0	19:16	R	Live TTC data monitor.	0x0
		LEN	15:8	R	Live TTC data monitor.	0x00
		FMT	7:0	R	Live TTC data monitor.	0x00
...						
0xA7A0	0	FELIG_MON_TTC_0_23				
		L1ID	63:40	R	Live TTC data monitor.	0x000000
		XL1ID	39:32	R	Live TTC data monitor.	0x00
		BCID	31:20	R	Live TTC data monitor.	0x000
		RESERVED0	19:16	R	Live TTC data monitor.	0x0
		LEN	15:8	R	Live TTC data monitor.	0x00
		FMT	7:0	R	Live TTC data monitor.	0x00
FELIG_MON_TTC_1_ARR						
0xA7B0	0	FELIG_MON_TTC_1_00				
		RESERVED1	63:48	R	Live TTC data monitor.	0x0000
		TRIGGER_TYPE	47:32	R	Live TTC data monitor.	0x0000

		ORBIT	31:0	R	Live TTC data monitor.	0x00000000
...						
0xA920	0	FELIG_MON_TTC_1_23				
		RESERVED1	63:48	R	Live TTC data monitor.	0x0000
		TRIGGER_TYPE	47:32	R	Live TTC data monitor.	0x0000
		ORBIT	31:0	R	Live TTC data monitor.	0x00000000
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.	0x00000000
		FC_ERROR_COUNT	31:0	R	When FG_DATA_SELECT is 1, this counter reports the number of detected data errors.	None
...						
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.	0x00000000
		FC_ERROR_COUNT	31:0	R	When FG_DATA_SELECT is 1, this counter reports the number of detected data errors.	None
FELIG_MON_FREQ_ARR						
0xAAB0	0	FELIG_MON_FREQ_00				
		TX	63:32	R	FELIG regenerated TX clock frequency[Hz].	0x00000000
		RX	31:0	R	FELIG recovered RX clock frequency[Hz].	0x00000000
...						
0xAC20	0	FELIG_MON_FREQ_23				
		TX	63:32	R	FELIG regenerated TX clock frequency[Hz].	0x00000000
		RX	31:0	R	FELIG recovered RX clock frequency[Hz].	0x00000000
0xAC30	0	FELIG_MON_FREQ_GLOBAL				
		XTAL_100MHZ	63:32	W	FELIG local oscillator frequency[Hz].	0x00000000
		CLK_41_667MHZ	31:0	W	FELIG PCIE MGTREFCLK frequency[Hz].	0x00000000
FELIG_MON_L1 A_ID_ARR						
0xAC40	0	FELIG_MON_L1A_ID_00	31:0	R	FELIG's last L1 ID.	0x00000000
...						
0xADB0	0	FELIG_MON_L1A_ID_23	31:0	R	FELIG's last L1 ID.	0x00000000
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.	0x00000
		ERROR	20:0	R	Value indicates RX to TX frequency tracking error.	0x00000
...						
0xAF30	0	FELIG_MON_PICXO_23				
		VLOT	53:32	R	Value indicates TX clock (recovered RX clock) to RX reference clock frequency offset.	0x00000
		ERROR	20:0	R	Value indicates RX to TX frequency tracking error.	0x00000

0xAF40	0	FELIG_RESET				
		LB_FIFO	63:48	W	One bit per lane. When set to 1, resets all loopback FIFOs.	0x0000
		FRAMEGEN	47:24	W	One bit per lane. When set to 1, resets all FELIG link checking logic.	0x000000
		LANE	23:0	W	One bit per lane. When set to 1, resets all FELIG lane logic.	0x000000
0xAF50	0	FELIG_RX_SLIDE_RESET	23:0	W	One bit per lane. When set to 1, resets the gbt rx slide counter.	0x000000
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.	0x0
		ITKS_FIFO_DATA	16:0	W	itks emu data 16:last word 15-0:data word	0x0000
...						
0xB0D0	0	FELIG_ITK_STRIPS_DATA_GEN_CONFIG_23				
		ITKS_FIFO_CTL	19:17	W	data fifo control 2:rst 1:rd 0:wr.	0x0
		ITKS_FIFO_DATA	16:0	W	itks emu data 16:last word 15-0:data word	0x0000
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.	0x0
...						
0xB250	0	FELIG_MON_ITK_STRIPS_23	2:0	R	data fifo status 2:write done 1:full 0:empty.	0x0
FELIG_DATA_GEN_CONFIG_USERDATA_ARR						
0xB260	0	FELIG_DATA_GEN_CONFIG_00_USERDATA	15:0	W	Sets static payload word. When FELIG_DATA_GEN_CONFIG.PATTERN_SEL=1.	0x0000
...						
0xB3D0	0	FELIG_DATA_GEN_CONFIG_23_USERDATA	15:0	W	Sets static payload word. When FELIG_DATA_GEN_CONFIG.PATTERN_SEL=1.	0x0000
0xB800	0	FMEMU_EVENT_INFO				
		L1ID	63:32	R	32b field to show L1ID	0x00000000
		BCID	31:0	R	32b field to show BCID	0x00000000
0xB810	0	FMEMU_COUNTERS				
		WORD_CNT	63:48	W	Number of 32b words in one chunk	0x0020
		IDLE_CNT	47:32	W	Minimum number of idles between chunks	0x0003
		L1A_CNT	31:16	W	Number of chunks to send if not in TTC mode	0x0100
		BUSY_TH_HIGH	15:8	W	Assert BUSY-ON above this threshold	0x14
		BUSY_TH_LOW	7:0	W	De-assert BUSY-ON below this threshold	0x0F
0xB820	0	FMEMU_CONTROL				
		L1A_BITNR	63:56	W	Bitfield for L1A in TTC frame	0x30
		XONXOFF_BITNR	55:48	W	Bitfield for Xon/Xoff in TTC frame	0x20
		EMU_START	47:47	W	Start emulator functionality	0x0
		TTC_MODE	46:46	W	Control the emulator by TTC input or by RegMap (1/0)	0x0
		XONXOFF	45:45	W	Enable Xon/Xoff functionality (1/0)	0x1

		INLC_CRC32	44:44	W	0: No checksum 1: Append the data with a CRC32	0x0
		BCR	43:43	W	Reset BCID to 0	0x0
		ECR	42:42	W	Reset L1ID to 0	0x0
		CONSTANT_CHUNK_LENGTH	41:41	W	Data source select 0: Random chunk length 1: Constant chunk length	0x0
		INT_STATUS_EMU	40:32	R	Read internal status emulator	0x00
		FFU_FM_EMU_T	16	W	For Future Use (trigger registers)	0x0
		FE_BUSY_ENABLE	0	W	Enable the BUSY mechanism if L1A counter passes threshold	0x1
0xB830	0	FMEMU_RANDOM_RAM_ADDR	9:0	W	Controls the address of the ramblock for the random number generator	0x00
0xB840	0	FMEMU_RANDOM_RAM				
		WE	any	T	Any write to this register (DATA) triggers a write to the ramblock	0x0
		CHANNEL_SELECT	39:16	W	Enable write enable only for the selected channel	0x000000
		DATA	15:0	W	DATA field to be written to FMEMU_RANDOM_RAM_ADDR	0x0000
0xB850	0	FMEMU_RANDOM_CONTROL				
		SELECT_RANDOM	20	W	1 enables the random chunk length, 0 uses a constant chunk length	0x0
		SEED	19:10	W	Seed for the random number generator, should not be 0	0x200
		POLYNOMIAL	9:0	W	POLYNOMIAL for the random number generator (10b LFSR) Bit9 should always be 1	0x240
0xB860	0	FMEMU_CONFIG_WRADDR	9:0	W	write enable for the FMEmu ram block	0x00
0xB870	0	FMEMU_CONFIG				
		WE	any	T	Any write to register WRDATA triggers a write to the ramblock	0x0
		CHANNEL_SELECT	55:32	W	Enable write enable only for the selected channel	0x000000
		WRDATA	31:0	W	DATA field to be written to FMEMU_RANDOM_RAM_ADDR	0x00000000
Wishbone						
0xC000	0	WISHBONE_CONTROL				
		WRITE_NOT_READ	32	W	wishbone write command wishbone read command	0x0
		ADDRESS	31:0	W	Slave address for Wishbone bus	0x00000000
0xC010	0	WISHBONE_WRITE				
		WRITE_ENABLE	any	T	Any write to this register triggers a write to the Wupper to Wishbone fifo	0x0
		FULL	32	R	Wishbone	0x0
		DATA	31:0	W	Wishbone	0x00000000
0xC020	0	WISHBONE_READ				
		READ_ENABLE	any	T	Any write to this register triggers a read from the Wishbone to Wupper fifo	0x0
		EMPTY	32	R	Indicates that the Wishbone to Wupper fifo is empty	0x0
		DATA	31:0	R	Wishbone read data	0x00000000
0xC030	0	WISHBONE_STATUS				
		INT	4	R	interrupt	0x0

		RETRY	3	R	Interface is not ready to accept data cycle should be retried	0x0
		STALL	2	R	When pipelined mode slave can't accept additional transactions in its queue	0x0
		ACKNOWLEDGE	1	R	Indicates the termination of a normal bus cycle	0x0
		ERROR	0	R	Address not mapped by the crossbar	0x0
IP Bus						
0xC800	0	IPBUS_WRITE_ADDRESS	31:0	W	Address of the IPBus Write RAM	0x00000000
0xC810	0	IPBUS_WRITE_DATA				
		WRITE_ENABLE	any	T	Any write to this register triggers a write to the Wupper to IPBus inout RAM	0x0
		DATA	63:0	W	IPbus data to write to RAM	0x0000000000000000
0xC820	0	IPBUS_READ_ADDRESS	31:0	W	Address of the IPBus Read RAM	0x00000000
0xC830	0	IPBUS_READ_DATA	63:0	R	IPbus data from Read RAM	0x0000000000000000
0xC840	0	IPBUS_PKT_DONE	0	R	IPbus packet ready to read	0x0
ITK_STRIPS_CTRL						
0xD000	0,1	GLOBAL_STRIPS_CONFIG				
		TEST_MODULE_MASK	63:59	W	(for tests only) contains R3 mask for the simulated trigger data	0x0
		TEST_R3L1_TAG	58:52	W	(for tests only) contains R3 or L1 tag for the simulated trigger data	0x0
		TTC_GENERATE_GATING_ENABLE	51	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)	0x0
		TTC_GATING_OVERRIDE	50	W	Overrides and disables gating signal generation when set to '1' (use if the elink is deadlocked and commands don't reach it).	0x0
		INVERT_AMAC_IN	4	W	Invert the polarity of all FELIX AMAC_IN elinks	0x0
		INVERT_AMAC_OUT	3	W	Invert the polarity of all FELIX AMAC_OUT elinks	0x0
		INVERT_DIN	2	W	Invert the polarity of all FELIX 8-bit IN 8b10b elinks	0x0
		INVERT_R3L1_OUT	1	W	Invert the polarity of all FELIX R3L1 elinks	0x0
		INVERT_LCB_OUT	0	W	Invert the polarity of all FELIX LCB elinks	0x0
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	0x0
0xD020	0,1	STRIPS_R3_TRIGGER	any	T	(for tests only) simulate R3 trigger (issues 4-5 sequential triggers)	0x0
0xD030	0,1	STRIPS_L1_TRIGGER	any	T	(for tests only) simulate L1 trigger (issues 4-5 sequential triggers)	0x0
0xD040	0,1	STRIPS_R3L1_TRIGGER	any	T	(for tests only) simulate simultaneous R3 and L1 trigger (issues 4-5 sequential triggers)	0x0
MRO Registers						
0xF000	0	MROD_CTRL				
		OPTIONS	15:8	W	Extra options for MROD	0x00
		ENASPARE1	7:7	W	Enable spare1	0x0
		ENAMANSIDE	6:6	W	Enable Manual Slide in Rx Locking	0x0
		ENAPASSALL	5:5	W	Enable PassAll in EmptySuppress	0x0
		ENATXCOUNT	4:4	W	Enable SimpleCount in TxDriver for locking	0x0



		GOLTESTMODE	3:0	W	GOL Test Mode (emulate CSM): 0: Run Data Emulator when 1; 0: stop, load emulator filio 1: Enable Circulate when 1; 0: send filio data only once 2: Enable Triggered Mode when 1; 0: run continuously (no TTC) 3: Enable pattern generator	0x0
0xF010	0	MROD_TCVRCTRL				
		SLIDEMAX	23:16	W	Maximum RXSLIDES before fire a TCVR reset	0xFF
		SLIDEWAIT	15:8	W	RXclk delay in TCVR for next RX_SLIDE operation	0x20
		FRAMESIZE	7:0	W	Number of 32 data words in 1 frame	0x14
0xF020	0	MROD_EP0_CSMENABLE	23:0	W	EP0 CSM Data Enable channel 23-0	0x000000
0xF030	0	MROD_EP0_EMPTYSUPPR	23:0	W	EP0 Set Empty Suppression channel 23-0	0x000000
0xF040	0	MROD_EP0_HPTDCMODE	23:0	W	EP0 Set HPTDC Mode channel 23-0	0x000000
0xF050	0	MROD_EP0_CLRFIFOS	23:0	W	EP0 Clear FIFOs channel 23-0	0x000000
0xF060	0	MROD_EP0_EMULOADENA	23:0	W	EP0 Emulator Load Enable channel 23-0	0x000000
0xF070	0	MROD_EP0_TRXLOOPBACK	23:0	W	EP0 Transceiver Loopback Enable channel 23-0	0x000000
0xF080	0	MROD_EP0_TXCVRRESET	23:0	W	EP0 Transceiver Reset all channel 23-0	0x000000
0xF090	0	MROD_EP0_RXRESET	23:0	W	EP0 Receiver Reset channel 23-0	0x000000
0xF0A0	0	MROD_EP0_TXRESET	23:0	W	EP0 Transmitter Reset channel 23-0	0x000000
0xF0B0	0	MROD_EP1_CSMENABLE	23:0	W	EP1 CSM Data Enable channel 23-0	0x000000
0xF0C0	0	MROD_EP1_EMPTYSUPPR	23:0	W	EP1 Set Empty Suppression channel 23-0	0x000000
0xF0D0	0	MROD_EP1_HPTDCMODE	23:0	W	EP1 Set HPTDC Mode channel 23-0	0x000000
0xF0E0	0	MROD_EP1_CLRFIFOS	23:0	W	EP1 Clear FIFOs channel 23-0	0x000000
0xF0F0	0	MROD_EP1_EMULOADENA	23:0	W	EP1 Emulator Load Enable channel 23-0	0x000000
0xF100	0	MROD_EP1_TRXLOOPBACK	23:0	W	EP1 Transceiver Loopback Enable channel 23-0	0x000000
0xF110	0	MROD_EP1_TXCVRRESET	23:0	W	EP1 Transceiver Reset all channel 23-0	0x000000
0xF120	0	MROD_EP1_RXRESET	23:0	W	EP1 Receiver Reset channel 23-0	0x000000
0xF130	0	MROD_EP1_TXRESET	23:0	W	EP1 Transmitter Reset channel 23-0	0x000000
MRO Dmonitors						
0xF800	0	MROD_EP0_CSMH_EMPTY	23:0	R	EP0 CSM Handler FIFO Empty 23-0	0x000000
0xF810	0	MROD_EP0_CSMH_FULL	23:0	R	EP0 CSM Handler FIFO Full 23-0	0x000000
0xF820	0	MROD_EP0_RXALIGNBSY	23:0	R	EP0 Receiver Aligned monitor 23-0	0x000000
0xF830	0	MROD_EP0_RXRECDATA	23:0	R	EP0 Receiver Data monitor 23-0	0x000000
0xF840	0	MROD_EP0_RXRECIDL	23:0	R	EP0 Receiver Idle monitor 23-0	0x000000
0xF850	0	MROD_EP0_TXLOCKED	23:0	R	EP0 Transmitter Locked monitor 23-0	0x000000
0xF860	0	MROD_EP1_CSMH_EMPTY	23:0	R	EP1 CSM Handler FIFO Empty 23-0	0x000000
0xF870	0	MROD_EP1_CSMH_FULL	23:0	R	EP1 CSM Handler FIFO Full 23-0	0x000000
0xF880	0	MROD_EP1_RXALIGNBSY	23:0	R	EP1 Receiver Aligned monitor 23-0	0x000000
0xF890	0	MROD_EP1_RXRECDATA	23:0	R	EP1 Receiver Data monitor 23-0	0x000000

0xF8A0	0	MROD_EP1_RXRECIDLES	23:0	R	EP1 Receiver Idle monitor 23-0	0x000000
0xF8B0	0	MROD_EP1_TXLOCKED	23:0	R	EP1 Transmitter Locked monitor 23-0	0x000000

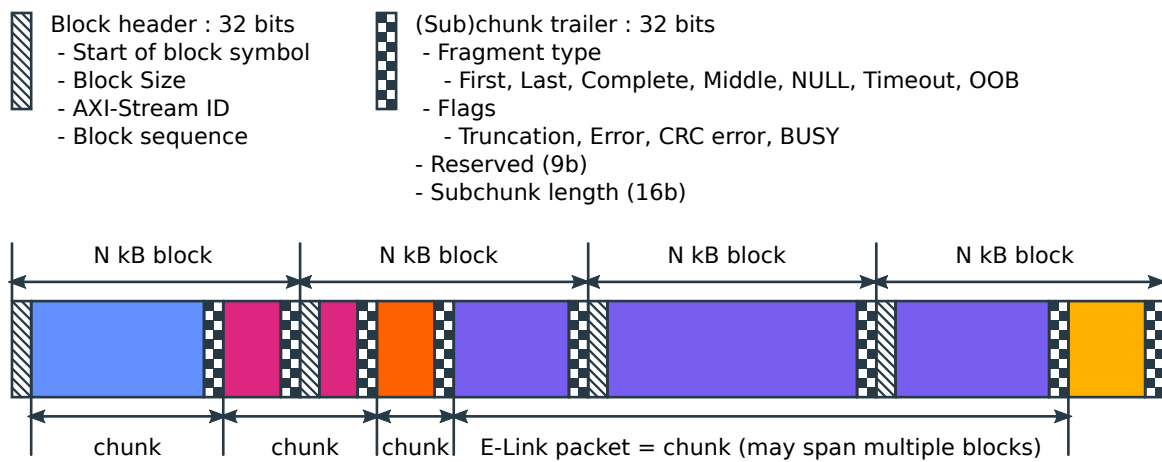
**Table B.3:** FELIX register map BAR2.

## B.2 DATA FORMATS

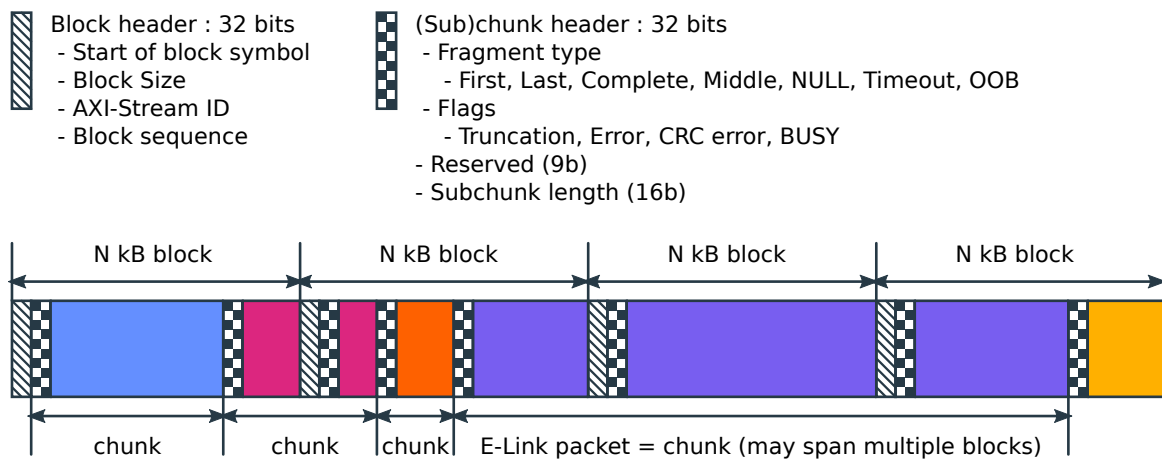
### B.2.1 CRTToHOST BLOCK FORMAT

In Phase I FELIX, the ToHost **Block** format was defined in [28]. 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 was changed to include the block size, as well as an indication that the chunk 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.

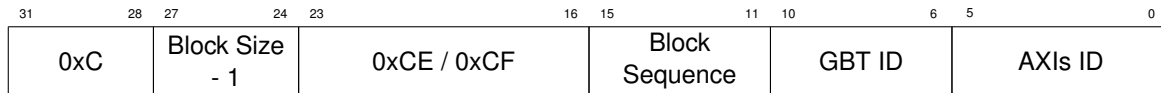
Chunk trailers will be replaced by chunk headers from firmware version 5.2. The TOHOST\_DATA\_FORMAT register was added to be able to detect firmware that generates chunk headers, in which case the register will be equal to 1. The constant at the start of the block header also changes based on this, as described in Figure B.3.



**Figure B.1:** FELIX ToHost Block format with chunk trailers.



**Figure B.2:** FELIX ToHost Block format with chunk headers.



**Figure B.3:** Block Header Format.

- **0xC** 4b, Header identifier
- **BlockSize - 1**: 4b, Block size in kB-1, 0: 1kB, 3: 4kB etc.
- **0xCE / 0xCF** 8b, Header identifier. 0xCE for chunk trailers, 0xCF for chunk headers.
- **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 this number is equal to the Egroup \* 8 + the Epath ID within the E-Group. For IpGBT this number is equal to the Egroup \* 4 + the EPath ID within the E-Group. In Pixel firmware, each decoder separates DAQ and register read data. DAQ data gets AXIs ID 0, 4, 8, etc. Register data gets AXIs ID 1, 5, 9, etc.



**Figure B.4:** Chunk trailer/header format.

- **Type** 3b:
  - 0: NULL header, 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, 512-bit or 1024-bit (depending on the PCIe generation, Gen3, Gen4 or Gen5) block at the input of the CRFromHost represents a packet. In case of Each packet consists of a 32 bit header followed by 224, 480 or 992 bits of payload. Previous versions of the data format contained a 16-bit header, Figure B.5, Figure B.6, Figure B.7 and Figure B.8 show how the bits are assigned in that packet.

The register FROMHOST\_DATA\_FORMAT shows the version of the FromHost data format which is implemented in a certain firmware build. From firmware version 5.1, only data format version 4 (PCIe Gen3 with a DATA\_WIDTH of 256b), version 5 (PCIe Gen4 with a DATA\_WIDTH of 512b) and version 6 (PCIe Gen5 with a DATA\_WIDTH of 1024b) will be used. Data formats 0 to 3 are described here for legacy reasons.

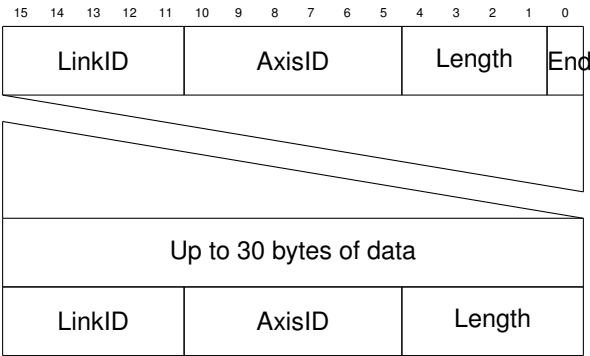


Figure B.5: Fromhost data format with 16-bit header (version 0 †End of life from version 5.0).

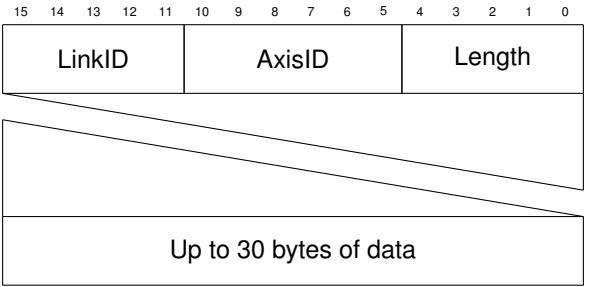


Figure B.6: Fromhost data format with 16-bit header (version 1 †End of life from version 5.0).

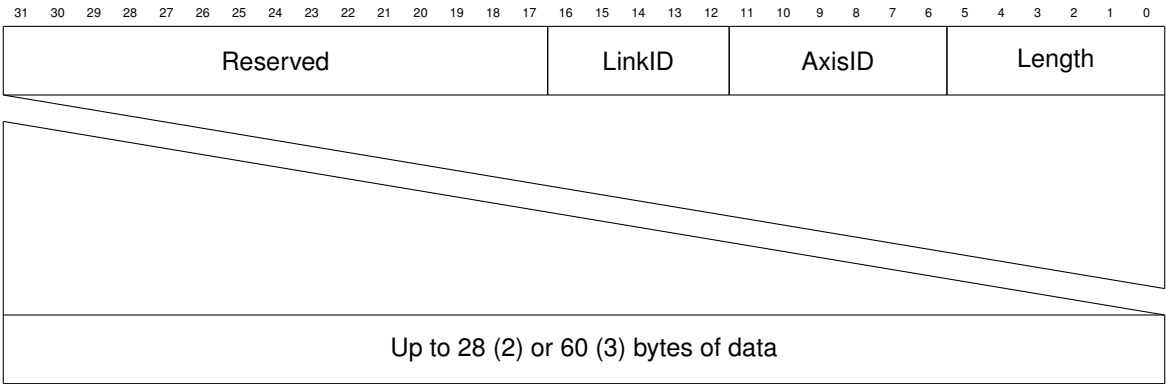
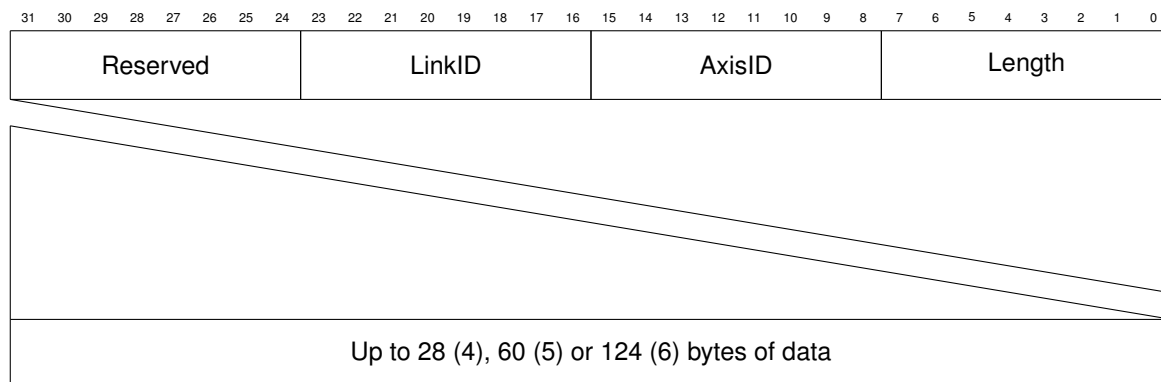


Figure B.7: Fromhost data format with 32-bit header (version 2 and 3 †End of life from version 5.1).



**Figure B.8:** Fromhost data format with 32-bit header (version 4, 5 and 6).

The fields in Figures ?? ?? 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, 64 or 128-byte block. After the header, 28, 60 or 124 payload byte positions are available, when the packet is longer, the header is repeated. If the message is shorter than the available number of 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 Length field contains the value 255 (0xFF) in version 4, 5 or 6 of the data format. (In version 2 and 3, the value for a continuation would be determined by 0x3F, version 1 this would be 0x1F, in version 0 the end of the message/chunk would be determined by the "End" field and the length of the packet was not in bytes but in 16-bit words).

### B.2.3 TTC ToHOST DATA FORMAT

Figure B.9 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.

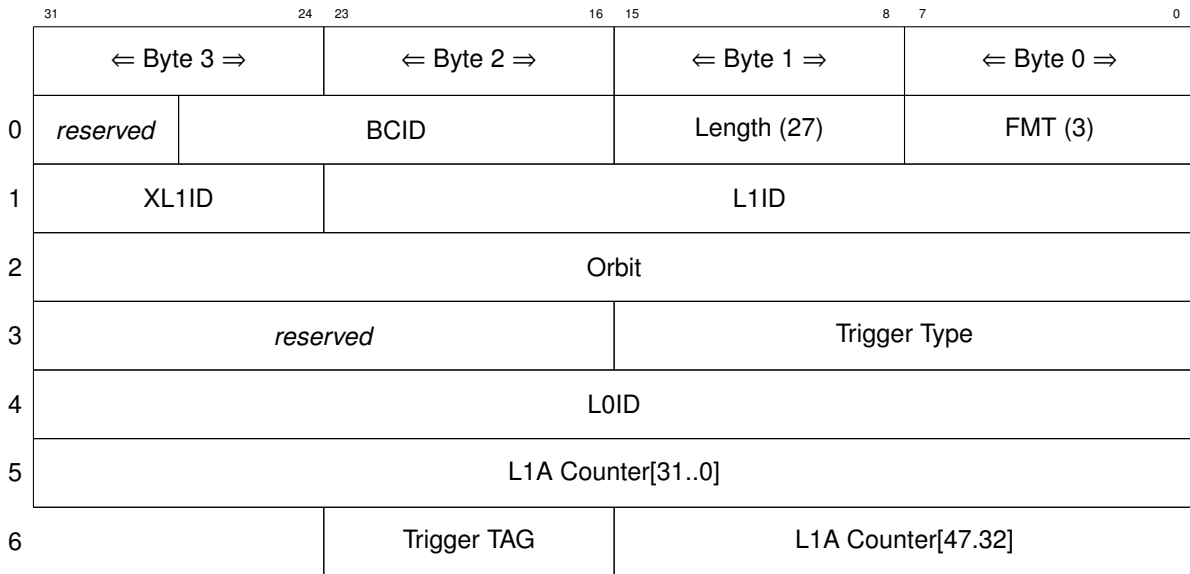


Figure B.9: TTC ToHost data format.

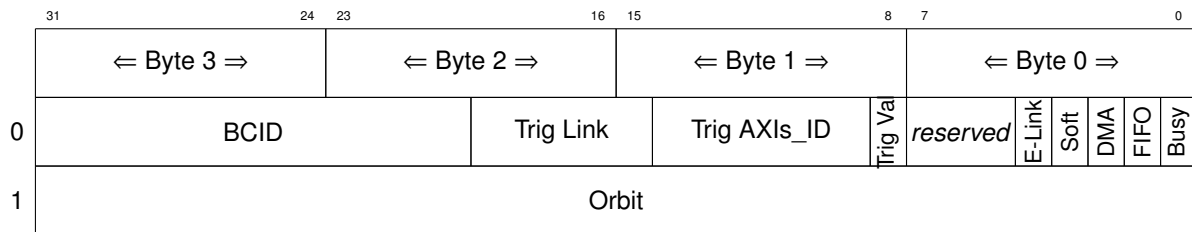
The contents of the packet can be described by a C/C++ struct type as a number of bitfields as shown below. Such a 'TTC-to-host' packet in memory can be cast directly to this type:

```
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;
    unsigned int trigger_tag : 8;
} __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 [8.4.18](#)) produces a chunk of data on any change of BUSY.



**Figure B.10:** 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 IpGBT 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, CRTtoHost, 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.



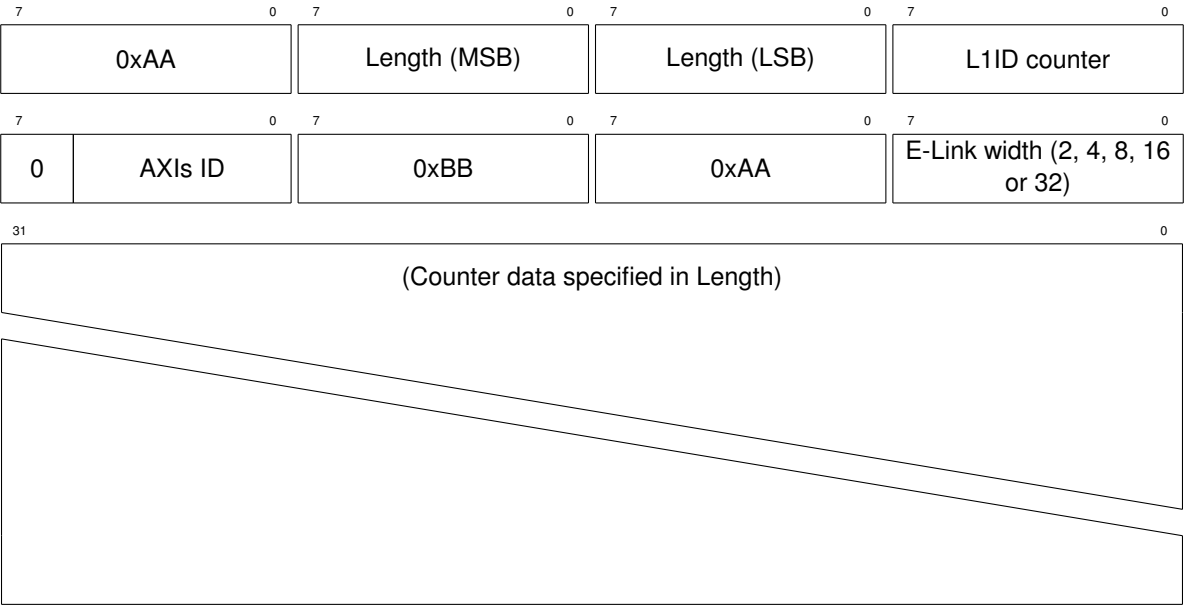


Figure B.11: Default Emulator payload.

# Appendix C

## TERMS, DEFINITIONS AND GLOSSARY

### LIST OF DEFINITIONS

### LIST OF REQUIREMENTS

5.1	.....	26
9.1	UVVM Testbenches .....	160
9.2	CI Simulation .....	161
9.3	CI Build .....	161

### LIST OF RECOMMENDATIONS

### LIST OF REMARKS

8.1	ToDo .....	52
8.2	Direct mode .....	68
8.3	Adjusting LCB frame phase .....	93
8.4	Time-critical command sequences read out from trickle memory .....	95
8.5	BC gating and stuck elinks .....	98
8.6	BC gating and the guard interval .....	98
8.7	Adjusting LCB frame delay .....	98
8.8	Direct mode .....	109
8.9	TTC for phase II .....	110

### LIST OF TABLES

2.1	Firmware Flavours and their configurations .....	3
2.2	E-Link configurations and AXIs IDs for the Firmware Flavours .....	4

3.1	K-characters used for 8B/10B coded links. BUSY-ON/OFF arrive from the front-end in both FULL mode and GBT mode cases. . . . .	11
3.2	Bit fields in the configuration registers of a single Wupper engine (i.e. two per FELIX card) for control and monitoring of the generation of XON and XOFF signals. . . . .	13
3.3	Bit fields in the configuration registers for control of assertion and de-assertion of the BUSY signal. . . . .	14
3.4	Common bit fields in the configuration registers of a single Wupper engine (i.e. two per FELIX card) for control and monitoring of the generation of BUSY conditions for both GBT and FULL mode firmware. Each firmware flavour adds extra fields on top of this common base, as shown in Tables 3.5 and 3.6. . . . .	18
3.5	Extra bit fields in the configuration registers of a single Wupper engine (i.e. two per FELIX card) for control and monitoring of the generation of BUSY conditions for FULL mode firmware. . . . .	18
3.6	Extra bit fields in the configuration registers of a single Wupper engine (i.e. two per FELIX card) for control and monitoring of the generation of BUSY conditions for GBT mode firmware. . . . .	18
3.7	“Virtual E-link” message format. . . . .	19
4.1	Protocols supported by FELIX . . . . .	24
5.1	Available resources on the different development platforms for FELIX Phase II . . . . .	25
5.2	Resource utilization for all firmware flavours estimated for the different hardware platforms. The numbers for FM1802 and VP1552, and also for KU115 for the LPGBT, PIXEL and STRIP flavours are estimations based on the build for VU37P . . . . .	26
6.2	Power Requirements . . . . .	27
6.4	Power Requirements . . . . .	27
7.1	IO pins . . . . .	31
8.1	Ports to/from CRTtoHost. . . . .	39
8.2	Ports to/from Link Wrapper. . . . .	39
8.3	Ports to/from Wupper. . . . .	39
8.4	Resource consumption in GBT mode, fully configurable . . . . .	40
8.5	Estimated resource consumption for Decoding Gearbox. . . . .	43
8.6	Estimated resource consumption for Decoding Gearbox in GBT mode. . . . .	44
8.7	Estimated resource consumption for Decoding Gearbox in lpGBT mode (8b10b). . . . .	44
8.8	Estimated resource consumption for Decoding Gearbox in lpGBT mode (Aurora). . . . .	44
8.9	Resource consumption of Endeavour Decoder module . . . . .	47
8.10	FPGA resource consumption of 64b/66b Aurora decoders for ITkPix. There is one decoder per an lpGBT link . . . . .	50
8.11	Comma characters with a special meaning in different firmware flavours . . . . .	57
8.12	32 bit axi stream interface . . . . .	63
8.13	K-characters used in FULL Mode . . . . .	64
8.14	Resource consumption for the FullToAxis entity . . . . .	66
8.15	Description of the stream controller input and output signals . . . . .	67
8.16	TTC ToHost Virtual E-Link Resource utilization . . . . .	71
8.17	Busy Virtual E-Link Resource utilization . . . . .	74
8.18	Interlaken receiver user interface . . . . .	76
8.19	Interlaken receiver AXI-Stream signals . . . . .	76
8.20	Interlaken receiver clock signals . . . . .	76
8.21	64b67b interpretation (Interlaken) . . . . .	78
8.22	Interlaken receiver configurations . . . . .	79
8.23	Interlaken RX lane latencies . . . . .	79
8.24	Interlaken receiver status signals . . . . .	79
8.25	Interlaken m_axis.tuser status signals . . . . .	80
8.26	Interlaken core error signals . . . . .	80
8.27	Interlaken resource utilization RX logic . . . . .	80

8.28	Estimated resource consumption for Encoding Gearbox, depending on different build-time configurations . . . . .	85
8.29	Endeavour protocol . . . . .	87
8.30	Resource consumption of Endeavour Encoder module . . . . .	87
8.31	Strips ToHost elink mapping. In this table, elink mapping of lpGBT optical link 0 is listed. To find elink IDs for encoders of another optical link, add $0x40 * (\text{lpGBT link ID})$ to the elink IDs listed in the table. . . . .	91
8.32	LCB link configuration registers . . . . .	94
8.33	Resource consumption of LCB encoder module for XKCU115 . . . . .	100
8.34	R3L1 link configuration registers . . . . .	103
8.35	Resource consumption of R3L1 encoder module . . . . .	104
8.36	Comma characters with a special meaning in different firmware flavours . . . . .	106
8.37	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. . . . .	110
8.38	Below is a copy of the bits found in 8.37 but extended with the external testpulse (TP), and with an adjustable delay (0-15 BC) . . . . .	111
8.39	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. . . . .	111
8.40	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 <b>following</b> the other six bits. See Figure 11 of [12]. 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. . . . .	112
8.41	Configuration registers associated with the GBT, lpGBT and FULL Mode data emulators . . . . .	121
8.42	GBT Emulator resources . . . . .	122
8.43	lpGBT ToHost Emulator resources . . . . .	122
8.44	lpGBT ToFrontEnd Emulator resources . . . . .	122
8.45	FULL Mode Emulator resources . . . . .	122
8.46	Post-synthesis TTC emulator resources for FLX712. . . . .	125
8.47	The TTC resources are post-implementation (place and rout) for FLX712. We expect similar resource utilization in Versal FPGAs. . . . .	127
8.48	CRTToHost Resource utilization . . . . .	137
8.49	CRFromHost Resource utilization . . . . .	140
8.50	Wupper Generics . . . . .	143
8.51	DMA descriptors types . . . . .	147
8.52	PCIe interrupts . . . . .	151
8.54	AXI4-Stream streams . . . . .	152
8.55	Wupper Resource utilization . . . . .	153
11.1	Institutes contributing to FELIX Firmware . . . . .	166
B.1	FELIX register map BAR0 . . . . .	B.3
B.2	FELIX register map BAR1 . . . . .	B.4
B.3	FELIX register map BAR2 . . . . .	B.39

## LIST OF FIGURES

2.1	The FELIX firmware top level block diagram using PCIe Gen4. On Gen5 capable hardware, the diagram will have 4 endpoints, each with a PCIe Gen5x4 link . . . . .	5
-----	---	---

3.1	Connections of 24 FULL mode links from the front-end and E-link assignment for GBT links to the front-end, using two fibre assemblies that each connect one row in a MTP48 feedthrough to a single miniPOD. The upper GBT link can be used for XON-XOFF signalling for all 24 links. With the optional E-links implemented the lower GBT link can also be used for this purpose, but with the E-link assignment adapted to use the lower set of links. It is also possible to use the two GBT links in conjunction with two independent sets of 12 FULL mode links. The E-links specified are 80Mb/s (2-bit) wide. The numbers in the rectangular blocks are GBT bank numbers.	12
3.2	Bandwidths of internal data paths for FULL mode firmware.	15
3.3	Bandwidths of internal data paths for GBT mode firmware configured for NSW MicroMegas detectors.	16
3.4	Bandwidths of internal data paths for GBT mode firmware configured for NSW sTGC detectors.	16
3.5	FelixCore buffer schematic in from-front-end direction. In reality DCS will also have a to-front-end path, but this has no bearing on the BUSY logic.	21
4.1	The timing mezzanine for FLX-712, with different configuration	23
6.1		28
8.1	Clocking scheme for the FELIX Phase II firmware.	33
8.2	The FELIX firmware top level detailed schematic.	34
8.3	The decoding block in the toplevel diagram	35
8.4	The decoding block, instantiating all decoder entities based on FIRMWARE_MODE [6]	35
8.5	Block diagram of a single E-Path decoder in GBT mode	36
8.6	Block diagram of an E-Group decoder in GBT mode	37
8.7	Block diagram of an E-Group decoder in lpGBT/8b10b mode	38
8.8	Block diagram of a single E-Path decoder in lpGBT / Pixel (RD53b) mode	38
8.9	Example waveform of a typical AXI stream 32b transfer. [7]	38
8.10	The Decoding GearBox entity	41
8.11	DecodingGearBox running with 8 bit input, 10 bit output. The data is constant 0x305 (k28.5+). [7]	41
8.12	The Endeavour deglitcher entity	46
8.13	The Endeavour decoder entity	46
8.14	ITkPix output data consists of data or idle blocks interrupted by periodic service blocks. The content of each block is shown before scrambling. NS stands for New Stream bit and ID is the two least significant bits of chip ID	48
8.15	Dataflow in the 64b/66b decoder	49
8.16	The RD53b Dataprocessor entity	51
8.17	RD53B Decoder latency for different number of events per stream ( $N_{event}$ ) with a binary-tree encoded hitmap	54
8.18	RD53B Decoder latency for different number of events per stream ( $N_{event}$ ) with uncompressed hitmap.	55
8.19	The 8b10b Decoder entity	56
8.20	The HDLC decoder entity	59
8.21	The HDLC decoder waveform	60
8.22	Block diagram of both the FrontEnd and FELIX ends of a Full mode link in the ToHost direction	62
8.23	The FULL mode decoder entity	63
8.24	The format of the data transmitted between the serializer and deserializer of the Full mode wrapper	64
8.25	block diagram with the user's data source and to-FELIX Full mode stream controller	66
8.26	The TTC ToHost Virtual E-Link entity	69
8.27	The Busy Virtual E-Link entity	72
8.28	The Interlaken receiver entity	75
8.29	Interlaken Burst	77
8.30	Interlaken Metaframe	77
8.31	Encoding overview	78

8.32	The encoding block in the toplevel diagram . . . . .	81
8.33	The encoding block, instantiating all encoder entities based on FIRMWARE_MODE . . . . .	81
8.34	The Encoding GearBox entity . . . . .	83
8.35	EncodingGearBox running with 8 bit output, 10 bit input. The data is alternating 0x305/0x0FA (k28.5+). [7] . . . . .	83
8.36	The Endeavour encoder entity . . . . .	86
8.37	example of waveform . . . . .	87
8.38	Dataflow in the ITkPix encoder . . . . .	88
8.39	The RD53A/B encoder entity . . . . .	89
8.40	Functional diagram of ITk Strips LCB Encoder module . . . . .	92
8.41	LCB link configuration command format . . . . .	93
8.42	No operation command format . . . . .	95
8.43	IDLE command format . . . . .	96
8.44	L0A command format . . . . .	96
8.45	Fast command format . . . . .	96
8.46	Register read command format . . . . .	97
8.47	Register write command format . . . . .	97
8.48	Block write command format . . . . .	97
8.49	Functional diagram of ITk Strips R3L1 Encoder module . . . . .	102
8.50	R3L1 link configuration command format . . . . .	103
8.51	The 8b10b Encoder entity . . . . .	105
8.52	The HDLC encoder entity . . . . .	107
8.53	The HDLC encoder waveform . . . . .	108
8.54	The TTC Encoder entity . . . . .	110
8.55	The TTC message sent from the FELIX to Frontend (32 bytes) presented as six 32-bit words . . . . .	114
8.56	The link wrapper in the toplevel diagram . . . . .	115
8.57	Block diagram for the GBT module in the link wrapper . . . . .	116
8.58	Integration test between FLX-712 and ATLAS Phase-II Strip Stave . . . . .	117
8.59	Simplified block diagram of TCLink . . . . .	117
8.60	Block diagram for the serializer and deserializer modules for Full mode . . . . .	118
8.61	The Front End data emulator in the toplevel diagram . . . . .	120
8.62	The GBT and IpGBT data emulator[6] . . . . .	120
8.63	The FULL Mode and Interlaken data emulator[6] . . . . .	121
8.64	Transmission Frame Format . . . . .	123
8.65	The TTC message sent to the Back end software (20 bytes) presented as five 32-bit words . . . . .	126
8.66	The LTI-TTC interface in the toplevel diagram . . . . .	128
8.67	The TTC message sent from the LTI to FELIX (32 bytes) presented as six 32-bit words . . . . .	129
8.68	The TTC message sent from FELIX to the LTI (12 bytes) presented as six 16-bit words . . . . .	129
8.69	The ToHost Central Router (CRToHost) in the toplevel diagram . . . . .	130
8.70	CRToHost interface symbol . . . . .	131
8.71	CRToHost Block Schematic . . . . .	132
8.72	The process of writing data into the HIFIFO . . . . .	133
8.73	The process of reading data from the HIFIFO that was written in fig. 8.72 . . . . .	134
8.74	A slightly simplified version of the writing logic of the HIFIFO . . . . .	134
8.75	The state machine diagram for the state machine used by the HIFIFO . . . . .	134
8.76	An explanation of every state of the state machine at the output of the HIFIFO . . . . .	134
8.77	The signals the state machine controls . . . . .	135
8.78	The FromHost Central Router (CRFromHost) in the toplevel diagram . . . . .	138
8.79	The FromHost or Downstream Central Router entity . . . . .	138
8.80	Example waveform of a typical FromHost Central Router transfer with its FIFO interface. [7] . . . . .	139
8.81	Example waveform of a typical AXI stream 8b transfer. [7] . . . . .	139
8.82	Wupper in the toplevel diagram . . . . .	141
8.83	Wupper interface symbol . . . . .	142
8.84	Structure of the Felix PCIe Engine . . . . .	146
8.85	Endless DMA buffer and pointers representation diagram in ToHost mode . . . . .	149

8.86	Endless DMA buffer and pointers representation diagram in FromHost mode . . . . .	150
8.87	The housekeeping interface in the toplevel diagram . . . . .	154
8.88	Housekeeping interface symbol . . . . .	155
8.89	Clock and reset block diagram . . . . .	157
9.1	Results summary of a UVVM successful simulation . . . . .	160
9.2	Continuous Integration Pipelines as seen in the Gitlab interface . . . . .	161
B.1	FELIX ToHost Block format with chunk trailers . . . . .	B.40
B.2	FELIX ToHost Block format with chunk headers . . . . .	B.40
B.3	Block Header Format . . . . .	B.41
B.4	Chunk trailer/header format . . . . .	B.41
B.5	Fromhost data format with 16-bit header (version 0 †End of life from version 5.0) . . . . .	B.42
B.6	Fromhost data format with 16-bit header (version 1 †End of life from version 5.0) . . . . .	B.42
B.7	Fromhost data format with 32-bit header (version 2 and 3 †End of life from version 5.1) . . . . .	B.42
B.8	Fromhost data format with 32-bit header (version 4, 5 and 6) . . . . .	B.43
B.9	TTC ToHost data format . . . . .	B.44
B.10	BUSY ToHost data format . . . . .	B.45
B.11	Default Emulator payload . . . . .	B.46

## 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. [36](#)

**BC** Bunch Crossing, The CERN LHC bunch crossing clock frequency is 40.07897 MHz first. [46](#), [47](#), [86](#), [87](#), [90](#)

**Block** Fixed section of memory with a specific formatting, headers and trailers first. [B.40](#)

**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. [127](#)

**DMA** Direct Memory Access first. [141](#)

**FELIX** Front End Llnk eXchange. [i](#)

**FIFO** First In First Out, a type of memory to store data, also used to cross clock domains first. [36](#), [82](#)

**FLX128** Xilinx VCU128 / VU37P Development kit with FELIX firmware. [25](#)

**FLX712** FELIX Phase I PCIe card (BNL712) with FELIX firmware. [25](#)

**FromHost** Direction of data communication, in ATLAS also referred to as Downlink. Data flows from the Host PC towards the FPGA first. [138](#)

**GBT** VersatileLink GigaBitTransceiver, a protocol and chip (GBTx) with 4.8Gb/s communication and logical links (E-Links) first. [32](#)

**IpGBT** low power GigaBitTransceiver, a successor of GBT with 9.6Gb/s Uplink, 2.56Gb/s Downlink and logical links (E-Links) first. [32](#)

**ToHost** Direction of data communication, in ATLAS also referred to as Uplink. Data flows from the FPGA towards the Host PC first. [130](#)

**Wupper** An implementation of a PCIe DMA controller for Xilinx FPGAs first. [141](#)