Development of an FPGA Emulator for the RD53A Test Chip

Dustin Werran

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

Master of Science in Electrical Engineering

University of Washington
2019

Committee:
Scott Hauck
Shih-Chieh Hsu

Program Authorized to Offer Degree:
Department of Electrical and Computer Engineering

Abstract

In 2024 the LHC will be shut down for an extended period to perform upgrades to the instrument and the detectors located on it. One such upgrade is the Front End ITk upgrade in ATLAS Pixel. New hardware is being developed for this upgrade, and a test chip called the RD53A has been created as a prototype version of that future hardware. Primarily because of restrictions associated with the RD53A chip, including its limited availability to researchers, its inability to generate realistic data without radiation present, and the speed at which changes to the physical chip can be made when issues are found, The Adaptive Computing Machines and Emulators (ACME) lab at the University of Washington built and maintains code to emulate the RD53A. The Emulator solves the problems posed by the physical chip: The code is downloadable from an online repository and runs on a commercially available FPGA and thereby is easily accessible to researchers; It generates realistic hit data without need for irradiation; and, when the Emulator needs to be altered to fix issues or target certain research applications, turnaround can be completed in hours, not weeks. The motivation for the Emulator, its current state of development, and my personal work on the architecture and logic are explained in this thesis.

Contents

Chapter 1: Introduction

CERN

The European Organization for Nuclear Research (CERN) is an international scientific collaboration and research center focusing on high energy physics. Though headquartered on the Swiss-French border, groups contribute to its goals from around the world. It has been fundamental in progressing many areas of science and technology, including discovering antimatter and the Higgs Boson, and creating the World Wide Web [CERN. About].

The Large Hadron Collider

The Large Hadron Collider (LHC) is the largest, most powerful particle accelerator in the world, and is housed at CERN. The purpose of the instrument is to complete the Standard Model by discovering the Higgs Boson and look for physics beyond the standard model, often referred to as "new physics." In 2012, it succeeded in its first stated task, confirming the existence of the Higgs Boson and finally completing the long-standing current model of physics [CERN, About].

The Standard Model is the accepted model of particle physics. It was developed in the 1970s to concatenate disparate knowledge of particles into one internally consistent model. It describes all the particles we know of, the categories those particles inhabit, and how they interact. Alone, it provides explanation for most of the physical phenomena we have discovered. The Standard Model predicted the existence of multiple particles before they were found experimentally, including the Higgs Boson. As of now, all the particles predicted by the Standard Model have been found experimentally [CERN, Standard Model].

Not all observed phenomena are completely explained with the physics of the Standard Model, so we posit that more particles may exist. These particles would fall under "new physics" and are still being searched for with the LHC. So far, none have been found [Niewodniczanski Institute].

Physically, the LHC occupies a 27-kilometer tunnel crossing under the French and Swiss border outside of Geneva. During usual operation, hydrogen atoms are first ionized by an electric field, then passed through multiple smaller acceleration stages before finally being released into the LHC itself. Acceleration occurs via rippling electric fields applying force to the ionized hydrogen atoms (at this point referred to simply as protons), pushing them to higher and higher speeds in each stage [CERN, About]. You can see the acceleration chain in Figure 1. Protons are organized into clumps called "bunches," which are evenly spaced around the collider. Each bunch contains an average of $1.15 * 10^{11}$ protons. They are spaced 25 ns apart [RD53A].

Once the protons reach their maximum speed of 99.9999990% the speed of light, they collide together with immense energy. Collisions occur at "collision points," of which there are four spaced around the circumference of the detector. When a collision occurs, the energy contained in the two colliding protons converts into mass via Einstein's mass-energy relation. This mass can manifest as other, rarer particles, which stream outwards from the collision point. Each collision point has an associated experiment measuring and recording data from the collisions that occur there [CERN, About].

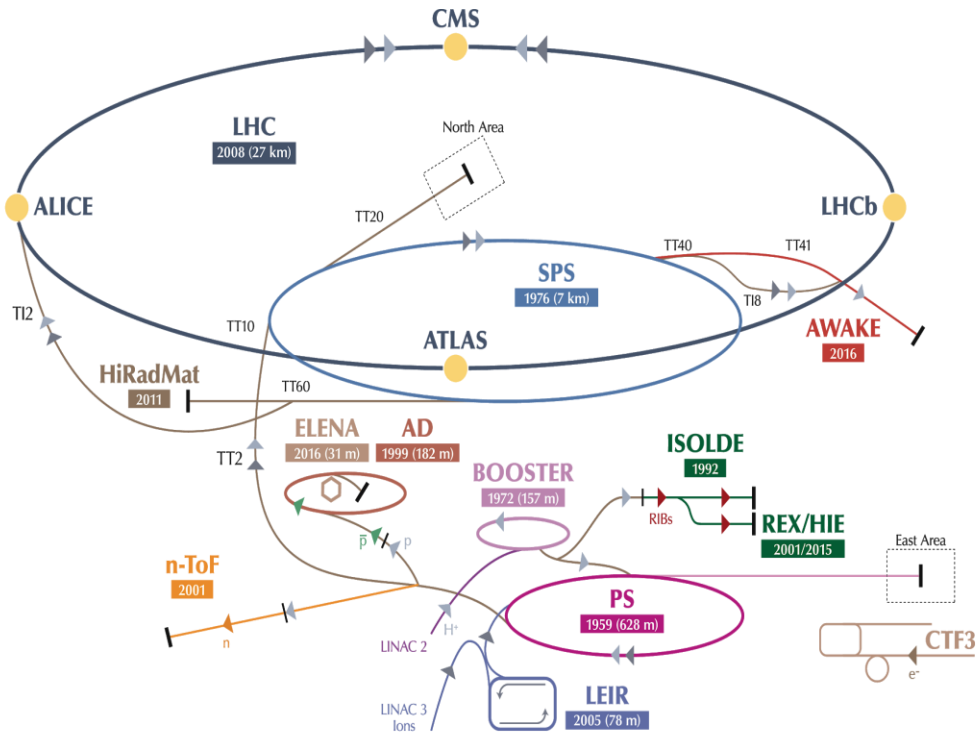


Figure 1. Protons bound for the LHC are released first at the LINAC2 before passing to the BOOSTER, the PS, the SPS, and finally the LHC. Though the LHC is the most important instrument used in the acceleration chain, you can also see examples of some other experiments at CERN here [CERN, About].

ATLAS

One such experiment located at the LHC, and the experiment relevant to this research, is ATLAS. ATLAS is a backronym for A Toroidal LHC Apparatus. It is one of the two general purpose detectors on the LHC, the other being the Compact Muon Solenoid (CMS). ATLAS gathers data from a variety of different instruments that don't necessarily have one singular goal in mind, but instead simply recording as much information as feasible for later analysis [ATLAS, Detector and technology].

ATLAS is made of concentric cylinders of detectors known as "layers," all situated about the interaction point. Each layer provides a different means of interaction with the particles, since not

all particles interact similarly. In this way, as many particles can be characterized as possible. As a rule, the closer the layer is to the collision point, the higher the density of particles it experiences, and therefore the higher the resolution of the detection mechanism. ATLAS also generates an incredibly strong magnetic field, which the particles move through as they pass through the layers. This magnetic field deflects the particles that have electric charge and helps us isolate their signatures by measuring their momentum. The detector is 46 meters long and 25 meters in diameter, weighing 7000 tonnes. Figure 2 displays a cutaway diagram of the ATLAS layers [ATLAS, Detector and technology].

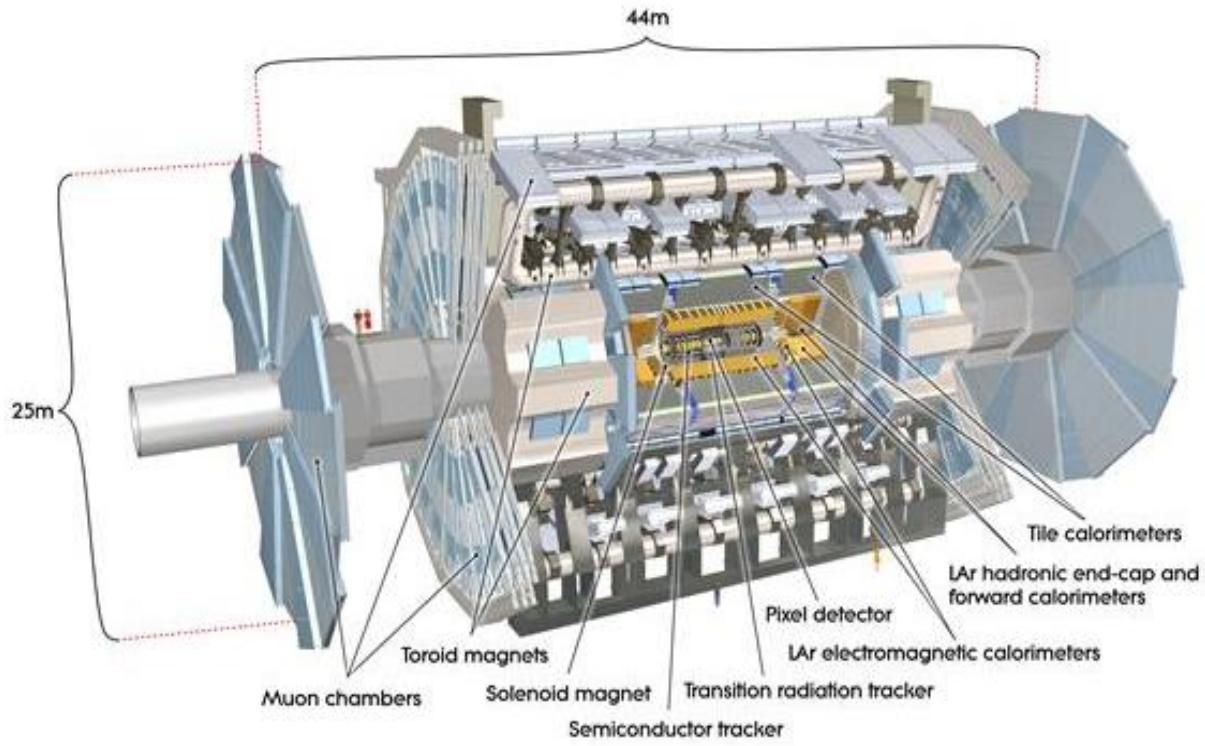


Figure 2. The dimensions and layers of the ATLAS experiment. The RD53A is built for the Pixel detector, closest to the beam [ATLAS].

Incredible amounts of data are generated by ATLAS. If all data channels were fully collected, 60 terabytes per second of storage would be required to save the raw data. This is untenable, so ATLAS uses a complex series of data analysis procedures to throw data away. When data is flagged as interesting, a “trigger” is sent to the read-out electronics telling it to store the data. Three levels of triggers are used in ATLAS, paring down the data in a cascade and ultimately storing only 100 out of 1 billion events [ATLAS, Trigger-DAQ].

Inner Detector

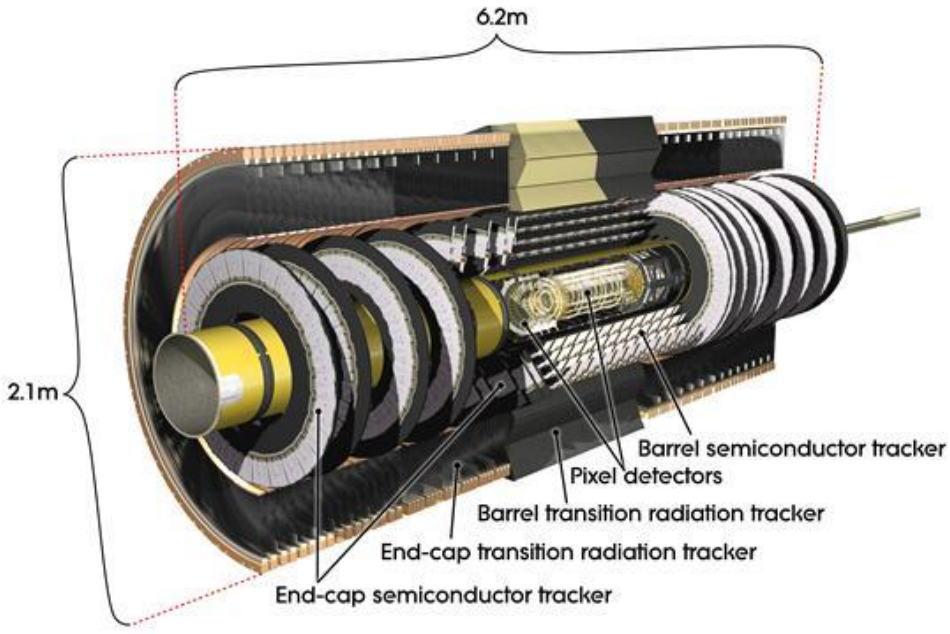


Figure 3. An image of the ATLAS Inner Detector. [ATLAS].

Different layers of the ATLAS detector are grouped by purpose. The innermost group, called the Inner Detector, is made up of three layers: The Transition Radiation Tracker (TRT), the Semiconductor Tracker (SCT), and the Pixel detector. As a group, their primary use is to measure the momentum of charged particles. The Pixel detector will be the focus of this paper. You can see a representation of the Inner Detector in Figure 3 [ATLAS, Detector and technology].

Pixel Detector

Pixel is the detection layer closest to the beam, experiences the most luminosity (particle flux), and is by far the most sensor dense of all the layers. In an area of 1.7 square meters, it provides 80 million independent channels of data [ATLAS, The Inner Detector]. Each channel corresponds to a “pixel,” which is the individual detection mechanism of the Pixel Detector. When a charged particle moves through a pixel, charge is deposited and digitized. Pixels channels report a “time over threshold” value, which is how long the digitized charge has been above a configurable threshold value and is proportional to deposited charges [G. Aad et. al., 10]. The Pixel Detector has a resolution of 14 micrometers by 115 micrometers [ATLAS, The Inner Detector].

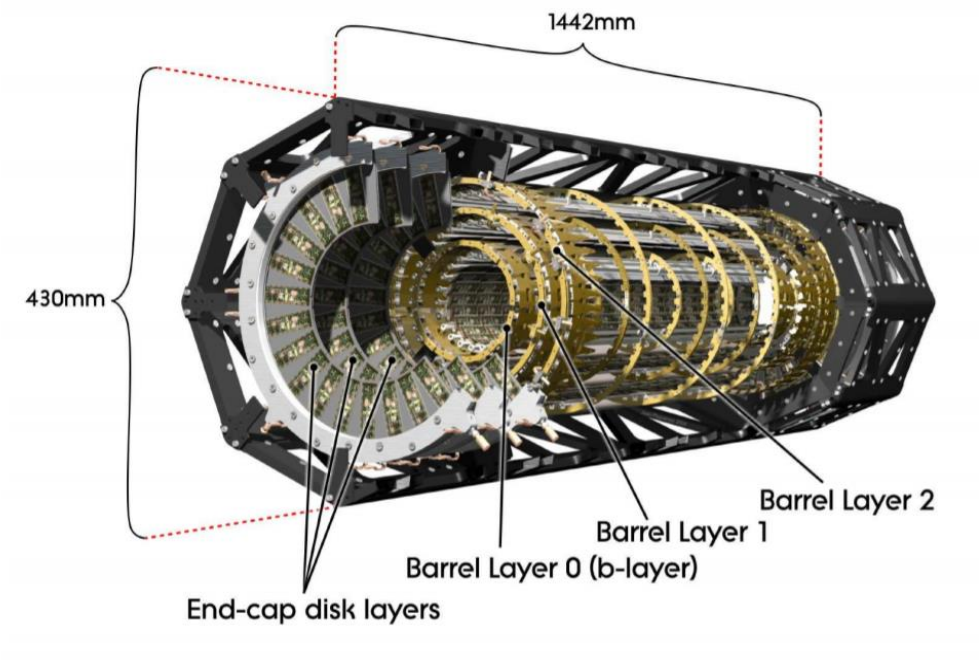


Figure 4. The layers of the Pixel Detector sans the IBL, which was installed later. All layers use the same Front End chips. The Detector is quite small compared to the whole of ATLAS and has by far the highest resolution density [G. Aad et. al.].

Reflecting the design of ATLAS in general, Pixel is made up of multiple overlapping detection layers. Originally just three, a fourth layer called the Insertable b-Layer (IBL) was added closest to the detector in 2014. Figure 4 is a rendering of the Pixel layers. These layers serve to track particles as they curve through the magnetic field present in ATLAS. Layers are overlapped to ensure all particles passing outwards can be captured by at least one sensor [G. Aad et. al., 5].

The chips containing the pixels themselves are called Front Ends (FEs) as they are the first device in the chain of devices that gather data from the interaction. Each front end contains a grid of 18 x 180 pixels of size 50 x 400 micrometers each. When a pixel is struck by an incoming charged particle, the analog part of the pixel has some charge deposited on it, which then gets amplified by an accompanying circuit. If the amplified signal is larger than a threshold value, the address of that pixel and some associated hit data is transferred to the digital part of the FE chip. There, the FE calculates how long that pixel has been over the threshold, and reports that back up the chain [G. Aad et. al., 8-10].

16 FE chips share a Module Control Chip (MCC) and make up a module. The MCC distributes control signals among its FEs and concatenates data coming from them to send up the chain. An Opto-Board (OB) takes the electronic data from the MCC and converts it to optical data to be sent by optical fiber to the off-detector readout systems, which do further processing on the data [G. Aad et. al., 8].

RD53A

HL-LHC

The LHC has scheduled shutdowns in order to upgrade the detector. In 2013-2014, one such shutdown allowed the installation of the IBL in Pixel described above. In 2024, a "long shutdown" (LS3) is scheduled that will extend over multiple years, allowing for significant upgrade of the detector [LHC Schedule Update].

After the end of the LS3, the LHC is scheduled to go to "high luminosity" (HL-LHC). Luminosity is a measure of the amount of collisions that occur per bunch crossing, and therefore the particle flux through the detector. This, in turn, will increase the amount of measured rare events so it can gather data on them more quickly, thereby delivering results sooner. Luminosity is scheduled to increase 10x from the original operating specifications after the upgrade. Figure 5 shows the shutdowns from 2010 to 2035 and their target luminosities [LHC Schedule].

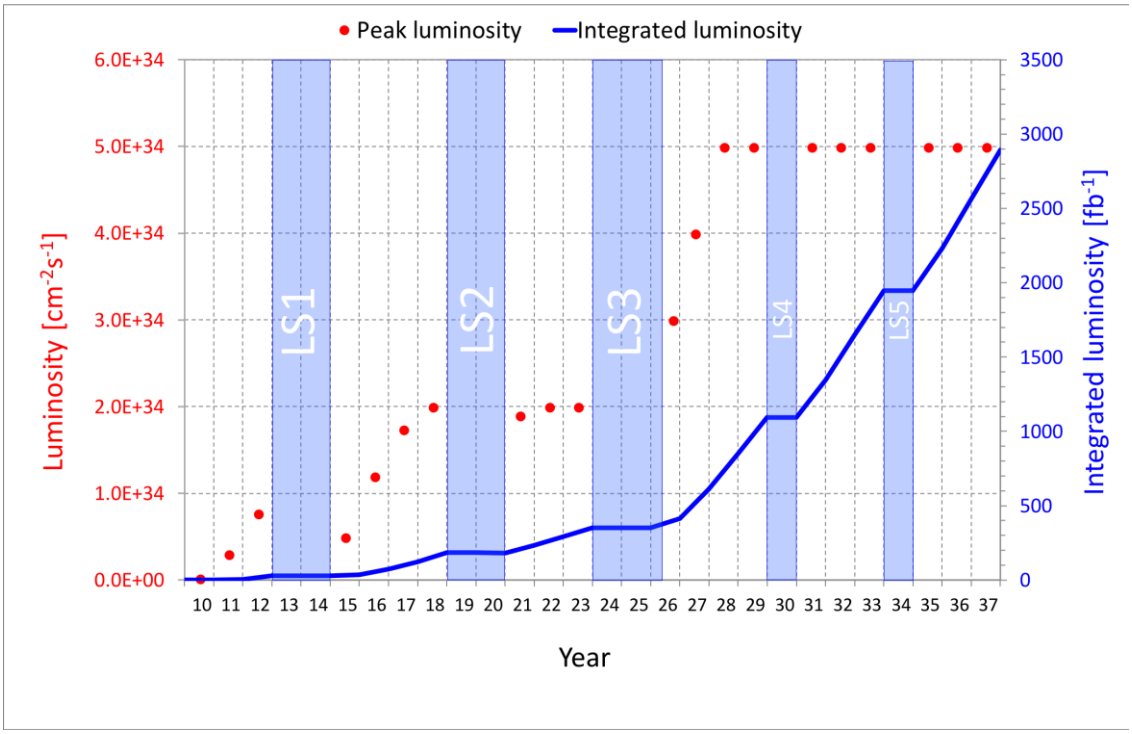


Figure 5. Planned and completed long shutdowns allowing for upgrades and maintenance, the peak luminosity at those dates, and the integrated luminosity at those dates [LHC Schedule].

Naturally, much of the hardware installed for the original detector luminosity will not be able to effectively process such a massive increase in information output. Some parts of the detector will need to be upgraded, including the entire inner detector [ITk Design Report, 5].

Inner Tracker Upgrade

As part of the upgrades needed to process the amount of data generated in the HL-LHC, the entire Inner Detector is being upgraded. The number of independent layers in the Inner Tracker (ITk) will be lowered from three for the current Inner Detector to two: the ITk Strip and ITk Pixel Detectors. New readout chains are being constructed as well and are still under heavy development [ITk Design Report, 5].

ITk Pixel Upgrade

During the ITk Pixel Upgrade, all parts of the Pixel readout chain will be affected, and many will be replaced. One such device being replaced is the front end readout system of the Pixel Detector. The functionality of the FE and MCC chips in the prior generation will be combined into a single Front End chip. Each chip will have an array of 50 x 48 pixel cores, each containing an 8 x 8 array of pixels. Each pixel core is surrounded by digital logic to read out and control that specific core. This architecture is referred in the documentation to as "analog island in a digital sea." At the bottom of each chip will be a dedicated "Chip Bottom" containing both analog and digital control signals, as well as digital readout. At the very bottom edge of the chip will be I/O. The new front end will be the first ATLAS chip made with a 65 nm technology node [ATLAS-TDR-030, 156].

The RD53A

The RD53A, or Research and Development Chip 53-A is a test chip fabricated to test out the technologies intended to be used by the HL-HLC FEs. There are a few main points of interest.

- The FE will be the first on-detector, ATLAS-made chip in 65 nm CMOS technology. Different technology nodes have different performance characteristics when exposed to the radiation present in the detector. It's important to ensure that the 65 nm node can perform at expectations while being irradiated. Not only must it be tolerant of damage due to irradiation, but it also must maintain stable low threshold operation and specification-satisfying trigger and hit rates, which both may be compromised by the energetic particles [RD53A, 3].

- The chip is a test bed for different analog pixel architectures. Three unique architectures have been built into the detection area of the chip. In this way, their merits can be compared through physical testing [RD53A, 3].

- Much of the design of the RD53A reflects what will be eventually be built into the FE, so the RD53A helps ensure those designs are effective and bug-free before the FE design is completed [RD53A, 3].

- The RD53A architecture is similar to that of the future FE with some slight rearrangements. The pixel matrix uses the "analog island in a digital sea" architecture for the pixel matrix and

includes a digital chip bottom for readout and control, as well. Some of the device placement is slightly rearranged [RD53A, 5].

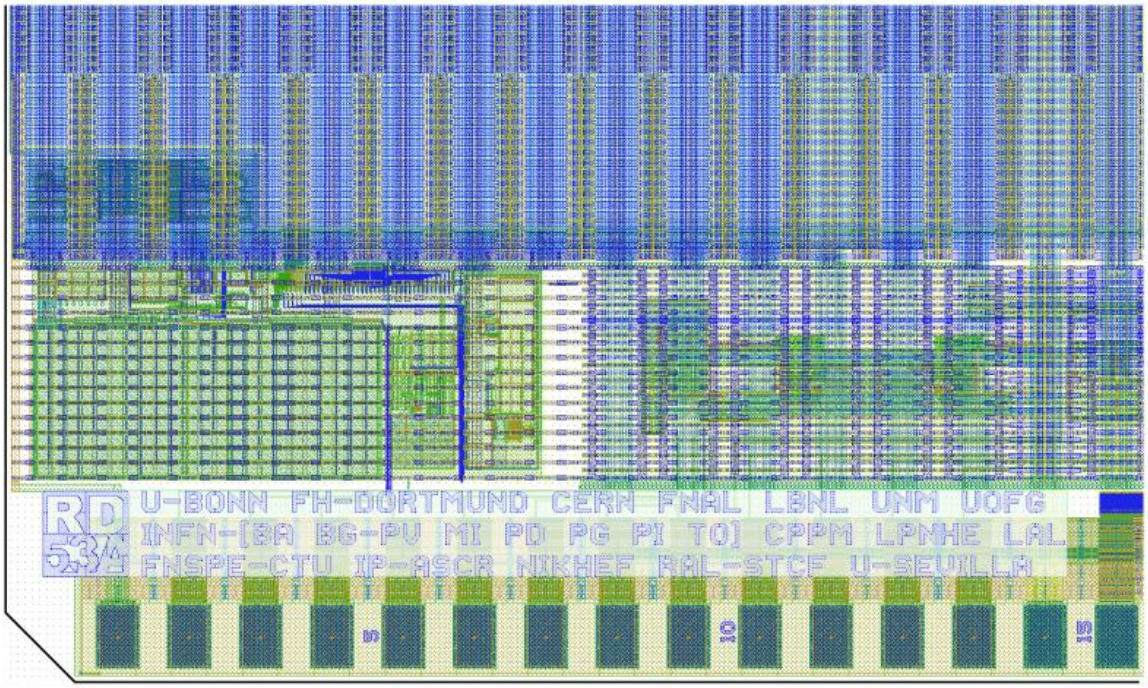


Figure 6. The final layout the of RD53A chip. Pixels are arranged as a grid on the top end of the chip [RD53A, 0].

The RD53A design was finalized October 8, 2017. You can see the final layout of the chip in Figure 6. The wafers returned from the manufacturer on November 27, 2017. They were diced at and distributed from CERN to the collaboration.

Chapter 2: RD53A Emulator

The RD53A Emulator (Emulator) is a project developed by ACME Lab at the University of Washington to mimic functionality of the RD53A test chip. Specifications for the RD53A chip were provided by the CERN collaboration [RD53A]. It targets a Kintex 7 FPGA installed on a KC705 breakout board [KC705].

The Emulator is a team project with contributions from many students over the years of its development. Although I did not personally contribute to all the components discussed below, it is valuable to record the workings of the entire system for future reference. It also helps to understand the system architecture before discussing developments in depth. My contributions will be specifically noted later in the document.

Motivation

The motivation for the continued development of this project has evolved over its lifetime. At first, the primary goal of the Emulator was to provide a RD53A-like device to help test various data acquisition (DAQ) systems in concurrent development with the chip, before the chip itself arrived from fabrication. These DAQ systems are intended to read out data from the RD53A chip. DAQs already exist in Pixel, but they must be upgraded both to keep up with the increase in data bandwidth in the HL-LHC and to conform to the specifications of the new Front End chips. These new data acquisition systems must be tested and debugged, which was made more difficult without the test chip to interface with. The Emulator solved this issue by providing a stripped-down facsimile of the chip. Additionally, the simplicity of the Emulator's design supported this goal by allowing for quick changes to the Emulator to fit the user's goals or fix small issues.

Later, as the RD53A chips began returning from fabrication, the need for the Emulator as a stand-in for the chip lessened. Some groups in the Collaboration would not be able to receive the chip for some time afterward, during which the Emulator provided an equivalent platform, but, ultimately, most interested groups received a chip after a waiting period. The purpose of the Emulator then evolved into its current purpose, which can be summarized in four points: Hit data emulation, comparison against the physical chip to isolate hardware issues, modifiability, and availability.

Arguably, the greatest motivation for continued development of the Emulator is its ability to emulate hit data. The RD53A must receive stimulus in the form of high energy particles to return interesting data. Although CERN houses an accelerator, and some labs working with the chip do as well, high energy particles are often not available to researchers on demand. A researcher might be able to interface correctly with the chip via their DAQ, but not all functionality of the DAQ systems can be tested without incoming hit data to process. The Emulator solves this problem by generating faux hit data when instructed by the user. Since the Emulator will never actually be used to gather data from the accelerator, we can generate it ourselves as a service for the end user. The data is intended to reasonably reflect the types of data patterns one would expect from the RD53A if it were irradiated by high energy particles.

The Emulator can also be used as a point of comparison when debugging issues interfacing with the RD53A chip. It is often the case that an engineer will be faced with an issue at the interface between two systems without any clear indication of which system is causing the problem. A tactic to solving this problem is replacing one of the systems with an equivalent and surveying if they issue disappears, thus isolating the problematic system. If an engineer has a problem interfacing with the RD53A chip, that engineer can substitute the Emulator for the chip to help solve that problem.

The Emulator provides an alternative to the RD53A chip that is easier to modify. Integrated circuits like the RD53A cannot easily be modified after delivery. If an issue arises with the chip, sometimes the only solution is to make a new version, which can take months [China Water Risk]. On the other hand, because the Emulator targets an FPGA, changes can be made magnitudes more quickly,

requiring only edits to the source code and a recompile. Thus, users can leverage the Emulator as a framework to test specific parts of their systems by targeting the code to do so. It also means bugs and other issues can be solved in a matter of hours, rather than months.

Finally, since the Emulator targets a commercial FPGA, it is much easier to acquire than an RD53A chip. The KC705 board must be purchased and the code downloaded from the online repository, but nothing more. If extra RD53A chips are required, they must be requested through collaboration channels and may not be available altogether if supplies run out.

Current State of Development

Architecture

The current overall system architecture is shown in Figure 7.

Trigger and control data enter the Emulator from the DAQ system via the Trigger and Timing Control (TTC) line. Triggers tell the system to generate hit data and control words alter the state of the system. The data rate and encoding scheme are described more fully in the following sections.

A synchronization pattern is sent along with the TTC data every N words, where N is defined by the DAQ. The synchronization pattern is chosen to be DC balanced and only valid in one rotational orientation; that is, there is no repeated pattern in the sync command. The pattern is used in the Shift Register (SR) Channel Alignment block to align the data so words are correctly bit rotated. The block has 16 different 16-bit registers, each of which shift in the incoming data rotated by one bit. Eventually, one of the 16 registers will see the expected sync pattern. Once it does, the block locks to the data in that channel and passes it to the next module. Sync patterns must be sent from the DAQ often for the channel to remain locked. Though it is not expected to occur, if the data slips such that the originally-correct channel is no longer correct, a new channel will be chosen based on whichever channel continues to see valid sync patterns. Most of the work described in this paragraph was done by Joseph Mayer, a Master Student in the ACME lab from 2014-2016 [FPGA DAQ Thesis].

Although the official word size for TTC data is 16 bits, it was decided early on to process it in 8 bit chunks in the Emulator. The data resulting from alignment falls into the “Word FIFO”, which splits it into two separate 8-bit words and helps it cross from a 160 MHz clock region to an 80 MHz clock region. There, triggers and control words are separated and processed in the “Hit Data Generator” and “Command Finite State Machine (FSM)” respectively. Each of these modules have their own dedicated explanations below.

The various types of data generated by the Hit Data Generator and Command FSM pass into FIFOs, which then get interleaved by the “Command Out” module, which serves as a frame formatter for outgoing data. This, too, has its own dedicated section below. One important note not covered in the

dedicated Command Out section is the module also generates Channel Bonding frames, which are a special type of frame used by our data protocol (also explained below).

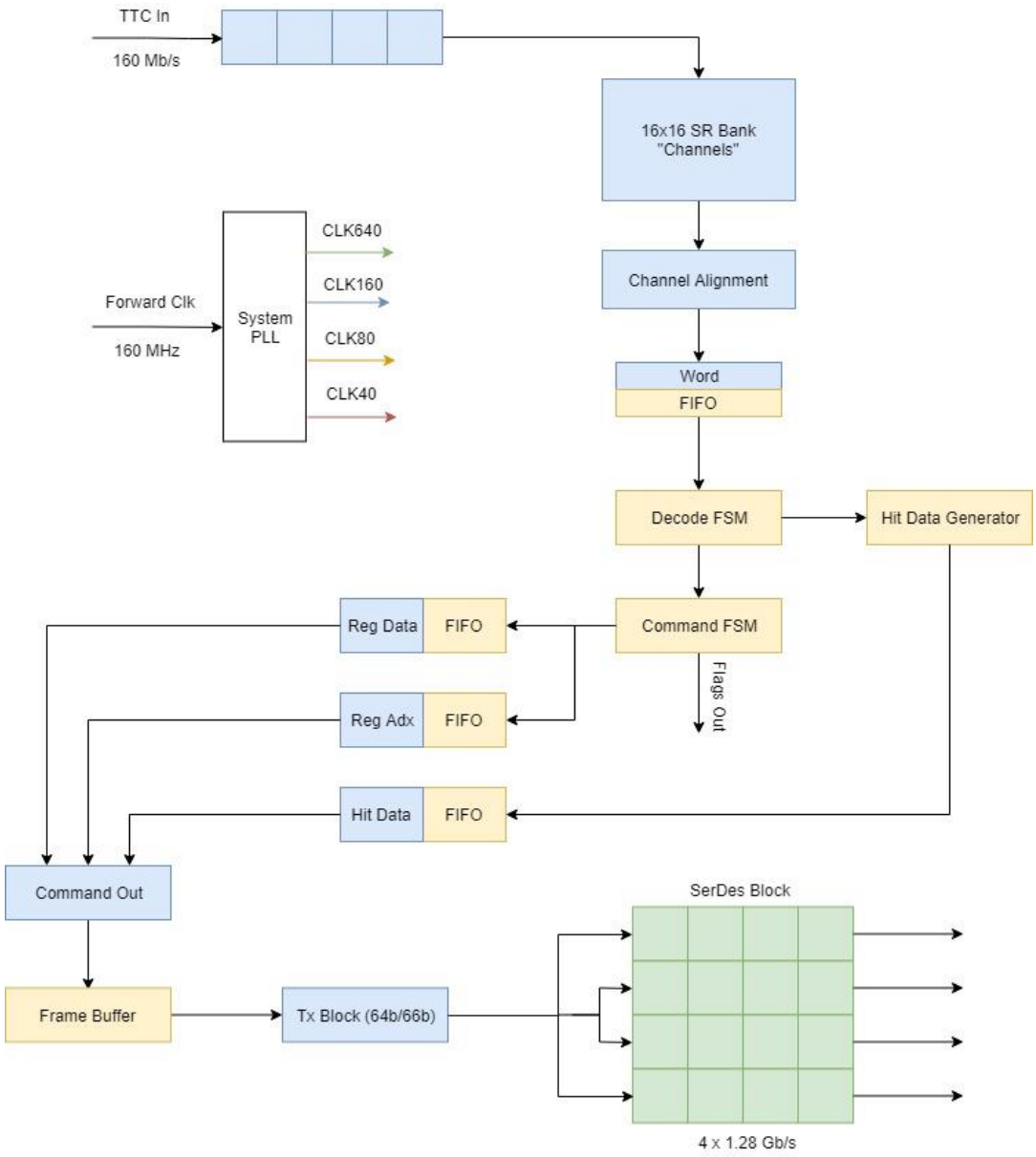


Figure 7. High level architecture for the RD53A Emulator.

Once frames have been formatted, they transition to the "Frame Buffer", a module that aligns four frames to be sent out in parallel by the transmission (Tx) block. As part of the data transmission protocol, data frames must be aligned in a specific way to ensure accurate data transfer. The Frame Buffer accomplishes that by storing four incoming serial frames, then raising a flag to let the Tx block know data is ready to be read out in parallel from the Buffer. After frames are buffered, they exit the system via the Tx block. See the Data Transmission section for more information.

Hit Data Generation

It could be argued that the most valuable use case of the Emulator after the official release of the RD53A chip is as a means to generate pseudo hit data. The RD53A chip being intended for irradiation requires some kind of particle stimulation to provide any data of interest--it does not have a means to generate data on command. This, of course, would be entirely unnecessary once the chip is present in the detector, since it generally will be under constant irradiation anyway. However, the chip is still going through testing, as well as the DAQ systems that read out from it. It is not always easy to provide radiation to stimulate a chip like the RD53A; sometimes, testing is more feasible outside a high energy environment.

The Emulator solves this problem by not requiring radiation to generate hit data. We want to recreate the RD53A chip in black box style, but it need not use the same mechanisms internally. To that end, we have implemented hit data generation on the Emulator. It recreates particle stimulus one might see when the RD53A chip is placed in the detector without needing to move it from your desk. Such a hit generator can be used to test the DAQ systems that read data out of the RD53A.

To determine the best way to recreate lifelike data, we both consulted with ITk experts including Timon Heim of Lawrence Berkeley National Lab and analyzed software simulated hit data. We were given "frame" snapshots of simulated pixel arrays at various positions around the collision point. Quickly, it was noticed that a single particle "hit" could stimulate multiple adjacent pixels, which we went on to call a "bunch". We then did analysis on the probability of how many bunches would appear in a single frame and what those bunches would look like. It was determined that two major shapes would appear. One, a circular blob of pixels over threshold we took to mean a particle flying through the chip. Two, a collection of pixels aligned linearly that would extend many pixels in length, which we took to be an interaction occurring in the chip ejecting a particle perpendicularly to the direction of source particle travel. Our ITk experts agreed these two shapes would likely make up most expected interactions and confirmed for us the rate at which they would occur. The top 11 cluster shapes and their appearance probabilities can be seen in Figure 8.

Our next step was then to implement that hit generation. At the time of writing this thesis, a simple version has been constructed that generates the above blobs, which we refer to as "Tetris pieces". Triggers are decoded in the "Decode FSM", then passed into the hit data generator and fall into a Trigger Table, which stores them and their associated metadata until they resolve. When a trigger resolves, the module generates a few Tetris pieces, which are read two hit regions (group of four pixels) at a time by an accompanying FSM. However, sometimes the FSM will throw that data out entirely and output an empty frame representing the absence of a hit. The likelihood of a blank frame being generated is proportional to the value stored in Global Register 134, SKIP_TRG_CNT. The FSM then loads a hit data FIFO with the resulting hit data.

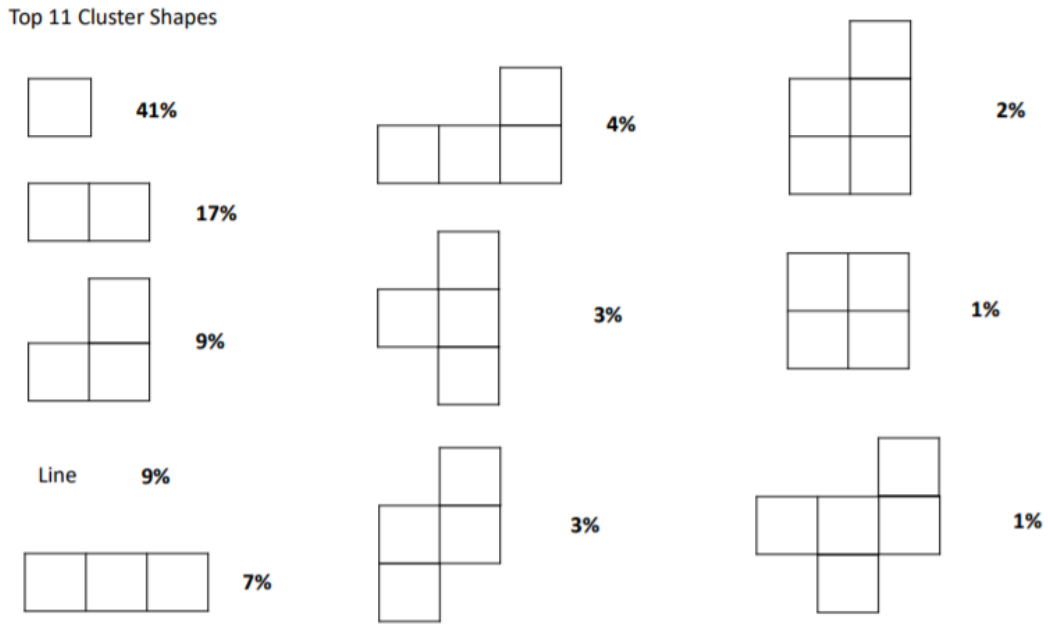


Figure 8. The eleven most common cluster shapes (orientation-independent) and their appearance likelihoods.

Data Transmission

Data is transmitted bidirectionally to and from the Emulator on dedicated transmission lines with line-specific data rates and topologies. Control data is sent to the Emulator via the TTC line and produced hit data is read out via 4x1.28 Gbps SerDes.

The TTC line is read via a single differentially-terminated Input SerDes (ISERDES) running at 160 MHz. It is generated in the DAQ. Unlike the Chip itself, the Emulator does not include clock recovery on the TTC line, instead receiving a forwarded clock from the interfacing DAQ. Attempts at recovering the clock from the TTC line were made, but eventually abandoned due to design limitations of the FPGA. The TTC data is sent as 16-bit blocks (therefore, blocks are received at 10 MHz). The TTC data is encoded according to the specifications of the RD53A documentation and is outlined in Table 1.

Each block sent is DC balanced. A DC balanced signal spends an equal amount of time in the high and low voltage states over the length of the signal. DC unbalanced signals can tend to bias the logic receiving them towards whatever voltage level is dominant in the signal, and in extreme cases can cause data transmission failures. Naturally, this is something we want to avoid.

Triggers are sent in sets of 4 describing which of the proceeding 4 bunch crossings should collect data. Triggers are encoded in 8 bits of data. Since each data block is exactly 16 bits, half of the block

is devoted to which bunch crossings the triggers occur on, and half the block is a data identifier for that set of triggers.

When triggers are not being sent, the TTC line is used to control the state of the Emulator. Commands are initiated with 8 bit encoded words, repeated twice, except for the sync command, which is 16 bits by default. Some commands, such as global register read and write, will have extra data fields appended to the transmission. To ensure DC balance, these fields need to be encoded as well, and are done so with an 8 bit → 5 bit mapping described in Table 2.

Unlike TTC data, chip readout data follows a standardized protocol and implementation. The protocol used by chip readout is called 64b/66b and is used in various communications standards including

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

Table 1. Trigger patterns that can be received on the TTC line. Two patterns will be received per block [RD53A, 46].

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	1010_1010	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

Table 2. Data encoding scheme to fill the applicable data fields of non-trigger commands [RD53A, 48].

10G/100G Ethernet [Ethernet, 5], InfiniBand [InfiniBand, 92], and Thunderbolt [Thunderbolt]. The coding standard is described in Xilinx's Aurora IP Protocol Specification [Aurora] but is complicated and lies outside the scope of this report. More relevantly, the protocol delivers on a few requirements: Run length, DC balance, Hamming distance, and limited overhead.

It is necessary to have regular voltage transitions to clock recover on an incoming data line. The 64b/66b protocol guarantees a transition occurs at least once every 66 bits; that is to say, the run length of the 64b/66b protocol is at most 66. It does this by prepending a "sync header" to the front of the 66 bit data frame, thus the name 64b/66b. The sync header is always either a "10" or a "01", with "11" and "00" failure modes [Aurora, 41].

As described above, a DC unbalanced signal can cause biasing at the receiving data pin. Though 64b/66b does not guarantee DC balance, it greatly improves it, especially in the case of sequential data runs.

The 64b/66b protocol also guarantees at least a 4 bit Hamming distance between valid data frames. In simple terms, this means at least 4 bits must be flipped in transit for the protocol to not be able to detect a failure.

Finally, the 64b/66b protocol is concerned with minimizing protocol overhead. Similar protocols like the 8b/10b protocol have overheads of 25%, whereas 64b/66b only has 3.125%, thereby using many less "extra" bits to convey the same amount of information.

Xilinx provides IP in the form of the Aurora 64/66b IP to implement this protocol and instantiate it on the device. However, as part of our project developing the Emulator, our team also developed a "soft" version of the Aurora 64b/66b protocol that doesn't use the "hard" Xilinx IP. The Xilinx IP requires using a specialized part of the FPGA called a GTX Transceiver, which is a "hardened" piece of logic devoted to sending data at high speeds. Unfortunately, there are only a few such GTXs on most FPGAs, and some Xilinx FPGAs have none whatsoever. Using them, therefore, implies allocating very limited resources to the task at hand, often sacrificing other important utilities. Our team solved this by recreating the protocol as described in the documentation, using ubiquitous SerDes pins instead of the limited GTX pins. To that end, the Emulator can be used in more varied environments.

Readout data is formatted as described in Figure 9. Data frames are prepended with the sync header "01", and all other frames are prepended with "10". Register frames hold information from the global register reads, and are sent every N frames, where N is programmable via a parameter in the Emulator code. Specialty frames of note are idle frames ((c) in Figure 8) where no data is available and therefore none is sent, and channel bonding frames, which help align data on the receiving end of the Aurora protocol. Channel bonding frames are sent every M bundles of 4 frames, where M is programmable via a parameter in the Emulator code.

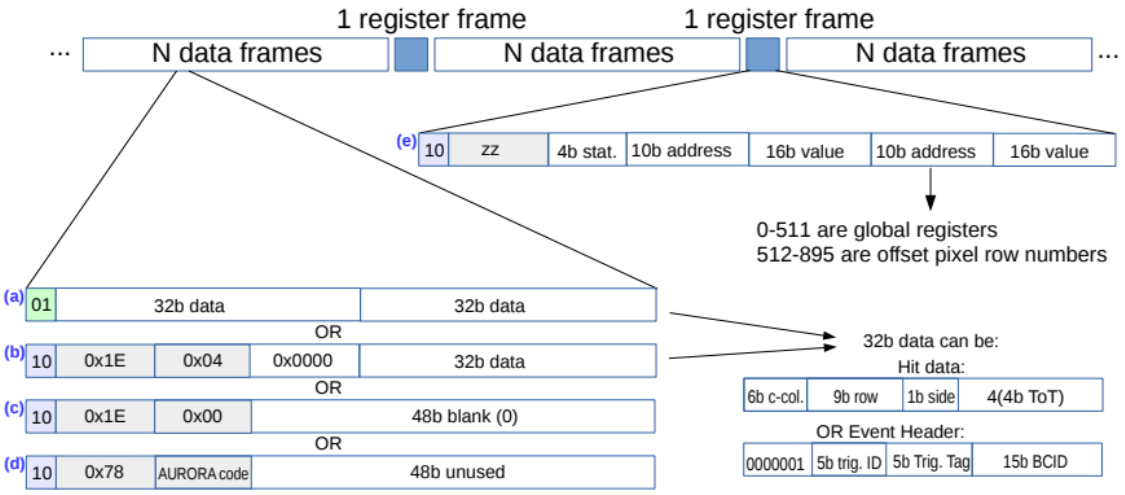


Figure 9. Formatting of outgoing readout frames in the Emulator. (a) is a double data frame, (b) is a single data frame, (c) is an idle frame, (d) specialized Aurora frame usually used for Channel Bonding, and (e) is a register frame. Data frame formatting is still under development [Aurora, 54].

The vast majority of the work on the soft Aurora protocol was done by Lev Kurilenko, a Master Student at ACME lab from 2016-2018 [FPGA High Speed I/O Thesis].

Personal Contribution

These are the major developments I have personally spearheaded in the Emulator as a Master Student.

Architecture

Near the beginning of my work on the Emulator, the system architecture looked akin to Figure 10. For reference, the current state of the architecture is seen in Figure 7. The system was significantly more complicated than it is today. More clocks were used in the architecture, generated from more independent circuits. Unnecessary blocks were added that served purposes not required by the specification. Now, the Emulator architecture is streamlined, partially due to some deviations from the specification. It now also has more configurability for the user.

The largest rearchitecting step I undertook was removing clock recovery on the TTC line and all associated logic and clock paths. In the RD53A chip itself, the clock *is* recovered on the TTC line, and the recovered clock serves as the clock reference for the entire system. This is functionality the Emulator originally attempted to reproduce. However, the decision was made to recover the clock in an all-digital "Clock Recovery" module (see Figure 9), which ultimately proved to be too imperfect a clock recovery scheme to effectively lock data channels via the Soft Aurora data interface. When it

came time to sync up the Emulator with a testing DAQ, the data Tx was unable to lock with the Rx on the DAQ side. The decision was therefore made to forward the clock from the DAQ to the Emulator to ensure the clocks are synchronized. Extra pins were available on the interface between the Emulator and the DAQ, pins which could be requisitioned for forwarding the clock.

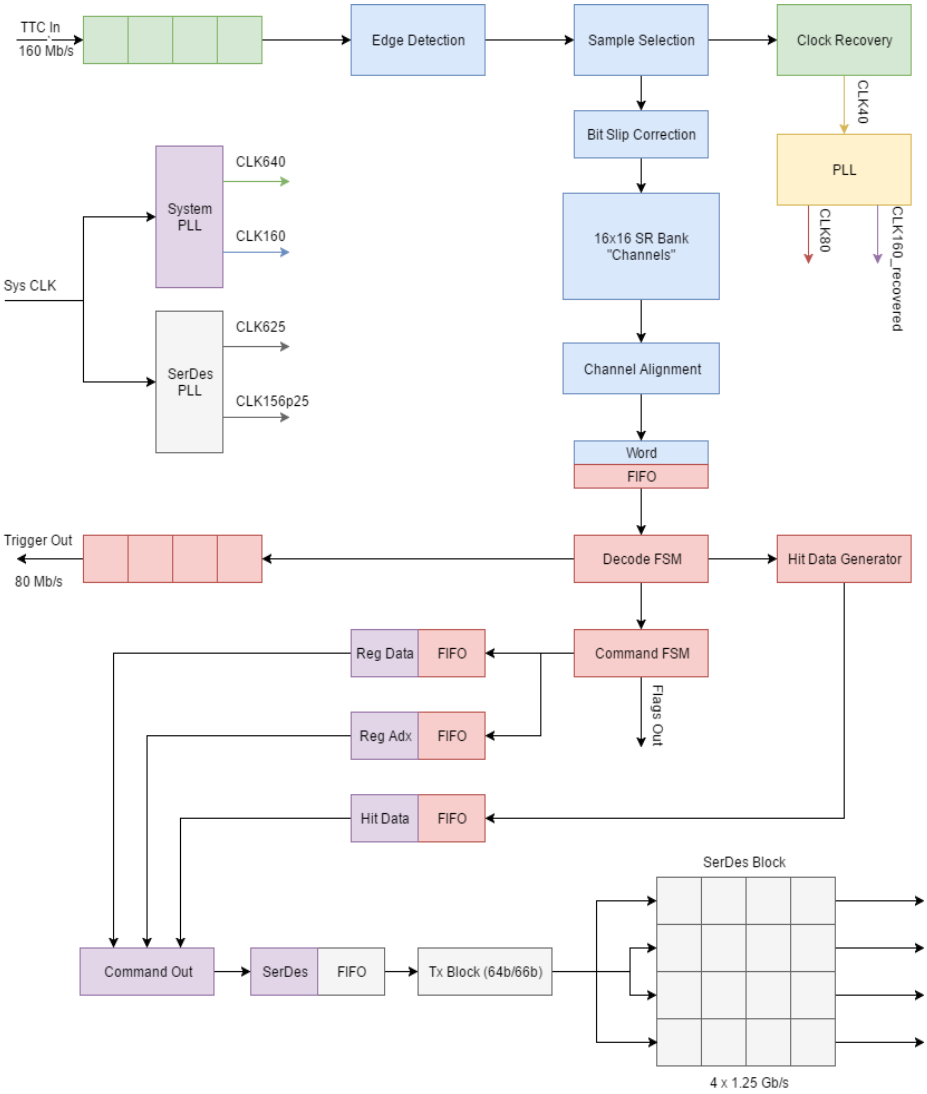


Figure 10. The state of the Emulator system architecture in June

It then became my task to go about editing the architecture to make use of a forwarded clock. The System PLL (sys PLL in Figure 10) was removed entirely. The Clock Recovery block and Recovered Clock PLL (PLL) were removed as well. I also simplified the clocking scheme by removing the SerDes PLL, as it was found that the output SerDes could run at 1.28 Gbps, even though they are only rated to 1.25 Gbps.

Using multiple clocks in a system presents the issue of synchronizing the points where data crosses from one clock domain to another. If the clocks are generated at separate sources, you cannot

guarantee that they run at exactly the same frequency or maintain the same phase, which means you cannot deterministically pass data from one such clock domain to another. To solve this, we originally implemented a plethora of FIFO blocks. These FIFO blocks can be run in two separate clock domains at once, and guarantee data is preserved when passing from one to the other. However, after lowering the number of clock sourcing PLLs from 3 to 1, I could then deterministically calculate when data would be handed off from one domain to another, therefore allowing me to remove some of the domain-crossing FIFOs. In the end, the design was greatly simplified by forwarding the clock, and, more importantly, solved the synchronization problem in the Soft Aurora cores.

Another, smaller change I made to the Emulator architecture was the removal of the "Trigger Out" unit also present in Figure 10. This was originally developed to help with debugging the Emulator's trigger-recognizing functionality. It eventually became incorporated into the permanent architecture, despite not being part of the specification or providing any real use. I removed it, as well.

The final change I made to the system architecture, and indeed one of the last changes I personally made to the Emulator altogether, was to modularize the Emulator as an entire unit and allow instantiation of multiple Emulator copies at the same time. This was done at the top level of the design. The entire Emulator was bundled into a single module, its pins connected through parameterized ports in a higher, top-level module where the copies are instantiated. I implemented a parameterized generate statement to allow the user to specify how many Emulator instantiations should be made. All the user need do is change one parameter. Each Emulator alone takes about 3% of the Kintex 7's resources, so multiple of them can fit on a single chip. Helpfully, even though they generate entirely separate output data streams and therefore need dedicated output pins for each, they all may share a single forwarded clock from the interfacing DAQ, which amortizes the extra pins required in implementing the clock forwarding across all the Emulator instances.

Command Processor

My development on the command processor was the first significant project I undertook as a member of the Emulator team. In short, it takes in encoded TTC data that has already been filtered for triggers, then decodes and responds to it. Since my time developing the module, a lot of further functionality has been implemented; I will touch on it here when relevant, but the focus of this section is to explore my contributions, not the contributions of others.

The Global Registers are instantiated in the command processing module. When I first developed it, it simply instantiated an empty memory which could be written to and read from via the relevant commands. Since then, the Global Registers have seen further development. Each register now has a specified reset value, which is defined in the command processor module. At startup, all registers are set to their initial values.

Most Global Registers are not relevant to our Emulator, as they serve functions in the analog domain such as setting voltage biases. In this case, they reset to their default value as expected, and can be

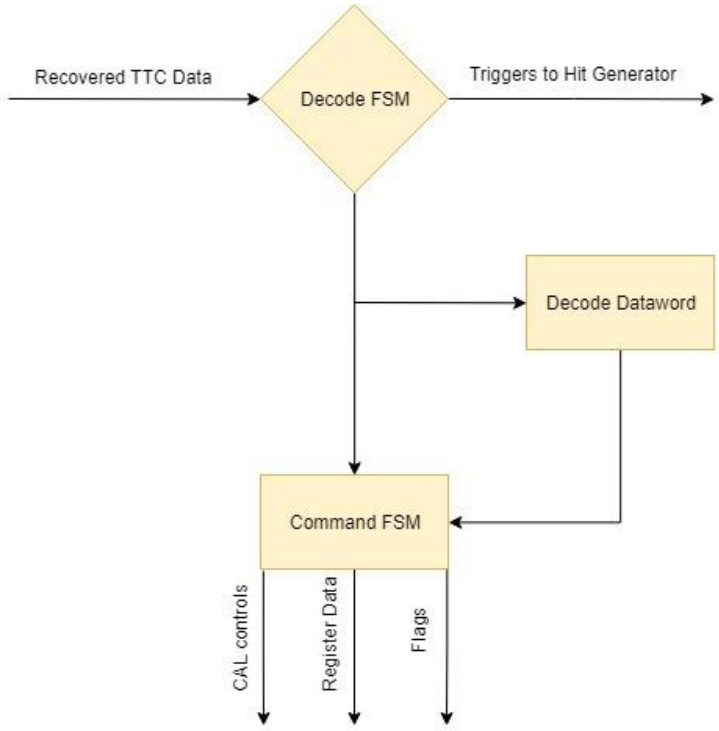


Figure 11. A brief microarchitectural description of the command processing system. Once triggers have been filtered out of the TTC data, the Decode FSM passes the data to the command process module, which simultaneously interprets it as a command and a dataword, eventually discarding the incorrect interpretation.

read and written as expected. However, they serve no other purpose as they do not control any logic. All registers are read/write; there are no read-only registers.

Regarding the data decoding microarchitecture, see Figure 11 for a visual representation. The microarchitecture is not terribly complicated, but it is worthwhile to briefly consider it.

Encoded data from the TTC alignment and recovery block is passed first to the "Decode FSM". The Decode FSM is a small FSM that demultiplexes trigger and command data, which then gets passed on to the interpreting module. If the data is a trigger, the trigger is decoded and passed to the hit generator. If it is command data, it is left encoded and passed to the command process module. Though the Decode FSM was originally developed before me, the trigger encoding scheme was altered in RD53A, and I implemented the updates.

Once triggers have been filtered out of the TTC data, the Decode FSM passes the data to the command process module, which can be seen as two separate blocks: First, a Command FSM interprets commands, interfaces with the Global Registers, sets flags, and generally responds to the sequential nature of multiword commands. Second, a Decode Dataword block decodes incoming data as

ID/Address/Data 5-bit Fields, rather than command initialization words, which then get fed back to the Command FSM.

Command	Encoding	ID/(A)ddress/(D)ata 5-bit Fields					
ECR	2× 0101_1010						
BCR	2× 0101_1001						
Glob. Pulse	2× 0101_1100	ID<3:0>,0	D<3:0>,0				
Cal	2× 0110_0011	ID<3:0>,D15	D<14:10>	D<9:5>	D<4:0>		
WrReg	2× 0110_0110	ID<3:0>,0	A<8:4>	A<3:0>,D<15>	D<14:10>	D<9:5>	D<4:0>
WrReg	2× 0110_0110	ID<3:0>,1	A<8:4>	A<3:0>,D<15>	D<14:10>	9×(D<9:5>	D<4:0>)
RdReg	2× 0110_0101	ID<3:0>,0	A<8:4>	A<3:0>,0	00000		
Noop	2× 0110_1001						
Sync	1000_0001_0111_1110						

Table 3. Non-trigger commands receivable and their data fields, if applicable [RD53A, 46].

There are 9 possible commands as described in Table 3. By default, when a command is seen, a flag is set for one 80 MHz clock cycle corresponding to the seen command. These flags are available as ports of the command processor for users to interface with but are currently unused.

Other than the flag behavior, not all the commands are implemented in the module. Specifically, ECR, the Event Counter Reset, and BCR, the Bunch Counter Reset, do not change the system state in any way other than raising their respective flags for one cycle. In that way, they respond very similarly to a Sync or a Noop command--raising their flag, but otherwise doing nothing (the Sync command is used in other design modules).

The Global Pulse command sets a value in a dedicated register equal to the value of the datawords following the command word. This register is not a Global Register and is not accessible externally. At the same time as the register is set, the Global Pulse flag is raised. The register decrements every cycle until it reaches zero, at which time the Global Pulse flag is lowered.

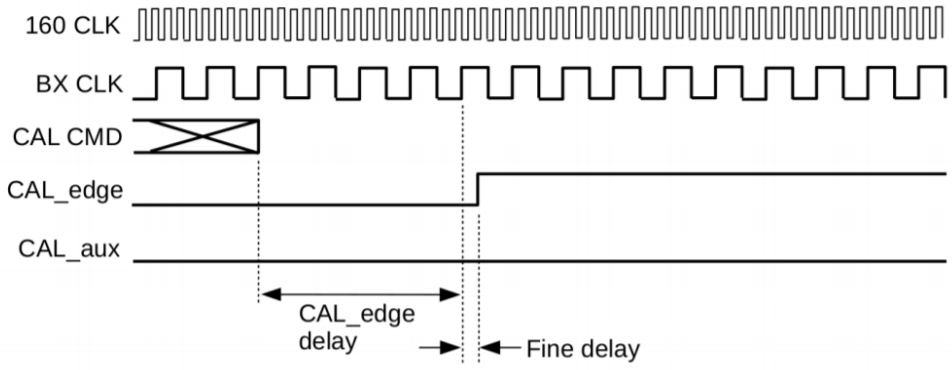


Figure 12. Diagram of the basic functionality of the Cal command. Fine delay is not implemented [RD53A, 48].

The CAL command is used in the RD53A chip to calibrate its analog front ends. As we do not have an analog front end, we interpreted this calibration step as transitioning the hit generator from producing incorrect data to correct data. So, the cal command must be sent from the DAQ before hit data can be generated. Internal registers track the data provided by the Cal command as described in Table 3. In order to replicate the bunch crossing rate, 40 MHz, I double the incoming values and count them down with the available 80 MHz clock. CAL_edge goes high after "cal edge delay" number of cycles, as long as "cal pulse" is not set. CAL_aux goes high after "cal aux delay" number of cycles, as long as "cal aux active" is set. "Cal edge count" does nothing at this time. Once CAL_edge and CAL_aux are set, they remain set until the global reset line is raised. You can see some of the functionality of the Cal command in Figure 12, though no delay "fine tuning" is possible in our design.

The read and write functionality is as expected from the manual, except there is no support for the alternative extended write command. If the extended write command bit is set and the Emulator is being simulated, an error statement will be printed, but otherwise the system will continue assuming the normal length write command. All the Global Registers can be read and written via this command, independent of whether they are supposed to be read-only in the specification or not.

Frame Formatter

Though the frame formatter, "Command Out", has evolved some since my time working on it, in many respects it is the same. Relative to the command processor, it is very simple. The frame formatter time multiplexes hit data with register data to be read out through the Aurora module. Frames are produced at 80 MHz. Since each frame is 64 bits, that makes for 5.12 Gbps generated data, which is the amount supported by the output SerDes. The formatter is fed by three FIFOs: the "Hit FIFO", the "Register Data FIFO", and the "Register Address FIFO". Those FIFOs are fed data by the hit generator and the command processor respectively.

A parameter in the Command Out code controls how often a register frame is released. By default, it is set to 49; that is, 48 frames of data with one following register frame. To prepare a data frame, it requests data from the Hit FIFO. If data is available, it sends out that data on the next cycle. If data is not available, it sends out an IDLE frame as defined in Figure 8.

When a register frame is about to be sent, the frame formatter prepares by requesting data from the Register Data and Register Address FIFOs early. Since each FIFO provides one word at a time and register frames can hold up to two register reads, the FIFOs are each requested twice, with the result of the first request stored in temporary local registers. If data is not available, the frame formatter will send an IDLE frame. If one register read is available, the data will be sent padded with zeros as per Figure 8. If two or more register reads are available, two register reads will be sent.

Chapter 3: Conclusion and Future Work

This thesis describes the RD53A chip Emulator, with emphasis on the design contributions by Dustin Werran. The Emulator is an FPGA-targeted hardware design that recreates the black-box functionality of the RD53A test chip. Multiple factors may make the use of the RD53A chip itself difficult, including the availability of the chip, the limited development speeds imposed by physical hardware design, and the inability for the chip to self-generate realistic particle hit data. These issues are all solved by the Emulator.

I personally saw to the rearchitecting of the Emulator to simplify its design. A forwarded clock is expected from the DAQ system it interfaces with, which is then split into various clocks across the Emulator to drive all the components of the system. I also modularized the Emulator so that multiple copies of the design could be implemented in parallel on a single FPGA, allowing a DAQ system to interface with more than one Emulator at the same time.

I also implemented most of the command processing logic in the Emulator. The command processing logic takes control information from the connected DAQ and changes the state of the Emulator based on that control data. The most common command types are Global Register reads and writes, which are specialized registers that control certain functions of the RD53A chip. Though most of the equivalent registers in the Emulator do not have equivalent functions, they can still be accessed by the commands as expected. The command processing logic is implemented as a finite state machine.

A smaller task I undertook is the development of the Frame Formatter, which formats data frames going out of the Emulator to the DAQ. Multiple types of frames can be sent from the Emulator to the connected DAQ, including data frames, which read out generated hit data, and register frames, which read out information from the registers when requested.

The Emulator code is wholly contained in a repository hosted on CERN's Gitlab. The repository address is linked in Appendix A. The repository also contains other code specific to some of the various DAQs currently in development for ITk.

Future Work

Work on the Emulator is ongoing. At the time of writing, most of the work being done is focused on improving the hit data generation functionality of the Emulator. Currently, the Emulator only generates a single type of hit we call a "Tetris" hit. In the short term, more types of hits will be implemented, including "Lines". The generator will also be made more robust to increase the number of hits possible at the same time.

In the long term, the Emulator will likely be repurposed to emulate the RD53B chip, which is the successor to the RD53A, and is currently in development. There are some significant changes to the

data protocols used in the RD53B that will need to be implemented in the future Emulator to do so. The use case for the RD53B Emulator will likely be very similar to the RD53A Emulator. Because a lot of the code in the RD53A Emulator could be reused for these updates, the RD53B Emulator could be finished in less development time, possibly releasing before the chip itself. If so, the Emulator could be provided as an alternative to the chip as an early platform for testing ITk DAQs.

References

[CERN, About]. CERN, “About,” January 22, 2019. Website.
[CERN, Standard Model]. CERN, “The standard model,” 2019. Website.
[Niewodniczanski Institute]. The Henryk Niewodniczanski Institute of Nuclear Physics Polish Academy of Sciences, “'New physics' charmingly escapes us,” August 3, 2018.
[RD53A]. The ATLAS Collaboration, “The RD53A integrated circuit,” Memo. CERN-RD53-PUB-17-001, September 15, 2018.
[ATLAS, Detector and technology]. ATLAS, “Detector and technology,” 2019. Website.
[ATLAS, Trigger DAQ]. ATLAS, “Trigger and Data Acquisition System,” 2019. Website.
[ATLAS, The inner detector]. ATLAS, “The inner detector,” 2019. Website.
[G Aad et. al]. G Aad et. al., “ATLAS pixel detector electronics and sensors,” *Journal of Instrumentation, JINSTI*3 P07007, 2008.
[LHC Schedule]. LHC Commissioning, “Nom-to-2037,” 2019.
[ITk Design Report]. ATLAS Collaboration, “Initial design report for the ITk,” 2014.
[ATLAS-TDR-030]. ATLAS Collaboration, “Technical design report for the ATLAS Inner Tracker Pixel Detector,” Memo. CERN-LHCC-2017-021; ATLAS-TDR-030, June 15, 2018.
[Ethernet]. M Hajeduczenia, “64b/66b line code.”
[InfiniBand]. InfiniBand Trade Association, “InfiniBand Architecture Specification Volume 2, Release 1.3,” November 6, 2012.
[Thunderbolt]. C Loberg, “Understand and test 10G Thunderbolt technology,” July 20, 2012.
[FPGA DAQ Thesis]. J. Mayer, S. C. Hsu, S. Hauck, “Three generations of FPGA DAQ development for the ATLAS Pixel Detector,” June 3, 2016.
[Aurora]. Xilinx, “Aurora 64B/66B protocol specification,” October 1, 2014.
[FPGA High Speed I/O Thesis]. L. Kurilenko, S. C. Hsu, S. Hauck, “FPGA development of an emulator framework and a high speed I/O for the ITk Pixel upgrade,” June 6, 2018.

Acknowledgements

It's hard to condense the vast number of people who have supported me, directly or indirectly, into a single list. If you're not noted here and feel like you should be, I'm sorry to disappoint you.

First and foremost, thank you to my advisors Scott Hauck and Shih-Chieh Hsu. This goes without saying, but my work would have not been possible without them. I remember walking into Scott's office as an undergrad looking for a graduate research position, not at all sure of myself. By the end of the conversation, after the idea of a trip to Geneva had been floated, I knew my decision had been made.

Thank you to my peers who shared the work on this project with me. It's impossible to state just who worked on what parts of the Emulator, since we collaborate so closely and ultimately all have our fingers in everybody's business. Nonetheless, in short:

- Architecture: Joseph Mayer, Logan Adams, Lev Kurilenko, Douglas Smith
- Channel Alignment: Joseph Mayer, Logan Adams
- TTC Processing: Joseph Mayer, Logan Adams, Douglas Smith, Tony Faubert
- Hit Data Generation: Douglas Smith, Tony Faubert, Jessica Lan
- Frame Formatting: Douglas Smith, Michael Walsh
- Frame Buffering: Michael Walsh
- Aurora: Lev Kurilenko, Timon Heim

Thank you to the ITk Pixel team at LBNL, and especially Timon, for his encyclopedic knowledge of ATLAS and sage, if snarky, advice. Our team would certainly drift off track without his continued presence at our weekly meetings.

A big thank you to the ATLAS Pixel Online team at CERN. My trip to CERN was my first trip outside the country, and my first trip entirely alone. Marcello Bindi, JuanAn Garcia, Pierfrancesco Butti, Oldrich Kepka, and so many others made me feel at home. I miss my time there very much.

In the last six months, I've become well-acquainted and integrated with the Fermilab CMS team as well, specifically those working on hls4ml and Azure ML for particle physics. Thank you to all the team for opening your arms to UW, and especially to Nhan Tran, Javier Duarte, and Mia Liu for being so instrumental in supporting my work with Azure ML.

Finally, thank you to all those I know from outside an academic setting. Thank you to my parents Holly and Gary, who are, in my opinion, the best parents anyone could ask for, and without which I would likely have never gone to graduate school. Thank you to my girlfriend, Sam, who has somehow been by my side the whole time I've been a Master student and is my loudest cheerleader. Thank you to the rest of my family and friends, who make up the majority of the indirect support I receive, which, while maybe not helpful in deciding whether to instantiate a LRAM or a BRAM, still make all the difference in whether it's worth it at all.

Appendices

Appendix A: Emulator

Emulator code repository:
https://gitlab.cern.ch/dgsmith/rd53a_hardware_emulator/tree/master

Previous related theses:
Joseph Mayer: http://cds.cern.ch/record/2198312
Lev Kurilenko: http://cds.cern.ch/record/2631488

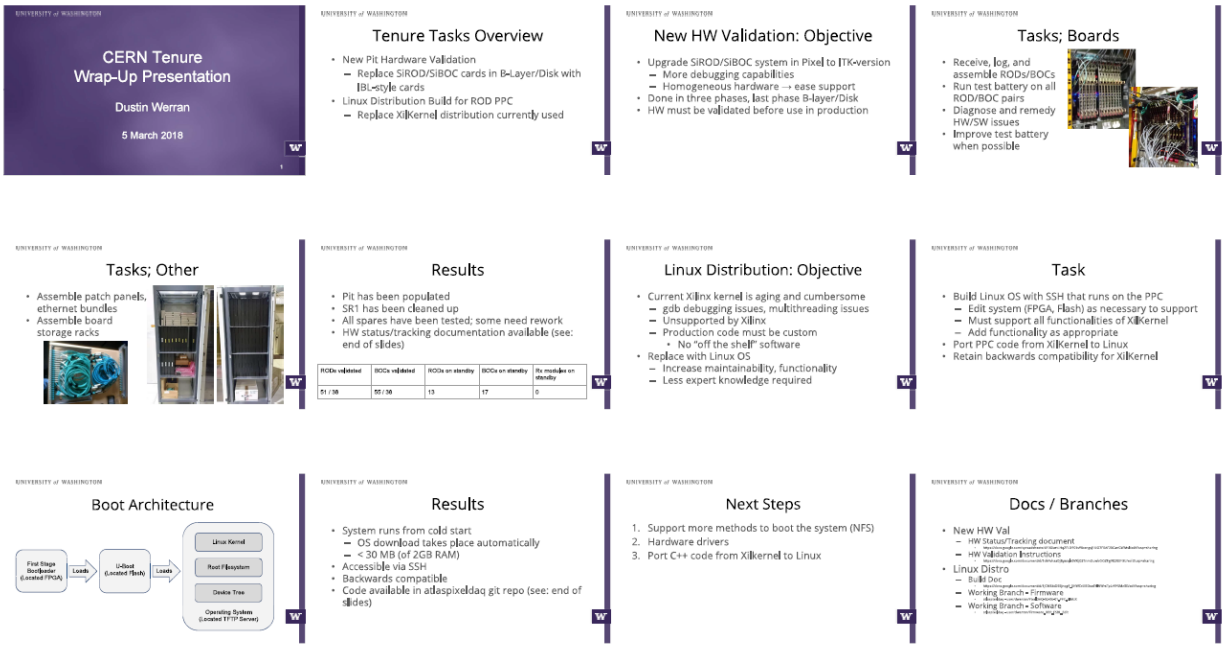
Appendix B: Linux development for the Pixel ROD

During my time at CERN, I worked on a project entirely separate from my work on the Emulator. There, I focused my development on a distribution of Linux targeting an embedded PowerPC on one of the FPGAs on the Pixel Readout Driver (ROD). It was developed as a replacement for the aging and insecure Xilinx-proprietary kernel currently used on that PowerPC. The Xilinx kernel has had support dropped for it since 2011, and at such time updates were frozen. Additionally, being proprietary, the Xilinx kernel is not supported by many standard utilities like SSH. Transitioning to a Linux-based distribution provides for consistent security updates and easier use of third party programs, as well as easier changes to the distribution binary via boot over TFTP.

Though it has not yet been deployed, there has been renewed interest in using it in the near future, before the ITk update. Currently, changes and updates are being done to bring the image up to specification.

Here is a presentation I gave at the end of my term of developing the distribution:
https://drive.google.com/file/d/1XGFsye1QHRjVPHoLbG-qUooFXGSaN4D6/view?usp=sharing

Here is the working instruction document for building and using the distribution. It is rather technical:
https://docs.google.com/document/d/1jCM5ktD55jnqg9_5HWCcUBOseEYINWrsTpLrVP5MzBE/edit?usp=sharing



Appendix C: FPGA-accelerated machine learning inference as a service for particle physics computing

Since November 2018, my work has focused on leveraging the Azure ML Brainwave service for accelerated machine learning inference targeting a Top tagging dataset. Though Top tagging is a contrived example, deep learning has seen huge interest in the area of offline tagging and in high energy physics generally as an alternative to conventional algorithms. However, a problem with deep neural networks is they require a lot of processing time to generate a result.

In response, specialized hardware has been developed focused on accelerating Deep Neural Networks (DNNs) to inference times that are acceptable for latency-bound operations like Top Tagging. One such technology being developed by Microsoft is called Brainwave and applies FPGAs to the task of machine learning acceleration.

Over the past few months, I have been working with a team at Fermi National Accelerator Laboratory (FNAL) to leverage the Brainwave service as described. Results so far have been positive, with round trip latency times of about 30 miliseconds and labeling accuracy up to 90%, though they are expected to improve with some simple changes to the training methodology.

A paper will be publicly released in the short term titled the same as this appendix section. Further details will be available there.