Development of an FPGA Emulator for the RD53B Chip

Niharika Mittal

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

Master of Science in Electrical Engineering

University of Washington

2020

Committee:

Scott Hauck

Shih-Chieh Hsu

Program Authorized to Offer Degree:

Electrical and Computer Engineering

University of Washington

ABSTRACT

Development of an FPGA Emulator for the RD53B Chip

Niharika Mittal

Chairs of Supervisory Committee:

Professor Scott A. Hauck
Electrical Engineering

Assistant Professor Shih-Chieh Hsu
Physics

The world's largest particle accelerator, called Large Hadron Collider (LHC), will go through a long shutdown (LS) in 2024. Particle detectors and systems will be upgraded in order to increase the particle collision rate. So far, the LHC has been able to complete the standard model of Physics by detection of the Higg's Bosun, but after the new upgrades it is expected to contribute towards the exploration of new physics. One such upgrade will be seen in the ATLAS detector's Inner Tracker (ITk), where the entire tracking system will get replaced with improved systems. A second prototype version of a test chip for the new hardware is being developed by the RD53 collaborators, called as RD53B (succeeding the RD53A). The Adaptive Computing Machines and Emulators (ACME) Laboratory at the University of Washington develops and maintains the Field Programmable Gate Array (FPGA) focused emulator of the test chips. The primary reason for this effort is the inadequate availability of the chip to researchers, the inability to generate realistic data, and the time it takes to make any changes to the real system. The emulator resolves all these issues faced by physical chips, as it is in an open source repository, produces realistic data without need of a radiation source, and takes hours instead of months to make changes to the System Verilog code. The necessary background information, emulator design, and work done on it is explained in the thesis.

TABLE OF CONTENTS

CHAPTER 1

LARGE HADRON COLLIDER

1.1 INTRODUCTION

The quest to find answers to unsolved scientific questions has led mankind to build a machine which deserves to have word “large” in its name, as it not only weighs over 38,000 tons, but runs for 27km (16.5mi) in a circular tunnel 100 meters beneath the ground. The Large Hadron Collider (LHC) is by far the world’s largest and most powerful particle accelerator laboratory built to date. It is located at the European Organization of Nuclear Research (CERN), and straddles of the borders of Switzerland and France just outside of Geneva, CH in Central Europe as shown in figure 1 (below). CERN is an international scientific and research laboratory which is dedicated to the pursuit of fundamental science. It has provided remote access to its facilities to researchers around the globe by developing wide area network, which gave birth to the World Wide Web [1].

Figure 1: Circular vision: Aerial terrain view with an overlay of the Large Hadron Collider and its experiment sites (Courtesy: CERN)

The LHC was designed with an aim to allow physicists to test the predictions of different theories of particle physics, such as completing the Standard Model by measuring the properties of the Higgs Boson and searching for new particles predicted by supersymmetric theories. In 2012, physicists were able to detect the Higgs Boson particles which led to the completion of the Standard Model. With this, the focus of LHC is now moved to explore beyond the standard model, often referred as “new physics”.

Normal operation of the LHC starts with stripping electrons from hydrogen atoms to obtain protons by passing hydrogen through an electric field. Being positively charged particles, protons will accelerate when in an electric field and bend in a circle when in a magnetic field. After being ionized, protons are passed on to LINAC2, a machine that accelerates beams of protons into an accelerator called the PS booster which accelerates the protons. Once the beam of proton reaches the right energy level, the PS booster injects it into another accelerator called the Super Proton

Synchrotron (SPS). By now, the beam is travelling nearly at the speed of light, and is grouped in small bunches, with each bunch containing 1.1 x 10^{11} protons. The SPS splits and injects the beam into the LHC, with one beam travelling clockwise and the other going anti-clockwise. The two beams converge at one of the four collision sites: ALICE, ATLAS, LHCb, and CMS. The facility (LHC) is capable of creating 600 million bunch collisions per second. The CERN Accelerator complex is depicted in the figure 2 (below). During the collisions, energy contained in the two colliding protons converts into mass through Einstein's mass-energy relationship. Each collision site has unique equipment and resources to measure and collect data from the collisions occurring there [1].

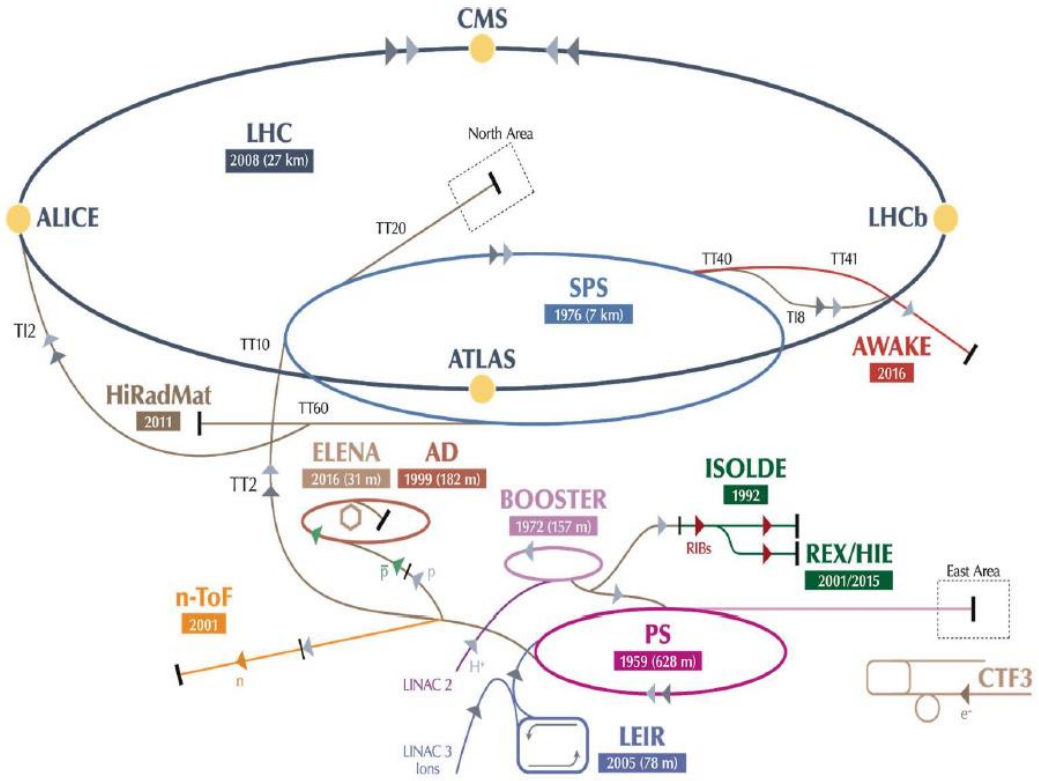


Figure 2: Proton trajectory: protons are first released at LINAC2, passing through the booster, the PS, the SPS, and then the LHC, where they make 11,245 trips around the LHC every second.

1.2 INNER TRACKER PIXEL UPGRADE

One of the eight major experiments at the LHC is the ATLAS (A Toroidal LHC ApparatuS), which is of primary interest to this thesis. It is the largest general-purpose particle detector ever constructed for a particle collider, shaped like a cylinder 46 meters long, 28 meters in diameter, weighing almost 7000 tons.

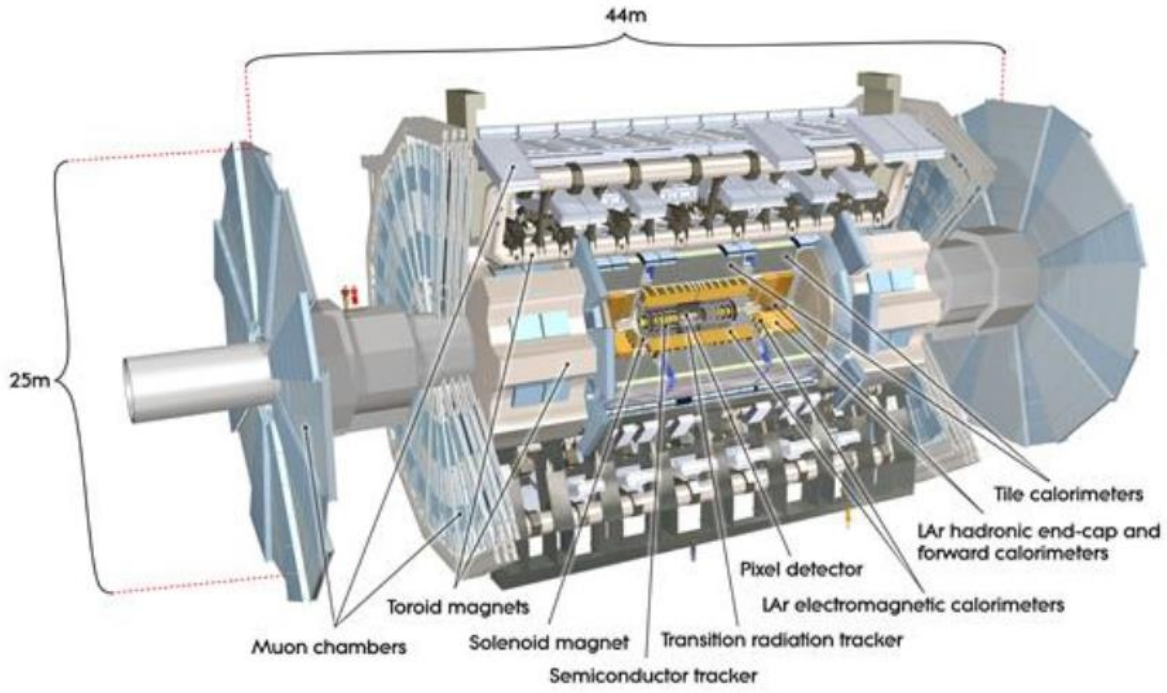


Figure 3: Computer generated image of the whole ATLAS generator showing all the layers of the detector in detail [1]

Over a billion particle interactions take place in the ATLAS detector every second, generating roughly 30 petabytes of data per year [1]. This amount of data storage is not feasible; ATLAS uses a complex series of data analysis which results in saving data for only 1 in a million events. The detector is a many-layered instrument (see figure 3) designed to record the trajectory, momentum and energy of particles, allowing them to be individually identified and measured [3]. It consists of six different detecting subsystems wrapped concentrically in layers around the collision points. The four major components of the detector are the Inner Detector, the Calorimeter, the Muon Spectrometer, and the Magnet System.

The Inner Detector is the innermost layer of the ATLAS and sees the decay product of the collisions prior than any other layer of the detector. It is therefore very compact and sensitive [1]. It is further divided into 3 different components (listed from inner to outer): The Pixel Detector (Pixel), the SemiConductor Tracker (SCT), and the Transition Radiation Tracker (TRT). This can be seen in figure 4. The basic function of this layer is to track charged particles by detecting interactions with the sensors at distinct points. This detailed information is then used to measure the direction, momentum and charge of those electrically charged particles.

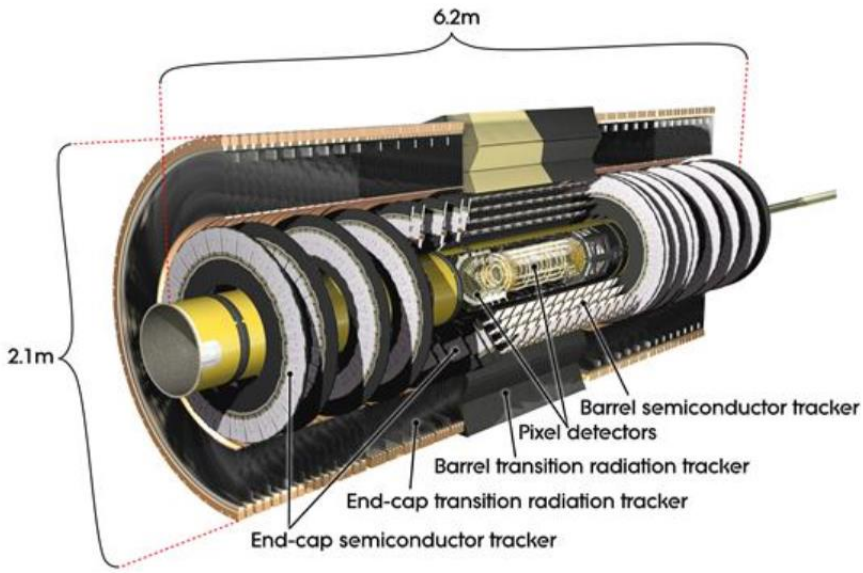


Figure 4: ATLAS Inner Tracker and its layer of systems [1]

The Pixel Detector is capable of effective operation with the projected particle densities and interaction rates close to the LHC collisions. The silicon chip present at the detector is referred to as Front-End (FE) electronics/chip [6]. The basic operation of this chip is to convert charge, which gets deposited on the pixel when particle passes through the pixel due to collisions, into a digital value. It also records the time over threshold value (ToT), i.e., the length of time the digital charge has been over a threshold value. These recorded values are gathered from the pixels/sensors at a regular interval, with a precision of a single bunch crossing. Data processing is performed on these values, after which it gets sent to the readout chain for further processing [5].

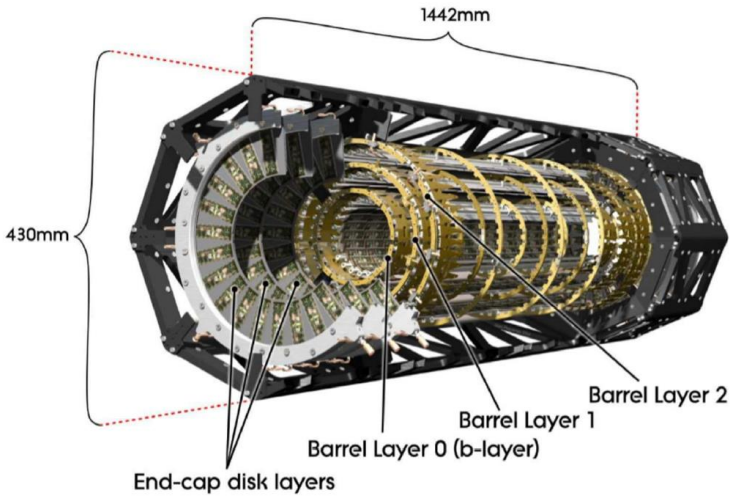


Figure 5: Inner structure of Pixel detector which is divided into 4 layers. The size of the detector is very small as compared to the whole ATLAS detector shown in figure 3. [1]

To maximize the number of particles that get captured by the sensors, the pixel detector has four different overlapping layers of FE chips. The next generation of Pixel chips that make up these layers are officially called ITkPix. A Research and Development (R&D) effort to design these chips has resulted in RD53 chips (FE electronics) and is what the emulator discussed in this paper is emulating.

1.3 UPCOMING HL-LHC/ITk PIXEL UPGRADE

Since its startup in 2010, the LHC has undergone two scheduled long shutdowns (LS), in 2013 and 2018 respectively. This paper focuses on the upcoming LS3 in the end of 2023, which aims at upgrading the central tracking system for the ATLAS experiment for operation at High Luminosity LHC (HL-LHC) starting in the middle of 2026 [5]. Luminosity refers to the amount of collisions that occur per bunch crossing, and therefore the particle flux through the detector.

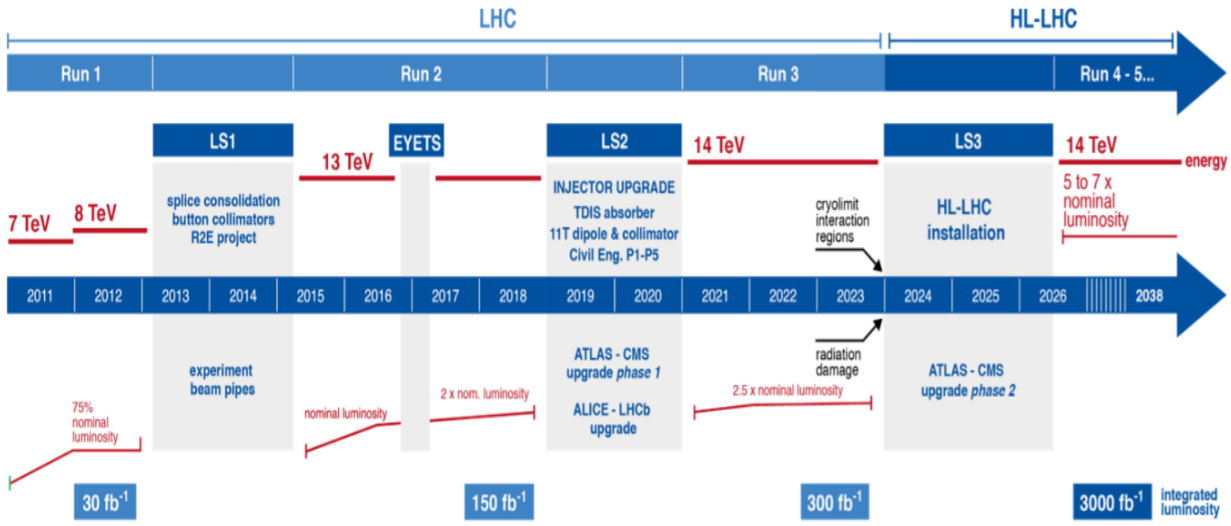


Figure 6: List of shutdowns planned after 2010; some have been completed and some are on their way. The figure also shows the peak and the integrated luminosity before and after those upgrades. [1]

The series of upgrades shown in figure (6) will ultimately result in an accumulated integrated luminosity for proton-proton collisions of 3000 fb^{-1} [5]. This represents an order of magnitude more data than what would be collected prior to the HL-LHC run. This amount of increase in the luminosity will make the current sensors and the readout systems inadequate [4]. As a result, all the electronics, trigger and detector components need to be replaced with the upgraded one.

The current inner tracker will be replaced with an all-silicon tracker, using state-of-the-art silicon technologies to cope up with the increase in occupancy, bandwidth and radiation damage that result from the increase of the instantaneous luminosity by a factor of 5 to 7.5 [16]. This in turn will impact the pixel detector layer as the current inner three layers will be replaced with just two: ITk strips and ITk pixel detectors. The outer three layers will stay in place as they are designed to operate well at the full HL-LHC data taking period.

With the ITk upgrade, future Pixel Detectors at the HL-LHC require a new generation of readout chip that can handle the severe requirements in terms of speed, noise, power consumption and radiation hardness. The next section with talk about the upgradation of FE pixel chips called RD53, their evolution since the first test chip has been development, and their on-going progress.

1.4 RD53

The upgrade of the pixel detector will lead to the replacement of the pixel readout system, which has led to the development of new FE pixel chips. The RD53 collaboration, i.e., Research and Development -53, is tasked to develop the next generation of pixel readout chip for both the ATLAS and the CMS pixel detector upgrades at the HL-LHC [8]. There are notable differences in the pixel detector layouts of the two experiments: CMS pixel detector is closer to the beam, and the ATLAS pixel detector is bigger. The final chip design will get installed at both ATLAS and CMS detectors [17]. As this paper is more focused on the ATLAS pixel detector's readout chip, all the technical numbers regarding the chip's architecture will be as per ATLAS requirements.

1.4.1 RD53A to RD53B

RD53A was designed as a technology demonstrator pixel chip to serve as a test platform for different designs, as well as to meet prototype (not production) specifications [7]. It was intended to demonstrate the suitability of the selected CMOS technology node – 65 nm for the HL-LHC upgrades of the ATLAS and CMS, including radiation tolerance, stable low threshold operation, and high hit and trigger rate capabilities [10]. It was never intended to be the final production as it contains multiple design variations for testing purposes. Since 2013, the RD53 collaboration has developed the RD53A readout chip, with which we can successfully exercise all the required features: bump bonding, test beams and irradiations with final geometry sensors, high hit rate operations, high speed I/O etc. [7].

The RD53A prototype allowed the chip to go through a comprehensive testing program, including x-ray, beta, gamma, and proton high and low dose rate irradiations, source and test beam measurements of the modules etc. This has helped in selecting one of the three alternate implementations of low power analog front-end variants, that are present in RD53A, as a common front end for both ATLAS and CMS chips [7][10].

Table 1: Difference in the physical parameters of RD53A and RD53B

Parameter	RD53A	RD53B
Pixel bump pitch	50µm x 50 µm	50µm x 50 µm
Pixel row (H)	192	384
Pixel column (W)	400	400
Core rows	24	48
Core columns	50	50
Chip width	20mm	20.054mm
Chip height	11.6mm	21.022mm

RD53A has a fixed array of 50x24 pixel cores, each containing an 8x8 array of pixels. While preserving the core organization, RD53B is designed to take the size of the pixel core as a parameter. This is done to support the CMS requirement of 54x42 pixel cores, and the ATLAS requirement of 50x48 pixel cores. In RD53B, any different behavior between ATLAS and CMS will be addressed with changing electrical configurations, not with design differences. Thus, the RD53B chip size produced for ATLAS will be capable of meeting the CMS functional requirements and vice-versa [7]. RD53B is an evolution of the RD53A integrated circuit to incorporate all production requirements defined by the experiments which we will discuss in the following section.

1.4.2 RD53B

The lessons learned from the RD53A chip have been incorporated into a follow-on prototype chip. RD53B is a pixel readout chip framework that can be instantiated into different sized physical chips [11]. The design framework is built upon the RD53A framework but with a number of technical improvements aimed at improving integration and verification of the final layout [8]. There are few main points to know before moving ahead:

- With the successful fabrication of RD53A in 65-nm CMOS technology at TSMC, RD53B will also get fabricated in the same technology node. A thorough testing has been done on the RD53A chip, which ensures that this 65-nm node can perform at expectations while being irradiated [7][12].

- RD53B consists of a pixel matrix and chip bottom. The pixel matrix is built up of identical 8x8 pixel cores stepped and repeated in columns and rows. The chip bottom contains both the analog and digital system functionality and should be viewed as a fixed element that does not depend on matrix size [11].

- The pixel matrix of the RD53B chip uses “analog islands in a digital sea” architecture [11][10]. Each pixel core is surrounded by a sea of digital logic which controls that set of pixel.

- The front end (FE) for ATLAS is based on the Differential FE of RD53A, with a few modifications [11].

- A lot of known defects of RD53A design has been addressed while designing RD53B, such as higher than desired timing variation of charge injection pulses from column to column [8].

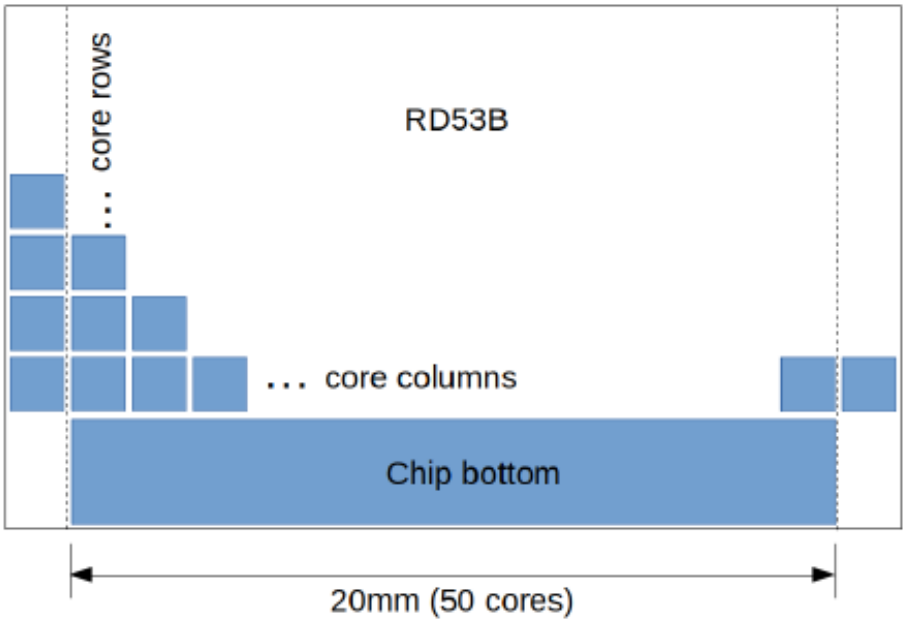


Figure 7: Conceptual depiction of RD53B framework, with a matrix composed of 50 or more columns by up to 50 rows of identical cores, and a fixed chip bottom. The dashed lines indicate the minimum width of 50 cores. Source [11]

There are a ton of changes made in RD53B from the RD53A design, and a lot of new features have been introduced. Some of the important differences were mentioned above; more about this can found in [8] and [11].

Chapter 2 of this thesis presents the development of the RD53B emulator at the University of Washington, along with its current state. This chapter will also introduce the new elements developed for RD53B and the corresponding upgrades made for the emulator. This includes both my own efforts, and those of the team around me. In chapter 3, the connections needed for communication between the RD53B emulator and the present readout systems are described. Chapter 4 is the conclusion of my thesis, and the future scope of the emulator project is discussed at length.

CHAPTER 2

RD53B EMULATOR

The RD53B emulator is a project developed and maintained by ACME lab at the University of Washington. This project aims at providing a platform that can be used to test various DAQ readout systems before the general availability of the RD53B physical chips. The design of the emulator is based on the specifications provided by the CERN collaboration [11].

It is a team project, started with the development of an emulator for the first prototype chip called RD53A several years ago [12][13][14][15]. The development of the emulator began much before the actual physical RD53A chip's specifications were fully finalized. This resulted not only in the development of emulator, but also a small DAQ core to communicate with the emulator that served as a proof of concept for the next-generation ITk DAQ systems [15]. To communicate between the emulator and the DAQ system, a serial communication protocol called Aurora is used [14]. Instead of using an inbuilt Xilinx IP block, Lev Kurilenko [14] developed a custom Aurora protocol core in anticipation of the RD53A chip coming back in 2018; this custom core is still present in the emulator system design.

Dustin Werran's thesis [12] talks about optimizations made to the emulator's architecture, such as removal of clock recovery block, reduction of the number of clocks used in the system, and addition of a unit called the Hit Data Generator. This hit data generator made the emulator useful even after the RD53A chip is present, and a whole section 2.2.3 (below) is focused on the development and upgrades made for the RD53B emulator design. Meanwhile, other students like Douglas Smith [13] worked on establishing the communication link between the RD53A emulator and the available DAQ systems from laboratories around the world. The successful communication between emulator and the DAQ systems provided the proof of correctness of our RD53A emulator's design. This chapter will discuss about the RD53B emulator's architecture, and new additions to the design of the front-end RD53B chip which led to modifications of the emulator.

It would not have been possible for me to complete this RD53B Emulator project without the collaborative research and efforts of many graduate and undergraduate students. Though most of the logical elements were kept intact from RD53A, introduction of new features and the upgrades with respect to the RD53B specification were done by:

- Architecture: Niharika Mittal, Geoff Jones
- Command Processor: Donavan Erickson
- Hit Data Generation and Data Encoding: Tony Faubert, Niharika Mittal
- Trigger Processing and Frame Formatting: Niharika Mittal, Geoff Jones
- Readout communication: Niharika Mittal, Donavan Erickson, Geoff Jones

2.1 MOTIVATION

The development of RD53 emulator has evolved over time at UW. From the beginning of the project, the primary purpose of the whole team was to provide researchers around the world with a platform they can use while the physical chip is under development. It is an effort to provide a RD53B look-alike to help test and develop various DAQ (Data Acquisition) readout systems before the real chip is done. These DAQ systems are responsible for successfully reading out the data from readout chip, i.e., RD53B. Having a stand-in for the real chip provides many advantages.

The upgrade of the Pixel Detector to meet the requirements of HL-LHC has also led to the redesigning of the entire readout chain installed at the detector site, not just the FE chips. Therefore, DAQ systems needs to be revamped to be able to keep up with the increase in data bandwidth in the HL-LHC, and to be able to communicate with the new front-end chips successfully. Several research laboratories are working on the development of new DAQ readout systems such as YARR, FELIX, and RCE [20][21]. Generally, these labs have to wait for the real chip to be designed, verified and fabricated in order to start testing with the whole system in place. But with the emulator, this period of waiting for the real chip can be eliminated. At the time of writing this paper, the RD53B emulator has been designed and verified, when the real RD53B chip is still under development. This will help the DAQ teams use our emulator for their system testing purposes. Moreover, the emulator's simple design supports this goal by allowing for quick changes to the emulator code to meet the user's goals or fix small issues.

Generation of realistic hit data is one of the new features that RD53B emulator provides. Realistic hit data means that the data format which the emulator sends out is consistent with the data format that the real RD53B chip will send when it is installed at the LHC at CERN, and provides generated particle hit data that is consistent with what is expected to be seen in practice. This will increase the usability of the emulator even once the real chips come out of fabrication, since not all research teams around the world have a mechanism in the lab to generate high energy particle collisions for testing purposes. This lack of realistic data limits the complete functionality check. The emulator solves this problem by producing realistic data on a lab setting. This can allow researchers to perform more complete testing and ensure that their system works on realistic data.

The RD53B emulator is designed to target multiple types of FPGA boards, including Xilinx Kintex KC705, FEB (SLAC) board, Xilinx Virtex VC709. Anyone can download the source code from the online repository while sitting in their lab and program it into existing FPGA boards and can then work with it. However, in the case of the real chip, they need to place an order and wait for the delivery which might take weeks or months. As the whole project is written in System Verilog, and is open source code, changes can be made while providing it to everyone in hours. In the case of a real chip, if an issue arises, the only solution is to wait for the next version, if any.

Last but not least, emulator is best suited for the testing and debugging of the DAQ systems that will interact with the RD53B chip in future. When problems arise with the real chip, it can be unclear whether the problem is with the DAQ, or with the chip itself. At that time, the emulator can be used as a replacement to the real chip to check whether the issue was in the physical chip or the readout systems.

2.2 ARCHITECTURE

The physical RD53B chip architecture contains both analog and digital components but, in the emulator, we are only emulating the digital part of it. The overall architectural diagram of the emulator is shown in figure 8.

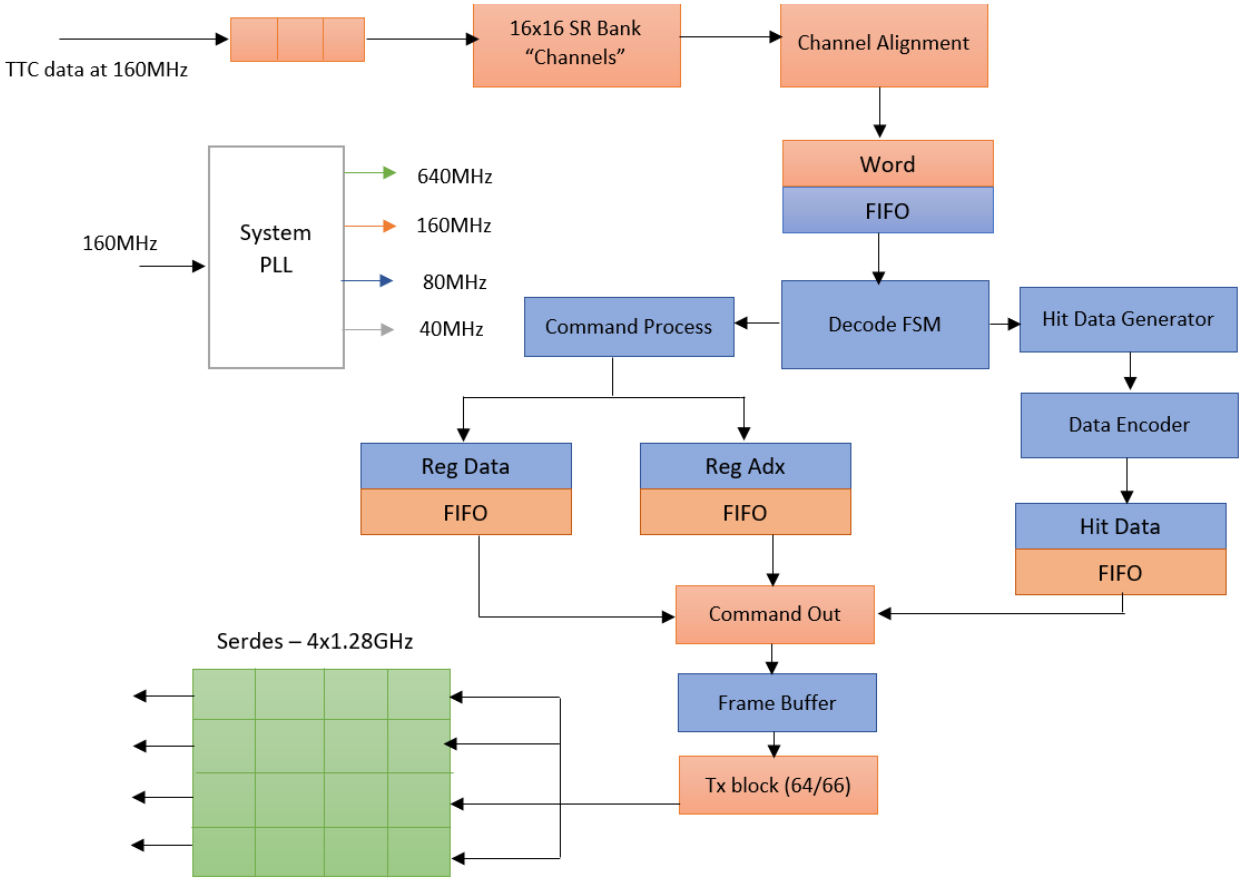


Figure 8: Overall architecture of the emulator [12]

A Phase Locked Loop (PLL) macro-block is used to generate these different clock domains (640MHz, 160MHz, 8MHz and 40MHz) from a single clock (160MHz) received by the system through an oscillator on the FPGA board. The four main components of the design are:

1. TTC data processing block (trigger and timing control block)
2. Command Processor
3. Hit Data Generator
4. Data Encoder (includes hit-map encoder).

Each are discussed in detail in the following sub sections.

2.2.1 TTC BLOCK

The emulator has a single input channel known as the TTC line (trigger and timing control line), which takes in a serial stream of data at 160MHz clock. The TTC line receives a continuous stream of bits which contains a synchronization pattern after every N words, where N is a variable defined by the readout systems from which the input is sent.

While designing the emulator, all the minute details of the chip's specifications are kept in mind. Although not much has changed in this unit compared to RD53A, in order to understand the rest of the architecture of the emulator one needs to understand the input data processing. The incoming data received is in the form of 16-bit frames which contains 2 8-bit symbols. Three different types of data can be received by the emulator. One is the trigger data, which tells the system to generate hit data. The second is the command data which performs some functions inside the system. The third is data which can be read from, or written to, the registers present in the emulator. Along with these, a unique sync frame (value = 0x817E) is also sent as an input to the emulator after approximately every 32 frames. The purpose of this sync frame is to ensure that the incoming data stream is locked with the RD53B, which aids in the correct decoding of the symbols received. Each frame and symbol are DC balanced, which means that there are equal number of 1's and 0's over a given interval. The symbols are separated by a Hamming distance of 2 in order to reduce the amount of error that can happen if a single bit is flipped. The symbols do not start nor end with three or more 1's or 0's in order to maintain a run length of 4 except for the sync frame which has a run length of 6.

In order to find the correct alignment of the incoming data, and thus determine which of a series of 16 bits is the "start" of a data word, the data is sent to a16x16 bank of shift registers, one at each possible alignment. Each bank is referred to as a channel. In order to get a channel locked with the transmitter, the TTC processor must receive a sync pattern (sent periodically), and a specific number of valid data words (set to 16 for now). Once the channel gets locked, the output data of that particular channel proceeds to the command processor block, where the data gets decoded and further processing on it happens as shown in figure 9.

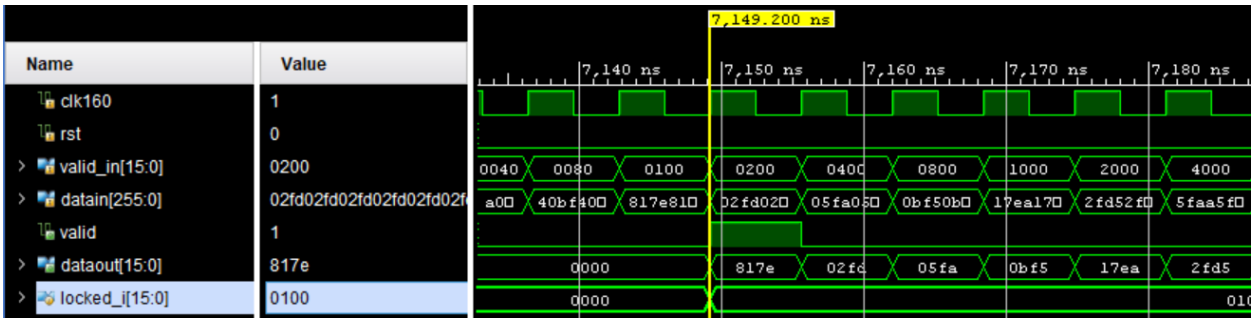


Figure 9: Simulation example of channel getting locked at 7.149 ns

2.2.2 COMMAND PROCESSOR

The Command Processor has been modified to support the changes in the RD53B specification manual. The majority of those were – adding new commands and changing the way commands get decoded [11].

The 16-bit data frame that gets received by the system from the TTC block needs to be decoded. Each data frame is encoded by the readout system (sender) and it is the responsibility of the emulator to decode it to understand what the command means. RD53B uses a decoding protocol which takes in 8 bits of symbol and outputs 5 bits of decoded data (table 2). This decoding protocol is not backward compatible with RD53A; therefore, we updated the emulator with the latest protocol.

Table 2: List of symbol decoding which is used to decode the incoming 8-bit symbol. Data_18 is the PLLlock command, which was not present in RD53A. [11]

Symbol Name	Encoding	Data Value	Symbol Name	Encoding	Data Value
Data_00	0110_1010	5'b00000	Data_16	1010_0110	5'b10000
Data_01	0110_1100	5'b00001	Data_17	1010_1001	5'b10001
Data_02	0111_0001	5'b00010	*Data_18*	*0101_1001*	5'b10010
Data_03	0111_0010	5'b00011	Data_19	1010_1100	5'b10011
Data_04	0111_0100	5'b00100	Data_20	1011_0001	5'b10100
Data_05	1000_1011	5'b00101	Data_21	1011_0010	5'b10101
Data_06	1000_1101	5'b00110	Data_22	1011_0100	5'b10110
Data_07	1000_1110	5'b00111	Data_23	1100_0011	5'b10111
Data_08	1001_0011	5'b01000	Data_24	1100_0101	5'b11000
Data_09	1001_0101	5'b01001	Data_25	1100_0110	5'b11001
Data_10	1001_0110	5'b01010	Data_26	1100_1001	5'b11010
Data_11	1001_1001	5'b01011	Data_27	1100_1010	5'b11011
Data_12	1001_1010	5'b01100	Data_28	1100_1100	5'b11100
Data_13	1001_1100	5'b01101	Data_29	1101_0001	5'b11101
Data_14	1010_0011	5'b01110	Data_30	1101_0010	5'b11110
Data_15	1010_0101	5'b01111	Data_31	1101_0100	5'b11111

In order to convert a 16-bit data frame into 8-bit data symbols, the incoming TTC data is stored in a FIFO. Three types of symbols can be received by the emulator: trigger, data, and command. The FIFO outputs the symbols to the above-mentioned 8-bit to 5-bit decoder, where it gets decided whether the incoming symbol is a trigger, data, or command. If it is a trigger symbol, or data associated with the trigger, then it is passed on to the hit generator block (discussed in the next section) or if it a command symbol it goes to the command processor (discussed in the current section).

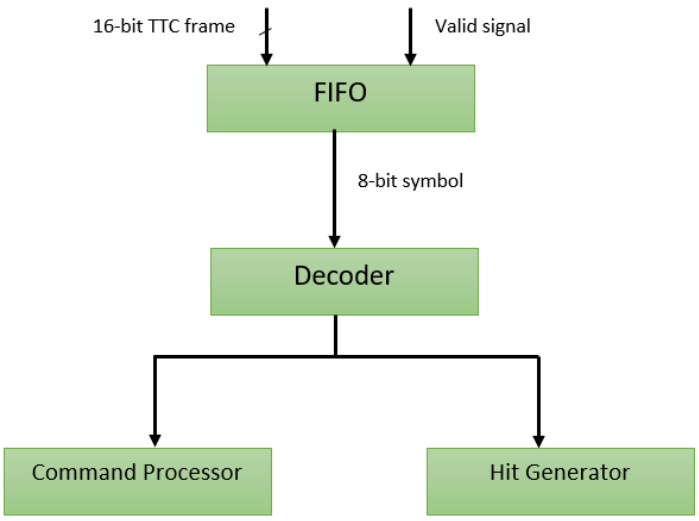


Figure 10: A microarchitectural depiction of the data flow before input is passed on to either the command processor or the hit generator.

A whole new updated command protocol has been implemented in RD53B. It consists of one sync frame, 7 non-trigger commands, 15 trigger symbols and 32 data symbols (shown in table 2). Below is a figure showing all the possible non-trigger commands.

Table 3: RD53B protocol commands and address or data field associated with each. [11]

Command	Encoding		(T)ag, (A)ddress or (D)ata 5-bit content					
Sync	1000_0001	0111_1110						
PLLlock	1010_1010	1010_1010						
Trigger	tttt_tttt	Tag[0..53]						
Read_trigger	0110_1001	ID<4:0>	00,T<7:5>	T<4:0>				
Clear	0101_1010	ID<4:0>						
Global Pulse	0101_1100	ID<4:0>						
Cal	0110_0011	ID<4:0>	D<19:15>	D<14:10>	D<9:5>	D<4:0>		
WrReg(0)	0110_0110	ID<4:0>	0,A<8:5>	A<4:0>	D<15:11>	D<10:6>	D<5:1>	D<0>,0000
WrReg(1)	0110_0110	ID<4:0>	1,xxxx	xxxxx	N×(D<9:5>	D<4:0>)		
RdReg	0110_0101	ID<4:0>	0,A<8:5>	A<4:0>				

All the executable non-trigger commands consist of a command symbol which helps to identify which of the 7 commands it is, and a data symbol which specifies the chip ID that the command is addressed to as shown in table 3 [11]. Chip ID is used when a single readout system is connected to multiple FE chips at once. Specifying the chip ID helps to distinguish which chip the readout system is trying to communicate with. The PLLlock command is considered as a no operation (noop) command, since its only purpose is to lock the Phase Locked Loop (PLL) to the correct frequency at the start of operation. This command does not have a chip ID as shown in figure 12, instead is repeated twice to produce a clock pattern [11]. The Clear command is used to clear the entire data path of the system. The Global Pulse command is used for a variety of purposes;

depending on where in the system this pulse is sent, it will perform different functions. Cal (Calibration Injection) is used for both analog and digital injection. WrReg(0) (Write Register, Single) command and WrReg(1) (Write Register, Multiple) commands both have two modes: single write and multiple write to register 0. The system distinguishes whether it is a single write, or a multiple write based on the first bit of the payload (0=single, 1=multiple). The only difference between the commands is that WrReg(0) has 9 bits of address and 16 bits of data whereas, WrReg(1) has no address and no data. The RdReg command (Read Register) is used to readout the addressed register.

The Write and Read commands were also present in the RD53A Emulator, but they used to only read or write to Register 0, which was not the correct implementation of those commands. The Emulator design has a set of global/configuration registers which should get updated with the write data value, just like the real RD53 chip. So, in the process of upgrading the emulator from one version to the other, we also added new functionalities to the system which were missing in the RD53A emulator. Earlier (in RD53A) the global registers were present in the system but were never updated while the system is running. We made the read and write commands to read or write to the global registers and update them appropriately. In RD53B emulator, readout systems can send the read command with a valid address and will receive the corresponding data back; likewise, the write command will update the data at a specific address. For example: a write to address 1 will change the value in the register one, which a subsequent read of address 1 will then return.

The Trigger command tells whether data should get sampled at a bunch crossing or not. A bunch crossing refers to the moments in the time when particle collisions occurs. The first 8 bits of this command provide the trigger pattern, and the next 8 bits gives the trigger tag value which will be returned with the data corresponding to that trigger. Encoding for the trigger pattern is kept the same as RD53A and is shown in the table 4 below.

Table 4: Trigger commands and symbols used to encode trigger patterns. Patterns are read left to right[11]

Symbol Name	Encoding	Trigger Pattern	Symbol Name	Encoding	Trigger Pattern
			Trigger_08	0011_1010	T000
Trigger_01	0010_1011	000T	Trigger_09	0011_1100	T00T
Trigger_02	0010_1101	00T0	Trigger_10	0100_1011	T0T0
Trigger_03	0010_1110	00TT	Trigger_11	0100_1101	T0TT
Trigger_04	0011_0011	0T00	Trigger_12	0100_1110	TT00
Trigger_05	0011_0101	0T0T	Trigger_13	0101_0011	TT0T
Trigger_06	0011_0110	0TT0	Trigger_14	0101_0101	TTT0
Trigger_07	0011_1001	0TTT	Trigger_15	0101_0110	TTTT

The Read_trigger command is used to read the trigger. This command does not have much effect to the functioning of the emulator system.

All of the commands are supported by the emulator except global pulse and cal. These commands are accepted by the system, but they do not perform any operation as they exclusively influence the analog circuitry of the chip, which is not the part of the emulator.

Given all the information about the types of commands and the decoding, the command processor module in the emulator is a huge finite state machine (FSM). Each state of the FSM machine is defined by each command present in the table 3. A state called neutral state acts as a transition state, because after the completion of any command, the FSM comes back to it. If you are in a state, there are only three ways you can come out of it. First, an error has occurred, such as the chip ID of the command does not match. Second, if the command has successfully completed. Third, if the system has encountered noop/sync commands. In the case of noop/sync, the system returns back to the previous command it was on once the noop/sync has finished its processing. Figure 11 below gives an outline of the command processor unit.

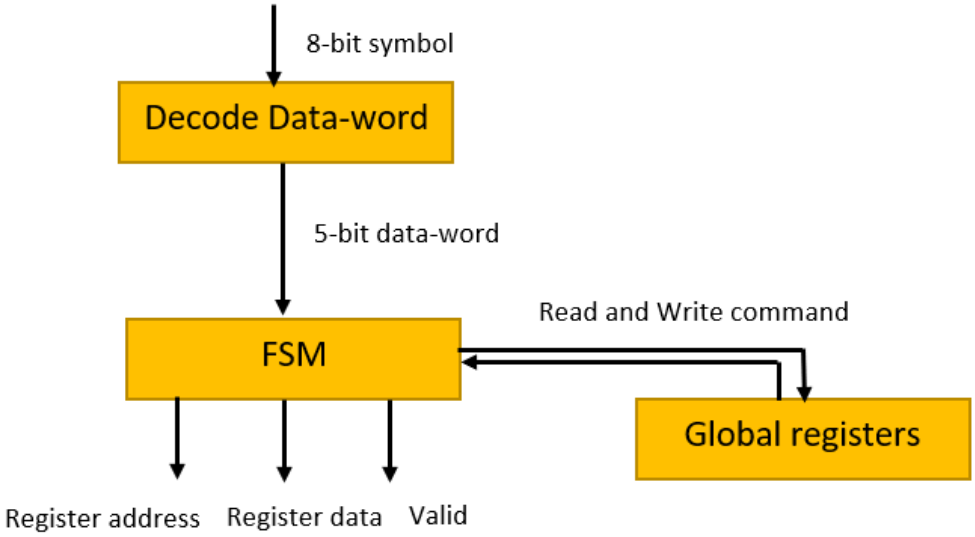


Figure 11: Overview of the command processor unit.

2.2.3 HIT GENERATOR

This unit in the emulator is useful not only before the real RD53B chips are available, but also once they have been deployed. The main reason behind this is that before the installation of the RD53B chip at the LHC, the entire systems needs to go through a series of tests, some of which require particle collisions at the same rate as HL-LHC. Because not every lab in the world can generate those collisions, the emulator can mimic the real data at the required data rate. The hit generator unit acts as a stand-alone unit which produces data similar to that one might see in the real detector setting. To understand the changes between the RD53A hit generator and RD53B one, we need to understand what the unit is mimicking.

In the LHC, events are known as bunch crossings, where particle collisions take place. Triggers are the signal which tells whether to record the specific bunch crossing data or not. Each bunch crossing occurs at an interval of 40MHz, while a 16-bit data frame is received by the system at 160MHz clock per bit; therefore, one 16-bit frame spans 4 LHC bunch crossings. Thus, a trigger command must specify a 4-bit map (encoded as per table 3) indicating which of the 4 bunch

crossings are actually triggered: hence 15 unique trigger patterns. This encoding of the triggers has stayed the same as in RD53A.

Similar to RD53A, RD53B also has an analog front-end comprised of a rectangular bed of analog sensors, but with a grid of 50 x 48-pixel cores. In the actual setting at LHC, particles collide with these sensors when a collision occurs, and the time for which each sensor experiences charge over its defined analog threshold is known as the Time over Threshold (ToT) value. This ToT value gets recorded within the RD53B chip. Each pixel is a set of 4 sensors, and each pixel gives a different ToT value; therefore, four sensors are dedicated to produce a single ToT value. These 153,600 pixels are arranged in an arrangement called cores. The addressing scheme for RD53B has been preserved from RD53A. The pixels are in a matrix of cores and regions, where cores were like postal codes and regions were like street address.

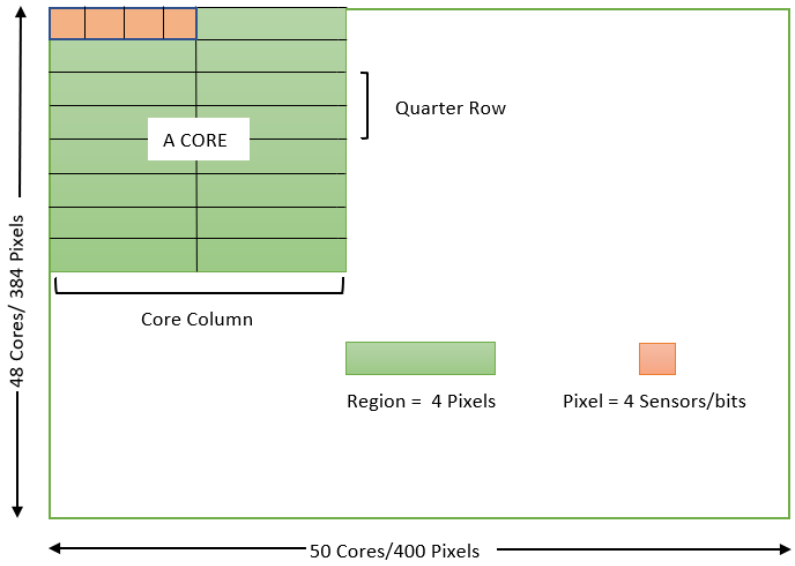


Figure 12: Data is emulated based on presented addressing scheme

Digitally emulating the behavior of each and every sensor is not practical. Thus, a model was created for the RD53A hit generator, using data collected from software simulation by different research laboratories. A set of shapes (combination of adjacent pixels) were identified that were repeated with higher probability, and the frequency of their occurrence in a single bunch crossing was observed. The data achieved from this study has stayed the same in the design of the RD53B emulator's Hit Generator. Figure 13 shows all the "tetris" shapes that can be the output of the hit data generator and their all possible orientations.

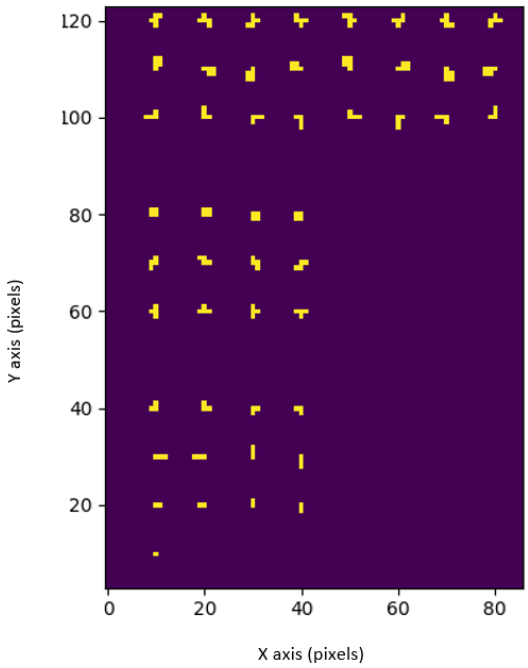


Figure 13: Different Tetris Shapes produced by RD53B emulator (all the possible orientations).

The RD53B Hit Generator works as follows: When the trigger and non-trigger commands get separated and go to their respective units, the trigger commands enter into a unit called HitMaker. This unit is responsible for managing the incoming trigger commands and outgoing hit data. The received trigger commands are passed through a trigger table, where the trigger tag and the trigger pattern are extracted (see table 4). The trigger tag refers to the 6-bit code received with each trigger command and can only be the values between 1 and 55. The tag value and the trigger pattern are stored in separate FIFOs. Using the trigger tag, an extended tag (8-bits) is generated, with 6 bits of tag and 2 bits indicating to which of the 4 bunch crossing within the trigger command the data corresponds. For example: command Trigger_03 in table 4 with tag = $(110100)_2$ will result in extended tag value = $(11010010)_2$ and $(11010011)_2$. The FIFO storing trigger patterns (which of the 4 bunch crossings should be actually triggered) is designed as a 4 to 1-bit FIFO, which acts as a trigger pulse to the hit generator. Once the trigger pulse goes high, the FSM present inside the hit generator changes its state from IDLE to WORK. While in the WORK state, continuous hit data is randomly generated, using a lot of circuitry including a Bloom filter, multiplexers, pseudo random number generators, cluster generators, line generators etc. [23]. In RD53B, before the generated hit data goes to the HitMaker, it passes through a unit called Data Encoder (explained in the next section). The data comes in to the HitMaker unit, where it gets stored in a FIFO. A signal called done tells the HitMaker when a specific trigger has been completed and the hit generator is ready to receive a new trigger.

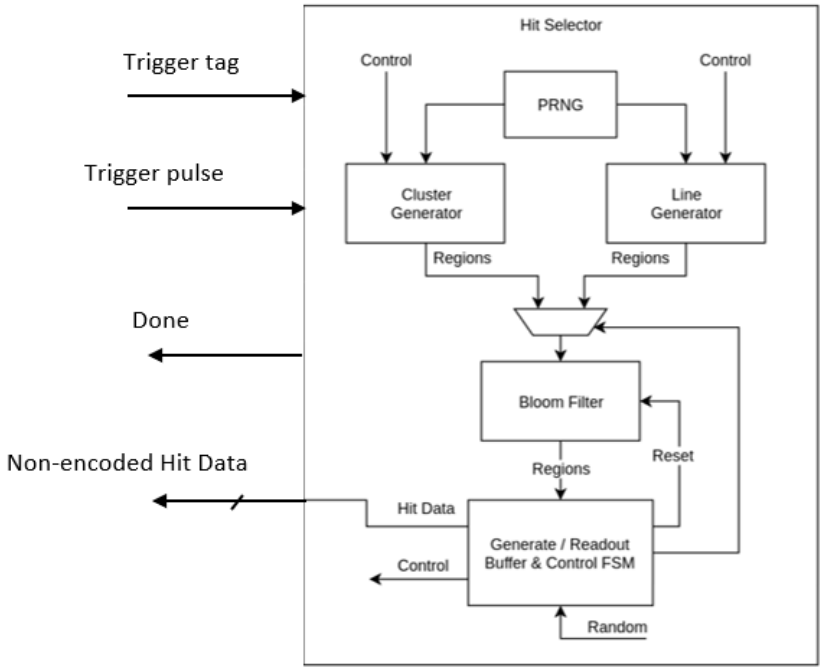


Figure 14: Elaborated architecture of Hit Generator Unit (Picture Credit: Tony)

Although the Hit Generator has changed significantly between RD53A and RD53B, the logic at the level of figure 14 is roughly the same. This is because the major hit generator revamping has happened in the logic where the data is generated for the specific addressed pixels. In RD53B, the amount of data that gets read out of the analog sensor bed increased by a factor of 4, which required a new compressed reading format. In RD53A, data of 4 pixels can be read. To tell the readout system the location of the "hit" pixels in the pixel bed, their core column address and core row address was sent along with the pixel data. The RD53A format is shown in figure 15:

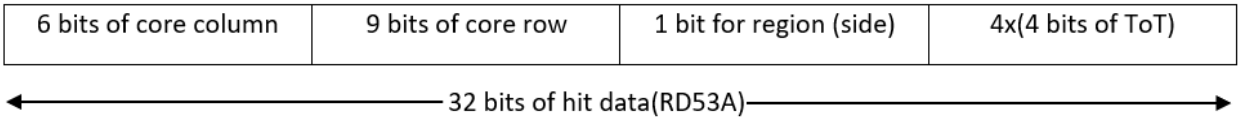


Figure 15: Hit Data Format coming out of the Hit Generator (RD53A)

In RD53B, this whole data format is replaced with respect to the new specifications, where there is no concept of core row. A new term called quarter row has been introduced which represents two rows simultaneously. The format of the output data is as follows (figure 16):

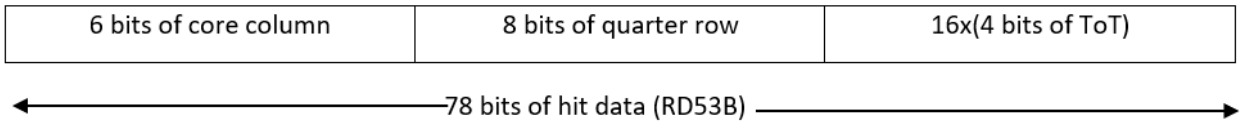


Figure 16: Hit Data Format coming out of the Hit Generator (RD53B)

Because of this difference in the size of the analog grid, and the amount of the data getting read out, the hit generator was revamped.

2.2.4 DATA ENCODER

Figure 16 implies that we will need to send 78 bits of information to specify the data of 16 pixels. To reduce this overhead RD53B introduces an on-chip encoding of the hit data. This gave birth to a new unit called “Data Encoder” in the RD53B Emulator. The block takes in the output of the hit generator and perform a series of bit packing (encoding) on that data before it gets passed to the Aurora block for AURORA processing, which is the same as the RD53A AURORA protocol. To replicate the new format of the real RD53B chip, the output hit data needs to be in the form of a continuous stream, instead of discrete frames. A stream is a self-contained, variable length data container beginning with a tag (8 bits), and then compressed hit data, or maybe other tags (internal tags) [11]. To understand the functioning of this unit, one needs to know about the new encoding protocol a little.

Hit data encoding uses a hierarchical address of core column (ccol), quarter-core row (qrow) within that column, and a 2 pixel x 8 pixel quarter-core hit map, in a compressed format [11]. The hit data (64 bits) is converted into a 16 bit hit map which represents whether the pixel has a hit or not (Figure 17 left). This 16 bit hit map gets encoded using a binary tree algorithm which takes place recursively in three steps. First, divide the quarter-core row in two halves vertically and label each of them with one if it contains any hits, and 0 if it does not. Second, repeat the first step for each half but dividing them horizontally this time. Third, read step 1 for each new section while again dividing them vertically. An example of how this encoding is performed is shown in the figure 17 below:

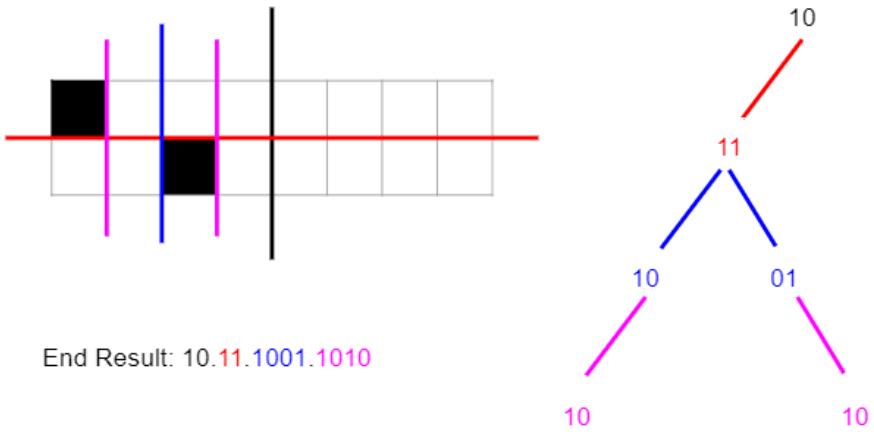


Figure 17: Depiction of binary tree for a quarter-core map.

The output of the binary tree algorithm goes through a bit replacement technique, since pattern ‘00’ never occurs in the tree (00 is an empty region, which would have been removed from the tree at a higher node). Therefore, all ‘01’ patterns get replaced with only ‘0’. This compression of the hit data produces a variable length Hit-map (5 to 30 bits). Following the hit map are the ToT values for all the hit pixels in the quarter-core, i.e., 4 to 64 bits [11]. The order of the ToT values is left to right, top row first. In a single quarter-core row, row number increases from top to bottom. The qrow address field begins with two flag bits called islast and isneighbor. The islast bit is set if this is the last qrow address in the ccol and 0 otherwise, while the isneighbor bit is set if the

previous address was qrow-1 and 0 otherwise. Similar to a form of Huffman Encoding, when isneighbor is set, the qrow address is omitted, as it is known to be the previous address + 1 [11]. Similarly, if islast is false, the next quarter-row will not have to include the ccol. All streams start with a New Stream bit and 8 bits of extended trigger tag.

This whole encoding and formation of streams is the reason our team had to develop a unit to perform this compression. The Data Encoder unit performs all the operations required to generate the stream, including sending the addresses, encoding hit data using the binary tree algorithm, keeping track of the trigger tag, knowing when a new event is starting, when to end a stream, when to end an event, etc. To be able to do all this, a lot of operations like shifting, or-ing and appending has to be done, as the formation of a stream depends on the previous, present and incoming addresses of the hit pixels, length of the encoded hit map and the number of ToT values that needs to be appended into the stream. For the simplicity of the emulator, features where the ToT values can be suppressed by a configuration option is not included in its design.

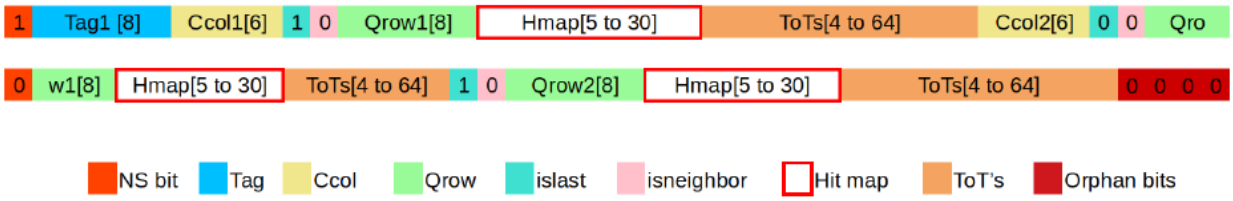


Figure 18: Encoded output data depicting one hit quarter-core in one core column and two adjacent hit quarter-cores in another core column spanning 2 64-bit block (2 AURORA blocks)

The data is encoded as per the following steps:

- Qrow encoding: The 78 bits generated by the hit generator (shown in figure 16) comes in as an input to the Data Encoder block.
- Hit Map Compressor (shown in figure 19 left) converts 64 bits (16x4 ToT values) of this incoming hit data to the corresponding compressed hit map of 5 to 30 bits (as depicted in the figure 17) along with generating its length variable.
- Simultaneously, this 64 bits hit data is also sent to the Hit Data Compressor unit (shown in figure 19 left). If the ToT value is not 0, the ToT length variable is incremented by 4 and this is done for all the 64 bits. Based on the resulting ToT length, the variable length (4 to 64 bits) ToT value is generated by shifting the ToT values to the most significant side and the rest of the bits are flagged 0.
- At the same time, the values of islast and isneighbor bits are set as defined previously in this section. The values of qrow address, ccol address, islast bit and isneighbor bit are merged depending upon how islast and isneighbor bits were set initially in the Address Compressor Unit (shown in figure 19 left).
- In the Data Aligner (shown in figure 19 left), a sequence of shifting and merging operations are performed on the compressed address, compressed hit map and the compressed ToT data. The first logical right shift is performed on the compressed hit map where the bits are shifted by the length of the compressed address. The second logical right shift is performed on the compressed ToT data where bits are shifted by the summation of the lengths of the

compressed address and compressed hit map. The resulting output of the data aligner unit, “qrow stream” (110-bits of variable-length data) is generated by performing an OR operation on the compressed address, shifted hit map and shifted ToT data. A flow chart depicting RD53B emulator encoding is shown in figure 19 (right).

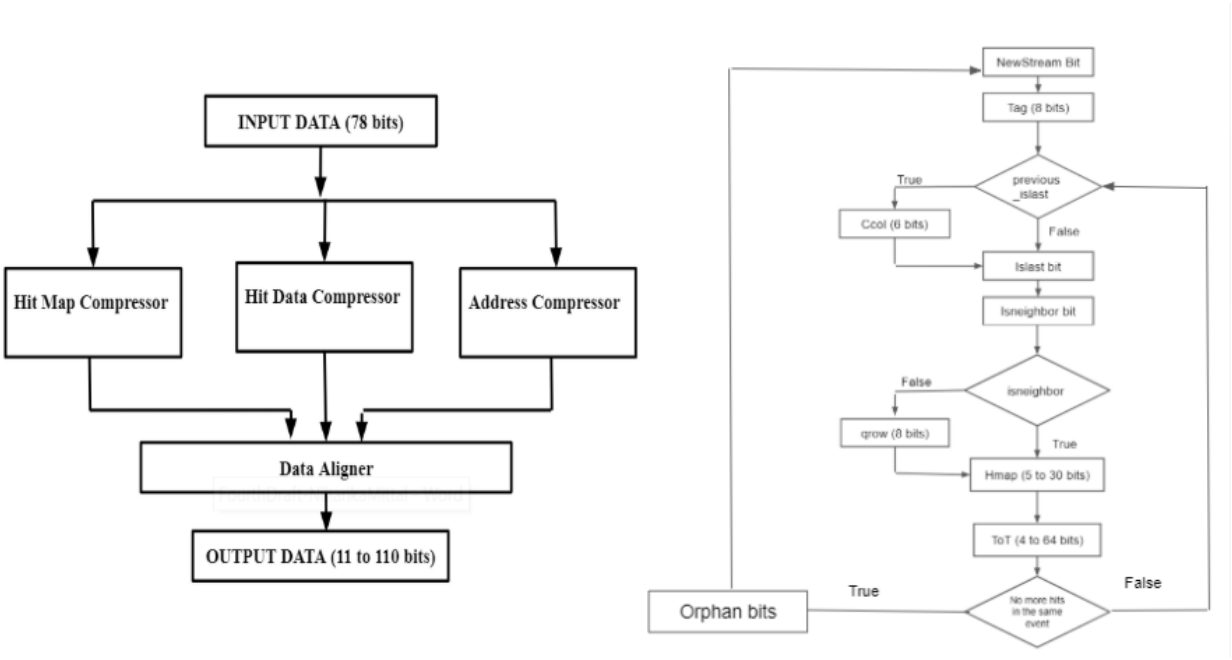


Figure 19: Left: Block diagram of the Data Encoder Unit. Right: Flow chart of RD53B emulator encoding

After the hit data is encoded, the final output stream of the RD53B emulator is created by performing the following steps:

- Block packing: We then create a “stream” (173 bits) whose left 64 bits are denoted as an aurora block and the remaining space is used to accommodate an up-to-110 bit qrow stream. We selected 173 bits because the worst case situation is when you are short of 1 bit in an aurora block and the next qrow stream that comes in is full of 110 bits (length of the coming in qrow stream is 110). Therefore, now we need 63+110 = 173 bits to hold the whole information.
- On the top of all these units, an input/output block is present which outputs an aurora block and requests for a qrow stream. When it requests a qrow stream, it takes in a qrow stream along with its length. Multiple checkers are present which checks whether the incoming qrow stream is the start of a new event or not, whether or not chip ID (in case of using multiple emulators) is added. In the reset state, it just requests for the new qrow stream (start of a new event).
- Then it continually checks if the stream length is at least the size of the aurora block, and outputs the aurora block and then shifts the stream left. If the stream length is less than the size of the aurora block, then instead it will request for a new qrow stream and add the qrow stream’s length to the stream length.

- A few levels up, a state machine is present which takes in the aurora blocks and finds the ones with the new stream bit (NS) as 1'b1, in order to add the correct tag bits in the right place, i.e., exactly after the NS bit.

All the above mentioned units have enables on them which lets them be paused. That way if the output FIFO fills up, the enable turns off, pausing all the units. There are a lot of optimizations details that make it more complicated than explained above.

2.2.5 DATA OUT

So far, I have described how the emulator can decode the incoming commands, and how it generates hit data. Next, I will describe how it sends out the generated output data to the readout system. The output readout data is formatted as shown in the figure below (figure 20). The hit data and command data are sent in a ratio of N:1, N blocks of hit data is followed by one command block. The command block only contains the output of the read register command. If there is a register data pending to be transmitted it will have the format depicted in the figure 20 below. Else the data will come from one of the auto read registers present in the global register bank. The value of aurora code (zz) decides how many auto and register reads will be present in the register block. In case of B4, both register fields will be of type auto-reads. If it holds value 55, then first read will be from auto-read and second from register command. Likewise, in case the value is 99, first read will be register command and second will come from auto-read. Last but not the least is the D2 value, it tells both the reads in the readout data is from register read commands.

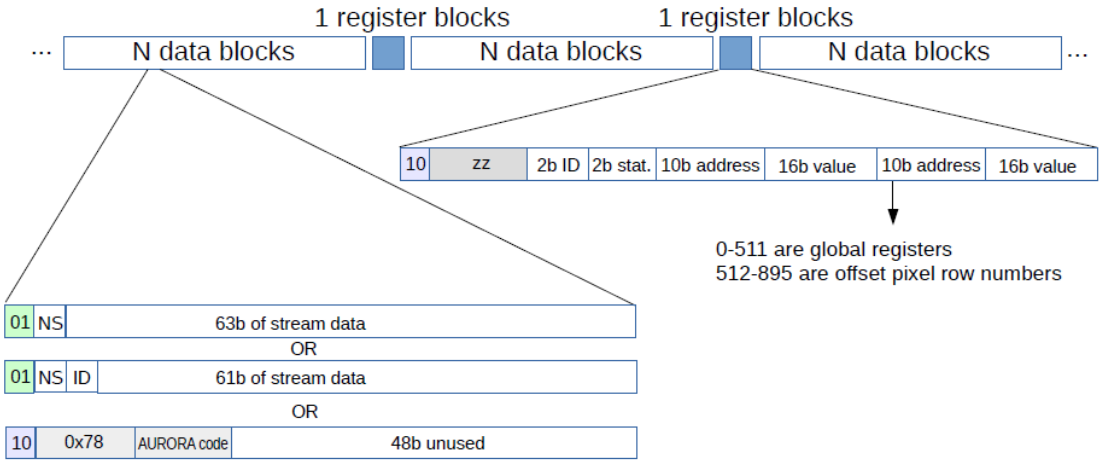


Figure 20: Readout data format of the hit data and the register reads [11]

Similar to the RD53B chip, the emulator also sends the data out over four separate communication lanes. Each lane in the LHC will be communicating at a speed of 1.28GHz, which leads to a combined speed of 5.12Gbps. Like the RD53B chip, the emulator can also communicate at speeds as slow as 160Mbps, though the speed is set before FPGA compilation. In the RD53A emulator register reads were hardcoded to transmit a fixed data and address value. In the process of upgrading the emulator, we now read out the correct addressed register.

The transmission protocol for RD53B is the same as the one developed for RD53A (AURORA protocol) but it has introduced a new feature called “strict alignment”. Strict alignment requires that all the lanes will send the same type of AURORA block at any given time. In a 66-bit aurora packet, the first two bits are the sync-header bits (either ‘01’ or ‘10’), and the remaining 64 bits are the data bits. A header value of 01 indicates that the 64-bit data associated with it represents hit data, while 10 is used for anything else.

Along with the RD53B emulator, we also developed testbenches for the system. Since existing readout systems are designed for RD53A and are not yet upgraded for RD53B, we tweaked the existing readout systems in order to make it communicate with RD53B emulator. This was done so that DAQ teams can utilize the emulator while upgrading their DAQ readout systems for RD53B chip. The technical details on how me made the modifications on the readout systems will be discussed in the next chapter.

CHAPTER 3

READOUT SYSTEMS

The term ‘Readout System’ has been mentioned multiple times in previous chapters and one must wonder what this has to do with the emulator functionality. During the Long Shutdown 3, the LHC is upgrading not only the inner layer of the pixel detector, but also the systems which are responsible for the selection and conveyance of the interesting physics data [19]. These data acquisition (DAQ) systems (readout systems) will need to be upgraded to work with the new devices, both within the LHC, and within initial test setups to run scans, calibrations, and characterizations of the RD53B chips [18].

So far there have been three readout systems that are getting developed for this purpose: YARR (Yet Another Rapid Readout) [20], RCE (Reconfigurable Cluster Element) and FELIX (Front End Link eXchange) [21]. One of the major uses of the emulator developed at UW is to help in the debugging of these systems while they are in the development phase. Therefore, the emulator’s compatibility with these systems is a top priority at the ACME lab. After the RD53A emulator was designed, we made sure that it communicates well with these readout systems [13].

At the time of writing this thesis, none of the above mentioned readout systems were upgraded to communicate with RD53B chip, which is still under development. Therefore, our efforts to make the RD53B emulator communicate with these readout systems were done with versions designed for the RD53A chip. The sub-sections below will talk about each readout system, and what changes were made in their program to make them communicate with RD53B emulator.

3.1 YARR

YARR is a readout system which is based on the concept of moving intelligence from the FPGA firmware into the host computer software [20]. It is developed and maintained by Lawrence Berkeley National Laboratory (LBNL) under the supervision of Timon Heim. The system has been able to provide an easy to understand and yet robust DAQ system for the previous generations of pixel readout chips, including FE-14 and RD53A, and is under development to provide the same support for RD53B.

The main purpose of this system is to send commands to the readout chips and receive data back from them. Three important components of YARR are its firmware, which is written in VHDL, software written in C++, and the PCIe bus connected to the host computer. The FPGA firmware is simple, as a major emphasis of YARR is doing the data processing in software instead of on an FPGA. The read data coming out of the readout chips are spread across four lanes (Figure 20). Data from the YARR firmware to the YARR software uses the PCIe bus [18]. YARR software plays a very crucial role in the proper working of the whole system as it is responsible for sending the relevant commands to the YARR firmware while simultaneously receiving read data from the PCIe bus to process it further.

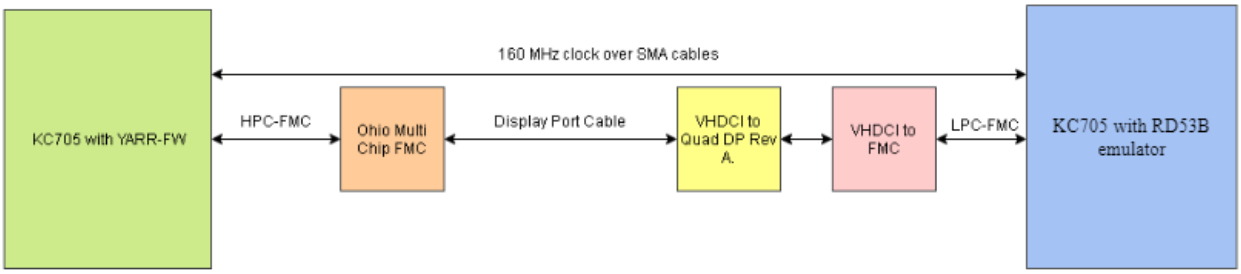


Figure 21: Schematic diagram for the YARR setup at UW

To communicate between the YARR system and the RD53B emulator, we followed the same hardware and software setup procedure as done with the RD53A emulator. A brief overview of the setup is discussed, and an elaborated schematic diagram is drawn in figure 21 where it shows all the I/O from the hardware that is required to establish a successful communication. For the YARR software setup, a custom designed PC by Timon is utilized, with a specific motherboard known to work well; some motherboard's PCIe implementations have caused issues. For the YARR firmware, a Xilinx KC705 board was used. The KC705 board is programmed with the YARR firmware code. There were two reason for using KC705 board for the communication setup: first, all the changes required for the communication were already made for the RD53A emulator in the YARR constraints; second, multiple KC705 boards are present in our ACME lab.

Another KC705 Xilinx board is used to host the RD53B emulator. The board hosting the YARR firmware was connected to the Ohio MMC (Multi-Media Card) card [20], which gets connected to the FMC (FPGA Mezzanine Card) chip on the board hosting RD53B emulator through display port cables (DP). A photo of the whole setup at UW is shown in figure 22.

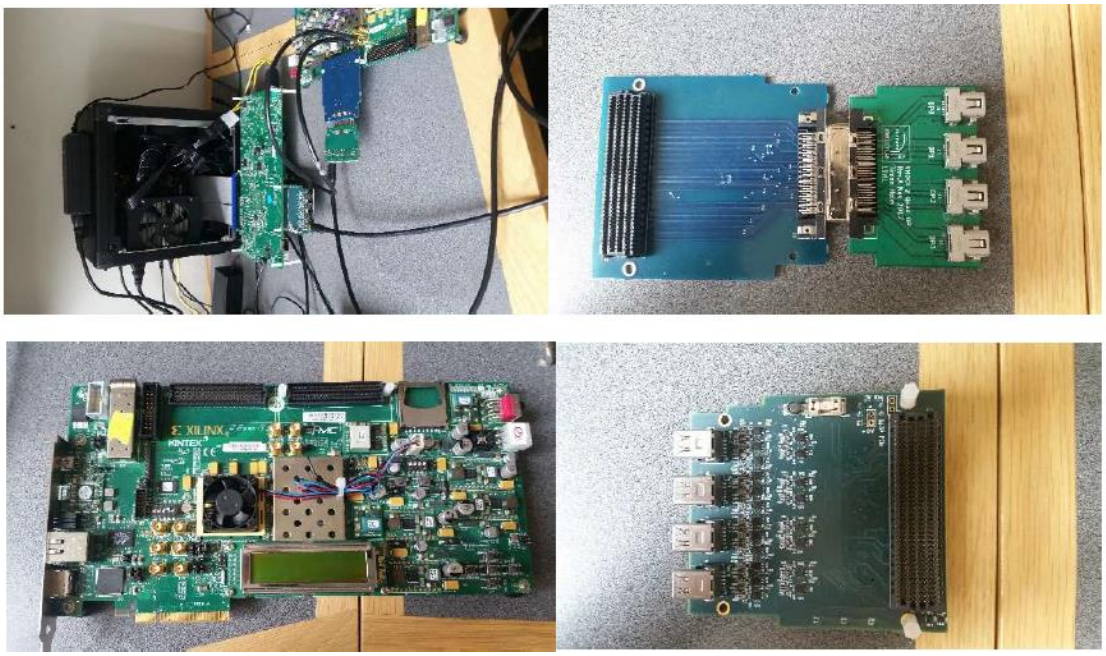


Figure 22: Top left: YARR setup at UW. Top right: FMC to DP breakout board. Bottom left: KC705 Xilinx board. Bottom right: Ohio MMC card [13]

3.1.1 YARR MODIFICATION FOR RD53B EMULATOR

As mentioned before, an updated version of the YARR system was not available at the time we were setting up the communication link between the RD53B emulator and YARR. A lot of tweaks and edits were done by our team to make the YARR for RD53A communicate with the RD53B emulator. The RD53A and RD53B chips are very similar. The primary differences are in terms of:

- the kind of data they send/receive
- the amount of processing required on the received data

If YARR is used with the old version (RD53A), we found that it was not sending the commands which the RD53B emulator was looking for. To fix this, we found that there is a software file which is responsible for sending the commands and receiving the read data back via the YARR firmware. Therefore, we created a new software file from scratch to make YARR RD53B compatible. After running it successfully, we saw the commands appear on the RD53B emulator side (ILA). Once this was done, it was time to see whether the emulator is generating the corresponding data for the received commands.

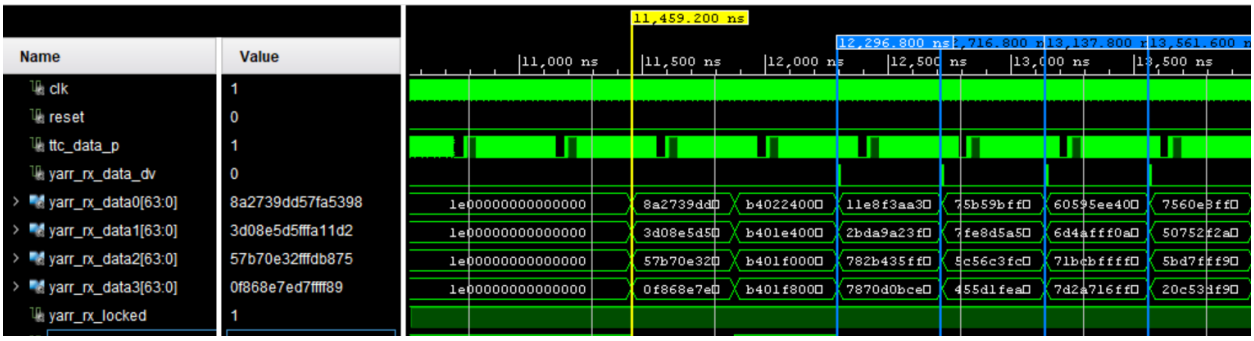


Figure 23: Encoded hit data produced by the RD53B emulator when the trigger command is received. Strict alignment required all the four lanes to have hit data at the same time

A multi-level demultiplexer is present in the YARR firmware, which steers the data based on its type, whether it is hit data, read register data, or auto read register data. First the data is separated based on the incoming aurora 2 bit sync-header: if it is a '01' then it is a hit data, else if it is a '10' then it is a user data type. If the incoming data is hit data, it gets stored in a FIFO and then gets sent to the YARR software for further processing. If it is any other user data type, the incoming data goes through another layer of demultiplexer, where the routing of data is based on the aurora code shown in figure 20 and discussed in section 2.2.5. The entire demultiplexer was rewritten to make it compatible with RD53B specifications.

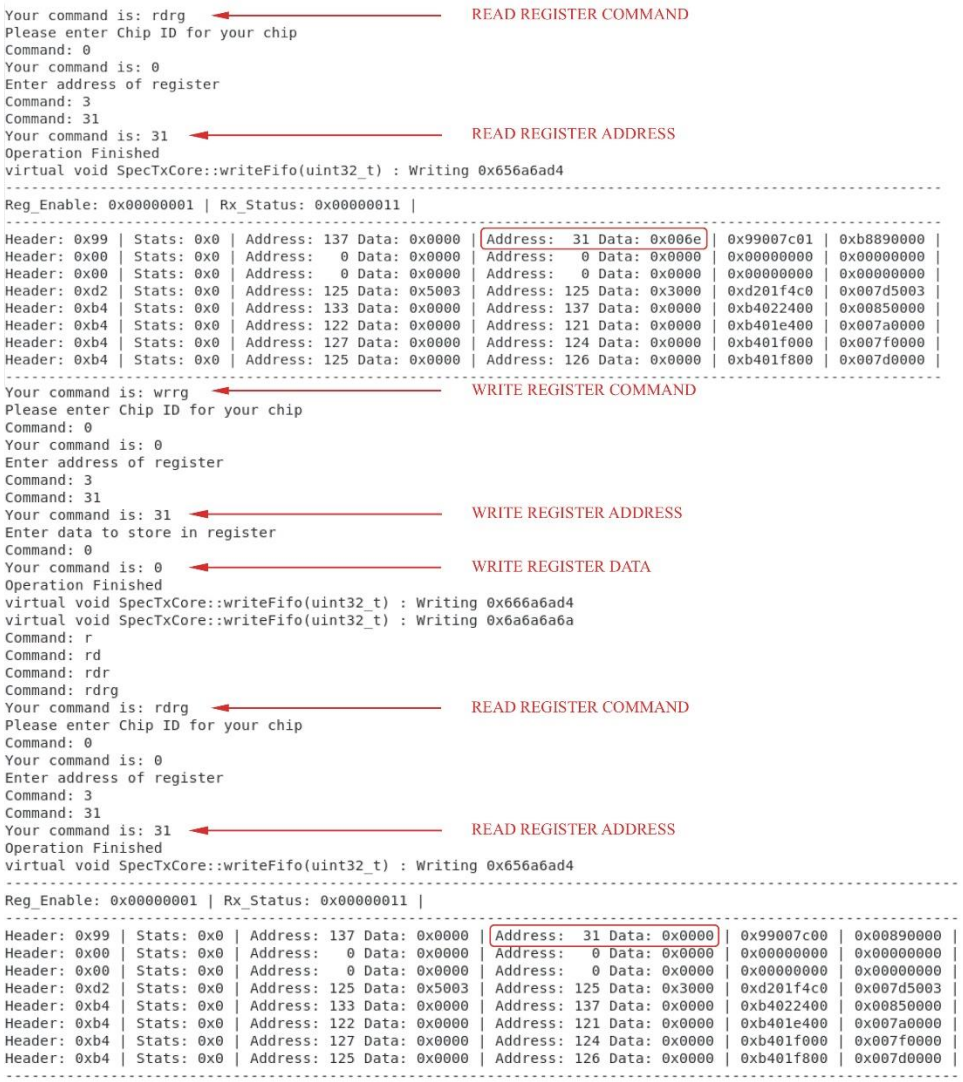


Figure 24: A series of read and write register commands were sent on the same register address to show that the data present at the address location gets changed by the write command, and the next read register command on the same address reads the new data value

The last thing which needed adjustment was the order in which YARR software was receiving the hit data. The hit data was getting received by the YARR software but in a scrambled lane order. This happened because the arbiter present in the YARR firmware was designed to pass data from any lane once data is made available to it. This needed significant adjustment, since RD53B sends encoded data which needs to be in the correct order for it to get correctly decoded back to the pixel hits. For that, we designed our own arbiter which waits for all the lanes to have the same type of data (based on strict alignment, which is introduced in RD53B itself) and then starts reading from them in the increasing order, 0 to 3.

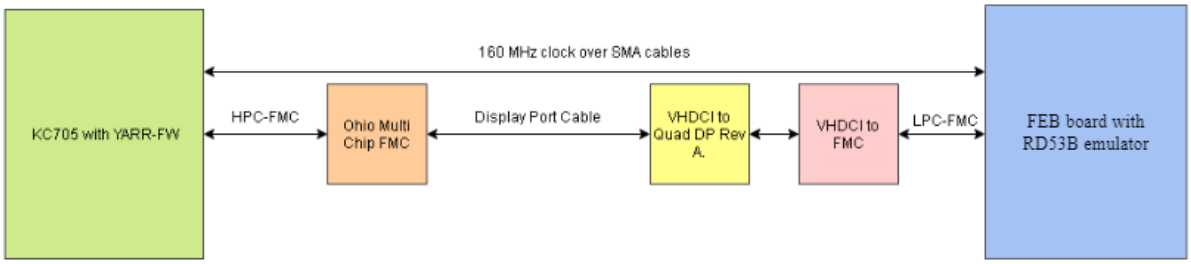
All these modifications to YARR system was made to provide a valid proof for the correct functioning of the RD53B emulator. The work at LBNL is going on to upgrade the YARR readout system to communicate with RD53B chip.

3.2 RCE

RCE is a DAQ system designed and managed by the RCE development lab at the SLAC National Accelerator Laboratory. It is a general-purpose cluster data acquisition system based on the concept of system-on-chip design. A major goal of this effort is to be able to explore new DAQ architectures for the upcoming ATLAS upgrade. The unique approach of RCE has led to designing their own custom FPGA boards in order to successfully communicate between the readout chips and the RCE system. The RCE system is so versatile that it supports multiple FPGA boards and different speeds of communication all by changing a few settings. RCE is also comprised of two main components knows as firmware and software. Most of its software are imported from YARR. The RCE firmware also adopts a lot of functionality and features from the YARR firmware.

To communicate between the RCE system and RD53B chips, SLAC has designed a custom FPGA board called FEB (front end board) for the front-end readouts; we use this same board to host the emulator firmware. To host the RCE firmware, SLAC has done a great job at supporting different types of FPGAs. This includes a Xilinx zynq board called ZCU102 (combination of an FPGA and an ARM microprocessor), which can be used as a standalone DAQ system hosting both firmware and software. Multiple hardware connection options are provided if someone wants to use Xilinx Ultra-scale Kintex 8 (KCU105) board to host RCE firmware. In order to make connections between a KCU105 and a FEB board hosting our RD53B emulator, one can either use optical fiber to connect with the HSIO board hosting the RCE software, use a PCIe connector to connect with the HSIO board hosting the RCE software, or an optical fiber can be used to connect with the host PC with a 10GB ethernet card using an ethernet cable having optical fiber I/O's.

Since most of the above mentioned hardware configurations are not present at UW, we decided to have the emulator programmed on the FEB board and keep the rest of the hardware setup similar to the YARR setup as shown in the figure 25 below. This way we were able to have the RCE hardware chain and do all the communication testing which is required on the emulator side.



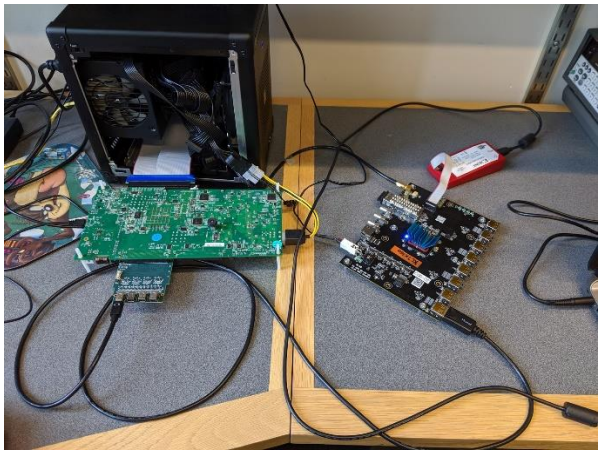


Figure 25: Top: Schematic diagram of the RCE setup at UW. Bottom: RCE hardware setup at UW

3.2.1 RCE MODIFICATION FOR RD53B EMULATOR

Since most of the RCE firmware and software is adapted from the YARR system, all the changes and additions we introduced to the YARR system were also used for the RCE system. The only difference was the clock speed which the RCE firmware sends out to the FPGA board hosting emulator firmware (160MHz) instead of 250MHz generated by the YARR firmware. This difference was already taken care of by our team while communicating between the RCE system and the RD53A emulator. Once the communication was achieved with the YARR and RD53B emulator, it was easy for us get the RCE setup working as well. Some of the results can be seen in the figure 26.

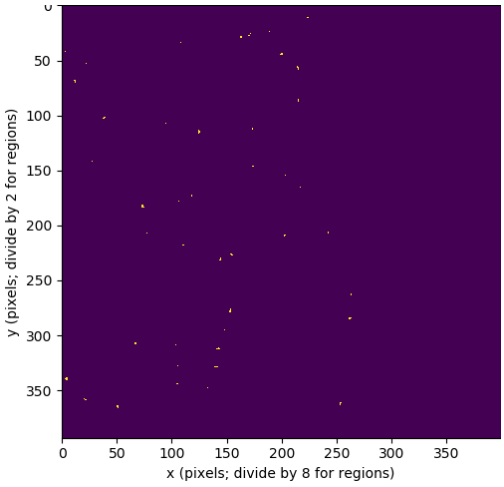


Figure 26: Displaying the decoded hit data acquired on the RCE side of the setup. The same hit data (encoded) is shown in figure 23

3.3 FELIX

The last readout system that will be discussed in this thesis is the FELIX DAQ system, which is developed and maintained by Argonne National Laboratory, Illinois. The FELIX system is designed to handle the significantly increased data volumes in newer pixel detectors [21]. It is expected to be installed alongside existing readout setups at the ATLAS site between 2021 – 2023. The purpose of the FELIX readout is to receive and identify different information streams on its incoming optical links, and route packets to client processing applications via a commercial switched network [21]. It also receives packets sent from the network and forwards them to the specified on-detector modules [21]. Its generic architecture enables it to do data processing in software; therefore, FELIX acts as a general purpose readout system and is used outside of the ATLAS project as well.

Similar to both the readout systems mentioned previously, FELIX is also split into firmware and software. The FELIX firmware is capable of supporting two modes of operation, such as GBT and FULL mode. The GBT (Gigabit Transceiver) protocol is designed for ATLAS, as it provides bi-directional communication, which is necessary as per the LHC post upgrade requirements. The FULL mode (FPGA-to-FPGA protocol) is for non-ATLAS projects where high speed (~9.6Gbps), single direction communication is needed. The FELIX software is designed with a multilayered approach. Part of it is dedicated to configuring the firmware deployed on the FPGA boards using FELIX Application Programming Interface (API). The main software is called FELIX core application and is used during data taking [21].

To support the FELIX setup requires a Supermicro X10SRA-F motherboard with 32 GB of DDR4 Ram and an Intel® Xeon™ E5 family CPU. For the RD53A emulator we used the same PC that was used for YARR and RCE. Since then we have purchased a whole new setup for FELIX hardware and software, using the exact technologies specified for it. To host the FELIX firmware, either Xilinx VC709 or BNL-12 boards can be used. Since our lab has multiple VC709, we went with that. Unlike other readout systems, FELIX uses optical cables instead of display port. One of the main reasons is to achieve higher transfer speeds. To the VC709 board, a TTCfx mezzanine card is attached which helps to loop a 160MHz clock back into the system through the SMA ports on the VC709 (visible in the figure below). The FELIX setup utilizes another board called VLDB which is connected to the VC709 hosting FELIX firmware through SFP cables. The main purpose of this board is to connect the FELIX to multiple chips at the same time via mini-HDMI cables. For emulator it sends the data over to another VC709 hosting the RD53B emulator via an HDMI cable. To connect the VLDB board and the VC709 hosting the emulator, a cable chain converting mini-HDMI to mini-display port (goes in an FMC card on the VC709) is used (figure 27). The VC709s are connected using optical cables.

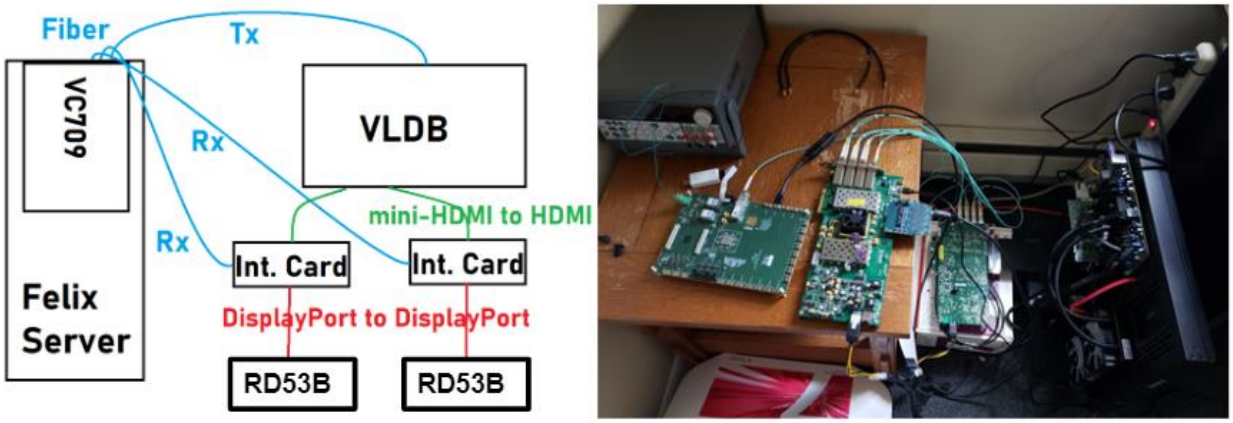


Figure 27: Left: Schematic diagram of the FELIX setup [22]. Right: FELIX hardware setup at UW

The FELIX DAQ has not yet been updated to provide communications with the RD53B chip. An effort at UW is ongoing to establish communications between FELIX and RD53A emulator. Once this is completed, we will move on to supporting RD53B and our new emulator. At the time of writing this thesis, the hardware setup has been done, while work is still ongoing to develop the communication link between the FELIX readout system and our emulators.

CHAPTER 4

CONCLUSION AND FUTURE WORK

4.1 CONCLUSION

Increasing the luminosity at the LHC has led to the revamping of major units of the detectors present across the LHC ring. This resulted in the replacement of the pixel detector, the inner-most layer of the electronics at ATLAS site. A lot of research and development is going into designing the perfect front-end chip for the HL-LHC. This led to the beginning of this project and is what this thesis is based on.

This thesis in general talked about the development of the RD53B chip's emulator, the changes and the additions to its architecture our team did in order to move from RD53A to RD53B emulator design. Our main aim had always been to stay as close to the real chip architecture as possible, while developing an FPGA based design.

As a team project, change is a constant. While designing the emulator, apart from upgrades of it to meet the new specifications of the physical chip, we also added new functionalities to the system which were missing in the RD53A emulator. Hit generator for the RD53B emulator, and a corresponding hit map encoder were written both in hardware (System Verilog) and software (Python). As the project moved on, the need for multiple hit generators came into picture to meet the fill data-rate.

The communication between RD53B emulator and the YARR readout system was created, which led to fixing bugs in the emulator code and making changes to the YARR firmware code to make it compatible with RD53B. This early availability of the emulator to the researchers is a success of our team as now they can use our emulator to test their upgraded DAQ system designs. The data coming out of the emulator was correctly formatted which the readout system will expect to see.

4.2 FUTURE WORK

At the time of writing this thesis, the RD53B emulator is designed and is ready to be used by the DAQ teams. Currently, work is going on in order to make the emulator run at a speed of 1.28G bits per lane, which means that the I/O should be able to run at 5.12Gbps total across 4 lanes, whereas the current design works at 640Mbps.

The emulator will see its successor, known as RD53C, in the future. The emulator effort should shift its focus to the development of the RD53C emulator, while maintaining this current and previous versions.

ACKNOWLEDGEMENT

I am grateful to many remarkable people who have educated, supported, and guided me throughout my career at the University of Washington.

First and foremost, I would like to thank my advisors Scott Hauck and Shih-Chieh Hsu for their incredible encouragement and for providing valuable insights into the direction of this project. I remember I used to walk into Scott's office to seek his guidance and I have benefited enormously from those discussions. Shih Chieh has played a central role in giving me a positive motivation towards the completion of this project. His valuable feedback has always pushed me forward. They both have been a great source of inspiration to me and I feel an immense amount of gratitude for both of them. I would also like to thank US Department of Energy for supporting this project.

Special thanks go to Timon Heim for guiding me through my summer internship at LBNL where he was my supervisor, and I benefited a lot from his immense knowledge on the RD53 project throughout my work at UW. In addition, being surrounded by the other talented and creative people at the Berkley lab every day was a privilege.

I'd like to thank my wonderful hard-working teammates: Douglas Smith, Geoff Jones, Donavan Erickson and Tony Faubert at UW without whom the project in this thesis would have never gotten done. Many thanks to Douglas whose guidance in my early days helped me get started on this project. Geoff has taught me so much about writing quality code and programming practices in general and his great efforts to teach me VHDL. I feel incredibly lucky to have worked with you all.

Finally, I would like to thank my family - my mother Anjali Mittal, my father Nischal Mittal, and my brother Shubham Mittal. Their emotional support is the primary reason for my success in completing this thesis and obtaining a graduate degree. I would not be able to survive through the rainy days of Seattle without their encouragement.

REFERENCES

[1] https://home.cern/ “CERN website”, CERN, [Online]

[2] https://atlas.cern/discover/detector “CERN atlas website”, CERN, [Online]

[3] https://atlas.cern/discover/detector “CERN atlas detector webpage”, CERN, [Online]

[4] https://atlas.cern/updates/atlas-news/preparing-ATLAS-for-future “HL-LHC upgrade in 2024”, CERN, [Online]

[5] Technical Design Report for the ATLAS Inner Tracker Strip Detector, CERN, Geneva

[6] “The ATLAS Pixel Detector”, M. Garcia Sciveres, Lawrence Berkeley National Laboratory

[7] “RD53 Collaboration Proposal: Extension of RD53”, Chistiansen, Jorgen; Loddo, Flavio [CERN-LHCC-2018-028]; [LHCC-SR-008]

[8] “RD Collaboration Proposal: Development of pixel readout integrated circuits for extreme rate and radiation”, Chistiansen, Jorgen (CERN); Garcia-Sciveres, M (LBNL) [CERN-LHCC-2013-008]; [LHCC-P-006]

[9] “The RD53A integrated circuits”, Garcia-Sciveres, Maurice (Lawrence Berkeley National Laboratory (US)), [CERN-RD53-PUB-17-001]

[10] The ATLAS Collaboration, “The RD53A integrated circuit”, Memo. CERN-RD53-PUB-17-001, January 30, 2017

[11] “The RD53B Pixel Readout Chip Manual”, Version 0.38, April 14, 2020

[12] “Development of an FPGA Emulator for the RD53A Test Chip”, D. Werran, S. C. Hsu, S. Hauck, March 22, 2019

[13] “FPGA Development of an Emulator of the RD53A Prototype Chip and its Integrated with Carious Readout Systems”, D. G. Smith, S. C. Hsu, S. Hauck, June 12, 2019

[14] “FPGA Development of an Emulator Framework and a High Speed I/O Core for the ITk Pixel Upgrade”, Lev S. Kurilenko, S. C. Hsu, S. Hauck, 2018

[15] “Three Generations of FPGA DAQ Development for the ATLAS Pixel Detector”, Joseph A. Mayer II, S. C. Hsu, S. Hauck, 2016

[16] “ATLAS ITk Pixel Detector Overview”, Andreazza, Attilio (ATLAS Collaboration), ATL-ITk-SLIDE-2018-1053

[17] “RD53: Making the challenging pixel detector chips for the Phase-II upgrades of ATLAS and CMS”, Jorgen Christiansen

[18] “Upgrade of the YARR DAQ System for the ATLAS Phase-II Pixel Detector Readout Chip”, Nikola Lazar Whallon, Timon Heim, Maurice Garcia-Sciveres, Arnaud Sautaux, Hideyuki Oide, Karolos Potamianos, Shih-Chieh Hsu, Proceedings of Science - 2017

[19] “Evolution of the Readout System of the ATLAS experiment”, A. Borga, G.J. Crone, B. Green, A. Kugel, M. Joos, J.G. Panduro Vazquez, J. Schumacher, P. Teixeira-Dias, L. Tremblet, W. Vandelli, J.C. Vermeulen, P. Werner, F.J. Wickens, Proceedings of Science - 2014

[20] “https://yarr.readthedocs.io/en/latest/”, YARR documentation [Online]

[21] “FELIX: The New Detector Readout System for the ATLAS Experiment”, G. Unel, Proceedings of Science, Oct’2018

[22] “RD53A readout with FELIX and YARR system setup”, B. Abbott, J. Lambert, F. Metcalfe, A. Paramonov, M. Trovato, ATLAS Note, January 2020

[23] “Hit generator Documentation”, Hit Generator Documentation by Tony Faubert, UW

APPENDICES

APPENDIX A: LIST OF TABLES

APPENDIX B: LIST OF FIGURES