FPGA Development of an Emulator Framework and a High Speed I/O Core for the ITk Pixel Upgrade

Lev S. Kurilenko

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Electrical Engineering

University of Washington
2018

Committee:

Scott Hauck

Shih-Chieh Hsu

Program Authorized to Offer Degree:

Department of Electrical Engineering

University of Washington

Abstract

FPGA Development of an Emulator Framework and a High Speed I/O Core for the ITk Pixel Upgrade

Lev S. Kurilenko

Chairs of the Supervisory Committee:

Professor Scott A. Hauck
Electrical Engineering

Assistant Professor Shih-Chieh Hsu
Physics

The Large Hadron Collider (LHC) is the largest accelerator laboratory in the world and is operated by CERN, an international organization dedicated to nuclear research. It aims to help answer the fundamental questions posed in particle physics. The general-purpose ATLAS detector, located along the LHC ring, will see an Inner Tracker (ITk) upgrade during the LHC Phase II shutdown, replacing the entire tracking system and providing many improvements to the detector technology. A new readout chip is being developed for this upgrade by the RD53 collaboration, code named RD53A. The chip is an intermediary pilot chip, meant to test novel technologies in preparation for the upgrade. The work contained in this thesis describes the Field-Programmable Gate Array (FPGA) based development of a custom Aurora protocol in anticipation of the RD53A chip. Leveraging the infrastructure developed to facilitate hardware tests of the custom Aurora protocol, a cable testing repository was created. The repository allows for preliminary testing of cabling setups and gives the users some understanding of the cable performance.

Contents

Chapter 1: Introduction

Located in Geneva, Switzerland, the Large Hadron Collider (LHC) aims to help answer the fundamental questions posed in particle physics. The LHC is the largest particle physics accelerator laboratory in the world, operated by CERN (French: Conseil Européen pour la Recherche Nucléaire), an international organization dedicated to nuclear research [1].

The LHC contains several stages of particle acceleration, including the Linear Accelerator (LINAC 2), Proton Synchrotron Booster (BOOSTER), Proton Synchrotron (PS), Super Proton Synchrotron (SPS), and finally the LHC itself [1]. The various stages accelerate bunches of particles (usually protons), to close to the speed of light, at which point the particles collide at collision points located around the LHC ring (as shown below in Figure 1.1). General purpose particle detectors placed around the ring detect the particles resulting from the collisions. The two general-purpose particle detectors are called ATLAS (A Toroidal LHC ApparatuS) and CMS (Compact Muon Solenoid), located in Switzerland and France respectively. Other detectors, such as ALICE and LHCb are more specialized in their functionality, looking for specific particle signatures.

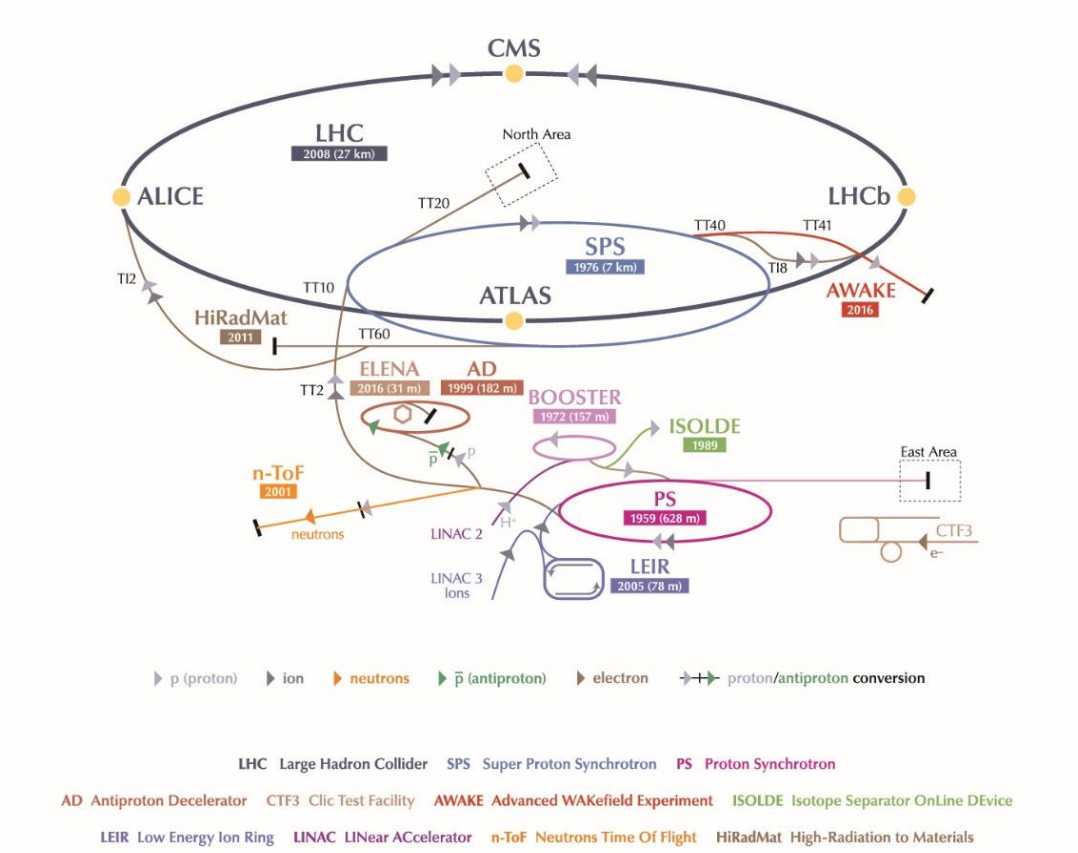


Figure 1.1: The layout of the LHC and its various acceleration stages [2]

1.1 The LHC Ring

The largest and final stage of particle acceleration, the LHC, is a 27-km circumference ring accelerator and can achieve energies of up to 13 TeV at the collision points [3]. The LHC is located

100 meters underground and has sections located in both Switzerland and France. Generally, as the size of the accelerator increases, higher energies can be achieved. For comparison, the two preceding acceleration stages, SPS and PS, operate at 450 GeV and 25 GeV respectively [4]. These stages are significantly smaller in size than the LHC and now act as booster stages, accelerating particles as much as they can, before launching them off to the next stage of acceleration (PS to SPS, and SPS to LHC).

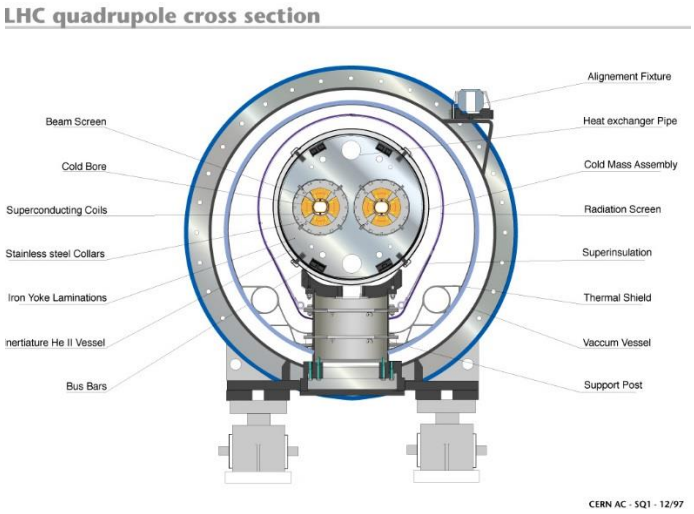


Figure 1.2: Cross-section of an LHC dipole element [5]

To accelerate particles to high energies, the ring in the LHC is composed of superconducting magnets, allowing the conduction of electricity without resistance [4]. These magnets provide the requisite magnetic field needed to accelerate charged particles to nearly the speed of light. The particles used in the accelerator are usually protons; however, heavy ions such as Lead are also used for specific experiments [4]. Protons are charged particles and are significantly more massive than electrons, allowing for more effective acceleration due to lower energy loss per turn through synchrotron radiation [4].

The particles are accelerated in bunches, where a bunch may consist of many protons (up to 10^{11} protons) [1]. The bunches travel around the LHC in opposite directions, in separate beam lines, at energies of 6.5 TeV per beam. The cross section of an LHC dipole element is shown in Figure 1.2, where the two beam lines can be seen. There are four points throughout the ring where the beams intersect, allowing the particles to collide. This collision is referred to as a bunch crossing. The energy of the bunch crossing is the sum of the energy of each of the beam lines; in the case of the LHC, the resulting collision energy is 13 TeV [4]. The bunch crossing rate at the LHC is 40 million bunch crossings per second [4].

The Phase II upgrade, scheduled for the LHC around 2025, will increase the luminosity to 3000 fb^{-1} and will reach energies up to 14 TeV within 10 years after the upgrade [6]. The increase in luminosity results in more collisions at the interaction point, increasing the amount of data generated. The high luminosity detector is abbreviated as HL-LHC [6].

1.2 ATLAS Detector

The developments in this work focus on the ATLAS (A Toroidal LHC ApparatuS) general-purpose particle detector, located on the Swiss side of the LHC. The ATLAS detector is shown in Figure 1.3, with one of the distinguishing features being the large toroid magnets located on the outer parts of the detector [1]. The detector is roughly 44 meters in length and 25 meters in height and is significantly larger than the CMS detector located on the French side. This difference in size is attributed to differences in technical and design approaches.

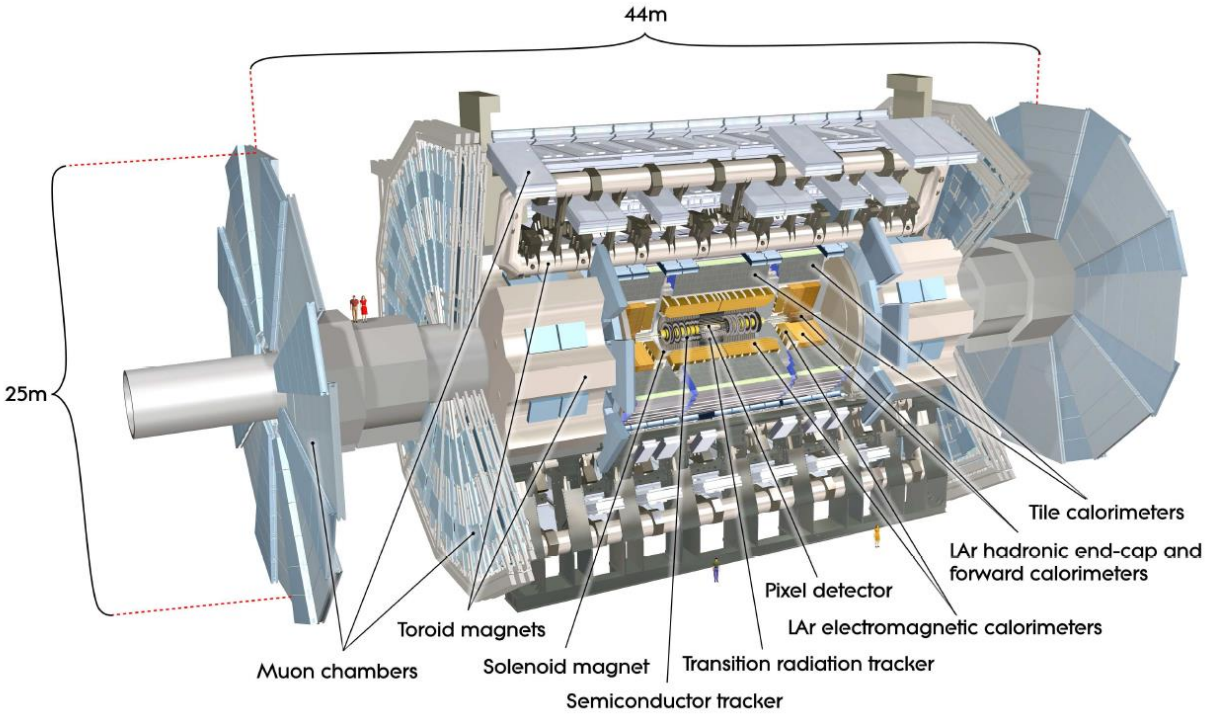


Figure 1.3: ATLAS Detector Side View [1]

When particles collide at extremely high energies, they produce many resulting sub-atomic particles that travel radially from the collision point. Examples of such particles are electrons, protons, neutrons, muons, and photons, among others [1]. Recording and analyzing the data from the resulting particles is important in the progression of the Standard Model of Particle Physics, understanding Dark Matter, and many other areas of research. The data allows for analyses such as the measurement of transverse momentum, trajectory reconstruction, and locating the position of radiation charged particles [7]. Physicists analyze this data to study the nature of these particles.

The characteristics of the particles are factored in the design of the detector, and drive the layout of the detector, as well as the detector technologies used [1]. Working our way from the inside (point of collision in the beamline) out, the detector consists of an Inner Detector, Calorimeters, and Muon Detectors. The Inner Detector can be further subdivided, using the same inside out approach, into the Pixel Detector (Pixel), followed by the Semi-Conductor Tracker (SCT) and the Transition Radiation Tracker (TRT). The calorimeters consist of the Liquid Argon Calorimeter (LAr) and the Tile Calorimeter. The Muon detectors are located on the outermost parts of the detector and take up a large amount of the volume of the overall detector [1] [7]. The components

of the Inner Detector will be replaced by an all silicon detector with the ITk upgrade described in Chapter 2 [8].

Each of these detector systems serve a specific purpose. The overall detector must be properly coordinated in order to filter collisions, properly format and store data, monitor the status of each subsystem, and many other considerations. While general-purpose particle detectors are massively complex and contain many systems, the Frontend Electronics (FE) and Data Acquisition (DAQ) systems will be the focus of this work. FEs are used for the detection of particles. They record data that can be readout by systems further upstream. DAQs perform the readout and processing of the data and can perform more specialized functions [9].

1.3 Triggering and Data Acquisition

Determining whether a significant collision event has occurred is handled by the triggering system, which works in conjunction with the DAQ. Triggering systems have several levels of triggers and look at the data obtained from a "bunch crossing", i.e. bunches of particles crossing from opposite sides, to determine whether the collision constitutes an event worth keeping [9]. This allows for filtering of useless events and retaining only the interesting data.

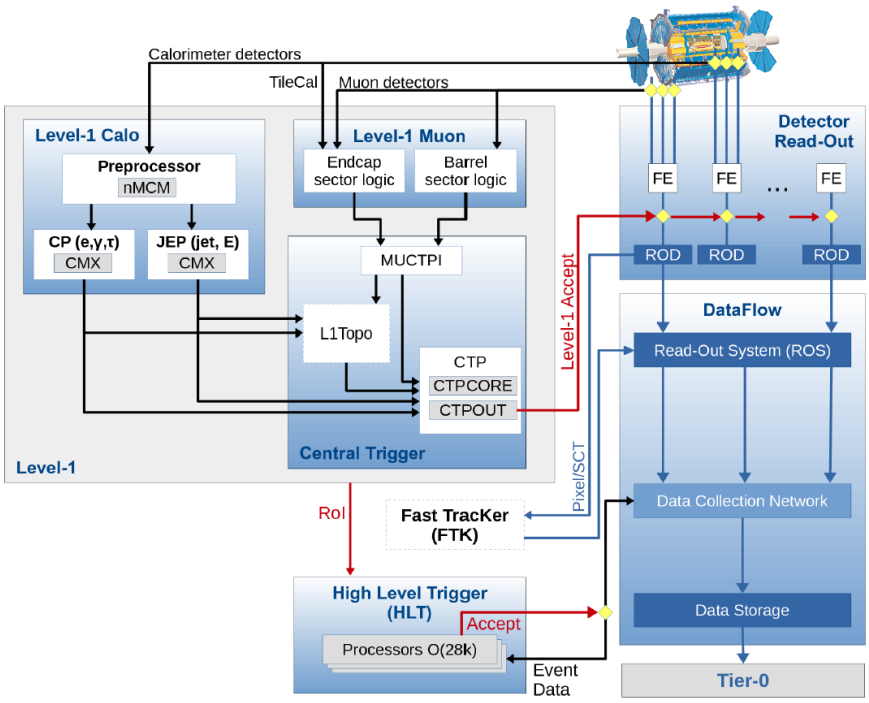


Figure 1.4: Upgraded ATLAS TDAQ for LHC Run 2 after long shutdown 1 [9]

To reduce the data to manageable levels, several levels of triggers are used at the LHC. Level-1 (L1) triggers process information from the Muon and Calorimeter detectors. The calorimeter detectors provide information about energy observed in a region of the detector. Muon detectors target muon particles, which are longer lived and highly penetrating. If these detectors provide data to the L1 trigger processor that matches some preprogrammed characteristics, a 'Level-1 Accept' will be issued by the processor [9].

If a Level-1 Accept has been issued, High-Level triggers analyze regions of interest identified by L1 triggers and perform a detailed analysis of event data, deciding whether the event is interesting enough to keep. If this is the case, the event data is sent for permanent storage to disk; otherwise the data is discarded [9].

L1 triggers are processed using custom ASICs and FPGAs, whereas High-Level triggers are processed using CPUs. The algorithms for L1 triggers and data processing architectures are always being improved and fine-tuned. As more data is collected and a better understanding of the Standard Model is obtained, L1 triggering algorithms are changed accordingly. This requires changes to the FPGA image [9]. The evolving nature of triggering algorithms make FPGAs a capable and well-chosen technology.

Chapter 2: Inner Tracker Upgrade (ITk) at the Large Hadron Collider

The ATLAS general-purpose detector is scheduled for a replacement of the entire tracking system around 2025, during the LHC Phase II shutdown [8]. The innermost Pixel detector upgrade is called the Inner Tracker (ITk) upgrade and is a massive R&D effort investigating detector layout, sensors and front-end electronics, powering and detector control, and readout architecture. Planned for the upgrade is a new 5-layer Pixel detector with improved tracking performance and radiation tolerance and a new 4-layer Strip detector [8].

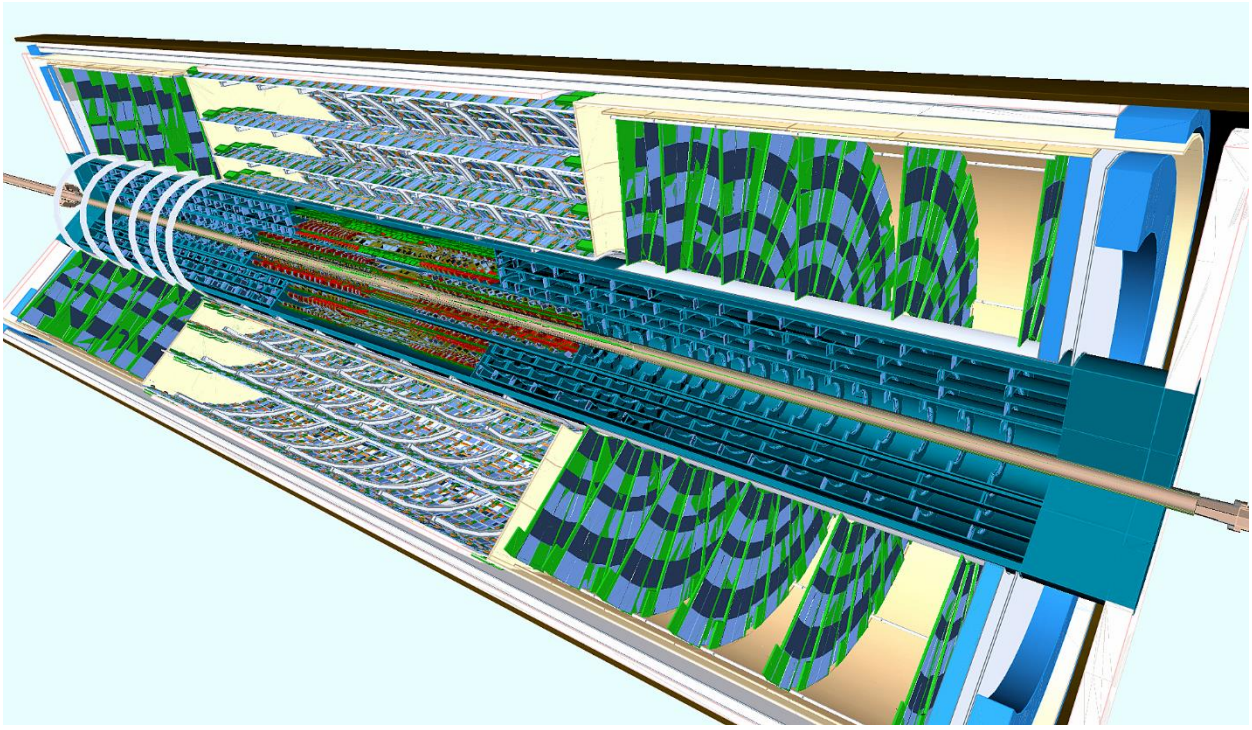


Figure 2.1: Side view of the planned ITk Pixel Detector [7].

The ITk Pixel detector will replace the entirety of the existing inner Pixel detector. The HL-LHC environment will have far greater radiation than is currently present in the detector, requiring new radiation hardened electronics to be developed. Additionally, the trigger rate in the HL-LHC will increase to five times that of the current LHC (200 kHz to 1 MHz), requiring increased bandwidth in the readout electronics [6].

2.1: RD53A Pixel Readout Integrated Circuit

The ITk upgrade requires an Integrated Circuit (IC) that can handle high radiation levels, 1 MHz trigger rates, high bandwidth communication, and other demanding requirements while also keeping in mind factors such as power consumption [8]. The RD53A readout chip is an intermediary pilot chip meant to test several front-end technologies and is not meant to be the final Pixel readout chip [6].

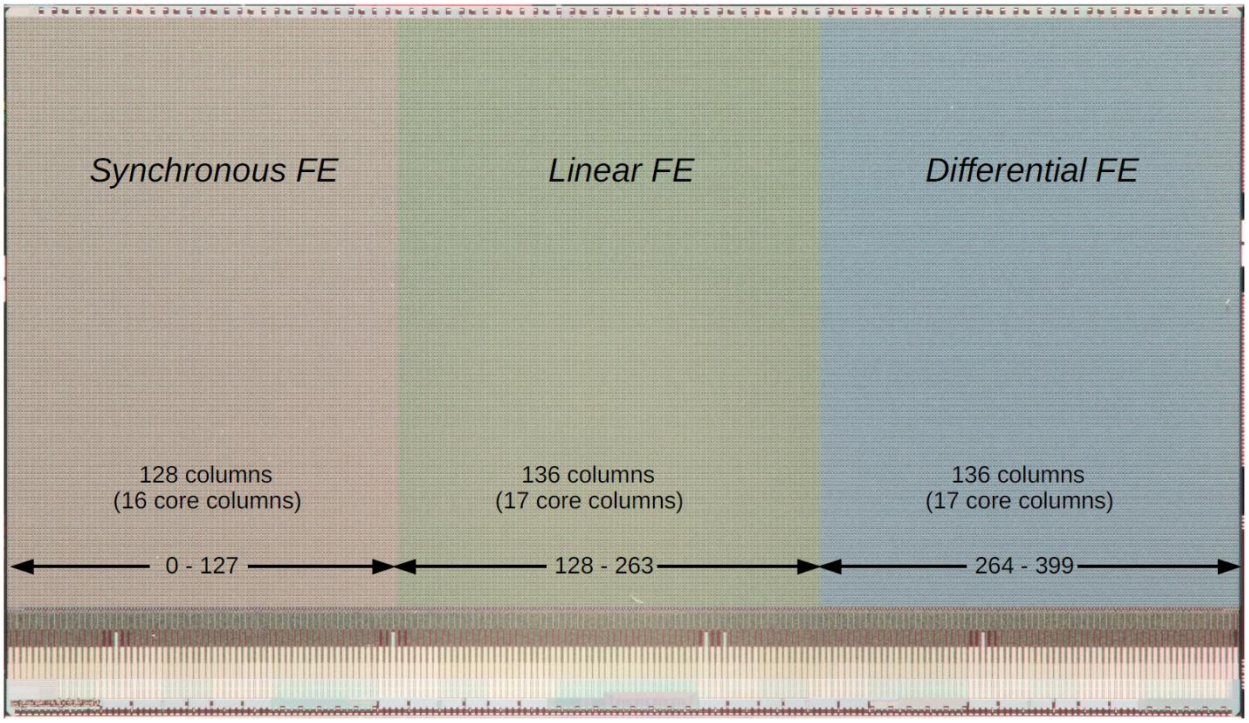


Figure 2.2: Three front-end flavors on the RD53A chip [10]

Figure 2.2 shows the three front-end flavors being tested on the RD53A chip; Synchronous, Linear, and Differential analog front-ends. Each front-end technology contains a different approach at solving the same problem: detecting charged particles and measuring particle characteristics, such as Time-over-Threshold (ToT) [10]. Performance characteristics of the three front-ends will be evaluated and only the best performing front-end design will be used in subsequent chips, making the entire pixel matrix uniform [10]. The bottom of the chip contains the periphery logic, which implements all of the control and processing functionality [10].

FPGAs have been instrumental in the process of developing the necessary technologies for the ITk upgrade [8]. Use cases include FPGA emulation of future FE ASICs, High Speed Communication, Data Aggregation, and Data Processing.

2.2: RD53A FPGA Emulator

The RD53A FPGA Emulator is a development effort at the University of Washington aiming to provide a platform that can be used to test various DAQ systems before the general availability of the RD53A IC.

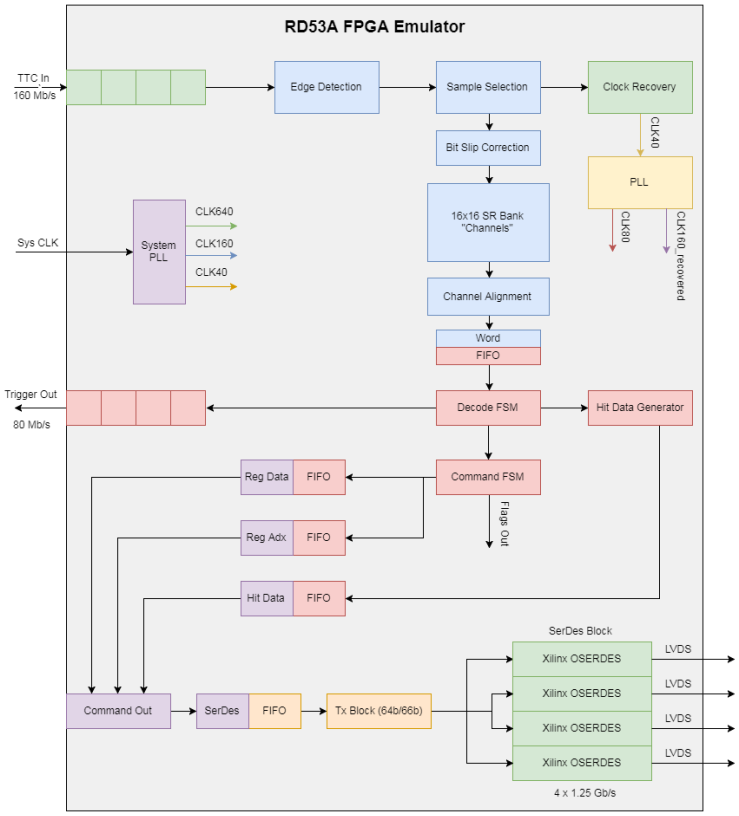


Figure 2.3: RD53A Emulator Block Diagram

The RD53A emulator block diagram is shown in Figure 2.3. The RD53A chip contains both analog and digital components; however the FPGA emulator is restricted to only emulating the digital components of the IC. Two major components that are emulated are the digital I/O communication block logic for the TTC stream and the Aurora 64b/66b output stream. The logic for the TTC stream includes clock and data recovery logic, as well as channel alignment. The logic for the Aurora 64b/66b stream supports multi-lane 1.28 Gbps output links.

In addition to the communication logic, the FPGA emulator also emulates the Global Registers, Command Decoding, and Hit Data Generation. The functionality supported by the emulator allows for simple tests with DAQ systems. One such test may be sending a trigger from the DAQ over the TTC line and receiving corresponding hit data over the Aurora 64b/66b links. Chapter 3 covers the development of the Custom Aurora 64b/66b High Speed IO Core, which contains both a Tx core and a Rx core. The Tx core is integrated into the emulator, while the Rx core can be used in DAQ systems.

Chapter 3: Custom Aurora 64b/66b High Speed IO Core

High speed communication links may use data encoding techniques to achieve better transmission performance across several metrics. Imagine a transmission (Tx) block sending data to a corresponding receiver (Rx) block over a single point-to-point connection as shown in Figure 3.1. A constant stream of data is sent to the Rx block across a communication medium. There is no accompanying clock sent with the data, so the only thing the Rx block sees is a serial stream of 1's and 0's. Additionally, the Tx and Rx blocks operate using different local clock(s).

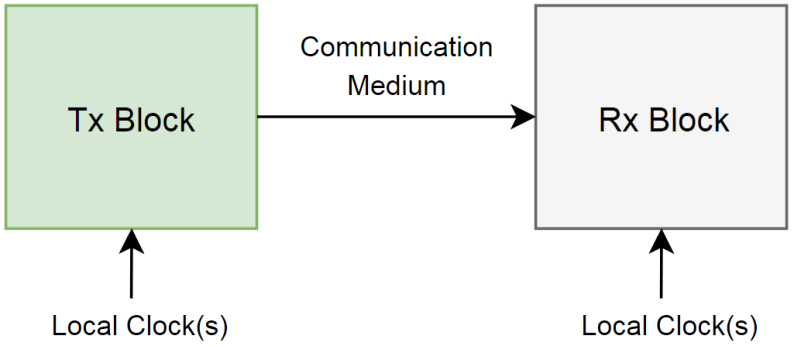


Figure 3.1: Tx core sending data to a Rx block through some communication medium

With any such system, several challenges become immediately apparent and need to be addressed. First, there is no guarantee that the local clock(s) of each system will be in phase. This can lead to an undesired condition where the incoming data stream and the local Rx sampling clock are sufficiently out-of-phase, causing the data to be sampled improperly. Second, if the Tx core sends a long run of consecutive 1's, DC drift may occur. This occurs when circuits with capacitive coupling on the receiver end accumulate enough charge, potentially causing issues with level detect thresholds [11]. In addition to these issues, there are a myriad of other things to worry about, such as line termination in differential transmission, clock phase drift, clock jitter, synchronization, etc.

To address the first issue of the Rx sampling clock being out-of-phase with the data, a clock recovery scheme can be implemented. Under the assumption that bit-to-bit transitions occur sufficiently often in the incoming serial stream, a usable sampling clock can be created. This is achieved by phase aligning the sampling clock to the transitions of the incoming data using a phase-locked loop (PLL). On a similar note, DC drift can be mitigated when the number of 1's and 0's in the incoming data stream are approximately equal and transitions happen frequently enough. If these conditions are met, the data stream is considered "DC-Balanced".

All these challenges are present in the RD53A to DAQ system and must be addressed with a line code for the data. A line encoding technique provides mechanisms to ensure the appropriate maximum run length, DC-balance, etc. Although there are several line encoding techniques, the 64b/66b line code was chosen, which has been used in technologies such as 10 Gigabit Ethernet and InfiniBand [12]. The maturity of this technology and the support from Xilinx FPGAs, prevalent in many LHC DAQ systems, made this encoding an appropriate choice.

The RD53A chip contains a 64b/66b Tx core capable of driving 4 current model logic (CML) outputs at 1.28 Gbps each [10]. Xilinx provides an implementation of the 64b/66b encoding scheme for FPGAs in the form of an IP core named ‘Aurora 64B66B’ [13]. The Tx core in the RD53A chip is compatible with this Xilinx core, allowing for a capable Front-End chip to DAQ communication link. However, there are some limitations which will be covered in Section 3.1.

3.1: Motivations

As previously mentioned, the 64b/66b encoding has been used in many different interfaces and platforms. While the encoding is well defined, implementation of the encoding may vary from system to system. More specifically, the Xilinx implementation of the 64b/66b encoding, called ‘Aurora’, may differ in implementation details when compared to InfiniBand or 10 Gigabit Ethernet [12]. In essence, Aurora 64b/66b encoding uses the Xilinx implementation of 64b/66b encoding.

Xilinx provides an IP core called ‘Aurora 64B66B’ with many configuration options such as line rate, dataflow mode, flow control, etc. The core itself is a mixture of hardened IP in silicon and soft IP described in an HDL language. The IP leverages the GTX gigabit transceivers found in 7 Series Xilinx FPGAs [13]. The GTX blocks are highly flexible and support a myriad of protocols and standards. When using the Aurora IP core, a GTX block primitive is instantiated and configured in a way that will support the 64b/66b encoding. In addition to instantiating a GTX core and supplying it with proper configuration, additional HDL “wrapper” code is used to describe elements of the encoding not contained in the GTX core, such as scrambling and channel bonding, which will be described in more detail later.

The Aurora IP core provided by Xilinx is well documented, highly flexible, and is straightforward to integrate. However, there are several limitations to using the core in the context of the RD53A test chip. The first limitation is that the core is not compatible with Artix FPGAs. While many DAQ systems at CERN use compatible FPGAs, such as Kintex 7 and Virtex 7, there are existing DAQ systems that use the Artix 7 FPGA. The next limitation is that the Aurora IP core requires a GTX core. There are a limited number of GTX cores inside an FPGA and using one may incur significant overhead, especially when considering the additional logic around the Aurora IP core to ease interfacing. The final limitation is that the minimum bitrate for the IP core is 500 Mb/s [13]. The RD53A test chip has the capability to drive 64b/66b encoded data at bitrates lower than that: 320 Mb/s and 160 Mb/s. Driving data at a lower bitrate is useful when the link between the Tx and Rx contains cables that cannot handle the higher bitrate.

To address these issues and attempt to meet the needs of the DAQ systems which will be interfacing with the RD53A chip, a custom Aurora protocol was developed. The custom Aurora protocol is flexible and makes use of the SERDESE2 blocks in the FPGA. Additionally, the custom protocol removes a lot of incurred overhead when using the Xilinx Aurora IP core. This is achieved by reducing the features the core supports to fit the scope of the RD53A chip. A key advantage of the custom protocol is that it makes use of the SERDESE2 blocks, allowing the use of regular I/O for data transmission, giving more flexibility regarding where the data is being driven i.e. as opposed to having to use dedicated GTX I/O [13]. Finally, the custom protocol leverages RTL code from the RD53A IC, allowing us to conform closely to the output expected from the chip.

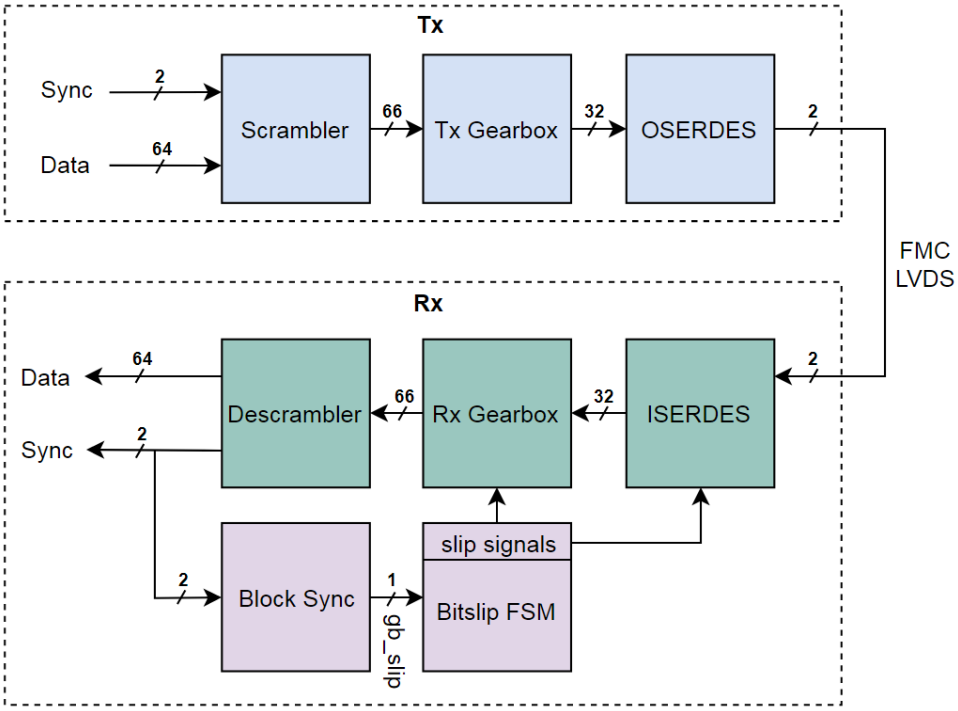


Figure 3.2: Single lane Custom Aurora block diagram

The sections that follow will cover the specifics of the custom protocol in more detail. An overview of a single lane Tx to Rx connection is shown in Figure 3.2. The custom protocol implementation can be broken up into two parts, the Tx core and the corresponding Rx core. The Tx core is meant to model the chip code RTL as closely as possible, while the Rx core is developed to properly synchronize to the Tx core. Hardware board-to-board tests could be performed on the custom protocol before the actual RD53A chip arrives. This provides some utility such as cable and hardware testing and debugging in preparation for the chip.

While the work described in this thesis uses an existing encoding and leverages several existing modules, the novelty comes from reducing the overhead, integrating the SERDESE2 primitive, and developing many new modules (Rx Gearbox, Bitslip FSM, Channel Bonding, Top Level Encapsulation, etc) to provide a packaged Rx core that can be used in DAQ systems at CERN (or derivatives of it).

3.2: Tx Core

Up to this point, the specifics of the 64b/66b encoding have been mostly overlooked. To begin, the basic elements of the 64b/66b encoding are a 2-bit sync header and 64-bits of data. A block is specified as 2 bits of sync followed by 64 bits of data and is shown in Figure 3.3.

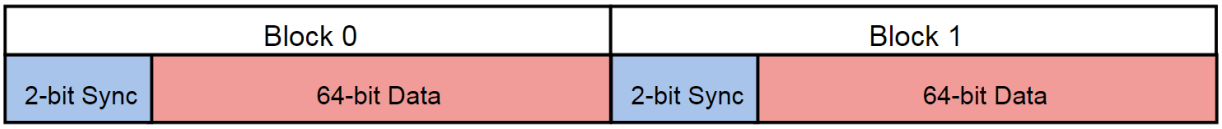


Figure 3.3: A stream of 64b/66b encoded blocks

The sync header is described in Table 3.1.

Table 3.1: Sync headers and their functionality

Sync Header	Binary Representation	Description
Control	10	The control sync header is used to specify control blocks i.e. 64-bits of control data.
Data	01	The data sync header is used to specify data blocks i.e. 64-bits of user data.
Invalid Header	00 or 11	Invalid headers that cannot be used.

The sync header can be split into three categories: control, data, and invalid. Depending on the value of the sync header, the 64-bits of data that follows will be interpreted accordingly. With a sync header of 10 the data is treated as a control block, and with a sync header of 01 the data is treated as user data. 00 or 11 sync headers are invalid [14].

The reason valid sync headers are limited to 10 or 01 is because the 64b/66b encoding is meant to guarantee a maximum run length that is below a certain value, allowing the clock recovery circuits to operate properly. Run length is described as the number of consecutives 1's or 0's. Consider the case where the 64-bits of data are either all 1's or all 0's. If 00 or 11 sync headers were allowed, the 66-bits may be all 1's or all 0's, giving you a run length that exceeds 66 bits. Limiting valid sync headers to 01 or 10 guarantees transitions at 66-bit intervals, even when the data is all 1's or 0's [14]. In addition to guaranteeing a maximum run length, the sync headers are also used for synchronization on the receiver end.

In the Xilinx Aurora specification, when a control sync header is present, the control block that follows contains an 8-bit 'Block Type Field' specifying the type of control block being sent. Table 3.2 is from the Xilinx Aurora protocol specification and contains the control block names, along with their corresponding 'Block Type Field' values [14].

Table 3.2: Xilinx Aurora Control Block Type table [14]

Control Block Name	Block Type Field (BTF) Value: Block Code[2:9]
Idle/NotReady/Clock Compensation/ Channel Bonding	0x78
Native Flow Control	0xaa
User Flow Control	0x2d
Separator	0x1e
Separator-7	0xe1
User K-Block 0	0xd2
User K-Block 1	0x99
User K-Block 2	0x55
User K-Block 3	0xb4
User K-Block 4	0xcc
User K-Block 5	0x66
User K-Block 6	0x33
User K-Block 7	0x4b
User K-Block 8	0x87
Reserved	0xff

The specific function of each control block is covered in greater detail in the Xilinx Aurora protocol spec, SP011 [14].

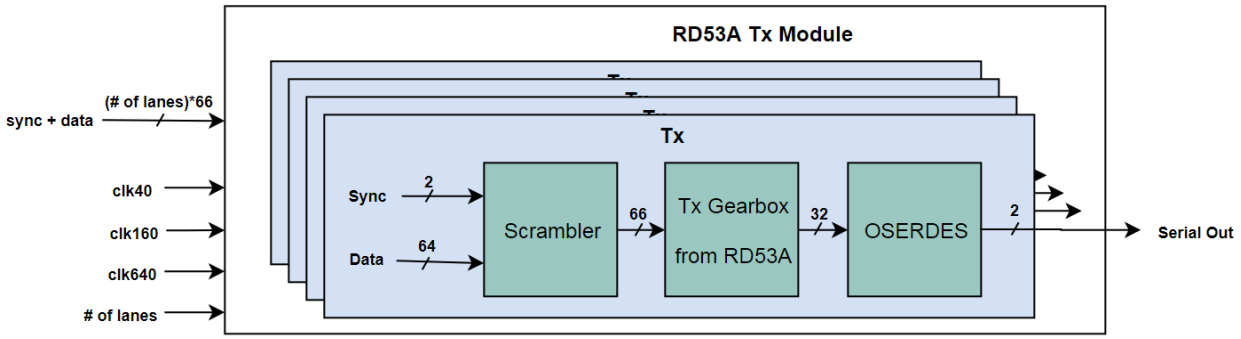


Figure 3.4: Four lane custom Aurora Tx core

Figure 3.4 shows the custom Aurora 4-lane Tx module. The module is meant to be plug-and-play and attempts to closely represent the Tx core in the RD53A chip. The top-level 4-lane Tx module can be further subdivided into single-lane Tx modules, which are instantiated four times. In the figure, the blocks highlighted in green are the same for every lane. Each lane receives its own sync header and data, which is fed into the scrambler. The scrambler is a multiplicative self-synchronizing scrambler and only scrambles the 64 bits of data, leaving the 2-bit sync header unscrambled [14]. More in-depth coverage of the scrambler functionality is covered in Section 3.2.1.

After the scrambler, the 66-bits of the block (sync header + scrambled data) are fed into the Tx gearbox. The Tx gearbox takes the 66-bit block and outputs 32 bits at double the rate i.e. for every 66-bit block, two instances of 32-bits are normally generated. The Tx gearbox is useful for taking the 66-bit encoded data and converting it into bit sizes that are easy to work with when serializing. The fact that for every 66 bits provided to the gearbox, only 64 bits are outputted, means the gearbox inputs need to pause periodically to allow the internal buffer to "catch up". This is discussed in more depth in Section 3.2.2. Finally, the 32 bits from the Tx gearbox are sent to the output serializer, or OSERDES (Serializer/Deserializer). The OSERDES serializes the data and transmits a differential signal (LVDS in this case).

ModelSim Testing

The sections that follow look into the specific details of each component in the Tx module. When designing, testing, and integrating these modules, each module was tested in a standalone ModelSim simulation. After each component was determined to be functional, they were integrated into higher levels of simulation. For instance, the scrambler module was first simulated with a corresponding descrambler module. The Tx gearbox was simulated with a corresponding Rx gearbox. After each component was determined to be properly functioning, they were integrated into a higher-level simulation i.e. scrambler + Tx gearbox to Rx gearbox + descrambler. This simulation and testing hierarchy was applied to the rest of the components in the design e.g. OSERDES, ISERDES, etc.

3.2.1: Scrambler

In the introduction of this chapter, the necessity for bit-to-bit transitions to occur sufficiently often was discussed. This ensures that the PLL on the receiving end can generate a usable sampling clock and that the line is DC-balanced [11]. The scrambler acts as the main component for this function. As the name implies, the scrambler takes incoming data and scrambles it, giving roughly the same amount of 1's and 0's post-scrambling. The data can then be descrambled on the receiving end using a corresponding descrambler. The scrambler does not act as encryption and is not used with that in mind.

The custom Aurora protocol uses the scrambler provided by the Xilinx Aurora 64B66B IP. The scrambler is a multiplicative self-synchronizing scrambler. This means the Tx scrambler and Rx descrambler may be initialized at different points in time, or be in different states, but can still achieve synchronization. If the Tx scrambler properly scrambles the data, the Rx descrambler will synchronize to the Tx scrambler after two blocks of scrambled data is received [14].

Scramblers can be described using a polynomial, which tells you what tap values to use for feedback. Figure 3.5 shows an example scrambler that uses a polynomial function of $1 + x^{18} + x^{23}$.

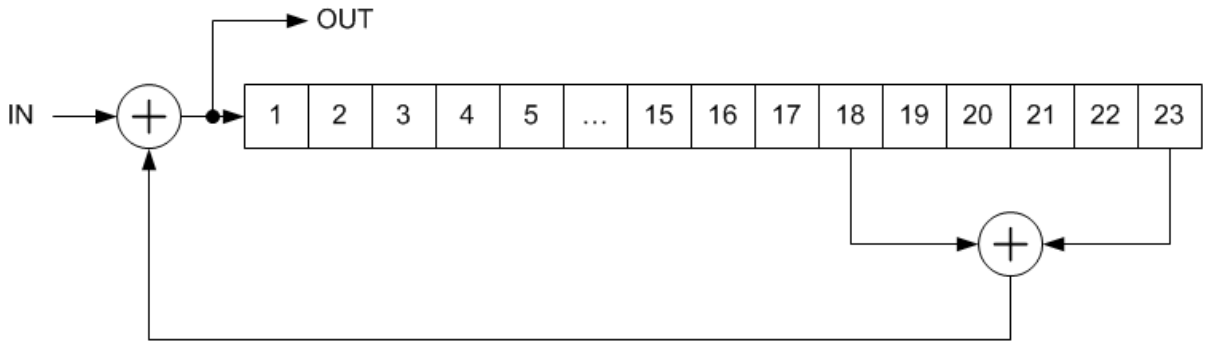


Figure 3.5: Example of a scrambler with function $1 + x^{18} + x^{23}$ [15]

As per the Xilinx documentation, SP011, the Aurora 64b/66b protocol uses the following polynomial function [14]:

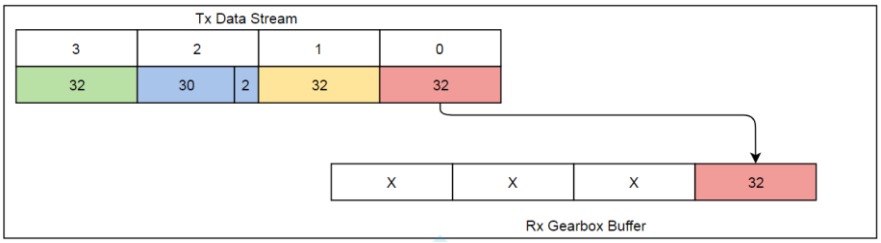
$$G(x) = 1 + x^{39} + x^{58}$$

In the custom Aurora implementation, the scrambler module was leveraged from the Xilinx Aurora 64B66B IP core [13]. In the IP core, the scrambler module is implemented in HDL, with the code available in the wrapper logic of the IP. The descrambler was also leveraged in a similar manner. The scrambler and descrambler modules were tested extensively in simulation before being integrated into the custom Aurora protocol.

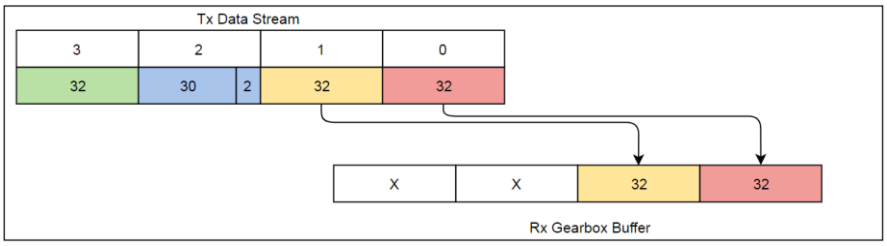
3.2.2: Tx Gearbox

In the custom Aurora implementation, the Tx module is used directly from the RD53A chip RTL code. This ensures that the custom Aurora Tx core behaves similarly to the chip, and that the corresponding Rx core is compatible with the chip. The Tx gearbox module comes after the scrambler and receives the 2-bit sync header and 64-bit scrambled data as it's input from the scrambler. The gearbox aggregates the sync header and the scrambled data to form a 66-bit block. The block is then normally sent in 32-bit instances at double the rate of incoming sync and scrambled data. For every new 66-bit block, the gearbox sends two 32-bits outputs. Every 32 blocks, the flow control in the Tx Gearbox tells the data driver further upstream to pause data for one block.

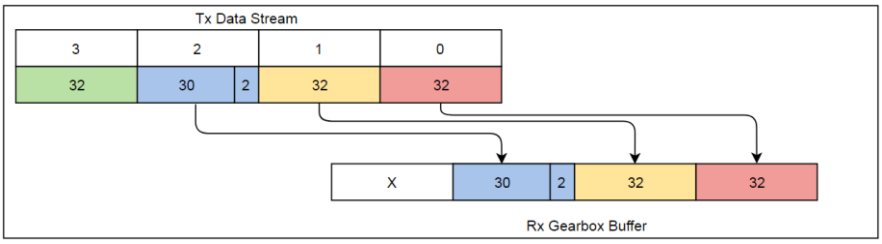
To help illustrate the functionality of the gearbox, Figure 3.6 goes through a data stream processed by the Tx gearbox and sent to a corresponding Rx gearbox.



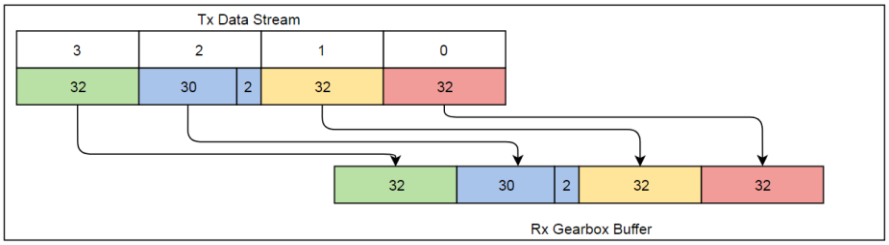
(a)



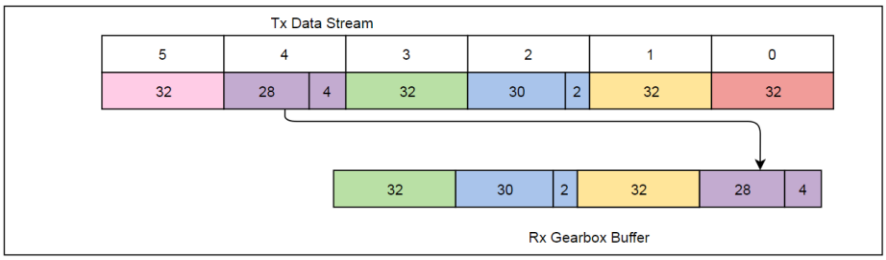
(b)



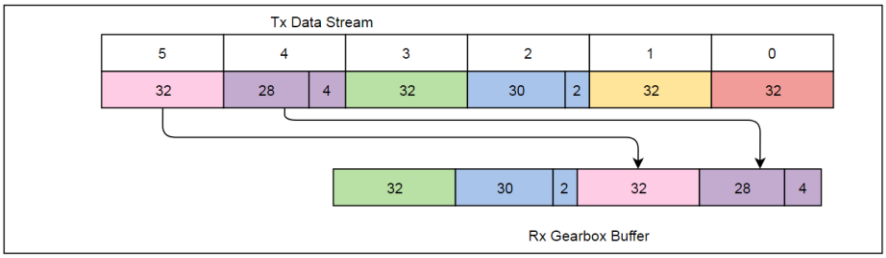
(c)



(d)



(e)



(f)

Figure 3.6: The Tx gearbox operation, along with the Rx gearbox buffer is described in (a) through (e).

Figure 3.6 (a) shows the beginning of a transmission from the Tx gearbox to the Rx gearbox. The details of serialization and deserialization are abstracted away for the purpose of demonstrating the high-level functionality. The figure shows a Tx data stream with 32-bit chunks labelled with corresponding numbers. 0 corresponds to the 32-bits in the red chunk, 1 corresponds to the 32-bits in the yellow chunk, and so on. The first encoded Aurora block (2-bit sync header + 64-bit scrambled data) is distributed across the first three chunks i.e. 0, 1, and 2. Chunks 0 and 1 contain the 64 bits of data, and chunk 2 contains the 2-bit sync header and 30-bits of data of the next encoded Aurora block. The Rx gearbox has just been initialized and is only receiving its first chunk, which goes into the 32 LSB of the Rx gearbox buffer. The rest of the buffer contains unknown data, meaning the data can be treated as don't cares.

Figure 3.6 (b) shows the next chunk in the Tx data stream being sent to the buffer. The rest of the operation is covered in Figures 3.6 (c) through (f). The important takeaway here is that the 66-bit blocks are distributed across the chunks. This is apparent in chunks 2 and 4, where some of the previous 66-bit block and some of the next 66-bit block is contained. Due to the way the buffer in the Tx is designed, the Tx gearbox inputs will eventually need to be paused for one block, allowing the Tx gearbox buffer to "catch-up" [16]. The pause operation is done every 32 blocks, meaning after 32 blocks are sent the gearbox inputs are paused for 1 block, resulting in 32 valid blocks for every 33 blocks. This has absolutely no bearing on the bandwidth of the link, since the 32-bit chunks are still being serialized during this one-block pause. Another way to put it is that the Tx gearbox needs to finish outputting the 32nd encoded block before accepting the next block. This is a consequence of appending a 2-bit sync header to the 64-bit data. In the layers above the Tx core, the pausing of the Tx gearbox will cause back-pressure on any FIFOs or storage elements containing data for transmission.

3.2.3: Output SERDES

The output SERDES, or OSERDES, is the final block in the Tx core. Its function is to serialize the data coming out of the Tx gearbox. The OSERDES is a primitive provided by Xilinx and can be customized for the required data rate. The OSERDES was customized using the Xilinx IP Generator and integrated into the Tx Core. Table 3.3 shows the settings used for the custom Aurora Tx OSERDES.

Table 3.3: Settings for the OSERDES

Property	Setting
Bandwidth	1.28 Gbps
Interface Template	Custom
Data Bus Direction	Output
Data Rate	DDR (Dual Data Rate)
Serialization Factor	8
External Data Width	1
I/O Signaling	Differential (LVDS)
clk_in	640 MHz
clk_div_in	160 MHz

3.3: Rx Core

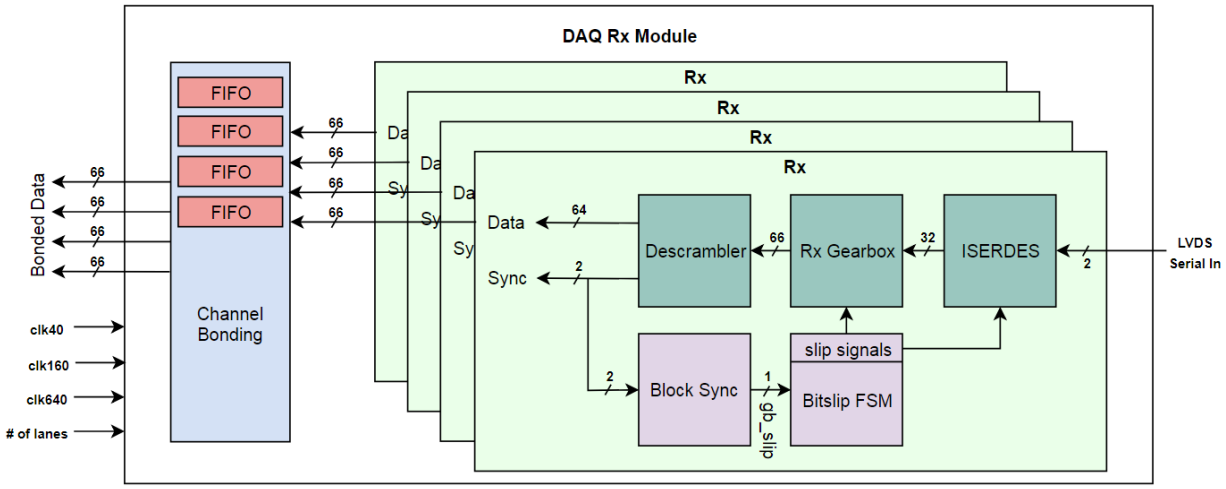


Figure 3.7: Four lane custom Aurora Rx block

The custom Aurora Rx core is depicted in Figure 3.7. As with the custom Aurora Tx core, the Rx core is encapsulated in a top-level module and is designed to be plug-and-play. The core supports 1 to 4 lanes at bitrates of up to 1.28 Gbps per lane. The top-level module can be subdivided into four instantiations of single lane Rx modules and one instantiation of a channel bonding module. Each lane receives a differential serial stream and is identical to all other lanes. Each individual Rx lane contains five submodules: ISERDES, Rx Gearbox, Descrambler, Block Sync, and Bitslip FSM. When more than one lane is used, a channel bonding module is necessary to compensate for differences in lane-to-lane signal arrival time [16]. The Rx core was tested in many different configurations, utilizing Xilinx provided Integrated Logic Analyzer (ILA) and Virtual Input/Output (VIO) debug cores. Additionally, the LCD on the board was used to display status information about the link when interfacing with the debug cores through JTAG was not possible. Bit-Error-Rate and Packet-Error-Rate utilities were also implemented to allow for performance evaluation of the link. The hardware tests showed successful transmission of data and channel bonding when four lanes were used. These results are covered in more depth in Section 3.4.

3.3.1: Input SERDES

The input SERDES (Serializer/Deserializer), or ISERDES, is the first block in the Rx core. The ISERDES function is to deserialize the incoming data stream with a deserialization factor of eight. Table 3.4 describes the settings used for the ISERDES. The settings are very similar to the OSERDES, with the data bus direction specified as ‘Input’ instead of ‘Output’. Additionally, an IDELAYE2 block precedes the ISERDES, allowing for delay control of the incoming serial stream at finite delay values.

Table 3.4: Settings for the ISERDES

Property	Setting
Bandwidth	1.28 Gbps
Interface Template	Custom
Data Bus Direction	Input
Data Rate	DDR (Dual Data Rate)
Deserialization Factor	8
External Data Width	1
I/O Signaling	Differential (LVDS)
clk_in	640 MHz
clk_div_in	160 MHz

The Xilinx ISERDESE2 primitive in 7 Series FPGAs can nominally perform 4x asynchronous oversampling at 1.25 Gb/s, as per Xilinx XAPP523 [17]. However, the bandwidth required by the custom Aurora protocol is 1.28 Gb/s. To solve this limitation in bandwidth, the Xilinx XAPP1017 was used [18]. The XAPP1017 utilizes the IDELAYE2 block and a per-bit deskew state machine to control the delay of the incoming serial data stream. This allows for a dynamic, self-adjusting system which tries to align the serial data to the sampling clock in the best possible arrangement [18].

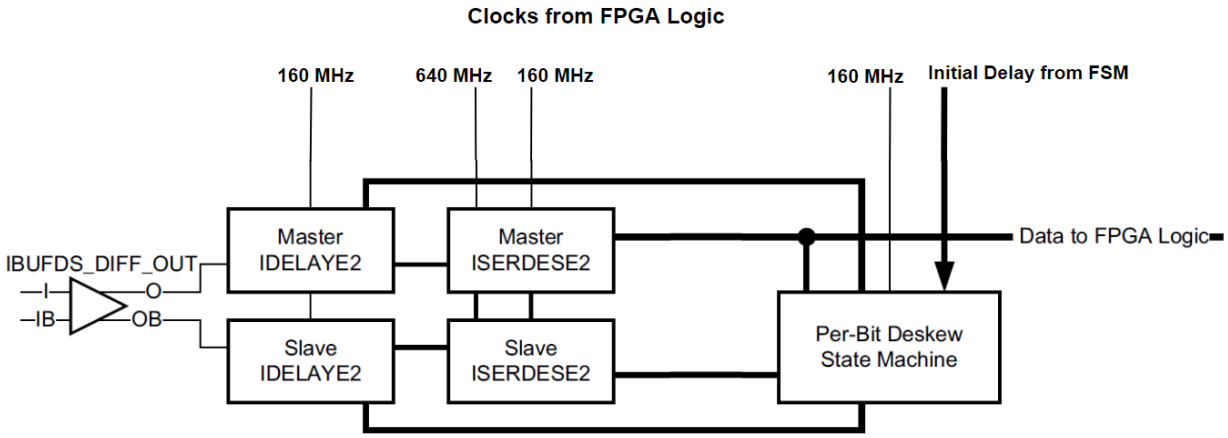


Figure 3.8: Modified Xilinx XAPP1017 Clock and Data Receiver Logic. Figure partially leveraged from XAPP1017 [18]

Figure 3.8 depicts a modified version of the clock and data receiver logic in the Xilinx XAPP1017, which is used in the custom Aurora protocol implementation [18]. Parts of the source code for the module, contained in the XAPP, were integrated into the custom Aurora protocol; however, many changes were made to accommodate the specific requirements of the RD53A to DAQ setup.

The input to the module is a differential serial stream, which is passed through an input buffer with a differential output, called IBUFDS_DIFF_OUT. After the serial stream is buffered, the negative and positive differential components are sent through separate IDELAYE2 and ISERDESE2 blocks, as shown in Figure 3.8. The “Master” block corresponds to the positive component and the

"Slave" block corresponds to the negative component. The IDELAYE2 block controls the delay of the incoming serial stream based on a tap value provided to the block. Using the tap value, the serial stream is delayed by a multiple of some Δt, which is dependent on the reference clock supplied to the IDELAYE2 block i.e. 200MHz, 300MHz, etc. After the serial stream goes through the IDELAYE2 block, it is deserialized in the ISERDESE2 block according to some deserialization factor. The deserialized data from both the Master and Slave circuitry is passed to the "Per-Bit Deskew State Machine", which controls the Master and Slave delay tap values on a per-bit basis [18]. A delay tap value that will sample as close to the middle of the eye as possible is desired, which the per-bit deskew state machine dynamically adjusts to try and achieve. The mechanism works in a feedback fashion, allowing for a self-regulating system. The specific details of how the delay tap values are changed are explained more comprehensively in the Xilinx XAPP1017 documentation [18].

The difference between the Xilinx module and the modified module used in the custom Aurora protocol is that the Xilinx module assumes an accompanying clock with the incoming data stream, which is not the case in the custom Aurora protocol. The custom Aurora implementation forwards the clock from the Rx to the Tx, and receives data from the Tx to the Rx, meaning the serial data stream coming into the Rx does not have an accompanying forwarded clock. Due to this difference, the circuitry that generates the clocks for the Master and Slave blocks from the incoming clock and the circuitry that trains on the incoming clock was removed. Instead, clocks from the FPGA logic are provided to the module.

3.3.2: Rx Gearbox

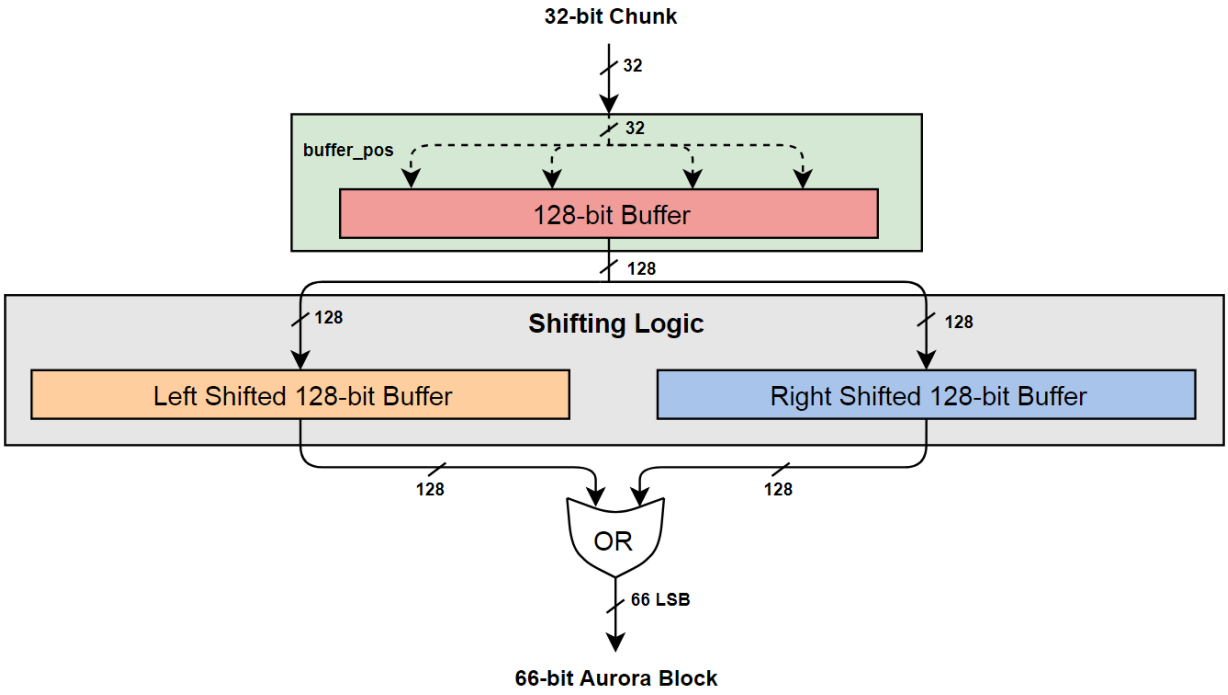


Figure 3.9: Rx Gearbox Functional Block Diagram

The Rx gearbox was not leveraged from the Xilinx Aurora IP and is a novel module that was designed to interface with the Tx gearbox in the Tx core. As a reminder, the Tx gearbox was used directly from the RD53A RTL chip code. The Rx gearbox uses a similar buffering technique used in the Tx gearbox, however the function is now the opposite (i.e. take 32-bit incoming data chunks and generate 66-bit blocks roughly every two 32-bit chunks). Figure 3.6 in Section 3.2.2 shows the Tx to Rx gearbox sequence. As with the Tx gearbox, the Rx gearbox outputs 32 valid Aurora blocks, followed by one block that should be ignored due to the buffer "catching up", giving 32 valid blocks every 33 blocks total. The Rx gearbox provides flow control to signify when the next output block should be ignored while the buffer is catching up. This again has no bearing on bandwidth, with the link operating at 1.28 Gb/s per lane.

The functional block diagram of the Rx Gearbox is shown in Figure 3.9. The Rx Gearbox contains a 128-bit buffer that stores 32-bit chunks in the sequence described in Section 3.2.2. The buffer contains the 66 bits necessary to generate the 66-bit Aurora block, however these bits may be out of order and spread across several 32-bit chunks. To align the 66 bits, the 128-bit buffer is shifted to the left and to the right, depicted by the "Shifting Logic" block. The left and right shift amount is calculated using an internal counter value. Further detail on how this is done can be found in the source code. Once the 128-bit buffer is shifted to the left and to the right, the intermediate shifted results are passed through a bitwise OR operation. The 66 least significant bits of the result correspond to the 66-bit Aurora block. Another thing to note is that the internal counter value can be slipped, affecting the shift values that will be calculated. This becomes relevant when trying to synchronize the Rx gearbox to the Tx gearbox. Figure 3.10 shows a detailed progression of the mechanism with the counter value, left and right shifts amounts, clock cycle, and 128-bit buffer state shown.

Counter	Shift Left	Shift Right
0	128	0
0	128	0
0	128	0
0	128	0
1	62	66
1	62	66
2	124	4
2	124	4

	Bit Distribution in Rx Buffer			
Clock Cycle	127 96	95 64	63 32	31 0
0	X	X	X	32
1	X	X	32	32
2	X	30 2	32	32
3	32	30 2	32	32
4	32	30 2	32	28 4
5	32	30 2	32	28 4
6	32	26 6	32	28 4
7	32	26 6	32	28 4

Figure 3.10: Rx Gearbox Shifting Mechanism Tables

Color Code:

- Blue, green, red, yellow is the order of the 66-bit incoming blocks
- Purple means a shift operation takes place during this clock cycle and a 66-bit block is generated
- Red Text represents the 32-bit chunk that was loaded during the clock cycle

The two tables show the shift values necessary to properly generate 66-bit blocks from the 32-bit incoming data chunks, and the state of the buffer during the shift operation. The table of the left-hand side contains three columns that keep track of the internal counter value, left shift value, and right shift value. The table of the right-hand side contains the contents of the buffer, bit assignments, data widths, and the current clock cycle. The 66-bit blocks are color coded based on the order they come in i.e. blue, green, red, yellow. In chunks where data is shared across two blocks, the chunks are subdivided and contain two colors with the number of bits associated with each color**.** The chunks where the bit number is red represent the chunk that was loaded during the respective clock cycle. The purple blocks in the clock cycle column represent the clock cycle where a 66-bit block was generated. The way to interpret this table is to look at the left and right shift values, shift the buffer in two separate instances by each shift amount in the appropriate shift direction, and perform an OR operation on the resulting shifts. The 66 least significant bits of the result represent the 66-bit Aurora block sent across. Due the scrambling operation on the Tx side, the 66-bit block is still 2-bit sync and 64-bit scrambled data at this point. The buffer needs to be fully loaded before the first shift operation can take place, which is why no shift operations take place until the 3rd clock cycle. The next section describes the descrambler and the process by which the data is descrambled. As a final note, the mechanism by which the shift left and shift right values are calculated can be studied in the code.

3.3.3: Descrambler

The descrambler comes after the Rx gearbox and descrambles the 64-bit scrambled data. As with the scrambler, the descrambler is also multiplicative and self-synchronizing [14]. The job of the descrambler is to perform the reverse operation done during scrambling, giving us the original data contained in the block on the Tx side. The descrambling module used in the custom Aurora implementation leverages the descrambler provided in the Xilinx Aurora IP, with some minor changes to accommodate for the RD53A setup. Figure 3.11 shows a high-level diagram of a descrambler given a polynomial.

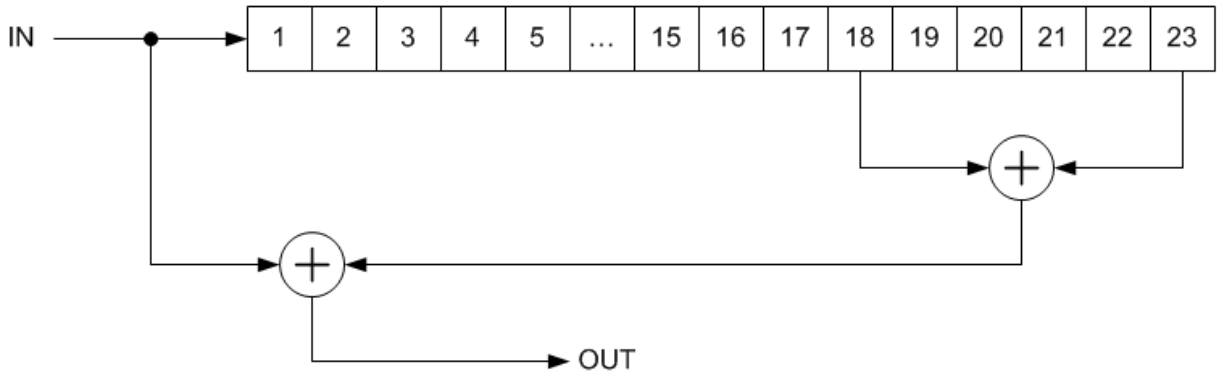


Figure 3.11: Example of a descrambler with function $1 + x^{18} + x^{23}$ [15]

The descrambler used in Aurora is specified by the same polynomial used in the scrambler, namely [14]:

$$G(x) = 1 + x^{39} + x^{58}$$

The exponents in the polynomial specify what tap values to use from the shift register when computing the out bit in the descrambler. The operation involves using these tap values in the descrambler polynomial buffer and XOR'ing them with the input scrambled bit.

3.3.4: Block Synchronization

The block synchronization module's purpose is to synchronize the Rx with the Tx. The module is leveraged from the Xilinx Aurora IP core and integrated into the custom Aurora implementation. The connection between the Tx and Rx is Simplex, meaning there is no communication from the Rx core to the Tx core i.e. no handshaking or similar provisions exist between the two cores. However, communication from the Rx to the Tx is not needed, as the Rx can be synchronized using only the incoming 2-bit sync header.

As a reminder, the 2-bit sync headers are not scrambled on the Tx side, only the 64-bits of block data. On the receiver side, when the Rx gearbox outputs the 66-bit blocks, the output is comprised of a 2-bit sync header and 64-bits of scrambled data. Following the Rx gearbox module, the 64-bits of data goes through the descrambler logic and the 2-bit sync header is passed through directly. Valid sync headers are either 10 or 01 in binary, meaning those are the only sync headers that should be seen on the receiver if synchronization is achieved. An invalid sync header of 11 or 00 indicates that the Rx core needs to adjust itself.

The block synchronization module counts the valid and invalid sync headers received. If the valid sync count equals some user defined value and no invalid sync headers were received, the link is considered synchronized. However, if the link is not yet synchronized and a single invalid sync header is received, a 'rxgearboxslip' signal is pulsed, and the internal counters are reset. Finally, if the link is synchronized, some number of invalid headers, specified by the user, are permissible before the link is required to resynchronize.

Two components in the Rx core can be adjusted when synchronizing to the Tx core: the ISERDES and the Rx Gearbox. When these components are properly adjusted, the incoming serial data stream will be properly deserialized and the 66-bit Aurora blocks will be properly assembled and descrambled. The Bitslip FSM module shown in Figure 3.7 acts as the interface that adjusts the ISERDES and the Rx Gearbox. The Bitslip FSM module was not leveraged from the Xilinx Aurora IP and is a novel design that was developed to address the fact that the custom Aurora implementation contained two components that needed to be slipped.

This module takes the 'rxgearboxslip' signal from the block synchronization module that tells it a component needs to be adjusted. Due to a deserialization factor of 8, the ISERDES can be bit-slipped 8 times before reaching the original bit orientation. The Rx gearbox contains an 8-bit counter used to calculate shift values, shown in the table in Figure 3.10, that can be slipped 129 times before all legal counter values have been tested. The mechanism for slipping these components is coordinated in a nested fashion. For every 8 ISERDES slips, the Rx Gearbox is slipped once, eventually exhausting all combinations. The Bitslip FSM goes through the slipping process continually, which allows for synchronization even if the Tx core is initiated much after the Rx core.

3.3.5: Channel Bonding

The final component of the custom Aurora Rx core is the channel bonding module that is needed when more than one lane is used. This module was not leveraged from the Xilinx Aurora IP and is a novel design, accounting for differences in the arrival time of the serial signal received by each lane. These differences can be caused by mismatched PCB traces, cable mismatches, etc.

As mentioned earlier, due to the sync header overhead, the Tx gearbox inputs need to be paused for one block every 32 blocks. As a result, there is one Aurora block generated by the Rx Gearbox that is ignored for every 32 valid Aurora blocks, since the Rx buffer needs to catch up internally. The Rx will signal the components upstream that the block should be ignored. The goal is to make sure the valid blocks are bonded so that the four blocks across each lane correspond to the original four blocks sent by the Tx. In the case where the channel is not bonded, improper states can occur where one or more blocks across the four lanes arrive a block late or are invalid at different points in time.

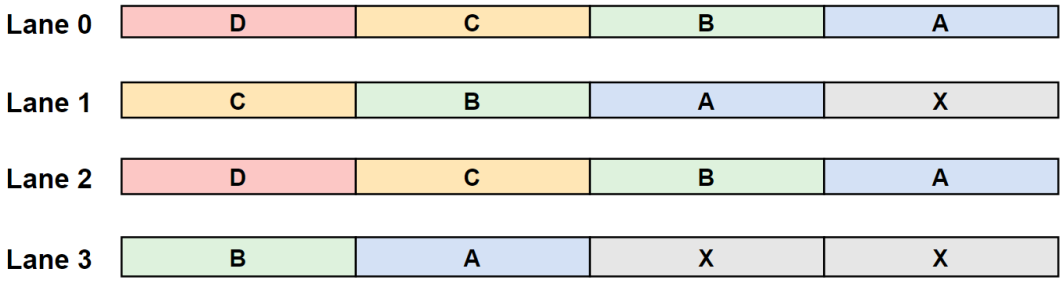


Figure 3.12: Four lane Aurora stream with misaligned blocks

Figure 3.12 shows data being sent across four lanes. Letters A through D are used to label the Aurora blocks, with A arriving first. The letter X depicts a don't care and can be ignored for the purposes of this discussion. When the channel is properly bonded, the blocks should arrive at the same time across all four lanes i.e. every lane has block A, followed by block B, etc. However, in the figure this is not the case. Lane 0 and lane 2 are properly aligned to each other, but lane 1 and lane 3 are not aligned to each other or any of the other lanes. The channel bonding module fixes this problem and outputs the correct Aurora blocks on every lane.

The Aurora protocol specification states that a unique 'Channel Bonding' frame should be sent by the Tx across all four lanes at some interval [16]. If there are no differences in the arrival times of the serial stream across the four lanes, the channel bonding frames should arrive at exactly the same time. However, this is not realistic in a physical implementation, and the differences in arrival times will manifest in blocks arriving one or more blocks late, or the Rx one block ignore point occurring at different points across the four lanes. The difference in ignore points is due to synchronization potentially configuring the ISERDES and Rx Gearbox blocks to different slip values across the four lanes.

The channel bonding module uses four first in, first out (FIFO) buffers, one per lane. When the link is not properly bonded, the 'channel_bonded' signal is LOW. In this state, the channel bonding module will search for unique channel bonding blocks in each lane. If no such block is present,

the FIFO will be read continuously, and the blocks will pass through without accumulating in the FIFO. However, if a channel bonding block is seen on any lane, the FIFO in that lane will not be read until every lane has received a channel bonding block. This means that the lanes that are ahead will effectively wait for the lanes that are behind to catch up i.e. present a channel bonding block. Once this state is achieved, the FIFOs will be read in unison, using the same read signal. The final nuance is the fact that the Rx receives 32 valid blocks, followed by one block being ignored via flow control. In the channel bonding module, this characteristic of the Aurora protocol manifests itself in one or more FIFOs becoming empty when there is an invalid block. To maintain proper channel bonding, reading of the FIFOs must be paused after the first FIFO(s) becomes empty. The empty signal of that specific FIFO(s) will be used to pause reading of all FIFO in the channel bonding module.

3.4: Simulation and Hardware Testing

Testing of the Rx core was performed at different bitrates, across several different cable setups, and with a variety of data (incrementing data, user specified data, controlled invalid packets, etc). Packet-Error-Rate (PER) and Bit-Error-Rate (BER) debugging functionality was designed and implemented in test Vivado Projects, allowing for monitoring of the performance of the link. The Xilinx Integrated Logic Analyzer (ILA) and Virtual Input/Output (VIO) debug cores were utilized to perform more advanced testing and debugging in the hardware [19].

Before any hardware evaluation of the Tx and Rx cores was performed, the custom Aurora design was tested extensively in simulation using ModelSim. Xilinx Vivado simulation libraries were compiled and allowed for simulating the Xilinx IP cores contained in the design. Many simulations were created to provide testing at different levels of granularity i.e. Tx gearbox to Rx gearbox, Tx core to Rx core.

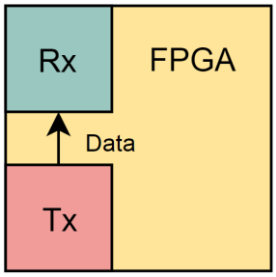


Figure 3.13: Single FPGA test with Tx and Rx custom Aurora protocol

Figure 3.13 depicts the first test performed in hardware. The Tx and Rx core were instantiated onto a single FPGA and serial data was transmitted at a much lower, 320 Mb/s, bitrate. This basic test eliminated a lot factors that needed to be considered later, such as poor cable performance, different clock domains, and differential termination.

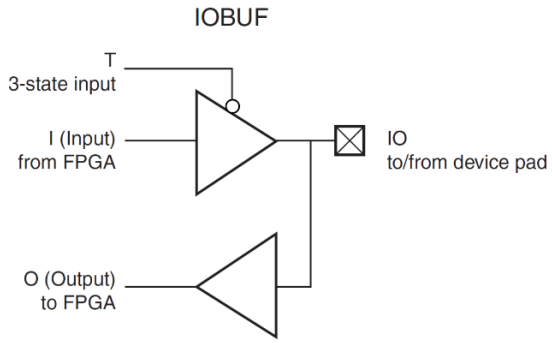


Figure 3.14: Xilinx Input/Output Buffer Primitive (IOBUF) [20]

One of the challenges in performing this test was the fact that the Tx and Rx cores utilized the OSERDES and ISERDES Xilinx IP core, which interface with the I/O pins of the FPGA. A workaround was developed by first replacing the OSERDES used in the Tx core with a soft custom serializer provided by Dr. Timon Heim from Lawrence Berkeley National Lab. The custom serializer was compared to the OSERDES IP in simulation to make sure the functionality was the same. A Xilinx Input/Output Buffer (IOBUF) primitive was used, shown in Figure 3.14, to allow driving data into the ISERDES on a single FPGA without failing constraints. The input is specified as the Tx serial stream, the output is fed into the ISERDES, and the T signal is always held LOW, allowing for the input to go directly to the output. After this workaround was put in place, the design was successfully tested in hardware.

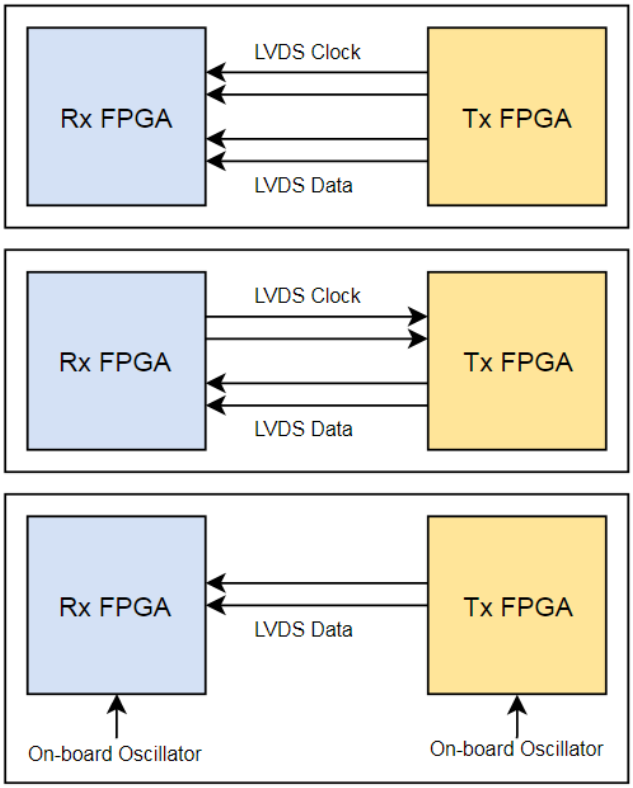


Figure 3.15: Board to Board configurations tested

Figure 3.15 shows the different configurations tested in hardware at 320 Mb/s. In the first configuration, the Tx FPGA forwards a 160 MHz LVDS clock, which is used to derive the clocks in the fabric of the Rx FPGA. In the second configuration, the opposite is the case, where the Rx FPGA forwards a 160 MHz LVDS clock from which the Tx FPGA clocks are derived. The final configuration derives the clocks in the Tx and Rx FPGAs using their respective on-board oscillators, with no clock forwarding between the two FPGAs. The first two configurations were successful, with synchronization and proper data transmission achieved over extended periods of time. In the final configuration, synchronization is achieved periodically, but permanent synchronization is never achieved. This can be attributed to phase-drift in the LVDS data stream being sent to the Rx core.

The second configuration was chosen for testing of the custom Aurora Tx and Rx cores at higher bitrates because the configuration most closely resembles the setup of the RD53A chip to DAQ system. To clarify, the actual DAQ system will not forward a 160 MHz clock to the RD53A chip. However, the DAQ system does send a serial stream of Timing, Trigger, and Control (TTC) data at 160 Mb/s, which is used to recover a 160 MHz clock in the RD53A chip [10]. Forwarding a 160 MHz clock from the Rx core (DAQ) to the Tx core (RD53A IC) is functionally similar to the actual setup, considering that the Tx and Rx cores are being tested in standalone tests.

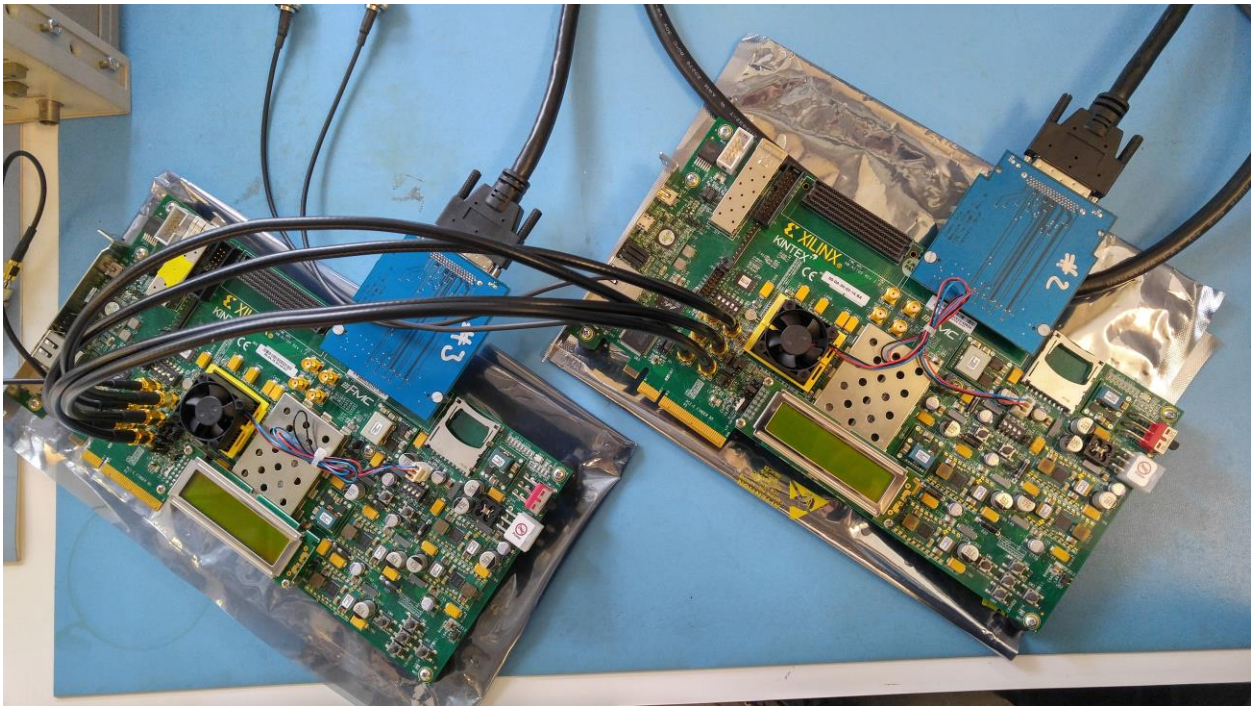


Figure 3.16: KC705 to KC705 Board setup with SMA and FMC communication links

The Xilinx KC705 Development Boards were used for testing the Tx and Rx cores. The boards are flexible, allowing for tests over SMA and FMC communication links. Figure 3.16 shows two KC705 boards connected over SMA and FMC interfaces. The SMA wires are the six black wires with gold tips going from one board to the other. The FMC daughter cards are the blue cards with "#3" and "#2" written on them. In the FMC interface, the FMC to VHDCI daughter cards are used

to transmit data over a VHDCI cable [21]. The FMC to VHDCI daughter cards were developed by Dr. Timon Heim at Lawrence Berkeley National Lab [21].

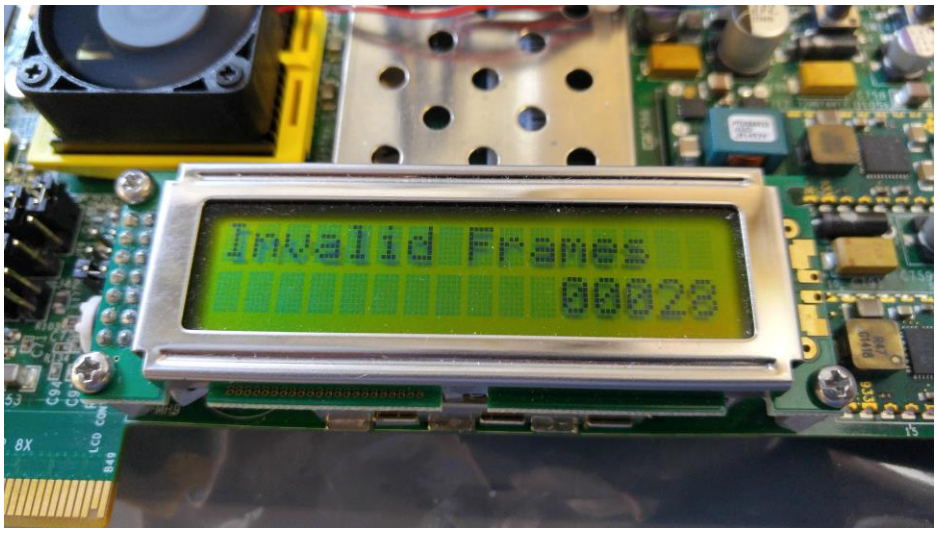


Figure 3.17: Invalid frames displayed on LCD screen. Image depicts invalid frames being sent deliberately to test proper functionality of LCD.

The ILA and VIO debug cores require a JTAG connection with a PC. In some extended test cases, when the tests lasted over a week, a JTAG connection to a PC was not always available. To eliminate the need for the PC, but still provide status information on the link, the Liquid Crystal Display (LCD) screen was used to display the number of invalid frames received by the Rx (Figure 3.17). This required an LCD driver that can take the binary invalid frames value in the design and represent it in base 10 decimal. A generic LCD driver was found online and modified to display the "Invalid Frames" text and the decimal representation of the number of invalid frames received [22].

Table 3.5: Summary of Hardware Tests

Interface	Cable	Lanes	Bitrate (Mb/s)	Success (Yes/No)
SMA	SMA	1	1280	Yes
FMC to VHDCI Daughter Card	VHDCI	4	640	Yes
FMC to VHDCI Daughter Card	VHDCI	4	1280	No
CERN I/O Buffer Daughter Card	VHDCI	4	640	Yes
CERN I/O Buffer Daughter Card	VHDCI	4	1280	No

The summary of the hardware tests is shown in Table 3.5. The table describes the hardware interface, cable, number of lanes, bitrate, and whether the test was successful or not. The highest bitrate at which a test succeeded is included in the table, which is to say that tests at other bitrates may have been performed but are not listed in the table. As described earlier, the second configuration in Figure 3.15 was used when performing these tests.

Although the FMC to VHDCI Daughter Card and the CERN I/O Buffer Daughter Card both failed when the link was configured to 4 lanes at 1.28 Gb/s, these will not be the cards used in the final DAQ to RD53A chip setup [21] [23]. A DisplayPort FMC daughter card has been developed that

seeks to address the issues of the other cards, namely poor or nonexistent buffering and poor shielding in the cable [24]. However, the two aforementioned cards were successfully tested when configured to 4 lanes at 640 Mb/s.

3.5: Integrating the Tx Core into the RD53A FPGA Emulator

Once the custom Aurora protocol was tested in standalone hardware tests and confirmed functional, the Tx core was integrated into the RD53A FPGA Emulator described in Chapter 3. The emulator was tested in simulation, with a corresponding Rx cores instantiated in the top-level of the testbench. The simulation showed proper synchronization in the Rx core using the Aurora Tx blocks generated in the emulator and transmitted using the Tx core.

Chapter 4: Cable Testing Infrastructure

After the custom Aurora Tx and Rx modules were successfully tested in a single-lane SMA setup at 1.28 Gb/s, the tests were expanded to four lanes over other connectivity options, such as FPGA Mezzanine Cards (FMC). A repository was created containing Xilinx Vivado projects for various hardware setups. Tutorials on how to setup the link and use the debugging utilities are provided in the repository for the SMA and FMC hardware setups.

4.1: Motivations

The motivation for creating a cable testing repository was to help with the bring-up of future DAQ systems, where DAQ hardware and/or RD53A chips were not yet available. The idea is that the custom Aurora protocol Tx and Rx cores may be used for evaluating the performance of various cable setups, while fully taking advantage of the VIO and ILA debugging utilities, as well as the custom bit-error-rate and packet-error-rate debugging link performance metrics

4.2: Repository Structure

The repository structure separates the different hardware setups into their own folders, which contain a corresponding Tx and Rx Xilinx Vivado project. Each of the projects is self-contained and does not depend on any other project or resources.

4.2.1: SMA Single Lane at 1.28 Gb/s

Two Xilinx Vivado projects have been developed for setting up a single lane Aurora link at 1.28 Gbps. A Xilinx KC705 board acts as the Tx and Rx platform and uses Kintex 7 FPGAs. Data and clock are transmitted over SMA cables. The SMA connections can be found on the board and are labeled as described below.

SMA Connections:

- Data: USER_GPIO_P, USER_GPIO_N
- Clk: USER_CLK_P, USER_CLK_N

The connections go from one board to the other to their corresponding names e.g. USER_GPIO_P on the Tx KC705 connects to USER_GPIO_P on the Rx KC705. Once the hardware is properly setup and the bitstreams are loaded to the FPGAs, the ILA and VIO can be used to monitor and stimulate the link. If everything is setup properly, valid data should be observed on the Rx side. The Rx data will correspond to the Tx data.

4.2.2: FMC Four Lane at 640 Mb/s

Projects were also developed for setting up a four lane Aurora link at 640 Mbps. A Xilinx KC705 board acts as the Tx and Rx platform and uses Kintex 7 FPGAs. Data is transmitted over FMC and clock is transmitted over SMA cables. The SMA connections can be found on the board and are labeled as described below. The connections go from one board to the other to their corresponding names.

SMA Connections:

- Clk: USER_CLK_P, USER_CLK_N

The FMC card needs to go on either the FMC High Pin Count (HPC) mezzanine connector or the FMC Low Pin Count (LPC) mezzanine connector. The difference between these connectors is the pin count (HPC = High Pin Count, LPC = Low Pin Count). All of the data LVDS pairs are set to locations compatible with both HPC and LPC, so either connection can be used. The firmware in its current state is setup for the HPC connection, but location code is in the constraints file for both HPC and LPC. Comment out the one you don't need and uncomment the one that's being used.

The biggest differentiator with four lanes as opposed to one is that the lanes need to be channel bonded. As a result, a channel bonding module is included in the firmware, which adapts to for differences in the arrival time of the signal. These differences can be cause by mismatched PCB traces, cable mismatch, etc. Once the channel is bonded, the 'channel_bonded' signal will be asserted HIGH.

4.2.3: Aurora Rx Brute Force Alignment

The Aurora Rx brute force alignment project uses the same hardware setup as the single lane SMA setup at 1.28 Gb/s. As described in Section 3.3.1, the Rx deserializer block uses an IDELAYE2 Xilinx primitive to delay the incoming data stream. The IDELAYE2 block gives the user the option to specify the specific amount of delay desired by setting a 5-bit tap value, giving a total of 32 delay taps.

With the single lane SMA setup in Section 4.2.1, the Xilinx XAPP controls the tap values automatically, whereas with the Aurora Rx brute force alignment project the tap values may be specified by the user using the Xilinx VIO interface. This allows the user to set a desired tap value by either giving a specific 5-bit tap value directly or allowing the tap-values to be stepped through automatically using the brute force sequence designed in the firmware. This project allows for more control but should really only be used for debugging purposes.

4.3: Automating the Build Procedure

Automating the build procedure of the custom Aurora hardware tests and the RD53A emulator can allow for more control over the specific requirements of the project, such as what warning should be elevated to an error. Some headway was made in automating the build procedure of the Aurora projects using the CERN build framework, hdlmake, with a successful compilation of an RD53A Emulator build tested. However, additional work is required to flush out all the bugs and tidy up the process.

Chapter 5: Conclusion and Future Work

The work contained in this thesis describes the development of a custom Aurora protocol in anticipation for the RD53A chip. The RD53A chip is a research development chip at CERN designed to test novel detector technologies in anticipation for the ITk Upgrade. The work done on the custom Aurora protocol was leveraged to create a cable testing repository. The custom Aurora protocol Tx core was integrated into the RD53A FPGA Emulator.

Throughout the development of the custom Aurora protocol, many new modules, such as the Rx Gearbox and Channel Bonding modules, were designed and tested. Existing IP and firmware was also leveraged where it made sense, such as using the RTL from the RD53A chip for the Tx Gearbox or using the Xilinx Scrambler from the Xilinx Aurora 64B66B IP. As the project evolved, simulations in ModelSim were performed every step of the way.

When the cores reached maturity and simulations showed proper behavior, hardware tests across different platforms and in varying configurations were performed. To aid with characterizing and debugging the performance of the custom Aurora protocol, Xilinx ILA and VIO debugging cores were used, bit-error-rate and packet-error-rate utilities were implemented, and an LCD display was utilized for displaying the number of invalid packets received. These tools helped aid in debugging communication issues and understanding the areas where further development was needed.

As a result of the hardware tests and the infrastructure developed to facilitate them, a cable testing repository was created. The repository allows for preliminary testing of cable setups and gives the user some understanding of the cable performance. Several hardware configurations are supported and include tutorials in the repository. A more specialized debugging configuration is also provided, allowing the user to have finer control over some elements of the communication link.

Finally, the custom Aurora Tx core was integrated into the RD53A emulator and preliminary simulations showed proper synchronization. Hardware tests still need to commence to test certain aspects of the connection such as channel bonding more extensively.

Throughout the development process of the custom Aurora protocol, three new repositories were created, namely aurora_tx, aurora_rx, and cern_cable_test (Links in Appendix A). The aurora_tx and aurora_rx repositories contain the code and IP cores for the respective Tx and Rx core. These repositories are meant to be integrated into a build framework that can pull the repos and integrate them into a project. Further testing of building the cores using something such as CERN's hdlmake can help in making the repositories suited for easier integration.

Future Work

While the custom Aurora protocol is functional and has been tested extensively, there is still work that can be done to improve it. Some of the modules, such as the channel bonding module, can be refactored to reduce the overhead. On that same note, the code can also be made more modular and configurable in the process, allowing for a broader range of applications. Another improvement that can be made is the removal of the 40 MHz clock from the design. The clock is used in the design due to some early modules, such as the scrambler, making use of it. However,

it is feasible to remove the clock altogether since the design still uses a 160 MHz clock throughout. This can reduce some clock routing overhead at the cost of some enable logic overhead. Finally, some requirements for the RD53A chip evolved and changed throughout the development of the custom Aurora protocol, so an evaluation of the most up-to-date specification would be important if further work were to continue.

Bibliography

[1] G. Aad et al, "The ATLAS Experiment at CERN Large Hadron Collider," *The ATLAS Collaboration, Journal of Instrumentation,* vol. 3, August 2008.

[2] C. De Melis, "CERN Document Server," 2016. [Online]. Available: https://cds.cern.ch/record/2119882?ln=en.

[3] "CERN Document Server," CERN, [Online]. Available: https://home.cern/about/engineering/restarting-lhc-why-13-tev.

[4] C. a. O. G. Education, "LHC the guide," CERN, 2017. [Online]. Available: http://cds.cern.ch/record/2255762/files/CERN-Brochure-2017-002-Eng.pdf.

[5] J.-L. Caron, "CERN Document Server," May 1998. [Online]. Available: https://cds.cern.ch/record/841485?ln=en.

[6] N. L. e. a. Whallon, "Upgrade of the YARR DAQ System for the ATLAS Phase-II Pixel Detector Readout Chip," *PoS TWEPP-17,* vol. 076, 2018.

[7] C. ATLAS, "Technical Design Report for the ATLAS Inner Tracker Strip Detector," *Detectors and Experimental Techniques,* 2017.

[8] T. Flick, "The Phase II ATLAS Pixel upgrade: the Inner Tracker (ITk)," *IOP Science,* vol. 12, 2017.

[9] M. z. Nedden, "The LHC Run 2 ATLAS trigger system: design, performance and plans," *IOP Science,* vol. 12, 2017.

[10] The RD53 Collaboration, "The RD53A Integrated Circuit," CERN-RD53-PUB-17-001, 2017.

[11] National Instruments, "High-Speed Serial Explained," NI, 2016.

[12] Teledyne LeCroy, "64b/66b Symbol Decode," Teledyne LeCroy, 2012.

[13] Xilinx Inc, "Aurora 64B/66B v10.0 PG074," Xilinx Inc, 2015.

[14] Xilinx Inc, "Aurora 64B/66B Protocol Specification SP011," Xilinx Inc, 2014.

[15] Wikipedia, "Scrambler," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Scrambler. [Accessed 28 5 2018].

[16] Xilinx Inc, "7 Series FPGAs GTX/GTH Transceivers UG476," Xilinx Inc, 2016.

[17] M. Defossez, "LVDS 4x Asynchronous Oversampling Using 7 Series FPGAs and Zynq-7000 AP SoCs," Xilinx Inc, 2017.

[18] M. Defossez and N. Sawyer, "LVDS Source Synchronous DDR Deserialization (up to 1,600 Mb/s) XAPP 1017," Xilinx Inc, 2016.

[19] Xilinx Inc, "Programming and Debugging UG908," Xilinx Inc, 2014.

[20] Xilinx Inc, "7 Series FPGAs SelectIO Resources UG471," Xilinx Inc, 2018.

[21] T. Heim, "CERN Twiki," CERN, [Online]. Available: https://twiki.cern.ch/twiki/bin/view/Main/TimonHeim?forceShow=1#FMC_to_VHDCI_Rev_B.

[22] C. Talarico, Eastern Washington University, [Online]. Available: https://web.ewu.edu/groups/technology/Claudio/ee360/Cad/lcd_ct-1.vhdl.

[23] G. Kasprowicz, "FMC DIO 32ch LVDS a, Open Hardware Repository," 26 5 2016. [Online]. Available: https://www.ohwr.org/projects/fmc-dio-32chlvdsa/wiki/wiki.

[24] M. Cohen-Solal and J. Jimmy, "Testing RD53 at Lal 140318," 2018.

Acknowledgements

I would first and foremost like to thank my advisors, Dr. Scott Hauck in the Electrical Engineering department and Dr. Shih-Chieh Hsu in the Physics department. Without their guidance in the many technical aspects of the project, both in Digital Design and Physics, this thesis would not have been possible. Additionally, their support and guidance in the non-technical aspects, such as communication, scheduling, and logistics has proven invaluable. The opportunity to work at CERN and Lawrence Berkeley National Lab is something that I will remember for the rest of my life and I owe those experiences to my advisors.

I would also like to thank the RD53A group at Lawrence Berkeley National Lab, with whom the development of the custom Aurora protocol was done in collaboration. I had the opportunity to work at the lab during the summer of 2017 and am incredibly grateful for this experience. I would like to thank Dr. Timon Heim and Dr. Maurice Garcia-Sciveres, who helped get me get situated at the lab and provided many valuable technical insights throughout the development of the project. The experience at the lab was very memorable and I was fortunate to work with such a great collective of people.

Although not mentioned in this thesis, I had the opportunity to work with the ATLAS New Small Wheel (NSW) Muon Electronics group during my 7 weeks stay at CERN. I would like to thank the following members of the group: Panagiotis Gkountoumis, Christos Bakalis, Paris Moschovakos, George Iakovidis, Theodoros Alexopoulos, and Venetios Polychronakos. They were very welcoming and took the time out of their busy schedules to get me up to speed. Throughout the development of the project assigned to me there, a Dynamic IP Configuration module, they helped me understand a lot of the key concepts that are critical to the FE and DAQ electronics in the LHC.

Additionally, I would like to thank my office mate Logan Adams for helping me with the logistics of getting to CERN, since our trips there overlapped for roughly 4-6 weeks. Being a year ahead of me, he provided guidance in the technical, logistical, and organizational aspects of the work done at the lab, as well as general advice for graduate school.

Finally, I would like to thank my family for supporting me throughout the entirety of my graduate experience: my father Sergey Kurilenko, my mother Olga Kurilenko, and my siblings Anton Kurilenko, Nikita Kurilenko, Viktoriya Kurilenko, Andrey Kurilenko, and Dennis Kurilenko. Without their continued support, I would not have been able to finish my graduate degree.

Appendix

Repositories

- https://bitbucket.org/levkurilenko/aurora_rx
- https://bitbucket.org/levkurilenko/aurora_tx
- https://bitbucket.org/levkurilenko/cern_cable_test
- https://bitbucket.org/levkurilenko/CERN_work
- https://bitbucket.org/levkurilenko/acme_lhc_docs

Ownership of the repositories may have been transferred over to my lab colleague, Dustin Werran, at:

https://bitbucket.org/DustinWerran/

If the repositories are not found under my name, please check there.

Tx Gearbox Shift Equations

- Right Shift Amount = $(counter[6:0] * 32)\%132$
- Left Shift Amount = $132 - (counter[6:0] * 32)\%132$

Rx Gearbox Shift Equations

- Right Shift Amount = $(counter[7:1] * 66)\%128$
- Left Shift Amount = $128 - (counter[7:1] * 66)\%128$