FELIX User Manual

ATLAS FELIX Group

<<<

Table of Contents

Chapter 1. Welcome to the FELIX User Manual

1.1. Overview

This document is intended to support all users of the Phase-I FELIX readout infrastructure with installation, maintenance and operation of their system. The document covers all aspects of the FELIX system from recommended hardware to firmware and driver installation and maintenance. Finally the full suite of FELIX software will be presented, including useful test tools leading up to the primary 'FelixCore' and 'FelixSTAR' dataflow application which is intended to form the backbone of all data taking sessions. FelixSTAR has been developed to eventually replace FelixCore. For more information users should consult the following locations for updates:

The FELIX users mailing list:

atlas-tdaq-felix-users@cern.ch

The FELIX Project Website:

https://atlas-project-felix.web.cern.ch/atlas-project-felix

The FELIX release distribution site:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/dist/

User support requests from users to the FELIX team should be made via the dedicated JIRA project:

https://its.cern.ch/jira/projects/FLXUSERS

User support via SharePoint has been discontinued. Please report any broken links of obsolete material to help improve the overall quality of our documentation.

1.2. Document Compatibility

Please refer to the link below to see the compatibility between different, firmware, software, driver and user manual versions:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/releases.html

The table in the link above will keep track of the versions of each which should be considered covered by a given version of this manual. Note - as of Register Map 4 the version number format for this document was changed to better align with other version numbers in the FELIX release. As such, the last version which is compatible with RM3 is 0.8x (i.e. in the old numbering).

Chapter 2. Introduction to FELIX

FELIX is a new detector readout component developed as part of the ATLAS upgrade effort. FELIX is designed to act as a data router, receiving packets from detector front-end electronics and sending them to programmable peers on a commodity high bandwidth network. Whereas previous detector readout implementations relied on diverse custom hardware platforms, the idea behind FELIX is to unify all readout across one well supported and flexible platform. Rather than the previous hardware implementations, detector data processing will instead be implemented in software hosted by commodity server systems subscribed to FELIX data. From a network perspective FELIX is designed to be flexible enough to support multiple technologies, including Ethernet and Infiniband.

Given the general purpose nature of the FELIX effort, the system has also been adopted by several non-ATLAS projects. This document is therefore targetted at users both within and outside of the ATLAS upgrade effort.

2.1. FELIX Variants and Functionality

FELIX supports two different link protocols for the transfer of data to and from front-end peers. Each is supported by the same hardware platform, with separate firmware revisions both based on the same core modules.

2.1.1. Gigabit Transceiver (GBT) and the Versatile Link

The Gigabit Transceiver (GBT) chipset and associated technologies were developed as part of CERN's Radiation Hard Optical Link Project[gbtmainpage]. The goal was to develop a radiation hard bi-directional link for use in LHC upgrade projects. GBT provides an interface an optical connectivity technology known as the Versatile link[versatilelink], which provides high bandwidth and radiation hard transport of data between GBT end points.

The GBT transmission protocol is designed to aggregate multiple lower bandwidth links from front-end electronics components into one radiation hard high bandwidth data link (running at up to 5 Gb/s). The logical lower bandwidth links which make up a GBT link are known as E-links. The details of how E-links are supported within the FELIX project are discussed in Section 6.1.5 of this document.

The GBT protocol has been implemented both in dedicated hardware (e.g. the GBTx chip[GBTx]) as well as directly on FPGA platforms, the latter of which has been built on for use by the FELIX project[GBTmanual].

2.1.2. FULL Mode

Within the context of the ATLAS upgrade (and subsequently externally) a requirement arose for a higher bandwidth data link from detector to FELIX than was possible with GBT, which has to support radiation hardness. These newer clients did not require radiation hardness, and were able to support a protocol which could be implemented in FPGAs on both sides of the link. The resulting development is known as 'FULL mode'[fullmodespec], referring to full bandwidth.

The FULL mode protocol is a implemented as a single wide data stream with no handshaking or logical substructure (i.e. no E-links). The reduced constraints mean that FULL mode links can operate at a line transmission rate of 9.6 Gb/s, which accounting for 8b10b encoding means a maximum user payload of 7.68 Gb/s.

Note that FULL mode in FELIX is currently only implemented in the from detector to FELIX direction, as there are currently no requirements for the to detector direction. FELIX FULL mode variants therefore implement to detector links with the GBT protocol, as this is sufficient for the required payloads.

2.1.3. Propagation of ATLAS TTC Information

As well as transferring data to and from front-ends, FELIX is also required to interface with the ATLAS Timing, Trigger and Control (TTC) system. FELIX must provide TTC information both to the front-ends at full granularity, and to network peers in a reduced form. The propagation of TTC information to the front-end is performed via dedicated E-links.

Chapter 3. Hardware Requirements and Setup

3.1. Recommended Hardware Platforms

3.1.1. FPGA I/O Hardware: VC-709 (Commodity Platform)

The hardware platform recommended for FELIX small scale tests at local test stands is based on the Xilinx® VC-709 Connectivity Kit [xilinxvc709]. This platform provides 4 optical transceivers compatible with both GBT and 'full mode' operation as well as a Xilinx® Virtex®-7 series FPGA and 8-lane PCIe Gen 3.0 interface. The TTC interface for the system is provided by the TTCfx v3 FMC mezzanine card. An image of the VC-709 board and guide to features is presented below.

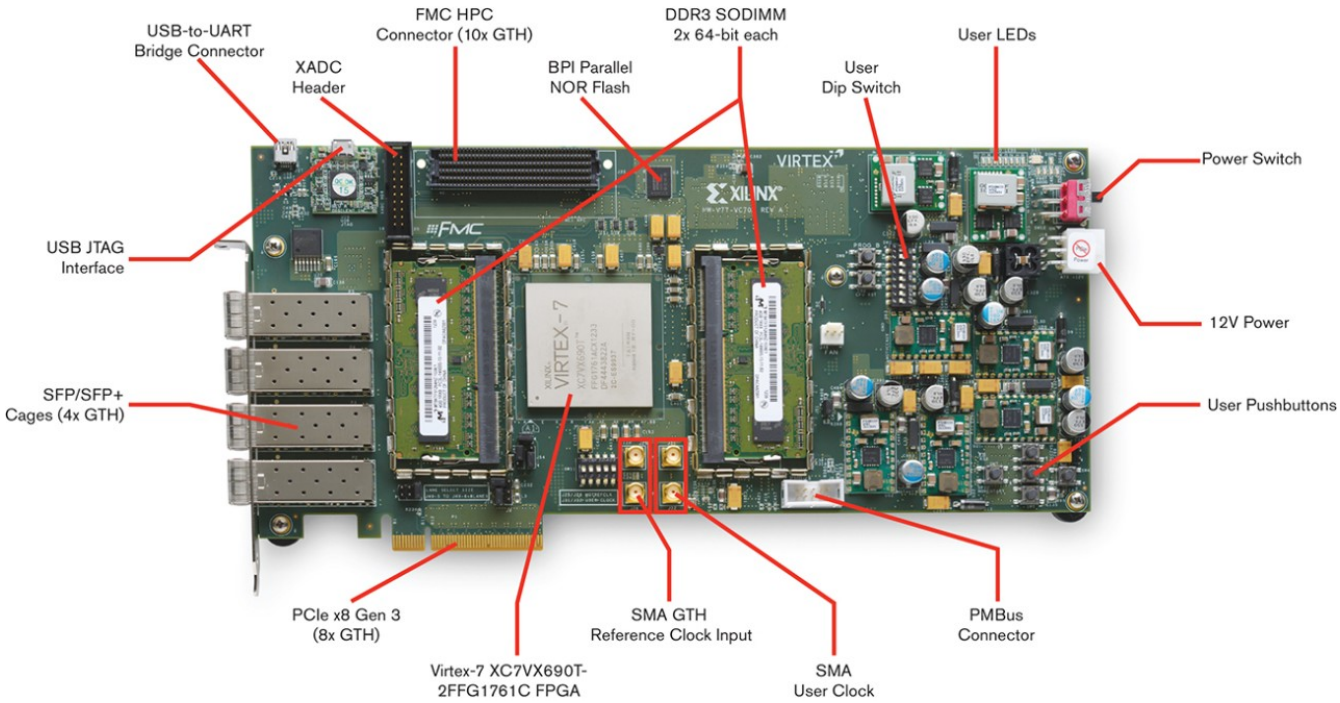

Figure 1. The VC-709 development board.

3.1.2. FPGA I/O Hardware: Custom Platforms

The hardware platform chosen for the final FELIX implementation in Phase-I is a custom interface board designed by BNL. The initial version used for FELIX is known as the BNL-711, with a modified design (BNL-712) selected for the final system. This section describes the BNL-712, which is currently the recommended card to use. A description of the BNL-711 can be found in the Appendix A.

BNL-712

The BNL-712 (a.k.a. BNL-711 v2) is similar to the BNL-711 in that it hosts a Xilinx® Kintex® UltraScale FPGA on a board capable of supporting 48 high speed optical links via MiniPOD transceivers, with a 16-lane PCIe Gen 3.0 interface. The BNL-712 is a smaller card, saving space by not hosting the unused SO-DIMM slots. Various other improvements to the FPGA pin-out to make it

easier to route signals to both PCIe endpoints. In order to make the board compatible with future developments, the TTC circuitry has been moved to a mezzanine. This makes it possible to connect both to current ATLAS clock and control systems and candidates for future implementations. An image of the BNL-712 and its key features are presented below. BNL-712 cards are available for subdetector test stands for commissioning and integration. A more detailed hardware overview of the BNL-712 can be found here:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/docs/BNL-711_V2P0_manual.pdf

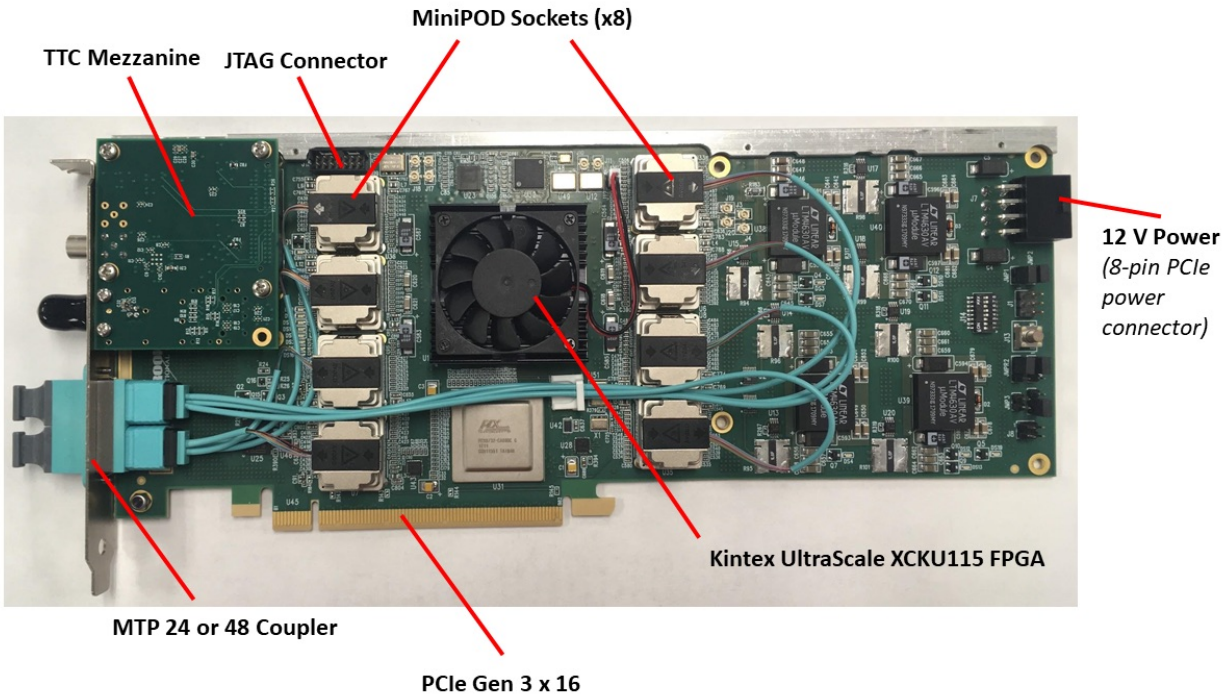

Figure 2. The BNL-712 board.

The timing mezzanine for the BNL-712 which is compatible with the ATLAS TTC System in Phase-I is show in Figure 3.

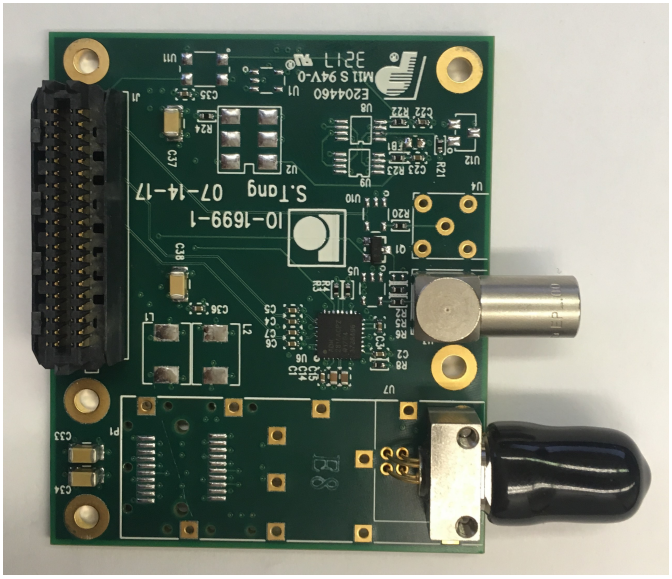

Figure 3. The ATLAS Phase-I TTC Mezzanine for the BNL-712.

For more detailed information on the BNL-712, please consult the user manual [flx-712].

3.1.3. FELIX Host Systems

The current recommended hardware platform for the BNL-712 is based on the following configuration: CPU: Intel Xeon E5-1660 V4, Motherboard: Supermicro MBD-X10SRW-F [X10SRA-F], 2U Chassis: Supermicro SC825TQC-R740WB, Memory: 32 GB ECC REG DDR4-2400 (in 4 DIMMs).

For use in a lab a 4U high server may be more convenient than a 2U high server (PCIe cards are mounted vertically, which facilitates access). It would also cost less. A Xeon 1650V4 (6 cores @ 3.6 GHz) based machine probably also is fine for lab use, the CPU costs $500 less than the 1660 V4.

The current recommended hardware platform for a VC-709 FELIX system is also based on the Supermicro® X10SRA-F motherboard [X10SRA-F]. The system should be populated with at least 32 GB of DDR4 RAM and an Intel® Xeon™ E5 family CPU (v3 or v4) with at least six real cores. Since the VC709 is biggerer than the BNL-712, it requires a 4U Chassis.

Please see the motherboard manufacturer specification for more details.

The full recommendations for the FELIX Server hardware can be found here:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/serverhw.html

3.1.4. Network Configuration

Getting low-latency, high-throughput RDMA communications to work flawlessly usually requires some specific configuration both on the FELIX hosts and the intermediate network devices.

The recommended solution uses RoCE v2 (RDMA over Converged Ethernet version 2) to implement RDMA communications. The underlying physical network is a classical Ethernet network. Specific configurations of the host networking as well as the network switches have to be applied and both have to co-operate in order to work properly. We make use of DSCP (Differentiated Services Code Point), an IPv4 QoS mechanism, and Explicit Congestion Notification (ECN), an extension to IP that allows end-to-end notification of network congestion, to minimize the impact of network issues inherent to Ethernet.

The recommended network adapter depends on the firmware mode you are using:

- GBT: dual-port 25 GbE Mellanox ConnectX-5 EN SFP28 3.0 x8 (part number: MCX512A-ACAT)
- FULL-mode: dual-port 100 GbE Mellanox ConnectX-5 EN QSFP28 3.0 x16 (part number: MCX516A-CCAT)

It is recommended to use this network adapters for FELIX data transfer only. Since they will be tuned to support RDMA communications, other forms of traffic such as control or monitoring may interfere with the data communications and lead to unpredictable performance degradation.

- Mellanox firmware version: 16.28.2006
- Mellanox and RDMA driver version: MLNX_OFED package 4.7-1.0.0.1 or 4.7-3.2.9.0
  Available from Mellanox website

For the configuration of the hosts:

- Enable ECN for all priorities:

```
for i in `seq 0 7`; do
  VAL=`cat /sys/class/net/$IFACE/ecn/roce_np/enable/$i`
  if [ "x$VAL" != "x1" ]; then
    echo 1 > /sys/class/net/$IFACE/ecn/roce_np/enable/$i
  fi

  VAL=`cat /sys/class/net/$IFACE/ecn/roce_rp/enable/$i`
  if [ "x$VAL" != "x1" ]; then
    echo 1 > /sys/class/net/$IFACE/ecn/roce_rp/enable/$i
  fi
done
```

These parameters are not persistent and need to be set after each boot. The openibd service (Mellanox drivers) provides a post-start hook that can be used for this. Create this file with permissions as follows and add the previous commands in it (for each Mellanox cards):

```
$ ls -l /etc/infiniband/post-start-hook.sh
-rwxr-xr-x 1 root root 406 Dec  8 13:47 /etc/infiniband/post-start-hook.sh
```

The recommended network switches are Juniper QFX 5120 or 5200. Please see this Juniper document for minimum software release. However, based on our tests, we recommend Junos OS 20.2R1 for these two platforms. Please refer to the device documentation for generic configuration instructions.

The recommended switch configuration consists of:

- creating a dedicated drop profile:
  We aim for "continous" congestion notifications as this seemed to work better with shared-buffer configuration.

```
drop-profiles {
    dp1 {
        interpolate {
            fill-level [ 1 90 ];
            drop-probability [ 0 5 ];
        }
    }
}
```

- partitioning the shared ingress buffer as follows:
  40% is assigned to lossless (RoCE) traffic
  55% provides extra space for lossless traffic when congestion are signaled
  5% is left to other, lossy traffic

```
shared-buffer {
    ingress {
        buffer-partition lossless {
            percent 40;
        }
        buffer-partition lossy {
            percent 5;
        }
        buffer-partition lossless-headroom {
            percent 55;
        }
    }
}
```

- defining a custom traffic class:
  lossless traffic class called "roce" assigned to queue 2

```
forwarding-classes {
    class roce queue-num 2 no-loss;
}
```

- configuring congestion signalling:
  DSCP is configured for Mellanox NICs' default code-points, 110000.
  DSCP and PFC are mutually exclusive (per port) on this platform so PFC is left inactive but visible for the sake of making it clear where to enable it if needed.

```
congestion-notification-profile {
    cnp1 {
        input {
            inactive: ieee-802.1 {
                code-point 110 {
                    pfc;
                }
            }
            dscp {
                code-point 110000 {
                    pfc;
                }
            }
        }
    }
}
```

- binding everything together:
  Create a scheduler assigning the drop profile to all traffic
  Enable ECN
  Assign this scheduler to roce traffic class

Apply all of this to all interfaces

```
schedulers {
    s1 {
        drop-profile-map loss-priority any protocol any drop-profile dp1;
        explicit-congestion-notification;
    }
}
scheduler-maps {
    sm1 {
        forwarding-class roce scheduler s1;
    }
}
interfaces {
    et-0/0/* {
        congestion-notification-profile cnp1;
        scheduler-map sm1;
        unit 0 {
            forwarding-class roce;
        }
    }
}
```

3.2. Installation of VC-709

For full details regarding the VC-709 please consult the manual provided with your equipment. In terms of installing the card into a FELIX system please follow the following guidelines. The VC-709 should be installed into an 8-lane or 16-lane Gen 3 PCIe slot on the host motherboard, taking into account the need for clearance on all sides. The board must be connected to power from the system's internal ATA power supply via a custom Molex adapter provided with the board. The power socket on the board is shown on the upper right hand corner of Figure 1, labelled '12V Power'. Ensure that the power switch, just above the socket, is switched to the on position.

The FPGA aboard the VC-709 is configured via an on-board JTAG programmer, which can be connected to a mini-USB cable with the 'USB JTAG Interface' on the top left of Figure 1. A right angled mini-USB connector is recommended to minimise obstruction of the hosts case lid, although a straight cable is provided for free with your kit. Note that this has currently only been tested for USB2, which is the recommended interface. In order to be able to program the card please connect it to a convenient USB port on your host machine, or to another machine which you wish to use as a programming server. Finally, ensure that the link transceivers are safely inserted into the on-board cages.

3.3. Installation of BNL-711 and BNL-712

The BNL-711/712 should be installed in a 16-lane Gen 3 PCIe slot on the host motherboard. The board must be connected to power from the system's internal ATA power supply via an 6-pin (recommended) or 8-pin PCIe power connector (of the type commonly used for graphics cards).

Note that the board does not support use of Xilinx power connectors.

The BNL-711/712 provides a JTAG connector to which programmers can be connected for FPGA configuration. The Digilent® HS2 programmer is recommended for this purpose. While this programmer fits comfortably into the 4U chassis, the 2U chasses will need an additional flexible adapter. For the 2U chassis, the programmer can be used with a flexible cable fabricated from the Digilent XUP flywire assembly and a pin header. Aboard the BNL-711/712 are a series of jumpers to permit users to reconfigure various I/O properties of the board. For a full specification of these please consult this section for the BNL-711 and the user manual of the BNL-712 [flx-712]. The Figure 14 and Figure 17 in the BNL-712 user manual show the mapping for on-board 24-ch or 48-ch fibers, which are for 24-ch and 48-ch cards separately.

3.4. Connecting to an existing TTC system

This section is only relevant to users who wish to connect their FELIX system to a ATLAS TTC infrastructure. Other users should skip this section and proceed directly to clock configuration.

3.4.1. VC-709 Only: TTCfx v3 Overview and Installation

For VC-709 systems the TTCfx mezzanine card [CERN_TTC_FMC] is designed to connect your FELIX card to the ATLAS TTC system as used throughout Run 1-3 operations [ttc]. BNL-711/712 systems do not require this component as the same logic is implemented on the BNL-711/712 itself. The TTCfx is a small FMC mezzanine card, as shown in Figure 4, which can be attached to the VC-709 via the single FMC slot on-board (top left of Figure 1).

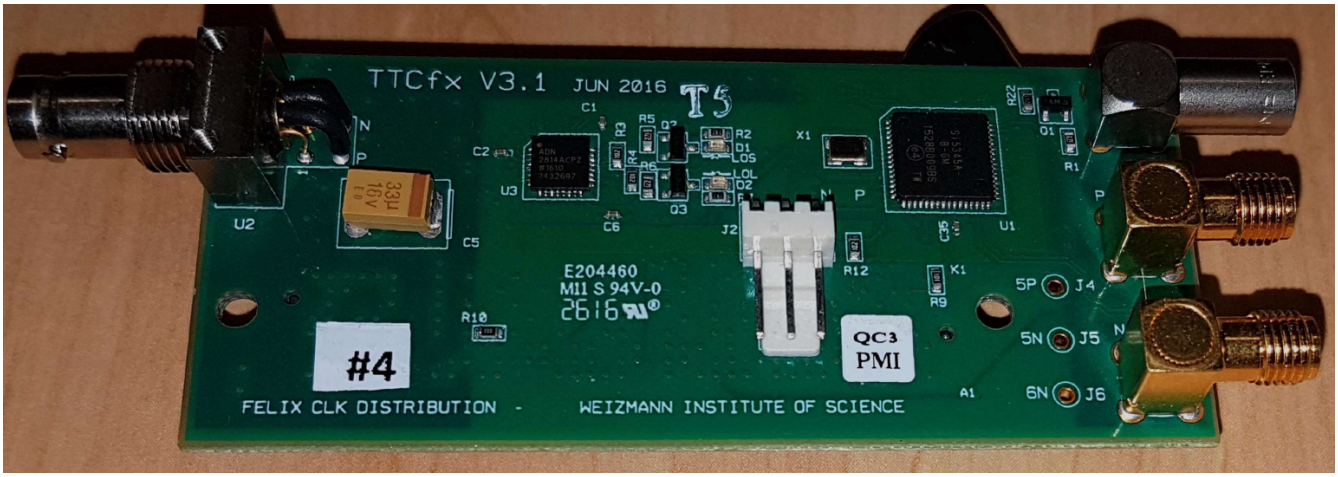

Figure 4. Image of a TTCfx v3 card.

To complete the installation, you must then connect the P and N SMA GTH Reference Clock inputs on the VC-709 (middle bottom of Figure 1) to the SMA connectors on the TTCfx v3 (P to P and N to N) via suitable SMA cables, [1]. The right angle side goes on the VC-709, to make the cable bending a bit more gentle. If you have space in your chassis, straight SMAs on both ends will do the job as well.

The TTCfx mezzanine card requires no specific firmware programming, and should work out of the box once connected to a TTC peer and a software configuration script is run. More detail is provided in the next section.

3.4.2. Connecting TTC and BUSY

This section assumes you are connecting a TTCvx-based system to FELIX. Notes on setting up such a system are available here. Once set up, connect a TTC output from the TTCvx to the TTCfx v3 using a Multi-Mode fibre with ST connectors. The connector on the TTCfx v3 end is visible in Figure 4, on the upper left hand side. On the BNL-711 the ST and LEMO connectors are located on the upper left part of the board, as shown in Figure 43. On the BNL-712 they are integrated into the TTC mezzanine, as shown in Figure 2.

Finally, use a LEMO connector to connect the TTCfx v3 or BNL-711/712 to a destination for BUSY signals (as per your use case). The connector on the TTCfx v3 is visible in Figure 4 on the upper right hand side. The BUSY signal is the ATLAS standard open-collector BUSY signal, but with a weak 24 kOhm pull-up to 5 V to allow viewing on an oscilloscope.

3.5. Configuring FELIX Clock

This section assumes you have set up the FELIX software infrastructure as in Section 5. If you have not, then please do so before proceeding.

3.5.1. Clock Source Selection

FELIX requires a clock source in order to synchronise propagation of signals both within the FPGA and to external peers. The FELIX firmware supports the use of both a received clock from an external TTC source as well as an internally generated clock for users who don't need or have access to a such a system.

On reset, the local clock is selected by default. The TTC clock source can be selected using the following command:

```
flx-config set MMCM_MAIN_LCLK_SEL=0x0
```

It is also possible to select your clock source via the *elinkconfig* graphical tool. More information on this feature will be provided in Section 6.1.1.3.

To view the overall clock status, one can run the FELIX info tool, or `flx-info`. This can be run with no command line parameters to dump summary information for your board as follows:

```
flx-info
```

Clock settings can then be viewed in the `Clock resources` section of the output, shown here:

```
Clock resources
---------------------------------
MAIN clock source     : LCLK fixed
Internal PLL Lock     : YES
ADN2814 TTC Status    : ON
```

The TTC clock is successfully configured when the 'Clock resources' section looks like this:

```
Clock resources
---------------------------------
MAIN clock source     : TTC fixed
Internal PLL Lock     : YES
ADN2814 TTC Status    : ON
```

3.5.2. TTC Clock Recovery: ADN2814

Should you wish to use a TTC clock source, you must next check that your FELIX board's ADN2814 clock recovery chip[ADN2814] is functioning correctly. Non-TTC users can skip this section.

The overall status of your ADN2814 is reported in the `Clock resources` report from `flx-info` as shown in the previous section. For more detail on the chip's status, run with the following extra parameter:

```
flx-info ADN2814
```

If your chip is functioning correctly, and you have a TTC system connected, you should see output like this:

```
$ flx-info adn2814
This is an FLX-712
TTC Status: ON
Loss of Signal Status: 0
Static Loss of Lock:   0
Loss of Lock Status:   0
```

If the output differs (e.g. if you see a loss of lock reported) please check your connections before resetting the ADN2814 using the following:

```
flx-reset ADN2814
```

3.5.3. Clock Jitter Cleaning

Whether you are using an internal or external clock, the signal must be cleaned to minimise jitter and ensure stable performance. FELIX uses one of two dedicated chips for jitter cleaning depending on your clock source and hardware.

TTC clocks should be cleaned by the *Si5345* chip[Si5345], which is hosted by the TTCfx v3 for VC-709 systems as well as on-board the BNL-711/712. Non-TTC clocks can also be cleaned by the Si5345, but for those who don't have a TTCfx v3 the VC-709 also hosts a different cleaning chip, the *Si5324*[Si5324], which offers sufficient jitter correction for the non-TTC case.

The Si5324 is currently only supported with a dedicated firmware build for those wishing to connect optical links to FELIX using the FULL mode protocol. Users of GBT must have a Si5345-based system. Should you wish to use the Si5324 please make sure to check the filename of the firmware tarball provided on the FELIX firmware distribution site to ensure the name of the cleaner is present.

Whichever your use case, your FELIX card must be configured to the correct jitter cleaner in order to function correctly. This can be achieved using the flx-init command line application without any arguments. The flx-init utility will automatically detect the used jitter cleaner and initialize it accordingly.

3.6. Connecting and Initialising Optical Links

Assuming you have set up your FELIX clocks specified above for your use case, set up the FELIX software environment as described in Section 5 and programmed the FPGA aboard your VC-709 or BNL-711/712 as described in Section 4 you are now nearly ready to attempt to connect the system to a peer via optical link using either GBT or FULL mode protocols.

To see the mapping between the Xilinx FPGA transceiver pins and the female MTP fiber connector inserted into the MTP adapter, please see appendix of this manual, or Appendix C of the FLX-712 (BNL-711 V2) Manual:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/docs/BNL-711_V2P0_manual.pdf

The first step to bringing up your links is to connect your fibres to the transceivers aboard the VC-709 or BNL-711/712, ensuring not to place excessive strain on them. Once the connectors are properly seated, you can check the physical status of your links.

3.6.1. Physical Link Layer Status: VC-709

In order to check the status of your physical connections for a VC-709 (which are SFP based) run the following:

`flx-info SFP`

Look for the line marked 'Link Status' in the output:

```
$ flx-info sfp
This is an FLX-709
               0    1    2    3
---------------------------------
Link Status  | Ok   Ok   Ok   Ok
```

3.6.2. Physical Link Layer Status: BNL-711/712

In order to check the status of your physical connections for a BNL-711/712 (which are MiniPOD based) run the following:

`flx-info POD`

There will be many lines of output, but you should check the section labelled `MiniPODs` as shown below:

```
MiniPODs
========
Only the 8 active MiniPODs will be shown
NOTE: The MiniPODs of both devices will be shown

               |  1st TX |  1st RX |  2nd TX |  2nd RX |  3rd TX |  3rd RX |  4th TX |
4th RX |

|=========|=========|=========|=========|=========|=========|=========|=========|
Temperature [C]|      49 |      52 |      46 |      46 |      54 |      50 |      49 |
49 |
3.3 VCC     [V]|    3.24 |    3.29 |    3.27 |    3.28 |    3.26 |    3.30 |    3.26 |
3.31 |
2.5 VCC     [V]|    2.42 |    2.40 |    2.44 |    2.42 |    2.42 |    2.44 |    2.44 |
2.43 |

How to the read the table below:
# = FLX link endpoint OK     (no LOS)
- = FLX link endpoint not OK (LOS)
First letter:  Current channel status
Second letter: Latched channel status
Example: #(-) means channel had lost the signal in the past but the signal is present
now.

Latched / current link status of channel:
        |   0  |   1  |   2  |   3  |   4  |   5  |   6  |   7  |   8  |   9  |   10
|   11  |

|======|======|======|======|======|======|======|======|======|======|=======|=======
|
 1st TX | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) |  -(-)
|  -(-) |
 1st RX | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | #(#) | #(#) | #(#) | #(#) |  #(#)
|  #(#) |
 2nd TX | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) |  -(-)
|  -(-) |
 2nd RX | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) |  -(-)
|  -(-) |
 3rd TX | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) |  -(-)
|  -(-) |
 3rd RX | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) |  -(-)
|  -(-) |
 4th TX | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) |  -(-)
|  -(-) |
 4th RX | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) | -(-) |  -(-)
|  -(-) |
```

If your physical link is working correctly you should see loss of latch status '#' for the relevant MiniPOD RX (receive) or TX (transmit). For a physical map of MiniPOD locations please consult this

section (same for both BNL-711 and BNL-712).

3.6.3. Logical Link Layer Initialisation

Once you have established a successful physical connection, the next step depends on your choice of logical protocol.

If you are connecting with GBT you will need to train the links to bring them up by running the following:

`flx-init`

This should run reporting no errors. You can then print the status of your GBT as well as FULL mode links with:

`flx-info GBT`

The results should look like this (this is for a BNL-712 with 24-channel firmware):

```
$ flx-info gbt
This is an FLX-712
LINK CHANNEL ALIGNMENT STATUS (entire FLX-712/711):
Channel |  0    1    2    3    4    5    6    7    8    9   10   11
        ------------------------------------------------------------
Aligned | YES  YES  YES  YES  YES  YES  YES  YES  YES  YES  YES  YES

Channel | 12   13   14   15   16   17   18   19   20   21   22   23
        ------------------------------------------------------------
Aligned | YES  YES  YES  YES  YES  YES  YES  YES  YES  YES  YES  YES
```

For GBT mode firmware: If this looks correct your GBT links should now be fully operational (configured, trained and locked).

For FULL mode firmware: The links of the front-end must be up before issuing `flx-init`, if the Frontend links require a stable link from FELIX for clock recovery, the FELIX receiver will not lock immediately. Therefore, if the link allignment shows "NO" for one or more links, issue the following command. `flx-reset GTH`, followed by `flx-info GBT` to check the link status again.

Before attempting to transfer data please ensure you have followed the guide in Section 6.1 for details on how to configure your E-links.

[1] An example SMA cable can be found here: https://eu.mouser.com/ProductDetail/Amphenol-RF/135103-01-0600?qs=sGAEpiMZZMv9ULLAfKm5f6h8NuxRC2dJ

Chapter 4. Firmware Releases and Programming

4.1. Firmware Distribution Protocol

4.1.1. Release Announcements and Distribution

FELIX firmware (and software) releases will be announced on the following e-group:

atlas-tdaq-felix-users@cern.ch

Please subscribe to this group to stay up to date with the latest updates. All new releases will include a detailed change list and reference to the associated version of this user manual.

For a full list of releases, please see:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/releases.html

The release page includes links to tarballs of firmware releases (containing both .bit and .mcs files) are made available. Versioning information is available in the on-site 'bitfiles_change_log.md', please download the latest version as indicated.

all recent firmware revisions are labelled 'CLKSELECT' to indicate that they support both dual TTC and local clock sources. Older revisions were dedicated to one clock or another, but these should now be considered deprecated. Please upgrade to a newer revision if you have such a version.

4.1.2. Supported Link Protocols & Encoding

FELIX firmware builds currently support the following link protocols and encoding options as part of the standard release.

Mode	Support
GBT Normal Mode 8b/10b	Y
GBT Normal Mode HDLC	Y
GBT Normal Mode Direct	N
GBT Wide Mode	N
FULL Mode	Y

For the VC 709 (miniFELIX) bitfiles are released depending on two different Jitter Cleaners. If the TTCfx3 mezzanine card is installed, the Si5345 on the mezzanine can be used, and the TTC clock can optionally be selected as an input clock through elinkconfig. For FELIX users who don’t have the TTCfx3 mezzanine, we provide a special firmware build which supports the Si5324 jitter cleaner on the VC709 board.

The version with Si5324 support can be recognized with the SI5324 keyword in the archive filename:

FLX709_FULLMODE_4CH_CLKSELECT_GIT_master_rm-4.9_278_200619_14_13.tar.gz: TTCfx3 / Si5345 support

FLX709_FULLMODE_4CH_CLKSELECT_SI5324_GIT_master_rm-4.9_278_200619_05_38.tar.gz: VC709 standalone / Si5324 support

4.2. Firmware Programming

The FPGAs aboard both the VC-709 and BNL can be programmed directly via a JTAG interface using the Vivado™ software suite[vivado2016]. For the VC-709 this method also makes possible to program the on-board FLASH ROM. A configuration programmed into the FPGA directly will be lost if the machine is switched off, whereas a configuration programmed in the FLASH will persist. This will make it possible to retain the desired programming state of the card e.g. if transported. This section will describe how to program the card using all available methods.

4.2.1. JTAG Connectivity

The VC-709 comes with an on-board JTAG programmer, accessible via USB, as described in Section 3. The BNL-711/712 does not have an on-board programmer, and as such you will need to acquire a USB-accessible programmer. The FELIX developers recommend the Digilent® HS2 for this purpose.

4.2.2. Setting up the Vivado™ Suite

Specific installation instructions for the Vivado™ suite are provided with your development kit. Note that the instructions in this section are compatible with the 2014, 2015 and 2016 releases of the suite. We recommend you install the software locally on the PC you wish to use as your programming server. This should be connected to your VC-709 in your FELIX host via USB as described in Section 3.2. When you first connect your system via USB you will need to run a Xilinx® setup script to configure the bus properly (path may vary depending on product year):

```
source Xilinx/Vivado/2018.1/data/xicom/cable_drivers/lin64/digilent/install_digilent.sh
```

The Vivado™ environment can be started with the following commands:

```
source Xilinx/Vivado/2018.1/settings64.sh
vivado &
```

You will then be presented with the Vivado™ splash screen, where you should select 'Open Hardware Manager' as shown in the red box in Figure 5.

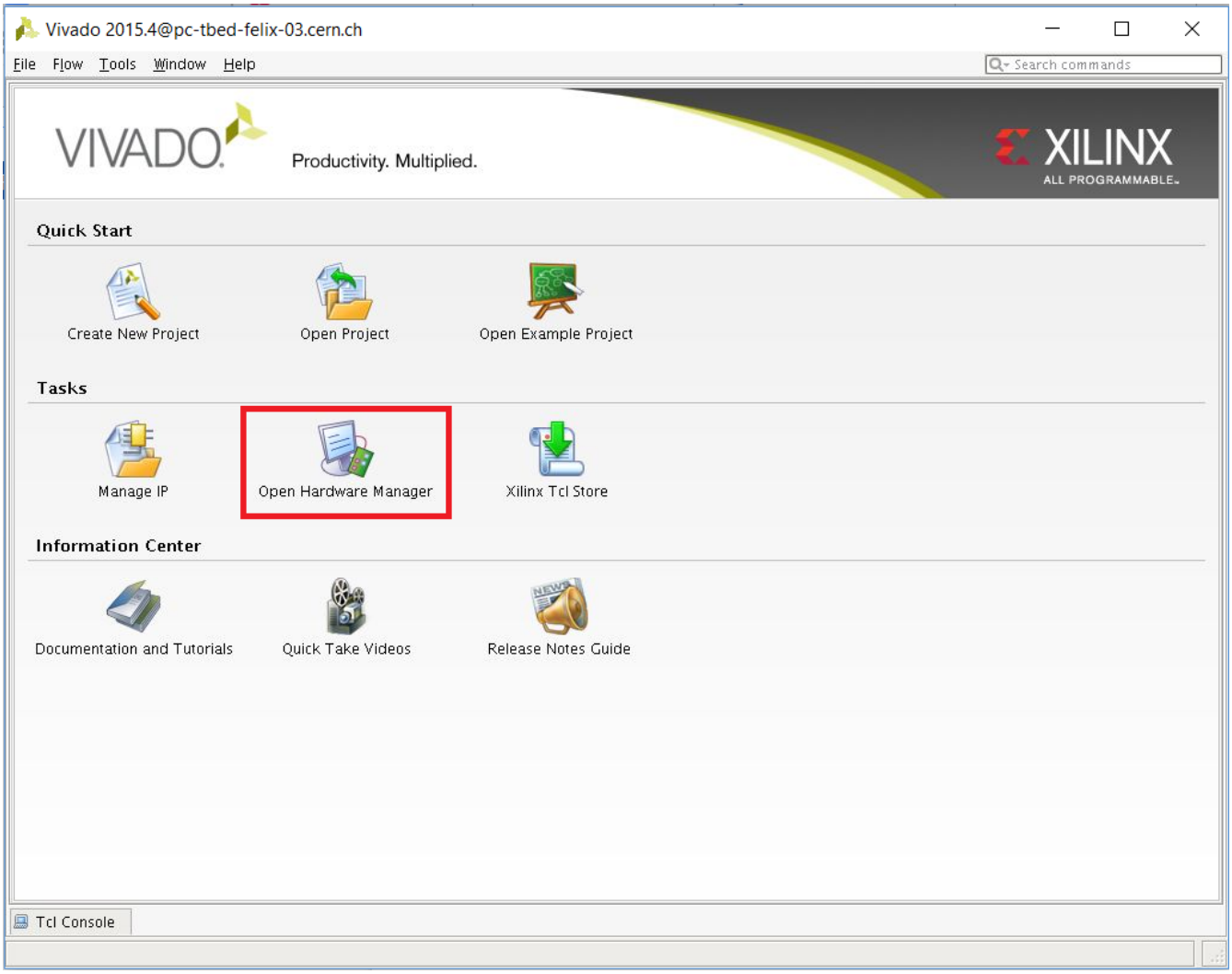

Figure 5. Vivado™ Splash Screen.

From the hardware manager select 'Open Target' on the top left as shown in Figure 6 and choose 'Open New Target'.

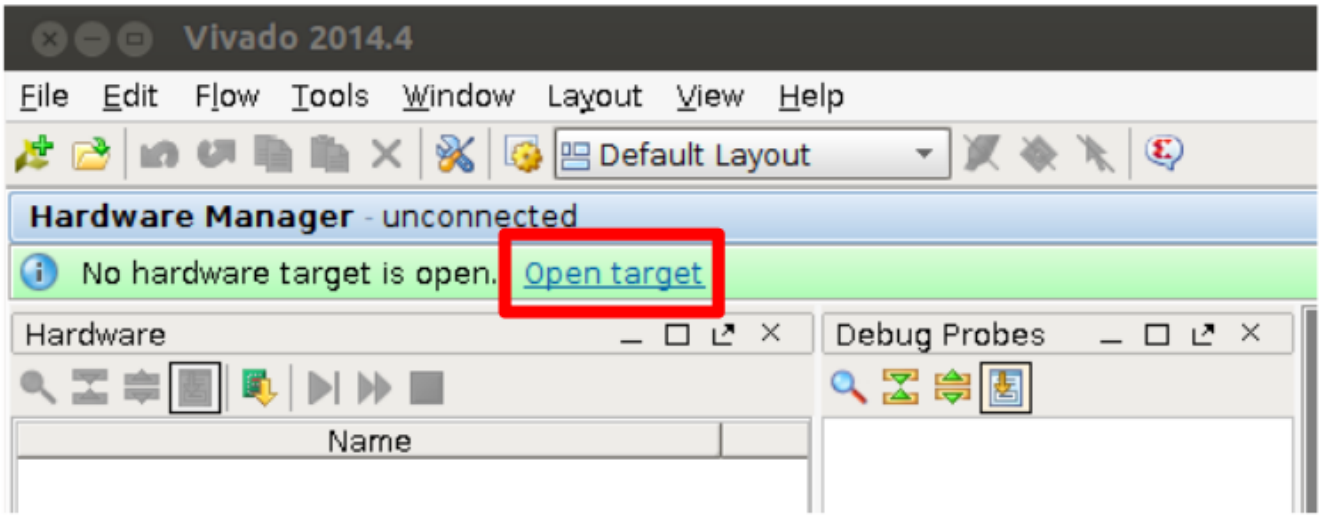

Figure 6. Vivado™ Hardware Manager.

From this point, select 'Next' on the following screen and 'Connect to Local Server' after that, once again press 'Next'. This should bring you to the hardware list. On this screen select the FPGA on your VC-709 or BNL-711/712 from the uppermost list (if you have only one board there should be

only one entry, if not, find yours in the list by name). The screen you will see is shown in Figure 7. Once you have found your FPGA and selected it press 'Next' on the bottom right and 'Finish' on the following screen.

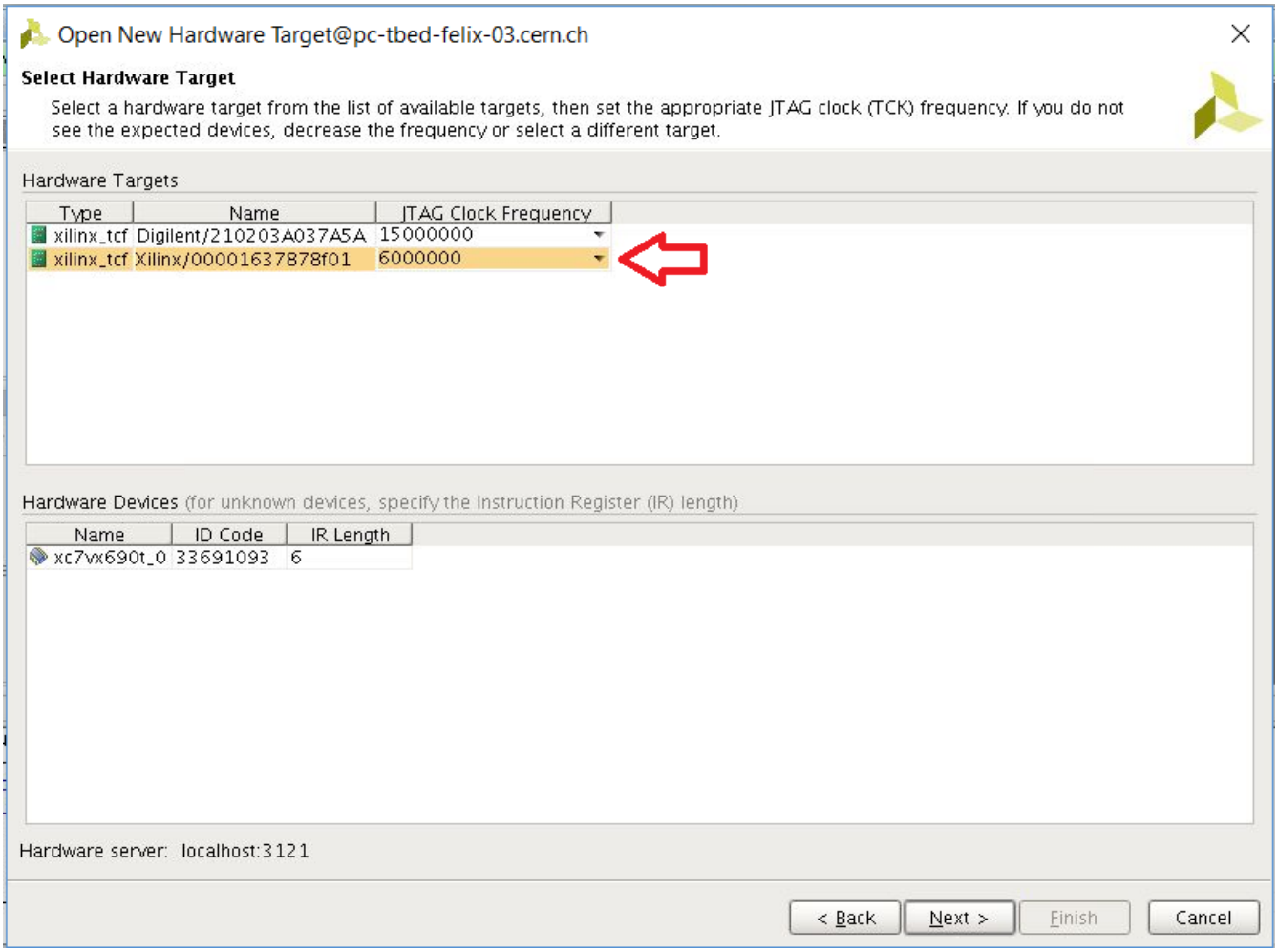

Figure 7. Vivado™ Target Selector with VC-709's Virtex7 FPGA selected, as indicated by red arrow (FPGA ID will vary from model to model). The BNL-711/712's Kintex Ultrascale FPGA will appear as xkcu115_0

From here, you will be taken to the main programming interface, as shown in Figure 8. You are now ready to program your FPGA or FLASH.

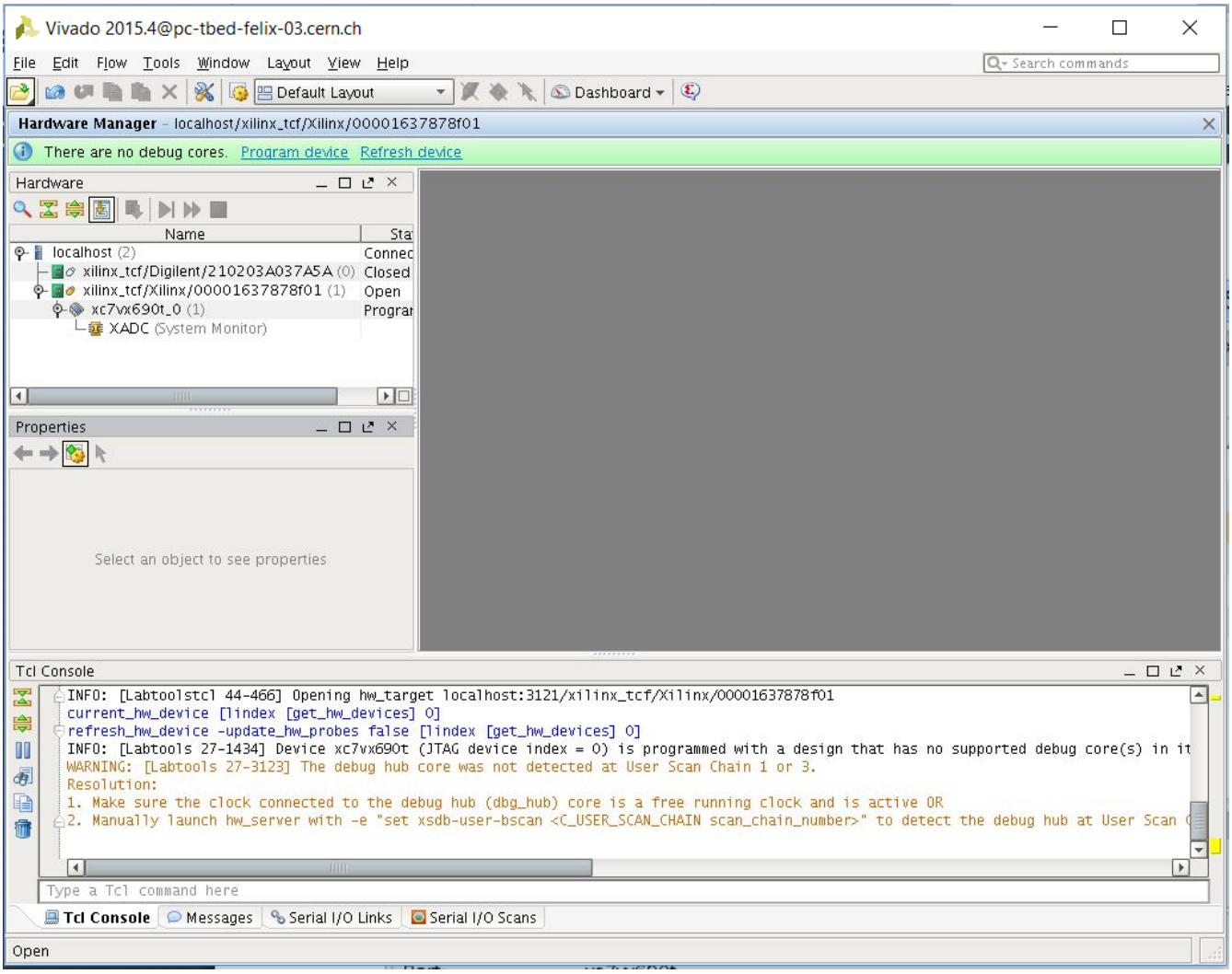

Figure 8. Vivado™ Programming Interface.

4.2.3. Programming the FPGA Directly

To program an FPGA directly, select it from the device list on the main programming window (as shown in Figure 9, right click and select 'Program Device'.

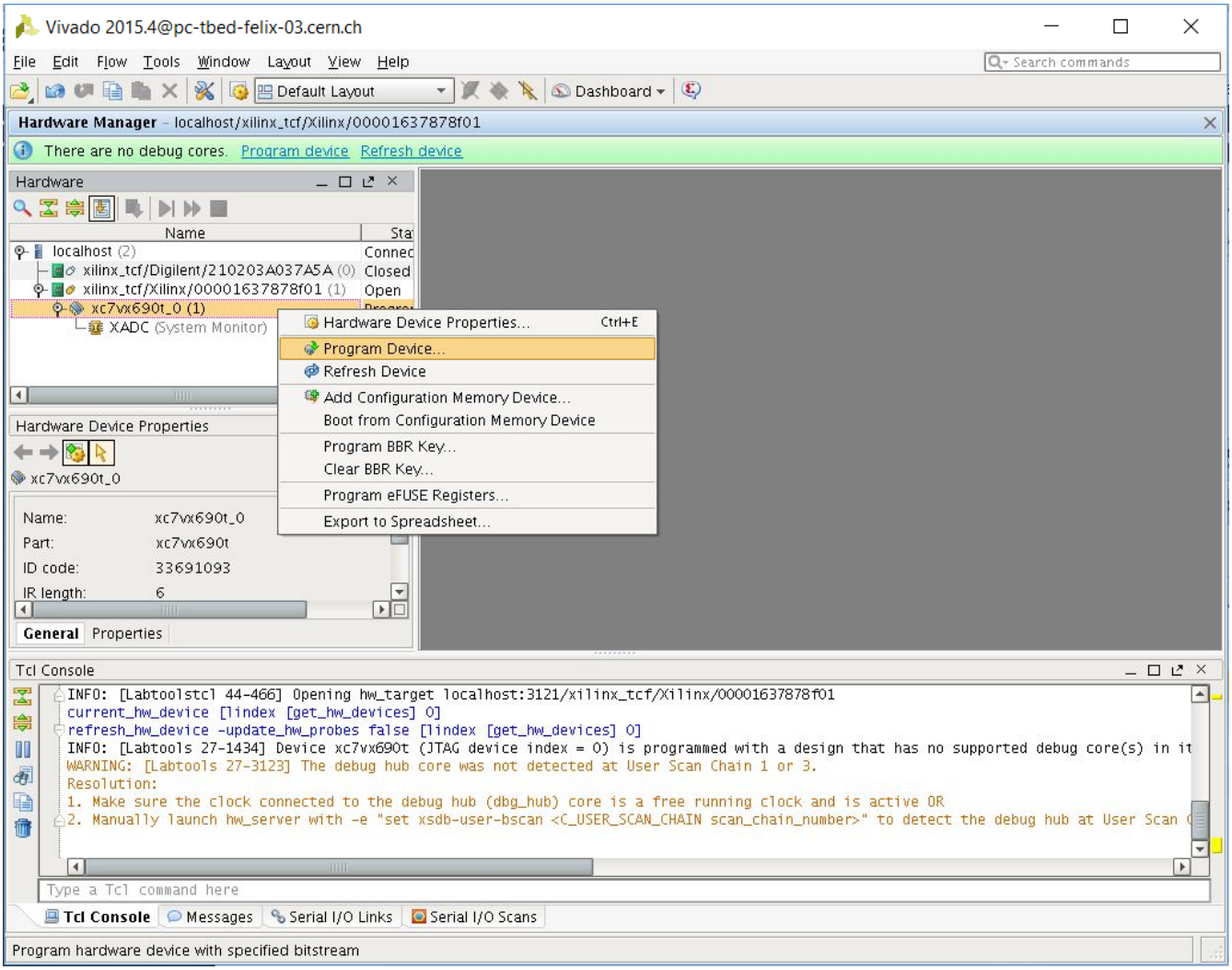

Figure 9. Selecting Device to Program.

You will now be asked to select a .bit file as shown in Figure 10. This is available in the firmware release tarball as specified at the start of this chapter. You do not need to select a debug probes file. Once a file has been chosen, select 'Program' on the bottom right to write the file to the FPGA. Once complete your FPGA should now be fully reprogrammed.

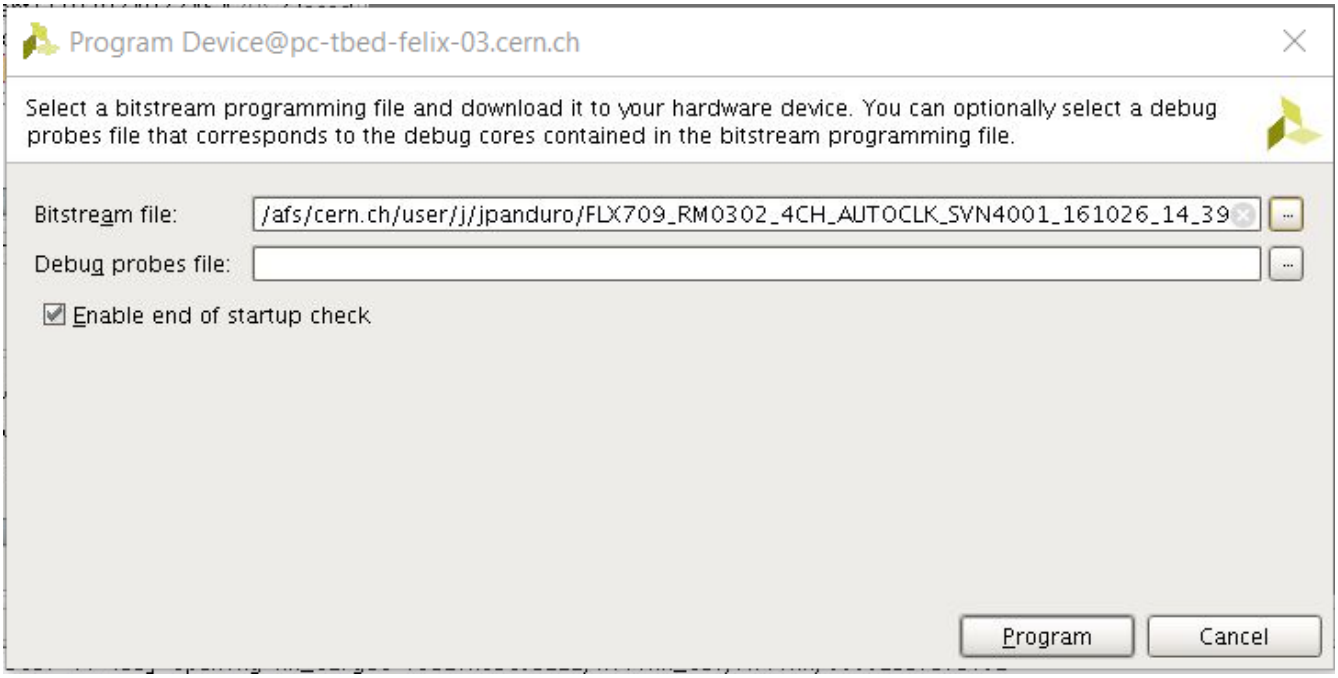

Figure 10. Selecting Bit file to Program.

4.2.4. Programming the FLASH ROM (VC-709)

To program the FLASH ROM start once again from the main programming window. Find and right click on your FPGA and select 'Add Configuration Memory Device' in the list, as shown in Figure 11

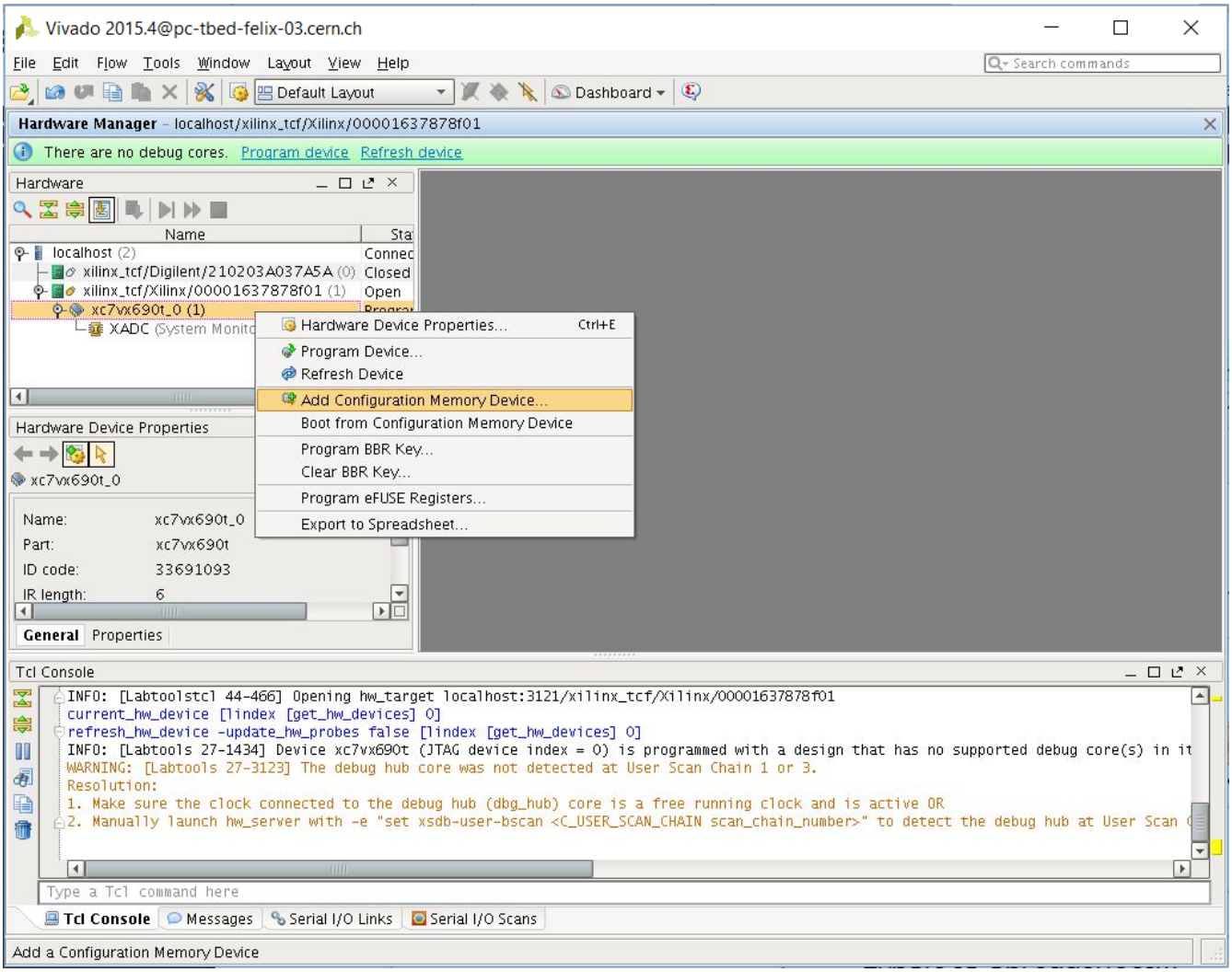

Figure 11. Select Vivado™ Flash Programming Dialog.

From here you will be taken to the a dialog requesting that you select the memory device you wish to program. On the VC-709 this will typically be a Micron memory device with given parameters. To find it quickly enter the criteria demonstrated in Figure 12 and select the device as shown. Look for the device with alias '28f00ag18f'.

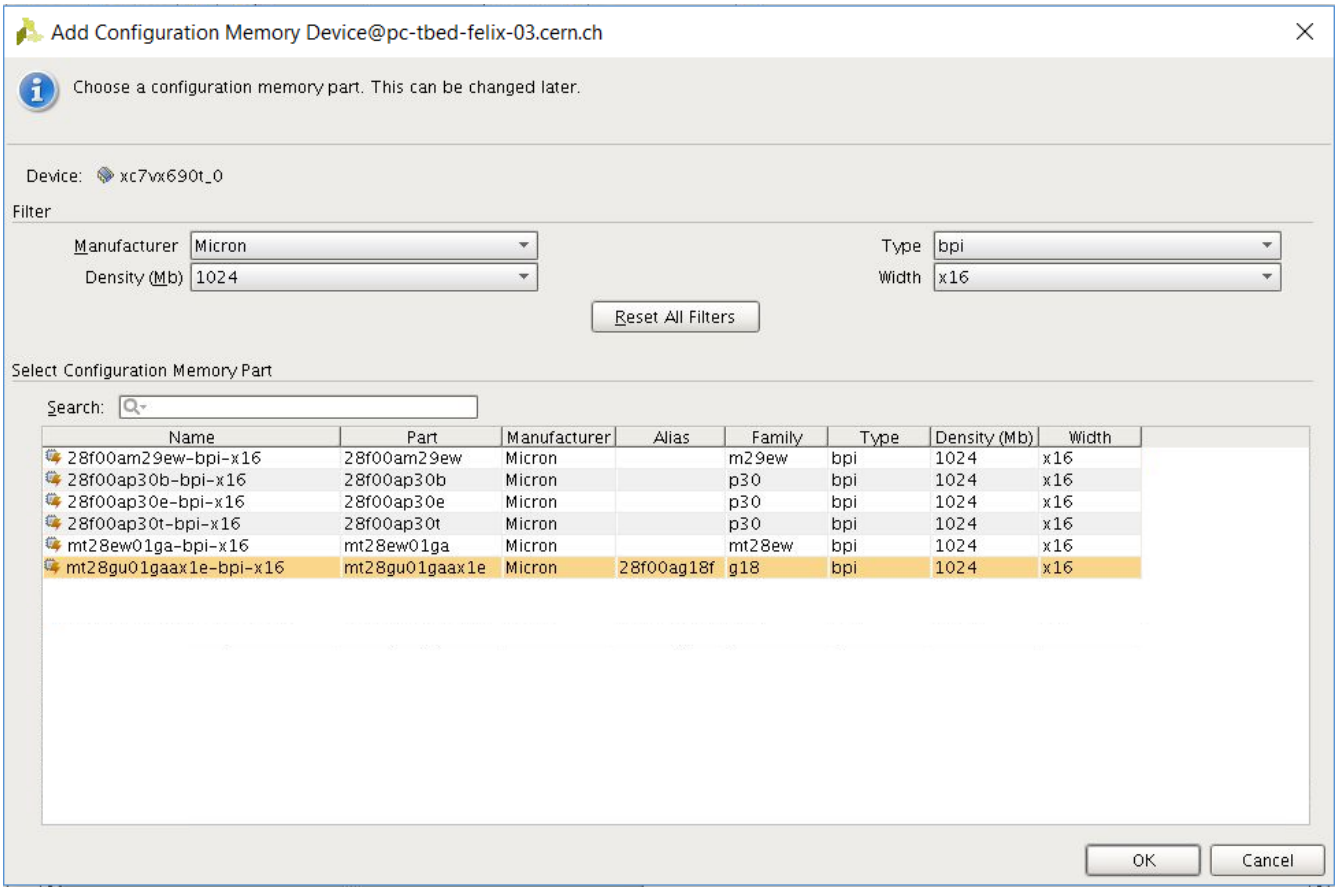

Figure 12. Memory Device Selection Interface.

Once selected, press 'Ok' on the bottom right and 'Ok' again on the following window asking 'Do you want to program the configuration memory device now?'. On the subsequent dialog, choose the .mcs file you wish to program (provided with your firmware release) as shown in Figure 13. Select 'Ok' at the bottom to program the FLASH. Once complete your card should be programmed with a non-volatile firmware installation that will survive loss of power to the host.

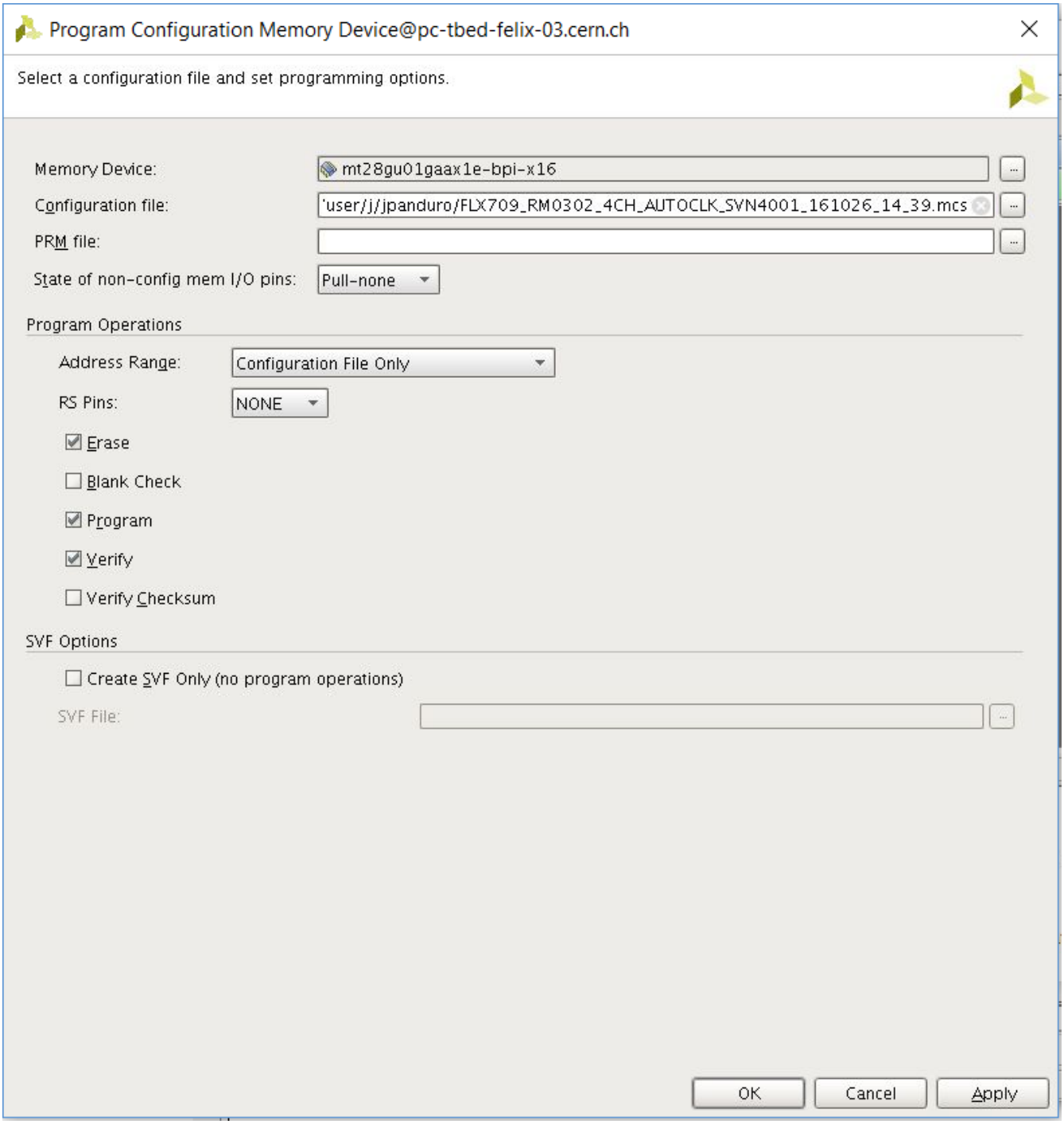

Figure 13. Selecting .mcs file to program.

4.2.5. Programming the FLASH ROM (BNL-711/712)

FLASH programming for the BNL-711/712 is done by means of the `fflashprog` application, which is provided as part of the FELIX software suite. Please consult the instructions provided in Section 6.4.9. Note that the BNL-711/712 has 4 different FLASH partitions which can be programmed with 4 different firmware images. The board will by default come up from powercycle loaded from the partition specified by the jumper configuration described in [app:bnl711]. Please ensure you program the correct sector in order to see the expected image loaded. If required, the firmware image from another partition can be loaded into the card after power-on using the `fflash` application (see Section 6.4.8).

4.2.6. Enabling new FPGA Configuration

If you have programmed our FPGA directly, please soft reboot your machine to pick up the new configuration. For changes to the FLASH ROM a full powercycle may be needed to pick up the new firmware, unless you have manually programmed the FPGA from FLASH as described in Section 6.4.8. In the latter case a soft reboot will be sufficient.

PCIe hotplug procedure

Should you wish to avoid rebooting your machine (assuming a powercycle isn't required) it is possible to rescan the PCIe bus re-synchronise with the new firmware image. Note that this procedure is not stable under all circumstances, and may produce inconsistent results. This procedure should only be attempted if a reboot is prohibited. First load the firmware using Vivado software over the USB programming cable. Next detach the PCI devices using the command:

```
pcie_hotplug_remove.sh
```

After this step, the PCI devices can be rescanned and reattached using the command:

```
pcie_hotplug_rescan.sh
```

The last step also restarts the TDAQ drivers. The software for detach / rescan can be found in the felix software library, "pcie_hotplug" package.

This procedure works for the VC 709 card on both PEX 8732 and PEX 8747 bridges.

For the BNL 712, there are come caveats: the procedure will fail if the same firmware is uploaded as was already on the BNL 712 card; the procedure will also fail if GBT firmware is loaded on a card which already had Full Mode firmware loaded. Finally, if the PCI detach fails, this condition is reflected in the PCI tree and can be seen with a simple command like lspci. Re-execution of PCI detach after a second is usually successful.

It is well known that the PCI subsystem of CC7 can be problematic, depending on the kernel version. This procedure was tested to work with Centos7.7 kernel 3.10.0-1062.4.3.el7.x86_64, as well as the more recent kernel v5.00.x, but is not guaranteed to work with all intermediate kernels.

Chapter 5. Software Distribution and Installation

5.1. Software Distribution Protocol

5.1.1. Pre-requisites

FELIX software is formally supported for systems using the CentOS 7 operating system.

5.1.2. Release Announcements and Distribution

FELIX software (and firmware) releases will be announced on the following e-group:

atlas-tdaq-felix-users@cern.ch

Please subscribe to this group to stay up to date with the latest updates. All new releases will include a detailed change list and reference to the associated version of this user manual.

For a full list of releases, as well as download information, please see:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/releases.html

Here users will be able to find the latest firmware and software. The newest recommended version is marked in all cases. Installation instructions for the software suite and driver can be found below.

FELIX Driver

The latest version of the FELIX driver is available on the distribution site within 'software/drivers'. For driver release notes and installation details, please refer to

Driver release notes

FELIX Software Suite

The latest version of the FELIX software suite is available on the distribution site in within 'software/apps'. The software release is provided as in tarball and rpm forms to suit different use cases. A docker image with the release pre-installed is also available.

5.2. Software Installation Instructions

5.2.1. Driver RPM Installation Instructions

DKMS

The FELIX driver makes use of 'Dynamic Kernel Module Support' (DKMS) to automatically track kernel changes once installed. Users should therefore only need to change their installation if a new

version of the driver itself is released.

Removal of Existing Driver Installations

In order to update the FELIX driver it will first be necessary to remove any existing driver installations from your system. To do this please follow the procedure outlined below. You will require superuser privileges in order to perform the driver de-installation itself and subsequent cleanup.

To check if a driver is already installed issue the following command:

```
rpm -qa | grep tdaq
```

If a driver rpm is installed you'll see a response along the lines of:

```
tdaq_sw_for_Flx-4.3.0-2dkms.noarch
```

To remove the driver do the following (substituting 'filename' for the results of the search in the previous step):

```
rpm -e filename
```

Once this operation is complete you will be in a position to install the latest FELIX driver.

Installation of New Driver

To install the FELIX driver RPM, run the following command (superuser privileges required):

```
yum install tdaq_sw_for_Flx-4.3.0-2dkms.noarch.rpm
```

(this should take 1-2 minutes to complete, due to the need to compile the driver for your kernel as per the DKMS framework)

Once the driver is installed you should start it as follows (as superuser):

```
./etc/init.d/drivers_flx start
```

Once started you can check the status of the card using:

```
cat /proc/flx
```

You should see output similar to what is shown below (will vary depending on your firmware version(s) and the number of cards in your system; here 2 cards are installed):

```
$ cat /proc/flx
FLX driver 4.3.0 for RM4 F/W and TDAQ release tdaq710_for_felix_4.3.0. Distributed
with driver RPM 4.3.0
Debug                          = 0
Number of devices detected     = 4

Locked resources
      device | global_locks
=============|=============
```

```
           0 |   0x00000000
           1 |   0x00000000
           2 |   0x00000000
           3 |   0x00000000

Locked resources
device | resource bit |     PID |  tag
=======|==============|=========|=====

Device 0:
Card type                   : FLX-712
Device type                 : 0x0427
FPGA_DNA                    : 0x0105968534800345
Reg Map Version             : 4.8
GIT tag                     : rm-4.8
BUILD Date and time         : 22-10-2019 at 11h02
GIT commit number           : 46
GIT hash                    : 0xdc11cebb
Firmware mode               : GBT
Number of descriptors       : 2
Number of interrupts        : 8
Interrupt count |      0 |      0 |      0 |      0 |      0 |      0 |      0 |
0 |
Interrupt flag  |      0 |      0 |      0 |      0 |      0 |      0 |      0 |
0 |
Interrupt mask  |      1 |      1 |      1 |      1 |      1 |      1 |      1 |
1 |
MSI-X PBA       00000000

Device 1:
Card type                   : FLX-712
Device type                 : 0x0428
FPGA_DNA                    : 0x0105968534800345
Reg Map Version             : 4.8
GIT tag                     : rm-4.8
BUILD Date and time         : 22-10-2019 at 11h02
GIT commit number           : 46
GIT hash                    : 0xdc11cebb
Firmware mode               : GBT
Number of descriptors       : 2
Number of interrupts        : 8
Interrupt count |      0 |      0 |      0 |      0 |      0 |      0 |      0 |
0 |
Interrupt flag  |      0 |      0 |      0 |      0 |      0 |      0 |      0 |
0 |
Interrupt mask  |      1 |      1 |      1 |      1 |      1 |      1 |      1 |
1 |
MSI-X PBA       00000000

Device 2:
Card type                   : FLX-711
```

```
Device type                : 0x7038
FPGA_DNA                   : 0x0105122244b0c205
Reg Map Version            : 4.5
GIT tag                    : rm-4.5
BUILD Date and time        : 5-7-2019 at 15h37
GIT commit number          : 198
GIT hash                   : 0xb2f51706
Firmware mode              : Unknown (firmware_mode = 7)
Number of descriptors      : 2
Number of interrupts       : 8
Interrupt count |      0 |      0 |      0 |      0 |      0 |      0 |      0 |
0 |
Interrupt flag  |      0 |      0 |      0 |      0 |      0 |      0 |      0 |
0 |
Interrupt mask  |      1 |      1 |      1 |      1 |      1 |      1 |      1 |
1 |
MSI-X PBA       00000000

Device 3:
Card type                  : FLX-711
Device type                : 0x7039
FPGA_DNA                   : 0x0105122244b0c205
Reg Map Version            : 4.5
GIT tag                    : rm-4.5
BUILD Date and time        : 5-7-2019 at 15h37
GIT commit number          : 198
GIT hash                   : 0xb2f51706
Firmware mode              : Unknown (firmware_mode = 7)
Number of descriptors      : 2
Number of interrupts       : 8
Interrupt count |      0 |      0 |      0 |      0 |      0 |      0 |      0 |
0 |
Interrupt flag  |      0 |      0 |      0 |      0 |      0 |      0 |      0 |
0 |
Interrupt mask  |      1 |      1 |      1 |      1 |      1 |      1 |      1 |
1 |
MSI-X PBA       00000000

The command 'echo <action> > /proc/flx', executed as root,
allows you to interact with the driver. Possible actions are:
debug     -> Enable debugging
nodebug   -> Disable debugging
elog      -> Log errors to /var/log/message
noelog    -> Do not log errors to /var/log/message
rm3       -> Enable compatibility with RM3 F/W
rm4       -> Disable compatibility with RM3 F/W
swap      -> Enable automatic swapping of 0x7038 / 0x7039 and 0x427 / 0x428
noswap    -> Disable automatic swapping of 0x7038 / 0x7039 and 0x427 / 0x428
clearlock -> Clear all lock bits (Attention: Close processes that hold lock bits
before you do this)
```

Driver Flags

The `/proc/flx` interface makes it possible to toggle certain parameters by issuing the following command:

```
echo <action> > /proc/flx
```

By substituting `<action>` it is possible to do the following (only a selected list below):

- Enable/disable compatibility mode with RM3 with the parameter 'rm3' or 'rm4' respectively.
- Enable/disable automatic re-ordering of FELIX cards to a more intuitive order w.r.t device type ('swap' or 'noswap').
- Clear all device locks with 'clearlocks'

5.2.2. Installation of FELIX Software Suite

The FELIX software release is available pre-compiled as a tarball which can be installed anywhere and then set up for use by running a command line script. Each user can download their own version, or the release can be installed centrally and the location of the script shared with users.

To unpack the tarball, run the following command:

```
tar -xvzf <filename>
```

Once unpacked, a setup script must be run to enable access to all libraries and binary files. The script can be run as follows from the release base directory:

```
source felix-04-01-00/x86_64-centos7-gcc8-opt/setup.sh
```

This script will need to be run with every new session, or added to the environment setup procedure. Once complete you should have access to all FELIX software. In the next section we will describe how to test your installation to verify full functionality.

5.2.3. Installation of FELIX rpm

An rpm of FELIX is available from the Release Distribution Site. The rpm needs to be normally installed. v `yum install felix-04.01.00-1.el7.cern.x86_64.rpm`

No further setup is needed. FELIX commands will be available on the path and users can compile and link against the FELIX libraries.

5.2.4. Installation of FELIX in CVMFS

Installation of the latest and nightly versions of the FELIX software are available in cvmfs under:

```
/cvmfs/atlas-online-nightlies.cern.ch/felix/releases
```

```
/cvmfs/atlas-online-nightlies.cern.ch/felix/nightlies
```

to set up run:

```
source felix-04-01-00/x86_64-centos7-gcc8-opt/setup.sh
```

Chapter 6. Basic Tools

The FELIX software suite comprises both high and low level tools. At the highest level, the **felixcore** or **felixstar** application is responsible for communication and bulk dataflow in a full slice system. At a lower level, the suite provides a number of tools, both command line and GUI based, to facilitate system configuration and testing. This chapter will describe these low level tools such that users will be able to effectively communicate with, configure and test their system.

If you are looking to set up a full system slice with data output to a network please consult Section 7, which describes the **felixcore** application and **netio** library. This section assumes that you have set up your FELIX software environment as described in Section 5. None of the tools in this section should require superuser privileges to run. All tools presented below work in both GBT and FULL mode, and for VC-709 and BNL-711 or BNL-712 cards, unless otherwise stated. Where special parameters are needed to distinguish modes this will be indicated.

A quick reference for all tools to be covered in this section is presented in Table 1.

the FELIX software suite contains a number of tools which are considered for developer use only. All tools which are rated for use by front-end users are listed in this document. Use of any other software is not recommended unless asked to do so by a FELIX developer.

Table 1. List of all recommended user tools. For more information on each please click the tool name to visit the dedicated section of this chapter.

Low Level Tools	
flx-info	View FELIX hardware and firmware information, LTC2991[ltc2991] or Minipod devices on a BNL-711 or BNL-712.
flx-config	View and modify low-level firmware parameters by reading and writing FELIX firmware registers.
flx-init	Initialise FELIX, as well set as low level GBT and clock/jitter cleaning parameters.
flx-reset	Reset FELIX or specific component.
fcap	View FELIX GBT E-link configuration capabilities (an addition to the information from flx-info).
felix-cmem-free	Manually deallocate memory in CMEM buffer.
Dataflow Tools	
fdaq	Receive data from a single FELIX device and save to files or perform sanity checks.

Low Level Tools	
fdaqm	Receive data from multiple FELIX devices and save to files or perform sanity checks.
fupload	Upload data through FELIX to a front-end E-link.
FELIX Configuration Tools	
elinkconfig	GUI for link and data generator configuration.
felink	Calculate E-link IDs given inputs with differing formats.
fereverse	Reverse the endianness of data passing through an E-link.
fgpolarity	Switch 0/1 polarity of all data coming or going through a specific GBT link.
feconf	Upload link and/or data generator configuration to FELIX from the command line.
femu	Control the FELIX data generators.
fttcemu	Control the FELIX TTC data generator.
fttcbusy	View FELIX E-link TTC-BUSY status and settings, as well as other BUSY-related settings.
feto	Configure FELIX timeouts (global, TTC and link data, a.k.a *instant timeout*).
fflash	Command line tool for loading a firmware image in BNL-711 or BNL-712 from the card's flash memory into the card's FPGA.
fflashprog	Command line tool for programming and verifying firmware images in BNL-711 or BNL-712 onboard flash memory.
General Debugging Tools	
fcheck	Perform configurable sanity checks on data from a file from fdaq or dump selected data chunks or blocks to screen.
fedump	Dump data blocks from a FELIX device (or selected E-link) to screen.
Remote Communication and Configuration Tools	
fice	Read or write GBTX chip registers via the GBT-link IC channel.
fgbtxconf	Read or write GBTX registers via a GBT-SCA I2C channel.
Tools for GBT-SCA access	

Low Level Tools	
fec	Demo control and communication with a GBT-SCA chip (GPIO, ADC, DAC and/or I2C).
fscaid	Read and display a GBT-SCA chip ID.
fscaio	Set and get GBT-SCA GPIO lines.
fscaadc	Read out GBT-SCA ADC input channels.
fscadac	Set and get GBT-SCA DAC output channels.
fscai2c	Read and write to I2C devices connected to GBT-SCA I2C channels.
fscads24	Read out a 1-Wire 64-bit ID chip connected to a GBT-SCA GPIO pin.
fscajtag	Program a bit-file into a Xilinx FPGA connected to a GBT-SCA JTAG port.
fxvcserver	Interfaces Vivado with a GBT-SCA chip's JTAG port (and connected Xilinx FPGA(s)).

6.1. E-link Configuration with elinkconfig

Before FELIX can be used to transfer data its input and output links must be configured. The link configuration for a given FELIX card can be accessed and modified using the *E-link configurator* application called **elinkconfig**. This is a GUI based tool to compile an E-link configuration or to inspect and/or edit the E-link configuration read from a given card, and to write the configuration and/or emulator data contents to a selected device (a BNL-712 card consists of 2 devices). The tool supports both GBT and FULL mode.

the link configuration must be manually refreshed every time a FELIX FPGA is reprogrammed, including power-cycling of a host, or after having executed flx-reset to reset the registers to their defaults.

To run the **elinkconfig** application issue the following command:

```
elinkconfig&
```

The main configuration panel similar to the one shown in Figure 14 will appear.

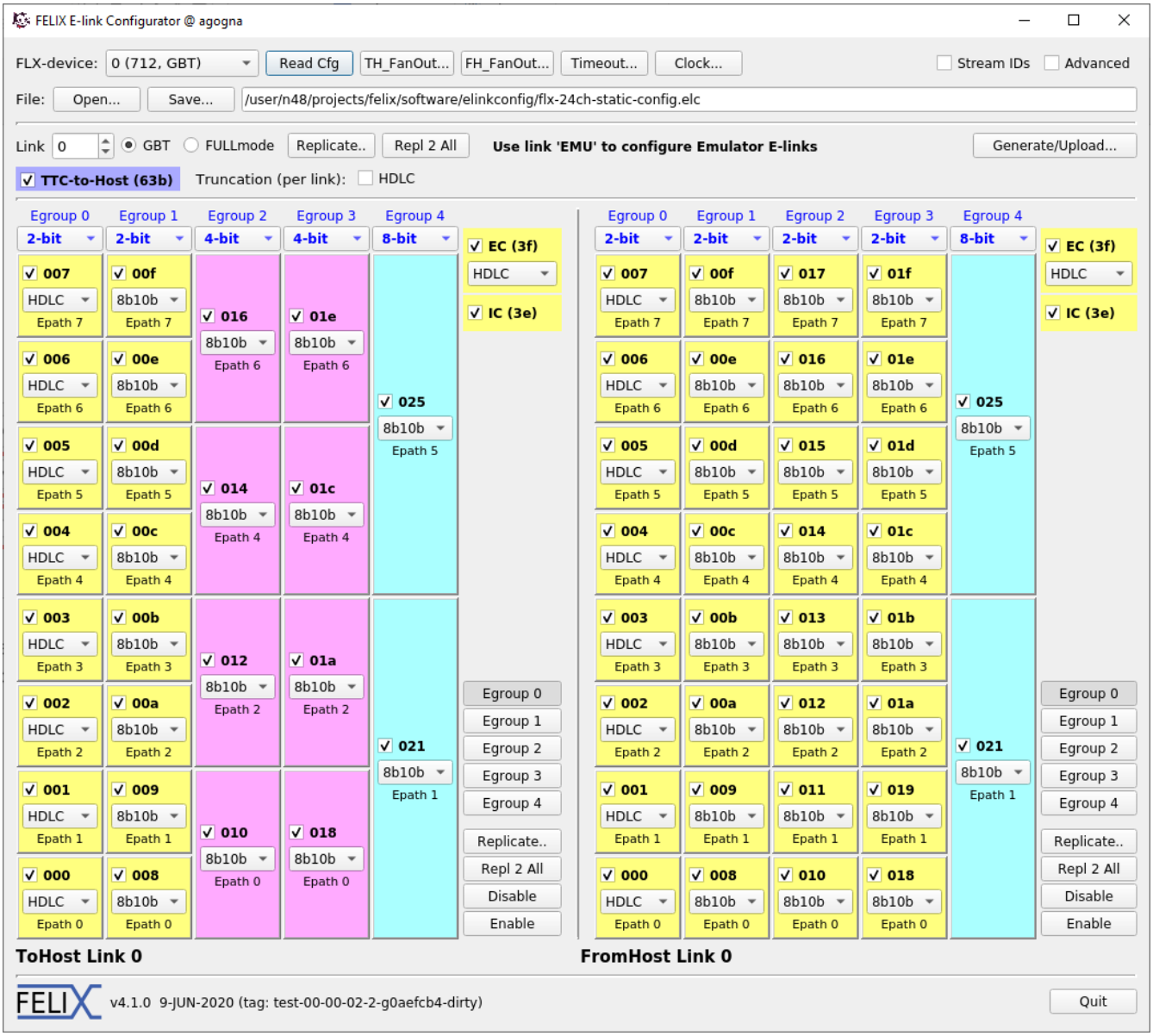

Figure 14. Main panel - elinkconfig

The **elinkconfig** interface is split into three main areas. At the top there are two control bars to set FELIX card parameters, open/save configuration files as well as link selectors. The left main panel displays the from front-end to FELIX/host configuration for the selected link.

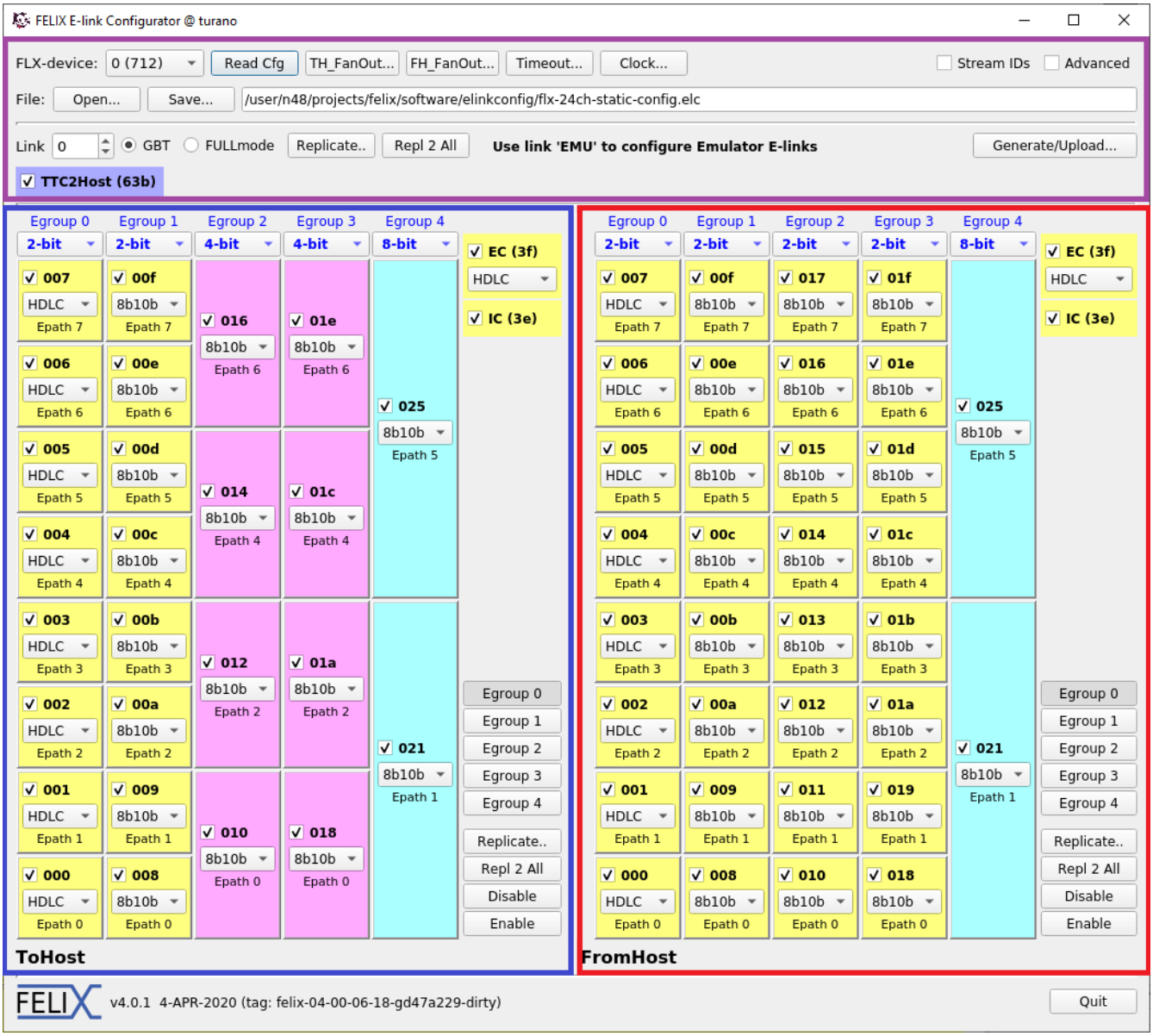

Figure 15. elinkconfig panel split. The uppermost panel (purple box) controls global settings and GBT selection. The left main panel contains the E-link configuration for the from front-end to host direction, the left main panel the from host to front-end direction

6.1.1. Global Panel

The **elinkconfig** global panel, shown in more detail in Figure 16 provides the top level interface for the tool. From there it is possible to select from which FELIX device within your system you wish to read out its (link) configuration, or read and set a number of global settings (using the top row of buttons). Note that a BNL-712 card consists of 2 separate FELIX devices (as shown in the FELIX device selection dropdown menu). From there one selects a link number for display and/or configuration in the ToHost and FromHost panels (see below), as well as an associated link mode (note that the link mode is a global card parameter; different links can not have different modes). It is also possible to open previously saved configuration files and save new ones. Also there is a button to open the dialog to write the selected or manually configured link configuration to any of the FELIX devices in your system. There is also a checkbox to enable *truncation* of data chunks on HDLC E-links, so that chunks with a size larger than can be expected from a GBT-SCA device (which is 12 bytes maximum) from such links are suppressed (it happens sometimes that unconnected links produce a lot of random data).

This panel also contains an advanced feature (if you tick 'Advanced') allowing you to select the maximum chunk size for a given E-link width - users are advised not to change these settings as they may cause unexpected behaviour (NB: currently this feature is broken, and should not be used at all).

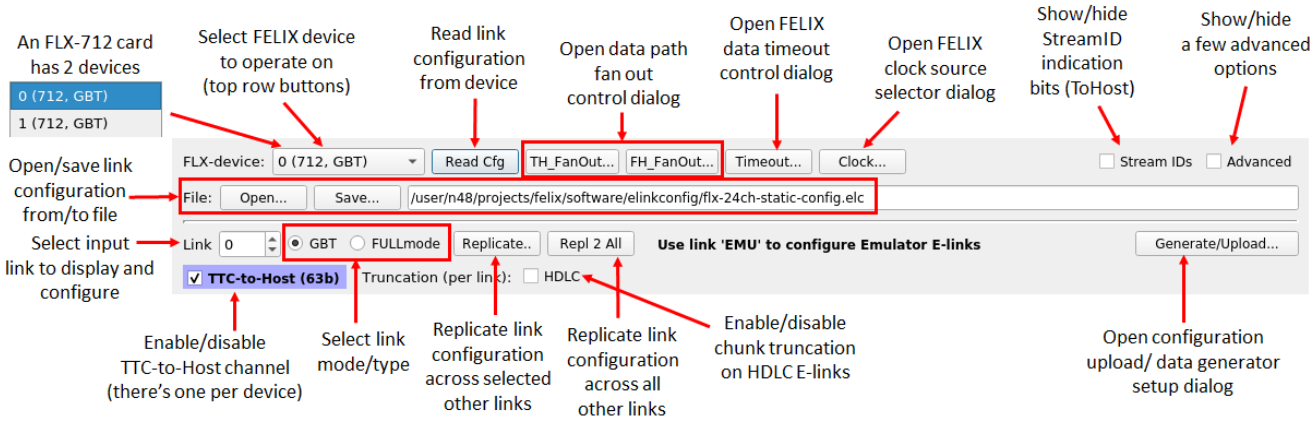

Figure 16. **elinkconfig** *global panel.*

From the global panel it is possible to access a number of sub-panels, as indicated in Figure 16. These give access to more advanced global configuration options, details of which are presented below.

The global panel also contains a tickbox to enable the socalled TTC-to-Host channel, a virtual E-link carrying for each TTC Level-1 Accept a packet containing the corresponding TTC information.

Data Path Fan Out Selectors: TH_FanOut and FH_FanOut

FELIX operates two separate data generators within its firmware, one attached directly to the data path going to the host, and one attached to the path going towards the front-end. While the generators are attached, they have mutually exclusive access to the data path with regular non-emulated data in both directions. To avoid the two data types colliding only one type may access the path at a time. The fan out selectors control this access by ensuring that only internally emulated data or external data can be configured to pass at any one time. The FELIX applications and tools configure these selectors automatically, but for the purposes of user testing it may be necessary to set these values manually. The selectors are accessed via the TH_FanOut (to host) and FH_FanOut (from host) buttons in the global panel. The resulting dialogs are presented in Figure 17.

Figure 17. Fan out control for to-host (top) and from-host (bottom) directions. The setting for each link is displayed separately (in this case for a 12-link FLX-712 device, i.e. for a 24-link BNL-712 card). It is also possible to (soft)lock the settings using the dedicated checkbox, meaning that certain tools will not touch the settings when the lock is set.

In order to switch the selector value simply open the required dialog and click on the link number you wish to toggle. A link displayed with its number alone is set to external data, if a link is displayed with its number plus 'E' it is in emulator mode. It is possible to set/unset all values at once using the *All* and *None* buttons provided.

Changes made here are immediately propagated to the selected FELIX device once you select *OK*.

In some cases a user may wish to prevent other applications from automatically changing these settings. For example, if a specific link is nominated for TTC information transfer it may be convenient to fix this to external data for the duration of a test. In this case it is possible to lock the values by selecting the *locked* check box. Applications will then refrain from changing these settings until the card is reconfigured from this interface or the FPGA is reprogrammed. More information on configuring TTC transfer to the front-end are available in Section 6.1.3 below.

Data Timeout Control Dialog

FELIX offers the facility to time out pending incoming data after a configurable window from receipt of the first related packets. This is applicable for both regular and TTC data (in the to-host direction). Should data time out then all available blocks are transferred to the host. The timeout feature is enabled by default, but can me modified or disabled/re-enabled via the control dialog accessible by selecting the *Timeout* button in the global panel. This will open the dialog shown in Figure 18. From here it is possible to disable/enable both regular data and timeouts on the TTC-to-Host channel using the check boxes, as well as modify the timeout window sizes. This should typically only be done under the guidance of a FELIX developer for debugging purposes.

Changes made here are immediately propagated to the selected FELIX device once you select *OK*.

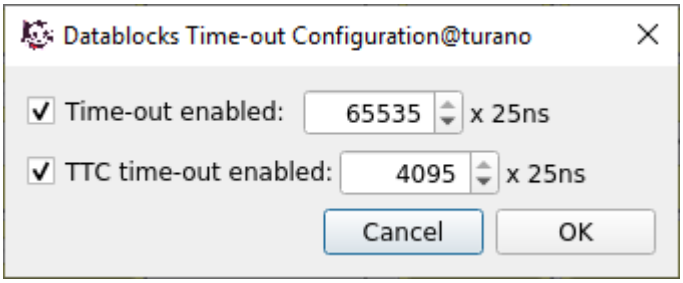

Figure 18. elinkconfig data timeout control dialog.

EC and TTC2Host data are always subjected to an immediate time-out, independent of the global time-out. In addition it is possible to enable a time-out per E-link independent of the global time-out setting, which may be important for E-links carrying irregular and small data fragments such as those connected to GBT-SCA devices.

Clock Source Selection Dialog

As mentioned in Section 3.5.1, FELIX supports two different firmware clock sources. It is possible to switch between these sources from **elinkconfig** from the clock source selection dialog, accessible by clicking the *clock* button in the global panel. The selection dialog is shown in Figure 19, and is a simple two button toggle between TTC and local clock.

Changes made here are immediately propagated to the selected FELIX device once you select *OK*.

Please also consult Section 3.5.3 before making any clock changes, to ensure you correctly configure your FELIX card's jitter cleaner post-clock change to ensure continued stable operation.

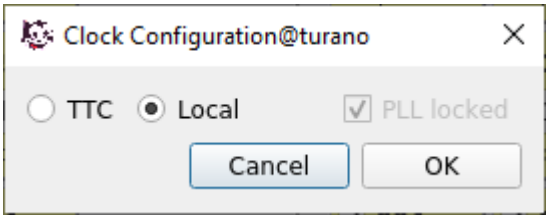

Figure 19. **elinkconfig** *clock source selection dialog.*

6.1.2. ToHost Panel

The *to-host* panel provides access to the configuration of the currently selected link (GBT or FULL mode) in the to-host direction. The type of panel to show can be selected in the global panel as described in Figure 16. In the GBT case it is possible to configure the complete set of E-links associated with this link, split up by E-group, as well as the EC (External Control) and IC (Internal Control) channels. For each link it is also possible to select the type of encoding to be used, although 8b10b is recommended for all regular data links. If the *Stream IDs* tickbox in the global panel is ticked the panel shows Stream ID indication tickboxes. It is a bit per E-link on the device that can be set to indicate e.g. to data-acquisition software that the protocol on the corresponding E-link features a *Stream ID*, which can then be taken into account by the software. A more detailed look at this panel is presented in Figure 20.

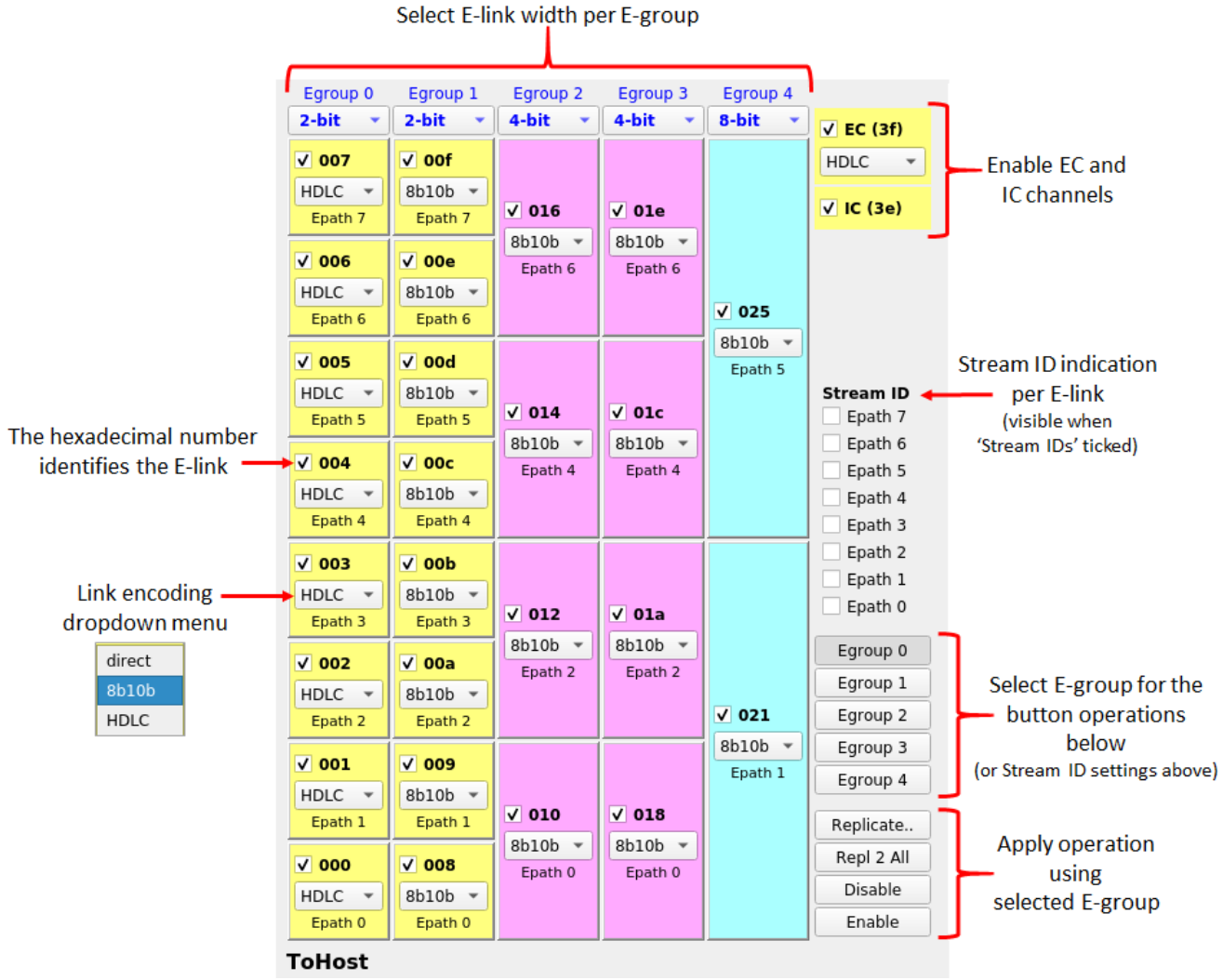

Figure 20. **elinkconfig** *configuration panel for to-host direction (GBT mode). Various configuration options and tools are indicated as they appear in the panel.*

In FULL mode this panel provides only a few options, as this link mode does not contain logical E-link subdivisions. This version of the panel is presented in Figure 21.

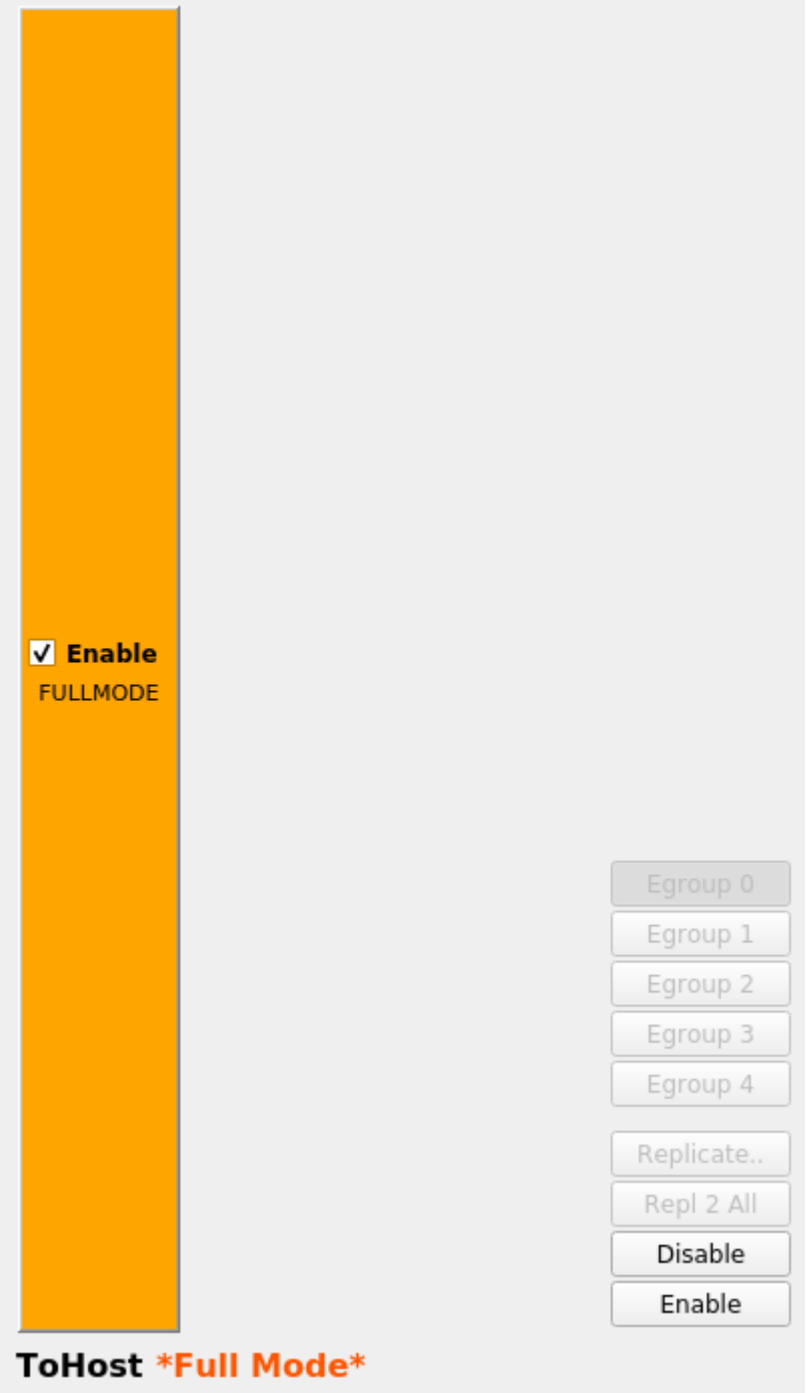

Figure 21. **elinkconfig** *to-host panel (FULL mode).*

6.1.3. FromHost Panel

The *from-host* panel makes it possible to configure the GBT links transporting data from FELIX towards connected front-end electronics. This panel only exists in GBT mode form as FULL mode is only a to-host protocol, and any FULL mode firmware will implement from-host links as GBT. A more detailed look at this panel is presented in Figure 22. A key difference between this panel and the to-host panel is that the link encoding available also includes several different TTC paths (in this case, for a 2-bit E-link TTC-0 and TTC-6 are shown) which are for the propagation of TTC information from FELIX to the front-end. Depending on the E-link width used TTC paths from 0 to 6 are made available. Using this encoding selector it is therefore possible to nominate specific E-links to carry TTC data as needed.

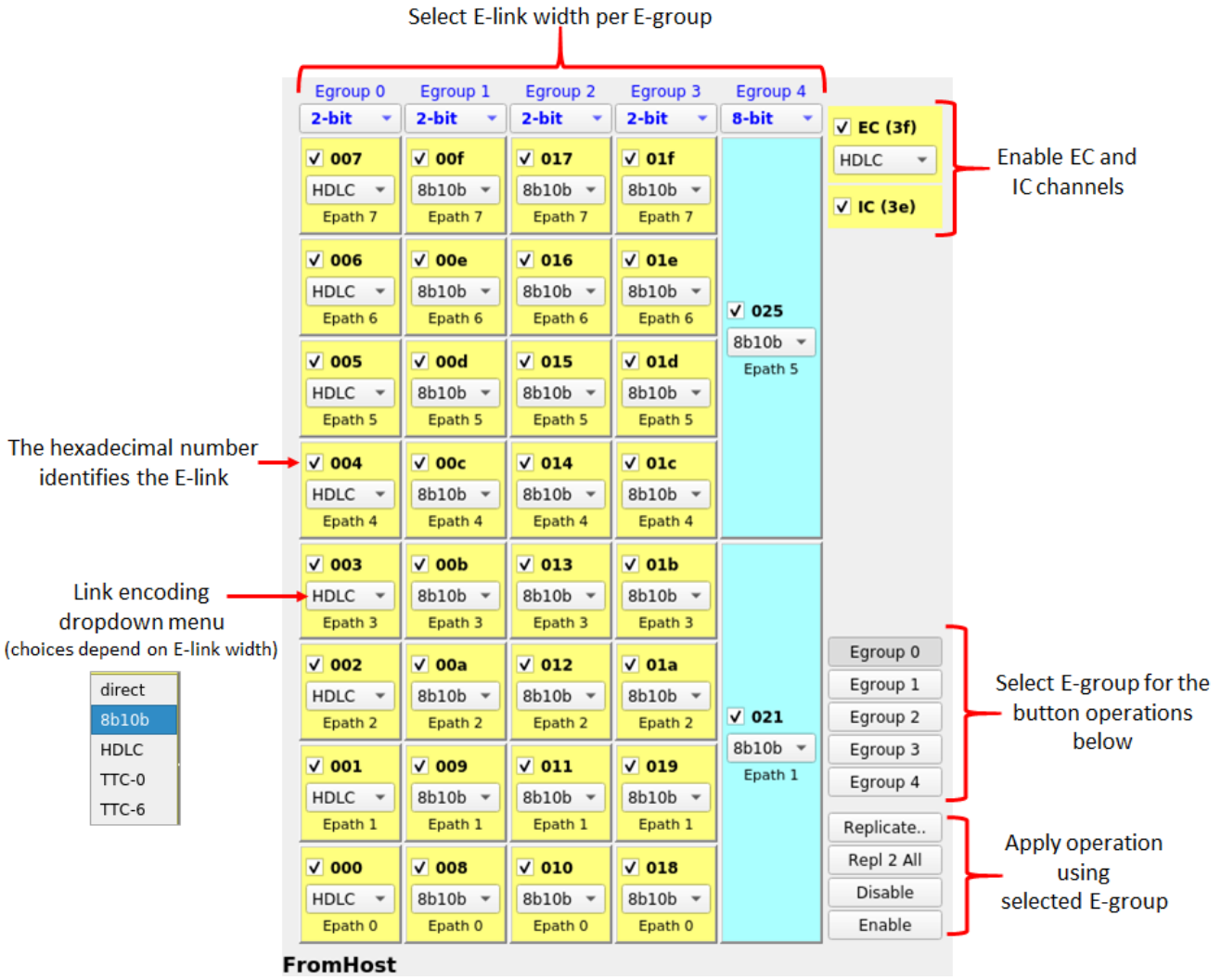

Figure 22. **elinkconfig** *configuration panel for from-host direction (GBT mode). Various configuration options and tools are indicated as they appear in the panel.*

6.1.4. Link and Data Generator Configuration Upload Dialog

The to-and-from host panels allow you to put together a complete configuration set for all links handled by a given FELIX card. Once you have prepared your desired configuration, you can upload it to a FELIX device in the current host machine by selecting the *Generate/Upload* button in the upper panel on the right. This will open the upload dialog, as shown in Figure 23. The GBT version is shown, but the FULL mode variant is essentially identical, beyond some disabled developer features.

The E-link mapping for the FELIX data generators can be configured by selecting the *EMU* link in **elinkconfig** (in previous versions this was done by selecting GBT link 0). If the option to save to a file is used the emulator link configuration is now saved separately to the rest of the links. If an older configuration file (which does not contain the separate emulator configuration data) is read into the tool and uploaded, the configuration of link 0 is automatically used in its place.

Once the panel is prepared, select *Upload* from the middle box labeled *E-link Configuration* to write your configuration to the device (Note: you have to upload to both devices of a BNL-712 card if you want to configure the full card). If you also wish to configure the FELIX on-board data generators for tests in emulation mode select the *Upload* button in the lower *Emulator Data* box.

If you are running in emulation mode and wish to change your E-link configuration you must remember to upload to the emulator every time you upload a change.

In FULL mode the data generators will only produce FULL mode data in the to-host direction. In the to-front-end direction GBT data will be produced. In GBT mode GBT data will be produced in both directions.

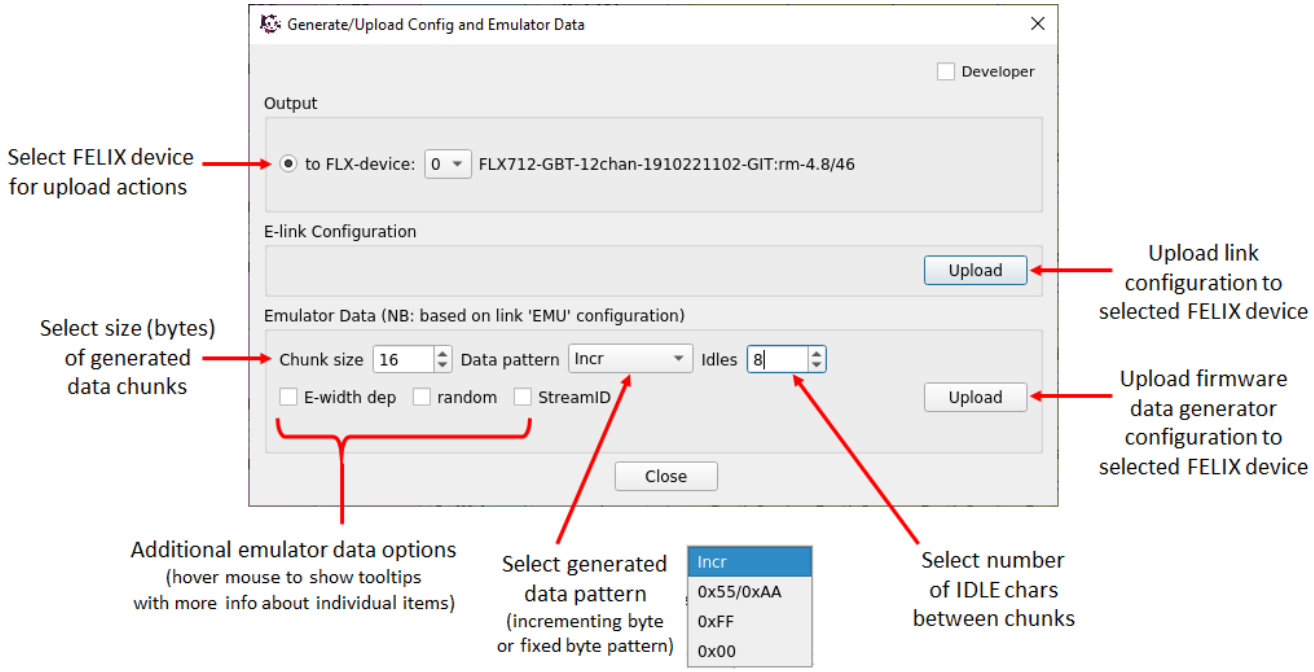

Figure 23. **elinkconfig** *upload panel.*

Once your configuration is uploaded you can then proceed to use the FELIX system as normal, the new settings will take effect immediately. To avoid unexpected behaviour please avoid reconfiguring the links while the FELIX device in question is in active use in your system.

6.1.5. Guide to Valid E-link Configurations

The E-link configuration uploaded to a FELIX card is actually a set of instructions to configure a component known as the *Central Router*. It is responsible for sending incoming data (in either direction) to the correct remote end point, as defined by E-link number. For FULL mode there is no such thing as an E-link, and so the Central Router merely propagates a wide stream of bits across the link. In the GBT case, E-links are defined as separate logical links within a given physical GBT link. E-links can have (in the current implementation) three different bit widths, which given the link clock defines the maximum bandwidth they can sustain. The widths are 2, 4 and 8 bits, running at 40, 80 and 160 MHz respectively. There is currently no support for 16 bit E-links. A GBT link can therefore be considered as a logical aggregation of low bandwidth links into one high bandwidth transfer. For full details please consult the official documentation [GBTx].

In *Normal* mode, a GBT link is 80 bits wide, and this puts an upper limit on the number of E-links. It is therefore possible to have few wide 8 bit links, a larger number of narrower 2 or 4 bit links, or a mixture of the two. Should a GBT be operated in *Wide* mode (not currently supported) then a further 32 bits are available within the GBT link (i.e. 112 in total), allowing for more E-links. The structure of a normal mode GBT frame is shown in Figure 24. It is up to the user to decide how

much of the GBT width to utilise as per their front-end needs. It is permitted to leave link bandwidth unused by not assigned E-links to that part of the GBT frame.

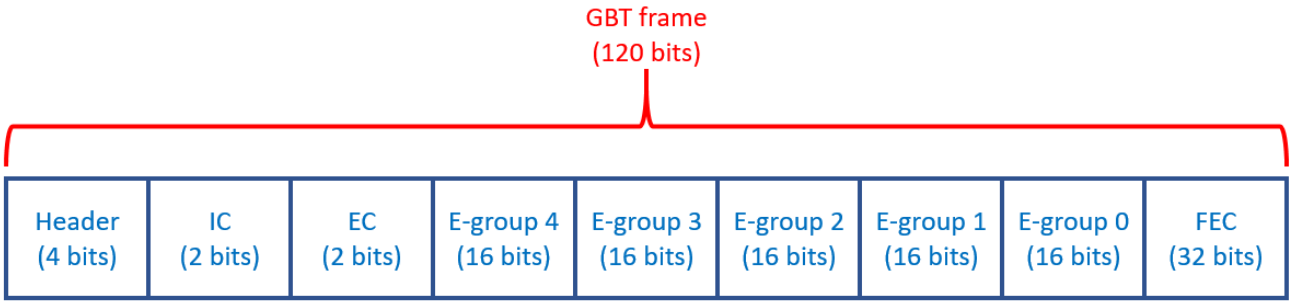

Figure 24. Bit structure of a GBT frame, showing E-groups, IC and EC links, as well as GBT header and Forward Error Correction (FEC).

Within a given GBT link, logical links are subdivided for management purposes into 16 bit wide *E-groups*. Each E-group logically contains a combination of E-links up to an aggregate of 16 bits of width, looking at either extreme this means up to 8 of the narrowest 2 bit E-links at one end, or two of the widest 8 bit E-links at the other. The E-group is the unit of connectivity around which the **elinkconfig** interface is built, with the to and from-host panels designed around the E-group granularity of one complete GBT link.

Looking within the E-group, there is one further layer of link identification to consider. Each group supports up to 8 logical *E-paths*. These correspond to the logical connection end-points which the Central Router supports. In the to-host panel in Figure 20 and the from-host panel in Figure 22 this is reflected in the "Epath" labels shown in each of the E-link selection boxes. Depending on the chosen E-link width in a an E-group certain E-paths are used and some not (for the 4-, 8- and 16-bit widths). The *E-path* used for a particular E-link is reflected in the E-link IDs (an 11-bit significant hexadecimal number) in the E-group, so an E-link ID may not be unique for different E-link widths (for example, certain E-link IDs may refer to either a 2-bit or 4-bit wide E-link).

Within a given link map, each E-link can be configured to use different encoding formats as per front-end requirements. This area is still subject to active development, and it is strongly recommended that users work towards basing systems on 8b10b encoding. For FULL mode 8b10b is also the default.

There is a known issue with GBTX chips whereby links disconnected from any front-end source generate spurious data at random intervals. If using FELIX with a GBTX it is strongly recommended that any links which are disconnected from the front-end be deactivated in the FELIX device using **elinkconfig**. This will prevent spurious data causing confusion in front-end testing.

Semi-Static Firmware E-link Configuration

In the 24-channel GBT mode build, the FELIX firmware does not support fully configurable E-links due to the need to conserve FPGA resources. The set of configurable E-links in this case is described below.

	To Host	From Host
EC link	2 bit HDLC	2 bit HDLC
E-Group 0	2 bit HDLC, 8 bit 8b/10b	2 bit HDLC
E-Group 1	2 bit HDLC, 2 bit 8b10b, 8 bit 8b/10b	2 bit HDLC, 2 bit 8b10b
E-Group 2	4 bit 8b/10b, 8 bit 8b/10b	2 bit 8b/10b
E-Group 3	4 bit 8b/10b, 8 bit 8b/10b	-
E-Group 4	8 bit 8b/10b	8 bit 8b/10b

6.1.6. Guide to common configuration tasks

Working with E-link configurations stored in files

elinkconfig can read and store configuration sets in .elc files. In order to load a previously existing configuration set into the tool simply select *Open* from the global panel and choose the file to be loaded. The GUI will be automatically updated to reflect the new configuration. From here you can modify the configuration (if needed) by e.g. using the to and from-host panels to enable/disable E-links. Once your changes are complete you can upload the new configuration to the FELIX card of your choice using the *Generate/Upload* button in the global panel. Make sure to upload both the link and data generator configurations if you wish to use the latter. Finally, you can save your modified configuration to a file by selecting *Save* from the global panel.

Modifying the existing E-link configuration on a FELIX card without a file

If you are working without .elc files and wish you modify the existing configuration on a FELIX device you must first load it into the tool by selecting the device in question via the global panel and then pressing the *Read Cfg* button. This will populate the GUI with the configuration currently active on the device. From here you can modify the configuration as required and upload a new version to the device (or devices) as advised above. You can also save your configuration to a file for later reuse.

Note that after starting up the tool or after modifying the selected device number the *Read Cfg* button shows the button text in bold face, to indicate the configuration has not been read from the device yet and turns to non-bold as soon as it is read.

Configure the to-host Level-1 Accept info E-link (TTC E-link)

The FELIX firmware implements a dedicated 'virtual' E-link (*virtual*, because it is not an E-link in the GBT/FELIX sense) for the purpose of forwarding TTC Level-1 Accept information to the host system for transfer to subscribers on the network. Each FELIX endpoint (or device) provides one such 'E-link', which is activated by ticking the `TTC-to-Host` checkbox in the global (top) panel in **elinkconfig**, as shown in Figure 16. The E-link ID on which the data will be transferred is shown there as well. (The exact value of the E-link ID is dependent on the number of (GBT or other) links supported by the firmware.)

Such E-links produce a 26-byte *L1AInfo* data packet containing information about each Level-1

Accept. The contents of the packet are presented in Table 1 of appendix. More details on this and other FELIX data structures can be found in the appendix.

Configure the to-front end TTC E-links

Users may configure any number of to-front-end links for the purpose of transferring TTC information to their electronics. The TTC data arriving at the FELIX card will be automatically decoded, and subsets made available to users for relay to front-ends in a configurable manner. The subsets which can be sent depend on the width of the E-link chosen for the transfer.

The current configuration sets are presented in Table 2, although this can evolve based on user requirements. For example, 2-bit E-links can be configured in 'TTC-0' or 'TTC-6' mode, For 'TTC-0' mode only the L1A and the full, non-decoded B-channel data stream can be sent. Alternatively, 4-bit E-links can be configured to send L1Accepts, Bunch Counter and Event Counter Resets, and a choice of either the non-decoded B-channel data stream or a user defined broadcast bit. Detector groups should communicate to the FELIX group which bits in which locations they need. If their needs are not met by an existing option, an option can be added.

Table 2. Currently defined TTC options. Brcst[7:2] are the TTC user defined broadcast command bits. Brcst[1] is ECR, Brcst[0] is BCR. Bit 0 is the first bit transmitted out.

E-link option	**bit 7**	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
0: 2 bits							B-chan	L1A
1: 4 bits					B-chan	ECR	BCR	L1A
2: 4 bits					Brcst[2]	ECR	BCR	L1A
3: 8 bits	B-chan	Brcst[5]	Brcst[4]	Brcst[3]	Brcst[2]	ECR	BCR	L1A
4: 8 bits	L1A	Brcst[3]	Brcst[2] *2	ECR	BCR *1	L1A	Brcst[5]	ECR
5: 4 bits					BCR	BCR	BCR	BCR
6: 2 bits							BCR	BCR
7: 8 bits	L1A	Brcst[3] *3	Brcst[2] *2	ECR	BCR *1	L1A	Brcst[5]	ECR

TTC Option 4 is as requested by the New Small Wheel and FELIX performs custom functions for several of the input TTC bits, such as stretching to multiple BC's and copying input bits to more than one output bit. (Needed for compatibility with a two-level trigger)
Notes for Option 7:
*1 TTC Broadcast bit 7 is used to request that FELIX send two consecutive BCR's, which are used as OCR (Orbit Count Reset Request).
*2 Brcst[2], used for TestPulse, is held active by FELIX for 16 BC's.
*3 Brcst[3], used for SoftReset, is chosen from the six "toggle" versions of the Brcst[7..2] in TTC-7. Feature will be added in firmware version 4.10

The broadcast bits[7..2] do not behave as they do for the legacy TTCrx or TTCrq ASICs. For FELIX, these bits persist only as long as their transmission from the TTC system is repeated. Whereas for the legacy ASICs, they persist until they are transmitted again, whereupon they are inverted. It is intended to provide the legacy behavior as a per-bit option.

By selecting the encoding box (as shown in Figure 25) on the to-host panel for any given link it should be possible to see which TTC options are available for that link.

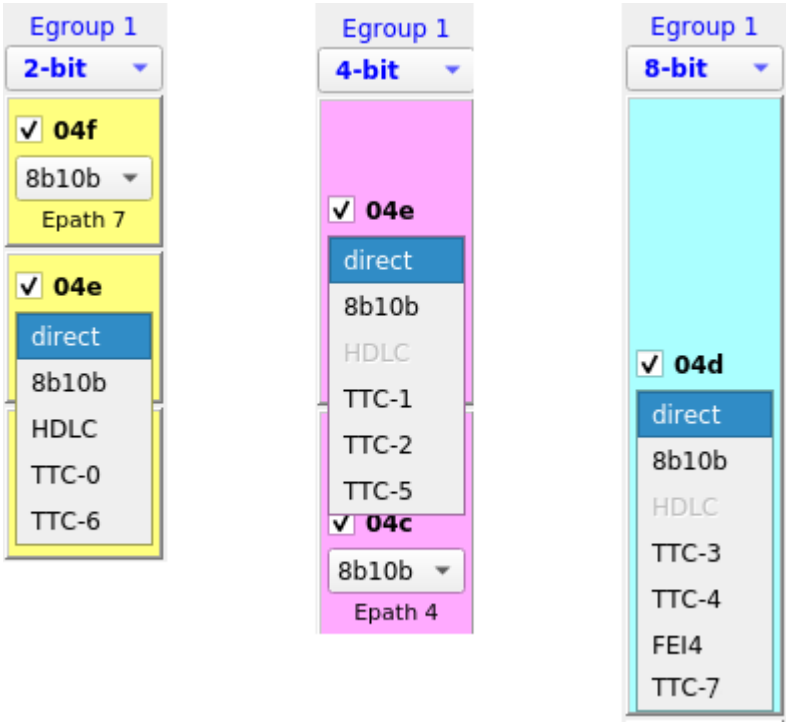

Figure 25. Example drop-down encoding menu for a 2-bit, a 4-bit and an 8-bit E-link, from left to right, resp. What the available options represent is shown in Table 2.

Configure GBT-SCA E-links to/from host

E-links connected to GBT-SCA ASIC devices are 2-bit wide HDLC-encoded links. Such E-links are designed to carry slow control information to and from any desired front-end location. Users may set any number of 2-bit E-links (in either direction) into this mode. Note that the "EC" (External Control) E-link is not restricted to be an HDLC E-link but can be a 2-bit 8b/10b E-link. Communication with a GBT-SCA ASIC requires both a to-host HDLC E-link and a from-host HDLC E-link enabled and connected to the GBT-SCA ASIC. They are usually, but not required to be, in corresponding positions in the E-groups. FELIX performs the HDLC encoding and decoding.

An OPC-UA server and client for the GBT-SCA ASIC are provided to allow high level communication with the GBT-SCA's I/O channels by user software.

IC channel

The GBT IC channel is a 2-bit wide channel in the GBT protocol dedicated to control of the GBTX chip at the remote end of the link. Provided the GBT link is up this channel is used for additional configuration of GBTX registers. In a FELIX device both IC *to-host* and *from-host* 'E-links' must be enabled for a user to gain access to his front-end GBTX devices.

6.2. Low Level Tools

The following section will cover some general tools which allow you to monitor the state of your FELIX system, as well as make configuration changes and reset the system as necessary. All tools provide more detailed descriptions of functionality through their help output, accessible by running the tool with option -h.

6.2.1. flx-info

The flx-info application is a command line tool which can print to the screen a range of monitoring and configuration information for your FELIX card(s). By default you will be presented with some system version and health status, as well as a basic link description. To access this default printout run the following command:

```
flx-info
```

This will give output similar to that what is shown below, including general information such as the firmware revision *GIT tag* and *firmware mode* 'GBT' or 'FULL'. To produce more verbose output pass option -v or -vv to the tool.

```
> flx-info
Card type: FLX-712

General information
----------------------------------
Card type           : FLX-712
Reg Map version     : 4.8
FW version date     : 19/10/22 11:02
GIT tag             : rm-4.8
GIT commit number   : 46
GIT hash            : 0x00000000dc11cebb
Active F/W partition: 3
Firmware mode       : GBT

Output of lspci:
19:00.0 Communication controller: CERN/ECP/EDU Device 0428
1a:00.0 Communication controller: CERN/ECP/EDU Device 0427
b5:00.0 Communication controller: Xilinx Corporation FPGA Card XC7VX690T
b6:00.0 Communication controller: Xilinx Corporation Device 7039

Interrupts, descriptors & channels
----------------------------------
Number of interrupts  : 8
Number of descriptors : 2
Number of channels    : 12

Links and GBT settings
----------------------------------
Number of channels    : 12
GBT Wrapper generated : YES
Optical transceivers  : 4

Clock resources
----------------------------------
MAIN clock source     : LCLK fixed
Internal PLL Lock     : YES

ADN2814 TTC Status    : ON
```

Beyond this basic output, flx-info also makes it possible to read many FELIX configuration registers in detail. The feature set that was previously implemented in the *flx-monitor* application has been merged into flx-info. These features include monitoring the Minipods' status and voltage/temperature monitoring through the LTC2991 [ltc2991] aboard BNL-711/712 cards (does not apply to the VC-709).

For a complete list of available options run flx-info with option -h. This will produce the output shown below:

Help text of flx-info:

```
Usage: flx-info [OPTIONS] [COMMAND] [CMD ARGUMENTS]
Displays information about a FLX device.

Options:
  -c NUMBER           Use card indicated by NUMBER. Default: 0.
  -D level            Set API debug output level.
                      0=disabled, 5, 10, 20 progressively more verbose output.
Default:0.
  -h                  Display help.
  -v                  Verbose mode.
  -V                  Display the version number
Commands:
  FPGA                Display the status of the FPGA
  LTC                 Display the status of the LTC2991
  POD                 Display the status of the active MiniPODs
    -a                Display the status of all MiniPODs (not only the active ones)
  TTC                 Display data from TTC related registers
    -z                Zero the counters
  FREQ                Display the RXUSRCLK frequency
  LINK (or GBT)       Display the channel alignment status.
  ELINK               Display the E-link alignment status.
  ADN2814             Display the ADN2814 register 0x4.
  CXP                 Display the temperature and voltage from CXP1 and CXP2
  SFP                 Display the information from Small Form Factor Pluggable
transceivers
  DDR3                Display the values from DDR3 RAM memory
  SI5324              Display the SI5324 status
  SI5345              Display the SI5345 status
  LMK03200            Display the LMK03200 status
  ICS8N4Q             Display the ICS8N4Q status
  EGROUP [chan] [RAW] Display the values from EGROUP registers:
                      If no channel is specified, display all available,
                      using hexadecimal notation if RAW is specified.
  ALL                 Display all information.
```

6.2.2. fcap

The fcap tool provides some information additional to what is obtained with flx-config, in particular the E-link configuration capabilities of FELIX GBT firmware. The firmware has a per-Egroup and per-direction setting for E-link configuration. Here's an example of the output of fcap:

```
> fcap
Firmware  : FLX712-GBT-12chan-2006041630-GIT:rm-4.9/244
Trailer   : 16-bit
Blocksize : 1024
(FromHost:YES DirectMode:NO Xoff:NO TTCemu:YES)
E-link configurability:
Egroup |    ToHost    |  FromHost
----------------------------------
   0       8,HDLC        HDLC
   1       2,8,HDLC      2,HDLC
   2       4,8           2
   3       4,8           ---
   4       8             8
```

Per *E-group* is shown what width and type of E-links can be configured, in both the *to-host* and *from-host* directions. For example '8' here means only 8-bit 8b10b-encoded E-links are possible for that group, '2,HDLC' means both 2-bit 8b10b-encoded or HDLC-encoded E-links can be configured for the group, 'HDLC' means the E-group consists of 2-bit HDLC E-links only. Individual E-links can always be disabled and enabled.

6.2.3. flx-config

The flx-config tool allows users to modify FELIX control and configuration registers from the command line. This should normally only be done on advice from a member of the FELIX development team. Other features are also available, but these should be considered for experts only unless advised otherwise by the development team. The two primary features users will use will be the 'list' and 'set' and 'get' features. List mode will dump the values of all known FELIX register items to the screen, including the address of the register containing the item (a register may contain multiple individual items), whether the item is readable and/or writable, the bits in the register the item occupies, its name, its *current* value and a short description of the item. This will be a large amount of output, but can be searched e.g by piping it through *more* or through *grep* with a keyword to get the desired information. To run list mode execute the following command:

```
flx-config list
```

To change a given register bitfield value (or item) use the 'set' feature, as follows:

```
flx-config set ITEMNAME=<val>
```

In this case ITEMNAME corresponds to the register bitfield to be changed and *<val>* to the new value be stored. Once set you can use list mode to confirm the change, or read the item explicitly, as follows:

```
flx-config get ITEMNAME
```

Note that instead of using 'set' and 'get' for single item write and read operations, the same can also be achieved as follows:

```
flx-config ITEMNAME <val>
```

```
flx-config ITEMNAME
```

If the given ITEMNAME does not correspond exactly to a known item, a list of items will be displayed containing the given name as a substring. Item names may be given in lower and/or upper case and with '_' characters replaced by '-'.

6.2.4. flx-init

This tool has the ability to reset the GBT wrapper and transceiver, and should be performed every time the FPGA is reprogrammed (including in case of loss of power). The tool should also be run if the GBT fibres are disconnected at any point before attempting to transfer data once again.

To run the basic initialisation issue the following command:

```
flx-init
```

If you wish to use more features (if instructed by a member of the development team), consult the help dialog shown below.

Help text of flx-init:

```
Usage: flx-init [OPTIONS]
Initializes an FLX device.
General:
  -d NUMBER           DEPRECATED ---- Use device indicated by NUMBER. Default: 0.
  -c NUMBER           Use card indicated by NUMBER. Default: 0.
  -h                  Display help.
  -D level            Configure debug output at API level.
                        0=disabled, 5, 10, 20 progressively more verbose output.
Default: 0.
  -E                  Execute the command even if resources are locked
  -v                  Display verbose output.
  -V                  Display the version number
GBT calibration:
  -a ONE|CONTINUOUS   Select alignment type. Default: ONE.
  -t FEC|WideBus      Select transmission mode. Default: FEC.
TTC calibration:
  -G NUMBER           Get and display the status of a SI53xx
                      Legal values are: 1 = SI5324, 2 = SI5345
  -C                  Set the registers of the ICS 8N4Q001L
  -T mode             Set a clock to a given frequency
                      Legal values for mode are:
                        -T not in command line = automatic default configuration
                        -T 0,1,2,3,etc.        = special configuration (currently not
supported)
  -I INSEL            To be used in combination with -T.
                      The value given will be written into register
HK_CTRL_FMC_SI5345_INSEL
                      Legal values are:
                        FLX-709: 0-->FPGA (LA01), 1-->FMC OSC, 2-->FPGA (LA18)
```

6.2.5. flx-reset

The flx-reset application makes it possible to selectively reset components of the FELIX firmware, or the complete board, as needed given the situation. This should only be done if advised by a FELIX development team member. To see the list of available parameters please consult the help output shown below:

Help text of flx-reset:

```
Usage: flx-reset [OPTIONS]
Tool to reset various resources on the card.

Commands:
Options:
  DMA_RESET        Resets the DMA part of the Wupper core.
  REGISTERS_RESET  Resets the registers to default values.
  SOFT_RESET       Global application soft reset.
  LINK (or GTH)    Reset the link RX (for FULL mode F/W only).
    -q             The individual quad (0..5) to be reset (default: all)
  ADN2814          Reset the ADN2814.
  ALL              Do everything. (Note: use -c, not -d:
                   resets the card and the resources of all devices of that card)
Options:
  -d NUMBER        Use device indicated by NUMBER (applies to
DMA/SOFT/REGISTERS_RESET; default: 0).
  -c NUMBER        Use card indicated by NUMBER (default: 0).
  -D level         Configure debug output at API level.
                   0=disabled, 5, 10, 20 progressively more verbose output (default:
0).
  -E               Execute the command even if resources are locked
  -h               Display help.
  -V               Display the version number

Note:
Use -c NUMBER with ADN2814 and LINK / GTH
Use -d NUMBER with DMA_RESET, SOFT_RESET and REGISTERS_RESET
If neither -c nor -d are given, card or device number 0 is used as appropriate.
```

To reset a given component simply pass the name to flx-reset on the command line:

```
flx-reset <component_name>
```

6.2.6. felix-cmem-free

The felix-cmem-free tool makes it possible for a user to manually deallocate memory from the CMEM buffer. In order to use the tool, the 'handle' of the allocated memory must first be found. To do this issue the following command:

```
cat /proc/cmem_rcc
```

This will dump the current allocation status in a format similar to what is shown below.

```
CMEM RCC driver (FELIX release 4.5.0)

The driver was loaded with these parameters:
gfpbpa_size    = 7500
gfpbpa_quantum = 4
gfpbpa_zone    = 0
numa_zones     = 1

alloc_pages and alloc_pages_node
   PID | Handle |         Phys. address |               Size | Locked | Order | Type |
Name

GFPBPA (NUMA = 0, size = 7500 MB, base = 0x0000000295c00000)
   PID | Handle |         Phys. address |               Size | Locked | Type | Name
 27549 |      0 |    0x0000000295c00000 | 0x0000000040000000 |     no |    4 |
FlxReceiver0

The command 'echo <action> > /proc/cmem_rcc', executed as root,
allows you to interact with the driver. Possible actions are:
debug    -> enable debugging
nodebug  -> disable debugging
elog     -> Log errors to /var/log/messages
noelog   -> Do not log errors to /var/log/messages
freelock -> release all locked segments
```

The 'handle' is shown in the second column in the 'GFPBGA' table, in the row according to the process whose memory you wish to deallocate. Finally, pass the handle to felix-cmem-free as follows:

```
felix-cmem-free <handle>
```

Future FELIX releases (software version 4.2 onwards) should incorporate more advanced automatic deallocation in the case of abnormal program termination, but in the short term felix-cmem-free should provide a manual workaround.

6.3. Dataflow from/to Front-end via FELIX to/from FELIX host PC

6.3.1. fdaq(m)

The fdaq tool is the primary tool for testing the FELIX data acquisition path (for a data stream from a single FELIX device; `fdaqm` is a version of the same tool to support multiple data streams, although the same could be achieved by running multiple instances of fdaq at the same time). The tool can run in multiple modes, from waiting for input for FELIX from a front-end source to running with one of the two internal data generators on the card activated, mostly for test purposes. In both modes fdaq will measure and report throughput for the duration of the test. Data can be dumped to a file or discarded upon receipt. If running in discard mode fdaq will check the integrity of the data blocks and chunks it receives (e.g. block headers and chunk sizes). If an error is found the test will,

by default, stop and fdaq will report on the first detected error. However, if option -D is used the run will continue with a regular report printed on all errors received.

This section assumes that your E-links and data generators are configured properly as specified in Section 6.1. In this section we will cover various scenarios, but a list of all options can be found in the help output, shown below.

In full mode it will likely only be possible to run fdaq or a couple of seconds as you will rapidly exceed the maximum rate at which you can write to disc, which will then cause the application to abort to avoid buffer overflow. For reference, for a typical SSD this limit is approximately 300MB/s.

Help text of fdaq:

```
fdaq version 21031200
Stream data from FLX-device to file(s). Whenever the set maximum file size
is exceeded a new file is created. Every second a status line
with data rates, data totals and memory buffer status is displayed.
(NB: if no filename is provided all data is consumed while checking the data blocks,
     i.e. blockheader and trailers; chunk truncation and error counts are reported.)

Usage: fdaq [-h|V] [-D] [-d <devnr>] [-b <size>] [-e|E] [-f <size>]
            [-i <dma>] [-I] [-r <runnr>] [-t <secs>] [-C] [-R] [-T] [-X] [-o]
[<filename-base>]
  -h         : Show this help text.
  -V         : Show version.
  -C         : Do *not* check for presence of data chunk CRC errors (when not writing
to file).
  -D         : Debug mode on, i.e. output some additional info;
               continue when memory buffer overflows.
  -d <devnr> : FLX-device to use (default: 0).
  -b <size>  : DMA (cmem_rcc) memory buffer size to use, in MB (default 1024, max
4096).
  -e|E       : Enable FLX-device data generator, internal (e) or
               external (E) (default: false).
  -f <size>  : Maximum file size, in MB (default 4096, max 8192).
  -i <dma>   : FLX-device DMA controller to use (default: 0).
  -I         : use interrupt to receive data (default: polling)
  -o         : Display status output not in columns (as before).
  -r <runnr> : Run number to use in file names (default: none).
  -R         : Flush and reset DMA and issue a 'soft reset'
               at startup (default: no resets).
  -t <secs>  : Number of seconds to do acquisition (default: 1).
  -T         : Do NOT add datetime as part of file names.
  -X         : Stream data from individual e-links to separate files (default: false).
 <filename-base>: Name to be combined with datetime+runnumber+counter of files created
                  (unless option -T is given)
```

Running a DAQ Test with External Data Source

The most simple configuration for fdaq to run in is to listen for any data coming into FELIX over the GBT/FULL mode link and measure the bandwidth as this arrives at the host. In this mode the data is discarded. The only parameter a user must define is the time in seconds for which fdaq should perform the test. The default time is 1 second. The syntax is as follows:

```
fdaq -t <secs>
```

For a three second test the output will resemble this:

```
$ fdaq -t 3
Consume FLX-device data while checking the data (blockheader and trailers),
counts errors including chunk truncation, halts when the memory buffer is near
overflowing.
Also counts chunk CRC errors.
Opened FLX-device 0, firmw FLX712-MROD-48chan-2004041603-GIT:RM4.10/1 (cmem
buffersize=1024MB)
**START** using DMA #0 polling
  Secs | Recvd[MB/s] | File[MB/s] | Total[(M)B] | Rec[(M)B] | Buf[%] | Wraps
-------|-------------|------------|-------------|-----------|--------|-------
     1           0.0          0.0             0           0        0       0
     2           0.0          0.0             0           0        0       0
     3           0.0          0.0             0           0        0       0
**STOP**
-> Data checked: Blocks 0, Errors: header=0 trailer=0
Exiting..
```

If you would like to dump your data to a file for analysis simply specify a filename after the other command line parameters:

```
fdaq -t <secs> testfile
```

This will run as above and produce a time-stamped .dat file in the directory you are running with a name of the format 'testfile-<timestamp>.dat'. You can specify the maximum size for the file with option -f specifying a size in megabytes (default 1024, max 4096). If you would like to split the input from multiple E-links into different files use option -X. The E-link numbers will become part of the filenames.

Running a DAQ Test with Internal Data Generation

A facility to use both data generators within the FELIX card for the purposes of testing is provided by fdaq. The 'internal' generator is connected directly to the data output path of the card (i.e. after input side of the GBT link interface). Data from this generator therefore passes through the full FELIX firmware data path with the exception of the link layer itself. The 'external' data generator is connected to the output path before the GBT layer. This means it can be configured to send data out of a GBT link. If some loopback fibres are connected it is therefore possible to send data out of one GBT transceiver and into another on the same FELIX and therefore test more of the data path. To access these options in fdaq one must use option -e for internal data generation and option -E for external data generation. The output from fdaq will be the same as shown for the external source

tests. Here is an example:.

```
$ fdaq -t 5 -e
Consume FLX-device data while checking the data (blockheader and trailers),
counts errors including chunk truncation, halts when the memory buffer is near
overflowing.
Also counts chunk CRC errors.
Opened FLX-device 0, firmw FLX712-GBT-12chan-1910221102-GIT:rm-4.8/46 (cmem
buffersize=1024MB)
**START(emulator)** using DMA #0 polling
  Secs | Recvd[MB/s] | File[MB/s] | Total[(M)B] | Rec[(M)B] | Buf[%] | Wraps
-------|-------------|------------|-------------|-----------|--------|-------
     1        1449.5          0.0        1449.5           0        2       1
     2        1451.2          0.0        2900.8           0        1       2
     3        1451.0          0.0        4351.8           0        3       4
     4        1451.5          0.0        5803.3           0        2       5
     5        1451.4          0.0        7254.7           0        1       6
**STOP**
-> Data checked: Blocks 7072497, Errors: header=0 trailer=0
Exiting..
```

6.3.2. fupload

The FELIX software suite makes it possible to transfer data from the FELIX host PC via the FELIX card to the front-end across any GBT E-link. This is done using the fupload tool. With this tool it is possible to transfer data either from a user defined file, or with predefined data chunks of a configurable size and data pattern on a specified E-link across a GBT connection. The full range of features of the tool can be seen in the help text shown below.

This tool works with FULL mode firmware versions, but uses a GBT link up to the front-end.

Help text of fupload:

```
fupload version 21011500
Upload data (test data or from file) to the given FLX-device E-link.
The E-link number is provided as a (hex) number directly (-e option),
as a set of -G/g/p options, or as a set of -G/I/w options,
unless option -R is given ('raw' unformatted upload).

Checks whether the E-link is valid and configured on the selected FLX-device,
unless option -c is given.

In ASCII data files one line is one data packet (hexadecimal byte values separated by
spaces),
while lines starting with certain characters may be used to:
  # insert a comment line
  + insert a packet of the given length containing bytes of the given byte value
  & insert a configurable delay in microseconds between two packets
```

```
  > change the E-link number to upload to

Usage: fupload [-h|V] [-D] [-d <devnr>] [-b <size>] [-c] (-e <elink>
               | (-G <gbt> (-g <group> -p <path>) | (-I <index> -w <width>)) [-i
<dma>]
               [-s <bytes>] [-P <patt>] [-f <speed>] [-R] [-t <secs>] [-u] [-x <size>]
[-X] [<filename>]
  -h         : Show this help text.
  -V         : Show version.
  -b <size>  : DMA (cmem_rcc) memory buffer size to use, in MB (default 128, max
4096).
  -B         : Contents of <filename> is read as binary data (default: ASCII).
  -c         : Do not check whether E-link is configured on FLX-device.
  -d <devnr> : FLX-device to use (default: 0).
  -D         : Debug mode on, i.e. display blocks being uploaded.
  -f <speed> : Speed up default upload rate of about 8MB/s by factor <speed> (default:
1)
  -i <dma>   : FLX-device DMA controller to use (default: auto).
  -P <patt>  : Test data pattern: 0=incr, 1=0x55/0xAA, 2=0xFF, 3=incr-per-chunk
(default: 0).
  -r <repeat>: Test data repeat count: upload <repeat>*<bytes> bytes of data (default:
30).
  -R         : Upload data unformatted, not as CR from-host data packets with header.
  -s <bytes> : Number of bytes per chunk to upload (default: 32).
  -t <secs>  : Number of seconds for DMA time-out or wait until DMA done when 0
(default: 0).
  -u         : Do not perform the actual upload operation.
  -x <size>  : Size of single-shot DMA transfers, in KByte (default: 1).
  -X         : Use continuous-mode DMA for upload (default: single-shot).

Options to define the E-link to use:
  -e <elink> : E-link number (hex) or use -G/g/p or -G/I/w options.
  -E <elink> : an optional 2nd E-link number to upload to
               (alternating with the first given E-link number).
  -G <gbt>   : GBT-link number.
  -g <group> : Group number.
  -p <path>  : E-path number.
  -I <index> : Index of first bit of E-link in GBT frame.
  -w <width> : E-link width in bits (2, 4, 8 or 16).

  <filename> : Name of file with data to upload (ASCII or binary),
               or test pattern data if no name is given.
```

6.4. FELIX Configuration Tools

6.4.1. felink

The felink tool is a link descriptor interpreter which allows you to work out the E-link ID for a given link given GBT/E-group/E-path (or vice versa). This is intended to be used in conjunction with e.g.

fupload to allow users to work out which link ID they should target with their data. Some examples of possible uses will be given below, but you can find all possible options in the help text below:

Help text of felink:

```
felink version 19101400
Convert a given E-link number into GBT, egroup and epath numbers
as well as GBT and bit-index and width, or the other way around.
The E-link number is provided as a (hex) number directly (-e option),
as a set of -G/g/p options, or as a set of -G/I/w options.
Optionally checks if this E-link is valid and configured on a given FLX-device (option
-d),
in either to- or from-host direction.
Use option -l to display a list of valid E-link numbers,
optionally in combination with -G or -g options to restrict the list
to a particular GBT-link and/or egroup.
(Note that E-link numbers are also indicated in the elinkconfig GUI).

Usage: felink [-h|V] [-d <devnr>] (-e <elink>
               | (-G <gbt> (-g <group> -p <path>) | (-I <index> -w <width>))
  -h         : Show this help text.
  -V         : Show version.
  -d <devnr> : FLX-device to use (default: 0).
  -e <elink> : E-link number (hex) or use -G/g/p or -G/I/w options.
  -G <gbt>   : GBT-link number.
  -g <group> : Group number.
  -l         : Show a list of valid E-link numbers (use options -G, -g, -p to restrict
the list).
  -p <path>  : E-path number.
  -I <index> : Index of first bit of e-link in GBT frame.
  -w <width> : E-link width in bits (2, 4, 8 or 16).
```

A list of all valid E-links and coordinates can be seen with list mode, available with the following syntax:

```
felink -l
```

Finding E-link ID from GBT/E-group/E-path of GBT/Bit address/width

Consider the example where a user wishes to know the E-link ID for a link connected to GBT link 2, within E-group 3 and E-path 4. This can be done as follows:

```
felink -G <GBT ID> -g <egroup ID> -p <epath ID>
```

Filling these in gives results as shown below, from which we can see that the E-link ID is 0x9C. The results also show alternative coordinates for the E-link in terms of GBT bit address and width.

```
$ felink -G 2 -g 3 -p 4
E-link 09C = GBT #2 group #3 path #4, bit#56 width=2|4
```

It is also possible to search for link ID using the GBT ID, bit address of the start of the E-link in the GBT frame and E-link width. The syntax is as follows, noting that the index must correspond to a valid E-link start point.

```
felink -G <GBT ID> -I <bit address> -w <E-link width>
```

If a user then wants to search for GBT 1, bit 4 and width 2 the results will be as shown below. This identifies the E-link in question as 0x42.

```
$ felink -G 1 -I 4 -w 2
E-link 042 = GBT #1 group #0 path #2, bit#4 width=2|4
```

These calculations can also be done in reverse, to yield the coordinates of a given known E-link ID. For this use the following syntax:

```
felink -e <E-link ID in hex>
```

If as user then wants to know the coordinates of e.g. E-link 0x55 the tool can be used to give the results as shown below. From this it can be seen that the GBT ID is 1, the E-group ID 2 and the E-path ID 5. An estimate for the bit address and width is also displayed.

```
$ felink -e 55
E-link 055 = GBT #1 group #2 path #5, bit#42 width=2 OR bit#40 width=8
```

6.4.2. fereverse

The fereverse tool makes it possible to swap the bit ordering of data transferred through a designated E-link (or set of links: all in one Egroup, or all in a GBT link), including for EC and IC links separately. For more details consult the help text below:

Help text of fereverse:

```
fereverse version 21011300
Enable, disable or display the bit-order reversal feature for e-links,
a setting per e-path (e-link).
Without keyword '(re)set' the current setting is displayed.

Usage: fereverse [-h|V] [-d <devnr>] [-e <elink>]
                 [-G <gbt> [-g <group>] [-p <path>]] [-E] [-I] [-f] [-t] [set|reset]
  -h         : Show this help text.
  -V         : Show version.
  -d <devnr> : FLX-device to use (default: 0).
  -e <elink> : E-link number (hex) or use -G/g/p options.
  -G <gbt>   : GBT-link number (default: all links).
  -g <group> : Group number (default: all groups).
  -p <path>  : E-path number (default: all paths).
  -E         : Display or enable/disable the EC channel 'bit swap'.
  -I         : Display or enable/disable the IC channel 'bit swap'.
(for options -E/-I use option -G, not -e; options -g/-p are ignored)
  -f         : Configure FromHost (FH) only.
  -t         : Configure ToHost (TH) only.
  set        : Enable e-link bit-reversal.
  reset      : Disable e-link bit-reversal.
```

In order to use the tool to toggle the bits for a given E-link use the following syntax:

`fereverse -d <FELIX ID> -G <GBT ID> -g <E-group ID> -p 1 <set/reset>)`

In this case the *set* option indicates the bits should be switched and *reset* indicates deactivation of the switch. It is also possible to pass the E-link ID directly using option -e. If neither set or reset are specified the tool will simply report back the current status. Examples of both cases are shown below.

```
$ fereverse -d 0 -G 1 -g 1 -p 1 set
GBT 1 egroup 1 epath 1 TH: ENABLED
GBT 1 egroup 1 epath 1 FH: ENABLED
```

```
$ fereverse -d 0 -e 49 reset
GBT 1 egroup 1 epath 1 TH: disabled
GBT 1 egroup 1 epath 1 FH: disabled
```

6.4.3. fgpolarity

The fgpolarity tool makes it possible for FELIX to adapt to the bit polarity of data produced by front-end systems and sent via a Versatile Link[VersatileLinkWebsite] transceiver. The transceiver, by design, swaps the polarity of incoming and outgoing bits (i.e. 0 becomes 1 and vice-versa). Some front-end systems may already account for the swap in their design, but in order to send and receive packets to and from those who haven't this tool configures FELIX to automatically swap the

bits for any designated GBT links (and therefore all E-links within). For more details consult the help text below:

Help text of fgpolarity:

```
fgpolarity version 17060900
Configure or display the GBT transceivers RX and TX polarity.
Usage: fgpolarity [-h|V] [-d <devnr>] [-G <gbt>] [set|reset]
  -h         : Show this help text.
  -V         : Show version.
  -d <devnr> : FLX-device to use (default: 0).
  -G <gbt>   : GBT-link number (default: all).
  -r         : Configure RX only.
  -t         : Configure TX only.
  set        : Set reverse polarity for given GBT transceiver(s).
  reset      : Set default polarity for given GBT transceiver(s).
 (without keyword '(re)set' the current setting is displayed)
```

In order to use the tool to toggle the polarity of a particular GBT link use the following syntax:

`fgpolarity -d <FELIX ID> -G <GBT ID> <set/reset>)`

In this case the *set* option indicates the activation of a polarity switch and *reset* indicates deactivation of the switch. It is also possible to modify the Tx and Rx directions separately using options -t and -r accordingly. By default both will be changed. The expected output from the tool is shown below.

```
$ fgpolarity -d 0 -G 1 set
GBT 1 RX polarity: 1
GBT 1 TX polarity: 1

$ fgpolarity -d 0 -G 1 reset
GBT 1 RX polarity: 0
GBT 1 TX polarity: 0
```

6.4.4. feconf

The feconf tool is a command line tool providing some of the functionality of **elinkconfig**. With this tool it is possible to upload a pre-defined E-link configuration file (.elc format) to a FELIX card. Alongside the basic configuration, feconf also makes it possible to configure the FELIX firmware data generators. For more information on the meaning of each parameter, consult Section Section 6.1 on **elinkconfig**. Feconf sets the so-called fan-out registers for front-end input (so *not* for emulator input). For a full list of commands consult the help text below:

Help text of feconf:

```
feconf version 20121700
Upload an e-link configuration from file (generated by elinkconfig) to the given FLX-
device,
including generation and upload of emulator data contents.
Usage: feconf [-h|V] [-d <devnr>] [-s <chunksz>] [-w] [-R] [-S] [-I <idles>]
<filename>
  -h          : Show this help text.
  -V          : Show version.
  -d <devnr>  : FLX-device to use (default: 0).
  -F          : Do not set FLX-card fan-out registers (default: set for DAQ, not
emulator).
  -L          : 8b10b-words LSB first (GBT only; default: MSB first).
  -n          : Don't write the configuration, just read it in and display some info.
  -R          : Generate emulator data chunks with pseudo-random size.
  -S          : Generate emulator data with a StreamID (first byte).
  -s <chunksz>: Emulator data chunksize to generate (default: 32).
  -w          : Generate emulator data chunksize dependent on e-link width (default:
false).
  -I <idles>  : The number of idles between generated emulator data chunks (default:
8).
 <filename>   : Name of .elc or .jelc file with FLX-device E-link configuration.
```

6.4.5. femu

The femu tool gives users command line control over the FELIX firmware data generators, both in the from and to host directions. The full range of features of the tool can be seen in the help text below:

Help text of femu:

```
femu version 20042100
Show or configure 'FanOut-Select' registers and start/stop emulator.
Usage: femu [-h|V] [-d <devnr>] [-e|E|n] [-l]
  -h         : Show this help text.
  -V         : Show version.
  -d <devnr> : FLX-device to use (default: 0).
  -e|E|n     : Enable FLX-device data emulator, internal (e) or external (E) or
disable emulator (n).
               When no option is given the current status is displayed.
  -f         : When disabling emulator set TOHOST_FANOUT to emulator (default: to
external).
  -l         : 'Unlock' FanOut-Select registers.
  -L         : 'Lock' FanOut-Select registers.
```

6.4.6. fttcemu

The FELIX TTC emulator can be programmed through the fttcemu tool available in the FELIX

software. The status of the FELIX TTC emulator is shown running the command ./fttcemu, which displays the values of the various FELIX 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
```

See for available options the help text below.

Help text of fttcemu:

```
fttcemu version 20062200
Show or configure TTC emulator registers and enable or disable the TTC emulator.

Usage: fttcemu [-h|V] [-c <devnr>] [-e|n] [-B <bc>] [-E <period>]
               [-f <freq>] [-L <cnt>] [-t <us>] [-R]
  -h         : Show this help text.
  -V         : Show version.
  -c <cardnr>: FLX-card selected (default: 0).
  -e|n       : Enable (-e) or disable (-n) the TTC emulator on the selected card.
  -B <bc>    : Set the BCR period, in units of BC (Bunch Count);
               for <bc> equal to 0 a single BCR is generated.
  -E <period>: Set the ECR period, in ms;
               for <period> equal to 0 a single ECR is generated.
  -f <freq>  : Set the TTC emulator L1A freqency, in Hz.
               (any individually generated L1As (option -L) done first)
  -L <cnt>   : Generate <cnt> L1A triggers, using the interval set by -t (default: 0).
               (any single BCR or ECR is generated first).
  -t <us>    : The interval (in microseconds, default: 0)
               between individually generated L1As (option -L).
  -R         : Reset the TTC emulator; also reset the TTC decoder.
Note: options -B, -E, -f and -L also enable the TTC emulator, if necessary.
When no option is given the current status (register contents) is displayed.
```

The TTC emulator can be enabled and disabled on the fly. 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').

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).

TTC_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 (e.g. in the ATLAS experiment the ECR period is set to 5 seconds, so that would be achieved by using option '-E 5000')

Set an L1A frequence 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
```

6.4.7. fttcbusy

The fttcbusy tool gives an overview of various FELIX firmware BUSY settings, such as the E-link TTC-BUSY status and enables, as well as the BUSY settings with respect to the main output FIFO and DMA operation and the status and settings of the board's BUSY output. Here's an example of fttcbusy output:

```
> fttcbusy -T
TTC-BUSY timing: Prescale = 15 Width = 15 Limit-time = 15
E-link TTC-BUSY status (latched BUSY requests and enables for BUSY output):
GBT #00: TTC-BUSY=000000000000000  BUSY-ENA=000000000000000
GBT #01: TTC-BUSY=00000000000A020  BUSY-ENA=000000000000000
  E = 045 = 1-0-5 (not enabled)
  E = 04D = 1-1-5 (not enabled)
  E = 04F = 1-1-7 (not enabled)
GBT #02: TTC-BUSY=000000000000000  BUSY-ENA=000000000000000
GBT #03: TTC-BUSY=000000000000000  BUSY-ENA=000000000000000
GBT #04: TTC-BUSY=000000000000000  BUSY-ENA=000000000000000
GBT #05: TTC-BUSY=000000000000000  BUSY-ENA=000000000000000
GBT #06: TTC-BUSY=000000000000000  BUSY-ENA=000000000000000
GBT #07: TTC-BUSY=000000000000000  BUSY-ENA=000000000000000
GBT #08: TTC-BUSY=000000000000000  BUSY-ENA=000000000000000
GBT #09: TTC-BUSY=000000000000000  BUSY-ENA=000000000000000
GBT #10: TTC-BUSY=000000000000000  BUSY-ENA=000000000000000
GBT #11: TTC-BUSY=000000000000000  BUSY-ENA=000000000000000
BUSY-by-DMA  : enabled=0
               ToHost=0 (BAR0:0) (latched=0)
               buffer free space: assert=200MiB (C800000) deassert=220MiB (DC00000)
BUSY-by-FIFO : enabled=0
               thresh: deassert=3FF assert=4FF
               status: low_crossed=0 high_crossed=0 (latched=0)
BUSY FullMode: busy=000000 (latched=000000)
TTC Bch/TType: 1
BUSY output  : status=0, inhibit=0, master=0
```

Settings for the TTC-BUSY signal timing and BUSY-out signal can be configured. See for available options the help text below.

Help text of fttcbusy:

```
fttcbusy version 20101300
Displays BUSY-related settings and optionally E-link TTC BUSY status and enables,
optionally clearing (latched) E-link BUSY bits.
With option -T the 'TTC BUSY accepted' register contents are displayed, as well as
the corresponding E-link numbers, while option -C clears these registers after being
displayed.
Also the tool may be used to configure the TTC-BUSY signal settings (limit, prescale,
width)
and BUSY output settings (master, inhibit, B-channel, DMA treshold).
Option -R resets the TTC decoder.

Usage: fttcbusy [-h|V] [-d <devnr>] [-G <gbt>] [-C] [-R] [-T]
                [-l <limit>] [-p <prescale>] [-w <width>]
                [-m <b>] [-i <b>] [-b <b>] [-B <b>] [-X <thresh>] [-x <diff>]
  -h           : Show this help text.
  -V           : Show version.
  -d <devnr>   : FLX-device number (default: 0).
  -G <gbt>     : GBT-link number (default: all).
  -C           : Clear (latched) TTC-BUSY register bits.
  -R           : Reset TTC decoder.
  -T           : Display per-(E)link TTC-BUSY info.
  -l <limit>   : Set TTC BUSY limit time parameter (16-bit).
  -p <prescale>: Set TTC BUSY prescale parameter (20-bit).
  -w <width>   : Set TTC BUSY width parameter (16-bit).
  -m <b>       : Set (1) or clear (0) Master BUSY.
  -i <b>       : Set (1) or clear (0) BUSY Inhibit (= BUSY off).
  -b <b>       : Enable(1) or disable(0) BUSY-by-DMA
                 (does not apply to BUSY-by-FIFO for the time being).
  -B <b>       : Enable(1) or disable(0) TTC B-channel/TriggerType.
                 (NB: limits TTCtoHost rate to ca. 200KHz max)
  -X <thresh>  : Set BUSY-by-DMA assert threshold (in MiB).
  -x <diff>    : Set BUSY-by-DMA difference assert/deassert thresholds (in MiB).
```

6.4.8. feto

The feto tool gives users command line control over the FELIX block timeout at the level down to individual E-links. If enabled FELIX will time out incoming data blocks taking longer than a designated period to arrive and attach a timeout trailer to the block. The block will then be transferred to the host as normal. The full range of features of the tool can be seen in the help text below.

Help text of feto:

```
feto version 17121300
Enable, disable or display the instant time-out setting,
a setting per e-path (e-link), or the so-called global time-out
and associated time-out counter (number of clocks until time-out),
or the TTC time-out and associated counter.
Without keyword '(re)set' the current setting of the requested
(group of) time-outs is displayed.

Usage: feto [-h|V] [-d <devnr>] [-e <elink>] [-G <gbt> [-g <group>] [-p <path>]]
            [-T] [set|reset] [<globcntr>]
  -h         : Show this help text.
  -V         : Show version.
  -d <devnr> : FLX-device to use (default: 0).
  -e <elink> : E-link number (hex) or use -G/g/p options.
  -G <gbt>   : GBT-link number.
  -g <group> : Group number (default: all groups).
  -p <path>  : E-path number (default: all paths).
  -T         : Read or configure TTC time-out.
  set        : Enable time-out.
  reset      : Disable time-out.
  <globcntr> : Global or TTC time-out counter value to set.
```

6.4.9. fflash

The fflash tool is designed specifically for loading a selected previously programmed firmware image from FLASH memory aboard a BNL-711 or BNL-712 into the card's FPGA, if the firmware image loaded at power-up is not the required one, or because one wants to switch between versions e.g. for test purposes. At power-up of the FELIX card the firmware image as selected by the setting of onboard switches will become the operational firmware version. Note that after loading a different firmware either a host machine reboot or PCIe hotplug operation is required (see Section 4.2.6 for details) to return the board to normal operation with the new firmware image operational. A full listing of commands and additional help text is shown below.

Help text of fflash:

```
fflash version 21020800
Tool for loading a firmware image from one of the partitions
of the onboard flash memory of an FLX-712 into the card's FPGA,
issueing commands to the host system I2C bus to achieve this.
A subsequent hotplug procedure or machine reboot is required.

Usage: fflash [-h|V] [-q] -f <flashnr>
              [[-L|I] [-U|P -d <devslot>] [-S] [-b <busnr>] [-r <chan>]
              [-s <saddr>] [-u <uaddr>] [-T <sec>]]
  -h          : Show this help text.
  -V          : Show version.
  -q          : Be quiet (only errors will be displayed).
  -f <flashnr>: Flash memory segment partition [0..3] selection (no default).
```

```
  -I          : Generate an INIT_B pulse on the FLX-card (to reset flash devices).
  -L          : Load firmware from the given flash partition into the card.

The following options are relevant in conjunction with the -L and/or -I option:
  -b <busnr>  : I2C-bus number (default=0).
  -r <chan>   : Riser card I2C-switch channel number (default=0).
                NB: I2C-switch has hard-coded I2C address 0x70.
  -s <addr>   : I2C-switch I2C-address (hex, default=0x77, expected range: 0x70-0x77).
                NB: 0x70 already taken by the riser card I2C-switch!
  -u <addr>   : uC I2C-address (hex, default=0x67, expected range: 0x60-0x67).
  -U          : Use USB I2C-dongle instead of system SMBus
                (requires scripts i2cset.py and i2cget.py installed in /opt/flx).
  -P          : Use 'ipmitool' to access system SMBus.
                !NB: use -d option to select 'device slot': 1 or 2.
  -T <sec>    : Set 'Prog-done' timeout [s] (default: 7)
  -d <devslot>: Device slot (1 or 2), only in combination with -P.
  -S          : Preceed calls to i2cget/set or ipmitool with 'sudo'.
                (default: 'sudo' not used; applies to options -L|I|P|U).
Examples:
---------
Load flash memory image partition #2 into the card:
  fflash -f 2 -L

Load flash memory image partition #2 into the card, using I2C-bus #1,
riser card I2C-switch channel #0, FLX-card I2C-switch address 0x75 and
FLX-card microcontroller I2C-address 0x65:
  fflash -f 2 -L -b 1 -r 0 -s 75 -u 65

How to determine the I2C-switch and uC I2C addresses
(options -s and -u respectively) :
Note 1: there is an I2C-bus number (option -b) to select as well,
  which is assumed to have the value '1' (following '-y') in the examples below.
Note 2: in the standard FELIX server there is an additional I2C-switch
  on the socalled riser card; its channel is selected using option -r;
  it means that the 2 FLX-cards in such a server may have identical
  '-s' and '-u' addresses, i.e. most likely their defaults
  while the riser card setting is: 'top' position = -r 0, 'bottom' = -r 1.
'sudo i2cdetect -y 1' should show you an address in the range 0x70-0x77,
let's say 0x77; this is then the address to use in option -s;
subsequently run 'sudo i2cset -y 1 0x77 1' to set the I2C-switch
causing an additional address in the range 0x60-0x67 to appear
in the output of 'sudo i2cdetect -y 1', so run that command again;
this is the address to use in option -u.
On the FLX-712 dipswitch J14 configures the '-s' and '-u' addresses:
  switch 1-3 to set 3 LSBs of '-s', i.e. 0x70-0x77
  switch 4-6 to set 3 LSBs of '-u', i.e. 0x60-0x67
```

6.4.10. fflashprog

The fflashprog tool is designed specifically for programming FLASH memory aboard a BNL-711 or

BNL-712 from an .mcs file containing a firmware image in Intel-HEX format. On the BNL-712 card up to 4 firmware images may be stored. One of the images, selected by onboard switches, will be loaded into the card at power-up and become the operational firmware; if another of the stored images is required, the `fflash` tool should be used to accomplish that.

In addition the fflashprog tool is used to verify an image against an .mcs file or if necessary, to erase a firmware image from memory.

A full listing of commands and additional help text can be seen in the help text below.

Help text of fflashprog:

```
fflashprog version 20040900
Tool for programming, verifying, erasing or dumping firmware images,
stored in a FLX-711/712 card's flash memory.
(to load a selected firmware image into the FLX-card's FPGA use fflash)
Usage: fflashprog [-h|V] [-q] [-c <cardnr>] -f <flashnr> [-D] [-E] [-F]
                  [<filename>] [prog]
  -h          : Show this help text.
  -V          : Show version.
  -q          : Be quiet (only errors will be displayed).
  -c <cardnr> : FLX-card selected (default: 0).
  -d <devnr>  : FLX-device to use (default: 0) OBSOLETE: use -c.
  -D          : Read and display contents of the selected flash partition or flash
file.
  -E          : Erase the selected flash partition.
  -f <flashnr>: Flash memory segment partition [0..3] to dump, to erase,
                to verify or to program (no default).
  -F          : Use the (slow) word-by-word instead of (fast) page programming method.

  <filename>  : Name of MCS file to dump, verify or program.
  prog        : Literal text string to initiate flash programming
                (or else flash verification will take place).
Examples:
---------
Read and dump to screen flash memory image partition #2:
  fflashprog -f 2 -D
Erase flash memory partition #2:
  fflashprog -f 2 -E
Verify flash memory partition #2 against mcs file <filename>:
  fflashprog -f 2 <filename>
Program flash memory partition #2 with the contents of mcs file <filename>:
  fflashprog -f 2 <filename> prog
Read flash ID only:
  fflashprog -f 0
Extra:
Read and dump to screen the memory image in mcs file <filename>:
  fflashprog -D <filename>
```

6.5. General Debugging Tools

6.5.1. fcheck

fcheck is a debugging tool which can analyse the .dat files produced by the fdaq tool and check for data integrity issues. The tool will perform checks on a file to a specified degree of severity. As well as running checks, the tool can also be used to dump selected data blocks to screen, either split into data chunks or as raw data, to facilitate closer inspection of any issues found. To run the check, specify the file name and check detail level as follows:

```
fcheck -B <severity> testfile.dat
```

A full list of features available with fcheck can be seen in the help text below.

Help text of fcheck:

```
fcheck version 20121600
Usage: fcheck [-h|V] [-A] [-B <id>] [-c|C|D] [-e <elink>] [-F <blocks>] [-S <blocks>]
              [-t|T] [-w] [-0|O] [-2|4|8] <filename>
  -h         : Show this help text.
  -V         : Show version.
  -A         : Interpret chunks that could be GBT-SCA frames.
  -B <lvl>   : Do a check on (emulator) data blocks according to <lvl>,
               and display a data summary (default: 2):
               0: Check for proper block headers at 1k boundaries,
                  for each block 1 line of output is produced.
               1: Same as 0, but only when an error is found a line is output.
               2: Full integrity checking of blocks, starting from
                  the block trailer going through all chunks.
               3: Same as 2, including a check on expected emulator data payload,
                  which must constitute an incrementing byte.
               4: Same as 3, but inconsistent maximum values of L1ID are not reported.
  -c         : Display data 'raw' datablocks (default: chunk data) (with option -F)
  -C         : Display chunk data bytes only, nothing else.
  -D         : Display only whole data chunks, i.e. the user's data frames.
  -e <elink> : E-link number (hex) to filter for block check or block display.
  -F <blocks>: Dump <blocks> 1K data blocks to display (overrules data check option
-B).
               Chunk types: BOTH="<<", FIRST="++", LAST="&&", MIDDLE="==",
TIMEOUT="]]", NULL="@@",
                OUTOFBAND="##" and "TE" for chunk truncation/error.
  -S <blocks>: Skip <blocks> of data blocks before starting check or display.
  -t         : Do NOT report chunk truncation/error/CRCerror.
  -T         : Do NOT report chunk CRCerror.
  -w         : Instead of displaying, write (binary) chunkdata to file (dataout.dat).
  -0         : Do NOT display time-out chunkdata bytes (zeroes).
  -O         : Do not display time-out chunks at all.
  -2         : Display data as 2-byte words (little-endian).
  -4         : Display data as 4-byte words (little-endian).
  -8         : Display data as 8-byte words (little-endian).
 <filename>  : Name of file containing data to check or display.
```

Run full integrity check of all data blocks in file 'file.dat', reporting data chunk errors as well data block corruption, including block number, E-link number and block word index:

```
fcheck -B 2 file.dat
```

Display the data chunks from 2 data blocks starting from block number 1000 in file 'file.dat':

```
fcheck -S 1000 -F 2 file.dat
```

Display the raw data from 2 data blocks originating from E-link 8 starting from block number 1000:

```
fcheck -S 1000 -F 2 -e 8 -c file.dat
```

Display the data from 2 data blocks originating from E-link 8 as 4-byte items starting from block

number 1000, and not displaying the zeroes of time-out chunks, just their sizes:

```
fcheck -S 1000 -F 2 -e 8 -4 -0 file.dat
```

Display the data chunks from 2 data blocks and interpret the chunks that look like they are replies from a GBT-SCA device, including control byte, transaction ID, channel number, length, error byte and data:

```
fcheck -F 2 -A -0 file.dat
```

6.5.2. fedump

The fedump tool is designed to make it possible to dump data arriving at FELIX to the screen for debugging purposes. Users of the tool can filter the data stream by E-link ID and FELIX card number, as well as having the option of displaying the data in raw format. More advanced options are available, but should only be used in consultation with the FELIX developers. For more details consult the help text below.

Help text of fedump:

```
fedump version 20061700
Dump selected E-link chunk data (optionally block-by-block)
received from an FLX-device to screen.
Chunk types are delimited by:
BOTH="<<", FIRST="++", LAST="&&", MIDDLE="==", TIMEOUT="]]", NULL="@@", OUTOFBAND="##"
Usage:
 fedump [-h|V] [-A] [-c|D] [-d <devnr>] [-e <elink>] [-i <dma>] [-I] [-0|O] [-2|4|8]
  -h         : Show this help text.
  -V         : Show version.
  -A         : Interpret chunks that could be GBT-SCA frames.
  -c         : Display data 'raw', block-by-block (default: chunk data).
  -d <devnr> : FLX-device to use (default: 0).
  -D         : Display only whole data chunks, i.e. the user's data frames.
  -e <elink> : E-link number (hex) to filter out for display (default: no filter).
  -i <dma>   : FLX-device DMA controller to use (default: 0).
  -I         : Use interrupt to receive data (default: polling).
  -0         : Do not display time-out chunkdata (zeroes).
  -O         : Do not display time-out chunks at all.
  -2         : Display data as 2-byte words (little-endian).
  -4         : Display data as 4-byte words (little-endian).
  -8         : Display data as 8-byte words (little-endian).
```

6.6. Remote Hardware Command and Configuration Tools

6.6.1. fice

The fice tool is designed for communication on the IC channel of a GBT link, to read and write registers on a GBTX chip present on a remote system connected to this link, in order to configure

the GBTX as required. Individual registers may be read or written or a file containing a range of register settings can be read in by the tool and written as one message to the GBTX.

For a full list of commands consult the help text below.

Help text of fice:

```
fice version 18102400
Tool to read or write GBTX registers via the IC-channel of an FLX-device GBT link
(using the dedicated FLX-device virtual E-link):
read or write a single byte from or to the given GBTX register address
or write to multiple consecutive GBTX registers using the contents of a file.
(i.e. ASCII file: 1 (register) byte value (hex) per line,
 e.g. the 'TXT' file generated by the GBTXProgrammer tool).
Provide a file name *or* use option -a with an optional additional byte value
to read resp. write a single GBTX register or, without option -a, to read all
registers.
Without option -a and file name all registers are read out and displayed
either in one IC read operation or one-by-one (option -o).
Option -t displays the register values in a format that could be used
as a 'TXT' file for this tool or the I2C-dongle GBTX programmer.
Usage:
 fice [-h|V] [-d <devnr>] [-r] [-G <gbt>] [-I <i2c>] [-t] [-a <addr> [<byte>] |
<filename>]
  -h         : Show this help text.
  -V         : Show version.
  -a <addr>  : GBTX register address (decimal or hex (0x.. or x..)) to read or write.
  -d <devnr> : FLX-device to use (default: 0).
  -G <gbt>   : GBT-link number.
  -I <i2c>   : GBTX I2C address.
  -o         : When reading all registers, do it one-by-one (default: one multi-reg
read op).
  -r         : Do NOT receive and process/display replies.
  -R         : Receive replies on any E-link.
  -t         : Display one register value per line in output (i.e. 'TXT'-format like).
 <byte>      : Byte value (hex) to write to GBTX register <addr> (option -a).
 <filename>  : Name of file with GBTX register data to write to consecutive registers.
=> Examples:
Read all registers of GBTX (I2C address 1) connected to FLX-device GBT link 3:
  fice -G 3 -I 1
Read GBTX register 32 (0x20):
  fice -G 3 -I 1 -a 32 (or: fice -G 1 -I 3 -a 0x20)
Write 0xA5 to GBTX register 32 (0x20):
  fice -G 3 -I 1 -a 32 A5
Write contents of GBTX-conf.txt to GBTX registers:
  fice -G 3 -I 1 GBTX-conf.txt
```

Example of reading all registers of a GBTX device connected to the GBT link #3 IC-channel, with I2C address 1 (each line starts with a register address, followed by 16 register byte values):

```
$ fice -G 3 -I 1
Opened FLX-device 0, firmw FLX712-GBT-4chan-2007301614-GIT:rm-4.10/538
>>> GBTX#3 I2C-addr=1: read all registers
Reply: Parity OK
  0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 16:  00 00 00 00 00 00 00 00 00 02 00 28 00 15 15 15
 32:  66 00 0d 42 00 0f 04 08 00 20 00 00 00 00 15 15
 48:  15 00 07 00 38 00 00 00 00 00 00 00 00 00 00 15
 64:  dd 0d 00 00 00 00 00 00 00 00 00 00 00 00 ff ff
 80:  ff ff ff ff 00 00 00 15 dd 0d 00 00 00 00 00 00
 96:  00 00 00 00 00 00 ff ff ff ff ff ff 00 00 00 15
112:  dd 0d 00 00 00 00 00 00 00 00 00 00 00 00 ff ff
128:  ff ff ff ff 00 00 00 15 dd 0d 00 00 00 00 00 00
144:  00 00 00 00 00 00 ff ff ff ff ff ff 00 00 00 15
160:  dd 0d 00 00 00 00 00 00 00 00 00 00 00 00 ff ff
176:  ff ff ff ff 00 00 00 00 00 70 00 00 00 00 00 00
192:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
208:  00 70 00 00 00 00 00 00 00 00 00 00 00 00 00 00
224:  00 00 00 00 00 00 00 00 00 70 00 00 00 00 00 00
240:  00 00 3f 3f 38 00 00 00 07 00 00 07 00 00 71 ff
256:  ff 01 ff ff 01 ff ff 01 ff ff 01 ff ff 00 00 00
272:  00 20 00 00 00 00 00 00 00 15 00 00 00 00 00 00
288:  00 00 00 00 00 ff ff ff 40 40 40 2a 2a 2a 00 00
304:  ff ff ff 40 40 40 2a 2a 2a 4e 4e 4e aa 0a 07 00
320:  ff ff ff ff ff 00 00 88 88 88 88 88 01 ff ff 01
336:  ff ff 01 ff ff 01 ff ff 01 ff ff 01 ff ff 01 ff
352:  ff 01 ff ff 01 ff ff 01 ff ff 00 00 00 aa 00 8e
368:  67 25 2e 80 80 80 a5 00 00 fd cd cd cd 00 00 00
384:  00 00 00 00 aa bb 9f ff ff ff ff ff ff ff 10 00
400:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
416:  00 00 00 00 00 00 00 00 00 00 00 0f 78 ff 00 61
432:  7d df f5 99
(chunks received: 1)
```

Write 0x34 to register 2 of that same GBTX device:

```
$ fice -G 3 -I 1 -a 2 34
Opened FLX-device 0, firmw FLX712-GBT-4chan-2007301614-GIT:rm-4.10/538
>>> GBTX#3 I2C-addr=1: write 0x34 to reg 0x2
Write 1 bytes: 34
Reply: Parity OK
  2:  34
(chunks received: 1)
```

Read register 2 of the GBTX device:

```
$ fice -G 3 -I 1 -a 2
Opened FLX-device 0, firmw FLX712-GBT-4chan-2007301614-GIT:rm-4.10/538
>>> GBTX#3 I2C-addr=1: read reg 0x2
Read 1 bytes
Reply: Parity OK
  2:  34
(chunks received: 1)
```

6.6.2. fgbtxconf

The fgbtxconf tool makes it possible to read and write GBTX registers accessible via the chip's I2C port, accessed through a GBT-SCA chip. The functionality of this tool is similar to fice. For a full description please consult the help output below.

Help text of fgbtxconf:

```
fgbtxconf version 17111700
Tool to read or write GBTX registers via an I2C-channel of a GBT-SCA chip,
connected to any FLX-device GBT (2-bit HDLC) E-link:
read or write a single byte from or to the given GBTX register address
or write to multiple consecutive GBTX registers using the contents of a file.
(i.e. ASCII file: 1 (register) byte value (hex) per line,
 e.g. the 'TXT' file generated by the GBTXProgrammer tool).
Provide a file name *or* use option -a with an optional additional byte value
to read resp. write a single GBTX register or, without option -a, to read all
registers.
Usage:
 fgbtxconf [-h|V] [-d <devnr>] [-e <elink>] [-G <gbt> [-g <group> -p <path>]] [-R] [-
r] [-W]
        -C <ichan> -I <iaddr> -a <addr> [<byte>] | <filename>
  -h         : Show this help text.
  -V         : Show version.
  -d <devnr> : FLX-device to use (default: 0).
  -e <elink> : E-link number (hex) or use -G/g/p options.
  -G <gbt>   : GBT-link number.
  -g <group> : Group number (default: 7=EC).
  -p <path>  : E-path number (default: 7=EC).
  -R         : Reset GBT-SCA.
  -r         : Do not receive and display the GBT-SCA replies.
  -W         : Read writable registers only (default: all).
  -C <ichan> : GBT-SCA I2C channel number.
  -I <iaddr> : GBTX I2C address (hex).
  -a <addr>  : GBTX register address (decimal or hex (0x.. or x..)) to read or write.
 <byte>      : Byte value (hex) to write to GBTX register <addr> (option -a).
 <filename>  : Name of file with GBTX register data to write to consecutive registers.
=> Examples:
Read all registers of GBTX (I2C address 1) connected to GBT-SCA I2C-channel 0,
GBT-SCA connected to FLX-device GBT link 3 EC-link:
  fgbtxconf -G 3 -I 1 -C 0 (or: fgbtxconf -e ff -I 1 -C 0)
Read GBTX register 32 (0x20):
  fgbtxconf -G 3 -I 1 -C 0 -a 32 (or: fgbtxconf -G 3 -I 1 -C 0 -a 0x20)
Write 0xA5 to GBTX register 32 (0x20):
  fgbtxconf -G 3 -I 1 -C 0 -a 32 A5
Write contents of GBTX-conf.txt to GBTX registers:
  fgbtxconf -G 3 -I 1 -C 0 GBTX-conf.txt
```

6.7. Tools for GBT-SCA device access

6.7.1. fec

The fec tool is designed for communication with a GBT-SCA chip present on a remote hardware system connected to FELIX via a GBT link. The tool allows to send pre-programmed commands to read out or write to a number of the hardware channels available on a GBT-SCA, such as GPIO, ADC

and DAC. The GBT-SCA can be connected to any 2-bit (HDLC-encoded) E-link of the GBT, besides the EC channel. For a full list of commands consult the help text below.

Help text of fec:

```
fec version 21032300
Demo tool for control and read out of various devices on a GBT-SCA
through a GBT link's EC channel or any 2-bit wide, HDLC encoded E-link.
Receives (and displays) GBT-SCA replies, unless option -Z is given
(in that case use e.g. fedump or fdaq to receive).
Usage:
 fec [-h|V] [-d <devnr>] [-i <dma>] [-I] [-N] [-G <gbt>] [-g <group>] [-p <path>]
     [-t <ms>] [-x <par>] [-A] [-C] [-R] [-T] [-P <secs>] [-X] [-Y <seq>] [-Z] [<ops>]
  -h         : Show this help text.
  -V         : Show version.
  -d <devnr> : FLX-device to use (default: 0).
  -i <dma>   : FLX-device DMA controller for receiving (default: 0).
  -I         : USE interrupt to receive data (default: polling)
  -N         : Receiver resets DMA at start-up (default: no reset).
  -G <gbt>   : GBT-link number (default: 0).
  -g <group> : Group number (default matches GBT EC 'group' = 7).
  -p <path>  : E-path number (default matches GBT EC 'path' = 7).
  -r <repeat>: Number of GPIO/ADC/DAC operations to perform (default: 1).
  -A         : Use SCA-V1 ADC commands (default: SCA-V2 ADC).
  -C         : Send GBT-SCA connect (HDLC control).
  -R         : Send GBT-SCA reset (HDLC control).
  -T         : Send GBT-SCA test (HDLC control).
  -t <ms>    : Time between some of the ops, in ms (default: 100).
  -P <secs>  : Enable FromHost (circular) DMA then pause for <secs> seconds
               (for DMA check/debug; default: no pause)
  -X         : Use continuous-mode DMA for upload (default: single-shot).
  -x <par>   : Parameter to use in operations, e.g. GPIO number, ADC or DAC channel
(default: 0).
  -Y <seq>   : Use <seq> as first HDLC 'receive sequence number'.
               (to keep receiving side happy in consecutive calls)
  -Z         : Do NOT receive and display the GBT-SCA replies.
  <ops>      : String of chars indicating which operation(s) to perform:
               o=GPIO-out, i=GPIO-in, a=ADC, d=DAC, I=I2C (no-string=default: none).
Examples:
Blink an LED on a VLDB (here connected to GBT link #3, EC-channel)
on GBT-SCA GPIO #18 (the other LED is on GPIO #21) 20 times
with a rate of 5Hz (100ms ON, 100ms OFF):
  fec -G 3 -t 100 -r 20 -x 18 o
Read GPIO inputs (GBT-SCA on GBT-link #3's EC-channel) 20 times
with a rate of 10Hz:
  fec -G 3 -t 100 -r 20 i
```

Example: on the VLDB with GBT-SCA connected to the EC-channel of GBT link #3 blink one of the LEDs (connected to GPIO #18; the other one is connected to GPIO #21) 25 times (-r 50) with an on and off period of 100 ms (the last symbol is an 'oh', which stands for 'GPIO output'):

```
fec -G 3 -r 50 -t 100 -x 18 o
```

6.7.2. fscaid

The fscaid tool reads out and displays a GBT-SCA chip's ID register.

Help text of fscaid:

```
fscaid version 18111300
Tool to read a GBT-SCA's Chip ID.
Usage:
 fscaid [-h|V] [-d <devnr>] [-e <elink>] [-G <gbt>] [-g <group>] [-p <path>]
        [-1] [-C] [-R] [-Z]
  -h          : Show this help text.
  -V          : Show version.
  -d <devnr> : FLX-device to use (default: 0).
  -e <elink> : E-link number (hex) or use -G/g/p options.
  -G <gbt>   : GBT-link number (default: 0).
  -g <group> : Group number (default matches GBT EC 'group' = 7).
  -p <path>  : E-path number (default matches GBT EC 'path' = 7).
  -C          : Send GBT-SCA connect (HDLC control).
  -R          : Send GBT-SCA reset (HDLC control).
  -Z          : Do not receive and display the GBT-SCA replies.
  -1          : Read ID from a GBT-SCA Version 1 (default: V2).
```

6.7.3. fscaio

The fscaio tool is used to read and write a GBT-SCA's GPIO lines, either individually or all 32 in one operation. Also the direction register can be configured.

Help text of fscaio:

```
fscaio version 21011800
Tool to write and/or read the GBT-SCA GPIO bits and direction register.
Usage:
 fscaio [-h|V] [-d <devnr>] [-e <elink>] [-G <gbt>] [-g <group>] [-p <path>]
        [-i <bit>] [-o <dir>] [-C] [-R] [-D] [-E] [-Z] [<value>]
  -h         : Show this help text.
  -V         : Show version.
  -d <devnr> : FLX-device to use (default: 0).
  -e <elink> : E-link number (hex) or use -G/g/p options.
  -G <gbt>   : GBT-link number (default: 0).
  -g <group> : Group number (default matches GBT EC 'group' = 7).
  -p <path>  : E-path number (default matches GBT EC 'path' = 7).
  -i <bit>   : Read or write GPIO bit number <bit> (default: all).
               NB: if a single I/O pin is written to, its direction bit
                   is set to output (independent of option -o).
  -o <dir>   : Set GPIO direction register to value <dir> (hex).
  -C         : Send GBT-SCA connect (HDLC control).
  -R         : Send GBT-SCA reset (HDLC control).
  -Z         : Do not receive and display the GBT-SCA replies.
  -D         : Disable GBT-SCA GPIO channel after operation (default: leave enabled)
  -E         : Do *not* enable GBT-SCA GPIO channel at start of operation, assume it
already is.
 <value>     : Value to write (0 or 1 for a single GPIO bit, or up to 0xFFFFFFFF
otherwise, hexadecimal);
               if no value is provided a read operation is performed.
```

6.7.4. fscaadc

The fscaadc tool reads out a GBT-SCA's ADC input channels, displaying raw as well as converted (to volts) values. In addition, ADC input channel current sources can be selectively enabled for the read-out.

Example output for a single ADC input scan for a GBT-SCA connected to the EC channel of GBT link #3 (here: the GBT-SCA on a VLDB):

```
$ fscaadc -C -G 3
Opened FLX-device 0, firmw FLX712-GBT-4chan-2008271931-GIT:rm-4.10/544
GBT-SCA connect
ADC enabled
GBT-SCA ADC readings:
 0: 57B  =  1403  =  0.343 V
 1: 7A7  =  1959  =  0.478 V
 2: 61C  =  1564  =  0.382 V
 3: 6C7  =  1735  =  0.424 V
 4: 64B  =  1611  =  0.393 V
 5: 76E  =  1902  =  0.464 V
 6: 819  =  2073  =  0.506 V
 7: 836  =  2102  =  0.513 V
 8: 791  =  1937  =  0.473 V
 9: 7DB  =  2011  =  0.491 V
10: 71F  =  1823  =  0.445 V
11: 8DB  =  2267  =  0.554 V
12: 857  =  2135  =  0.521 V
13: 7FF  =  2047  =  0.500 V
14: 6BA  =  1722  =  0.421 V
15: 720  =  1824  =  0.445 V
16: 6DD  =  1757  =  0.429 V
17: 5EC  =  1516  =  0.370 V
18: 7E7  =  2023  =  0.494 V
19: 892  =  2194  =  0.536 V
20: 7FF  =  2047  =  0.500 V
21: 859  =  2137  =  0.522 V
22: 8B5  =  2229  =  0.544 V
23: 6CB  =  1739  =  0.425 V
24: 8A2  =  2210  =  0.540 V
25: 699  =  1689  =  0.412 V
26: CE1  =  3297  =  0.805 V
27: 024  =    36  =  0.009 V
28: 77C  =  1916  =  0.468 V
29: 000  =     0  =  0.000 V
30: FFF  =  4095  =  1.000 V
31: A9D  =  2717  =  0.663 V (T=30.9C approx.)
```

Help text of fscaadc:

```
fscaadc version 19031300
Tool to read GBT-SCA ADC input channels and display the readings.
Usage:
 fscaadc [-h|V] [-d <devnr>] [-e <elink>] [-G <gbt>] [-g <group>] [-p <path>] [-c
<msk>] [-A]
         [-i <index>] [-n <kohm>] [-r <cnt>] [-t <us>] [-C] [-D] [-E] [-R] [-X] [-Z]
  -h         : Show this help text.
  -V         : Show version.
  -d <devnr> : FLX-device to use (default: 0).
  -e <elink> : E-link number (hex) or use -G/g/p options.
  -G <gbt>   : GBT-link number (default: 0).
  -g <group> : Group number (default matches GBT EC 'group' = 7).
  -p <path>  : E-path number (default matches GBT EC 'path' = 7).
  -A         : Use SCA-V1 ADC commands (default: SCA-V2 ADC).
  -c <mask>  : Enable current sources on the ADC inputs in bitmask <mask>
               (disabled again afterwards).
  -i <index> : Conversion of ADC input <index> only (default: all 32 inputs
consecutively).
  -n <kohm>  : NTC reference resistance value in KOhm;
               for ADC inputs with current source enabled (option -c)
               a temperature in Celcius is now calculated assuming they have such NTCs
connected.
  -r <cnt>   : Number of times to convert ADC input or inputs (default: 1).
  -t <us>    : Microseconds between ADC conversions (default: 200).
  -C         : Send GBT-SCA connect (HDLC control).
  -R         : Send GBT-SCA reset (HDLC control).
  -D         : Disable GBT-SCA ADC after operation (default: leave enabled)
  -E         : Do *not* enable GBT-SCA ADC at start of operation, assume it already
is.
  -X         : Use continuous-mode DMA for upload (default: single-shot).
  -Z         : Do not receive and display the GBT-SCA replies.
```

6.7.5. fscadac

The fscadac tool sets a GBT-SCA's DAC output channels, one at a time, or all four at the same time. In addition it provides an option to sweep through the DAC values for one or all channels within a configurable time period.

Help text of fscadac:

```
fscadac version 18111300
Tool to set and/or read back GBT-SCA DAC outputs.
In addition it allows a sweep through the DAC range for one or all DAC outputs
within a configurable time period.
Usage:
 fscadac [-h|V] [-d <devnr>] [-e <elink>] [-G <gbt>] [-g <group>] [-p <path>]
         [-i <index>] [-s] [-t <ms>] [-C] [-R] [-D] [-E] [-Z]
  -h         : Show this help text.
  -V         : Show version.
  -d <devnr> : FLX-device to use (default: 0).
  -e <elink> : E-link number (hex) or use -G/g/p options.
  -G <gbt>   : GBT-link number (default: 0).
  -g <group> : Group number (default matches GBT EC 'group' = 7).
  -p <path>  : E-path number (default matches GBT EC 'path' = 7).
  -i <index> : DAC index (0=DAC_A,1=DAC_B,2=DAC_C,3=DAC_D) to use (default: all).
  -s         : Sweep DAC value for the given DAC output(s).
  -t <ms>    : Sweep time from DAC value 0 to 255, in milliseconds,
               when option -s given (default: 1000).
  -C         : Send GBT-SCA connect (HDLC control).
  -R         : Send GBT-SCA reset (HDLC control).
  -Z         : Do not receive and display the GBT-SCA replies.
  -D         : Disable GBT-SCA DAC after operation (default: leave enabled)
  -E         : Do *not* enable GBT-SCA DAC at start of operation, assume it already
is.
```

6.7.6. fscai2c

The fscai2c tool provides low-level access to I2C-devices connected to GBT-SCA I2C channels, with control over register address size (1 or 2 bytes), register content size and 7-bit or 10-bit addressing. See help text below for some more information.

Help text of fscai2c:

```
fscai2c version 21011500
Tool to read or write from an I2C device register
on any I2C port of a GBT-SCA chip connected to any FLX-device E-link
(the latter given by options -G/g/p or option -e)
Usage:
 fscai2c [-h|V] [-d <devnr>] [-e <elink>] [-G <gbt>] [-g <group>] [-p <path>]
         [-R <rate>] -C <ichan> -I <iaddr> [-t] [-a|A <addr>] [-r <nbytes>]
         [-D] [-E] [<value-to-write>]
  -h         : Show this help text.
  -V         : Show version.
  -d <devnr> : FLX-device to use (default: 0).
  -e <elink> : E-link number (hex) or use -G/g/p options.
  -G <gbt>   : GBT-link number (default: 0).
  -g <group> : Group number (default matches GBT EC 'group' = 7).
  -p <path>  : E-path number (default matches GBT EC 'path' = 7).
  -C <ichan> : GBT-SCA I2C channel number.
  -I <iaddr> : I2C device address (decimal or '0x..' for hexadecimal).
  -t         : Use 10-bit I2C addressing mode.
  -a|A <addr>: I2C register address ('a':1-byte, 'A':2-byte).
               (decimal or '0x..' for hexadecimal).
  -r <bytes> : Register content number of bytes (default: 1).
  -R <rate>  : I2C transfer rate (1=100KHz, 2=200KHz, 4=400KHz, 10=1MHz, default: 4).
  -D         : Disable GBT-SCA I2C port after operation (default: leave enabled)
  -E         : Do *not* enable GBT-SCA I2C port at start of operation, assume it
already is.
 <value-to-write>: hexadecimal value to write, the number of nibbles determining
                   how many bytes to write.
=> Examples:
Read 2-byte register 6 from a device with I2C address 5 on GBT-SCA I2C channel 4
connected to the EC channel of GBT #3:
  fscai2c -G 3 -C 4 -I 5 -a 6 -r 2
Write 0x1234 to 2-byte register 6 from I2C device address 5 on GBT-SCA I2C channel 4
connected to the EC channel of GBT #3:
  fscai2c -G 3 -C 4 -I 5 -a 6 1234
```

6.7.7. fscads24

The fscads24 tool reads out the unique 64-bit ID from a device from the 1-Wire DS2400-family, connected to a GPIO pin of a GBT-SCA.

Example output, reading from a DS2411 connected to GPIO pin 3 of a GBT-SCA connected to the EC-channel of FLX-card #0 GBT link #3, including sending an initial 'connect' command:

```
$ fscads24 -C -G 3 -i 3
Opened FLX-device 0, firmw FLX712-GBT-4chan-2008271931-GIT:rm-4.10/544
GBT-SCA connect
ID:
01 63 41 a0 17 00 00 1a
Replies received: 288
```

Help text of fscads24:

```
fscads24 version 19112500
Tool to read out the 64-bit ID from a 1-Wire DS24xx chip.
Usage:
 fscads24 [-h|V] [-d <devnr>] [-e <elink>] [-G <gbt>] [-g <group>] [-p <path>]
          [-r <cnt>] [-C] [-D] [-E] [-R] [-X] [-Z] -i <pin>
  -h         : Show this help text.
  -V         : Show version.
  -d <devnr> : FLX-device to use (default: 0).
  -e <elink> : E-link number (hex) or use -G/g/p options.
  -G <gbt>   : GBT-link number (default: 0).
  -g <group> : Group number (default matches GBT EC 'group' = 7).
  -i <pin>   : Use GPIO bit <pin> for the 1-Wire protocol ([0..31]).
  -p <path>  : E-path number (default matches GBT EC 'path' = 7).
  -r <cnt>   : Number of times to read the ID (default: 1).
  -C         : Send GBT-SCA connect (HDLC control).
  -R         : Send GBT-SCA reset (HDLC control).
  -D         : Disable GBT-SCA GPIO after operation (default: leave enabled)
  -E         : Do *not* enable GBT-GPIO at start of operation, assume it already is.
  -X         : Use continuous-mode DMA for upload (default: single-shot).
  -Z         : Do not receive and display the GBT-SCA replies.
```

6.7.8. fscajtag

The fscajtag tool is used to program a 'bit' file into a Xilinx FPGA connected to the JTAG port of a GBT-SCA.

Without a file name it reads and displays the connected FPGA's ID and status register contents (optionally the status bits are individually listed and named).

Help text of fscajtag:

```
fscajtag version 19061800
Tool to program a bit-file into a Xilinx 7-Series FPGA connected to the JTAG port
of a GBT-SCA, connected to any FLX-device GBT (2-bit HDLC) E-link.
If a bit-file name is not provided the ID-code and Status register
of the FPGA are read out and displayed.
Usage:
 fscajtag [-h|V] [-D] [-d <devnr>] [-e <elink>] [-G <gbt> [-g <group> -p <path>]]
          [-c] [-R <rate>] [-r] [-s] [<filename>]
  -h         : Show this help text.
  -V         : Show version.
  -c         : Use continuous-mode FromHost DMA (default: single-shots).
  -d <devnr> : FLX-device to use (default: 0).
  -D         : Enable debug mode: display all GBT-SCA replies.
  -e <elink> : E-link number (hex) or use -G/g/p options.
  -G <gbt>   : GBT-link number.
  -g <group> : Group number (default: 7=EC).
  -p <path>  : E-path number (default: 7=EC).
  -R <rate>  : JTAG clock rate, in MHz, 1,2,4,5,10 or 20 (default: 20).
  -r         : Do NOT receive and process/display GBT-SCA replies.
  -s         : Display FPGA Configuration register bits.
  -x <devs>  : Number of devices preceeding the FPGA in the JTAG chain.
  -X <ibits> : Total number of preceeding BYPASS instruction bits, or
               (with option -z) the number of instruction bits per device.
  -y <devs>  : Number of devices trailing the FPGA in the JTAG chain.
  -Y <ibits> : Total number of trailing BYPASS instruction bits, or
               (with option -z) the number of instruction bits per device.
  -z <instr> : The BYPASS instruction value for each of the preceeding
               and trailing devices (when unequal to only '1'-bits).
 <filename>  : Name of .bit file containing the FPGA configuration.
```

6.7.9. fxvcserver

The fxvcserver tool connects to a selected FLX-card E-link connected to a GBT-SCA and listens for an XVC protocol connection from a Vivado Hardware Manager in order to connect it to a Xilinx FPGA connected to the JTAG port of the GBT-SCA device, allowing for the usual firmware programming and debugging operations from within Vivado.

(Note that the same tool is available in the ATLAS DCS software for GBT-SCAs, interfacing to the E-link via *felixcore*).

Help text of fxvcserver:

```
fxvcserver v20082600
Relays Xilinx XVC protocol JTAG bit streams to and from the JTAG port of a GBT-SCA,
through its connection to a FELIX system.
Usage:
 fxvcserver [-h|V] [-v] [-d <devnr>] [[-e <elink>] | [-G <gbt> [-g <group> -p <path>]]
            [-P <portnr>] [-R <rate>]
  -h         : Show this help text.
  -V         : Show version.
  -v         : Be verbose (for debugging only).
  -d <devnr> : FLX-device to use (default: 0).
  -e <elink> : E-link number (hex) or use -G/g/p options.
  -G <gbt>   : GBT-link number.
  -g <group> : Group number (default: 7=EC).
  -p <path>  : E-path number (default: 7=EC).
  -P <portnr>: IP port number to listen on (default: 2542).
  -R <rate>  : JTAG clock rate, in MHz, 1,2,4,5,10 or 20 (default: 10).

In Vivado's Hardware Manager in the TCL Console type "connect_hw_server"
(if necessary), followed by "open_hw_target -xvc_url <address>:<portnr>"
to connect to a Xilinx FPGA connected to the GBT-SCA JTAG port,
with <address> and <portnr> the IP address and port number
of the FELIX host running this fxvcserver.
```

Chapter 7. Felixcore Application and NetIO

7.1. Operational Principles

It is generally recommended that you upgrade to use felix-star. It has better memory management, better cpu usage and uses netio-next.

The FELIX core application (called *felixcore*) is the legacy version of the FELIX system's central process (now superceded by felix-star, but retained in the release during the transition to the new architecture). The user interacts with the felixcore application via network endpoints to receive data from E-links or to send data to E-links. Systems such as software RODs, DCS, calibration and monitoring systems all connect with FELIX via felixcore.

Felixcore also supports monitoring of operational data via a web front-end and publishing of E-link information via the FELIX bus system.

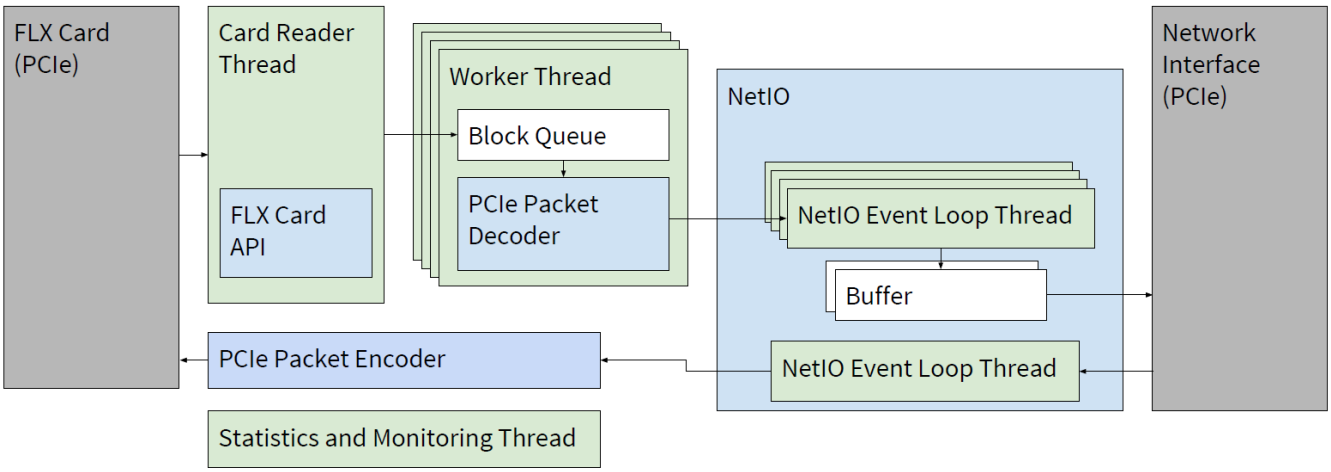

Figure 26. The architecture of the FELIX core application.

Figure 26 shows a diagram of the architecture of the FELIX core application. In the case of a system with multiple FLX cards, one application can be run per card.

7.2. Configuration

The configuration parameters of the felixcore application are listed in Command line interface options for the felixcore application. Each option can also be set in the configuration file. The option key in that case is the long option without the leading dashes.. The parameters can be passed to the application either on the command line or as part of a file.

Command line interface options for the felixcore application. Each option can also be set in the configuration file. The option key in that case is the long option without the leading dashes.

```
felixcore - FELIX Core Application

Usage: felixcore [(--verbose | --quiet) --device N... options]

  Run with card: 'felixcore'
```

```
  Run with file: 'felixcore -f <file> --notoflx --nobusy'
  Run with data generator: 'felixcore -g --notoflx --nobusy'

All Options:

General Options:
  -t, --threads N                       Run with the specified number of threads
[default: 1]
  -p, --port N                          Receive daq data starting from port [default:
12350]
  -r, --recv-port N                     Receive slow control data starting from port
[default: 12340]
  -P, --busy-port N                     Receive busy control signals on this port
[default: 12330]
  -w, --web-port N                      Port for webserver [default: 8080]
  -b, --buffer N                        Free Buffer size in kBlocks per thread
[default: 2000]
  -g, --data-generator                  Use internal data generator
  -l, --logfile <file>                  Write logs to the given filename
  --ttc-generator N                     Use internal ttc generator and elink #,
default is off [default: -1]
  --felix-id <id>                       Elink offset for this FelixCore [default: 0]
  --data-interface <iface>              Interface to publish data [default: eth0]
  --monitoring-interface <iface>        Interface to publish monitoring information
[default: eth0]
  --noweb                               Disable web server

Netio Options:
  -B, --netio-backend (posix|fi-verbs)  Use the given NetIO backend [default: posix]

Commandline Options:
  -h, --help                            Display this help
  -V, --version                         Display version
  -v, --verbose                         Switch logging on, level = debug
  -q, --quiet                           Switch logging off, level = warning
  -c, --config <file>                   Load configuration from the specified YAML
file
  --config-write <file>                 Store configuration in the specified YAML file
  -f, --file <file>                     Use the given filename as input

Card Options:
  -d, --device N...                     Use only listed devices to look for cards
  -m, --memory N                        Allocate CMEM memory in Gbyte [default: 1]
  -M, --megabyte N                      Allocate CMEM memory in Mbyte, overrides -m
[default: 0]
  --polling                             Use polling
  --poll-time T                         Poll time in micro-seconds [default: 50000]
  --dma D                               Use toHost dma channel [default: 0]
  --toflx-dma D                         Use toFlx dma channel [default: 1]
  --elinks <elinkrange>                 E-link numbers/number range, comma separated
for instance
```

```
  --streams <elinkrange>              List of E-Links that carry streams
  --notoflx                           Do not start the To-FLX thread(s)
  --nobusy                            Do not start the Busy thread(s)

Data Generator Options:
  --fixed-chunks                      Use fixed chunk size
  --chunk-size-fixed N                Fixed size of chunks [default: 1018]
  --rate-control-bool                 Use rate control
  --chunk-size-min N                  Minimum size of chunks [default: 128]
  --chunk-size-max N                  Maximum size of chunks [default: 4096]
  --e-link-min N                      Minimum number of elinks [default: 50]
  --e-link-max N                      Maximum number of elinks [default: 150]
  --e-link-id-min <id>                Minimum ID of elinks [default: 1]
  --e-link-id-max <id>                Maximum ID of elinks (<2048) [default: 255]
  --period T                          Period in ms [default: 100]
  --max-overshoot T                   Max overshoot in ms [default: 10]
  --duration T                        Duration in s (0 = run forever) [default: 0]

Debug Options:
  --nostats                           Disable statistics
  --display-stats                     Print out statistics
  --nohistos                          Disable histograms
  --pmstats                           Enable poor man's statistics, implies
--nostats
  --trace                             Enable tracing
```

7.3. Monitoring

7.3.1. FelixCore Native Monitoring

The felixcore application provides monitoring and status information via a web front-end. By default the web front-end is available on port 8080. To access the web front-end use a web browser and point it to *http://hostname:8080*. Figure 27 shows a screenshot of the web app.

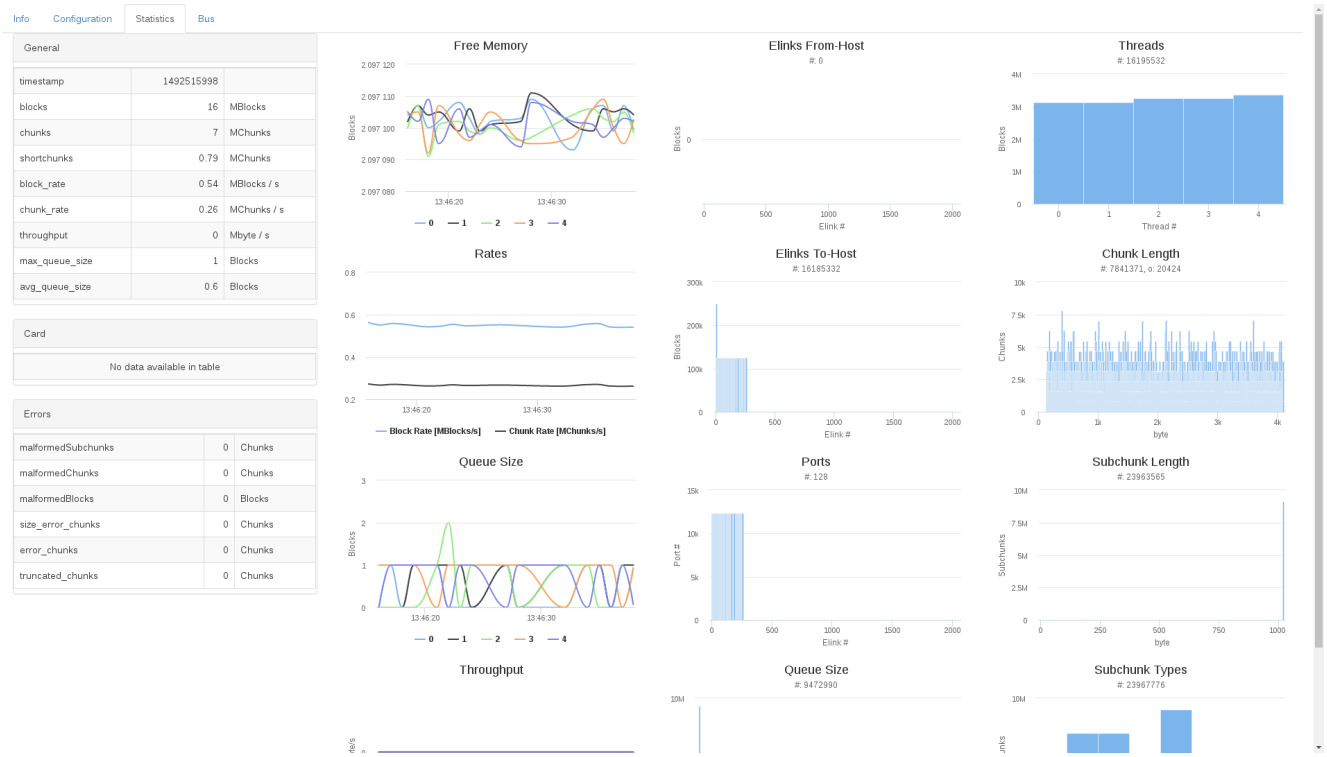

Figure 27. Screenshot of the integrated felixcore monitoring web app.

7.4. FelixCore Examples

When running with multiple threads, FELIX will publish data on multiple TCP/IP ports. The examples are all started with 1 thread using the option `-t 1`. This ensures that all data is published on a single TCP/IP port (default 12345), which facilitates debugging. To read out data from a felixcore application that is started as in the examples below, use a client (for example netio_cat or fatcat), and point it to port 12345 of the FELIX host. Of course, running with only one worker thread limits performance and, depending on the workload, FELIX might not be able to keep up with the load. In that scenario increase the number of worker threads accordingly. The tool `felix-bus-list` can be used to obtain the mapping of E-links to TCP ports of a FELIX system.

A running felixcore instance can be stopped by pressing `Ctrl+\`.

7.4.1. Tests without an FLX Card

Starting felixcore with input from a file (no card required):

```
$ felixcore -t 1 -f path/to/file.blocks --notoflx
```

7.4.2. Tests with an FLX Card

Starting felixcore with one processing thread and emulators enabled. Emulators are configured to send data via PCIe to the FELIX host software.

```
$ felixcore -t 1 --emu_to_host 0xff
```

E-link and emulator data configuration can be adjusted with the `elinkconfig` tool.

Starting felixcore with one processing thread and emulators enabled. Emulators are configured to

send data via detector links to external receivers or via loopbacks back to the FLX card:

```
$ felixcore -t 1 --emu_from_host 0xff
```

E-link and emulator data configuration can be adjusted with the `elinkconfig` tool.

7.5. Connecting to a felixcore instance using NetIO tools

The felixcore application uses the NetIO publish/subscribe system to distribute data. The tool `netio_cat` can be used to analyze published data. For example,

```
$ netio_cat subscribe -H 192.168.15.2 -t 15 -t 42 -e raw
```

will let `netio_cat` subscribe to E-links 15 and 42 of the felixcore application running on the host `192.168.15.2` with the default port 12345. The encoding is set to *raw*, which will simply write a hexdump of each received message. For other formatting and subscription options, see `netio_cat -h`.

7.6. Connecting to a felixcore instance using FATCAT

FATCAT is an advanced analysis and test client for FELIX and the successor of multiple ad-hoc tools like felix-client and felix-dcs. The application is currently experimental. For help options see

```
$ fatcat -h
```

In the future fatcat can be used to debug data streams coming from FELIX, send data to detectors via FELIX, record data to disk, manage data subscriptions among multiple FELIX hosts, benchmark FELIX systems and analyze recorded data.

7.7. Discovering E-links with the FELIX BUS system

Clients that want to receive data from or send data to specific E-links need to know by which FELIX instances the desired E-links are managed. The FELIX BUS system is used for this purpose. FELIX instances broadcast at regular intervals information about connected E-links. Clients can retrieve this information and build internal lookup tables using the library libfelixbus.

The E-Link IDs published by felixcore are global E-link IDs, i.e. they uniquely identify E-links across all FELIX hosts. This requires that the FELIX ID (command line option `--felix_id`) is set to a unique number for each felixcore instance. The default FELIX ID 0 is fine to use for tests where only a single FELIX is running.

The tool `felix-buslist` can be used to display the tables:

```
$ felix-buslist -t
Tables in FelixBus (ctrl\ to quit)

FelixTable::print(): table.size()=4
PeerId                           FelixID                           Address
1950EDDFD4821AF225D99EBCE9B22152  0468680B8D9D0388D8D1078B725D2E3B
tcp://10.193.16.62:12345
1950EDDFD4821AF225D99EBCE9B22152  238B9263BA79A05378652F433A25C949
tcp://10.193.16.62:12348
1950EDDFD4821AF225D99EBCE9B22152  4E418C3646578B46FD939AF6AC5FFB5B
tcp://10.193.16.62:12346
1950EDDFD4821AF225D99EBCE9B22152  C87FAE7D74FC1D99BFB454ACD74EBA23
tcp://10.193.16.62:12347

ElinkTable::print(): table.size()=10
PeerId                           FelixId                           ElinkId
1950EDDFD4821AF225D99EBCE9B22152  238B9263BA79A05378652F433A25C949  3735579
1950EDDFD4821AF225D99EBCE9B22152  C87FAE7D74FC1D99BFB454ACD74EBA23  3735580
1950EDDFD4821AF225D99EBCE9B22152  4E418C3646578B46FD939AF6AC5FFB5B  3735591
1950EDDFD4821AF225D99EBCE9B22152  0468680B8D9D0388D8D1078B725D2E3B  3735618
1950EDDFD4821AF225D99EBCE9B22152  238B9263BA79A05378652F433A25C949  3735676
1950EDDFD4821AF225D99EBCE9B22152  C87FAE7D74FC1D99BFB454ACD74EBA23  3735701
1950EDDFD4821AF225D99EBCE9B22152  4E418C3646578B46FD939AF6AC5FFB5B  3735729
1950EDDFD4821AF225D99EBCE9B22152  0468680B8D9D0388D8D1078B725D2E3B  3735732
1950EDDFD4821AF225D99EBCE9B22152  238B9263BA79A05378652F433A25C949  3735739
1950EDDFD4821AF225D99EBCE9B22152  C87FAE7D74FC1D99BFB454ACD74EBA23  3735741
```

The example shows that there is one felixcore instance running (peer ID 1950...) with four threads (FELIX ID 0468..., 238B..., 4E41..., C87F...), each reachable on a separate TCP port. This felixcore instance handles 10 different E-Links which are distributed among the four worker threads.

7.8. Debugging

7.8.1. Using the FelixCore event tracing framework

FelixCore includes a trace framework that can be used to gain a better understanding about latencies added by individual parts of a communication chain or to get detailed information about the dataflow of a single message through the system. The trace framework operates by logging events with precision timestamps to a file.

To enable the trace framework, start the felixcore application with the option \verb|--trace|. FelixCore will then timestamp events and record these events in a file "trace.csv" in the current working directory.

The contents of trace.csv will contain of lines similar to this:

```
20605900,MSG_RECV,0
20606530,FROMHOST_BLOCK_WRITE_START,0
20606584,FROMHOST_BLOCK_WRITE_COMPLETE,0
```

The CSV file contains three columns.

1. The first column contains the timestamp of the event in microseconds.
2. The second column contains the type of the event. Currently we have these events:

   TOHOST_BLOCK_READ
   : Issued when a 1 kB block is read from the FLX card

   FROMHOST_BLOCK_WRITE_START
   : Start of DMA transfer of blocks to the FLX card

   FROMHOST_BLOCK_WRITE_COMPLETE
   : DMA transfer to the card completed

   MSG_RECV
   : A message was received from the network for an E-link

   MSG_SEND
   : A data chunk is published and is being sent to clients in the network

3. The third column contains an E-Link id that associates an event with an E-link if applicable.

Note that the trace.csv file is **not sorted by timestamp**. If needed, the events in the file need to be sorted in a post-processing step.

Chapter 8. Felix-star Application and NetIO-next

8.1. Introduction

The felix-star application is the central process of a FELIX system, responsible for data transfers to and from the FELIX host. Active e-links, enabled with elinkconfig or similar, are advertised by felix-bus for both directions. In the to-host direction, felix-star sends data coming from the FELIX card to the clients according to their e-link subscriptions. In the from-host direction messages received through enabled e-links are transferred to the card.

The communication with felix-star over the network is managed by the NetIO-next library, based on libfabric and capable of exploiting RDMA technology. Two communication modes are defined to address the needs of different use-cases. The buffered mode implements data coalescing and is preferable when the data flow consists of many small messages (tens of bytes each). In practice, messages fill buffers called netio pages (of configurable size and number). Once the occupancy of a page exceeds a threshold (called watermark, also configurable) the page is sent. This approach decreases the overhead introduced by each transfer. The unbuffered mode targets a high-troughput scenario in which fewer links handle large messages (~kB). In this mode each message is transferred right away.

Client applications, such as the ATLAS SW ROD and OPC-UA server, do not have to interface directly with NetIO-next but are instead encourage to take advantage of the felix-client-thread C++ interface. In particular, the felix-client-thread reads the connection parameters (connection mode, number and sizer of netio pages) directly from the felix-bus.

Felix-star can print monitoring information formatted in JSON in local UNIX FIFOs and publish the same information over e-links. Later versions will include a web front-end.

8.2. Architecture

Different independent processes are responsible for the felix-star functionalities described above, as illustrated in Figure 28. Considering a setup with a single FLX-712 card, two *felix-tohost* processes, one per PCI-E endpoint, are instantiated to read the and forward the data coming from the card. Similarly, two *felix-toflx* processes are needed to serve all the from-host links. Both felix-tohost and felix-toflx write information about throughput statistics, recorded errors and published e-links in UNIX FIFOs. The *fifo2elink* utility reads the FIFOs and makes the read data available to clients via additional e-links (dubbed virtual as they do not correspond to e-links existing in firmware). The *dir2bus* utility reads exclusively the FIFOs containing information about the e-links and serves the felixbus. Finally the *felix-busy-tohost* and *felix-busy-toflx* applications allow to monitor and raise the busy status of a FELIX card. Details on each application are presented in Section 8.3, while their orchestration is discussed in Section 8.4.

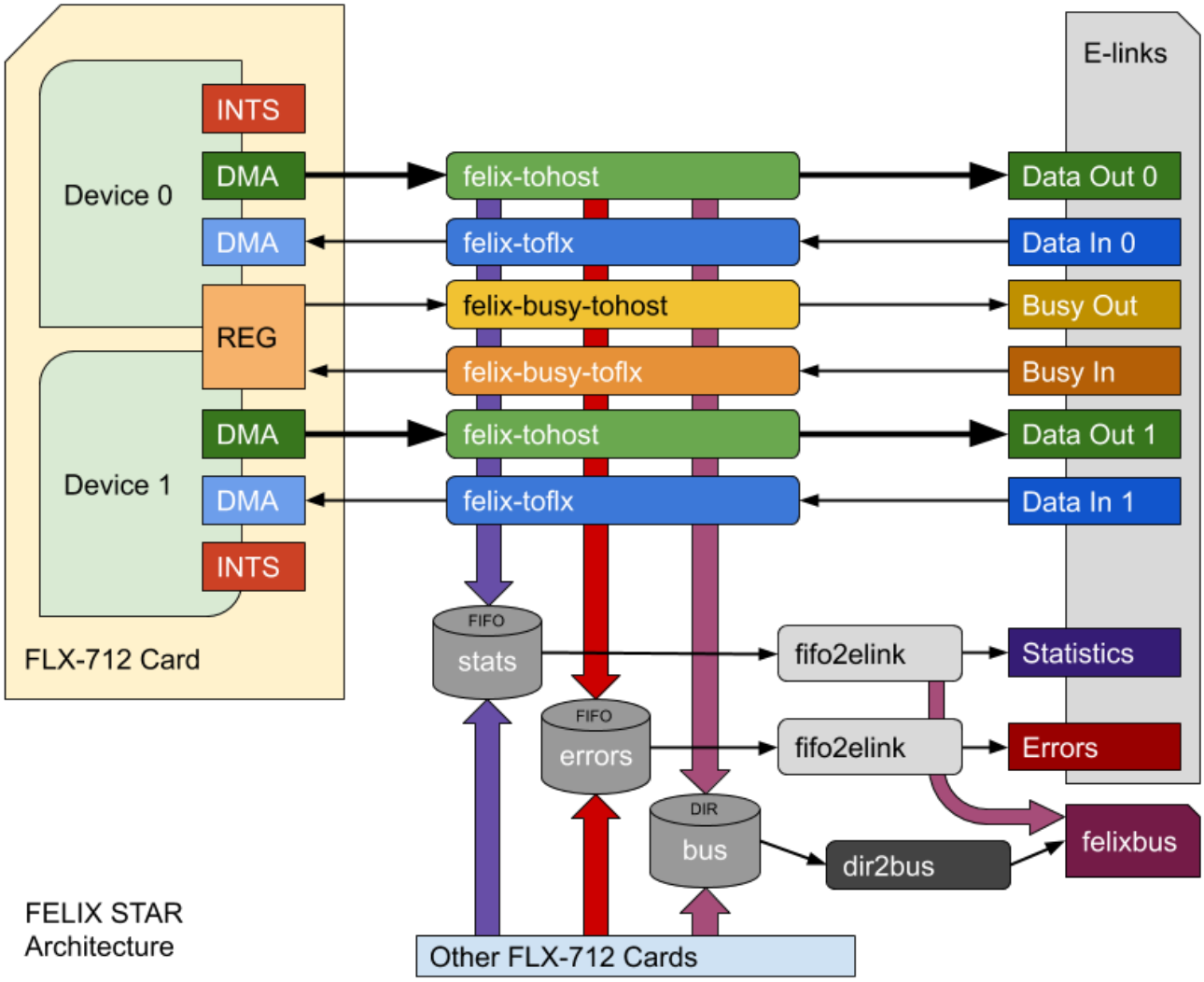

Figure 28. The architecture of the FELIX star application.

8.3. Felix Star commands

All felix-star-related commands are listed below. Some commands correspond to the application already described, while others consist of small utilities for configuration or debugging purposes (e.g. *file2host*, *display-stats*, *get-config-value*, *get-ip*,..)

8.3.1. felix-star

All processes can be launched using the felix-star command, or, alternatively, by the individual executables. Either way, a short help text is available from the command line.

```
Unresolved directive in 8_felix_star.adoc - include::include/felix-star.txt[]
```

8.3.2. felix-tohost

The main FELIX readout command is felix-tohost. Each instances reads only one PCI-E endpoint i.e. one "device". Felix-tohost is automatically run in buffered mode for GBT and unbuffered mode for FULL mode. By default data e-links are published on port 53100+X, while the TTC2H e-link is published on 53120+X, where X is the device number.

```
Usage: felix-tohost [OPTION...]
FELIX central data acquisition application

  -b, --netio-pagesize=SIZE  NetIO page size in Byte. Default: 64kB
      --buffered             Enable buffered mode (FULL-mode only)
      --bus-dir=DIRECTORY    Write felix-bus information to this directory.
                             Default: ./bus
      --bus-groupname=NAME   Use this groupname for the bus. Default: FELIX
  -B, --netio-pages=SIZE     Number of NetIO pages. Default: 256
  -c, --cmem=SIZE            CMEM buffer size in MB. Default: 1024
      --cid=N                CID (Connector Id) to set in FID (Felix ID),
                             incompatible with --co. Default: 0
      --co=N                 CO (Connector Offset) to offset FID (Felix ID),
                             incompatible with --did and --cid.
  -d, --device=DEVICE        Use FLX device DEVICE. Default: 0
      --did=N                DID (Detector Id) to set in FID (Felix ID),
                             incompatible with --co. Default: 0
  -D, --dma=ID               Use DMA descriptor ID. Default: 0
      --elink-offset=N       Offset for elinks as they are output and published
                             from felix-star (compatibility option). Default:
                             0
      --error-out=FIFO       Write error information to a UNIX FIFO
      --free-cmem            Free previously booked cmem segment by
                             name-<device>-<dma>
  -i, --ip=IP                Publish data on the ip address IP. Default:
                             localhost
  -p, --port=PORT            Publish data on port PORT. Default: 53100 +
                             10*device + dma
  -P, --poll-period=ns       Polling instead of interrupt-driven readout with
                             the given poll period in nanoseconds
      --stats-out=FIFO       Write periodic statistics data to a UNIX FIFO
      --stats-period=ms      Period in milliseconds for statistics dumps
  -t, --ttcport=PORT         Publish TTC2H data on port PORT. Default: 53300 +
                             10*device + dma
      --ttc-netio-pages=SIZE TTC Number of NetIO pages. Default: 256
      --ttc-netio-pagesize=SIZE   TTC NetIO page size in Byte. Default: 256kB
      --ttc-netio-watermark=SIZE   TTC NetIO watermark in Byte. Default: 56kB
  -v, --verbose              Produce verbose output
      --vid=N                VID (Version Id) to set in FID (Felix ID),
                             incompatible with --co. Default: 1
  -w, --netio-watermark=SIZE NetIO watermark in Byte. Default: 56kB
  -?, --help                 Give this help list
      --usage                Give a short usage message
  -V, --version              Print program version

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.

Report bugs to <atlas-tdaq-felix-developers@cern.ch>.
```

For example, a subsystem with Detector ID 0x12 wanting to connect to both devices on a single FELIX card would do the following:

```
felix-tohost --did 0x12 --cid 0x10 -d 0   # Device 0, links 0-11 in flx-info LINK,
links 0-11 in FelixID 'Link ID field'
felix-tohost --did 0x12 --cid 0x11 -d 1   # Device 1, links 12-23 in flx-info LINK,
links 0-11 in FelixID 'Link ID field'
```

Here, the connector ID `--cid` corresponds to each of the up to two optical fibre bundles connected to the FELIX card in question. As part of the FelixID structure these should also have their own ID, which is unique for a given Detector ID. In the example above they are 0x10 and 0x11 respectively.

Note - the FelixID field also includes a separate 'LinkID' structure, which counts from zero for each connector ID. Hence `link 0-ll` appears twice above, even though for the second device these links correspond to links 12-23 in the output of `flx-info LINK`.

8.3.3. felix-toflx

The main FELIX control command is felix-toflx. Each instance serves one PCI-E endpoint i.e. one "device". Upon startup felix-toflx listens the enabled e-links and forwards the received data to the card. The felix-toflx process can be run in either buffered (default) or unbuffered mode. The network port used by default is 53140+X, where X is the device number.

```
Usage: felix-toflx [OPTION...]
Host-to-Felix communication application (DCS)

  -b, --netio-buffersize=SIZE   NetIO receive buffer size in byte; maximum size
                              for a single message. Default: 64kB
      --bus-dir=DIRECTORY     Write felix-bus information to this directory.
                              Default: ./bus
      --bus-groupname=NAME    Use this groupname for the bus. Default: FELIX
  -B, --netio-buffers=SIZE    Number of NetIO receive buffers. Default: 256
  -c, --cmem=SIZE             CMEM buffer size in MB. Default: 20
      --cid=N                 CID (Connector Id) to set in FID (Felix ID),
                              incompatible with --co. Default: 0
      --co=N                  CO (Connector Offset) to offset FID (Felix ID),
                              incompatible with --did and --cid.
  -d, --device=DEVICE         Use FLX device DEVICE. Default: 0
      --did=N                 DID (Detector Id) to set in FID (Felix ID),
                              incompatible with --co. Default: 0
  -D, --dma=ID                Use DMA descriptor ID. Default: highest dma
      --error-out=FIFO        Write error information to a UNIX FIFO
      --free-cmem             Free previously booked cmem segment by
                              name-<device>-<dma>
  -i, --ip=IP                 Send data to the ip address IP. Default: localhost

  -m, --cdma                  Use circular DMA buffer. Default: one-shot
  -p, --port=PORT             Send data to port PORT. Default: 53200 + 10*device
                              + dma
      --stats-out=FIFO        Write periodic statistics data to a UNIX FIFO:
                              path to fifo. Default: disabled
      --stats-period=MS       Period in milliseconds for statistics dumps.
                              Default: 1000
      --stats-stdout          Prints stats to stdout. Default: false
  -u, --unbuffered            Use unbuffered mode
  -v, --verbose               Produce verbose output
      --vid=N                 VID (Version Id) to set in FID (Felix ID),
                              incompatible with --co. Default: 1
  -w, --netio-watermark=SIZE NetIO receive watermark size in byte; start of
                              flush. Default: 56kB
  -?, --help                  Give this help list
      --usage                 Give a short usage message
  -V, --version               Print program version

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.

Report bugs to <atlas-tdaq-felix-developers@cern.ch>.
```

8.3.4. felix-busy-tohost

The felix-busy-tohost command monitors the Busy signal and publishes its information on an e-

link. As each card has only one busy input you would only need to run one felix-busy-tohost process per card.

```
Unresolved directive in 8_felix_star.adoc - include::include/felix-busy-tohost.txt[]
```

8.3.5. felix-busy-toflx

The felix-busy-toflx process allows the busy to be raised via an e-link. As each card has only one Busy connector you would only need to run one felix-busy-toflx process per card.

```
Unresolved directive in 8_felix_star.adoc - include::include/felix-busy-toflx.txt[]
```

8.3.6. felix-fifo2elink

The felix-fifo2elink process is used to read out the error or monitoring FIFO and publish the information as an e-link. The FIFOs are written by all felix-tohost and felix-toflx processes at the same time.

```
Publish JSON fifo and routes to one or more e-links (FIDs)

Usage:
    felix-fifo2elink [options] <fifo> <hostname_or_ip> <port> <elink> (<did> <cid>)...

Options:
    -b, --netio-pagesize=SIZE       NetIO page size in kilobytes [default: 64]
    -B, --netio-pages=N             Number of NetIO pages [default: 256]
    -g, --bus-groupname GROUP_NAME  Group name to use for the bus [default: FELIX]
    -b, --bus-dir DIRECTORY         Directory to use for the bus [default: ./bus]
    -v, --verbose                   Verbose output
    -w, --netio-watermark=SIZE      NetIO watermark in kilobytes [default: 56]
    --error-out=fifo                (NA) Write error information to the error FIFO

Arguments:
    <fifo>              FIFO (JSON) to read from
    <hostname_or_ip>    Hostname or IP to publish from
    <port>              Port number to use
    <elink>             Elink to use for FID
    <did>               DetectorID to use for FID
    <cid>               ConnectorID to use for FID
```

8.3.7. felix-dir2bus

The felix-dir2bus process monitors the bus directory for new files and changes to those files. These files are one by one written by each of the felix-tohost and felix-toflx processes. The files are kept open as long as the process is running. felix-dir2bus reads the information from the files and publishes it to the FELIX bus.

```
Unresolved directive in 8_felix_star.adoc - include::include/felix-dir2bus.txt[]
```

8.3.8. felix-elink2file

The felix-elink2file debug utility writes the input on an E-link into a file. Not to be used for Data Acquisition.

```
Subscribe an e-link and output to file.

Usage:
    felix-elink2file [options] <local_hostname> <fid> <file>

Options:
    -b, --bus-dir DIRECTORY         Directory for the bus [default: ./bus]
    -g, --bus-groupname GROUP_NAME  Group name to use [default: FELIX]
    -v, --verbose                   Verbose output
    --error-out=fifo                (NI) Write error information to a UNIX FIFO

Arguments:
    <local_hostname>    Internet hostname to subscribe from
    <fid>               FID to subscribe to
    <file>              File to write to
```

8.3.9. felix-file2host

The felix-file2host takes a raw data file and reads it as if it was a FELIX card. The e-links used in the raw data file have to be specified in the command line.

```
Usage: felix-file2host [OPTION...] FILE
FELIX central data acquisition application

  -b, --netio-pagesize=SIZE  NetIO page size in Byte. [default: 64kB]
      --block-size=SIZE      Use as block size, multiple of 1024. [default:
                             1024]
      --buffered             Enable buffered mode (FULL-mode only)
      --bus-dir=DIRECTORY    Write felix-bus information to this directory.
                             Default: ./bus
      --bus-groupname=NAME   Use this groupname for the bus. Default: FELIX
  -B, --netio-pages=SIZE     Number of NetIO pages. [default: 256]
  -c, --cmem=SIZE            CMEM buffer size in MB. [default: 1024]
      --cid=N                CID (Connector Id) to set in FID (Felix ID),
                             incompatible with --co. Default: 0
      --co=N                 CO (Connector Offset) to offset FID (Felix ID),
                             incompatible with --did and --cid.
  -d, --device=DEVICE        Use FLX device DEVICE. Default: 0
      --did=N                DID (Detector Id) to set in FID (Felix ID),
                             incompatible with --co. Default: 0
```

```
  -D, --dma=ID               Use DMA descriptor ID. Default: 0
      --error-out=FIFO       Write error information to a UNIX FIFO
      --free-cmem            Free previously booked cmem segment by
                             name-<device>-<dma>
  -i, --ip=IP                Publish data on the ip address IP. [default:
                             localhost]
  -m, --full-mode            Use full-mode. [default: gbt-mode]
  -p, --port=PORT            Publish data on port PORT. [default: 53100 +
                             10*device + dma]
  -P, --poll-period=ns       Polling instead of interrupt-driven readout with
                             the given poll period in nanoseconds
      --regmap=hex_version   Register map version. Default: 0x0400
      --stats-out=FIFO       Write periodic statistics data to a UNIX FIFO
      --stats-period=ms      Period in milliseconds for statistics dumps
  -s, --ism                  Ignore check on sequence numbers in blocks.
      --ttc2h-enable         Enable TTC2H elink
      --ttcport=PORT         Publish TTC2H data on port PORT. Default: 53300 +
                             10*device + dma
  -t, --tag=TAG              Read tag (elink) from file. This option may be
                             used multiple times (2048 max).
      --trailer-size=SIZE    Use as trailer size, 2 or 4. [default: 4]
  -u, --stream=TAG           Read tag (elink streams) from file. This option
                             may be used multiple times (2048 max).
  -v, --verbose              Produce verbose output
      --vid=N                VID (Version Id) to set in FID (Felix ID),
                             incompatible with --co. Default: 1
      --vmem                 Use virtual memory. [default: cmem]
  -w, --netio-watermark=SIZE NetIO watermark in Byte. [default: 56kB]
  -?, --help                 Give this help list
      --usage                Give a short usage message
  -V, --version              Print program version

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.

Report bugs to <atlas-tdaq-felix-developers@cern.ch>.
```

8.3.10. felix-display-stats

The felix-display-stats debug utility connects to the monitoring FIFO and displays some statistics info on the terminal. As one can only have one reader on a FIFO this utility cannot be run at the same time as felix-fifo2elink on the same monitoring FIFO.

```
Shows stats for debugging.

Usage:
    felix-display-stats [options] <stats-fifo>

Options:
    -h, --help                  Display this help
    -V, --version               Display version

Arguments:
    <stats-fifo>                Fifo where statistics are written.
```

8.3.11. felix-get-config-value

The felix-get-config-value command retrieves a hostname-device-mode-key indexed value from the FELIX configuration file. Available keys are: 'iface', 'did', 'cid' and 'elink_config'. The configuration can be setup to run in FULL mode or GBT mode depending on the firmware found in the card.

This utility has been designed for deployment within the ATLAS infrastructure (rather than a user testlab). The elinkconfig .elc/.yml files listed indicate which configuration file has to be loaded on a card in the start-up sequence of a FELIX PC

An example of a config file follows:

```
# host_name       device#   mode   interface    detector id (8 bit)   connector id (20
bit)   elink config
pc-tbed-felix-04  0         gbt    eth-100-0    0x07                  0x12000
gbt-mode-2bits.elc
pc-tbed-felix-04  1         gbt    eth-100-0    0x07                  0x12010
gbt-mode-2bits.yml
pc-tbed-felix-04  2         gbt    eth-100-0    0x07                  0x1a000
gbt-mode-2bits.elc
pc-tbed-felix-04  3         gbt    eth-100-0    0x07                  0x1b000
gbt-mode-2bits.elc
pc-tbed-felix-04  0         full   eth-100-0    0x07                  0x12000
full-mode-8bits.elc
pc-tbed-felix-04  1         full   eth-100-0    0x07                  0x12010
full-mode-8bits.yml
pc-tbed-felix-07  0         full   eth-25-0     0x20                  0x1b000
full-mode-8bits.yml
pc-tbed-felix-07  1         full   eth-25-0     0x20                  0x1b010
full-mode-8bits.yml
```

```
Lookup variables from felix configuration file.

Usage:
    felix-get-config-value [options] <host_name> <device> <mode> <key>

Options:
    -h, --hex                   Hexadecimal output
    -v, --verbose               Verbose output
    -c, --config=FILE           Specify FELIX config file [default: felix.cfg]

Arguments:
    <host_name>                 Hostname or ip, as declared in the felix config
    <device>                    FLX device number
    <mode>                      Mode of the device ('gbt', 'full'), see felix-get-mode
    <key>                       Item to be retrieved (e.g. 'iface', 'did', 'cid' or
'elink_config')

Config file format:
    # comments
    host_ip device mode interface detectorid connectorid elinkconfig [# comments]
```

8.3.12. felix-get-ip

The felix-get-ip command retrieves the IP for a given interface name. This interface name is retrieved using the felix-get-config-value command. The IP is used to start felix-star.

```
Return (first or fastest) ip4 address for interface or the list of available
interfaces.

Usage:
    felix-get-ip [options] [<interface_name>]

Options:
    --fastest                   Return ip for fastest network
    -v, --verbose               Verbose output

Arguments:
    <interface_name>            Name of the ethernet interface, see ifconfig

Config file format:
    # comments
    host_ip device detectorid connectorid elinkconfig [# comments]
```

8.3.13. felix-get-mode

The felix-get-mode command returns the mode for a particular device. It is used to lookup values in the configuration using felix-get-config-value.

```
Return mode for device.

Usage:
    felix-get-mode [options] <device>

Options:
    -v, --verbose               Verbose output

Arguments:
    <device>                    Device number
```

8.3.14. felix-fid

The felix-fid utility translates and decodes FIDs to DID, CID, E-Link, TID and SIDs and vice-versa. Full details of the structure of an FID can be found in:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/docs/LinkMappingSpecification.pdf

```
Usage: felix-fid [OPTION...] <felix id>
  or:  felix-fid [OPTION...]
            <detector id> <connector id> <link id> <transport id> [<stream
            id>]
FELIX FID conversion (accepts hex 0x, binary 0b and octal 0)

  -c, --cid                  print Connector ID
  -d, --did                  print Detector ID
  -e, --elink                print E-link
  -h, --hex                  print in hex format
  -i, --vid                  print Version ID
  -l, --lid                  print Link ID
  -o, --co                   print Connector Offset
  -s, --sid                  print Stream ID
  -t, --tid                  print Transport ID
  -u, --update=VERSION       Update Version ID. Default: 1
  -v, --verbose              Produce verbose output
  -?, --help                 Give this help list
      --usage                Give a short usage message
  -V, --version              Print program version

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.

Report bugs to <atlas-tdaq-felix-developers@cern.ch>.
```

Verbose output of FID 0x123456789abc3def for example shows:

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|FID                                                            |
|                                               0x123456789abcdef0|
|                                              1311768467463790320|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|CO                         |GID                                |
|                  0x1234567|                        0x89abcdef0|
|                   19088743|                        36955807472|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|VID|DID    |CID            |LID          |TID          |SID    |
|0x1|   0x23|         0x4567|       0x226a|       0x3cde|   0xf0|
|  1|     35|          17767|         8810|        15582|    240|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|VID|DID    |CID            |V Elink            |D Proto|SID    |
|0x1|   0x23|         0x4567|1           0x09abc|1  0x5e|   0xf0|
|  1|     35|          17767|1             39612|1    94|    240|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

FID:   (64 bits) - Felix ID
CO:    (28 bits) - Connector Offset
GID:   (36 bits) - Generic ID
VID:   (4 bits)  - Version ID
DID:   (8 bits)  - Detector ID
CID:   (16 bits) - Connector ID
LID:   (14 bits) - Link ID
TID:   (14 bits) - Transport ID
SID:   (8 bits)  - Stream ID
V:     (1 bit)   - Virtual
Elink: (19 bits) - As used in the FELIX card
D:     (1 bit)   - Direction [0=tohost, 1=toflx]
Proto: (7 bits)  - Protocol
```

8.4. Startup and Configuration

Superviord provides a convenient solution to startup and shutdown all the processes needed to run a full felix-star configuration. Supervisord is included in the FELIX distribution and needs only a few configuration files for which examples are provided.

To launch supervisord type from the root directory of the distribution:

```
supervisord
```

which tries to find the 'etc' directory just below the current directory. It parses all the configuration files. The 'etc' directory also contains felix.cfg which has to be edited according to the user’s setup.

To start felix-star all processes in GBT mode type:

```
supervisorctl start gbt:*
```

To shutdown all processes and supervisord itself type:

```
supervisorctl shutdown
```

More commands in 'supervisorctl --help' are available.

neither flx-init nor feconf (elinkconfig) is run at this moment by supervisord.

8.5. Monitoring

Native monitoring is being developed and currently not part of the release. The felix-display-stats utility described above can be used for the moment.

8.6. Discovering E-links with the FELIX BUS system

Clients that want to receive data from or send data to specific E-links need to know by which FELIX instances the desired E-links are managed. The FELIX BUS system is used for this purpose. FELIX instances broadcast at regular intervals information about connected E-links. Clients can retrieve this information and build internal lookup tables using the library libfelixbus.

The E-Link IDs published by felixcore are global FELIX IDs, i.e. they uniquely identify E-links across all FELIX hosts. This requires that the Detector ID and Connector ID options are set (command line option `--did` and `--cid`). The default did and cid are 0 are fine to use for tests where only a single felix-tohost is running.

The tool `felix-buslist` can be used to display the tables:

```
$ felix-buslist -t
Tables in FelixBus (ctrl\ to quit)

FelixTable::print(): table.size()=1
PeerId                           FelixInstanceID                  Address  PubSub
Unbuffered  PageSize  Pages
1950EDDFD4821AF225D99EBCE9B22152  0468680B8D9D0388D8D1078B725D2E3B
tcp://10.193.16.62:53100 ...

ElinkTable::print(): table.size()=2
PeerId                           FelixInstanceId                  FID or ElinkId
1950EDDFD4821AF225D99EBCE9B22152  0468680B8D9D0388D8D1078B725D2E3B  3735618
1950EDDFD4821AF225D99EBCE9B22152  0468680B8D9D0388D8D1078B725D2E3B  3735732
```

The example shows that there is one felix-tohost instance running on a host (peer ID 1950...) with Felix Instance ID (0468...). This felix-tohost instance handles 2 different E-Links.

8.7. Subscribing to streams

In order to make use of the streaming feature, support must be enabled for the links in question using feconf or elinkconfig. This will cause felix-star to publish the full set of 256 FIDs corresponding to the available streams on that link. The stream ID is the last byte in the FID, which will therefore have the value 0-255 (rather than the default value of 0).

It is then possible to subscribe to a particular FID as normal by additionally setting the last byte to that StreamID, 0-255. The front-end corresponding to the links in question should then make sure to put the correct stream ID in the first byte of every packet sent over the link. FELIX will then guarantee to route the packet correctly to client applications with the appropriate subscription.

8.8. Quick start and testing procedures

8.8.1. Check connectivity and data transmission (no felix-bus)

Given a PC equipped with a FELIX card receiving or generating data the minimal set of commands to verify the felix-star functionality are, from any directory:

```
mkdir bus
mkfifo stats
felix-tohost --bus-dir ./bus --ip <IP of the network interface> --stats-out=stats
```

the monitoring statistics can be visualised with felix-display-stats or simply with `cat < stats `. It is then possible use a client application. In case the connection is buffered (default for GBT firmware):

```
felix-test-swrod-buffered <local-ip> <remote-ip> <port> -t <first e-link> -t
<second e-link> -t <third e-link> ...
```

In case the connection is unbuffered

```
felix-test-swrod-unbuf <local-ip> <remote-ip> <port> -t <first e-link> -t
<second e-link> -t <third e-link> ...
```

These applications use the e-link number and not the FID to identify e-links. For example, the 2-bit GBT e-links of group 1 are 8, 9, 10, 11, 12, 13, 14, 15, while for FULL mode the links are 0, 64, 128, 192, 256, 320, … .

8.8.2. Check connectivity and data transmission (incl. felix-bus)

In order to include the felix-bus in the test the minimal set of commands is

```
mkdir bus
mkfifo stats
felix-tohost --bus-dir ./bus --ip <IP of the network interface> --stats-out=stats
felix-dir2bus -i <interface name> ./bus
```

Then, on the client side the published links can be discovered with

```
felix-buslist -i <interface name> -t
```

Then it is possible to subscribe with

```
felix-client-thread-subscribe <local_ip_or_interface> <fids>
```

In this case the connection parameters are resolved via the felix-bus.

8.9. The felix-client-interface

Connecting to felix-star can also be done using a generic felix-client-interface. This interfaces defines a small number of methods to subscribe to and unsubscribe from an e-link, send data to an e-link and get notified when a connection succeeds and when it fails. On reception of data a callback function is called. The implementation of this interface takes care or re-subscribing if a connection is lost. It also takes care of talking to felix-bus to look up the machine, port number and other parameters for the connection. The user of the felix-client-interface is only concerned with data and E-links/FIDs. All these functions run in a single felix-client thread. If you want to execute a user function on the same thread an exec function is provided.

The interface looks like this:

```
class FelixClientThreadInterface {

public:
    typedef std::function<void (uint64_t fid, const uint8_t* data, size_t size,
uint8_t status)> OnDataCallback;
    typedef std::function<void ()> OnInitCallback;
    typedef std::function<void (uint64_t fid)> OnConnectCallback;
    typedef std::function<void (uint64_t fid)> OnDisconnectCallback;
    typedef std::function<void ()> UserFunction;
    typedef std::map<std::string, std::string> Properties;

    struct Config {
        OnDataCallback on_data_callback;
        OnInitCallback on_init_callback;
        OnConnectCallback on_connect_callback;
        OnDisconnectCallback on_disconnect_callback;
        Properties property;
    };

    virtual ~FelixClientThreadInterface() {};

    virtual void send_data(uint64_t fid, const uint8_t* data, size_t size, bool flush)
= 0;

    virtual void subscribe(uint64_t fid) = 0;
    virtual void unsubscribe(uint64_t fid) = 0;

    virtual void exec(const UserFunction &user_function) = 0;
};
```

Examples on how to use the interface are available at:

https://gitlab.cern.ch/atlas-tdaq-felix/felix-client/-/tree/master/examples

To use the felix-client-interface and link with its proxy use:

https://gitlab.cern.ch/atlas-tdaq-felix/felix-client-thread

and use the build-standalone.sh script to create the library to link with.

To run with the felix-client-interface you will need a felix installation setup. The proxy will find the felix installation and load all the necessary files underneath.

Chapter 9. FAQ, Troubleshooting and User Resources

This section is aimed at collecting useful information for front-end developers to aid the design and implementation of front-end firmware/hardware for interaction with FELIX. Useful tips based on experience so far will also be presented, in a section that will grow over time as more feedback is received.

9.1. Frequently Asked Questions

1. *Is GBT wide mode supported?*

   Not currently, can be reviewed on request.

2. *Is GBT 8b/10b mode supported?*

   8b/10b encoded E-links within GBT frames are supported, but this is different from native GBT '8b/10b frame mode', which is not supported.

3. *Is the phase of the eight "utility" clocks fixed with respect to the E-link clocks?*

   Yes, there is a fixed relationship with the E-Link clocks. Note that the eight utility clocks have *worse* jitter than the E-link clocks.

4. *Can the GBT output a 40 MHz E-link clock, use that clock in 40 MHz DDR mode for the to-frontend link, but accept data on the uplink at 160 or 320 Mb/s? (Assuming the FE ASIC multiplies the 40 MHz to 80, 160 or 320 MHz.)*

   Yes, that is possible. Also the to-frontend link can receive at 80, 160 or 320 Mb/s.

5. *Is there a maximum packet length on the E-link in 8b/10b mode?*

   No.

6. *Are direct mode a.k.a. unencoded E-links supported in GBT mode?*

   Not by default in the Phase-I firmware, but this being integrated into Phase-II builds. Please contact the FELIX team for more information.

7. *Can FELIX support additional (or custom) link protocols as they are developed?*

   FELIX will not support additional (or custom) link protocols unless well motivated by a detector requirement. If you think you will need to introduce an additional protocol please contact the FELIX team before making any final implementation decision.

9.2. Troubleshooting

9.2.1. Known Issues with GBTx

- Links disconnected from any front-end source generate spurious data at random intervals. If using FELIX with a GBTx, it is strongly recommended that any links which are disconnected from the front-end be deactivated in elinkconfig. This will prevent spurious data causing confusion in front-end testing. The effect of spurious data from accidentally or temporarily disconnected E-links can be minimized by using the packet truncation options.

- The loading of the configuration from the e-fuses on power-on is not reliable. The GBTx must be explicitly configured on every power-on. For GBTx' used only as transmitters, a way to configure them via I2C must be provided.

9.2.2. IOMMU

Some users have experienced problems with DMA transfers on systems with an active IOMMU. If your FLX card is recognized correctly but the DMA transfers are not working, check the kernel logs. If you find DMAR errors on memory access, it could be that the kernel is running with intel-iommu enabled, and you will need to disable every memory mapping by hand. This can be done by executing the command:

```
dmesg | grep -i -e DMAR -e IOMMU
```

After disabling IOMMU, the card should work without any issue.

9.2.3. File Descriptor (FD) Limit

Depending on the setup of your operating system, it may be necessary to change the FD limit to avoid running out of descriptors and experiencing the following types of error:

For felixcore:

```
terminate called after throwing an instance of 'std::system_error' what(): Too many open files
Aborted (core dumped)
```

For OPC-UA:

```
terminate called after throwing an instance of 'std::runtime_error' what(): could not connect
to endpoint 128.141.177.225:12351 Aborted (core dumped)
```

The current FD limit on can be seen by running:

```
ulimit -aH
```

and looking for the entry marked 'open files'.

The limit can be changed by modifying `/etc/security/limits.conf`. It is recommended to set the limit to the maximum available value, namely 65536.

9.2.4. Debugging Link Status

When using flx-info to check the status of your links, note that this only corresponds to alignment of incoming (Rx) links, for which it is possible to recover a clock. To check the status of links from FELIX to other electronics it is recommended to implement dedicated monitoring to check alignment there.

In the case of the FLX-712 card, flx-info does provide incoming and outgoing optical power measures, so this should be used to confirm whether a link problem is due to a problem with an optical fibre or light transmission from FELIX.

9.2.5. SMBus Access

SMBus access, as needed for the fflash tool, has been found to not work on Supermicro servers with an X11SPW-TF motherboard when using default factory configuration. However, Supermicro has provided custom firmware, which provides this support by means of "ipmitool raw" commands. This is firmware for the IPMI interface, i.e. not a BIOS, and has to be loaded via the IPMI network connection. The firmware is available here:

https://drive.google.com/open?id=1eVae35mJhdRZam3WlfW0YM2cpXpCz5yB

The file to be loaded is BETA_X11DP_Xilinx_Kintex_UltraScale_FPGA_982_20190628.bin. Follow this description to load it:

https://www.supermicro.com/manuals/other/IPMI_Users_Guide.pdf (chapter 2-99)

The following test with fflash has been done using the latest BIOS and after reloading the BETA firmware. Thanks to Henk Boterenbrood for performing the test.

The help information reported by fflash -h is as follows:

```
fflash version 21020800
Tool for loading a firmware image from one of the partitions
of the onboard flash memory of an FLX-712 into the card's FPGA,
issueing commands to the host system I2C bus to achieve this.
A subsequent hotplug procedure or machine reboot is required.

Usage: fflash [-h|V] [-q] -f <flashnr>
              [[-L|I] [-U|P -d <devslot>] [-S] [-b <busnr>] [-r <chan>]
              [-s <saddr>] [-u <uaddr>] [-T <sec>]]
  -h          : Show this help text.
  -V          : Show version.
  -q          : Be quiet (only errors will be displayed).
  -f <flashnr>: Flash memory segment partition [0..3] selection (no default).
  -I          : Generate an INIT_B pulse on the FLX-card (to reset flash devices).
  -L          : Load firmware from the given flash partition into the card.

The following options are relevant in conjunction with the -L and/or -I option:
  -b <busnr>  : I2C-bus number (default=0).
  -r <chan>   : Riser card I2C-switch channel number (default=0).
                NB: I2C-switch has hard-coded I2C address 0x70.
  -s <addr>   : I2C-switch I2C-address (hex, default=0x77, expected range: 0x70-0x77).
                NB: 0x70 already taken by the riser card I2C-switch!
  -u <addr>   : uC I2C-address (hex, default=0x67, expected range: 0x60-0x67).
  -U          : Use USB I2C-dongle instead of system SMBus
                (requires scripts i2cset.py and i2cget.py installed in /opt/flx).
  -P          : Use 'ipmitool' to access system SMBus.
                !NB: use -d option to select 'device slot': 1 or 2.
  -T <sec>    : Set 'Prog-done' timeout [s] (default: 7)
  -d <devslot>: Device slot (1 or 2), only in combination with -P.
  -S          : Preceed calls to i2cget/set or ipmitool with 'sudo'.
                (default: 'sudo' not used; applies to options -L|I|P|U).
```

```
Examples:
---------
Load flash memory image partition #2 into the card:
  fflash -f 2 -L

Load flash memory image partition #2 into the card, using I2C-bus #1,
riser card I2C-switch channel #0, FLX-card I2C-switch address 0x75 and
FLX-card microcontroller I2C-address 0x65:
  fflash -f 2 -L -b 1 -r 0 -s 75 -u 65

How to determine the I2C-switch and uC I2C addresses
(options -s and -u respectively) :
Note 1: there is an I2C-bus number (option -b) to select as well,
  which is assumed to have the value '1' (following '-y') in the examples below.
Note 2: in the standard FELIX server there is an additional I2C-switch
  on the socalled riser card; its channel is selected using option -r;
  it means that the 2 FLX-cards in such a server may have identical
  '-s' and '-u' addresses, i.e. most likely their defaults
  while the riser card setting is: 'top' position = -r 0, 'bottom' = -r 1.
'sudo i2cdetect -y 1' should show you an address in the range 0x70-0x77,
let's say 0x77; this is then the address to use in option -s;
subsequently run 'sudo i2cset -y 1 0x77 1' to set the I2C-switch
causing an additional address in the range 0x60-0x67 to appear
in the output of 'sudo i2cdetect -y 1', so run that command again;
this is the address to use in option -u.
On the FLX-712 dipswitch J14 configures the '-s' and '-u' addresses:
  switch 1-3 to set 3 LSBs of '-s', i.e. 0x70-0x77
  switch 4-6 to set 3 LSBs of '-u', i.e. 0x60-0x67
```

Example of the use of fflash to configure the FPGA from partition 2 of the FLASH memory (the program has to be run using sudo due to the use of "sudo ipmitool" commands):

```
fflash -f 2 -P -L -d 2 -u 63 -s 76
```

```
Parameters:
-f 2: partition 2
-P: use of ipmitool
-L: load firmware from FLASH memory
-d 2: card in slot 2 from riser
-u 63: i2c address of card is 0x63 (can be set with jumpers on the card)
-s 76: address i2c switch on riser is 0x76

Output program with these command line parameters for our machine:

=> Load firmware partition 2 (I2C via IPMI: switch=0xec, uC=0xc6):
sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 236 0 1

sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 198 0 17 0 16

sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 198 0 16 56

sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 198 0 17 48

sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 198 0 18 0

sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 198 1 12
Prog done (time: 1320 ms)
sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 198 0 16 0

sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 236 0 0

=> Pulse INIT_B (I2C via IPMI: switch=0xec, uC=0xc6):
sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 236 0 1

sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 198 0 19 4

sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 198 0 18 4

sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 198 0 19 0

sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 198 0 19 4

sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 198 0 18 0

sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 236 0 0
```

An attempt to read from an incorrect address looks like this:

```
sudo ipmitool raw 0x30 0x70 0xd5 27 1 2 208 1 12
```

```
Unable to send RAW command (channel=0x0 netfn=0x30 lun=0x0 cmd=0x70 rsp=0x83): Unknown
(0x83)
```

Form of the ipmitool raw command:

`ipmitool raw 0x30 0x70 0xd5 27 1 slot address readcount writedata`

```
slot: device slot: 1 or 2
address: i2 address
readcount: number of bytes to read, if 0: no bytes to read
writedata: bytes to be written
```

NB: The ipmitool raw command requires the addresses to be shifted one bit to the left

9.2.6. Problems with CMEM allocation on boot

Note - if you have a network-booted system (e.g. managed by the ATLAS TDAQ Sysadmins in the context of the ATLAS testbed infrastructure) then you likely won't be able to perform the actions listed below. In this case it might be necessary for you to consider moving to a local boot. Please contact markus.joos@cern.ch for more advice.

If you have a local boot with RPM driver installation, and are having problems configuring CMEM to consistently allocate memory on boot please attempt the following:

1. Contact the FELIX team to confirm the required amount of memory for your setup. The typical recommendation is 4 GB per FELIX card.
    - Depending on your setup and use case, it may be necessary to consider adding additional RAM.
2. If necessary, amend the allocation by opening `/etc/init.d/drivers_flx` and changing `gfpbpa_size=` to the new amount.
3. If this doesn't help, try configuring `systemd` such that `drivers_flx_sd.service` runs as the first service. This can be achieved as follows:
    - Run `systemd-analyze plot drivers_flx_sd.service > p.svg; eog p.svg`. You will see something like what is shown in Figure 29.
    - Open the file `/etc/systemd/system/drivers_flx_sd.service`
    - Look for the `[Unit]`section. Add the statement `BEFORE=` at the end of the `[Unit]` section and list all of the services that are in front of `drivers_flx_sd` (see example below).
    - Reboot the computer and check `systemd-analyze plot drivers_flx_sd.service > p.svg; eog p.svg` again. Keep adding services to the `BEFORE=` list until `drivers_flx_sd` is the first service, as shown in Figure 30
    - Try multiple reboots and confirm that this results in stable allocation.
4. If this still does not resolve the issue please contact markus.joos@cern.ch for additional support.

Example modification of [Unit] block:

```
[Unit]

Description=Start the drivers required by a TDAC PC or SBC
DefaultDependencies=no

Before=kmod-static-nodes.service systemd-udevd-kernel.socket dev-hugepages.mount
dev-mqueue.mount nss-user-lookup.target machine.slice dm-event.socket user.slice
slices.target
```

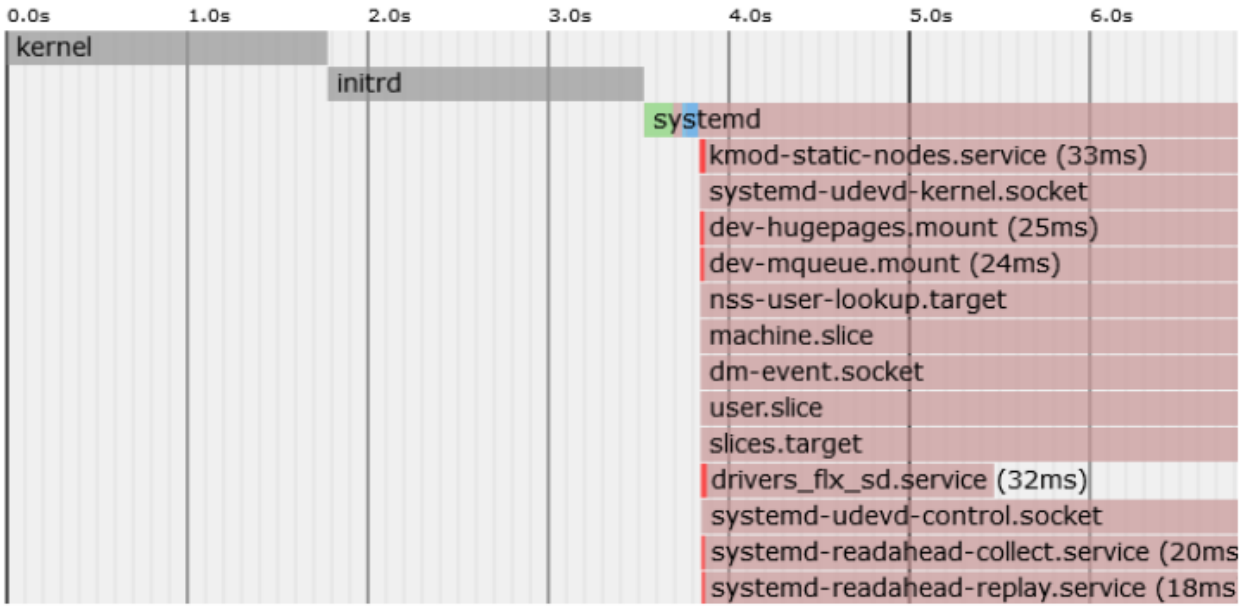

Figure 29. Example output of `systemd-analyse`.

[cmem systemdok output] | *figures/cmem_systemdok_output.png*

Figure 30. Example output of `systemd-analyse` *with drivers_flx first.*

9.3. Guide for System Designers

- For GBT-mode transmission to FELIX, use 8b/10b encoding on the E-links. Avoid the non-encoded fixed length or variable length formats, because no resynchronization is possible if bits are lost or repeated on the E-link. Comma symbols are used to align to *10-bit symbols* in the bit stream. They are considered idles and can be inserted in the data stream anywhere. Transmit "frequent" pairs of commas. This will minimize data loss when FELIX tries to resynchronize when the symbol boundary is lost due to a missed or repeated bit on the E-link. The out-of-band SoP (Start-of-Packet) and EoP (End-of-Packet) "K-Characters" are used to delimit packets in 8b/10b E-links.
- The E-link clock, input, and output data rates are independent. The only restriction is that within a GBT E-link group, all the clocks must have the same frequency, all the data inputs the same data rate and all the outputs the same data rate. However, groups can be setup independently from each other. Read the GBTx manual carefully to understand the GBTx group restrictions and bit order. Note, however, that a clock output is only available if its corresponding Tx is enabled. This means, for example, that a bank running with 320 Mb/s E-links can supply only **two** (identical) clocks, but they can be 40, 80, 160 or 320 MHz.

- E-link "chunks" or packets are even multiples of bytes or 8b/10b symbols. If an odd number of bytes are received from the front end, FELIX will add an extra padding byte. In the to-front end direction, the length must be an even number of bytes.
- Synchronization of 8b/10b encoding requires *two* consecutive comma characters.
- In 8b/10b encoded E-links, FELIX can be asked to assert BUSY by sending BUSY-ON and BUSY-OFF symbols (i.e. out-of-band symbols that can be sent any time, even within data packets). This should be done only in exceptional cases or at start of run. It should not be the normal mode of protecting against buffer overflow. Instead, complex dead time should be defined to prevent most buffer overflows.
- The event data sent to FELIX are not expected to be ATLAS-standard event fragments. FELIX just transports the data to the Software ROD where detector specific software may transform the data as required and format it into ATLAS-standard event fragments for the ATLAS Read out system (ROS).
- The use of a CRC or the IP checksum is recommended to detect any transmission errors for E-links run over cables.
- In addition to sending all events to the SW ROD, FELIX can send all, or a sample of, events to other network end points for monitoring. Extra monitoring data may be included as packets separate from event data packets in the E-link data stream by using FELIX's stream IDs at the start of the packet.
- Even if there is no hit data associated with a Level-1 Accept, it will still be necessary to send a packet to the SW ROD that contains at least the L1ID and BCID. Without this is will be almost impossible to properly recover from error conditions that may arise.
- DCS information may be included as packets separate from event data packets in an E-link data stream by using FELIX's stream IDs at the start of the packet.
- Any 80 Mb/s E-link can be used to connect to a GBT-SCA ASIC. The E-link clock must be configured to use 40 MHz, i.e. the data is sent in DDR mode.
- The "EC" link can be used as an ordinary E-link at 80 Mb/s; its E-link clock may be either 40 or 80 MHz.
- TTC: FELIX can send TTC Level-1 Accept information on any E-link declared as a 'TTC' E-link. 'TTC' E-links can be 80, 160 or 320 Mb/s E-links, to transfer, respectively, 2, 4 or 8 TTC bits on every BC clock. The contents of the TTC word is defined by the FELIX configuration and can be chosen from the ten bits in Table 3. Note: In all three cases, the E-link clock can be 40 MHz, i.e. the BC clock. The data is sent with FIXED latency.

Table 3. 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.

Brcst[7]	Brcst[6]	Brcst[5]	Brcst[4]	Brcst[3]	Brcst[2]	ECR	BCR	B-chan	L1A

- Network end-points, such as the SW ROD, can receive Level-1 Accept information (L1ID, BCID, Trigger Type, etc.) by subscribing to the Level-1 Info (virtual) E-link (also known as the TTC-to-host E-link). See Section 6.1.6.3.

9.4. FELIX Firmware Modules for Front-end Users

The FELIX team have produced a number of self-contained firmware modules which are intended for integration into front-end firmware both for testing and production purposes. These make it possible to test data transfer functionality from the output layer of the front-end firmware to FELIX and beyond, before integrating more of the front-end logic. Modules exist for both GBT and FULL mode use cases.

9.4.1. Downloading Firmware Source

A full description (including diagrams) of the modules discussed below, as well as the relevant firmware source, is available on the FELIX project distribution site:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/dist/examples/

The site contains multiple revisions, for compatibility with different FELIX firmware versions. GBT-compatible packages are labeled 'ElinkInterfaceSources' and FULL mode-compatible packages are labeled 'FullmodeInterfaceSources'. Please consult the documentation within the files for compatibility information.

9.4.2. GBT Test Modules

From a GBT perspective the modules provided depend on whether the GBT implementation is in an FPGA or with a GBTX chip. Common to both is a simple data generator module, which generates an incrementing counter and can be attached to the input port of the GBT module to provide a basic data source for link testing.

GBT-FPGA

For FPGA-based GBT a module will be provided to wrap and drive the GBT link in communication with FELIX (in both directions). All modules will be fully compatible with the official GBT-FPGA core[CERN_GBT_core].

GBTx

For GBTx chips all that is needed is to connect the provided data generator to a chip e-port, thus providing data on one E-link across the GBT.

9.4.3. FULL Mode Test Modules

Link Layer Tests

For FULL mode implementations the FELIX developers provide a link layer test package, making it possible to verify functionality at transceiver level (e.g. clock jitter stability, cleaning and configuration). Users should be able to integrate this into their front-end design for basic tests before implementing higher level link protocols.

Protocol Tests

Once the link layer is verified, users can integrate the 'stream controller' module, also provided by the FELIX developers, which manages the FULL mode link protocol and adds e.g. start and end of

packet markers. This is recommended for use not just in testing but also final implementation. Alongside this module a simple data generator is also provided which can be used for testing data transfer across the link.

9.4.4. E-link Wrapper

The E-link Wrapper is a standalone FPGA module that implements the E-link interface. It instantiates the Elink2FIFO and FIFO2Elink components alongside a reset sequence logic. The Elink2FIFO and FIFO2Elink contain FELIX's Central Router modules, which manage the TX/RX datarate (80, 160 or 320 Mbps) and the line code that is being used (8b10b or HDLC). These components provide a simple FIFO interface to the user that can write the data-to-be-sent into the FIFO2Elink buffer, and read the received data from the Elink2FIFO buffer. The wrapper can therefore be used as an interface between an FPGA and FELIX, via a GBTx or a GBT-FPGA instantiation [CERN_GBT_core], or as a means of communication between two FPGAs. More information on how to use the component can be found on [elinkWrapper_ug]. The source files themselves can be found in the repository which is included in the aforementioned user guide's References section.

9.5. External Software Resources and Tools

9.5.1. SCA eXtension — FPGA emulation of the SCA ASIC

A standard way to configure a front-end device is via the GBT Slow Control Adapter (GBT-SCA) ASIC [gbtmainpage]. The SCA ASIC implements several interfaces to communicate with other on-board devices, in order to configure them and monitor their status. The SCA eXtension (SCAX) FPGA firmware module [scax_ug] [scax_proc] emulates the SCA's communication protocol, in order to transparently provide access via FELIX to the registers of the FPGA logic in which it is instantiated. The use of SCAX enables DAQ and DCS to take advantage of the whole back-end OPC-UA software ecosystem to access FPGA registers. (See Appendix E.) SCAX also supports reading and writing FPGA memories and FIFOs. More information can be found in Appendix F.

9.5.2. IC-over-NetIO

The GBTx ASIC can be configured via a dedicated E-link, the IC E-link, provided that a bi-directional optical connection between that GBTx and FELIX exists. FELIX offers the ability to configure the GBTx via the low-level *fice* command, which accepts a file containing the values of the entire GBTx address space, and forwards them to the ASIC accordingly. The tool may also write values into specific registers, or read the values from some of the GBTx' registers.

The IC-over-NetIO application [icOverNetio_repo] is based on *fice's* logic, with the main difference being that it allows the user to perform the same operations via the NetIO interface, i.e. while the FELIX software is running, something that the *fice* application cannot do, being a low-level tool.

At the time of writing the application is in its testing phase, with some minor issues remaining to be addressed.

Appendix A: Setting up a TTC System for use with FELIX

This section is meant to help users of FELIX systems with the set-up of a TTC system. Both the new ALTI and legacy TTCvx/TTCvi systems are described below.

A.1. The ALTI System

The ATLAS Local Trigger Interface (ALTI) is an upgrade to the former TTC system, and replaces the functionality of the previous LTPi, LTP, TTCVi and TTCvx. The ALTI board provides functionalities such as rate counters for all TTC signals, per-bcid counters, busy monitoring, a pattern generation, and as monitoring/synchronization of input signals and programable phase shift of output signals. For a full list of functionalities and detailed ALTI instructions please see:

https://twiki.cern.ch/twiki/bin/viewauth/Atlas/LevelOneCentralTriggerALTI

The forward signals are ones sent by the CTP to the sub-systems (BC, ORB, L1A, TTR, BGO, TTYP), while the backward signals are sent by the subsystems to the CTP (BUSY, calibration). The front panel of the ALTI also has six SFP connectors for -mode fibre optic transmitter/receiver modules. The first five SFPs are used for dual-transmitters, while the last one can be used for TTC signal monitoring.

To test the FELIX with the ALTI card, connect system as shown in Figure 31. The cable from the LEMO connector of the FELIX timing card is plugged into the lower left connector labelled as BUSY OUT. In the future, it will be possible to configure the BUSY connector on the ATLI for either a NIM or TTL signal via software. Until this software feature is available in the ALTI, a NIM to TTL adapter can be used since the FELIX sends a signal compatible with a TTL signal, while the ALTI accepts a NIM signal by default.

To send TTC signals (L1A for example) from the ALTI to the FELIX, connect top SFP connector (using a single LC connector), to the ST connector on the FELIX timing card as shown in Figure 31. An LC to ST adapter will be needed for this.

The bottom SFP connector with the orange connections is connected in loopback mode for ALTI debugging and can be ignored for the purposes of FELIX commissioning.

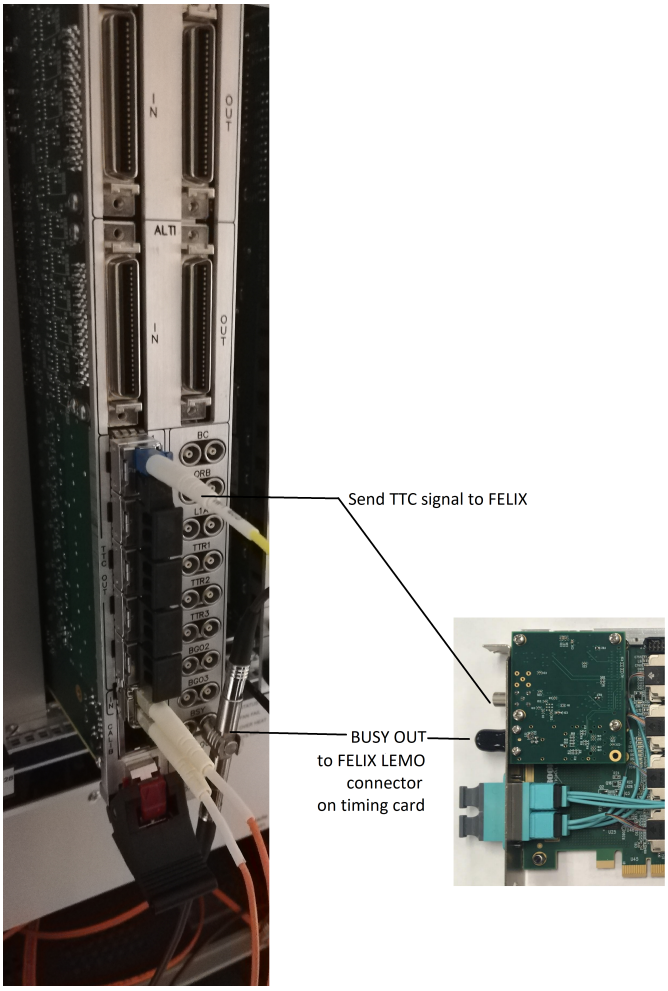

Figure 31. Image of cabled ALTI

A.1.1. Software Setup

Once the hardware is setup log into the SBC hosting the crate and setup the TDAQ and ALTI software either locally, in testbed, or at Point 1 following instructions at:

https://twiki.cern.ch/twiki/bin/viewauth/Atlas/LevelOneCentralTriggerALTI#Software

Once the TDAQ and ALTI environment is setup, configure the correct slot for the ALTI using the command:

```
testAltiInitial -s 6 -B -b 0x07000000 -R -S -c
```

where -s 6 is the ALTI slot number, -B -b 0x07000000 is to change the base address to 0x07000000 and -R -S -c is to reset and check the ALTI board.

A.1.2. Sending TTC Signals with ALTI

First make sure the FELIX is set to TTC clock following instructions in Section 3.5.1 and Section 3.5.2. After the software is setup you are ready to test sending L1A with the ATLI. You can use a configuration located here to send the L1A from the ATLI: /afs/cern.ch/atlas/project/tdaq/level1/ctp/l1ct-08-03-05/ALTI/data/cfg_1MHz.dat.

The file sends Level-1 Accepts (L1A) at a 1 MHz rate. You can either copy the file locally or read it directly from afs if you have access to it from your TTC crate. You can then start sending the L1A by

executing the command below from the SBC:

```
testAltiInitial -s 4 -R -S -C -f /afs/cern.ch/atlas/project/tdaq/level1/ctp/l1ct-08-03-05/ALTI/data/cfg_1MHz.dat
```

At this point the FELIX should forward the L1A to the front end, your front end should respond accordingly and send data through the FELIX, which you can monitor with FELIXcore.

If one would like to stop/start the pattern sending you can follow the commands below from the SBC, and selecting 7, followed by 3 (to enable L1A sending) or 4 (to disable L1A sending):

```
menuAltiModule
```

` 7 [PAT menu] Pattern generation memory`

` 3 enable pattern generation`

` 4 disable pattern generation`

If a different patter generation or a different frequency of pattern is required, it is possible to configure one following instructions on

https://twiki.cern.ch/twiki/bin/viewauth/Atlas/LevelOneCentralTriggerALTI#Scripts_to_generate_of_Pattern_G

A.1.3. Testing BUSY signal with ALTI

In order to test whether the ALTI is corresponding correctly to a BUSY from the FELIX, it is possible to force a BUSY in the FELIX using the commands below on the FELIX server:

```
flx-config set TTC_DEC_CTRL_BUSY_OUTPUT_INHIBIT=0 -d 2
```

```
flx-config set TTC_DEC_CTRL_MASTER_BUSY=1 -d 2
```

To remove the BUSY execute the commands below on the FELIX card:

```
flx-config set TTC_DEC_CTRL_BUSY_OUTPUT_INHIBIT=0 -d 2
```

```
flxcard $ ./flx-config set TTC_DEC_CTRL_MASTER_BUSY=0 -d 2
```

The ALTI should respond to the FELIX BUSY by stopping to send the L1A (or other generated) patterns. To test if the ALTI has stopped sending the patterns, the following can be executed on the SBC hosting the ALTI:

```
menuAltiModule
```

And select 11 CNT menu counters, and then select 1 to read counters. If selecting 1 to read counters several times does not make the counters go up, it means the ALTI has stopped sending the L1A (or other pattern), and thus correctly responded to the FELIX BUSY signal.

A.2. The TTCvi/TTCvx (A)

Figure 32 shows the final cabling of TTCvi and TTCvx modules for a TTC setup with B-channel. The

A-channel carries the Level-1 Accept; the B-channel carries BCR and the other TTC commands. The TTCvi-TTCvx pair should have already been tuned. If not, see Section A.2.1 below. Note: For a TTCex this may look different. A list of all the materials you will require to set up a TTC system is presented in Table 4.

Figure 32. Image of cabled TTC system with B-channel connections

Table 4. Materials needed to set up a TTC system

Item	Source	Remarks

VMEbus crate	Can be rented from the CERN Electronics Pool	Other crates may do as well
VMEbus master	We recommend a SBC from Concurrent Technologies (ATLAS standard). Support can be given for VP717, VP917 and VP-E24 (64 bit, compatible with TDAQ software release tdaq-05-05-00 and above).	TTCvi VMEbus card
Can be rented from the CERN Electronics Pool (but the Pool may be out of stock)	The TTCvi is no longer in production. Make sure the VME base address switches are set to match your software.	TTCvx VMEbus card or TTCex VMEbus card
TTCvx and TTCex can be rented from the CERN Electronics Pool (but the Pool may be out of stock)	The TTCvx/ex is no longer in production. The TTCvx has a LED driver, the TTCex has a laser driver	3 LEMO cables (1 or 0.5 ns)
		1 optical multi-mode (TTCvx) or single-mode (TTCex) fiber with ST connectors on both ends
	Max length of the fibre: TTCvx: 20 m; TTCex: 100 m	TTCoc & ATLAS (not clear who to ask; Maybe P. Farthouat) & TTC fan-out; needed if you have several FELIX
optical attenuator		Needed only for use with a TTCex without TTCoc. The optical attenuator has to be a single-mode attenuator of 3-20 dB and has to be connected directly to the TTCex output. The FTPDA-R155 should work with a TTCvx without attenuator. In case of a TTCex an attenuator of 3 dB is recommended for the FTPDA-R155. The FTPDA-R155 has a sensitivity of -31 dBm and saturates at +1 dBm.

If you need to tune the TTCvi-TTCvx pair, you need in addition:

- 2 LEMO cables (5-10 ns)
- 2 LEMO Y-adapters
- 2 LEMO-BNC adapters
- 2 50 Ohm terminators (Only required if your oscilloscope has no internal termination.)

A.2.1. Tuning a TTC system

If your TTCvi-TTCvx pair has not been tuned, follow the instructions in this section. Cable the TTC system as shown in Figure 33. Note: for a TTCex this may look different. For more information please consult the section "Tuning procedure 2" of the TTCvi manual (http://www.cern.ch/TTC/TTCviSpec.pdf).

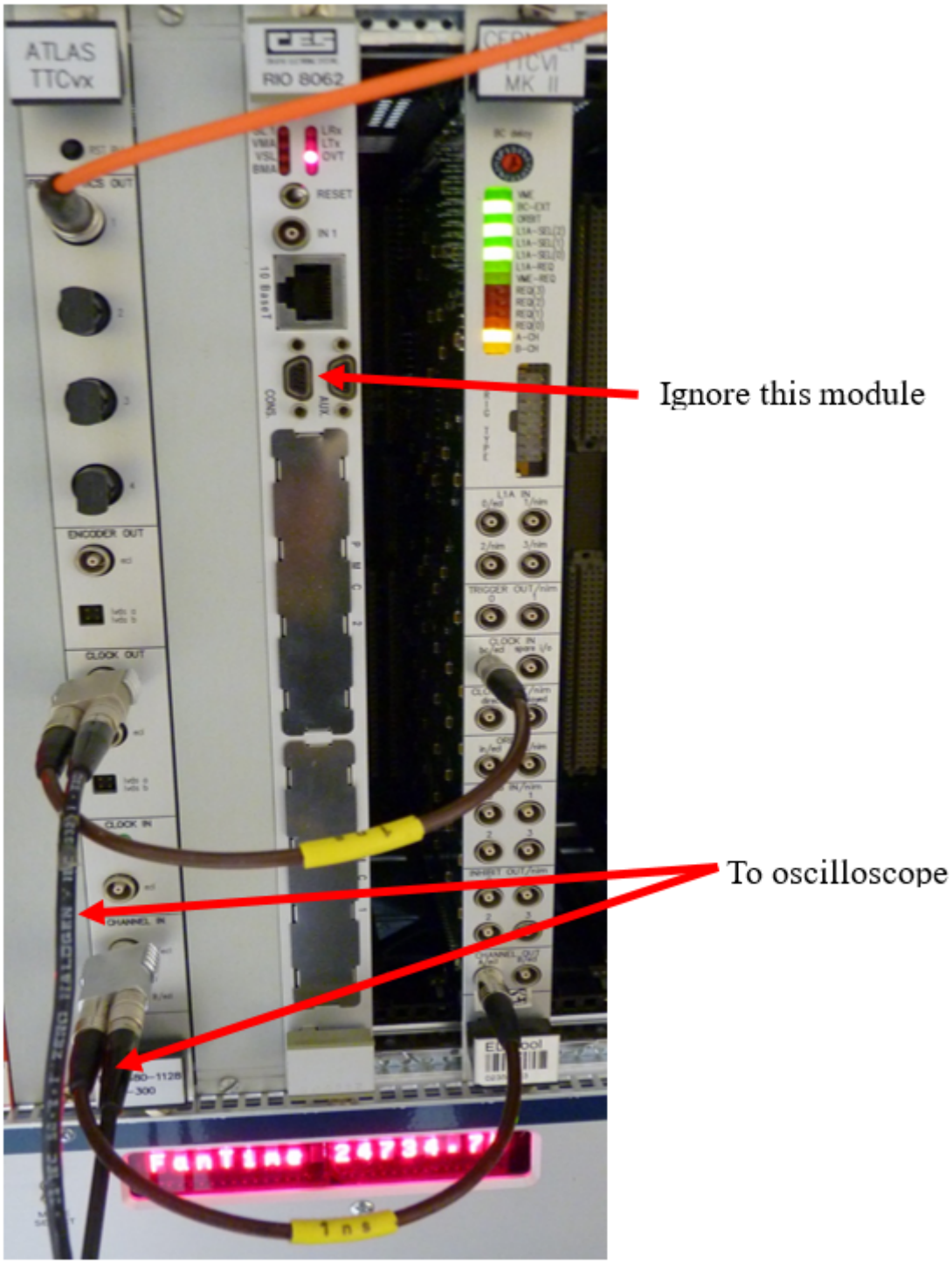

Figure 33. Image of cabling for tuning a TTC system

Note: The question has come up if channel A and channel B are correctly cabled in the picture above. Here is a reply from the TTC expert (Sophie Baron):

A "good" configuration when channel B is not used is indeed to have it tied to "1". And it is right that having Channel B connected to OUTPUT B gives a static "1" on channel B. However, the termination scheme at the TTCex inputs keeps as well unconnected channel inputs (both B and A) to "1" by default (it is negative ECL logic, and the Vin is at -2.08V by default). Therefore, both schemes could be used identically. One additional remark: of course, if you leave both A and B unconnected at the input of the TTCex, you will have both channels A and B to "1", and this is not good as the TTCrx needs to see two different behaviours on A and B to be able to differentiate them (the rule is that the A-channel must not have more than 11 consecutive "1" whereas B can have any type of

sequence).

This description can be broken down into the following points:

- Connect the TTCvi **A/ecl CHANNEL OUT** output to the TTCvx **A/ecl CHANNEL IN** input via a Y-adapter.

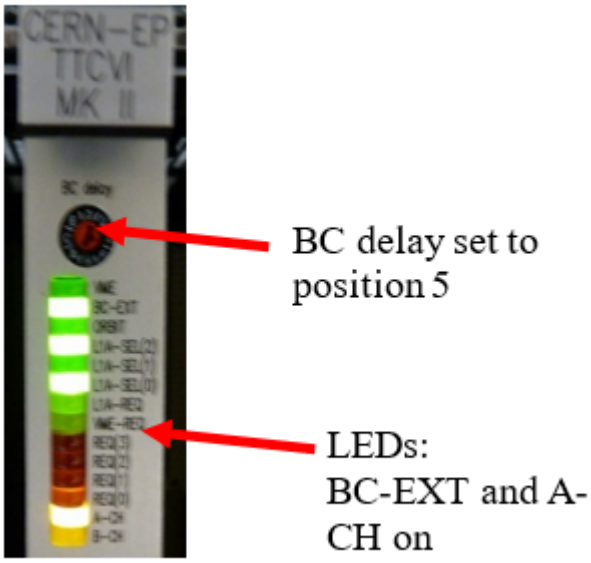

Figure 34. Image of cabling for tuning a TTC system

- Connect, via a Y-adapter, one of the **TTCvx CLOCK OUT/ecl** outputs to the **TTCvi CLOCK IN bc/ecl** input. Check that the BC_EXT indicator is lit on the TTCvi as shown below. The TTCvx internal clock may be used.
- Set the TTCvi trigger mode (= 5) to random at the highest rate (100 kHz) and disable the event/orbit/trigger-type transfers. In order to do this write 0x7005 to the D16 VMEbus CSR1 register at offset 0x80. This can be done easily for vme_rcc_test. Note: The A24 base address of the TTCvi in the CERN reference system in TBED is 0x555500. This should light up the TTCvi A-Ch yellow indicator and the A/ecl CHANNEL OUT output should now carry 25 ns long trigger pulses.
- With an oscilloscope look at the TTCvx Channel-A input in respect to the clock output, as shown below.

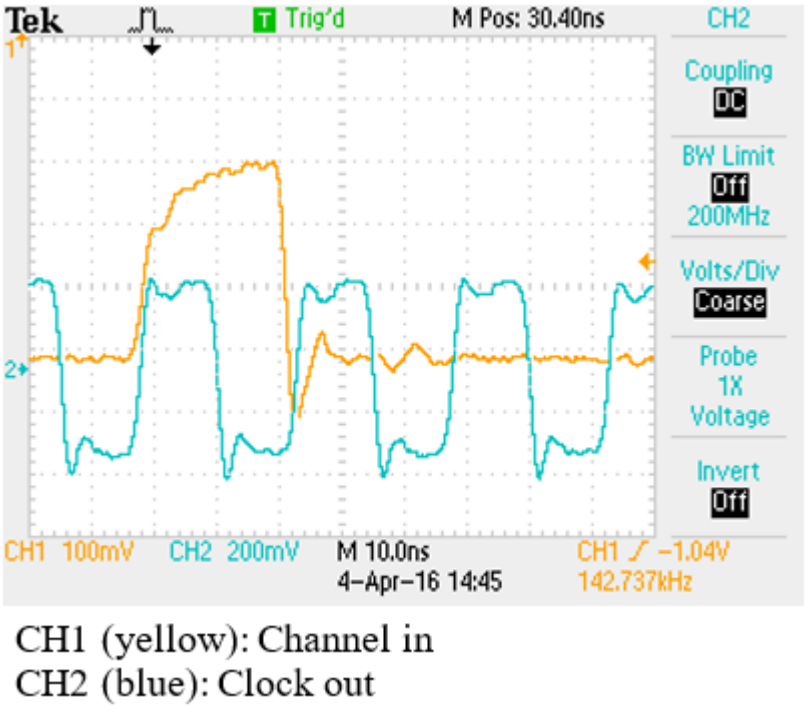

Figure 35. Image of cabling for tuning a TTC system

- Adjust the TTCvi BC delay switch such that the rising edges of the Channel-A pulses occur within 4 ns before to 2 ns after the rising edges of the clock signal.
- Setting the delay switch in position 2 and using 1 ns long interconnecting cables for the clock and the A and B channels corresponds to the above mentioned timing criteria. Note from Markus Joos: Even though I used 1 ns cables, I had to set the switch to position 5 (see picture above) in order to meet the requirement of step 5.

A.2.2. Guide to TTC Channel B

The following section describes the structure of the TTC 'B channel' data stream, and how it may be decoded and operated by users. The information in this section is provided courtesy of Alessandra Camplani and the LAr group.

The data stream arriving through TTC B channel can be of two types: short broadcast commands or long individually-addressed commands/data.

Short broadcast commands are used to deliver messages to all TTC destinations in the system, while long individually-addressed commands/data are used to transmit user-defined data and instructions over the network to specific addresses and sub-addresses. These two types of command have different dedicated frame formats, as shown in Figure 36:

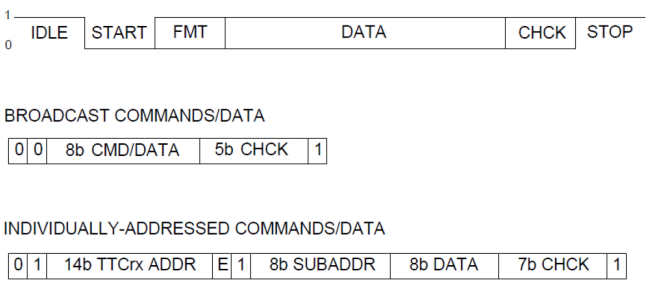

Figure 36. Image of cabling for tuning a TTC system

The difference between the two command types can be illustrated with the example below. When not in use the B channel IDLE state is set to 1. When a sequence of commands is sent, the data transmission state changes from 1 to 0. After the first zero received it is possible to distinguish between short broadcast and long address commands: if the second bit in the stream is a 0 then the command is a short broadcast, if it is a 1 then the command is of long address type.

```
IDLE=111111111111
    Short Broadcast, 15 bits:
    00TTDDDDEBHHHHH:
        T= test command, 2 bits
        D= Command/Data, 4 bits
        E= Event Counter Reset, 1 bit
        B= Bunch Counter Reset, 1 bit
        H= Hamming Code, 5 bits
    Long Addressed, 41 bits
    01AAAAAAAAAAAAAAE1SSSSSSSSDDDDDDDDHHHHHHH:
        A= TTCrx address, 14 bits
        E= internal(0)/External(1), 1 bit
        S= SubAddress, 8 bits
        D= Data, 8 bits
        H= Hamming Code, 7 bits
```

The short broadcast command type is used to send two important values: the Bunch Counter Reset (BCR) and the Event Counter Reset (ECR).

The BCR is used to reset the bunch crossing counter, which is increased every clock cycle on the 40 MHz clock. This is a 12-bit counter, also called BCID. A BCR command is sent roughly every 89 μs, corresponding to the time that a bunch needs to do an entire circuit of the LHC. During this time the BCID counter reach its maximum value, 3564 counts.

The ECR is used to increase the event reset counter. The periodicity of this reset is decided by each experiment, with ATLAS having it set to 5 seconds. The event reset counter combined with the L1A counter gives the Extended L1ID (EVID). This is a 32-bit value consisting the L1A counter in the lower 24 bits, and the event reset counter in the upper 8. Every time that an ECR is received the upper counter is increased by 1 and the lower part is reset to zero. Every time that a L1A is received the lower part is increased by 1.

BCID and EVID values are used as a label for the data accepted by the trigger.

The long address command type is used to transport another important value: the Trigger Type (TType). Each L1A transmission is followed, with variable latency, by an 8-bit TType word. This word is generated inside the LVL1 Central Trigger Processor (CTP) and distributed from the CTP to the TTCvi modules for each of the TTC zones in the experiment via the corresponding LTP modules.

The presence of a Trigger Type within long address commands is announced by a sub-address (8 bits) set to 0.

Table 5. Trigger type 8-bit word: Each bit represent the sub-detector which fired the trigger or the data type.

Sub-Trigger	physics	ALFA	FTK	LAr demonstrator	Muons	Calorimeter	ZeroBias	Random
Bit	7	6	5	4	3	2	1	0

As shown in Table 5, each bit has a specific role. In calibration mode, bits 0 to 2 can be used to distinguish between up to eight different possible types of calibration trigger within each sub-detector. Bits 3 to 6 are used to indicate which sub-detector or subsystem fired the trigger. Bit 7 represents physics trigger-mode when set to 1, and calibration mode when set to 0.

A.2.3. B channel decoding firmware

An effort is under way to provide a centrally maintained firmware module to decode TTC B-channel data. In the short term, users are advised to refer to a version produced for LAr front-ends by Alessandra Camplani. The module code can be found in gitlab:

https://gitlab.cern.ch/atlas-lar-ldpb-firmware/LATOME-ttc

The code itself is in the folder *code_ttc* and the files dedicated to TType decoding are: *Bchan_top.vhd*, *SMdecoding_cnt.vhd* and *TType_decoding*. The simulations for this specific part can be found in the *simulation* folder. Here there is a testbench for the *Bchan_top* entity and another one for *TType_decoding* entity.

Development of this module is ongoing, with the *latome_ttc* branch being actively maintained and kept up-to-date.

A.2.4. Channel B decoding software

In order to test channel decoding, it is recommended that users employ the menuRCDTtcvi application, provided as part of the ATLAS TDAQ software release. Within the application select 'BGO menu' and then option 13 'send asynchronous command'. From here it should be possible to select either a short of long command. In the case of a short command simply enter the data word to be sent. For a long command enter an address 0 (for broadcast), 0 for internal registers, subaddress 0 for trigger type, and the data word to be sent.

A.2.5. Useful documents

You may find additional useful information in this document from the ATLAS LAr group:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/community/CPPM_MiniFELIX_tests_results_and_TTC_system_experience.pdf

Appendix B: BNL-712 Technical Information

This appendix will collect technical information for the BNL-712 board.

B.1. Overall Design

The FELIX card hosts 4 MiniPOD transmitters and 4 MiniPOD receivers. Each MiniPOD has 12 channels. The TTC clock from the ADN2814 is cleaned by an Si5345 or LMK03200. The clean 240 MHz is used as a reference clock for the GTH transceivers. Two of the PCIe hardcore EndPoints within the FPGA are used, with the PEX8732 PCIe switch used to connect them to a 16-lane slot.

An on-board 2 Gb FLASH memory can store 4 different firmware bit files. An on-board microcontroller (which the host can communicate with either via SMBus or through the FPGA and PCIe interface) can be used to select one of the four FLASH memory partitions and trigger FPGA programming from the image stored in the selected partition.

As shown in Figure 37 and Figure 38, the FPGA has two Super Logic Regions (SLRs), referred to as devices in the software. To balance resource usage and 54 minimize the number of traces crossing the boundary each SLR has one 8-lane PCIe endpoint. For the 24-ch GBT firmware flavor, banks 126-128 and 131-133 are used.

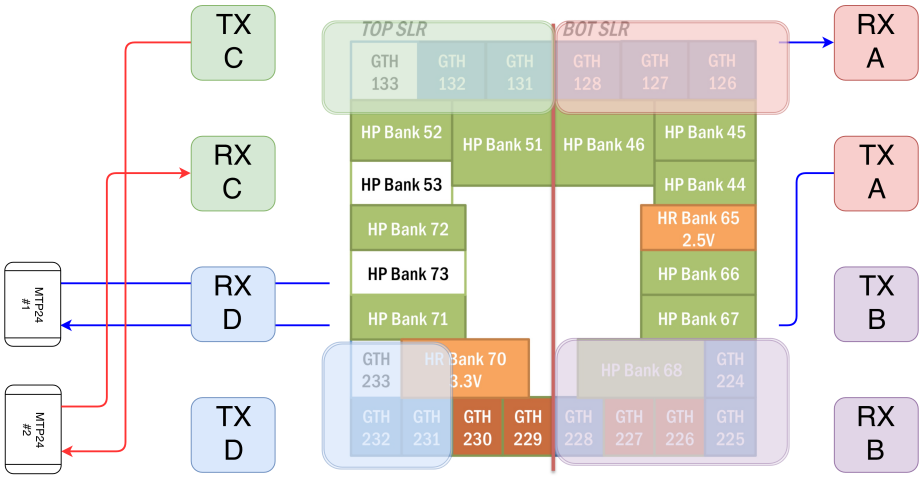

Figure 37. 24ch configuration.

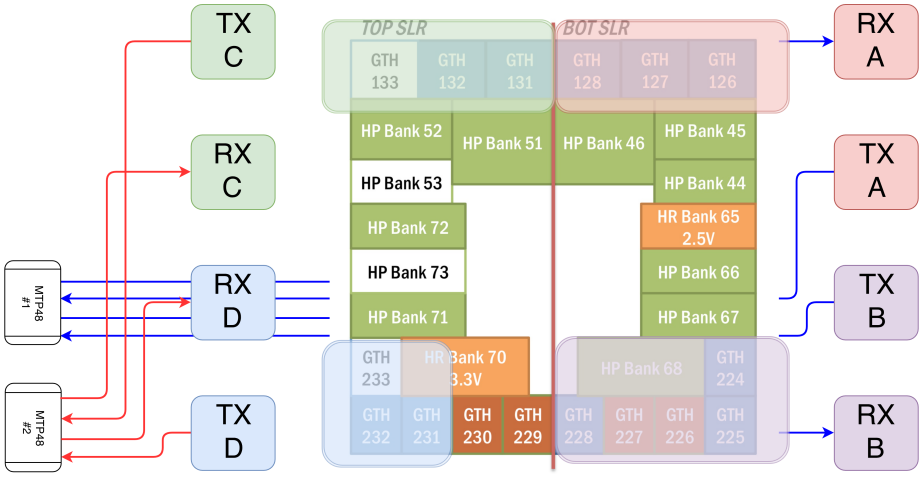

Figure 38. 48ch configuration.

B.2. Fibre Mapping and Connectivity

Every FLX-712 comes with PRIZM patches pre-installed, connecting the MTP inputs to the MiniPODs. These patches are custom made for the card and mean that the users should only need to connect to their data source to the MTP cable. No internal cabling work on the FLX-712 is required. The Fibre mapping and pin assignment for channels in each MiniPODs are shown described in this section. Two configurations are shown: 48 channel and 24 channel.

B.2.1. 24 Channel Version

For 24 channel version, the channel number is 12 for each MTP coupler. Figure 39 / Figure 40 show how the banks are connected to the MiniPODs in this case.

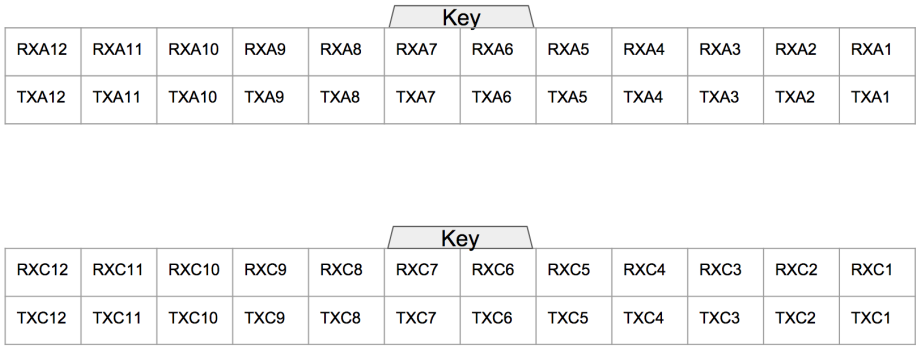

Figure 39. 24ch fibre mapping.

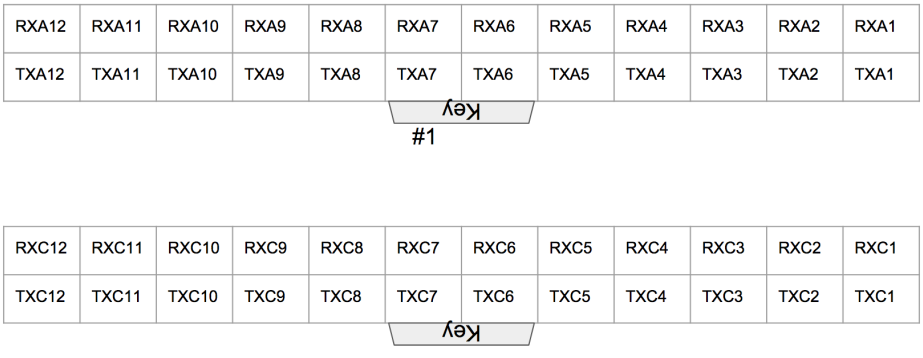

Figure 40. 24ch fibre mapping looking from MTP coupler.

B.2.2. 48 Channel Version

For 48 channel version, there are 24-TX and 24-RX in fibre connected to each MTP coupler. Figure 41 / Figure 42 show how the banks are connected to the MiniPODs in this case.

Key

RXA12	RXA11	RXA10	RXA9	RXA8	RXA7	RXA6	RXA5	RXA4	RXA3	RXA2	RXA1
TXA12	TXA11	TXA10	TXA9	TXA8	TXA7	TXA6	TXA5	TXA4	TXA3	TXA2	TXA1
RXB12	RXB11	RXB10	RXB9	RXB8	RXB7	RXB6	RXB5	RXB4	RXB3	RXB2	RXB1
TXB12	TXB11	TXB10	TXB9	TXB8	TXB7	TXB6	TXB5	TXB4	TXB3	TXB2	TXB1

Key

RXC12	RXC11	RXC10	RXC9	RXC8	RXC7	RXC6	RXC5	RXC4	RXC3	RXC2	RXC1
TXC12	TXC11	TXC10	TXC9	TXC8	TXC7	TXC6	TXC5	TXC4	TXC3	TXC2	TXC1
RXD12	RXD11	RXD10	RXD9	RXD8	RXD7	RXD6	RXD5	RXD4	RXD3	RXD2	RXD1
TXD12	TXD11	TXD10	TXD9	TXD8	TXD7	TXD6	TXD5	TXD4	TXD3	TXD2	TXD1

Figure 41. 48ch fiber mappping.

RXA12	RXA11	RXA10	RXA9	RXA8	RXA7	RXA6	RXA5	RXA4	RXA3	RXA2	RXA1
TXA12	TXA11	TXA10	TXA9	TXA8	TXA7	TXA6	TXA5	TXA4	TXA3	TXA2	TXA1
RXB12	RXB11	RXB10	RXB9	RXB8	RXB7	RXB6	RXB5	RXB4	RXB3	RXB2	RXB1
TXB12	TXB11	TXB10	TXB9	TXB8	TXB7	TXB6	TXB5	TXB4	TXB3	TXB2	TXB1

Key

RXC12	RXC11	RXC10	RXC9	RXC8	RXC7	RXC6	RXC5	RXC4	RXC3	RXC2	RXC1
TXC12	TXC11	TXC10	TXC9	TXC8	TXC7	TXC6	TXC5	TXC4	TXC3	TXC2	TXC1
RXD12	RXD11	RXD10	RXD9	RXD8	RXD7	RXD6	RXD5	RXD4	RXD3	RXD2	RXD1
TXD12	TXD11	TXD10	TXD9	TXD8	TXD7	TXD6	TXD5	TXD4	TXD3	TXD2	TXD1

Key

Figure 42. 48ch fibre mapping looking from MTP coupler.

Appendix C: BNL-711 Technical Information

This appendix will collect technical information for the BNL-711 board which may be relevant to user test stand installations.

The BNL-711 hosts a Xilinx® Kintex® UltraScale FPGA on a board capable of supporting 48 high speed optical links via MiniPOD transceivers, with a 16-lane PCIe Gen 3.0 interface. On-board clock jitter cleaning and TTC circuitry mean that no mezzanine attachment is required to connect with ATLAS clock and control systems. An image of the BNL-711 and its key features are presented below. While the board is still in use in a number of test stands, it is recommended that subdetector teams adopt the BNL-712 for any new developments.

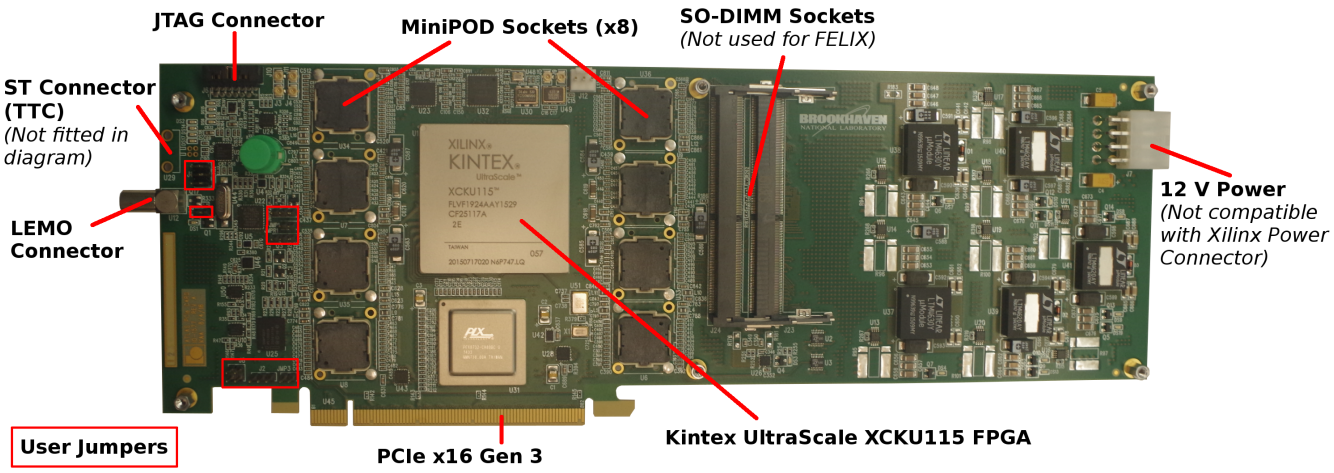

Figure 43. The BNL-711 V1.5 board.

C.1. User Jumper Map and Functional Specification

The BNL-711 provides a number of I/O connectivity options. These are can be selected by modifying the position of the user jumpers shown by the red boxes in Figure 43. A specific map of the relative position and name of each jumper is presented in Figure 44. A detailed description of the function of each jumper, and to which board configuration options it relates, is available in the sections below.

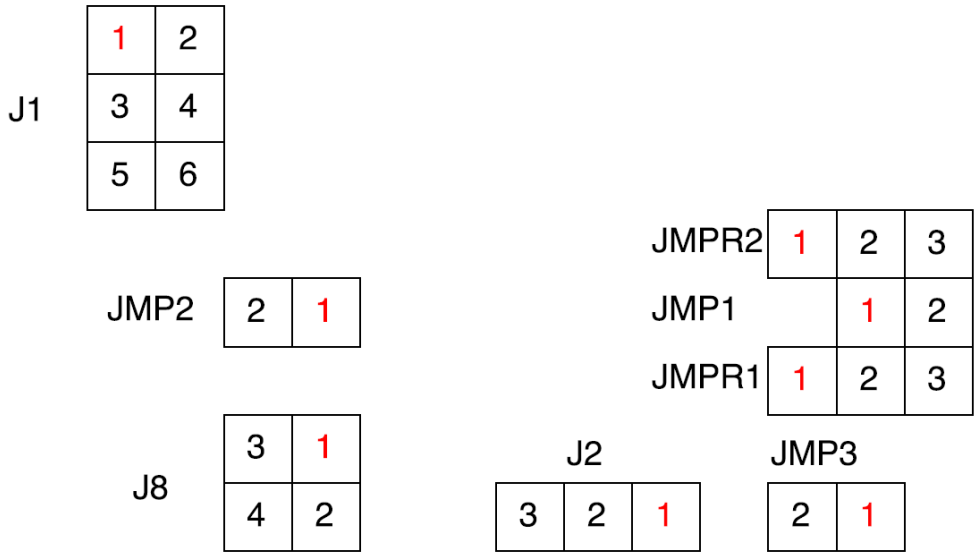

Figure 44. BNL-711 V1.5 User Jumper Map

C.1.1. J1

For uC configuration with 6-pin ISP programmer.

1	MISO
2	VTG-SYS25
3	SCK
4	MOSI
5	RSTn
6	GND

C.1.2. J2

PRSNT selection.

1	PRSNT_FPGA
2	PRSNT
3	PRSNT1

2 & 3 are connected.

C.1.3. J8

Backup I2C/SMB connector.

1	SYS33
2	PCIE_SCL

3	GND
4	PCIE_SDA

C.1.4. JMP1

Connect FPGA_PROG_B to the uC_FPGA_PROG_B (PC5). Connected by default.

1	uC_FPGA_PROG_B
2	FPGA_PROG_B

C.1.5. JMP2

Connect FPGA_INIT_B to the uC_FPGA_INIT_B (PD2). Connected by default.

1	uC_FPGA_INIT_B
2	FPGA_INIT_B

C.1.6. JMP3

WAKE_N from PCIe to FPGA. NOT connected by default.

1	PCIE_WAKE_N
2	PCIE_WAKE_N_FPGA

C.1.7. JMPR1 & JMPR2

JMPR1: FLASH_A25 selection.

1	GND
2	uc_FLASH_A25
3	SYS25

JMPR2: FLASH_A26 selection.

1	GND
2	uc_FLASH_A26

3	SYS25

The FPGA firmware has the highest priority to set FLASH_A25 and FLASH_A26. The Jumpers have lowest priority.

If uC is used, when uC_FLASH_A is '1', the FLASH_A will be '0'; when uC_FLASH_A is '0', the FLASH_A will be '1'.

If Jumpers are used, when it is connected '1', the FLASH_A will be '0'; when it is connected '0', the FLASH_A will be '1'.

As default, the first Flash partition can be used, then 2 & 3 are connected.

C.2. MiniPOD Connectivity Map

The BNL-711 hosts 4 MiniPOD Tx/Rx Transceiver pairs, located in two banks either side of the FPGA, as shown in Figure 43. The transceiver sockets have a specific logical order, presented in Figure 45. Users should connect their optics according to this map to ensure the correct link order is preserved.

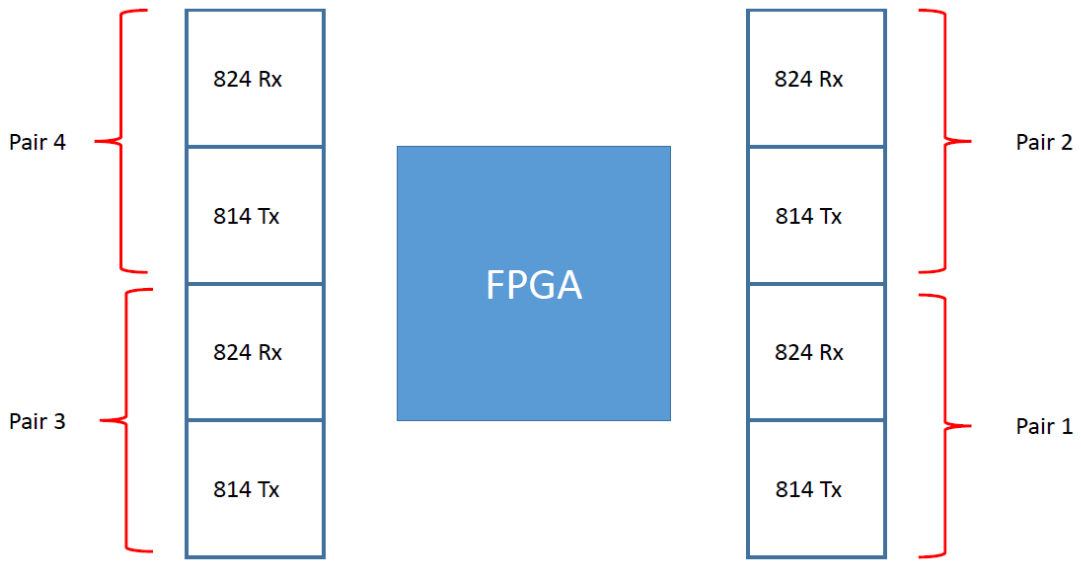

Figure 45. BNL-711 MiniPOD Connectivity Map.

Appendix D: Guide to FELIX Data Structures

- Data buffered in the FPGA per E-link or per FULL mode link and transferred under DMA control
- Fixed block size of 1 kB
- The blocks are transferred into a contiguous area, functioning as a circular buffer, in the main memory of the PC.
- The DMA runs continuously, thereby eliminating DMA setup overheads and achieving high throughput (about 12 GB/s for the 16-lane interface of the FLX-711).
- Event fragments or other types of data arriving via the FE links are referred to as "chunks" and can have an arbitrary size.
- 1 kB blocks of E-links or FULL mode links are multiplexed into a single stream.

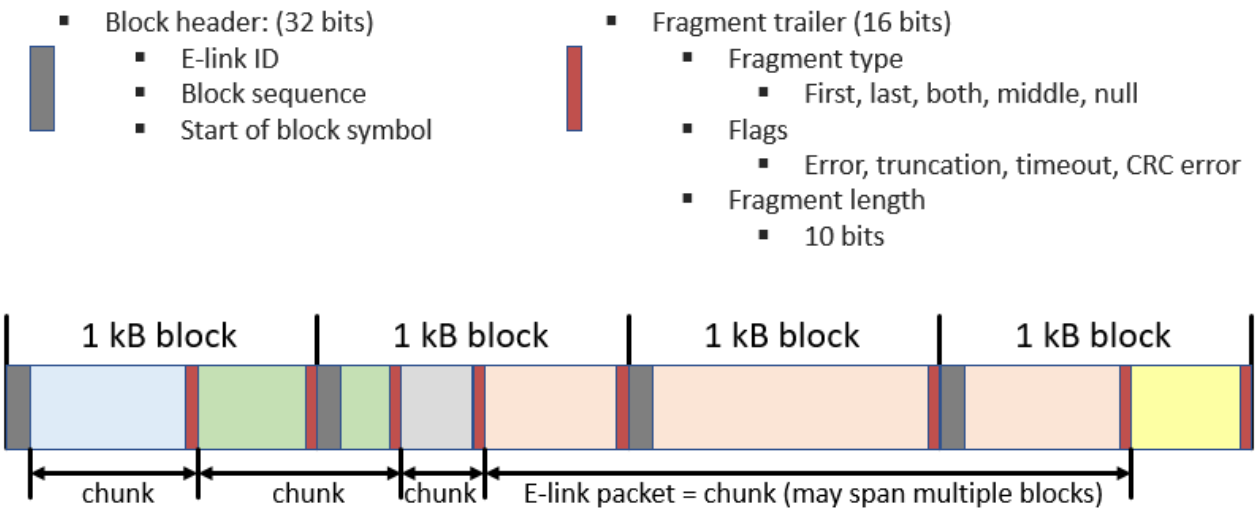

Figure 46. FELIX block structure

V1.5 1-May-20

From FPGA to Host

Block header

Data is transferred as 1 kByte blocks with this block header

Block header as *uint32_t* (*u_long*) read from memory

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xABCD (0xABCE for the 32-bit trailer format)																Sequence number								GBT						E-group			E-path		

Block header as *u_char* (bytes) read from memory

7	6	5	4	3	2	1	0
GBT-low		E-group			E-path		

7	6	5	4	3	2	1	0
Sequence number						GBT-high	

7	6	5	4	3	2	1	0
0xCD							

7	6	5	4	3	2	1	0
0xAB							

Block header as *uint16_t* (u_short) read from memory

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sequence number					GBT				E-group			E-path			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xAB								0xCD							

Sub-chunk trailer (16-bit trailer)

Payloads of the blocks are organized as chunks that may extend over several blocks. Each chunk has a trailer, if it extends over more than one block then each part of the chunk ("sub-chunk") has its own trailer

Sub-chunk trailer read as *uint16_t* (u_short) from memory

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Type			T	E	C	Sub-chunk length in bytes									

Type			
0	0	0	← signals fill pattern caused by timeout, all other bits are also 0
0	0	1	← first part of chunk consisting of more than one part
0	1	0	← last part of chunk consisting of more than one part
0	1	1	← chunk consisting of one part
1	0	0	← not first and not last part of chunk consisting of more than one part
1	0	1	← E-link timeout
1	1	0	← reserved
1	1	1	← out-of-band

on a 16-bit boundary, if the number of bytes is odd, a padding byte is added at the end

T: Truncation flag, 1 if chunk was truncated
E: Error flag, 1 if an error occurred
C: CRC20 Error in FULL mode, reserved in GBT mode

Out-of-band trailer (implemented in Full mode only)

Out of band trailers start with "111", but the other fields are defined here, the fields T, E, C and length are undefined.

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

(0xE05C) Indicates that the frontend is in busy state for the given Full mode link

JumboBlock' Sub-chunk trailer (32-bit trailer)

(text next to 'Sub-chunk trailer' applies)

Sub-chunk trailer read as *uint32_t* (32-bit word) from memory

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Type			T	E	C	B	reserved (9 bits)									Sub-chunk length in bytes																

Type			
0	0	0	← signals fill pattern caused by timeout, all other bits are also 0
0	0	1	← first part of chunk consisting of more than one part
0	1	0	← last part of chunk consisting of more than one part
0	1	1	← chunk consisting of one part
1	0	0	← not first and not last part of chunk consisting of more than two parts
1	0	1	← E-link timeout
1	1	0	← reserved
1	1	1	← out-of-band

T: Truncation flag, 1 if chunk was truncated
E: Error flag, 1 if an error occurred
C: CRC20 Error (FULL mode) Reserved in GBT mode
B: Busy state indication (1 means FE is busy)

Default emulator chunk payload

Internal header of chunk payload as loaded in emulator memory in .bit files. This payload can be replaced by the elinkconfig program

Internal chunk header as *u_char* (bytes) read from memory

byte 0

7	6	5	4	3	2	1	0
0xAA							

byte 1

7	6	5	4	3	2	1	0
length (msb)							

byte 2

7	6	5	4	3	2	1	0
length (lsb)							

byte 3

7	6	5	4	3	2	1	0
counter ('L1ID')							

msb: most significant bits
lsb: least significant bits

byte 4

7	6	5	4	3	2	1	0
0	0	E-group			E-path		

byte 5

7	6	5	4	3	2	1	0
0xBB							

byte 6

7	6	5	4	3	2	1	0
0xAA							

byte 7

7	6	5	4	3	2	1	0
E-link width (2, 4, 8 or 16)							

TTC: TTC-to-Host packet

Proposal for TTC information ("TTCtoHost") output via dedicated E-link as chunk payload, Phase-II compatible, FMT=0x1: 20-byte format

Words as *u_long* (32-bit) read from memory

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
reserved				BCID												length (26, or 20)								Format version (FMT)								Word 0
extended L1ID								L1ID																								Word 1
orbit																																Word 2
reserved																trigger type																Word 3
L0ID (currently equal to L1ID + extended L1ID)																																Word 4

Full: indication of overflow in TTCtoHost fifo, before truncation is issued

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
L1A counter (32 LSBs)																																Word 5
																Full								L1A counter (15 MSBs)								Word 6

Proposal for TTC information for testing, emulated using emulator memory

Words read as *u_long* (32-bit word) read from memory

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Format version								length (8)								reserved								BCID							

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
extended L1ID								L1ID																							

From Host to FPGA

Data is transferred as 32 byte messages with a 2-byte header and a 30-byte payload

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GBT					E-Group			E-path				length			eom

eom: 1 if last part of message
length: count of 2 byte words

Figure 47. List of FELIX Data Structures

Table 6 is a table version of E-links that produce a 26-byte *L1AInfo* data packet containing information about each Level-1 Accept.

Table 6. The TTC Level-1 Accept information 26-byte data packet sent to the host system, shown here as seven 32-bit words. Byte address space is from right to left, i.e. the words are little-endian.

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

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

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

Appendix E: Guide to Using FELIX with the GBT-SCA

This appendix is included thanks to Paris Moschovakos and the DCS team.

E.1. Introduction

The Slow Control Adapter ASIC (GBT-SCA, or SCA for short) is part of the GBT chip-set and is dedicated to the slow control of the front-end boards. It features several sub-devices that facilitate both front-end configuration and monitoring of environmental variables (voltages, temperatures, etc.) on and around the detector. The SCA contains an ADC, DACs, general purpose IO, and controllers for I2C, SPI and JTAG. An SCA is connected to a GBTX via any 2-bit E-link in 40 MHz DDR mode (80Mb/s) with HDLC encoding. Up to 41 SCAs can be potentially connected to a single GBTX with the corresponding link on FELIX configured accordingly.

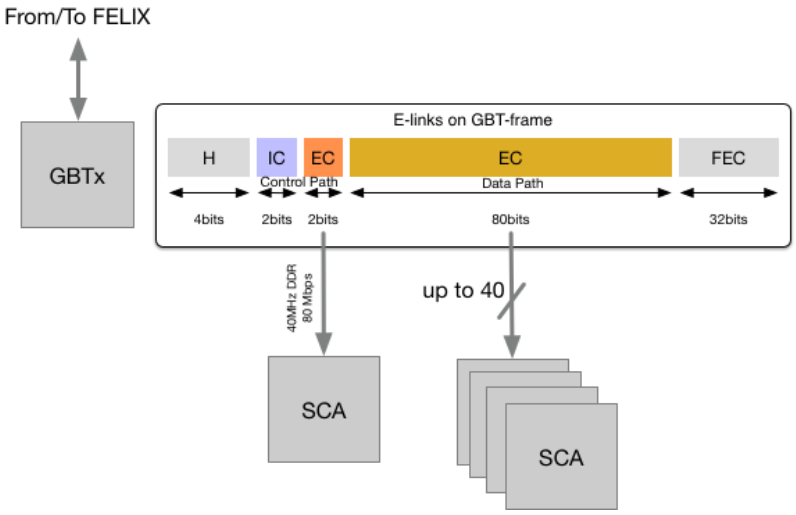

Figure 48. GBT frame paths and E-links.

E.2. Typical test setup

A typical test setup consists of a board with a GBTX that is connected to FELIX via an optical fibre and to an SCA via one of the GBTX's E-links.

The Versatile Link Demo Board (VLDB) (https://espace.cern.ch/GBT-Project/VLDB/default.aspx) contains both a GBTX and an SCA. It can be directly connected to a FELIX card. (The VLDB demo board can be procured from the GBT group). A schematic of such a setup is shown in Figure 49.

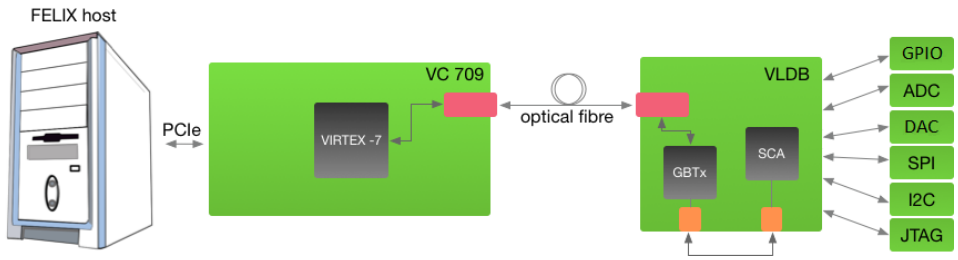

Figure 49. Evaluation setup with a GBT-SCA on a VLDB. SCA and GBTX are interconnected on the VLDB externally via a pair of mini-HDMI connectors (J32 (PRIMARY) and J33 (SCA PORT), in that case using the GBT EC E-link).

To simplify the evaluation of the setup, the VLDB possesses two LEDs connected to two general-purpose digital outputs of the SCA. This can be used to quickly validate visually the communication path and functionality all the way from the FELIX host to the SCA itself. In order to do that, the 'fec' tool, of the *ftools* family as mentioned in Section 6, can be used.

The following command instructs the SCA connected to the VLDB with the GBT link <*gbt_link_number*> EC link, to blink one of its LEDs 50 times at a rate of 5Hz (*-t 100*: 100ms on, 100ms off; last character is an 'o' for 'output'; *-x* selects the digital output: on the VLDB there's an LED on 18 and one on 21):

```
fec -G <gbt_link_number> -r 50 -t 100 -x 18 o
```

E.3. Procedure to set up an E-link to a GBT-SCA

A configuration procedure is needed both for FELIX and the GBTX itself. The configuration is mostly a description of the setup at hand and the mapping of the e-links that are connected. There are also some setup specific parameters to be configured.

In the case the SCA is connected to the dedicated EC E-link, one should check that this E-link is enabled using the *elinkconfig* GUI. That specific E-link is pre-configured with the appropriate HDLC SCA encoding and corresponding bit endianness and can be used directly for an SCA. Figure 50 shows an *elinkconfig* screenshot with the enabled EC channel in both the to-host and from-host direction, indicated by the checked tick boxes with yellow background labeled 'EC'; the hexadecimal number in brackets is the FELIX E-link ID associated with this E-link. Note that we happen to have selected GBT link #2, which is also reflected in the hexadecimal numbers next to the various E-link 'enable' tickboxes.

In the case that one wants to use an E-link from one of the GBT E-groups instead of or in addition to the EC E-link, one has to configure FELIX accordingly via *elinkconfig*. The SCA uses HDLC encoding instead of the typical 8b/10b which is the default for the data E-links, so this needs to be configured for the E-link. Moreover, the bit orientation is different from the 'normal' data E-links. By selecting the HDLC format in the drop-down menu, elinkconfig takes care both of the orientation and the encoding, indicating that an SCA is connected to that specific GBT group and path. As an example, Figure 50 shows that the 8th 2-bit E-link of E-group 0 of GBT link 2 has been enabled in the FELIX configuration in both the to-host and from-host direction. The FELIX E-link ID is shown here to have a value of 0x087. Using *ftools* tool felink one can confirm this is indeed the requested E-link:

```
> felink -e 87
E-link 087 = GBT #2 group #0 path #7, bit#14 width=2
```

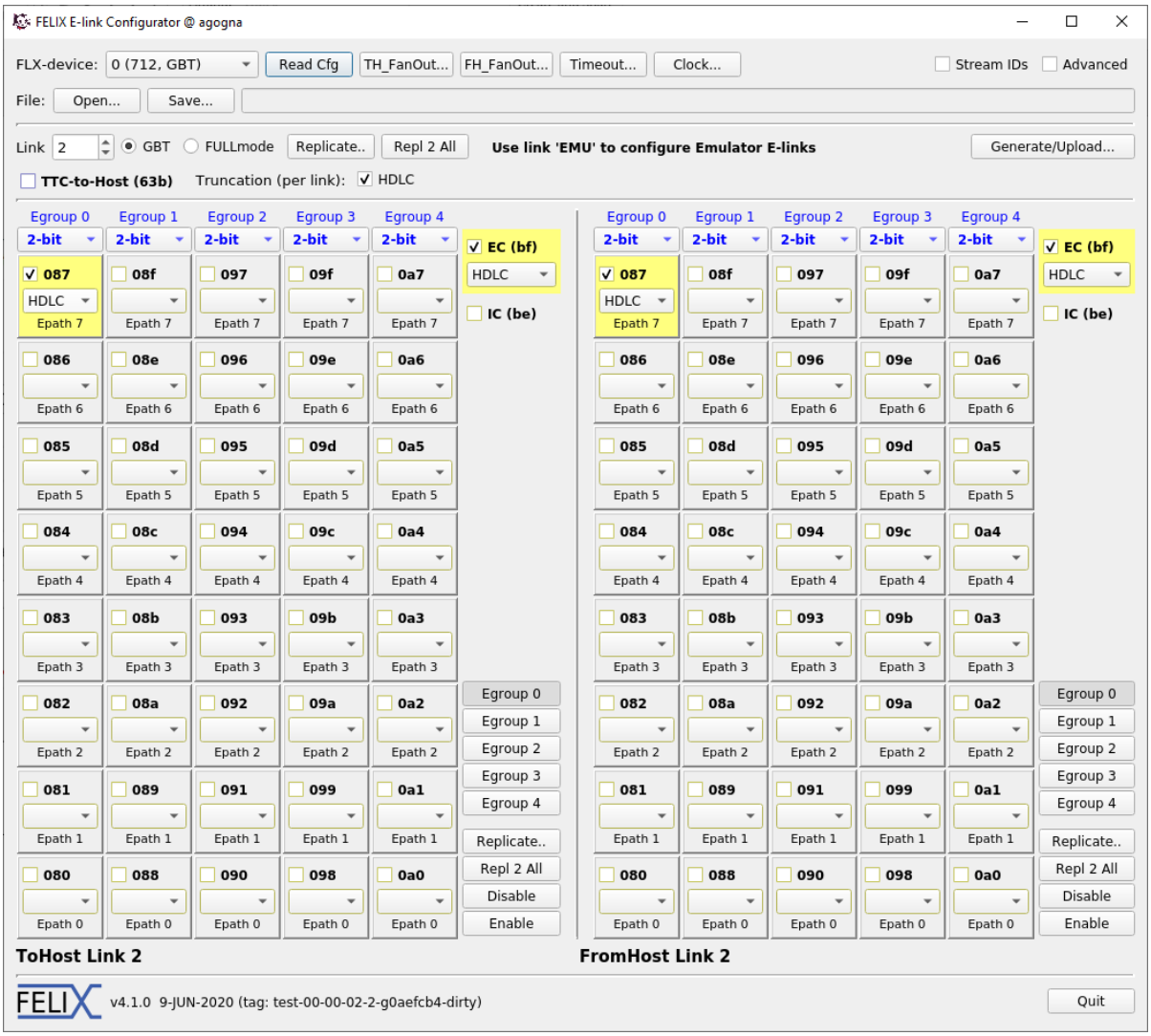

Figure 50. Enabling E-links connected to GBT-SCAs.

E.4. Low level operations with the *fec* tool

The fec tool, is a dedicated tool from the *ftools* suite of tools to demo the handling of a number of I/O channels available on the SCA. Please check the full list of possible operations in Section 6.

E.5. A Software Suite for the Radiation Tolerant GBT-SCA - The Production system

The on-detector DCS system that handles the slow control traffic and the configuration of the front-end electronics, based on the GBT-SCA. The global scheme is presented in the following figure.

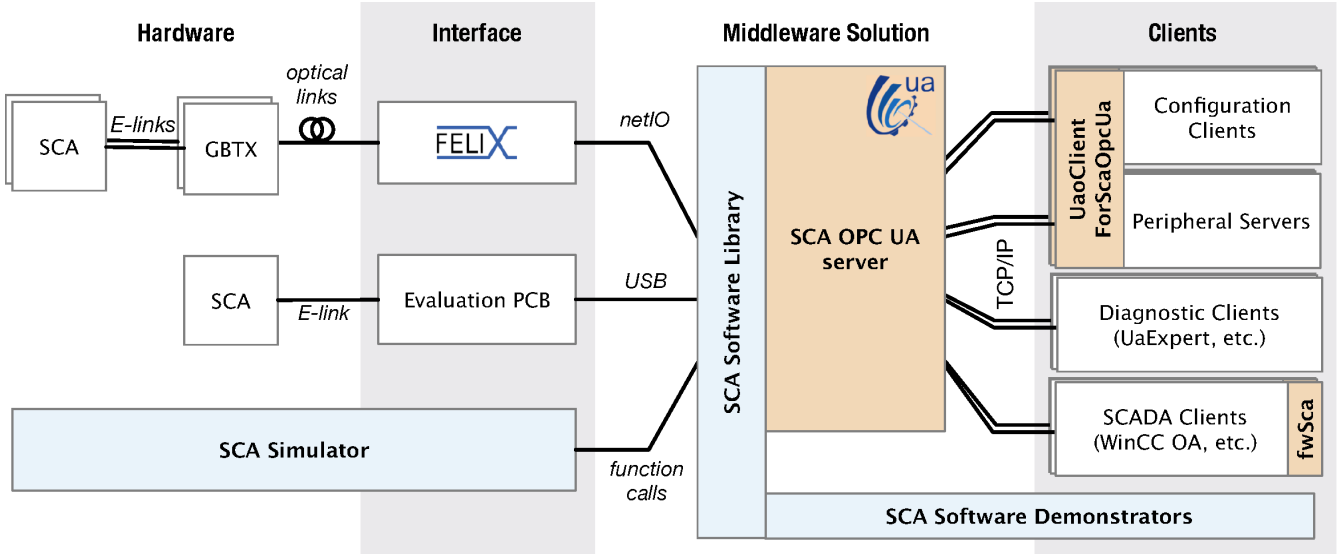

Figure 51. Global picture of the software suite. The SCA Software package, in light blue, comprises the SCA Software API to communicate with the SCA via different back-ends, the SCA Simulator to emulate SCA traffic for testing and development, and the Demonstrator tools which are used for standalone operations. The SCA OPC UA server and its ecosystem, in orange, is the middleware of choice to exchange data with the front-ends. UaoClientForScaOpcUa is a library that clients use to communicate with the SCA server. Finally, the fwSca module automatizes the integration of the server data into SCADA systems.

The slow control and configuration traffic, unlike physics data, has different requirements in terms of throughput, latency, availability and reliability. SCA DCS is the software that handles the SCA traffic arriving on the FELIX card. Towards the FELIX clients it is based on the middleware Open Platform Communications Unified Architecture (OPC UA, of the OPC foundation, (https://opcfoundation.org) which is an industry standard for secure and reliable exchange of data in industrial automation and other controls-related areas.

The server/client architecture that the platform uses, allows for different purpose clients to be served by a single server per FELIX host. The data flow to/from the ATLAS control room, not only serves the control and monitoring data of the detectors' conditions but also implements the configuration path of the on-detectors electronics and their initialization for data taking or calibration. In addition, system experts can monitor the status of the employed technology and get statistics and other information in order to diagnose the various system layers.

All those requirements potentially imply many different OPC UA clients that would like to receive SCA data from the setup at the same time. The chosen OPC UA architecture ensures the reliable and seamless data delivery and the compatible integration into the current DCS systems. This means that OPC UA clients in both DCS and a detector configuration server can communicate with the same SCA and the OPC UA server arbitrates their access.

E.5.1. OpcUaSca server

The provided OPC UA server implementation for the SCA is based on the ScaSoftware (explained below) intending to profit from all features of the ScaSoftware library and providing a high-level and user-friendly OPC UA address space to OPC UA clients.

The OpcUaSca server has been designed and implemented using the quasar framework (see https://github.com/quasar-team/quasar). Its design is presented in the following figure.

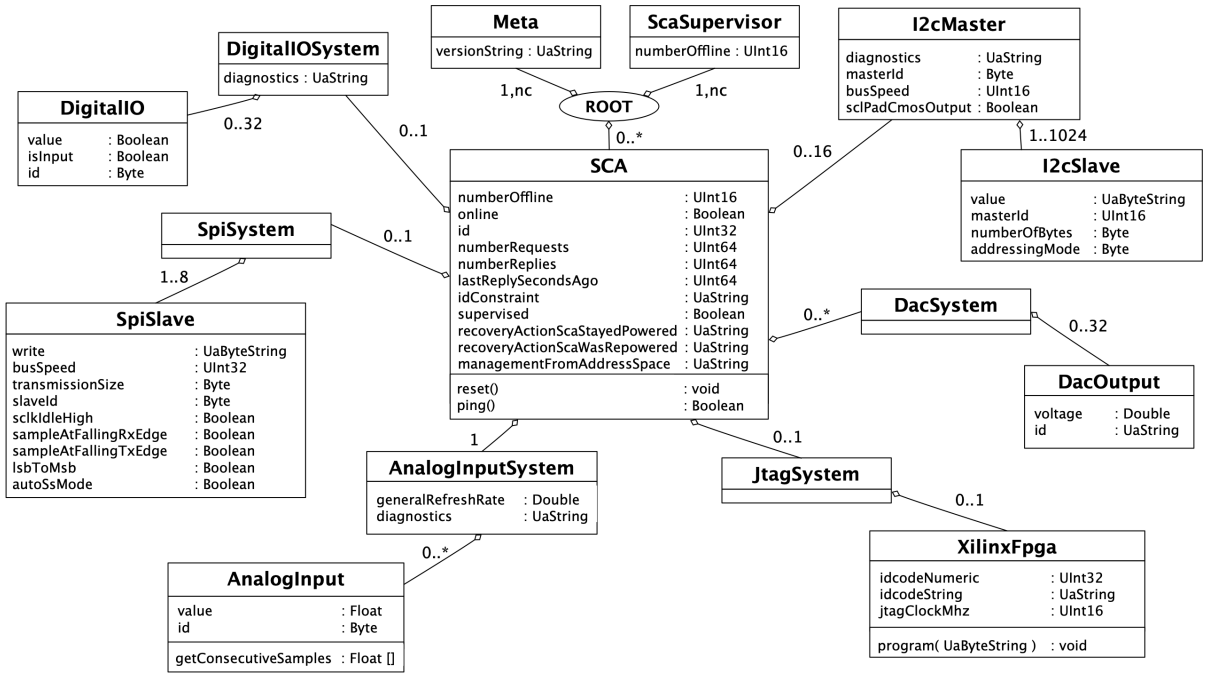

Figure 52. The quasar design diagram of the OPC UA server for the SCA.

As of May 2020, the server supports the following functionality:

- Communication with any number of SCAs, through NetIO or any other HDLC backend. Each SCA is identified in its address-space by a name, and its unique 24-bit "SCA identifier" which is written by the chip manufacturer in the SCA silicon. The identifier is read using the ScaSoftware library when a connection to given SCA is opened.
- Up to 32 ADC channels per SCA which are polled with the configured conversion frequency. Note that channel 32 has no external connection; it is connected to the on-chip temperature sensor that monitors the SCA temperature.
- Up to 32 General Purpose I/O pins per SCA. Each pin can be configured as an input or output through the server config file.
- Up to 4 DACs per SCA; the DACs take the desired voltage as a float (0..1V).
- Up to 8 SPI slaves per SCA. The SPI configuration (like speed, phase, mode . . .) can be configured in the server config file.
- Up to 16 independent configurable I2C master controllers
- Up to 1024 I2C slaves per I2C master controller
- A single Xilinx FPGA over JTAG interface

E.5.2. ScaSoftware Package

In the SCA Software package core there is a library that is structured in modules that implement the required functionality in various layers. The library was designed to be flexible and easily adaptable to the diverse systems intended to use it by its polymorphic HDLC back-end. A block diagram of the software architecture of this library is shown in Figure 53

Moreover, the SCA Software package contains the Demonstrators which are tools that directly use the library and are used for testing and for low level diagnostics. Finally, as part of the package, an SCA Simulator was developed that is able to generate SCA traffic, simulating realistic SCA behaviour, in order to allow for development and testing without real hardware.

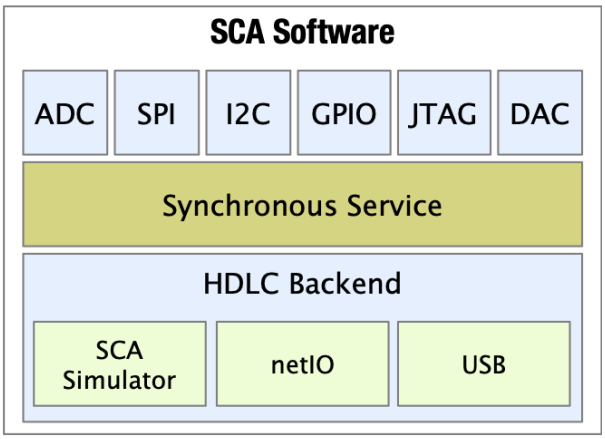

Figure 53. SCA Software Library stack.

- The library is a modular piece of software supporting SCA chips no matter how it is physically connected to the host system. Therefore the core part of the library operates on protocol data units of the SCA chip, which normally would be encapsulated in the HDLC protocol. There are a number of predefined "HDLC backends" which are services to send such encapsulated SCA requests and receive replies.
- The library scales from the simplest use cases up to scenarios of thousands of SCAs.
- The library is able to profit from concurrency features of the host system, including multi-core and multi-threaded operation.
- The library is written in a chosen version of the standard C++ dialect.
- The library is designed with reliability and robustness as a key design choice because it would serve critical, 24/7 communication.

The out-of-the-box ScaSoftware, as of May 2020, includes among its backends the NetIO backend which enables seamless communication with the FELIX software ecosystem, and particularly with the felixcore application which can route the traffic between an application based on ScaSoftware and any SCA connected through fibers to chosen FELIX machine.

Being backend-agnostic, the addressing scheme shown in Table 7 has been chosen for the library to identify a given SCA in case of NetIO.

Table 7. Addressing scheme

Backend type	Discovery variant	SCA address to use
NetIO	FELIX mapper not used	simple-netio://direct/hostname/portTx/portRx/elinkTx/elinkRx Where: 1. Hostname is a hostname of the FELIX machine where felixcore runs. 2. PortTx is the TCP/IP port on which FELIX will receive and transport further through fibers to the SCA. Typically it is 12340. 3. PortRx is the TCP/IP port on which FELIX will distribute the replies of the SCA chip. Typically it is 12350. 4. elinkTx is a two-digit hexadecimal E-link identifier on which OpcUaSca will transmit data towards the SCA. For example, 3F would mean the EC link of the first fibre of the FLX card. You can use 'felink' tool to compute the E-link identifier. Note that neither decimal format nor prefix/suffix are supported (e.g. it’s illegal to put 0x3f instead of 3F). 5. elinkRx is a two-digit hexadecimal E-link identifier on which OpcUaSca will receive data from the SCA.
netio-next	FelixMapper	Prototype definition as of May 2020.

E.6. SCA References

1. P. Moschovakos, P. P. Nikiel, et al., "A Software Suite for the Radiation Tolerant Giga-bit

Transceiver - Slow Control Adapter", presented at the 17th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'19), New York, NY, USA, Oct. 2019, paper WEPHA102.

2. OpcUaSca repository, https://gitlab.cern.ch/atlas-dcs-opcua-servers/ScaOpcUa
3. ScaSoftware repository, https://gitlab.cern.ch/atlas-dcs-common-software/ScaSoftware
4. Uao Client for OpcUaSca, https://gitlab.cern.ch/atlas-dcs-opcua-servers/UaoClientForOpcUaSca

Appendix F: Guide to Using FELIX with the SCA eXtension

The Slow Control Adapter eXtension (SCAX) is an FPGA module that emulates the GBT-SCA's communication with the back-end (i.e. FELIX and the OPC UA server) in order to provide access to registers in the FPGA. The OPC-UA server uses the protocol for the SCA's I2C interface. SCAX's connection to the registers is, however, parallel and not I2C.

F.1. Introduction

The SCA ASIC is typically installed on front-end boards in order to configure and monitor other front-end devices (usually other ASICs) on the board. Being radiation-tolerant, it can be deployed on front-end boards that are subject to high radiation doses. However, there may be several parts of the DAQ system that are FPGA-based, and are either situated in parts of the detector which are not exposed to a disruptive particle flux, or are even outside the experimental cavern, in the counting room (or USA15).

The New Small Wheel (NSW) DAQ system for instance, uses the SCA to configure and monitor six other front-end ASICs. The NSW electronics includes also FPGAs deployed on the rim of the wheel (i.e. Pad Trigger) and in USA15 (NSW Trigger Processor). The board of the latter FPGA-based system, does not feature an SCA, but like the ASICs, its parameters must be tuned and its status monitored. Even though the SCA is not present in the Trigger Processor, it was desirable to include it in a unified configuration and status monitoring scheme.

The solution came in the form of the SCAX, which makes use of the FPGA's direct interface with FELIX (via optical fiber and by deploying the GBT-FPGA [CERN_GBT_core] in its logic) , to communicate with the OPC UA server (see Appendix E). The SCAX's logic, has been designed in such as a way as to be completely transparent to the OPC server, which was initially designed to interface only with the SCA. By emulating the command-response protocol dictated by the SCA's specifications, the SCAX can establish a connection with a server as an SCA FELIX, and use the server's features to access the registers of the FPGA in which it is implemented; this is being achieved by mimicing the SCA's I2C device and to write into and read from the FPGA fabric registers. Full access to registers in the FPGA, using the already existing and well-established OPC software ecosystem is thus provided. The general scheme can be examined in Figure 54. SCAX has been deployed successfully in the NSW Trigger Processor FPGA, but it can be used by any FPGA device that features a direct connection with FELIX.

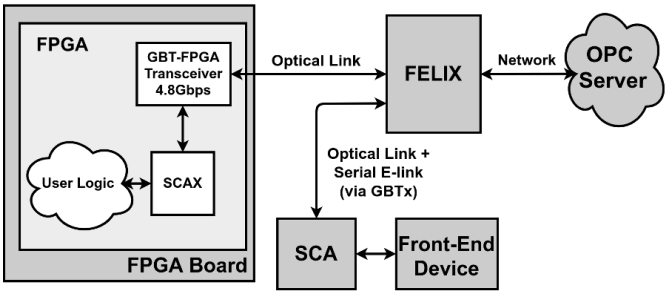

Figure 54. Connectivity of the SCA and the SCAX with the OPC Server.

F.2. Establishing a Connection between the SCAX and FELIX

This Appendix will focus on the procedure it must be followed by the user of the SCAX to connect the instance of the logic with FELIX. The reader may refer to [scax_ug] for a detailed user guide on how to deploy the SCAX in an FPGA, and to [scax_proc] for a detailed write-up of the module's architecture.

If a proper configuration is found, then going through all the steps mentioned below will not be necessary, but it is strongly recommended to follow them after the first attempt to establish a connection between all nodes.

Note also that it is not needed to connect the SCAX with any of the user logic registers when going through the connectivity validation procedure.

F.2.1. General Steps

In principle, the user must follow the procedure below to establish a system that allows access to the registers of their FPGA logic using the SCAX ←→ FELIX ←→ OPC Server arrangement:

1. Deploy the SCAX in their FPGA netlist
2. Connect the SCAX either with a GBT-FPGA instantiation, or with a GBTx device, either of which must feature a direct optical bidirectional connection with FELIX
3. Configure FELIX accordingly
4. Validate the SCAX's RX path
5. Validate the SCAX's TX path
6. Connect the OPC UA server with the SCAX instance in question

In the following subsections, each step will be addressed in more detail.

1. Deploying the SCAX in a pre-existing FPGA design

The procedure that must be followed by the user/designer to deploy the SCAX into their firmware is covered in greater detail at the associated user guide [scax_ug]. However, some recommendations will be mentioned in this subsection.

First of all, the clocking scheme must be chosen carefully, in order to avoid data corruption on both directions of the communication. For a GBT-FPGA-driven implementation, the SCAX's E-link clocks (40, 80, 160 and 320 MHz) must be related with the transceiver's reference clock. For a GBTx-driven use-case, the SCAX's E-link clocks must be derived from the E-link clock, as delivered by the GBTx to the user FPGA.

Also, for the first implementation iterations where the user is attempting to establish a connection with FELIX, the following pins must be tied to a Xilinx Virtual Input/Output (VIO) IP core: *rx_swap*, *tx_swap*, *dbg_fifo_rd*, and *ena_flx_test*.

Finally, it is strongly recommended to set the SCAX in 8b10b, 80 Mbps, and in *debug mode* via the

associated generics.

2. Connecting the SCAX to a GBT-FPGA or a GBTx

There are two use-cases that must be studied; they are depicted in Figure 55.

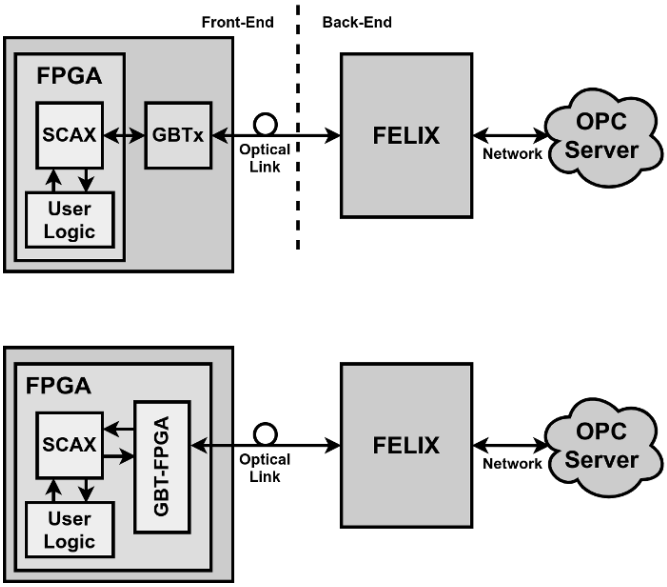

Figure 55. Two possible ways to connect the SCAX with the OPC Server.

In the case of connecting the SCAX with the FELIX/OPC Server via a GBT-FPGA, the user should connect the *parallel* ports of the SCAX (e.g. *rx/ tx_elink2bit*) with the 84-bit TX/RX bus of the GBT-FPGA instance. Depending on the bits of the bus that are chosen, the SCAX will belong to a different E-link. If the recommended SCAX interfacing configuration is chosen (i.e. 8b10b 80 Mbps), then the user should use the 2-bit TX/RX ports of the SCAX (2-bit is for 80 Mbps, 4-bit is for 160 Mbps etc.), and connect it to Egroup0 or Egroup1 of FELIX. Egroup0 EPATH0 corresponds to bits [1:0] of the GBT-FPGA's TX/RX bus. Egroup0 EPATH1 corresponds to bits [2:3] of the bus. Egroup1 EPATH0 corresponds to bits [16:17] of the bus, etc. If the same part of the bus is chosen for both directions, then SCAX will reside on the same E-link for both the To-Host and the From-Host paths, which eases the procedure.

In the case of connecting the SCAX with the FELIX/OPC Server via a GBTx, the user should connect the *serial* ports of the SCAX (i.e. *rx/ tx_elink*) with the desired GBTx pins. Testing so far has shown that by choosing a low data rate (i.e. 8b10b 80 Mbps), the communication of the SCAX with FELIX via the GBTx is easier, as it is probably not needed to train the GBTx on the datastream coming from the SCAX (see the GBTx manual [GBTx] for more details on how to perform training on the GBTx ports). If a faster data rate has to be chosen due to system restrictions, it is recommended to perform training on the GBTx bank that the SCAX's TX path corresponds to, in order to avoid data corruption on that direction. Finally, the user must deduce the E-link ID to which the SCAX corresponds, by going through the PCB's schematic and taking note to which GBTx package pins the SCAX's serial I/Os are connected.

Note that different Egroups of FELIX support specific protocols and data rates for semi-static builds.

3. Configuring FELIX Prior to Connectivity Testing

If the To-Host (SCAX-TX) and From-Host (SCAX-RX) E-links to which the SCAX corresponds are

known, then these must be activated on the FELIX side via the *elinkconfig* tool, as per the data rate and protocol to which the SCAX is configured via its generics. *felixcore* must be running throughout the connectivity testing.

4. Validating the SCAX's RX Path

In order to validate the SCAX's RX path (or *From-Host* direction, in FELIX jargon), then after configuring the FELIX E-links accordingly, the embedded SCAX's ILA must be used to probe the RX path. The ILA is activated if the SCAX is deployed in *debug mode*, as recommended for the first implementation attempt.

There are two ports that must be examined: these are the *din_dbg* and *drdy_dbg*. The 10-bit *din_dbg* bus yields the decoded data originating from the FELIX E-link. In the case of 8b10b encoding at 80Mbps, pairs of standard commas, K28.5, *must* appear periodically on the bus. This is the *0xbc* byte, accompanied by "11" in the first two bits of the bus. Hence, if the communication is sound, the *0x3bc* word will appear every five cycles, and the *drdy_dbg* will go high for the cycle the word appears. If any of these is not true, then the user should check if the optical link is aligned via the *$ flx-info GBT* command in FELIX, then check if the E-link configuration on the FELIX side is correct. If these appear to be OK, then the user should either attempt to probe the aforementioned ILA ports after attempting a different *rx_swap* state, or after attemtping a different *fereverse* configuration on the FELIX side.

5. Validating the SCAX's TX Path

In order to validate the SCAX's TX path (or *To-Host* direction, in FELIX jargon), then after configuring the FELIX E-links accordingly, the VIO connected to the SCAX's critical ports must be used to send a test packet to FELIX. First of all, the user should have *felixcore* running in FELIX, and *netio_cat* subscribed to the To-Host E-link of the SCAX, in order to ensure the test packet is indeed being received by FELIX. Then, the user should toggle the VIO's port *ena_flx_test* from low to high and then back to low (note that it *must* be kept low when not used). By doing this, the SCAX sends the following message to FELIX: *0xff 0x63 0xe5 0x5e*. If the message is reported by *netio_cat*, then the communication on that direction has been established. If not, then the user should check if the optical link is aligned via the *$ flx-info GBT* command in FELIX, then check if the E-link configuration on the FELIX side is correct. If these appear to be OK, then the user should attempt to send the message again after attempting a different *rx_swap* state, or after attempting a different *fereverse* configuration on the FELIX side. If a GBTx is used in the communication chain, then training the GBTx should also be considered.

6. Connecting the OPC Server

If both TX/RX paths are validated, then the user should attempt to connect the OPC server with the SCAX. Note that it is advised to check the system's state by trying to connect the server with already existing *SCA's*, prior to any SCAX connection attempts. If the server and FELIX seem to be working as they should, then the SCAX instance can be added to the OPC's configuration .xml. Choosing the correct E-link as a parameter in the .xml is crucial.

If the OPC server can connect to the SCAX (There is no reason not to, if Steps 4 and 5 have been validated by the user.), then the user may implement the SCAX again, in non-debug mode and by removing the VIO (note that the *rx_swap* and *tx_swap* values that work must be retained). If the

SCAX still connects, as it should, then the user may proceed with interfacing the SCAX with the rest of their logic, as per the SCAX user guide. Note that if a GBTx is used, it is recommended that the user always train the GBTx after configuring their FPGA.

If the server fails to connect, then Steps 4 and 5 should be revisited. Note that testing so far has shown that if the OPC server fails to connect, felixcore must be restarted prior to another connection attempt.

Appendix G: External emulators

Dedicated firmwares allow to turn a FELIX card into a data generator for testing and performance assessment purposes. A FELIX in such configuration is called external emulator, where the adjective external is meant to avoid confusion with the internal emulator present in the GBT and FULL mode firmware flavours. Two kinds of external emulators exist: FELIG for the GBT mode, and FMEmu for FULL mode. Instructions on how to operate FELIG and FMEmu are listed in the following.

G.1. FELIG

The FELIG firmware is available for the FLX-712 card (previously only the HTG-710). Full details are available in the dedicated user manual on CDS:

https://cds.cern.ch/record/2752360/

G.2. FMEmu

The FMEmu firmware is available for the FLX-712 and FLX-711 card models. It can be loaded on a 48-channel card but it will use 24 channels only. The FMEmu can be connected to a FELIX via a patch panel or using MTP-24 loopback fibres (in the latter case FELIX and FMEmu have to be hosted on FLX cards with the same number of channels). The FMemu can be set to receive the clock from the TTC system as a long as the TTC ST fibre is connected to it. The FMEmu can send data continuously or in triggered mode (upon reception of a L1A from FELIX). The FMEmu supports the XOFF traffic control system.

G.2.1. Quick start guide

Instructions on how to setup a FMEmu+FELIX system are listed in the following. Commands have to be entered in both the FELIX and FMEmu hosts. To distinguish between the two hosts the commands are preceded by the labels [FELIX] and [FMEmu]. Comments that are not commands are written within parenthesis.

Felix configuration: On elinkconfig enable all the desired ToHost links. In addition for each GBT link, enable the following FromHost e-links:

- Egroup 0, Epath 0 (2-bit wide, 000), 8b10b encoding (for the XOFF)
- Egroup 1, Epath 1 (8-bit wide, 009), TTC-3 encoding (8-bit wide, for the TTC)

Alignment procedure:

```
[FELIX] (configure clock selection and links with elinkconfig)
[FMEmu] flx-config set MMCM_MAIN_LCLK_SEL=0x0 (0x0 is TTC clock | 0x1 for local clock)
[FELIX] flx-init
[FMEmu] flx-init
[FELIX] flx-reset GTH
[FELIX] flx-info gbt (check for alignment)
[FMEmu] flx-info gbt (if not aligned use flx-reset GTH, then go back to previous step)
```

Once the links are aligned the FMEmu can be started in either continuous mode or triggered mode. In the latter case the L1A received by FELIX from a TTC system are forwarded to the FMEmu.

Continuous mode:

```
#[FMEmu]
#Stop emulator, a rising edge on the register is required to apply settings
flx-config set FMEMU_CONTROL_EMU_START=0x0
#Number of triggers, 0xFFFF is unlimited
flx-config set FMEMU_COUNTERS_L1A_CNT=0xFFFF
#1 for L1A triggered mode, 0 for self triggered mode
flx-config set FMEMU_CONTROL_TTC_MODE=0x0
flx-config set FMEMU_CONTROL_EMU_START=0x1
flx-config set FMEMU_CONTROL_ECR=0x1
flx-config set FMEMU_CONTROL_ECR=0x0
```

Triggered mode with optional XOFF:

```
#[FELIX]
# if you want to enable XOFF, else skip
flx-config set XOFF_ENABLE=0xffffff
#[FMEmu]
flx-config set FMEMU_CONTROL_EMU_START=0x0
#Number of triggers, 0xFFFF is unlimited
flx-config set FMEMU_COUNTERS_L1A_CNT=0xFFFF
#if you want to enable XOFF, else skip
flx-config set FMEMU_CONTROL_XONXOFF=0x1
#1 for L1A triggered mode, 0 for self triggered mode
flx-config set FMEMU_CONTROL_TTC_MODE=0x1
flx-config set FMEMU_CONTROL_EMU_START=0x1
flx-config set FMEMU_CONTROL_ECR=0x1
flx-config set FMEMU_CONTROL_ECR=0x0
```

G.2.2. FMEmu data format and payload

Each FMEmu message consists of a 32-bit field containing the extended L1ID, followed by an incremental sequence of bytes. The payload size can be set to constant using

```
flx-config set FMEMU_RANDOM_CONTROL_SELECT_RANDOM=0x0
flx-config set FMEMU_COUNTERS_WORD_CNT=<value>
```

Otherwise, the payload size is randomly drawn from a distribution. The payload size distribution can be generated with a script such as fragsizegen and loaded onto the FMEmu with

```
femuran <file.coe>
```

The FMEmu is capable of generating chunks of size up to 4-5 kB at 100 kHz.

References

- [1] CERN, GBT & Versatile Link, url: https://ep-ese.web.cern.ch/content/gbt-versatile-link.
- [2] F. Vasey et al., The Versatile Link common project: feasibility report, Journal of Instrumentation 7.01 (2012) C01075, url: http://stacks.iop.org/1748-0221/7/i=01/a=C01075.
- [3] CERN GBT Project, The GBTx Manual, V0.14 (2016), url: https://espace.cern.ch/GBT-Project/GBTX/Manuals/gbtxManual.pdf.
- [4] GBT Module for the FELIX Project, url: https://twiki.cern.ch/twiki/pub/Atlas/GBT2LAN/FELIX_GBT_MANUAL.pdf.
- [5] ATLAS Felix Group, Specifications for the FELIX FULL mode link, url: https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/docs/FullMode.pdf.
- [6] Xilinx, Xilinx VC709 Development Kit, url: http://www.xilinx.com/products/boards-and-kits/dk-v7-vc709-g.html.
- [7] ATLAS FELIX Group, BNL-711 v2 Manual, url: https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/docs/BNL-711_V2P0_manual.pdf.
- [8] Supermicro, Supermicro X10SRA-F Motherboard Model Specification, 2016, url: http://www.supermicro.nl/products/motherboard/Xeon/C600/X10SRA-F.cfm.
- [9] CERN TTC FMC project, url: http://www.ohwr.org/projects/optical-cdr-fmc/wiki.
- [10] TTC group, CERN TTC homepage, url: http://ttc.web.cern.ch/TTC.
- [11] Analog Devices Inc., ADN2814: Continuous Rate 10Mb/s to 675Mb/s Clock and Data Recovery IC with Integrated Limiting Amp, url: http://www.analog.com/static/imported-files/data_sheets/ADN2814.pdf.
- [12] Silicon Labs Inc., Si5345/44/42 Rev D Data Sheet - 10-Channel, Any-Frequency, Any-Output Jitter Attenuator/ Clock Multiplier, url: http://www.silabs.com/SupportDocuments/TechnicalDocs/Si5345-44-42-D-DataSheet.pdf.
- [13] Silicon Labs Inc., Si5324 Data Sheet - Any-Frequency, Any-Output Precision Clock Multiplier / Jitter Attenuator, url: https://www.silabs.com/documents/public/data-sheets/Si5324.pdf.
- [14] Xilinx, Xilinx Vivado Design Suite, 2016, url: https://www.xilinx.com/products/design-tools/vivado.html.
- [15] Linear Technology, LTC2991 Data Sheet - Octal I2C Voltage, Current, and Temperature Monitor, url: http://cds.linear.com/docs/en/datasheet/2991ff.pdf.
- [16] The Versatile Link Developers, The Versatile Link Common Project, 2008, url: https://espace.cern.ch/project-versatile-link/public/default.aspx.
- [17] CERN GBT-FPGA project, url: https://espace.cern.ch/GBT-Project/GBT-FPGA/default.aspx.
- [18] E-link Wrapper Deployment User Guide, url: https://espace.cern.ch/ATLAS-NSW-ELX/Shared%20Documents/Overview%20and%20General/elink_wrapper_userGuide.pdf.
- [19] SCA eXtension User Guide, url: https://espace.cern.ch/ATLAS-NSW-ELX/Shared%20Documents/NSW%20Trigger%20Processor/scax_userGuide.pdf.
- [20] SCA eXtension: a Design for FPGA Parameter Configuration within the ATLAS DAQ Scheme - IEEE/NSS Proceeding, url: https://ieeexplore.ieee.org/document/9059894.

- [21] IC-over-NetIO Gitlab Repository, url: https://gitlab.cern.ch/cbakalis/ic-over-netio