GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

Fakultät für Physik [q,p]=iℏ

Master's Thesis

Vorbereitung eines Stresstests der ITk-Pixel FELIX/RD53A Auslesekette

ITk-Pixel FELIX/RD53A Read-out Chain Stress Test Preparations

prepared by

Matthias Peter Drescher

from Gütersloh

at the II. Physikalischen Institut

Thesis number:	II.Physik-UniGö-MSc-2023/02
Thesis period:	1st October 2022 until 31st March 2023
First referee:	Prof. Dr. Arnulf Quadt
Second referee:	Priv.Doz. Dr. Jörn Große-Knetter

Abstract

Diese Masterarbeit beschreibt die Entwicklung eines FPGA-basierten System- und Stresstests der FELIX-basierten Auslesekomponenten des ATLAS Inner-Trackers (ITk). Der ITk ist Teil der ATLAS Phase-II Aufrüstung für den Betrieb am High-Luminosity Large Hadron Collider (HL-LHC). In dem Aufbau werden mithilfe der Xilinx UltraScale+ KCU116 und VCU128 FPGA Entwicklungsboards 24 Instanzen des lpGBT Aggregierungschips und 168 Instanzen des RD53A Auslesechips emuliert, sodass alle 24 Glasfaseranschlüsse der FELIX FLX-712 Hardwareplattform belegt sind. Die Trefferdaten des RD53A Emulators und die generelle Konfiguration der Emulatoren kann über ein Ethernet Netzwerk auf die Boards geladen werden. Es wurde eine Software zum automatisierten Testen des Aufbaus entwickelt, die den Aufbau und die YARR Auslesesoftware steuert. In diesem Aufbau wurden zuerst einzelne und dann mehrere RD53A Instanzen gleichzeitig gescannt. Beim Scannen einzelner RD53A Instanzen zeigten sich Datenverluste insbesondere im zweiten *device* von FELIX. Ein Scan einiger bis aller RD53A Instanzen gleichzeitig konnte zwar durchgeführt werden, dort traten aber die gleichen Probleme auf. Mit diesem Aufbau wurde ein Werkzeug bereitgestellt, mit dem Datentransfertests zur Qualifikation der finalen ITk-Pixel Auslesekette durchgeführt werden können.

Stichwörter: ITk Pixel Auslese, FELIX, lpGBT, RD53A, FPGA, Stresstest, Emulation

Abstract

This master thesis describes the development of an FPGA-based emulator setup for performing system and stress tests of the FELIX-based read-out chain components used in the ATLAS Inner-Tracker (ITk) upgrade. The ITk is part of the ATLAS Phase-II upgrade for operation with the High-Luminosity Large Hadron Collider (HL-LHC). The setup emulates 24 lpGBT data aggregator and 168 RD53A frontend instances on the two Xilinx UltraScale+ KCU116 and VCU128 FPGA development boards, to fully populate the 24 fibre links of the FELIX FLX-712 board. The hit data of the RD53A, as well as the general emulator configuration, is uploaded to the two boards over an ad hoc Ethernet network. A software for controlling the boards over this network and for automated testing using the FELIX software components and the YARR DAQ software has been developed. With the setup, single-frontend and multi-frontend scans were performed. The single-frontend scans showed data losses, especially in FELIX device 1. A multi-frontend scan of up to all RD53A instances could be performed, which however showed the same issues. With the setup, a tool to perform all kinds of data transfer assessment studies for the qualification of the final ITk-Pixel DAQ read-out chain has been obtained.

Keywords: ITk Pixel read-out, FELIX, lpGBT, RD53A, FPGA, stress test, emulation

Contents

1. Introduction

The current schedule of the Large Hadron Collider (LHC) [1] at the CERN nuclear research centre is shown in Fig. 1.1. During the Long Shutdown 3 from Dec. 2025 to Feb. 2029, the High-Luminosity Large Hadron Collider (HL-LHC) [2] upgrade to the LHC will be installed. The most important change is the increase in instantaneous luminosity to $\mathcal{L} = 7.5 \times 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ in the *ultimate* scenario. Over the whole lifetime, an integrated luminosity of up to $3000\,\mathrm{fb}^{-1}$ is expected, which will be around 10 times the value gathered with the LHC. Also, the centre of mass energy may be raised mildly from the Run 3 value of 13.6 TeV to 14 TeV.

An increase in luminosity will benefit the statistical uncertainty of measurements. For rare processes, an increase of the significance might make the process visible in the first place. Processes that are already discovered also benefit from this, since with rising statistics, also differential cross sections of increasing order can be measured. This greatly benefits the physics analyses of the data generated by the ATLAS and CMS experiments on the LHC, which are focused on validation and measurement of Standard Model parameters, measurements of the known particles properties and searches for and maybe even discovery of particles beyond the Standard Model.

For the computing and the experiments along the LHC themselves, however, the increased event rates cannot be handled with the current setups. Therefore, they also need to be upgraded during the Long Shutdown 3. As the background of this report, particular

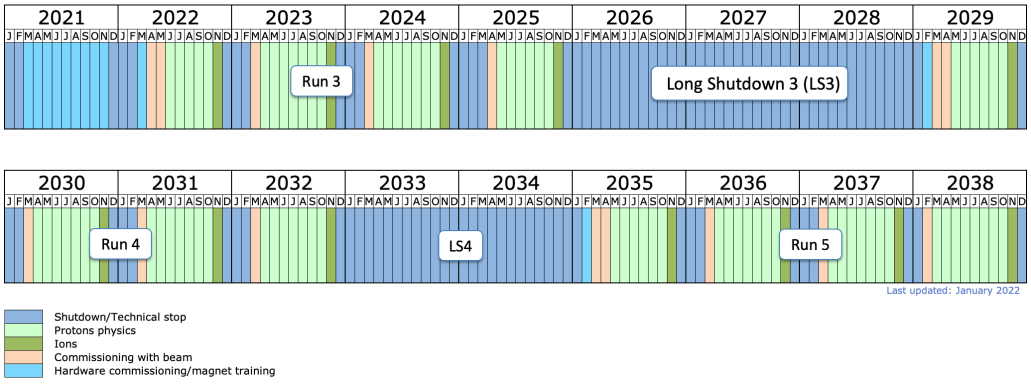


Figure 1.1.: The current LHC long-term schedule (© CERN).

focus will be given to the ATLAS experiment. The increase in luminosity will be achieved by increasing the number of collisions per bunch crossing (pile-up μ), as opposed to the bunch crossing frequency. After the upgrade, 200 pile-up events are expected to occur [3]. Consequently, for a given resolution of the tracking detector, the hit density rises, making it more difficult to distinguish between different tracks and to infer which tracks belong to a common interaction vertex. A higher fluence of particles also leads to higher radiation doses. Following the typical inverse-square law, this affects the inner layers, e.g. of the tracking detector, the most. This poses a problem for the silicon detectors used in the inner layers, whose performance degrades from the received radiation damage. Also, all electronics used in the detector must employ techniques guarding against single-event effects, where the electrical state of a circuit is changed from external radiation.

To make use of the increased event rates, a new and faster trigger system needs to be employed. The level 1 trigger rate is increased from 100 kHz to 1 MHz. Accordingly, the data needs to be read out and also processed faster, to a degree which is not possible with the current system.

In total, the upgrade consists of a new silicon-based inner tracking detector (ITk) [4][5] with higher spatial granularity and appropriate radiation hardness, a faster read-out system for the calorimeters, on which the trigger decisions are based, and a new trigger and data acquisition system (DAQ/TDAQ) [6]. The ITk will replace the whole ATLAS Inner Detector (ID) currently in use.

The hardware-based part of the read-out chain starts with the frontend (FE) chips, which read out the sensors and send the result as digital data over electrical connections. In the *uplink*, so in the direction from the detector to the counting room, multiple such electrical signals are equalized before being aggregated into one high-speed optical communication line. The Front-End LInk eXchange (FELIX) [7] component is a bidirectional bridge between these hardware-based on-detector components and the further software-based DAQ system, in particular the YARR [8] DAQ software. The *downlink* direction follows the same path in reverse, but carries trigger and configuration data instead of hit data. The bandwidth requirement for the uplink is much higher, so the communication is asymmetrical. FELIX operates as a custom PCIe board within a host server, which can in turn be connected to the TDAQ system via high speed commercial networking solutions. For our purpose, the data-generating on-detector components are the RD53A frontend chip [9], whose data streams are aggregated by the lpGBT transceiver [10]. The RD53A is a development version of the final ITkPix read-out chip. The ITkPix version 1 is also referred to as RD53B [11].

This report describes the preparation of a stress test setup for the FELIX read-out chain.

With the setup, the core FELIX functionality of accepting, decoding and reporting the received data to the rest of the read-out chain is tested at load levels, which are comparable to the ones in the final detector setup.

The data generation for this task is taking place on commercial Xilinx FPGA boards, with extensive utilisation of hardware emulators. There are two advantages to using emulators over using the actual hardware, in particular the RD53A and lpGBT, for this task. First, the amount of hardware needed for a full stress test is beyond what any laboratory has available. Even the ATLAS RD53A demonstrator at CERN, a test setup to benchmark the performance and interplay of the different components, which is at this time the largest setup made of upgrade components, uses less frontend chips than what would be minimally required for a full stress test. Secondly, emulating the behaviour of the chips on FPGA boards provides more control over the data being sent. Whereas for the RD53A an appropriate number of pixels would have to be statically masked to control the link occupancy, an emulator provides the opportunity to dynamically generate different data for each trigger, which increases the test power. With appropriate generation schemes it would also be possible to identify whether data is missing or changed in order.

Chapter 2 summarizes the background information about topics relevant for understanding the project setup and interpreting the obtained results. This includes the ATLAS detector in its current and upgraded form, a detailed description of the components used in the read-out chain and general digital design concepts, methodology and tools. In Chapter 3, the implementation of the concepts from Chapter 2 with respect to the stress test are described. The status of the stress test preparations, and the partial results of the current setup are presented in Chapter 4. An outlook of the next steps is given in Chapter 5 before a general summary in Chapter 6.

2. Background

2.1. ATLAS LHC detector

The ATLAS detector [12] is one of the four main experiments at the Large Hadron Collider (LHC). Like its counterpart CMS, its purpose is to perform precision measurements of the Standard Model (SM) particles and searches for Beyond the Standard Model (BSM) particles, which are generated in high energy proton-proton collisions. Enabled by the LHCs unique high centre of mass energy of $\sqrt{s} = 13.6\,\text{TeV}$, in particular the heavy SM particles are studied. These include the top quark and the Higgs boson, and to a lower extent the W and Z bosons, which were already studied with cleaner background at the Large Electron-Positron Collider (LEP), the LHCs predecessor.

This section describes the structure of ATLAS at the time of writing, so during Run 3, in order to show the parameters of both the current and upgraded version.

As a general purpose detector, ATLAS has the usual barrel geometry seen in Fig. 2.1. Because a particle can only be detected by its interactions with the detector matter and because there is no detector type that provides all-round information on all kinds of particles, the ATLAS detector is composed of individual subsystems arranged in concentric layers around the beam pipe. The subsystem measurements need to be combined to obtain information about the particles generated in an event. Furthermore, only particles that have a large enough lifetime to actually reach the detector volume have to be considered. These are protons p, neutrons n, photons γ, electrons/positrons $e^{\pm}$, muons $\mu^{\pm}$, charged pions $\pi^{\pm}$ and technically neutrinos ν, although they remain undetected due to their weak interaction with matter. Going from the inner to the outer layers, the subsystems are the pixel and strip semiconductor trackers, the transition radiation tracker (TRT), the electromagnetic calorimeter, the hadronic calorimeter and the muon chambers. Additionally, the solenoid and toroid magnet systems create a magnetic field within the tracking and muon detector to curve the trajectories of charged particles depending on the sign of the charge and the momentum.

The tracking detectors job is to measure the tracks of the particles for assigning the particles to the vertex they originate from, for relating calorimeter clusters to the

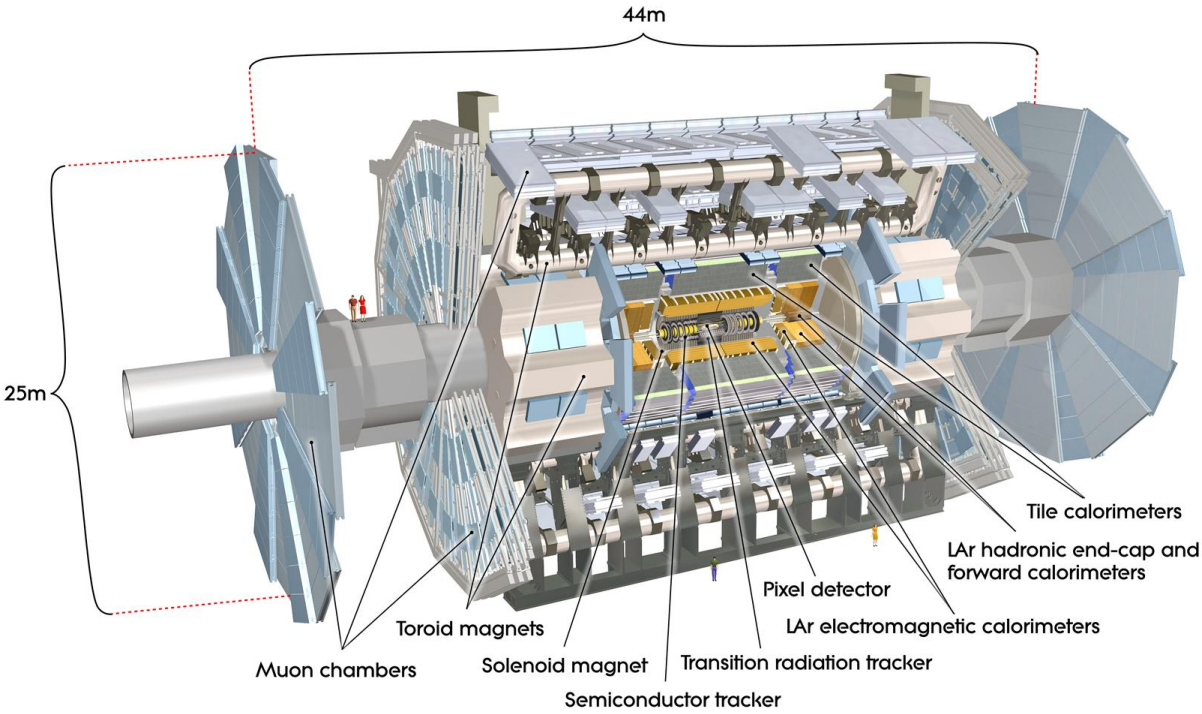


Figure 2.1.: A cross-sectional view of the ATLAS detector (© Cern).

tracks/particles and for measuring the particles' momentum from the curvature of their trajectories within the magnetic field. Instead of measuring a track directly, the track has to be reconstructed from the position information of around 47 hits the particle leaves within the tracking detector. In the current semiconductor tracker, a particle hit is recognized by its ionization of a depleted silicon diode in either pixel ($50\,\mu m \times 400\,\mu m$) or strip ($80\,\mu m \times 12\,cm$) form factor. The strip detectors have a small stereo angle between each other to provide the position resolution along the strip. In contrast, the TRT employs the operational principle of a gaseous detector to measure the ionization of Xenon gas due to transition radiation photons of the original particle generated in a radiator material. In the TRT, the spatial granularity comes from the so-called straws, which are 4 mm thin but 148 cm long cylindrical gaseous detector cells [13].

The calorimeters follow a different approach. In the calorimeters, the particle is stopped in its track by interactions with high mass material, from which secondary particles of lower energy are generated. These secondary particles do the same, and this process repeats until the energy becomes too low. The whole effect is called *showering*. By measuring the size of the shower, one can infer the energy of the original particle. Since the showers are intrinsically large objects, the spatial resolution of the calorimeters is kept

comparatively low. In the detector, a distinction is made between the electromagnetic and the hadronic calorimeters, since electromagnetic showers originating from e and γ happen on shorter distances, so that in the hadronic calorimeter, which is behind the electromagnetic calorimeter, in theory only hadronic showers are left. The ATLAS calorimeters are of sampling type, meaning the shower-generating medium is different from the active medium, which is used to detect the particles. As a result, only portions of the shower are sampled, but there exists more freedom in the choice of the absorber medium, so that the calorimeter behaviour can be more finely tuned. For the ATLAS calorimeters, the active medium is mostly a liquid Argon gaseous detector. The exception is the hadronic tile calorimeter, which uses scintillators in combination with photomultiplier tubes (PMTs) as the active material.

Muons, while theoretically producing electromagnetic showers, usually leave the calorimeters untouched, as the high mass compared to e suppresses the bremsstrahlung, which starts the shower. They are the only visible particles to do so, and thus the muon chambers as the outermost detector element are specifically designed to track them.

A further obligatory part of the detector design is the Data AcQuisition (DAQ) and Detector Control System (DCS). The DAQ system takes care of the detector electronics read-out and processing of the data up until long term storage. This is complicated by the large number of $\mathcal{O}(10^7)$ read-out channels, which generate data at the bunch crossing frequency of 40 MHz, so every 25 ns. However, only a small fraction of the events are actually of physical interest. Therefore, a trigger system is employed to select the possibly interesting from the uninteresting events and only pass the interesting ones to long term storage. For ATLAS, the trigger decisions are made in multiple stages. The first stage is the level 1 trigger, followed by the High Level Trigger (HLT) made up of the level 2 trigger and the Event Filter (EF). Selecting the physically interesting events implies reconstructing physics objects from the detector data, which takes a certain time, the trigger latency. Of course, the trigger decision can only be made for data that is already measured, so while the trigger is reconstructing a subset of an event's data, the rest of the data has to be buffered for the entire trigger latency. This is the motivation for the multiple stage design, as the coarser level 1 and level 2 triggers reduce the event rate successively to buy time for a better analysis in the event filter. The final event building rate is $\approx 3\,\text{kHz}$ with an event size of $\approx 1.5\,\text{MB}$ [14]. The hardware-based level 1 trigger with an accept rate of 100 kHz is only based on the calorimeters and muon chamber data, as reconstructing the tracks in the inner detector would take too much time. The DCS collects and processes hardware information from sensors within the detector to monitor the status of the experiment and to present it to the operator personnel. As the name

suggests, based on the status information it also controls and initializes the detector components. In the detector status, the data gathered by the DAQ system is also taken into account, so the two systems need to work together.

2.2. ATLAS ITk detector

The ATLAS Inner TracKer (ITk) detector [4] [5] is a major part of the ATLAS Phase-II upgrade [3]. Its description builds on the concept of the tracking detector presented in Sec. 2.1, with more focus on the changes to the current detector and on the technical details. At the time of writing, it is in a transition between the research and development phase to the first steps of production.

The layout of ITk can be seen in Fig. 2.2. With respect to the old detector, the TRT is removed and only semiconductor strip and pixel detectors are used to measure the hits, since they are better suited for the high occupancy environment of the HL-LHC. The image shows a cross-section of only one quadrant of the detector, so the layout has to be mirrored around $z = 0$ and rotated in 3D around the beam line at $R = 0$. The pixel modules shown in red are arranged in three areas, the flat barrel, the inclined barrel and the end-cap part. They are distinguished by the type of support structures they use (not shown in the figure) and the angle between the module and the beam pipe. Similarly, also the strip detectors shown in blue are arranged in a barrel and end-cap geometry. Within these regions, the detector has multiple layers, for example 5 for the pixel barrel region. The layout was optimized with the help of computer programs to provide at least 11 hits in the pseudorapidity region $|\eta| \leq 4$.

The basic building block of the ITk pixel detector is a module. A bare module is an assembly of a sensor read out by one, two or four ITkPix frontend chips. They are named single chip, dual and quad modules, respectively. To cover all detector layer requirements, there are a total of 9 module types. The module is created by gluing the bare module to a flexible Printed Circuit Board (PCB). Due to the nature of the sensor, each pixel of the sensor needs an electrical one-to-one connection to the corresponding analogue circuitry on the frontend chip. This is achieved by placing small solder bumps on the frontend chips contacts, placing the contacts on top of each other and letting the solder reflow at 250 °C (bump bonding). One frontend provides 384×400 pixels of size $50 \times 50\,\mu m^2$. The electrical connections to the flex PCB are established by wire-bonding.

For ITk strips, a module is built from 1-2 silicon sensors and up to 4 hybrids, which are powered by a *Powerboard*. The hybrids are PCBs that host up to 12 strip ABCStar read-out chips, whose data streams are collected by the HCCStar controller chip. They

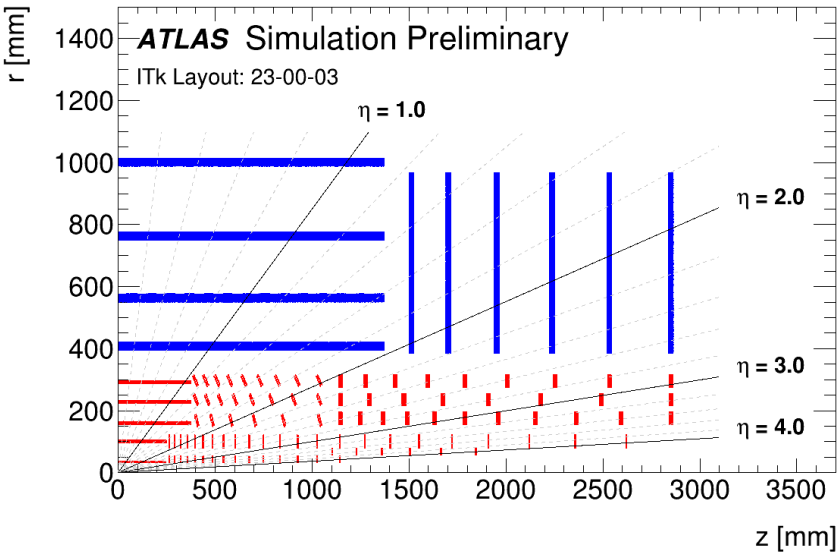


Figure 2.2.: The layout of the upper-right quadrant of ITk. The strip detectors are shown in blue and the pixel detectors are shown in red. The coordinates are the radial (R) and longitudinal (z) component of a cylindrical coordinate system in line with the detector symmetry [15] (© CERN).

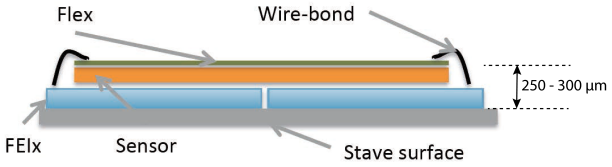


Figure 2.3.: Drawing of a quad pixel module to illustrate how the flex, sensor and frontend chips are connected [4] (© CERN).

are glued onto the sensor and then wire-bonded to it to provide the electrical connections. There are multiple sensor types used for the barrel layers and the end-cap layers. Their strip pitch ranges from 69.9 µm to 80.7 µm and the strip lengths from 15.1 mm to 60.2 mm.

An illustration of typical pixel and strip modules is given in Fig. 2.4 and Fig. 2.3. Overall, an active area of $\approx 12\,\mathrm{m}^2$ is covered by 10276 pixel modules and $\approx 165\,\mathrm{m}^2$ by 17888 strip modules.

For ITk pixel, the read-out chain is drawn in Fig. 2.5. The ITkPix frontend chip outputs 1.28 Gbps data over 1-4 lanes, depending on the bandwidth requirement of the layer the module is in. For the innermost layer, an average of 247.1 hits per chip are expected for $t\bar{t}$ events at $\mu = 200$, resulting in a maximum bandwidth requirement of 3.97 Gbps in the inclined barrel [4]. The frontend chip output is connected electrically over up to 5 m to the Patch Panel 1 (PP1) region at the side of the ITk detector. There, the signal quality, which experiences losses along the cable, is restored by the GBCR2 chip (Sec. 2.7.3).

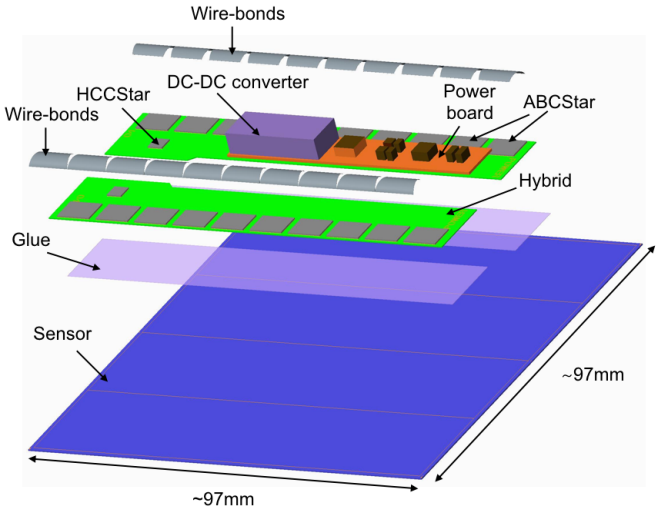


Figure 2.4.: Drawing of the components and the layout of a barrel short-strip module [5] (© CERN).

After further aggregation by the lpGBT (Sec. 2.7.1), the electrical signals are converted into optical signals in the VTRx+ optical transceiver (Sec. 2.7.2). Four GBCR2s, four lpGBTs and one VTRx+ are assembled on an *Optoboard*. One VTRx+ contains 4 uplinks, which operate at 10.24 Gbps, and one 2.56 Gbps downlink, which is shared between the 4 lpGBTs/GBCR2s. Together with powering and monitoring circuitry, 8 Optoboards form an *Optobox*. The optical interface of the VTRx+ concludes the on-detector components.

On the off-detector side, FELIX (Sec. 2.8) receives/sends the uplink/downlink lpGBT data frames over 24 duplex fibre links. For the whole DAQ system, so all subdetectors, the TDAQ TDR reports 545 FELIX cards within 279 FELIX servers to be used [6]. At

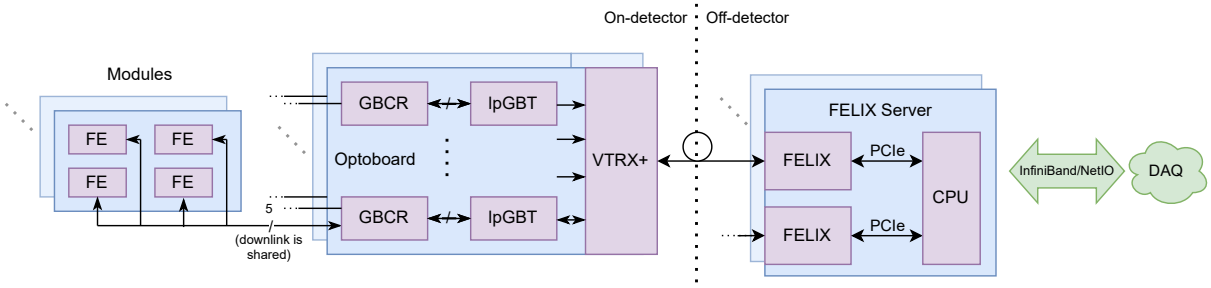


Figure 2.5.: A block diagram of the hardware-based part of the ITk pixel read-out chain. The number of connections per module is only an example, and changes between layers. Also, with the RD53B's link sharing feature [11], one master FE may be the only FE of a module, that is connected to an Optoboard.

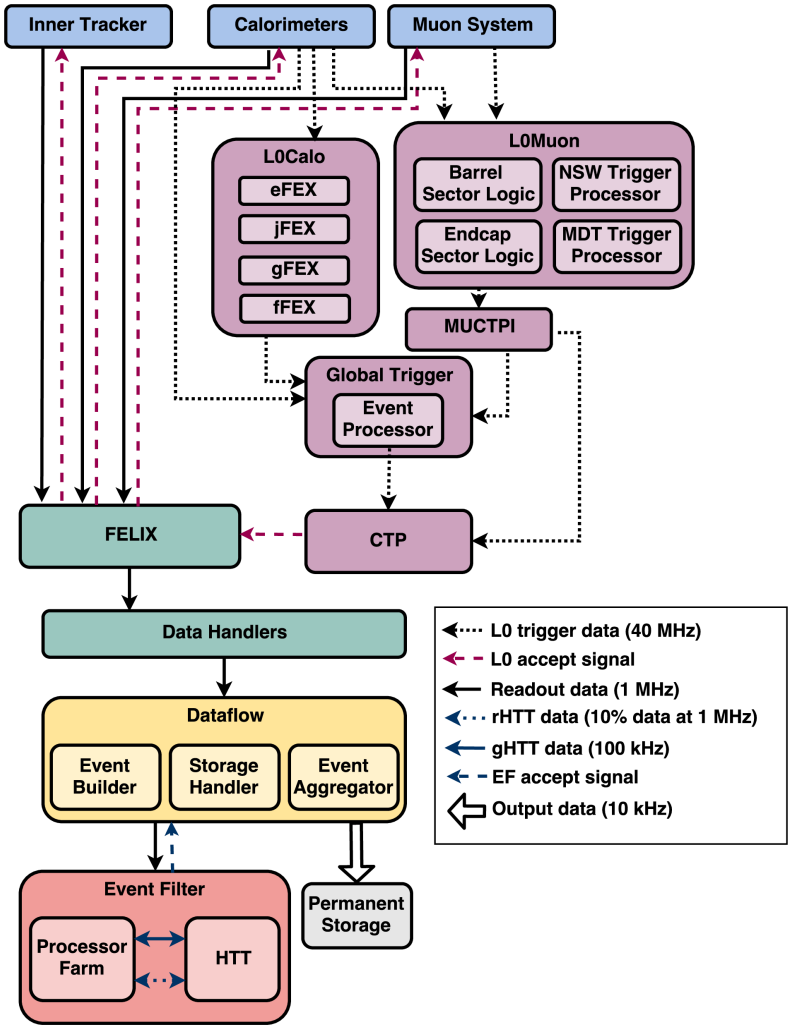


Figure 2.6.: An abstracted view of the ATLAS Phase-II detector dataflow [6] (© CERN).

this stage, the trigger system becomes relevant, so it is shown in Fig. 2.6. In parallel to the connections to FELIX and at the full 40 MHz, the calorimeter and muon chamber data is processed by the level 0 (L0) trigger hardware to generate the L0 accept at 1 MHz. The L0 accept signal is distributed via FELIX to all subdetectors, where the full data has to be buffered for the 10 µs trigger latency of the L0, to be read out on an accept signal arrival. From FELIX, the subdetector-specific data is passed via high-speed network connections $\mathcal{O}(100\,\text{Gbps})$ to the data handler PCs to be processed according to the subdetector type. The output of the data handler PCs is then passed on to the second trigger stage, the event filter (EF). Besides the EF itself, there are three components that are involved in this: The event builder collects the event fragments from the data handler and assembles them to the full event. The storage handler is the high capacity buffer for the read-out data, to store the event data while the EF processes it. Finally, the event aggregator is the

interface to the CERN permanent storage, which takes care of, for example, compression. In the end, the event size is $\approx 6\,\text{MB}$ at a rate of 10 kHz. In contrast to the Phase-I ATLAS detector in Sec. 2.1, the trigger system only has two stages. However, the possibility to extend L0 to a two level hardware trigger is left open. Also, the possibility to use tracking information in the L0 hardware trigger is explored. In this case, the ITk pixel 3rd layer and above are read out at a trigger rate of 4 MHz. For the two innermost layers, 4 MHz would not be possible, due to the bandwidth limitations and the high hit rates.

For the design of the detector it is very important to consider the high radiation levels present within it. The radiation levels are the largest in proximity to the beam pipe and decrease with radius. This is especially critical for the pixel detector, whose inner layers are closer than 5 cm to it. The radiation affects both the passive pixel sensors and the read-out electronics. For the sensors, radiation damage alters the electrical characteristics of the semiconductor material in several ways. Overall, the small leakage current, that passes the sensor despite being reversely biased, increases, which leads to more background and noise in the measurement. In extreme cases, even a thermal runaway may occur, where the leakage current heats up the sensor, which in turn increases the leakage current even more, eventually destroying the sensor. For the inner layers, the conventional sensor design is not enough to counter the radiation damage, so 3D sensors are used. The difference is, that their electrodes extend into the sensor bulk, as opposed to being layers that are applied on top of it. As a downside, for this reason they are harder to produce in large quantities. Also, it is within the specifications to replace the inner ITk pixel layers after half the detector lifetime, so after $2000\,\text{fb}^{-1}$ of integrated luminosity and after receiving a radiation dose of 7.2 MGy. For the electronics, single event upsets (SEU), in which a memory cell changes its state, represent a very big challenge. This not only introduces errors in the hit data, but more importantly also affects critical circuitry and modifies the configuration stored in the chips, changing their behaviour. These undesired effects can be reduced by providing redundancy of the registers in the form of triplication, which is the approach taken by the lpGBT. For the frontend chip ITkPix, the configuration is refreshed regularly, resetting any wrong registers. This technique is called trickle configuration, with refresh rates of up to 10 Hz. On a further note regarding radiation, the choice to use long electrical connections to the Patch Panel 1 region in the read-out chain is motivated by the inability to produce sufficiently radiation hard laser diodes for use in the inner pixel layers. For the off-detector components, the above considerations are not necessary, so that commercial hardware, most notably FPGAs (Sec. 2.6), can be used.

2.3. High-speed serial communication

In a world, where smart devices and digitalization are on the rise, and the processing capabilities of smartphones, PCs and internet services increase steadily, it is a necessity to have high-speed chip-to-chip and device-to-device communication, beyond just the particle physics context. For this, high-speed serial connections, especially over optical fibre, offer a very good solution. As seen in Sec. 2.2, particle physics experiments and in particular the ATLAS ITk detector are no exception to this trend. As such, they make large use of these technologies in the read-out chain.

2.3.1. Serial communication

In essence, serial communication is the act of transmitting data over a single wire, as opposed to multiple at the same time, which would be parallel communication. For serial communication, data is *serialized* at the transmitter (TX), meaning that an N-bit word at a transmission frequency f is split into single bits at a proportionally higher frequency of $f \cdot N$. At the receiver (RX), the signal is then *deserialized* back into the original word, reversing the serialization. On the downside, in theory serial communication seems to be less performant than parallel communication at the same line rate, simply because the multiple wires of the parallel signal can carry more information at the same time. Also, due to serialization/deserialization and due to both data and control signals being transmitted over the same wire, more logic is necessary at both ends of the communication. It still has several advantages: Using fewer wires leads to lower material cost, lower space consumption of the PCB traces or the wires, and lower pin count on the communicating Integrated Circuits (ICs). In comparison, with the possibility of having billions of transistors on one IC, the increase in logic hardly matters. The theoretical lack in speed is made up in practice by better electrical characteristics of one wire versus multiple wires. First, the data on multiple wires may affect each other as Inter-Symbol Interference (ISI) happens. Secondly, at high enough line rates the physical delay of the electrical signals within the wire becomes important. If there are significant delay differences between the wires of a parallel bus, bits of the same word will be assigned to different words at the receiver. Managing this problem becomes harder with increasing wire count, and serial communication therefore performs the best in this regard.

In addition to the data itself, it is important to transmit the information, on when the data bits are to be sampled by the receiver. In *source-synchronous* designs, the transmitter sends the clock over an additional wire. The receiver then samples the data for example on the rising edge of the clock signal. As an alternative, the receiver extracts the same

clock signal from the data alone in *self-synchronous* designs. This requires special Clock and Data Recovery (CDR) circuitry on the receiver side. Conceptually, the bit boundaries and therefore the frequency and phase of the sampling clock can be extracted by observing the timing of the leading and falling edges of the data. Again, due to the lower wire count and simpler delay management, usually the choice is in favour of the self-synchronous designs, despite the additionally required CDR circuit.

2.3.2. The physical layer

The simplest way of transmitting serial data is via electrical connections, since no conversion to another medium like optical signals is needed. Also, conventional PCB design can only print electrical traces. To send data with Gbps data rates, however, additional techniques have to be employed to ensure that the electrical medium is not distorting the data to a point, where it can no longer be understood by the receiver.

First, as opposed to single-ended inputs/outputs (I/O), where the voltage level of just one wire is compared to a threshold voltage to decide between the logical `0` and `1`, differential signalling is used. In differential signalling, the data is not only transmitted over one wire p, but also inverted and transmitted over another wire n at the same time. The receiver then takes the voltage difference between the two. If the voltage of p is larger than n, it is counted as logical `1`, or `0` otherwise. The advantages of differential signalling are, that ElectroMagnetic Interference (EMI) and other noise applies to both wires simultaneously, so that the difference remains unchanged. Also, for a given operating voltage U, the voltage swing of the signal is increased to $2U$. On the downside, two wires have to be used, which comes with the same difficulties as described above.

To reduce the effect of electromagnetic interference on the cable, it is shielded with an envelope connected to ground. For differential signalling, the wires inside the cable are twisted around each other to further improve the noise resistance. Finally, electrical termination is used to match the wire's impedance to the device it is connected to, so that there are no discontinuities in the impedance, which cause high frequency signals to be reflected back.

The capacitances and inductances of the cable smooth the supposedly steep transitions between the logic levels. To actively counteract this, the transmitter may place pre-emphasis on the signal, which over-pronounces the transitions in the beginning before resolving to the regular voltage level. On the receiver side, equalization may be used to counter the frequency dependent attenuation of the cable.

If the two ends are at different voltage levels, a capacitor may be used to block the Direct Current (DC) component and only let the Alternating Current (AC) component

through. This is used in the connection to the frontends, as required by ITks new serial powering scheme [16].

An alternative to electrical transmission is the optical transmission, where the initially electric signals are converted into optical signals by laser diodes or light-emitting diodes (LEDs). The light then travels the main distance between the RX and the TX within an optical fibre. Even when bending the fibre, the light stays within it due to total internal reflection. At the receiver, photodiodes convert the data back into electrical signals. The benefit of the fibre is the better transmission quality of the signal: The signal does not suffer from capacitances or inductances and does not suffer from electromagnetic interference. Overall, fibre optic cables allow for higher data rates over longer distances than electrical cables.

2.3.3. Serial communication techniques

Protocols and frames

With the techniques of the previous sections, bits can be transferred from one device to another. However, the goal is to transmit whole words in addition to status information. Therefore, a protocol is employed, which specifies the logical connections of the data for the TX and RX and with that provides the tools for the receiver to recover these two things from the stream of bits.

Very often, a protocol defines a *frame*, which is a fixed pattern that assigns meaning to the bits. The frame pattern assigns meaning to the frame bits by their position within the frame. A very simple example for this is a 10-bit frame, where the first two bits, the *header*, always have the constant value `10` and the rest is an 8-bit data payload. The transmitter repeats this frame continuously. The receiver, who initially sees only a stream of bits, may then extract the data payload by *aligning* the received data with the help of the header: Once a header is observed, the following 8 bits are the payload. Of course, the `10` pattern may also be present in the payload data. To prevent falsely locking to such patterns, a Finite State Machine (FSM) is employed, which for example requires a certain number of consecutive headers to be present before going into a *locking* state. In this example one can also see, that by using a protocol only a certain fraction of bits can be used for the payload, because some bits are needed for bookkeeping. In this case, the protocol *overhead*, so the ratio between non-payload bits and frame size, would be $2/10 = 20\,\%$. An alternative to using a header are synchronization frames, which are uniquely identifiable patterns that can be aligned to.

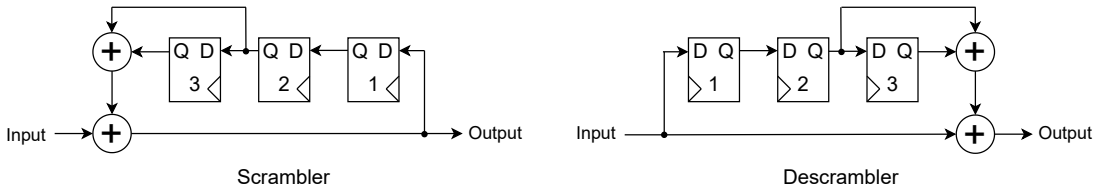


Figure 2.7.: An example scrambler and descrambler for illustrative purposes. Both components are built on flip-flop stages in a linear-feedback shift register and xor logic gates.

Channel bonding

In order to increase the bandwidth, one may transmit data over multiple channels. In this case, each channel is its own serial link, but the payload is shared between the links. This needs to be supported by the protocol. In particular, the receiver must be able to correct for different delays in the channels, which would make the data arrive at different times and corrupt it. This process is called *channel bonding*. Similar to the frame alignment, special channel bonding frames mark the frames which were sent at the same time, and the receiver hardware has to de-skew them in case they arrive at different times.

Scrambling

In particular for AC coupled serial lines, it is important for the bit stream to have on average the same amount of `1` and `0`, with sequences of the same value being kept as short as possible. This is to prevent DC components in the signal, which would be filtered out by the coupling capacitor. Accordingly, such a bit stream is called *DC balanced*. In the extreme case, a stream of constant `1` would be received as constant `0`, since there are no transitions and hence no AC component at all. Also, the CDR circuit, which relies on the transitions within the data, can operate more reliably if the transitions happen frequently.

A way to approach DC balance is to use a *scrambler*. For serial communication, a scrambler is usually implemented as a Linear-Feedback Shift Register (LFSR), which may negate the outgoing bit based on previous output values. The process can be reversed by *descrambling* the scrambled data at the receiver. An example scrambler and descrambler is shown in Fig. 2.7. Given a non-zero content of the flip-flops, the scrambler will break up long sequences of `1` and `0`. Multiplicative scramblers, like the one in Fig. 2.7, can be descrambled based on the bit stream alone, so a frame synchronization is not necessary. The scrambler is usually defined as a polynomial in $Z(2)$ space, where the exponents n of the x^n terms denote the position in the LFSR and the addition denotes the xor operation of the bits. The polynomial for the scrambler in Fig. 2.7 is $G(x) = x^3 + x^2 + 1$. Usually,

primitive polynomials of degree N are selected, since they generate the maximum number of $2^N - 1$ non-zero N-bit output values.

Forward error correction

Forward Error Correction (FEC) describes the process of sending redundant information with the data to be able to find and correct errors, that originate from an imperfect or noisy transmission line. In this way, up to a fixed degree, the receiver can recover the original correct data by itself. It is useful in situations, where the receiver cannot request a re-transmission of the data, for example due to the link being one-directional or a broadcast. The redundant information is calculated based on the data and an Error Correction Code (ECC), for example the Reed-Solomon code [17] used in the lpGBT. The number of recovered errors can also be used as a measure for the transmission quality.

Interleaving

Interleaving improves the error correction performance when block codes are used, which only work on fixed size inputs, and when a frame consists of multiple such blocks. The assumption is, that typically bit errors in the transmission will be made close to each other, when they are caused by the same event. These types of errors are called *burst errors*. Without interleaving, one string of such errors within a block may exceed the number of errors the ECC can correct. Interleaving reorders the bits before sending, so that bits, that are transmitted close together in time, are from different blocks. The receiver then needs to restore the original order in a de-interleaver. Thus, with interleaving, a burst error on the transmission line is distributed to many ECC blocks and may still be treated.

Bit error testing with pseudo-random data

If the output of a multiplicative LFSR of size k, like the one in Fig. 2.7, is fed back into itself and the flip flops have a non-zero value, a new bit is generated on each clock cycle. When the polynomial that generates the LFSR is a primitive polynomial, the LFSR value will reach all values from 1 to $2^k - 1$ exactly once within a fixed sequence before repeating. The value 0 is excluded, as this is a fix point of the LFSR. Since in such a stream each possible number appears with the same frequency, but the sequence in which they appear is fixed, the resulting bit sequence is a PseudoRandom Binary Sequence (PRBS). The number of shift register stages k is often appended in the name, so for example PRBS7 for a sequence with $k = 7$. These PRBS sequences are very easy to generate in hardware, and, when used as test data, allow checking for the number of errors in a transmission due

to their deterministic nature. In addition, they are DC balanced, which benefits their use in testing a serial link. The ratio of erroneous bits to the total number of bits received is quoted as the Bit Error Rate (BER) for assessing a links quality.

2.4. DMA and RDMA

In a processing system, Direct Memory Access (DMA) is a technique to accelerate moving data between a peripheral component and the systems Random Access Memory (RAM). To perform a device read without DMA for example, the Central Processing Unit (CPU) would need to request data from the component, save it in its internal registers and write it to the RAM manually. This is inefficient, as the CPU cannot process other tasks while reading. With DMA, the CPU instead just instructs the device to send or receive data to or from a certain location in the RAM and the device directly accesses the memory without the help of the CPU. Thus, the bandwidth is only limited by the memory bus speed and the CPU is free during the transfer. With DMA, more than one device may want to access the memory bus at a time, and some kind of arbitration is needed. Especially in old computer architectures, a custom DMA controller chip is used for this purpose. In the modern PCIe (Sec. 2.5.1), the arbitration is performed by the switches and host bridge.

Remote Direct Memory Access (RDMA) is an extension of DMA, which works between computers on a network as opposed to components within a computer. With RDMA, instead of accessing data from a computer through a network socket using the operating system and in turn the CPU, a data transfer is done directly by the network card hardware of the two ends.

2.5. Protocols

A number of protocols are used in the ATLAS ITk read-out chains electronics. To be able to refer to them later, this section summarizes the most important ones. Within the scope of this report, of course only a limited level of detail can be presented. Thus, instead of the technical details, more focus is put on the overall structure and capabilities of each protocol.

2.5.1. PCIe

PCIe [18] stands for Peripheral Component Interconnect Express and is a protocol designed for connecting the system processor to the system components. A PCIe link may

consist of 1 to 32 lanes made of one receiving and one transmitting differential pair. The lane speed increases with the PCIe version as technology advances. For versions 1.0 to 3.0, it is at 2.5 Gbps, 5 Gbps and 8 Gbps, respectively. The total bandwidth of a PCIe connection is then the product of the lane speed, the lane count and the decoding efficiency, which is 80 % for the 8b/10b encoding of versions 1.0 and 2.0 and ≈ 98.5 % for the 128b/130b encoding of versions 3.0 to 5.0.

The system of components interconnected by PCIe is called a *fabric*. The base of the fabric is the *root complex*, which connects the system CPU and system memory to the components via one or more PCIe ports. While a PCIe link is always point-to-point, a switch may be used to connect multiple PCIe endpoints to just one PCIe port on the root complex or on another switch. The switches also allow for peer-to-peer communication, where the PCIe components of the downstream ports of the switch may send messages to each other.

To provide structure to a PCIe implementation, the communication is split into three different layers. The uppermost layer is the *transaction* layer, which contains the *data link* layer and at the lowest level the *physical* layer. Transaction Layer Packets (TLP) are the highest level of packets and provide the main PCIe function. These are successively placed inside the packets of first the data link layer and then the physical layer. On the receiving side, the process is reversed. The job of the physical layer is to establish the communication between the points. This includes on the logical side the frame encoding, configuration negotiation and channel bonding, and on the physical side the CDR circuitry, buffers and electrical interfaces. The middle *data link* layer processes error and power state information. It generates a Cyclic Redundancy Check (CRC) code for outgoing packets, which can be used to detect errors in the receiver's data link layer. If an error is detected, the receiver data link layer may send a retry message to call for a retransmission of the package, which is stored in a retry buffer on the transmitters data link layer.

The interfacing between the system components and the CPU is done by mapping the component functions into the memory space(s) of the CPU, like the DMA technique in Sec. 2.4. A write or read to the CPU memory space may then in reality be transmitted not to the memory itself, but rather to the components over PCIe. As such, PCIe uses four address spaces with corresponding transaction types. The first two, the *memory* and *I/O* space correspond to transactions in the memory and I/O space of the CPU. The third *configuration* space is mapped to the device control registers, to control the device function and to access PCIe metadata. Finally, the *message* address space is used to replace physical interrupt wires, which are used in the preceding PCI protocol. In

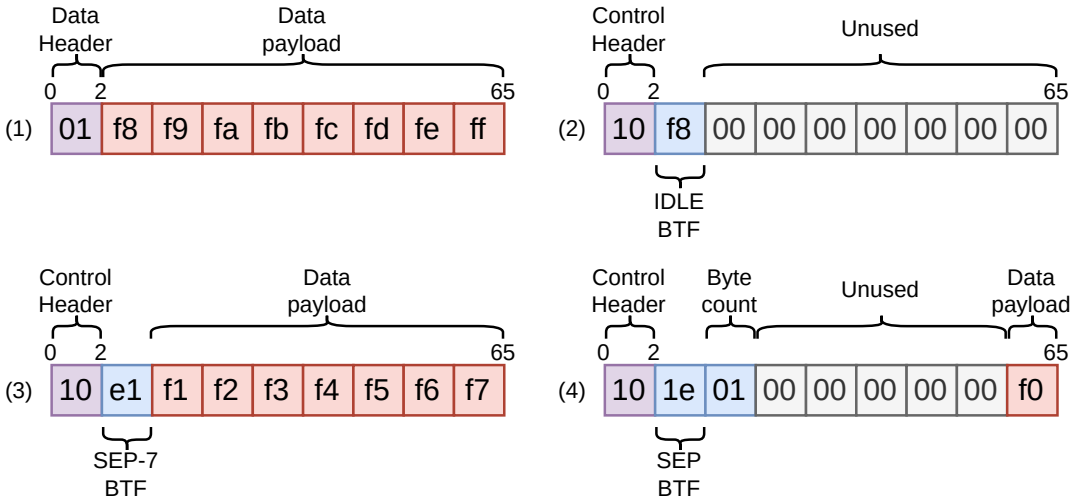


Figure 2.8.: An example Aurora 64b/66b transmission. First, the hexadecimal values `0xff` to `0xf1` are sent, counting down. Finally, the single byte `0xf0` is sent. The transmission is interrupted by an `Idle` block, without effect on the data.

contrast to the other three, the message space does not have read/write transactions.

2.5.2. Aurora 64b/66b

The Aurora 64b/66b protocol [19] is a protocol to be used for point-to-point communication over serial lanes. The link may be operated in one direction (simplex) or bidirectional (duplex) over one or multiple lanes. The basic structure is a 66-bit wide frame, which is split into 2 synchronization header bits and 64 bits of payload split into 8 octets. To obtain DC balance, the scrambler generated from the polynomial $G(x) = 1 + x^{39} + x^{58}$ is applied on the 64 data bits. The way the frame has to be interpreted by the receiver depends on the *block* type of the frame. If the header is `01`, the frame carries a `Data` block. If it is `10`, it carries a control block, and the block type is set by the value of the first octet, the Block Type Field (BTF). The header may still be used for synchronization, as the values `00` and `11` are illegal.

A data transfer of N bytes happens by sending the first $\lfloor N/8 \rfloor \cdot 8$ bytes via `Data` blocks, which have 8 octets available, and then signalling the end of the transfer with either a `Separator` or `Separator-7` block. These two blocks also carry the remaining $N \mod 8$ bytes. The `Separator-7` is a special case of the `Separator` block needed for the case where $N \mod 8 = 7$. An example transmission can be seen in Fig. 2.8.

The remaining blocks are used for control and initialization of the link (`Idle` / `NotReady` / `Clock Compensation` / `Channel Bonding`), for flow control (`Native Flow Control` /

`User Flow Control`) and for application dependent purposes to be defined by the user (User K-Blocks). They may intersperse the data stream depending on a priority hierarchy.

A special mode of operation is the strict alignment, where only either data or control blocks can be transmitted over all lanes of a multilane connection. The exception is the end of a data transfer, as marked by a `Separator` block.

2.5.3. TTC data format

The Timing, Trigger and Control (TTC) protocol is a custom protocol used to control the RD53A/ITkPix frontend chips over a 160 MHz serial line. The frames are 16-bit wide and consist of two DC-balanced 8-bit characters, which each are categorized into 32 `data` symbols for the values 0 to 31, 15 `trigger` symbols or 7 `command` symbols. Synchronization is done by aligning to a synchronization frame, which can be uniquely identified and needs to be regularly inserted into the stream. Otherwise, a frame is either a trigger or a command to the chip.

The trigger consists of a `trigger` symbol (first 8 bit) and a `data` symbol (second 8 bit), which encodes the trigger tag value to be reported during read-out of the corresponding hit data. There are 15 different triggers to signal the chip, which combination of the 4 bunch crossings that happened during the transmission of the trigger are supposed to be read out.

The `command` symbols are used to read/write to the registers, to control the charge injection calibration, to pass one frame (no operation), or to reset the chips internal counters. For performing one of these commands, the corresponding `command` symbol is repeated twice to obtain the first frame. Some commands, like the write register command, have argument frames, which are sent immediately after the command frame. The argument data is also encoded using the 32 `data` symbols. How many argument frames a command has and how the data is interpreted depends on the command.

2.5.4. I2C

The Inter-Integrated Circuit (IIC/I2C) bus protocol [20] is very different from the other protocols in this section, as it is a source-synchronous serial protocol over two single-ended wires, each of which is used bidirectionally. It is designed for configuring components on the same board and for passing control or status data. As such, it only has a comparatively low (baseline) speed of 100 kHz. The two lines, a Serial CLock (SCL) line and a Serial DAta (SDA) line, connect to usually one bus master and up to 127 slaves in the 7-bit addressing mode. The number of slaves on a bus can be extended to 1024 using 10-bit

addressing. If no device is sending data, the lines are held to logic `1` by pull-up resistors. A device, that wants to send data, then may pull the lines to logic `0` or leave them open for logic 1. The protocol is defined such that only one device at a time is allowed to write on the bus.

Simple N-byte write and read transactions are supported, which can be directed to any of the slaves by specifying its (usually 7-bit) address. Despite its simplicity, this is usually enough to configure a device by reading/writing to its registers or performing simple monitoring tasks like reading out a sensor every few seconds.

2.6. FPGAs

2.6.1. FPGA concept

The development of Application Specific Integrated Circuits (ASICs) is a very expensive process, both in required money and development time. One major issue with ASICs is, that they have a fixed functionality once they are produced. Together with the expensive manufacturing of ASICs, it is therefore often only possible to have a few revisions of a chip, before production readiness has to be reached.

To reduce the cost of low batch size production and provide more flexibility in the development, Field Programmable Gate Arrays (FPGAs) were invented. The overall goal is to provide one general integrated circuit, which can be (re-)programmed to perform an ASICs functionality, within limitations. This is achieved by designing a general *fabric* of Configurable Logic Blocks (CLB) with an interconnect structure. The logic blocks themselves only perform for example simple Boolean logic operations on a fixed number of inputs. Still, given a large enough number of them and enough freedom in choosing how they interconnect, complex functionality can be achieved. The configuration within the logic blocks and of the interconnect logic is saved in memory cells on the chip. Depending on whether the memory is permanent or volatile, an FPGA may be one-time programmable or reconfigurable at runtime.

The core of the CLB is the LookUp Table (LUT). LUTs take a fixed number of inputs and output a bit depending on the contents of a programmable truth table. A medium-sized modern FPGA would contain $\mathcal{O}$(100k) of them. In a CLB, one or more of these LUTs may be present, together with flip flops to store the result and carry connections for efficiently implementing binary addition. The design of the CLBs and the interconnect system depend on the manufacturer and chip family. Luckily, the hardware design tools used for programming the FPGAs abstract away the underlying features of the target

chip, so that the low-level design is usually not important to the designer.

Besides the CLBs, there are a lot of other blocks on an FPGA to enable common functionality that would be very inefficient or impossible to implement purely with CLBs. This includes for example a number of RAM blocks (BRAMs) in the kB size range for storing data and Digital Signal Processing (DSP) blocks for dedicatedly performing arithmetic operations like multiplication. Also, the I/Os of the chip are reconfigurable. Basic I/Os have a selectable I/O standard and can provide basic delay and deserialization/serialization features. For high-speed communication, usually also high-speed transceivers are built into the FPGA, with built-in CDR circuitry and hardware support for some common protocols. Some FPGAs also have a built-in processing system (PS) next to the programmable logic (PL). The processing system contains a (multicore) CPU and some peripherals to interface with the PL or with outside components. This PL-PS combination is called a System on a Chip (SoC). It is useful for either controlling the custom logic on the PL using the CPU, or for using the PL to accelerate code running on the CPU. A SoC may also be created by implementing a *soft* CPU inside the FPGA fabric.

2.6.2. FPGA design flow

From the programmers point of view, working on the LUT level directly would be incredibly inefficient and error-prone. Therefore, Hardware Description Languages (HDLs) are used, which were originally created for describing ASICs. The two most common ones are VHDL [21] and Verilog/SystemVerilog [22]. With them, the developer may, independent of the FPGA, define a system's logic functionality at a higher level of abstraction, which is then eventually compiled into the low-level bitstream used to program the FPGA. Besides hiding the hard to program low-level FPGA features, they enable code to be shared and reused in the form of libraries, which are called Intellectual Property (IP) cores in the hardware world.

To give an understanding of the abstraction level they provide, the language structure is sketched here. The basic building blocks are *entities* and their associated *architecture* in VHDL, or equivalently *modules* in Verilog. They perform a certain functionality on a set of inputs and outputs (*ports*), which can be, for example, one- or multi-bit signals. Their implementation details are defined in code and are hidden from other modules outside them. To achieve their functionality, they may use simple operators like the ones of Boolean logic, comparisons or simple addition. VHDL splits the port definition and the implementation details into *entity* and *architecture* definitions, respectively, while Verilog combines them into one *module*. Modules can themselves also instantiate and use other modules, so that a module hierarchy arises. With the use of flip flops and

latches, they may have a state. One may also instantiate predefined *primitives*, which directly translate into chip hardware features. Additionally, generic module parameters, specifying for example the width of the module ports, can be used for better reusability. One of the entities/modules is selected as the top level module, whose inputs and outputs are then mapped to the pins on the FPGA. There are languages with an even higher level of abstraction, like the Xilinx High Level Synthesis (HLS), which is based on the C programming language.

The FPGA projects in this report are prepared using the Xilinx Vivado Integrated Design Environment (IDE). It splits the FPGA design workflow into three main steps. The first step is the *synthesis*, where the HDL code is parsed and converted into a *netlist*. Within the netlist, the module code is turned into the hardware objects like LUTs and a list of the physical connections between them. Syntax errors and some basic logic errors, like mismatching bus widths, are reported. Also, logic optimization is performed in this step. For example, code that has no effect on the behaviour of a module is removed.

In the second *implementation* step, the netlist objects are mapped onto the physical device layout. This step is very processing intensive, since the switching behaviour and timing of a connection is constrained by the logic it implements. An appropriate choice has to be made for both the placement of the netlist objects and for the routing of the connections. The user also needs to give the tool information it cannot recover by itself. This is done in *constraints* files. For example, the user needs to specify, which top level ports are mapped to which device pins. The user may also specify a minimum or maximum delay of a path, or in reverse, regard a path's timing as irrelevant to notify the tool, that no timing optimization needs to take place. When the tool finds a solution to satisfy all timing constraints, *timing closure* is reached. If the timing constraints are too tight, for example because the design clock frequency is too high or because the data needs to traverse too many logic stages in a single clock cycle, the tool may not be able to do this. In this case, one can still try the design on board, but depending on the severity of the problem it may only partially work or not work at all. Correctly defining the Clock Domain Crossings (CDCs), where data from one clock is processed by another, is part of the requirement to achieve timing closure. The details of the implementation, like resource utilization, timing and power usage can be reported after this step to evaluate the quality of the implemented design.

Finally, a bitstream is generated in the *bitstream generation* step. The bitstream is the raw configuration data, which gets uploaded to the FPGA. In this step, also some Design Rule Checks (DRCs) are done, which are final checks to detect and prevent bitstream generation with errors in the design.

2.6.3. Debugging resources

To debug a design before it is being uploaded to the FPGA, Vivado provides the possibility of simulating it. First, a logic-level simulation may be used to check the code for logical errors. A *testbench* module is wrapping the component to be tested and generates a stimulus on its ports. The outputs can then be asserted to be of some value. For debugging wrong behaviour, the user may add any signal/wire in the hierarchy to an oscilloscope-like view to see how the value changes over time. A similar simulation may also be done on the synthesized and implemented design, although the structural information gets lost.

For debugging the design during runtime, Integrated Logic Analysers (ILAs) may be used. They are IPs that can be either automatically injected into the design or manually instantiated. ILAs have a configurable number of inputs, which are connected to the data lines that shall be observed. During runtime, they keep storing the data in a fixed-size buffer, which is read out to a computer connected to the FPGA via a Joint Test Action Group (JTAG) [23] connection on a trigger condition. The data is displayed in a way similar to the simulation result, but only a limited number of frames depending on buffer size is visible and only for the signals, that have been connected to the ILAs before implementing the design.

The output counterpart of ILAs are Virtual I/Os (VIOs). VIOs can be instantiated in the design and used for controlling the logic through the JTAG interface, like in the ILA case. Their outputs can be set to a certain value, and their inputs can be read back. There is no time resolution on the input, though, so it cannot be used to track fast changes in the inputs.

2.7. Special ASICs used in the read-out

2.7.1. lpGBT

The low-power GigaBit Transceiver (lpGBT) is an aggregator ASIC designed by CERN for use in High-Energy Physics (HEP) experiments. On the *E-Link* side, it accepts up to 7 groups of inputs and 4 groups of outputs with 4 channels each. The data is aggregated to a single high-speed output of either 5.12 Gbps or 10.24 Gbps and an input of 2.56 Gbps. It has several modes of operation, and not all E-Link groups/channels can be used in all cases.

The speed of the E-Links is configurable in three values. The numbers are 320, 640 and 1280 Mbps for the RX groups and 80, 160 and 320 Mbps for the TX groups. In the 5.12 Gbps mode of the uplink, the E-Link RX speeds are halved. To not exceed the

available bandwidth per group, at the highest speed only the first channel and at the medium speed only channel 0 and 2 may be used. The clock driving the E-Link TX is based on the recovered clock of the high-speed input.

The high-speed links use a custom frame format, which contains in each case a synchronization header, the group data, 2 bits of Internal Control (IC) and External Control (EC) data and a FEC code. The uplink FEC code may be chosen to be able to correct either 5 errors (FEC5) or 12 errors (FEC12). In the FEC12 case, only 6 uplink groups can be used. The uplink frame has an additional latency measurement (LM) field, which repeats the value of the downlink IC. The uplink and downlink frames are scrambled and interleaved.

The lpGBT registers can be configured in 4 ways. First, the lpGBT has an I2C slave interface, to which the configuration data can be sent independent of the lpGBTs state. Secondly, the IC and EC fields of the serial line can be used, but only when the links are already configured. The IC field is for configuring the chip, to which the high-speed link connects. The EC field on the other hand is output on electrical connections of the lpGBT, to be able to daisy-chain the configuration path of lpGBTs. The fourth option is to burn a configuration into the one-time programmable E-Fuses on the chip. This is designed to be used with the IC channel: The basic configuration of the chip is done by the fuses to be able to use the link, which provides the rest of the configuration via the IC channel.

The lpGBT has many features to aid the basic functionality of aggregating the data. They are all controlled through the registers. Regarding the E-Links, the RX has configurable equalization, and the TX can use pre-emphasis. It has 28 configurable clock outputs, that can be used by other components. Three I2C masters may be used to configure such components or read out sensors. The data quality on the E-Link groups can be checked by a Bit Error Rate Tester (BERT), which works on PRBS or fixed pattern inputs. Similarly, within the datapath the group or serializer data may be replaced by test data streams like PRBS or a clock signal. The quality of the high-speed downlink data may also be measured by an Eye Opening Monitor (EOM).

2.7.2. VTRx+

The Versatile Link Plus Transceiver (VTRx+) [24] is an optical transceiver ASIC platform for use in HL-LHC experiments. Its key features are radiation hardness and a small form factor. The data rate is adapted to lpGBT speeds, so to 10.24 Gbps on the TX and 2.56 Gbps on the RX. To meet the demand of the different collaborations, one VTRx+ board has 4 TX channels and 1 RX channel. For ATLAS, all channels are used in the

Optoboard, as described in Sec. 2.2. The single downlink configures one master lpGBT over the IC field and three slave lpGBTs over the EC field. The maximum radiation dose is specified as 1 MGy and the dimensions are 20 mm × 10 mm.

2.7.3. GBCR2

The Gigabit Cable Receiver 2 (GBCR2) [25] is a small equalizer ASIC, which is used to enhance the uplink signals of the ITkPix through equalization and to prepare the downlink signals for sending by adding pre-emphasis. It is adapted to the needs within the ITk detector and to the lpGBT E-Link format. As such, it directly transceives 6 uplink channels working at 1.28 Gbps and 2 downlink channels working at 160 Mbps. It does not perform any encoding/decoding or time multiplexing. The chip features are controlled through a set of registers, that are read and written through an I2C slave block.

2.7.4. RD53A

The RD53A is a prototype pixel read-out chip for testing certain design decisions before implementing the final version, the ITkPix. The chip has a pixel matrix of 192 × 400 pixels of size 50 µm × 50 µm. Common circuitry is at the *chip bottom*, separated into the Digital Chip Bottom (DCB) and Analogue Chip Bottom (ACB) part [9].

The digital part of the pixel matrix consists of 8 × 8 pixels wide *digital cores*. Within the cores, 2 × 2 pixel regions are grouped into an *analogue island* within the digital logic. There are three types of analogue frontends used (*linear, differential, synchronous*) to determine the best option for use in the production read-out chip. The digital core columns of the longer side of the chip are divided into 3 regions corresponding to the 3 types of frontend.

The purpose of the analogue circuitry is to amplify the signal of the attached sensor and compare it to an analogue threshold. The binary output of the comparator is evaluated by the digital core. Once this output signals a sensor hit, the digital logic determines the Time over Threshold (ToT) of the sensor signal, which is related to the amount of charge deposited in the pixel. The ToT values are buffered in the core logic, so that the external TDAQ system may have time to determine, if they are to be read out by sending a trigger. The time, for which they are buffered, is determined by the latency setting programmed into the global frontend registers. A new hit will start a latency timer programmed to this value. Once it runs out, either a trigger has to be given to read it out, or it is discarded. Similar to the different frontend flavours, there are two digital buffer architectures used in the chip to compare their performance. Both split the core further

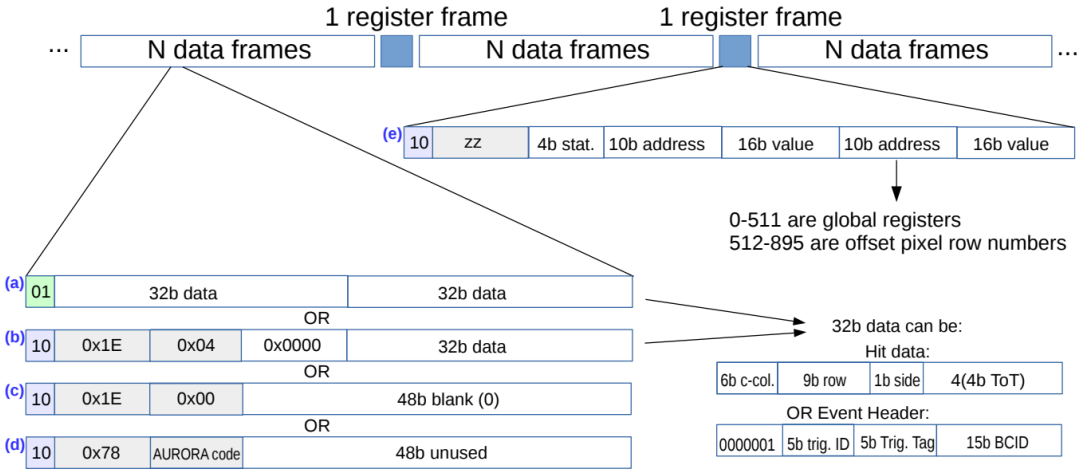


Figure 2.9.: The format of the RD53A output Aurora 64b/66b data [9] (© CERN).

into regions of either 4 or 16 pixels size, to do the bookkeeping of the ToT values and the latency timers. Instead of the comparator output, the input to the hit detection logic can also be sourced from the `cal_edge` signal. Using the calibration command, a pulse of configurable width can be generated on this signal, to fire a hit without involvement of the analogue parts or the sensor (*digital injection*). The digital core also stores the up to 8 bits of pixel configuration. The enable bit can be used to gate off the discriminator and `cal_edge` signals, so that no hits can arrive in this pixel.

The resulting information is buffered either for each pixel or centrally for the core, depending on which of the 2 digital architectures are used. On a trigger, the data is sent to the digital bottom to be combined into the full read-out.

The DCB contains the serial communication logic for both directions. On the receiver, a TTC stream (Sec. 2.5.3) at 160 MHz is accepted and decoded. Depending on the command, either the configuration memory is read/written or the calibration injection/trigger command is forwarded to the pixel matrix. The trigger commands are buffered in a 32 entries deep trigger table. The TTC stream indirectly also carries the bunch crossing clock, which is generated on the chip from the recovered clock output of the receiver CDR. On the transmitter, the Aurora 64b/66b protocol (Sec. 2.5.2) is used to transmit both hit data and service information over one to four lanes of nominally 1.28 Gbps. A diagram of the output format within the Aurora frames is given in Fig. 2.9. After the transmission of N data frames, one service frame is inserted, with the ratio N being programmable. The service frame carries a status code and two register values from either an automatic register readback or the value requested by a read register command. The data frames and the separator blocks at the end of a data stream contain 0 to 2 32-bit hit data blocks. A 32-bit block can either be the event header at the start of a triggered read-out or carry

4 4-bit ToT values of a pixel group plus the group coordinates within the pixel matrix. Empty hit groups are *zero-suppressed*, so they are skipped in the output. Thus, the data size of an event depends on the number of hit pixels. The Aurora output uses the strict alignment mode.

As a test feature, a PRBS7 stream can be transmitted on the output link instead.

2.8. FELIX project

The Front End LInk eXChange (FELIX) project is a platform for routing data between the optical links of the ATLAS detector components and the high-speed network of the following DAQ system. The project is based on an FPGA hardware platform running as a PCIe card on a host PC. As such, the FELIX project consists of software, device drivers and firmware targeting the hardware. The hardware platforms are either custom-designed, like the FLX-711 and FLX-712 boards, or fully commercial boards like the Xilinx VC709 or the Xilinx VCU128. Depending on the hardware platform, the FELIX firmware comes in a set of different flavours, which correspond to the protocol used on the optical link. For this report, the *pixel* flavour is used, which assumes a hardware structure as in Fig. 2.5. Alternative flavours are working with only an lpGBT link, or the ITk strips chain using the ABCStar read-out chip.

An image of the FLX-712 board is shown in Fig. 2.10. It provides a maximum of 48 optical links in hardware, 24 of which can be used with the current FELIX pixel firmware. The connection to the computer is over a 16-lane PCIe V.3 connection, which provides a data throughput of 15.754 GB/s. Internally, the 16-lane PCIe is switched to two 8-lane PCIe devices, each serving 12 links. In the pixel flavour firmware, the lpGBT encoding/decoding is done in the `GBTWrapper`, which has E-Link connections to the central router. Within, the RD53A uplink stream is decoded and the TTC signal generated in a separate entity is sent over the downlink. The uplink and downlink data is exchanged via DMA through the PCIe interface controlled by the Xilinx PCIe IP. Besides the link data, the status and configuration stored in the FELIX registers can also be written through the PCIe interface.

On the software side, multiple low-level command line tools have been developed to control and use FELIX. Some important ones are `flx-init`, which initializes the device after a reset or after uploading the FPGA firmware, `flx-config`, which reads/writes the configuration registers, and `flx-info` to see device information. Besides device version and sensor status, `flx-info` also shows, which lpGBT links and which E-Links within are currently aligned (receiving the correct synchronization headers). To manually send

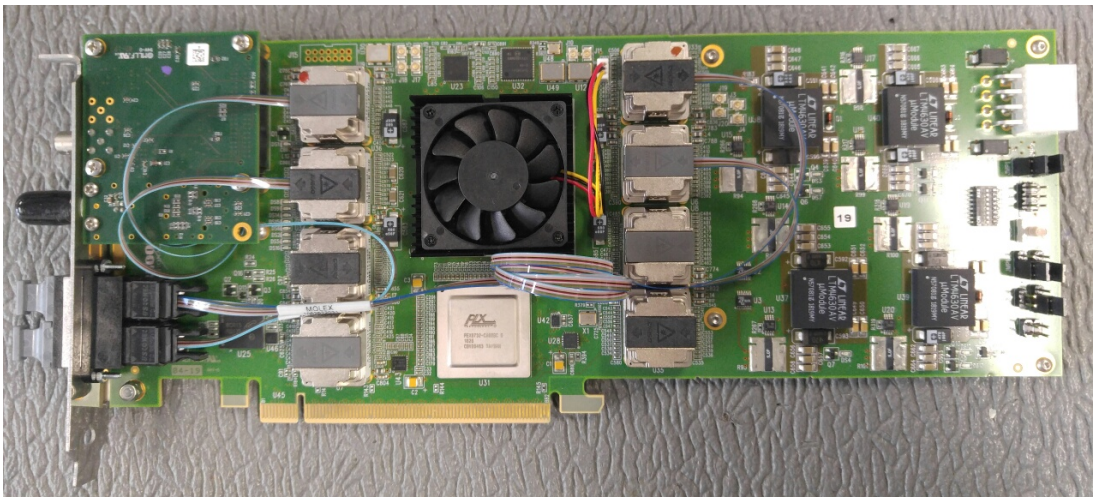


Figure 2.10.: An image of the FLX-712 custom FELIX hardware. Half of the miniPOD fibre optic transceivers are used, corresponding to the 24 of the maximum 48 available links. On the front panel, one MTP24 connector per FELIX device is exposed to the user.

data over one E-Link, the `fupload` tool can be used. In the uplink direction, the link data can be dumped using `fedump`, if both link and E-Link are locking. The main tool for the read-out, however, is `felixcore`, which opens a network connection over which the FELIX data is being sent [26]. NetIO [27] is used as the network communication library in this regard. In the final design, FELIX is meant to be controlled via RDMA built on InfiniBand connections.

2.9. YARR

Yet Another Rapid Readout (YARR) [8] is the DAQ software candidate for the ITk detector. Its premise is to perform most of the frontend output decoding in software, and not in hardware. In this case, the hardware is only used as an I/O bridge between the software and the frontends. Since the software includes all the decoding logic, it does not need to be redeveloped for the specific target platform. In fact, YARR supports multiple different FPGA-based backends. The one used in this report is the felixcore/NetIO backend. As described in Sec. 2.8, running felixcore on a FELIX server opens NetIO network ports as an interface to FELIX. With the felixcore/NetIO backend, YARR connects to the opened ports and uses them to send and receive the frontend data.

YARR is very flexible in what it can do, so 3 configuration files are used to make YARR aware of the concrete system one is using. The first one is the controller file, which is used to specify the backend and its settings. For felixcore/NetIO, the settings are for

example the port and IP of the felixcore instance. A second one is the connectivity file, where all frontends in the setup are listed. For each frontend, the RX and TX channels are specified and a path to a JSON file is given, where the frontend register values are stored. There needs to be one such JSON file per frontend. The RX and TX channels are integer numbers, and their interpretation is dependent on the backend. For the pixel flavour of FELIX, the value for an E-Link is computed as $64 \cdot [\text{link}] + 4 \cdot [\text{group}] + [\text{channel}]$.

The last configuration file is the scan definition, which defines the frontend-specific actions that are performed, and the analysis that is done on the result. The procedure of a scan is defined by a list of loop actions. Each iteration of the first loop in the list performs the full loop of the second list, and so on, so that the total the number of times the innermost actions is performed is the product of the loops' iteration counts. The last actions in the list are the actions for giving the trigger commands and the one for collecting the results.

The thesis results have been obtained with the digital scan. In a digital scan, a digital calibration injection pulse, bypassing the analogue circuitry, is used to generate hits in the frontend pixels. To read out the pixels, 16 consecutive triggers are sent to also record this hit. If the chip is performing well, the responses to 15 of these triggers are empty, while one contains the injected event. The calibration and trigger commands on the TTC link have to be timed according to the latency configuration of the chip. In the official digital scan template, this sequence is performed 100 times by the innermost `TriggerLoop` action. It is wrapped first in the `CoreColLoop` and then in the `MaskLoop`. The `CoreColLoop` is performed 5 times, and enables only selected pixel core columns in the global chip configuration. The `MaskLoop` with 64 iterations enables only a part of the chips pixels at a time through the pixel configuration. Both loops mask a given portion of the chip exactly once per loop. Hence, the occupancy map of a perfect result, showing the number of hits per pixel over the whole scan, will be 100, from the 100 iterations of the `TriggerLoop`. The total number of triggers sent to the frontend is $64 \cdot 5 \cdot 100 \cdot 16 = 512000$, though.

During a scan, the number of lost triggers is printed out for each run of the `TriggerLoop`. This is the difference between the number of sent triggers (by default $100 \cdot 16 = 1600$) and the number of received event headers.

From the user perspective, a scan is performed by running one instance of felixcore per FELIX device used. For each spawned felixcore instance, the YARR binary is executed with the aforementioned JSON files as program arguments, detailing the setup of the frontends in the assigned felixcore instance. After the scan is performed, the YARR program exits and the analysed results, for example the occupancy maps, are written to

the specified output directory.

3. Experimental setup

3.1. Project overview

In a laboratory environment, a scan chain for ITk pixel read-out development usually consists of the FELIX server and hardware, an lpGBT development board including the VTRx+ and a frontend ASIC development board or module. The lpGBT development board can be for example the VLDB+ [28] or an Optoboard. A setup using the VLDB+ is shown in Fig. 3.1. Optical fibres connect FELIX to the VTRx+, and the connections between lpGBT and the frontends are done electrically using a breakout board to, for example, SMA or mini-DisplayPort connectors. This setup is sufficient to test and develop most features of the components, where only one or a few frontends are used. For a system or stress test of FELIX with full link population, however, a lot of lpGBTs and frontends are needed, which would be very impractical and resource intensive to have in a normal laboratory. The biggest problem is the large number of required frontends, which depends on configuration, but is at least 48 assuming 2 frontends per link.

The project idea is, to make the setup of a fully populated FELIX more accessible, by emulating the lpGBTs and frontends using FPGAs. The assumption is, that multiple lpGBT and frontend instances can fit on a single FPGA board, so that the full system only consists of few of them. The frontend is chosen to be the RD53A instead of the ITkPix/RD53B, since the RD53B is not supported by the FELIX/NetIO/YARR scan chain currently, and since RD53A emulation was already available prior to the project. The high-speed links are realised by the Multi-Gigabit Transceivers (MGT) on the FPGA in conjunction with off-chip optical modules. Commercially available boards usually offer Small Formfactor Pluggable (SFP) connectors to connect these modules. A variant of the SFP connectors are the Quad-SFP (QSFP) connectors, which offer 4 links through the same cage instead of 1. Because each lpGBT emulator needs a high-speed link, the number of lpGBT emulators per board is limited by the number of (Q)SFP cages. For simplicity, the RD53A emulators should be on the same board as the lpGBT emulator they are connected to. The connections between the emulators are then done in the FPGA fabric.

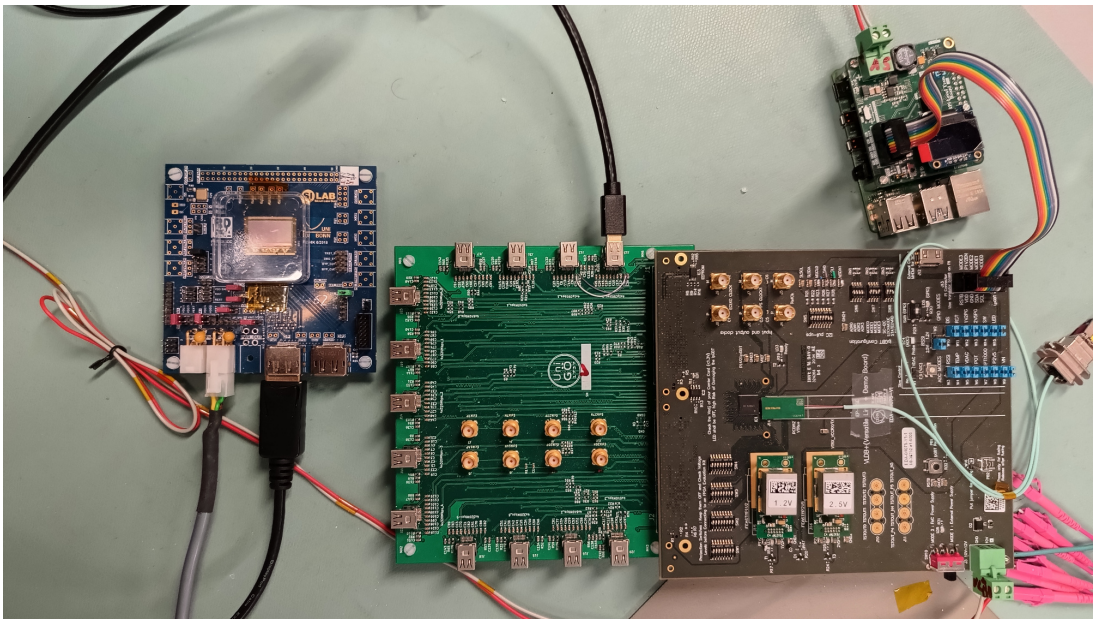


Figure 3.1.: A typical laboratory setup for FELIX/lpGBT/RD53A read-out development. The RD53A ASIC is embedded on a Single Chip Card (SCC) on the left. The lpGBT and the VTRx+ are contained on the VLDB+. Furthermore, a breakout board to mini-DisplayPort connectors and the piGBT toolkit for configuring the lpGBT via I2C are present.

As the target platforms, originally the Xilinx KCU116, the Xilinx VC709 and the Xilinx VCU128 were proposed. Using the onboard (Q)SFP connectors, 4, 4, and 16 fibres can be connected to the boards, respectively, adding up to the total of 24. In the current setup, the VC709 is replaced in favour of the KCU116, whose natural capabilities are extended by adding another 4 SFP connectors via an FPGA Mezzanine Card (FMC). This change eases the development of the setup, as only two platforms have to be targeted this way. With this, the KCU116 needs to host twice the originally estimated logic. Only the KCU116 can be chosen as the replacement, since the MGT channels connected to the FMC connector cannot access the necessary clocking resources on the VCU128. The basic structure of this setup is sketched in Fig. 3.2.

The two emulator projects are described in the following Sec. 3.2 and Sec. 3.3. During development, the need arose to implement a central, high-speed configuration path to the boards. This requires introducing a processing system, its code and controller software to the project. These three components are described in Sec. 3.4 to Sec. 3.6. The final setup is then shown in Sec. 3.7.

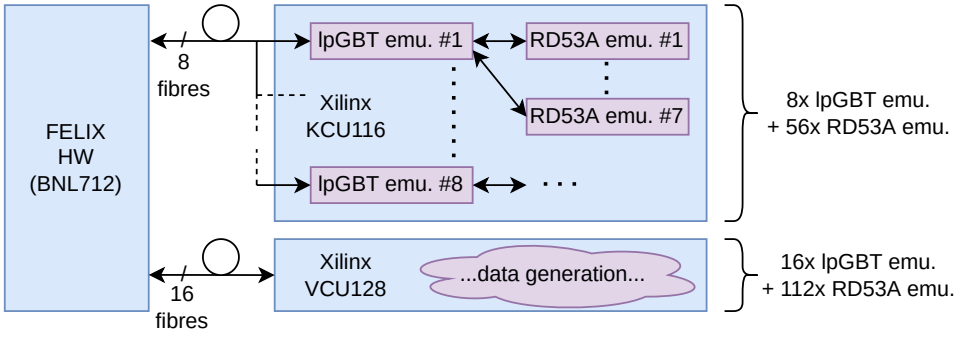


Figure 3.2.: A diagram sketching the basic idea and setup of the project. The design in the VCU128 has the same internal structure as the one in the KCU116.

3.2. lpGBT emulator

Originally, the lpGBT emulator this project is based on was developed as a stand-in replacement for the lpGBT ASIC, since it was not available at the time. The datapath blocks, so the de-/scrambler, de-/interleaver and FEC decoder/encoder, were also implemented from scratch, due to not being officially available. In this version, the Ohio FMC breakout board was used to electrically connect the frontend ASICs to the FPGA board.

In the current version, the only other lpGBT related blocks besides the core datapath are the register map and the I2C slave for configuring it. The register map is only partially implemented, and most of the registers control blocks of the previous version, which are now removed. Still, some settings regarding for example the high speed link remain relevant, and the registers have to be configured for the design to work. The I2C slave behaves like the I2C slave of the lpGBT ASIC. On the KCU116, it is connected to a PMOD connector on the board. On the VCU128, it is removed from the design, as no usable free connector on the board exists. There, the configuration has to be written via the processing system described in Sec. 3.4.

Besides the register map, some lpGBT settings can be changed in a VIO panel, which needs to be accessed by connecting the board to a PC via JTAG. These settings include the FEC and uplink speed mode, and flags to bypass the individual datapath blocks. By default, the FELIX FEC5/10.24 Gbps configuration is selected and the datapath blocks are active, so normally these settings can be left untouched.

Both the uplink and downlink part of the datapath run at 40 MHz, with the uplink datapath being 256 bits and the downlink datapath being 64 bits wide. Since the interface to the MGTs is selected to be 64 bits wide, the uplink frame is converted into 64-bit chunks running at 160 MHz before being fed into the MGTs. The interface to the E-Links are the 7 32-bit uplink and 4 8-bit downlink E-Link groups.

As required by the stress test, the design is configured to use multiple instances of the lpGBT datapath. The register map is shared between all these instances. For each lpGBT emulator instance, also a virtual E-Link block is generated and connected to it, which hosts 7 RD53A emulators. For the uplink, the 1.28 Gbps configuration means, that 7 uplink lanes, corresponding to the groups, are available. They are connected in order, so group 0 to RD53A emulator 0, group 1 to RD53A emulator 1 and so on. On the downlink side, 8 160 Mbps lanes are available, which are placed into the lane 0 and lane 2 of each of the 4 groups. Here, the connections to the emulator are also in order, so RD53A emulator instance $0, 1, 2...$ is connected to $(\text{group}, \text{lane}) = (0,0), (0,2), (1,0) \ldots (3,0)$. This means, that none of the downlinks are shared between two emulators. The number of lpGBT emulator instances, and the number of RD53A emulator instances within one such virtual E-Link is configurable before synthesizing the design. Using the single instance configuration decreases the build time from ≈ 1.5 h to ≈ 15 min with respect to the full build, so this is useful for debugging issues present in the individual instances.

For establishing the 10.24 Gbps/2.56 Gbps high-speed data link, the Xilinx Multi-Gigabit Transceivers (MGTs) built into the FPGA are used. There are different MGT types depending on the FPGA series. The FPGAs on the two boards used here contain the GTY [29] transceiver version. The transceiver channels are organized in so-called Quads. As the name suggests, each Quad contains 4 channels made of one TX and one RX part, as well as a Channel PLL (CPLL) clock buffer. For the whole Quad, two Quad PLLs (QPLLs) called QPLL0 and QPLL1 are available. The clocking of the RX and TX part of a channel may be independently based on either one of the QPLLs, or the CPLL of the same channel. The QPLLs are driven by externally supplied clocks. Each Quad has two such reference clock inputs. A QPLL may use either of the reference clock inputs of its own Quad, or the ones of the neighbouring Quads up to a distance of 2 Quads. For a given link speed, only a set of specific reference clock frequencies as dictated by the QPLL/CPLL design can be supplied. As highly specialized hardware, the MGT clock and high speed data connections are directly wired to the FPGA outputs, and cannot be accessed from the FPGA fabric. As such, in our case the design of the development boards dictates, which Quads are routed to the (Q)SFP cages, and in turn which reference clock resources are available to drive them. The different parts of the MGT can be directly instantiated in the logic as primitives. It is much more convenient, however, to use the Transceiver Wizard IP [30], which then instantiates, configures and connects these parts based on the entered high-level settings. Using the IP, the low speed user data is connected to the rest of the design through busses, with a width of 64 bits per channel per direction in this project. For all receiver and for all transmitter channels, so

independent of the Quad, one *user clock* is exposed, to be used with the user data bus.

A crucial part of the lpGBT emulation is the correct control of the high speed link. One complication of the design is the ITks requirement to operate the lpGBT and frontend chips based on the recovered FELIX clock. For the lpGBT emulator this means, that the clock of the data on the receiver side needs to be recovered to then clock the datapath and also the high-speed transmission. To achieve this, one fixed-frequency reference clock source, and a reference clock source depending on the recovered clock is needed. As mentioned in the MGT explanation, the reference clock sources are off-FPGA chips. In both boards, the same two chips are available to perform these tasks. The first chip is the Si570 [31] programmable fixed-frequency clock generator, used to clock the RX part of the MGTs. The second chip is the Si5328 [32] jitter attenuator, which takes two clock inputs and may output jitter cleaned and frequency multiplied versions of them on two clock outputs. The setup and configuration of the MGTs is different between the boards, and shown in Fig. 3.3.

On the KCU116, one whole Quad is connected to the on-board SFP cages (one channel per SFP), and the neighbouring Quad is connected to the SFPs on the FMC. A single Si570 reference clock at 320 MHz is used to clock each channel RX via the two QPLL0s of the Quads. In the RX, the CDR is performed, so that the user clock on the MGT Wizard IP running at nominally 40 MHz is related to the frequency of the incoming signal. The user clock, which is accessible from the fabric, is output on the FPGA pin, which is by the board design wired to the first input of the Si5328. In the Si5328, the input clock is jitter cleaned and frequency multiplied to 320 MHz. This adapted output clock is routed back into the FPGA through the board, and used as the reference clock of the channel TXs via QPLL1.

On the VCU128, each QSFP is connected to a transceiver Quad. The Quads for QSFP1 to QSFP3 are neighbours and can access each others clocking resources. These consist of one Si570 per QSFP, and the two Si5328 outputs. The Quad for QSFP4, however, does not have access to them. The QSFP4 can only access its Si570, the SMA connectors on the board and the (unpopulated) FMC clocking lanes. Therefore, the VCU128 structure had to be adapted by using two MGT Wizard IP instances. The first instance for QSFP1 to QSFP3 uses the same clocking structure as in the KCU116, meaning one Si570 (the one of QSFP2) clocks the RX part, which in turn drives the Si5328, which then drives the TX part. The TX user clock of this main Wizard IP, which is accessible to the fabric, is routed to the differential SMA output connector pair on the board. From there, two SMA wires loop the clock back into the board on the SMA reference clock input accessible to the QSFP4. This SMA reference clock is used to drive both the RX and the TX part of

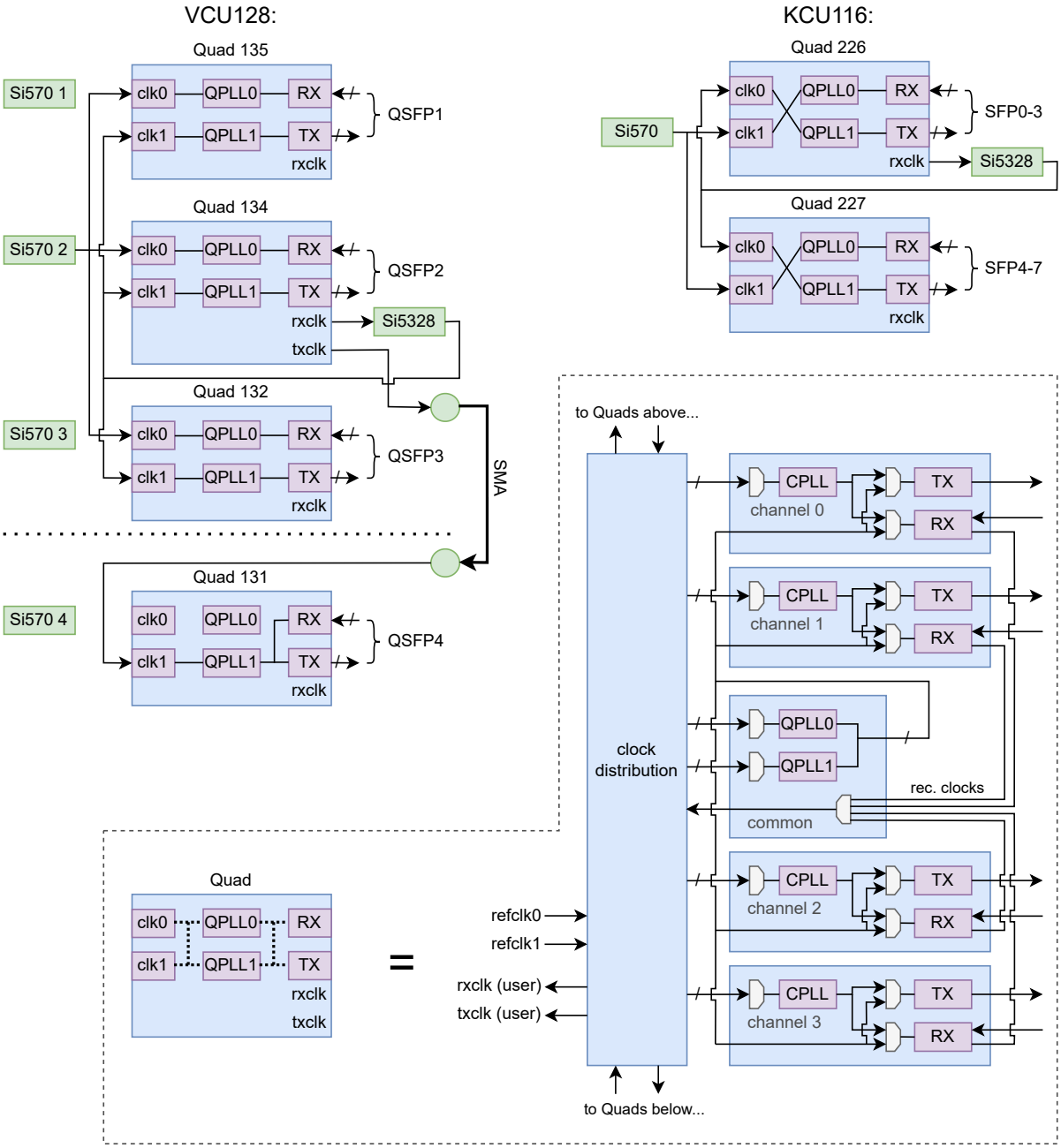


Figure 3.3.: A diagram showing how the MGTs are set up to achieve the required clock recovery scheme for the lpGBT emulator. To save space in the connectivity diagrams, the details of the basic building block, the MGT Quad, are shifted to the legend on the bottom. The MGT Quad diagram is based on [29].

the QSFP4 MGT IP. The input and output user data of the QSFP4 MGT IP is manually synchronized to the user clocks of the main Wizard IP, to be able to use one clock per direction for all the data.

The MGT requires the reference clocks to be stable before lifting the TX and RX reset. The correct start-up and reset procedure is therefore to first lift the reset on the RX part of the (main) MGT Wizard IP after the relevant Si570 has been configured. This leads to RX user clock operation, and in turn to a clock on the Si5328 output. When the Si5328 output is stable, as indicated by the Si5328s Loss Of Lock (LOL) register, the reset of the TX part of the (main) MGT Wizard IP may be lifted. For the VCU128 only, in a third stage the reset of the secondary MGT Wizard IP for QSFP4 is lifted. Reaching a stable Si5328 output can take a few seconds in the case of the VCU128.

3.3. RD53A emulator

3.3.1. RD53A emulator specifications

The RD53A emulators job is to fill the lpGBT E-Links with data mimicking the output of the RD53A ASIC. For this purpose, the abstracted view of the emulator is a black box taking the TTC command stream (Sec. 2.5.3) as inputs, and outputting the RD53A-specific Aurora 64b/66b (Sec. 2.5.2) data. In the FELIX 10.24 Gbps/FEC5 lpGBT use case, a 160 MHz downlink lane occupies 4 bits and a 1.28 Gbps uplink lane occupies 32 bits in their respective lpGBT frames. This is therefore the format, in which the data leaves/arrives at the emulators with a rate of 40 MHz.

There are several requirements the stress test has on the RD53A emulators. The simplest requirement is, that the RD53A emulator adheres to the described input and output format, and outputs data compatible with FELIX. With the FELIX version used in the current laboratory setup, this means a restriction to one-lane output. A requirement on the logic side is the ability to have an event generator, which is configurable in at least the event size. Since the received FELIX bandwidth is related to the trigger rate times the event size, and the trigger rate is fixed in the ITk specifications, this is the main way to control the load on the FELIX hardware/software. On the implementation side, the design needs to be lightweight. Due to the one-lane output requirement, the $24 \cdot 7 = 168$ frontend E-links of FELIX require us to also instantiate 168 RD53A emulator instances for full population. An increase in resource utilization in the RD53A emulator code is consequently amplified 168-fold across all boards. A hard limit on the resource usage of the whole design is given by the FPGA size on the boards that are used. But also below

this hard limit, saving resources is advisable, since it reduces the implementation time of the design and makes it easier for the tools to reach timing closure.

3.3.2. RD53A emulator design

RD53A emulator structure

The starting point of the development is a stand-alone RD53A emulator [33] targeting a Xilinx KC705 board. As a stand-alone project, the design includes not only the core emulator logic, but also components for managing the board and the input/output de-/serializers. The output format is fixed to 4 lanes running at 640 MHz. The events consist of a fixed pattern generated by hard-coded logic, which is always sent for each received trigger. A full RD53A register map (without the pixel configuration) is stored, but the logic behind the registers is not always implemented. From the difference between an lpGBT emulator design with and without this RD53A emulator, a resource usage of $\approx$ 5000 LUT and 5.5 36-kbit BRAMs is seen for the core emulator logic of this version.

For the currently used RD53A emulator, the core RD53A logic is reimplemented, but some of original emulators code is kept. The reused parts are the blocks concerning the Aurora 64b/66b encoding, so the 66- to 32-bit gearbox and the scrambler. This decision is based on a mismatch between the features of the original emulator and the specifications needed for the project described in the previous section. In order to adapt the emulator to our use case, the output would have to be reduced to one lane output at 1.28 Gbps, the hit generation had to be reimplemented, the TTC alignment would have needed change, the full register map could be removed, and as later turned out, the calibration command evaluation would have to be implemented. Furthermore, some structural changes would allow to save some resources, in particular the BRAMs used as buffers in the original emulator. All in all, a reimplementation of the core RD53A logic took relatively low time in exchange for high gain in terms of usability for the project. The two clock domains used in the original emulator are merged into a single 160 MHz one for convenience.

The output structure of the (new) RD53A emulator, as seen in Fig. 3.4, is very closely related to the RD53A output format diagram shown in Fig. 2.9, in the sense that one entity is responsible for each level in the nesting hierarchy. Upon reception of a trigger, its information is stored in the `Trigger table`, given that it is not full. When the `Data generation` block is not busy, it picks up the new trigger from the `Trigger table`, and outputs first the event header, and then the event data to the further entities. This is done in the form of the same 32-bit blocks, which are later embedded in the Aurora frames. The event header block is generated in the `Data generation` itself, whereas the event

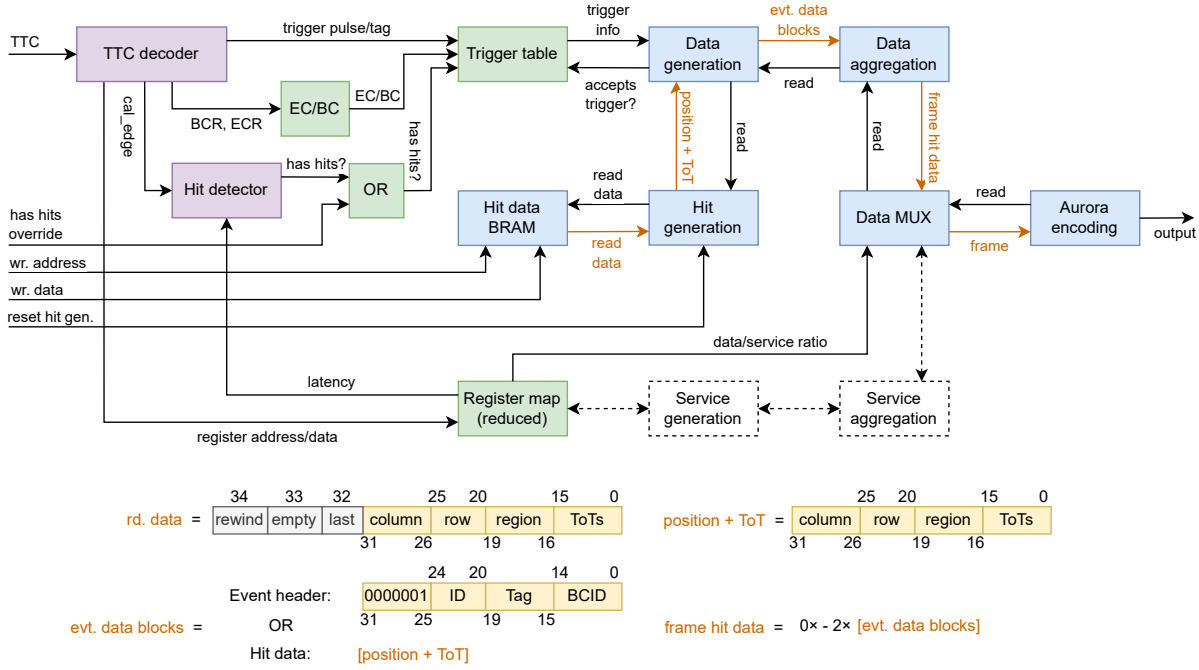


Figure 3.4.: The top-level structure of the RD53A emulator tailored to the stress test. The output data path is drawn in orange, with added details on the data it transports. Blocks with dashed lines are foreseen in the structure, but not implemented.

data is streamed from the `Hit generation`. A further difference between the two is, that the `Hit generation` does not have any trigger information, but only knows when a new event has started.

The number of blocks embedded in an Aurora frame dynamically ranges from 0 to 2, depending on the amount of data that is still to be sent. For the frame building it is convenient to have these blocks ready at the same time, so the 32-bit blocks from the `Data generation` are first aggregated by the `Data aggregation`. The `Data aggregation` requests the maximum number of blocks per frame over all lanes (so 2 times the number of lanes) from the `Data generation`, which may or may not have new data to send, depending on whether there are still some unprocessed triggers. The number of available blocks and the block data itself is then used by the `Data MUX` to build the Aurora frames for all lanes. According to the number of available blocks, it selects the right header and frame type. It also takes care of interspersing the data frames with the register/service frames in the programmed ratio (called N in the figure). The output is then encoded by the Aurora encoding blocks taken from the original RD53A emulator. At the moment, the service frame reports only the 4-bit status value. The register readback fields are tied to a constant value. This is done, because only a minimal set of registers is implemented

anyway, and the used YARR version does not require them in order to successfully perform a scan. In principle, the generation of the service frames would follow the same structure as the one for the data frames.

Due to the uneven ratio of 66 to 32 in the gearbox, after every 16 66-bit frames the data generation needs to be paused to send out the 32-bit accumulated excess data from the previous frames. Similarly, data generation needs to be paused during transmission of the service frame. Therefore, for every data stream going towards the output, a signal with the suffix `_read` goes in the opposite direction. It is pulsed by the receiver to signal reception of the currently output value and start generation of new data. The new data needs to be supplied within 1 160 MHz clock cycle for the `Hit generation` and the `Data generation`, and within 8 clock cycles for the `Data aggregation` and `Data MUX`. This ensures, that the system is in principle scalable to the currently not implemented 4 lane output case, in which 4 Aurora frames at 20 MHz, so in the worst case one 32-bit block at the 160 MHz clock needs to be supplied. This is also the reason for choosing the clock frequency of 160 MHz for the emulator.

On the input side, the `TTC decoder` block takes whole 16-bit TTC frames as input, which are required to be already aligned. The `TTC decoder` decodes the Bunch Counter Reset (`BCR`), Event Counter Reset (`ECR`), Calibration Injection (`Cal`) and Write Register (`WrReg`) commands as specified in Sec. 2.5.3. The bunch crossing and event counter are kept track of in the top-level emulator architecture and are reset by `BCR` and `ECR`, respectively. The `ECR` also clears the trigger table, as specified in the RD53A manual. As mentioned, the command for writing to the registers is implemented, but not the one for reading them. The reason for implementing the write register command is, that some read-out relevant settings are given through this configuration path. On the other hand, the read register command is not implemented for the same reason the register part of the service frame, which would be controlled by this command, is constant (see above).

As far as the register map is concerned, only 2 read-out relevant global registers of the RD53A ASIC are implemented. The first one is the ratio between data and service frames (register 60), which changes the maximum data bandwidth. The second one is the latency setting (register 500), which is required for a digital scan. Beyond the global registers, the RD53A ASIC also stores pixel configuration. This is also left out. Firstly, because the source of hits is fundamentally different between the ASIC, which has a physical pixel matrix, and the emulator, which echoes hits from a list. Secondly, it would take too many resources to store it. The most important consequence is the absence of an enable mask in the emulator. Usually, a digital scan masks off parts of the pixel matrix to scan the full chip in multiple mask stages. In the emulator, in each mask stage all hits are reported,

which increases the occupancy seen in the digital scan times the number of mask stages.

The `Cal` command generates a pulse with the specified delay and width on the `CAL_edge` signal. Analogue injection is not emulated, so the `CAL_aux` signal present in the ASIC is not implemented. The rising edge of `CAL_edge` starts a latency timer in the `Hit detector`, which counts down starting from the latency value programmed in the global register 60. During the bunch crossing, in which the timer runs out, the `Hit detector` signals that arriving triggers have hits. This signal can be forced to `1` by the user, in which case all incoming triggers have hits. Having hits in this case means, that the next event from the hit generator is sent as a response to a trigger (the event still may be empty, though). Not having a hit means, that an empty event is sent and the hit generator remains in its state. The latency timer behaviour is thus different to the one in the ASIC, which has multiple timers for regions of pixels, again due to the emulator not having a pixel matrix. The mode, in which the calibration circuit is not overridden, shall be used to perform digital scans, since in a digital scan the calibration command is used to trigger a hit. As long as only one calibration command at a time is given, which is the case for a digital scan, the difference in latency counting to the ASIC does not matter. The override mode can be used to mimic the behaviour as in detector operation, where the hits are of external origin instead of being artificially generated by the `Cal` command. In this case, the latency counter of the emulator is not involved at all. This is not a problem, since the hits from the BRAM do not correspond to physical events, and thus the timing between trigger and hits is irrelevant. A minor difference to the real ASIC is, however, that there are no ToT buffers in the emulator, which could overflow and hence introduce dead time.

Every incoming trigger is buffered in a `Trigger table` of depth 32. An entry consists of the values of the bunch crossing and event counter during reception of the trigger, the trigger tag given in the trigger command and the information, whether the trigger response is coming from the hit generation or is empty (whether it “has hits” or not). This information is used in the output part of the emulator to generate the response. The `Trigger table` is configured to use shift registers instead of BRAMs, to save resources.

Currently, the new RD53A emulator utilizes slightly under 1000 LUTs and uses no BRAMs in the logic itself. Thus, it significantly reduces the resource usage with respect to the original one. Of course this comparison cannot be made directly, since the original emulator serves four output lanes instead of one. But even in the extreme case where we assume a one lane adaptation of the original to be only a quarter of the size, which it would most probably exceed due to the Aurora output blocks being only a part of the logic, the new one is smaller.

3.3.3. Hit data generation

The requirement of being able of generating a new data block every clock cycle severely limits the choices of hit data generation schemes. Two candidates are considered, which both satisfy this requirement and provide a configurable event size.

For the first one, the pixel group positions, which are given by a 3-dimensional coordinate (row, column, region), are mapped onto a single index from 0 to 19199. The hit generator then places hits according to a user-defined average distance $\langle d \rangle$ between them on this index. The average distance is a floating point value, and is achieved by incrementing the hit position in steps of size $\lfloor \langle d \rangle \rfloor$ and $\lfloor \langle d \rangle \rfloor + 1$, in a way which minimizes the error to the non-discretised case. If a step goes beyond the range of 19200, the remainder is carried over to the new event. By setting the average distance, the user can specify the average event size, and thus select the payload size generated from this one emulator, as wanted for the stress test.

In the second hit generator, the hit data is directly streamed from BRAM blocks. In this configuration, the RD53A-specific encoding of the hit data into the 32-bit blocks is performed offline by software. The resulting data is then transferred to the BRAMs, either fixed during synthesis of the design or dynamically via some loading logic. The hit generation logic can freely index into the BRAM memory. On a hit, it first sends out the 32-bit data block at address 0, then 1 and so on. By including a special value as an event separator, which indicates the event boundaries within the data, also multiple hitmaps can be stored. The *hitmap* name is given due to the control over the pixel hit positions. Technically, it is not quite accurate though, since the information stored in the BRAMs is not the Boolean information on whether a pixel is hit, but rather its ToT. After all events are sent, the BRAM address is reset to 0, meaning the stored events are sent cyclically. In this configuration, the payload size is adjusted by letting the offline software generate events of different size. Since the event generation in the offline software is decoupled from the data output, the offline software has time to build arbitrarily complex event hitmaps.

In a stress test setup and using the BRAM streaming, having the BRAM events only written during synthesis is not really an option, since changing the load on FELIX would require rebuilding the whole hardware design. This means, that in selecting BRAM streaming as the hit data generation one also needs a configuration path to the boards BRAMs to upload the event data. This path also should be relatively fast, as the total available BRAM size between the boards is $\mathcal{O}(100\,\mathrm{Mb})$. For this reason, in total the first generator is easier to implement and configure, whereas the second one provides more flexibility in the data sent. In the end, due to it being the more general solution, BRAM streaming is chosen as the hit data generator, even though it requires implementing such

a configuration path. Of course, even with a configuration path present one can initialize the BRAMs to a default set of hit data before synthesis.

In the implementation, the BRAMs form a dual port RAM with a width of 35 bits. The lower 32 bits are carrying the data, while the upper 3 bits are status bits signalling the last hit block of an event (bit 32), an empty event (bit 33), or the end of the usable data within the BRAM ("rewind", bit 34). In the case of an empty event, the lower 32 bits are meaningless. While the RAM can access a new memory location every clock cycle, there is a read latency of 2 clock cycles. This means, that the address pointer needs to be all the time two places ahead of the actually used data. To ensure this, at start-up or reset there is a preload phase, where these two blocks are buffered before the module is actually usable. Also, the rewind bit is not set on the last data block, but rather at the data block two steps prior. This leads to a minimum size requirement of 3 blocks. One can still send only one event with one hit or just empty events though, by just replicating the event 3 times or padding with empty events, respectively. The read port, which is used in the hit generator, and the write port exposed on the emulator top level entity may be accessed from different clock domains. Technically, writing the hit data while the read-out is in progress is possible, but there is currently no logic in place preventing corruption of the currently read data due to the write operation. This could for example lead to old and new events merging during the read-out or similar glitches. For this reason and due to the preload of the data, the `Hit generation` block needs to be reset after writing new data.

Currently, the KCU116 can store up to 6656 blocks and the VCU128 can store up to 16896 blocks of hit data per RD53A emulator instance. This results in a BRAM utilization of near 100 % on both boards, when the design is configured to use the full number of emulator instances. In the case of the KCU116, this means using 6.5 36-kbit BRAMs per instance, which would have been reduced to only one 36-kbit BRAM when using the original RD53A emulator.

3.4. SoC design

As mentioned in Sec. 3.3, the total RD53A emulators hit data capacity is $\mathcal{O}(100\,\mathrm{Mb})$. Doing a full hit data upload through the I2C configuration path running at 100 kHz speed would therefore take unreasonably long, especially if one wants to do a stress test, where multiple scans with increasing load need to be done. Also, of the three available boards, only the KCU116 has exposed General Purpose I/O (GPIO) pin headers, to which the I2C hardware can connect. The IC configuration path at 80 Mbps would be fast enough.

It is also not an optimal solution, though, since using it to configure the RD53A emulators or other design features requires hacking the lpGBT register map, which is the target of the IC link. Also, it is not possible to use the `fice` FELIX low-level tool to read/write the IC data while felixcore is running for a scan, since felixcore locks the FELIX DMA channels. Instead of I2C and IC, the configuration is done via Gigabit Ethernet. It can be used independently of the FELIX link and it has a high data rate of up to 1 Gbps. The connector is present natively on the KCU116 and VCU128 boards and it allows a controller computer to connect to multiple stress test boards via an ad hoc local network by using an Ethernet hub. The controller computer may or may not also be the FELIX server. Also, no special or uncommon components are needed to create this Ethernet network, and many computers natively can connect to it. An Ethernet connection can be implemented in hardware only, but typically only the packet oriented User Datagram Protocol (UDP) [34] is supported in this case. In order to assure that the hit data arrives in the correct order and without losses, the stream oriented Transmission Control Protocol (TCP) [35] needs to be used, which strongly favours a processing system. As a second benefit, the processing system can be used to perform the configuration and initialization of the board components such as the MGTs, the Si570 and the Si5328.

A diagram of the processing system is given in Fig. 3.5. At the heart of it is the Xilinx MicroBlaze soft IP core representing a 32-bit RISC processor implemented in the FPGA fabric [36]. It is surrounded by various peripherals, either to complete the processing system functionality or to interface with the outside world and the rest of the design. In the first category are the Dual Data Rate (DDR) RAM controller, which drives the DDR RAM on the board, the Interrupt Controller and the MicroBlaze Debug Module for debugging the CPU code. Further, two timers are used in the design. The first one is used by the network library to provide the TCP functionality and the second one is used in the code for letting the CPU sleep for some time. The interface category starts with the Ethernet Subsystem. The high-speed data sent and received via the Ethernet Subsystem is accepted in the system memory through a DMA controller. Additionally, the Ethernet has a control interface to the CPU. A second DMA Controller is used to stream the control settings and hit data to the RD53A emulators. Besides using a DMA Controller to connect to the emulator logic, also a GPIO peripheral is used. With it, the logic state on up to 32 inputs and outputs can be individually read/written. For configuring the Si570 and Si5328, an I2C peripheral is used. Finally, the CPU code can print debug and status messages over the boards UART serial connection controlled by a UART peripheral.

The peripherals are connected to the CPU via the Advanced eXtensible Interface (AXI)

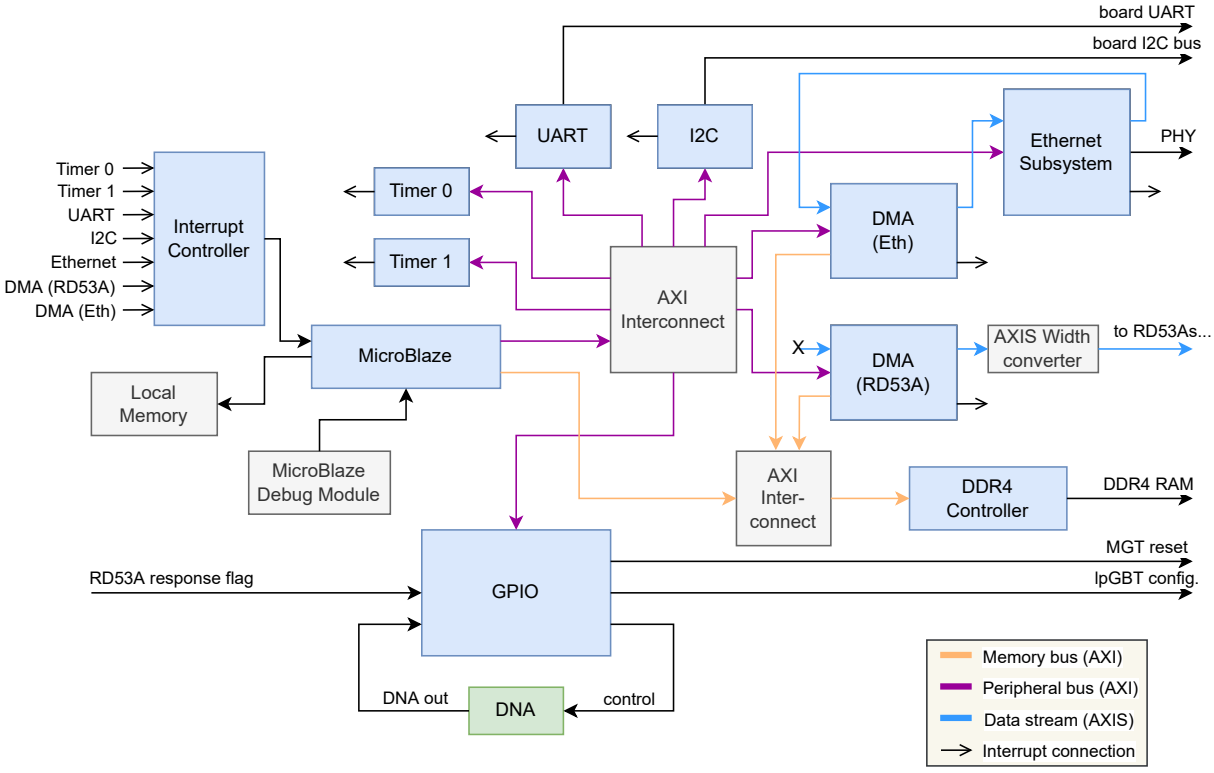


Figure 3.5.: A sketch of the processing system designed for the stress test. For the AXI, debug module and local memory connections, the arrow direction indicates a master/slave relationship rather than the direction of the data flow.

bus protocol of the Advanced Microcontroller Bus Architecture (AMBA) standard [37]. AXI switches, so-called Interconnects, are used to connect more than two peripherals to each other. There are two Interconnect paths. The first connects the CPU and the memory interface of the DMA engines to the DDR RAM controller. The second one is used for controlling the rest of the peripherals, including the control side of the DMA engines.

For the connection to the RD53A emulators, a DMA peripheral is used, because it provides significantly higher throughput than the GPIO peripheral. The DMA peripheral has two ports called Slave To Memory (S2MM) and Memory To Slave (MM2S), which are used to gather or send the system memories data, respectively. The two ports are using the AXI-Stream protocol [38] designed to carry N-byte messages unidirectionally. A special RD53A emulator controller block is needed as a logic entity, to relay the AXI-Stream commands to the RD53A emulator. Besides changing the emulator settings, also the emulator hit data BRAM is configured via such commands. Internally, the different functions are addressed through a register map, as usual. For configuring the RD53A emulators, the MM2S side of the DMA controller is distributed to all controller blocks

of the RD53A emulator instances. As such, all controller blocks receive the same data from the DMA. To still be able to address individual instances, the DMA messages to the emulators carry address information to select the target. The S2MM side is currently not implemented, where some sort of switching system would need to merge many controller block outputs into one single stream. Instead, a flag can be raised by any of the emulators. It can be read and cleared by the CPU through the GPIO peripheral. Depending on the command, the addressed RD53A controller either signals acknowledgement of the command, or whether the TTC input of the emulator is aligned. The current message format of the AXI-Stream to the emulators and the controller blocks register map is described in Sec. A.2 and Sec. A.1, but will most likely change in the future, as for example the S2MM side is implemented. In total, through the controller block the user can query the TTC locking status, enable/disable the emulator output, set the `Cal` command override (see Sec. 3.3), reset the hit generation state and of course program the hit data BRAMs.

The GPIO peripheral has four functions, one of which is control of the mentioned RD53A response flag. The second one is to perform the reset of the MGTs, as described in Sec. 3.2. Since the Ethernet configuration path is a replacement for the I2C path, also the lpGBT registers need to be written. This is the third function of the GPIO. For writing the registers, it temporarily takes over the control signals between the lpGBT registers and the I2C slave and writes the configuration received via Ethernet. Afterwards, the I2C hardware slave regains control, so the I2C configuration path can still be used. Finally, the GPIO connects to the so-called DNA block of the FPGA. On Xilinx FPGAs, this DNA block is an interface to an E-Fuse register holding a unique ID (the DNA), that has been written when the FPGA chip was produced. This DNA value is received by the GPIO to generate the boards Media Access Control (MAC) address, which should be unique on a network.

The processing system is designed as a *block design* using Vivado's IP Integrator Graphical User Interface (GUI), in which the components are shown as blocks, with lines between them representing the connections. It offers helpful automation features to generate and validate the PS. For the finished block design, Vivado automatically generates a wrapper entity, which can then be instantiated in the rest of the design (in the top level entity, in this case).

This concludes the description of the components present in the FPGA design, and finally the clocking structure of the total, top-level design (Fig. 3.6) can be described. Each FPGA board contains a fixed frequency oscillator. This clock source is used by the DDR RAM controller, to generate the 100 MHz system clock used in the PS. This includes

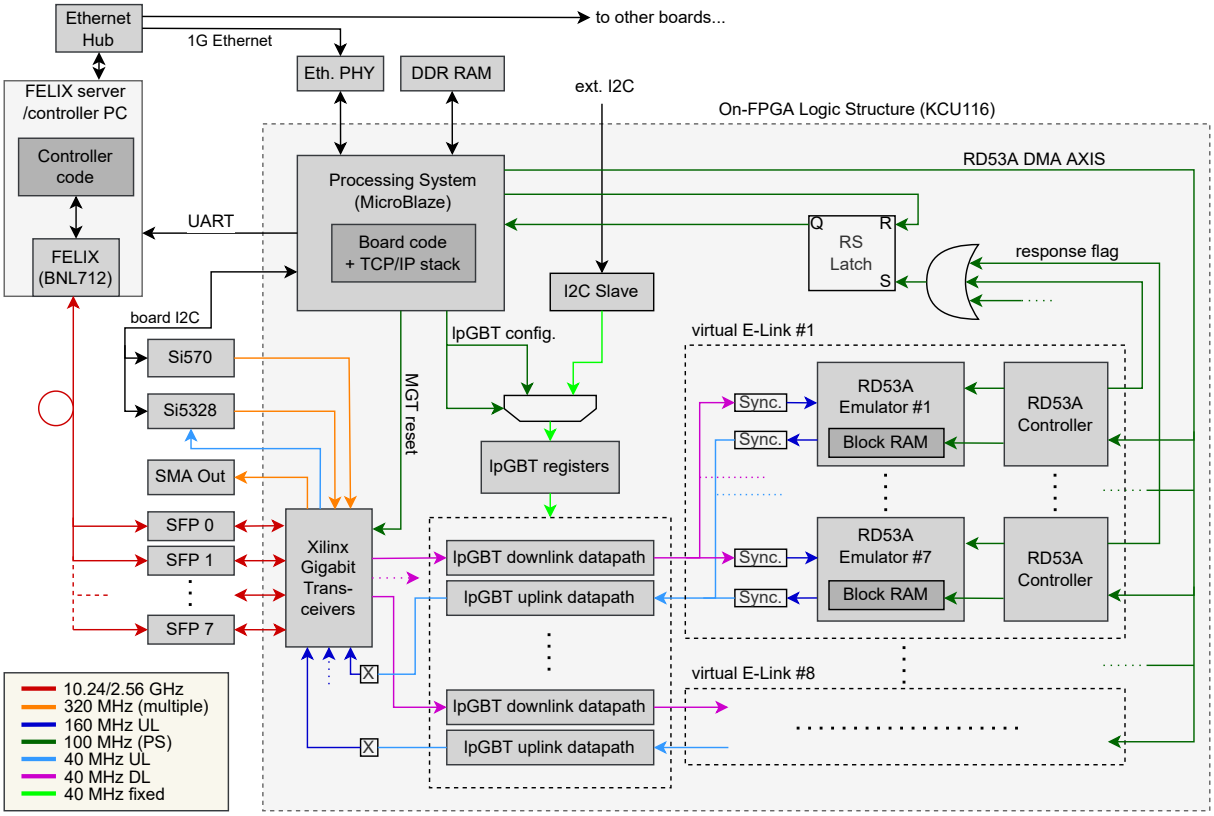


Figure 3.6.: The top-level stress test hardware design and its connections to the FELIX server. The detailed view of the RD53A emulator is already given in Fig. 3.4, and the one for the processing system in Fig. 3.5. The different clock domains are colour-coded. In contrast to the rest, the connections to the SMA Out, Si5328 and Si570 are clock, not data signals.

the RD53A-facing DMA, so that the RD53A AXI-Stream and the RD53A controller block are also clocked by this clock. In contrast to the RD53A control signals (like output enable), the hit data itself does not need any synchronization, as the BRAMs are dual port RAMs capable of using independent clocks on each side. From the 100 MHz system clock a 40 MHz system clock is generated for the lpGBT registers and as the auxiliary MGT clock. For the MGT reference clocks, the clock recovery scheme detailed in Sec. 3.2 is used. The single RX userclock runs at 40 MHz and is used to clock the downlink lpGBT datapath. The single TX userclock runs at 160 MHz and is used for the whole RD53A emulator. The TTC data of the downlink is synchronized to it, accordingly. From this TX clock, a related 40 MHz clock for the uplink lpGBT datapath and a 320 MHz clock for the SMA output are generated. The lpGBT register values and GPIO outputs controlling the behaviour of the design are synchronized to the datapath clocks as needed for each specific signal.

3.5. SoC firmware

The code running on the MicroBlaze CPU, referred to as the firmware, is written in C. The CPU is used in a *Bare Metal* configuration, meaning the code runs "directly" on the CPU without any involvement of an operating system. For development, the Xilinx Vitis development environment is used. A *hardware platform* exported from Vivado can be imported by Vitis, which adapts the firmware project to the specifications of the hardware. For example, the drivers of the various peripherals are included in the platform sources automatically. Also, a list of constants relevant to the system, for example the CPU frequency, the interrupt IDs of the different interrupt sources, the peripheral addresses etc. are generated.

The library used for the TCP communication is a version of the lightweight IP (lwIP) library [39] adapted by Xilinx for use with the Ethernet core. The firmware originates from an lwIP echo server implementation, that is available as a template when creating a new design. Starting from this template, the board initialization logic and the stress test specific network interface are added.

The initialization comprises making the driver calls to initialize the peripherals, the calculation and configuration of the Si570 and Si5328 register values and then a reset of the MGTs. The Si570 register values for a given frequency are not fixed, but depend also on calibration values flashed into the device registers from the factory. Based on these values, a new register map is dynamically generated and written to the chip over I2C. The Si5328 register values are fixed, so they can just be defined as hard-coded constants.

After the initialization, a TCP server using lwIP is started. The MAC address of the board is chosen from the FPGA's DNA value. The IP address is usually statically chosen to be `192.169.1.10` for the KCU116 and `192.169.1.11` for the VCU128, with a netmask of `255.255.255.0`. However, if the Dynamic Host Configuration Protocol (DHCP) [40] is enabled in the lwIP platform settings, the IP can also be dynamically configured by a central DHCP server based on the MAC address. This would be needed, in case the same development board type is used multiple times in a system.

The firmware code is the same for both boards. Some minor differences, for example the default IP or the layout of the I2C bus, are board dependently compiled based on a preprocessor constant.

The Ethernet interface defines 6 commands, which can be given to the board. The `Ping` command has a fixed response to check the version of the firmware. With the `Enumerate` command, the availability of RD53A emulators within the hardware including their position in terms of uplink link and group can be checked. The firmware can automatically discover, which RD53As are present, by writing a dummy command to the RD53A con-

trol module and checking through the response flag, if the message was acknowledged. Similarly, the `TTC Status` command informs the sender about the TTC alignment status of every emulator. To remotely reset the MGTs, the `Soft Reset` command is used. Writing to the RD53A emulators is done via the `Write Reg` command. With this command, both hit data is uploaded and the RD53A emulators settings are changed, since both of these features go through the RD53A control blocks register map. Finally, the whole set of (fusible) lpGBT registers is written with the `LpGBT Config` command. A detailed description of the command format is given in Sec. A.3.

3.6. Controller software

The purpose of the controller software is, to perform the tests and measurements of the system or stress test. Essentially, a test starts by setting up the boards to respond with the desired hitmaps. Then, a scan or general data taking is performed, and the results are collected and evaluated. As such, the controller software should have an interface to both the YARR/FELIX components on the data taking side and to the stress test FPGA boards on the data generating side.

The software is written in the Python scripting language, since the language allows quick prototyping and has a useful library ecosystem for creating JSON files and analysing/plotting results. General functionality related to board control and the YARR/FELIX interface are placed into a package called `stLib`. User board scripts and test scripts are programmed based on this package. The board scripts are similar to the FELIX low-level command-line tools, as each script is a tool to perform a specific function on the stress test boards. For example, there exists one script to upload an event set stored on disk to one or many RD53A emulators. These scripts can be run also on a machine other than the FELIX server. The test scripts perform the tests and measurements including YARR/FELIX. They are described in further details in Sec. 4.5.1.

In the library, the main stress test board functionality is implemented in the `Board` class. Its member functions provide higher level access to perform the board configuration and retrieve its status over the network. Two classes, `HMap` and `HMapEnc`, store the hitmaps, which can be uploaded to the boards. `HMap` stores the hitmaps in an uncompressed, 2D array format, whereas `HMapEnc` stores them as a list of encoded 32-bit blocks, according to the RD53A protocol.

The scan automation code can be thought of as a set of three parts, one for initialization, one for performing the scan, and one for analysing the results. The initialization of the scan is being performed by the `ScanSettings` and `ScanTemplates` classes. The

`ScanSettings` hold all information necessary to perform a scan, including the path to the YARR/FELIX binaries, the enabled frontends and read-out specific settings like the trigger rate. It also points to a set of template JSON files. These are the templates, from which the controller, scan and frontend configurations passed to YARR are generated. The connectivity configuration does not require a template, as it is completely generated by the code. The template files are read in by the `ScanTemplates` class. It adapts the configuration values within them according to the `ScanSettings`, and writes the full YARR setup to the temporary directory of the computer. This is especially useful, since initializing a full stress test system scan requires setting up 168 JSON files for all of the frontends. The `ScanSettings` themselves can also be loaded from a JSON file.

The scan is performed by the `Yarr` and `FelixCore` classes, which just spawn an instance of their respective programs and collect the resulting text output. The `ScanResult` class holds a summary of the scan based on the program outputs, and the `Histo2D` class holds the occupancy map histograms output by YARR. The whole system is tied together by the package-level `perform_scan` method, which takes a `ScanSettings` object, performs the scan as specified, and returns the evaluated results. It also takes care of spawning multiple instances of YARR and felixcore, when both FELIX devices are used. The scan results report the presence and count of trigger errors (like missing or surplus triggers), and can evaluate whether a frontend's occupancy map matches the hitmap uploaded to it.

3.7. Test setup

The current test setup is shown in Fig. 3.7. The FELIX server hosting the FLX-712 hardware cannot be seen in the figure, as they are below the table. On the FELIX server, the drivers and FELIX tools compatible with register map 5.0 are installed. The FELIX server hardware is based on the official recommendations, but with a slightly different motherboard (Supermicro X10SRA-F).

Connection to the FLX-712 card is done via two 24 fibre Multi-fibre Push-On (MPO) connectors, carrying the 12 TX and 12 RX channels of a FELIX device each. One such cable is first fanned out to two cables (one for TX, one for RX) carrying 12 fibres each, and then into 24 individual fibres. For cable management purposes, they are grouped into bundles of 4 links each and connected to the boards. The already mentioned 4-SFP FMC card and SFP modules supporting 10 Gbit data rates per link make up the board interfaces. For the VCU128, the QSFP module outputs themselves also need to be fanned out to connect to the FELIX links, whereas the KCU116s SFP modules can just take them

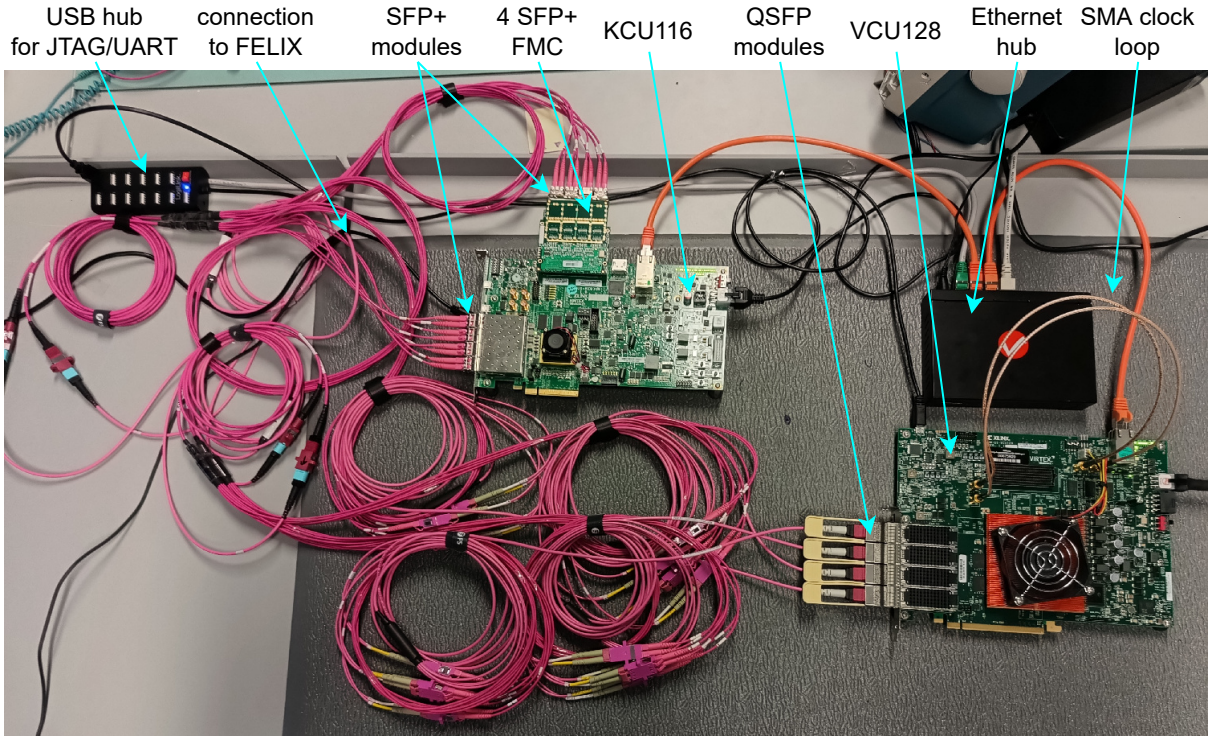


Figure 3.7.: An annotated picture of the stress test hardware setup, excluding the FELIX hardware and server, which are under the table.

directly.

Patch cables are used to connect from the boards RJ45 Ethernet connectors to an Ethernet hub. From the hub, one Ethernet cable goes to the FELIX server below the table, and one is kept above the table for connecting a laptop. The FELIX server is also connected to the boards via USB cables for configuring the boards via JTAG and for reading the UART print-outs (both done through USB). Finally, the VCU128s SMA clock loop used to clock the MGTs driving QSFP4 can be seen in the figure. As envisioned in the project specifications described in Sec. 3.1, the whole setup is very light on hardware, considering the amount of hardware it replaces.

4. Results

4.1. Overview

On the way to achieve the stress test for the YARR/FELIX read-out chain, several preparation tasks and possible solution tracks have been investigated in this work, arriving to the stress test setup detailed in Chapter 3. A wide spectrum of development tasks was required in hardware, firmware and software. This includes, in particular, adapting the hardware lpGBT and RD53A emulators to the setup implementation. For this, extensive use of the Xilinx Vivado simulation and debugging tools was needed. Another tedious task was figuring out the correct clock distribution scheme, clock domain crossings and timing optimizations to guarantee the design timing closure. The mentioned design is the SoC combining the full multi-instance lpGBT and RD53A emulation with a MicroBlaze PS including peripherals (Ethernet, UART, I2C etc.) as described in Chapter 3. Xilinx Vitis is then used to run the SoC bare metal firmware. The software part includes Python scripting tools to configure the setup and orchestrate the board functioning, as well as scripts and a software library to perform stress test runs and analyse and plot the obtained results.

Examples of the mentioned development steps are detailed in this chapter. Also, measurements of our laboratory YARR/FELIX read-out chain, as obtained by the scripts, are shown.

4.2. Achieving a stable link

Before using the RD53A emulators, real RD53A Single Chip Cards (SCCs) were connected through an Ohio board placed in the FMC connector of the FPGA board hosting the lpGBT emulator. The Ohio board adds 4 mini-DisplayPort connectors with programmable data direction to the host FPGA board. This prototype setup allowed debugging and validating the data path functioning step-by-step starting from the downlink datapath and downlink E-Links to the uplink E-Links, the uplink datapath and the uplink MGT part. The reason for this is the need to configure the RD53A via the downlink, in

order to send valid data and to enable the uplink test pattern generators.

A PRBS7 stream sent by the RD53A's test pattern generator was used to debug the E-Link RX. In the uplink datapath, the BERT block present in the lpGBT was partially re-implemented, to be able to check the BER of the connection. Using the BER as a reference measure, receiver logic based on the Alexander Bang-Bang phase detector algorithm [41] could be developed and fine-tuned. Eventually, the necessary BER threshold of $< 1 \times 10^{-8}$ % needed for E-Link alignment in FELIX was passed.

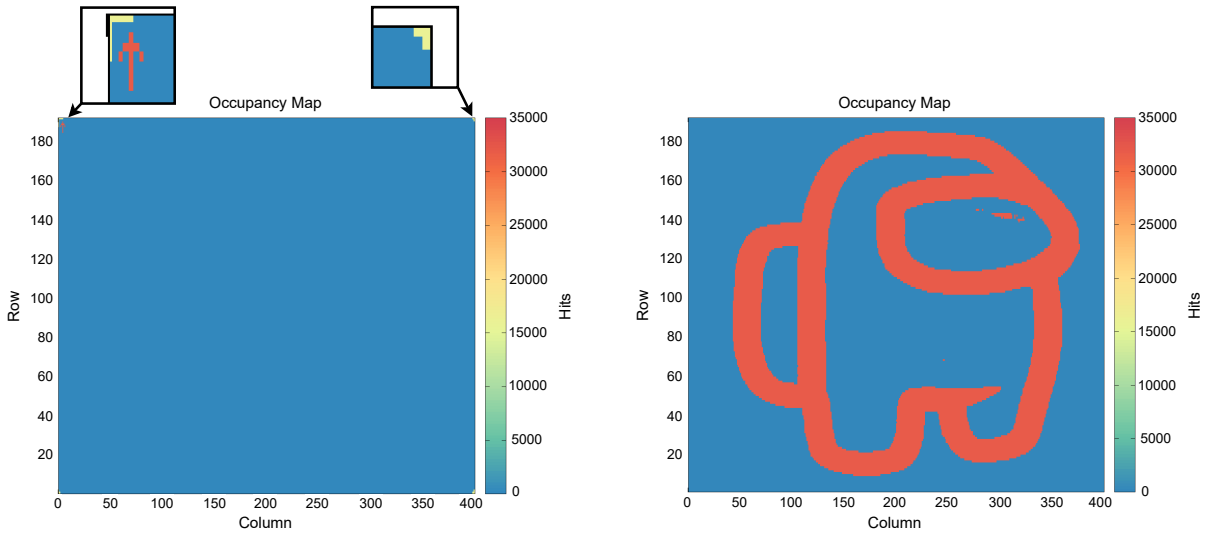
A major gain in link stability was achieved by implementing the clock recovery scheme and clocking structure as described in Sec. 3.2. At one point during development of the VCU128 port, the board clocked the MGT TX at a fixed frequency, which led to loss of RD53A alignment (using the RD53A emulators) in FELIX, despite alignment of the lpGBT frame. Changing the source clock to the recovered clock fixed this. This shows, that the clock recovery scheme is mandatory to achieve E-Link alignment, as otherwise the transmission quality is not good enough.

Finally, perfect scans in terms of YARR errors and occupancy map could be achieved with the RD53A SCC. The BERT block and the physical E-Link block are removed from the current project, though, since the ultimate goal was to use RD53A emulators instead of RD53A ASICs, and the Ohio FMC got in the way of the 4-SFP FMC.

4.3. RD53A emulator development

The RD53A emulator development started with the design of the higher-level architecture, as seen in Fig. 3.4. Block by block, the RD53A emulator was built starting from the Aurora encoding code taken from the old emulator. Each block was developed in conjunction with a corresponding testbench, testing just its functionality with the interfaces as envisioned from the abstract design. To check the integration of the individual blocks into the whole emulator, one testbench testing the top level entity was used.

After the emulator seemed to be finished in simulation, it was integrated into the lpGBT emulator and tested against FELIX. The RD53A alignment itself was quickly achieved, thanks to the adoption of the Aurora encoding code. However, the initial version of the emulator could not be scanned by YARR. This was investigated by adding debug printouts to the YARR source code. The issue was, that felixcore did not report any data to YARR, even though the FELIX low-level tool `fedump` showed it. It was solved by implementing the calibration command handling, so that not every event contains hit data. Running into this issue was unexpected, since felixcore or its YARR interface was assumed to be neutral regarding the data it publishes to the network, which apparently



(a) *(b)*

Figure 4.1.: Two occupancy maps as obtained from a digital scan of the RD53A emulator. Larger text labels and separate views of some details were added with respect to the original.

is not the case.

The first successful hitmap scans were digital scans of an event with just one hit pixel in the chip corner, which was hardcoded in the logic. From this, the BRAM streaming was implemented, with the event hitmaps being defined during synthesis. Examples showing the working BRAM streaming are shown in Fig. 4.1. The possibility of storing multiple different hitmaps for one emulator is tested in Fig. 4.1a. It shows the occupancy map of a scan of the emulator with two events uploaded to it. In the first of the two hitmaps, the shown pattern is fully present, whereas only some of its pixels (the arrow) are also present in the second hitmap. Thus, the pixels, which are only present in one hitmap, have only half of the maximum occupancy, as expected. Fig. 4.1b has a hit pattern, which would be very hard to implement in logic, showing the flexibility of the BRAM approach. It demonstrates the emulator's ability to also handle large hit patterns. In order to scan it, the YARR scan loop frequency had to be reduced to 2 Hz. With these two scans, also correct handling of the positioning of the pixels within the hitmaps is demonstrated.

4.4. Stress test development

Development of the three-part SoC began from the software side. The programs, which were used to generate the BRAM memory coefficient files for statically programming the BRAMs, were extended to include the networking code for controlling the boards.

Originally, the controller software was implemented as a shell application, where the user could give commands to the setup. However, performing more complex test scripts would have required the shell to implement some higher level language features such as loops, so this concept was abandoned in favour of a Python library, which directly offers scripting the setup using Python itself.

Since the controller software was the first part of the SoC to be programmed, there initially was no physical target to test the software against. Instead, a small test application was developed, which would host a server acting as a board on the same computer as the controller software, so that the controller software networking code can connect to it during testing. This test server was also used later as a reference during the development of the rest of the system.

The second step was building the hardware part of the SoC, meaning the PS definition and the RD53A controller block of the RD53A emulators. Similar to the approach used for developing the RD53A emulator, the components were first individually created and tested before being inserted into the lpGBT emulator project. Setup of the PS was first done in a new project without the emulators, allowing for faster iteration. In this new project, the capabilities of the lwIP example design and the functionality of the DMA controller were tested. For the RD53A controller block, validation was again done using a dedicated testbench simulation. The two parts were then integrated into the full project, replacing the hardware blocks which performed the board initialization before.

In the last step, the PS firmware was developed as a bridge between the two parts.

Scaling up the system also presented some challenges. First, the port to the VCU128 required some additional effort due to the different board layout, especially with regard to the MGTs. A second issue was the E-Link stability. Using the first iterations of the full designs, repeatedly running `flx-info` showed, that the E-Links for entire links would lose alignment in short time intervals. The problem was especially strong for the VCU128, where sometimes the E-Links on one link did not show alignment at all. Reduced versions of the board designs, where for example only one frontend per lpGBT was instantiated, did not have these issues. It turned out to be a timing closure problem, that took some time to solve as it needed several iterations, each requiring around 1.5 h to conclude. Currently, both the KCU116 and VCU128 design reach timing closure.

After treating these issues, simultaneous alignment of all links and E-Links can be achieved, as shown in Fig. 4.2. Sadly, this does not happen directly after configuring the boards, as usually some manual adjustment is needed. First, after a cold start one needs to wait until the (Q)SFP modules have reached a stable temperature. Otherwise, the initially stable configuration becomes unstable as the temperatures changes. Then,

```
lab34:~/Scripts$ flx-info link
Card type : FLX-712
Firmw type: PIXEL

Link alignment status
-----------------------
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
```

(a) Alignment of the lpGBT links.

```
E-link alignment status
-----------------------
Endpoint 0 ('*'=aligned, '-'=not aligned)
LNK 0        8       16       24       32
 0: *---*--- *---*--- *---*--- *---**-- -----
 1: *---*--- *---*--- *---*--- *---**-- -----
 2: *---*--- *---*--- *---*--- *---**-- -----
 3: *---*--- *---*--- *---*--- *---**-- -----
 4: *---*--- *---*--- *---*--- *---**-- -----
 5: *---*--- *---*--- *---*--- *---**-- -----
 6: *---*--- *---*--- *---*--- *---**-- -----
 7: *---*--- *---*--- *---*--- *---**-- -----
 8: *---*--- *---*--- *---*--- *---**-- -----
 9: *---*--- *---*--- *---*--- *---**-- -----
10: *---*--- *---*--- *---*--- *---**-- -----
11: *---*--- *---*--- *---*--- *---**-- -----
Endpoint 1 ('*'=aligned, '-'=not aligned)
LNK 0        8       16       24       32
 0: *---*--- *---*--- *---*--- *---**-- -----
 1: *---*--- *---*--- *---*--- *---**-- -----
 2: *---*--- *---*--- *---*--- *---**-- -----
 3: *---*--- *---*--- *---*--- *---**-- -----
 4: *---*--- *---*--- *---*--- *---**-- -----
 5: *---*--- *---*--- *---*--- *---**-- -----
 6: *---*--- *---*--- *---*--- *---**-- -----
 7: *---*--- *---*--- *---*--- *---**-- -----
 8: *---*--- *---*--- *---*--- *---**-- -----
 9: *---*--- *---*--- *---*--- *---**-- -----
10: *---*--- *---*--- *---*--- *---**-- -----
11: *---*--- *---*--- *---*--- *---**-- -----
```

(b) Alignment of the E-Links.

Figure 4.2.: Screenshots of the output of the `flx-info` low-level tool in a running stress test setup. With the `Yes` and `*` output it is reported, that all lpGBT and all E-Links are locking, indicating stable reception of the lpGBT and RD53A emulators data, respectively. The double column of stars in Fig. 4.2b are the lpGBTs IC and EC link, which are always active.

the MGTs should be reset using the `Soft Reset` command. For the KCU116 and the VCU128's QSFP1 to QSFP3, this usually brings up all the E-Links. If one of those links is not working, it is usually fixed by sending the `Soft Reset` command again. A special case is the QSFP4, whose E-Links are usually not aligned after this procedure. As a temporary workaround, one can replug the fibre cable connected to QSFP4, which prompts a reset. This brings correct alignment to the links connected to QSFP4, although this might require repeating the fibre replugging a few times. When initializing the setup, one should also pay attention not only to the uplink status, but also to the status of the TTC signal on the downlink.

Once it is first achieved, though, the uplink link stability is very good. The boards have been left running for multiple days several times, during none of which any E-Link lost alignment. To check the E-Link stability, additionally a small script was developed which queries `flx-info` every 0.5 s and parses the result to count the alignment losses per channel. In a running setup it did not detect any loss of alignment during 36000 tries.

4.5. System tests

4.5.1. Single-frontend scans

As a first test of the working setup, a test script was created to scan all frontends individually and compare their performances. The test script takes as arguments the serialized `ScanSettings` in the form of a JSON file, detailing the path to the other software components and some settings regarding the scan, the region of frontends to be scanned and a per-event occupancy. The scripts always assume the perspective of FELIX, so that the frontends are labelled by their uplink link (0 to 23) and group (0 to 6) within FELIX. How these are actually connected to the boards is specified in the scan settings.

For each specified frontend, a scan is performed as follows. First, a one-event hitmap is prepared, where μ pixels are randomly selected to have a hit, based on the per-event occupancy μ given in the program arguments. A network connection to the board containing the frontend to be tested is then opened, and the hitmap is uploaded. After this preparation, a scan is performed and the results are analysed. It is a known issue, that felixcore might crash when left running for too long. Therefore, a new felixcore process is spawned before each scan and terminated afterwards.

Multiple quantities are used to determine the quality of the scan. The first observable is the total number of lost triggers, as summed over all warning printouts given by YARR during the scan. If one iteration of the scan loop is missing some triggers, or due to other issues, additional event data may appear in subsequent iterations of the scan loop. As such, there may be a negative number of lost triggers, if there is more data received than expected. Over the whole scan, the negative and positive lost trigger warnings might have cancelled each other out and left this observable mostly unchanged. In order to identify such scans, where there are a lot of lost triggers, but the sum of them is small, the sum of absolute lost triggers is used as an additional complementary observable. Here, first the absolute value of the lost trigger count in the YARR warning message is taken, before the values are being summed. Two more observables are extracted from the occupancy map collected over all events. The first is the number of failing pixels. In this case, a pixel is counted as failing, when the expected occupancy does not equal the observed one. This observable is useful to identify a successful scan, but usually fails once there are lost triggers. As a softer version of it, a second one is a Boolean value signalling a failure in the hitmap pattern. A “pattern failure” is reported, when there exists at least one pixel with either an occupancy > 0, where 0 was expected, or an occupancy of 0, where one > 0 was expected. This observable still reports a success, even if there were lost triggers, if the shape of the hits is the expected one. An absent or completely failing frontend with

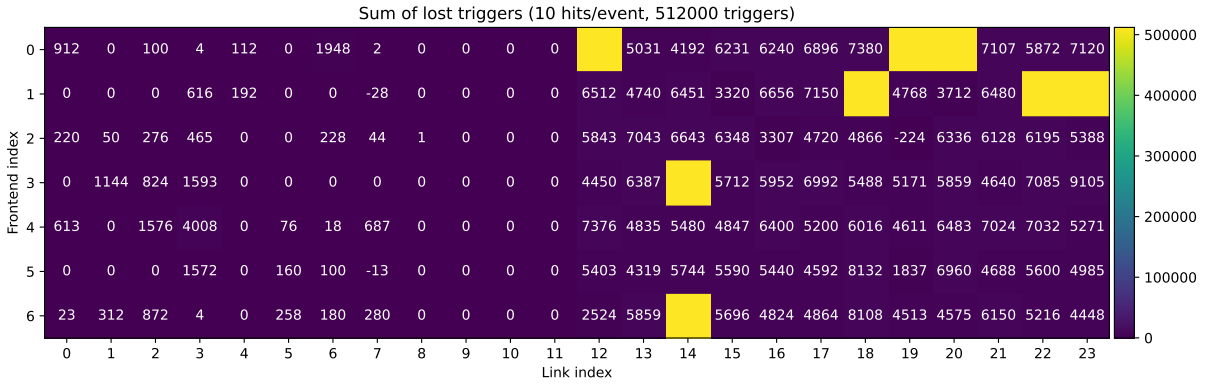


Figure 4.3.: The sum of lost triggers reported during a scan of all individual frontend instances in the stress test setup. The link and frontend indices are from the perspective of FELIX. The links 0 to 7 are served by the KCU116, the rest is served by the VCU128. Fields not showing a value are frontends with 512000 (100 %) lost triggers.

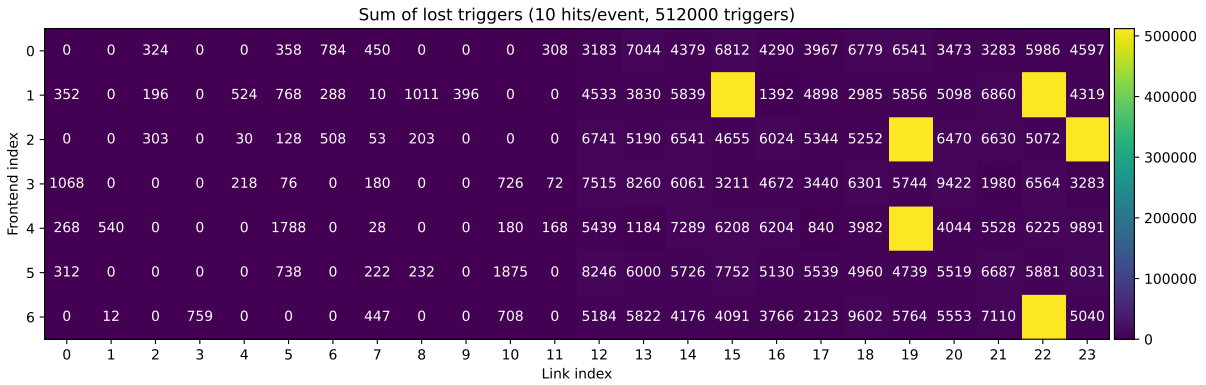


Figure 4.4.: The result of a measurement as in Fig. 4.3, but with swapped FELIX device connections. In this setup, the KCU116 serves link 12 to 19. The VCU128 serves the rest.

no results is counted as having the maximum number of (absolute) lost triggers, $192 \cdot 400$ failing pixels and a pattern failure.

It is seen, that sometimes missing triggers may not only bleed over into the next scan loop of the same scan, but also into the next scan, despite restarting felixcore. For this reason, if a frontend does not produce a perfect scan, a new scan of the same frontend is performed to obtain its results. This way, systematically failing frontends would still perform badly in the new scan, whereas good frontends suffering from the bad performance of the previous frontend are given a chance to recover.

This single-frontend test was performed with 512000 triggers per frontend, at a frequency of 3333 Hz and with a per-event occupancy of 10. The results are shown in

Fig. 4.3. In this figure, links 0 to 7 are served by the KCU116, and the rest of the links are served by the VCU128, with QSFP4 being at link 20 to 23. In order to investigate the source of the failing frontends in the links 12 to 23, the FELIX device 0 and device 1 connectors were swapped and the measurement was repeated. This means, that the links 0 to 11 are moved to the range 12 to 23, and vice versa. The new result can be seen in Fig. 4.4.

4.5.2. Multi-frontend scans

Beyond doing a digital scan only for one frontend at a time, another script is used for simultaneously scanning multiple frontends at a time. The frontends included in the scan are defined in the scan settings JSON file passed to it as a program argument, and the number of frontends included from it can be capped to a maximum. The performance of the scan is written as a JSON file, containing the sum of (absolute) lost triggers for the whole scan and the individual frontend performances.

To see how the scan chain reacts to large numbers of frontends, this measurement is repeated with an increasing number of maximum frontends. The step size is 7, so at each iteration of the measurement loop a new link, starting from link 0, is included. Like the single-frontend scans, this test was performed at a per-event occupancy of 10 and at a YARR loop frequency of 3333 Hz. Two datasets were generated, one with the full 512000 triggers and one with only 8000 triggers (one mask stage) given by YARR. The full measurement with 64 mask stages got stuck while scanning 16 links simultaneously. As such, the results consist of 24 JSON files for the reduced version and 15 files for the full version.

The output was processed further, by only counting the number of frontends with failing pixels and patterns in the occupancy map per file. The number of frontends with failing pixels per included link can be seen in Fig. 4.5. The number of frontends with failing patterns is between 0 and 2, with no obvious relationship to the number of scanned frontends. As another test, the number of lost triggers was divided by the number of triggers sent, in order to compare the two datasets with each other. Since triggers are given to the whole system, but “lost triggers” (missing data) can come from each frontend individually, this ratio may be larger than 1. The results can be seen in Fig. 4.6.

Given that the results of Sec. 4.5.1 were not perfect, one may assume that the lost triggers are simply the combined result of the individual frontend performances. To check this, the lost trigger values of the first n links of a measurement like in Fig. 4.3 are summed, giving an expected result for the multi-frontend scan of the first n links. The comparison between the expected and obtained values for the reduced dataset can be seen

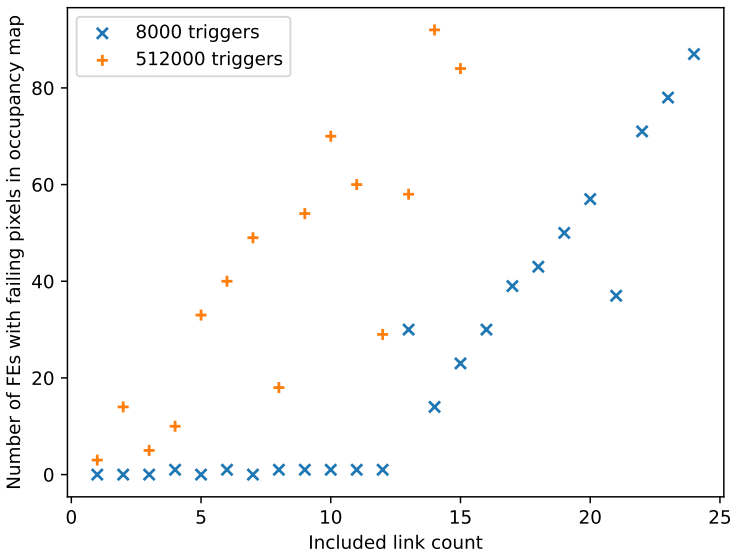


Figure 4.5.: A plot showing the number of failing frontends in terms of occupancy map values, depending on the number of frontends in a multi-frontend scan.

in Fig. 4.7. Doing the same using the sum of absolute triggers yields similar results. In the range, where it is defined, the full dataset matches the reduced one when normalised.

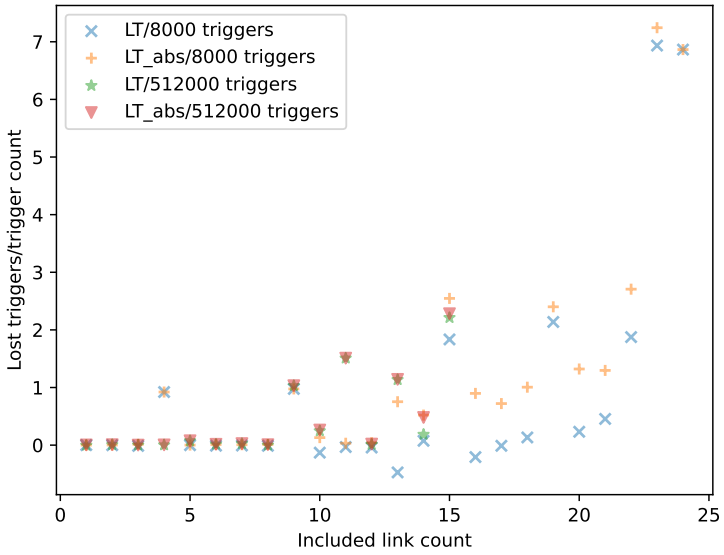


Figure 4.6.: A plot of the sum of lost triggers (LT) and absolute lost triggers (LT_abs) for the two datasets, depending on the number of scanned frontends in the multi-frontend scan. To be able to compare the two datasets, the ratio between lost triggers and total triggers is shown.

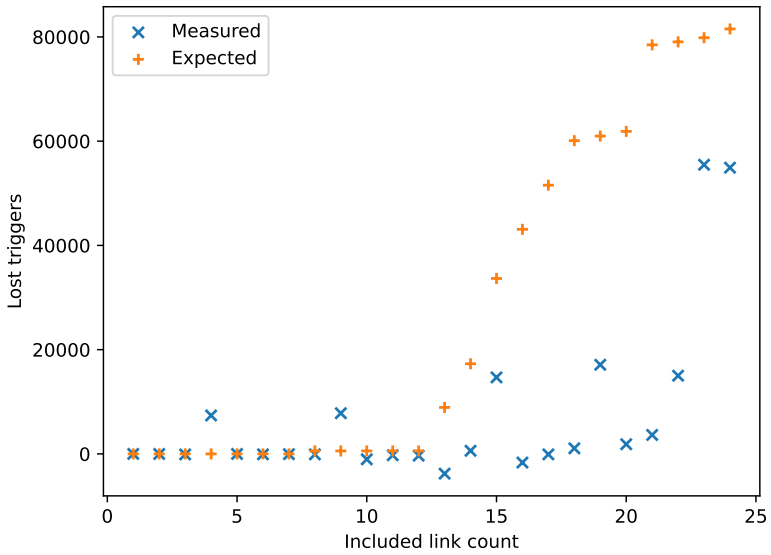


Figure 4.7.: The number of lost triggers observed in a multi-frontend scan compared to the number of lost triggers expected from the single-frontend scans. Both datasets were generated from the reduced number of 8000 triggers.

5. Discussion

5.1. Evaluation of the results

Comparing the two figures Fig. 4.3 and Fig. 4.4, showing the lost triggers per individual frontend, one sees a mismatch in performance between the device 0 and device 1 links of FELIX. For device 0, usually $< 1\,\%$ of the triggers are lost with some frontends being perfect, whereas for device 1 even the functioning frontends consistently show lost triggers in the area of $\approx 1\,\%$. Additionally, device 1 shows a handful of frontends not reporting any data at all.

The disparity is unlikely to come from the stress test setup, as swapping the device connectors yields a similar picture. This also means, that the failing frontends are working once switched, so they also seem to be an effect of the scan chain rather than the stress test boards. The error is therefore assumed to be somewhere in the FELIX hardware/firmware, felixcore or YARR. One criticism of YARR is a dependence on the timing of the data, which is changed by delays in the hardware, the trigger frequency and the CPU speed. While YARR is an unlikely source of the problem on its own, as YARR is decoupled from the device structure of FELIX by the NetIO network interface, it could create the problem in conjunction with different delays in NetIO or FELIX. Otherwise, the problem would likely stem from the FELIX firmware, as felixcore functions independently of the FELIX device used. Still, even in device 0 some data losses occur. For both devices, the positions of the badly behaving frontends change between retries of the test. As such, it could be a stability issue, which would be related to the FPGA timing or the high-speed link. Also, for a yet unknown reason, the VCU128s links of QSFP1 in Fig. 4.3 seem to be working near perfectly, whereas the rest of the QSFPs in Fig. 4.4 shows similar performance to the KCU116. A lot of times in device 0, and sometimes in device 1, the reported trigger values are negative. The reason for this is likely, that the first scan of the frontend resulted in a positive loss of triggers, which then reappeared in the second try of the scan. It is a known issue tied to the timing dependence of YARR. The conclusion from this test is, that errors are created within the chain even when only scanning one frontend. These issues would need to be fixed first, before taking a serious shot at a stress test.

Still, some preliminary tests were performed, where multiple frontends are scanned. As a positive result, it is possible to perform a simultaneous full system scan, albeit only when using the reduced number of triggers. Like one would expect from the single-frontend scan results, the performance becomes noticeably worse once device 1 links are involved, so after 12 included links in Fig. 4.5. Onset of the errors in the occupancy maps start way earlier for the dataset with 512000 triggers. It is expected, as they are caused by missing triggers, which have a higher chance of happening when more triggers are sent. Based on Fig. 4.6, the performance of the scan does not seem to depend on the length of it, since the normalised number of lost triggers is approximately the same for the two datasets.

The most surprising result is the one in Fig. 4.7, indicating that a multi-frontend scan performs better than the sum of the individual involved frontends. This result disfavours an FPGA timing or link stability issue as the reason for the errors, since these should in principle show up at the same rate, independent of the type of scan. Again, this could be explained by a timing dependence of YARR. It is conceivable, that the timing of the data is different enough between the larger amounts of data from the multi-frontend scans and the small amount of data of the single frontend scans to make this difference.

With just the measurements done so far, all the possible explanations have room for doubt. In general, more data and possibly inspection of the inner workings of YARR and FELIX during the scans is needed to get a better picture.

Usually, these tests were performed while keeping all devices running, so without restarting any of the hardware components. It happened a few times, that YARR would eventually stop either at the start or in the middle of the scan. It did not happen enough times, to where a pattern would emerge to make it reproducible, but it serves as a reminder to investigate also the long-term stability of the system, which needs to be pretty much perfect in the final system.

The biggest issue of the tests performed so far is, that the components under test are not at the current state of development. The two main components, YARR and the FELIX firmware, are from checkouts that were done around a year ago (02/25/22 and 03/03/22, respectively). With newer versions, the results might already look different. Another factor is, that felixcore and NetIO are no longer officially supported in favour of their successors, felixstar and NetIO-next. However, due to felixstar being new, the YARR NetIO-next interface corresponding to felixstar does not support the RD53A, yet. In this sense, there is currently no way around using outdated software. Given more time, it would have also been interesting to compare YARR to some alternative branches of development, which have emerged from it.

Conceptually, there are some limitations in using FPGA boards as a replacement for

the ASICs, and to what can be achieved with this setup. First, as discussed in Sec. 3.3, the RD53A emulator by design only has a reduced feature set with respect to the RD53A ASIC. The most important difference is the missing register map and pixel configuration. Generally, the tests rely on the emulator being accurate to the original. Of course, coarsely this is given, as the emulator can be scanned just as an ASIC would. However, some niche details might be different between the two, which could change how the data is received by the rest of the chain. Losses and delays in the electrical links between the lpGBT and the frontends are also not considered, as these connections are done internal to the FPGA. Based on observations in the ATLAS RD53A demonstrator, it would be wise to include a way to artificially introduce data loss on the E-Link connections in a future version of the setup.

In general, the performance of the stress test system seems very good, in the sense that no measurement produces errors, which can be directly traced back to the stress test system. In the link stability tests and when doing single scans on the device 0 links, which are known to work quite well in comparison, it is able to produce perfect results. This means, that it is working at least on the logical level. Also, while running the test scripts, failures in the occupancy map can only be seen in scans which are known to have lost triggers, meaning the hitmap uploading and generation system works reliably so far. Feature-wise it is also well-equipped, with the implementation of the BRAM streaming, which takes more effort to implement than the alternative logic-based generation. Of course, in the current version, the link stability of QSFP4 can still be improved, and there are some future features planned to be implemented. Still, even the testing capabilities of the current setup, especially concerning the BRAM streaming, have not been exhausted yet with only the few tests performed so far. Provided that some weak points of the read-out chain have already been identified by these few tests, it is certain that the real advantage of the developed stress test setup is to provide a precious tool during the next years of read-out development.

5.2. Outlook

While the current system already offers a lot of opportunities to test the YARR/FELIX chain, there are more features planned to improve it. First, of course it would be beneficial to remove the start-up issues of the VCU128s QSFP4. Once these are fixed, the system can be operated fully from a remote connection, allowing external users to connect to it to test their developments. The next target of debugging should be to perfect the performance of the single-frontend scans, so that all E-Links can be perfectly scanned.

Furthermore, the tests themselves can be improved in numerous ways. Currently, the software uses custom code to generate the frontend and YARR configuration files and collect the scan results. In the future, it would be good to exchange this part in favour of the official ITk frontend configuration database (ITk DB), around which code exists to perform the same tasks. By including the ITk DB, also the ITk DB itself would hopefully benefit from it, since like FELIX it is also a tool which should be tested with larger setups. Based around it, there is also code for evaluating the YARR outputs, which could be integrated as well.

So far, the tests did not test FELIX to the same extent, as would be required to ensure correct operation in the final system. For example, the final trigger rate of 1 MHz has not been tested yet. Going to a trigger rate of 1 MHz requires moving away from the digital scan used to obtain the results of this thesis, since digital scans, as explained in Sec. 2.9, have a special structure of the triggers and generate the hits only through the frontend calibration command, which takes some time on the TTC link. Analysing the data at the full trigger rate and at the expected occupancies would probably also require a network of computers beyond just the FELIX server used at the moment.

Since the FLX-712's PCIe bus limits the bandwidth to 126.032 Gbps, which is only around half of the 245.76 Gbps data rate of the 24 lpGBT uplink links, there definitely exists a breaking point of FELIX, at which the full data cannot be processed anymore. Finding out the value of this breaking point would be another future task.

A lot of possibilities of the BRAM-based hitmap generation have not yet been explored. One option to better use it, would be to include real measured events or events generated from simulation. Instead of using full real events, one could also use "snippets" of real hits and place multiple of them on the pixel matrix to generate the events. These two options would make the stress test a bit more realistic. In general, the effects of different event shapes on FELIX have not been studied. Besides changing the average link/hitmap occupancies, one can also experiment with the distribution of them between the different chips. For example, one could introduce a variance in the occupancies, instead of placing a fixed number of hits on the hitmaps. Or one could check if it makes a difference in the scan, when some frontends only carry few hits while other frontends carry more hits.

As mentioned in Sec. 5.1, there are different strains of development in terms of read-out software based on YARR, which could be compared to the base YARR version. Their key feature is, that they do not rely on the timing of the system like YARR does, which is meant to improve robustness. For the FPGA system, this could make a difference, as it might have a different latency or timing behaviour than a chain consisting of lpGBT/RD53A ASICs. On the FELIX side, the upgrade to NetIO-next/felixstar should

definitely be made, once it is available. The upgrade also promises improved performance and stability.

Ultimately, also RD53B/ITkPix emulation should be implemented, as this is the final version of the read-out chip. In particular, the output format is very different between the RD53A, which outputs fixed length 32-bit blocks, and the RD53B, which outputs variable length, compressed streams, so that especially for the high trigger rate tests it would make a difference. For the same reason, it is also harder to implement the RD53B emulation, although maybe a part of the compression can be done offline in the controller software. The complex output format could also result in a resource usage increase of the emulator, potentially leading to the smaller KCU116 running out of space. Another read-out relevant feature not seen in the RD53A is the data merging, in which one master RD53B accepts the data from other RD53Bs and sends it over just one E-Link. Besides the implementation in the hardware, this would also need to be considered in the controller software, since currently the frontends are identified by their uplink group, which would then not be unique any more.

Independent of the frontend type, some further features are planned for the frontend emulator. The most important one is a read connection from the frontend controllers to the PS DMA peripheral. Including it would make the stress test related RD53A/B emulator registers readable, which could be used to take in-system diagnostics while a scan is running. For example, readable counters counting the number of received triggers or the amount of data sent were envisioned. While calculating the expected link bandwidth for simple N-hit hitmaps is straightforward, this feature would make the job of the controller software easier when using more advanced hitmaps. One could also track, whether a trigger (or its response) was lost on the down- or uplink. Another feature, which would be good to include, is the multi-lane operation, where the output data is sent over multiple links. This feature is read-out relevant, as it increases the output bandwidth of an individual frontend. Potentially, having dynamic reconfiguration of the BRAMs during a scan as a feature could be interesting, although it is not necessary at the moment.

6. Summary

In order to verify the correct functionality of the FELIX DAQ components for the ATLAS Phase-II upgrade under high load, a stress test setup has been prepared. The stress test setup consists of a FLX-712 board hosted within a FELIX server, with all of its 24 fibre links connected to the Xilinx KCU116 and VCU128 FPGA development boards. Within the FPGAs on the boards, a total of 24 lpGBT emulator instances and 168 RD53A emulator instances are included to generate the total 245.76 Gbps of uplink data. The RD53A emulator was custom-built to meet the stress test requirements. One of its special features is the ability to store a set of hitmaps in the on-chip memory, which is sent back as the event data and can be used to control the amount and type of data used in the stress test. Beside the emulators, a MicroBlaze based PS is used to receive configuration and hit data from an Ethernet network, making the design a SoC. The stress test setup is completed by a controller software, which orchestrates the boards through this Ethernet network, and the PS firmware, which mediates between the two ends.

Scanning the individual frontends in this setup showed, that in our version of the YARR/FELIX chain, FELIX device 1 performed worse than the device 0. Still, simultaneously scanning all frontends in the setup is possible, although the results can still be improved. Now that a first full version of the stress test is developed, it can be used to correct such issues, and aid in the future ITk Pixel DAQ development in general.

A. Ethernet configuration resources

The following sections describe the technical details of the Ethernet configuration path used to control the RD53A emulators. The protocols/register map are purpose-driven, and will most likely change as the register reads are included, allowing for a cleaner implementation. Still, for the sake of completeness they are described here.

A.1. RD53A controller register map

Table A.1.: The register map of the RD53A controller block.

Register	Bit								Output flag response
	7	6	5	4	3	2	1	0	
0	BRAM_Portal[7:0]								ACK
1	BRAM_Address[7:0]								ACK
2	BRAM_Address[15:8]								ACK
3								HasHitsOverride[0]	ACK
4								OutputEnable[0]	ACK
5	ReportTTCStatus[7:0]								ACK+TTC
6								ResetHitGeneration[0]	ACK
Other									ACK

The register map of the RD53A controller block is seen in Tab. A.1. A write to the hit data BRAM is performed through registers 0, 1, 2 and 6. The BRAM_Address field controls the position of the write pointer within the BRAM. It needs to be set first. Afterwards, the 35 bit wide blocks can be uploaded by performing a multi-byte write to BRAM_Portal. The multi-byte write needs to have a payload of $N \cdot 5$ bytes, $N \in \mathbb{N}$. For each group of 5 bytes, the data under the BRAM write pointer is set to the value

$$\text{BRAM\_data[34:0]} \Leftarrow \text{byte0[2:0]\&byte1[7:0]\&byte2[7:0]\&byte3[7:0]\&byte4[7:0]}\,.$$

If the transmission ends before completion of a 5 byte group, the data is discarded and no action is performed. After a successful write, the BRAM address is auto-incremented,

also within the same AXI-Stream write command. There is no boundary check of the BRAM size when writing.

The settings HasHitsOverride and ResetHitGeneration are wired to the corresponding control signals in Fig. 3.4. If the OutputEnable setting is 1, the output of the emulator is forced to 32'b0. Otherwise, no change is made. No logic is wired to ReportTTCStatus, but it has the side effect of reporting the TTC alignment status: A write to all registers except register 5 pulses the controller output flag, if the lpGBT and RD53A addresses in the AXI-Stream command match (ACK). For register 5, both addresses have to match and also the TTC has to be aligned for the flag to be pulsed (ACK+TTC).

A.2. AXI-Stream protocol

All commands to the RD53A emulator are done via writes to the RD53A controllers register map. A write command on the AXI-Stream bus broadcasted to all emulator instances has the format

$$[\text{lpGBT address}]\,[\text{RD53A address}]\,[\text{register address}]\,[\text{byte 0}]...[\text{byte } N]\,.$$

Each of the entries is one byte wide. The payload, as given by [byte 0] to [byte N], contains the register values to be written, and may be any number of bytes > 0. Note, that the register address does not auto-increment between the bytes. The lpGBT address is the link index of the target instance (0-7 for KCU116, 0-15 for VCU128), and the RD53A address is the E-Link index (0-6) within the link.

A.3. Ethernet connection protocol

A command is given to the stress test board by first writing a byte identifying the command and then optional argument bytes depending on the command. The bytes are sent as a stream using the TCP protocol, and can be split over multiple packets. All commands return one or more bytes as a response. Their function are already explained in Sec. 3.5, so only the format has to be given (Tab. A.2).

If a response is said to be a "bit map", the response is of variable length. The first byte carries the number of bytes following it (N). Each of the next bytes carries Boolean information about the RD53A emulators in one link. The first byte carries the information for the link 0 frontends, the second one for the link 1 frontends, and so on. Within the byte of each link, bit 0 carries the information for frontend 0, bit 1 for frontend 1, and

so on. How the bits are to be interpreted depends on the command. Therefore, these bit maps are used to report a Boolean status for each available frontend.

In the `Write Reg` command, the payload size is given as a 16-bit value split over two bytes. This also limits the maximum burst write size to $2^{16} - 1$. The arguments to the command (except the payload size) are directly forwarded to the AXI-Stream connection.

Table A.2.: The possible commands to the stress test boards over the Ethernet connection.

Command	Parameter	Format
`Ping`	command byte arguments response	0x00 <none> 115, 116, 0
`Enumerate`	command byte arguments response	0x01 <none> Bit map, 1 if the RD53A emulator is present, 0 otherwise
`Write Reg`	command byte arguments response	0x02 [lpGBT index] [RD53A index] [target register] [payload size high] [payload size low] [payload]... 0x00
`LpGBT Config`	command byte arguments response	0x04 [config. byte 0] ... [config. byte 239] 0x00
`TTC Status`	command byte arguments response	0x05 <none> Bit map, 1 if the RD53As TTC input is aligned, 0 otherwise
`Soft Reset`	command byte arguments response	0x06 <none> 0x00

Bibliography

[1] L. Evans, P. Bryant, *LHC Machine*, JINST **3**, S08001 (2008)

[2] G. Apollinari, et al., *High Luminosity Large Hadron Collider HL-LHC*, CERN Yellow Rep. **(5)**, 1 (2015)

[3] ATLAS Collaboration, *Letter of Intent for the Phase-II Upgrade of the ATLAS Experiment*, Technical Report CERN-LHCC-2012-022, LHCC-I-023, CERN, Geneva (2012)

[4] ATLAS Collaboration, *Technical Design Report for the ATLAS Inner Tracker Pixel Detector*, Technical Report CERN-LHCC-2017-021, ATLAS-TDR-030, CERN, Geneva (2017)

[5] ATLAS Collaboration, *Technical Design Report for the ATLAS Inner Tracker Strip Detector*, Technical Report CERN-LHCC-2017-005, ATLAS-TDR-025, CERN, Geneva (2017)

[6] ATLAS Collaboration, *Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System*, Technical Report CERN-LHCC-2017-020, ATLAS-TDR-029, CERN, Geneva (2017)

[7] J. Anderson, et al., *FELIX: a High-Throughput Network Approach for Interfacing to Front End Electronics for ATLAS Upgrades*, J. Phys. Conf. Ser. **664(8)**, 082050 (2015)

[8] T. Heim, *YARR - A PCIe based readout concept for current and future ATLAS Pixel modules*, J. Phys. Conf. Ser. **898(3)**, 032053 (2017)

[9] M. Garcia-Sciveres (RD53), *The RD53A Integrated Circuit*, Technical Report CERN-RD53-PUB-17-001, CERN, Geneva (2017)

[10] lpGBT Design Team, *lpGBT Documentation* (2019), v0.16

[11] M. Garcia-Sciveres, F. Loddo, J. Christiansen (RD53 Collaboration), *RD53B Manual*, Technical Report CERN-RD53-PUB-19-002, CERN, Geneva (2019)

[12] ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST **3**, S08003 (2008)

[13] S. Haywood, et al. (ATLAS Collaboration), *ATLAS inner detector: Technical Design Report, 2*, Technical design report. ATLAS, CERN, Geneva (1997)

[14] P. Jenni, et al. (ATLAS Collaboration), *ATLAS high-level trigger, data-acquisition and controls: Technical Design Report*, Technical design report. ATLAS, CERN, Geneva (2003)

[15] ATLAS Collaboration, *Expected tracking and related performance with the updated ATLAS Inner Tracker layout at the High-Luminosity LHC*, Technical Report ATL-PHYS-PUB-2021-024, CERN, Geneva (2021)

[16] L. Gonella, et al., *A serial powering scheme for the ATLAS pixel detector at sLHC*, JINST **5**, C12002 (2010)

[17] I. S. Reed, G. Solomon, *Polynomial Codes Over Certain Finite Fields*, SIAM J. Appl. Math. **8(2)**, 300 (1960)

[18] PCI-SIG, *PCI Express®Base Specification* (2010), revision 3.0

[19] Xilinx, Inc., *Aurora 64B/66B Protocol Specification* (2014), SP011 (v1.3)

[20] NXP Semiconductors, *I2C-bus specification and user manual* (2014), rev. 6

[21] *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002) (2009)

[22] *IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*, IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012) (2018)

[23] *IEEE Standard for Test Access Port and Boundary-Scan Architecture*, IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001) (2013)

[24] J. Troska, et al., *The VTRx+, an optical link module for data transmission at HL-LHC*, PoS **TWEPP-17**, 048 (2017)

[25] W. Zhang, et al., *Characterization and quality control test of a gigabit cable receiver ASIC (GBCR2) for the ATLAS Inner Tracker Detector upgrade*, JINST **16(08)**, P08013 (2021)

[26] ATLAS FELIX Group, *FELIX User Manual* (2018), RM 3.x

[27] J. Schumacher, C. Plessl, W. Vandelli, *High-throughput and low-latency network communication with NetIO*, J. Phys. Conf. Ser. **898(8)**, 082003 (2017)

[28] S. Baron, et al., *The Versatile Link+ Demo Board (VLDB+)*, JINST **17(03)**, C03032 (2022)

[29] Xilinx, Inc., *UltraScale Architecture GTY Transceivers User Guide* (2021), UG578 (v1.3.1)

[30] Xilinx, Inc., *LogiCORE IP 7 Series FPGAs Transceivers Wizard v2.6* (2013), UG769 (v4.6)

[31] Silicon Laboratories, *Si570/Si571 10 MHZ TO 1.4 GHZ I2C PROGRAMMABLE XO/VCXO* (2018), Rev. 1.6

[32] Silicon Laboratories, *Si5328 ITU-T G.8262 SYNCHRONOUS ETHERNET JITTER-ATTENUATING CLOCK MULTIPLIER* (2013), Rev. 1.0

[33] D. G. Smith, *FPGA Development of an Emulator of the RD53A Prototype Chip and its Integration with Various Readout Systems* (2019), CERN-THESIS-2019-063

[34] J. Postel, *User Datagram Protocol*, RFC 768 (1980)

[35] W. Eddy, *Transmission Control Protocol (TCP)*, RFC 9293 (2022)

[36] Xilinx, Inc., *MicroBlaze Processor Reference Guide* (2021), UG984 (v2021.2)

[37] ARM Limited, *ARM AMBA AXI and ACE Protocol Specification* (2013), IHI 0022E (ID033013)

[38] ARM Limited, *AMBA 4 AXI4-Stream Protocol* (2010), IHI 0051A (RD030510)

[39] A. Dunkels, *Design and implementation of the lwIP TCP/IP stack*, Swedish Institute of Computer Science **2** (2001)

[40] R. Droms, *Dynamic Host Configuration Protocol*, RFC 2131 (1997)

[41] J. Alexander, *Clock recovery from random binary signals*, Electron. Lett. **11**, 541 (1975)

Acknowledgements

I would like to thank my family for the continued support during my studies. Also, I would like to thank the members of the II. Institute of Physics at the University of Göttingen, who were involved in this project. They are Arnulf Quadt, Jörn Große-Knetter and, on a day-to-day basis, Ali Skaf.

Erklärung nach §17(9) der Prüfungsordnung für den Bachelor-Studiengang Physik und den Master-Studiengang Physik an der Universität Göttingen:
Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe.
Darüberhinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, im Rahmen einer nichtbestandenen Prüfung an dieser oder einer anderen Hochschule eingereicht wurde.

Göttingen, den 8. Juni 2023

(Matthias Peter Drescher)