Three Generations of FPGA DAQ Development for the ATLAS Pixel Detector

Joseph A. Mayer II

A thesis
Submitted in partial fulfillment of the
Requirements for the degree of

Master of Science in Electrical Engineering

University of Washington
2016

Committee:

Scott Hauck

Shih-Chieh Hsu

Program Authorized to Offer Degree:

Department of Electrical Engineering

University of Washington

Abstract

Three Generations of FPGA DAQ Development for the ATLAS Pixel Detector

Joseph A. Mayer II

Chairs of the Supervisory Committee:
Professor Scott A. Hauck
Electrical Engineering

Assistant Professor Shih-Chieh Hsu
Physics

The Large Hadron Collider (LHC) at the European Center for Nuclear Research (CERN) tracks a schedule of long physics runs, followed by periods of inactivity known as Long Shutdowns (LS). During these LS phases both the LHC, and the experiments around its ring, undergo maintenance and upgrades. For the LHC these upgrades improve their ability to create data for physicists; the more data the LHC can create the more opportunities there are for rare events to appear that physicists will be interested in. The experiments upgrade so they can record the data and ensure the event won't be missed. Currently the LHC is in Run 2 having completed the first LS of three. This thesis focuses on the development of Field-Programmable Gate Array (FPGA)-based readout systems that span across three major tasks of the ATLAS Pixel data acquisition (DAQ) system. The evolution of Pixel DAQ's Readout Driver (ROD) card is presented. Starting from improvements made to the new Insertable B-Layer (IBL) ROD design, which was part of the LS1 upgrade; to upgrading the old RODs from Run 1 to help them run more efficiently in Run 2. It also includes the research and development of FPGA based DAQs and integrated circuit emulators for the ITk upgrade which will occur during LS3 in 2025.

Contents

Section 1: Introduction

Modern experimental particle physics seeks to find answers to questions like: Is the Standard Model complete or are there particles we don't yet know about, and what is Dark Matter? To start to discover the answers to these questions an experiment is needed that can produce large amounts of data at energies higher than have ever been probed by humans before. Thankfully such an experiment exists in the form of the Large Hadron Collider (LHC). The LHC is located at the European Center for Nuclear Research (CERN), and straddles of the borders of Switzerland and France just outside of Geneva, CH in Central Europe. CERN itself is a massive institution consisting of 174 contributing institutions from 38 different countries in the ATLAS collaboration alone. Such a large collection of scientists and engineers from a myriad of nations is necessary in order to make a machine as large as the LHC possible.

The LHC itself is a 27km ring that sits 100m below ground and consists of both superconducting magnets and accelerators to boost and control the speed of the particles around the ring. The LHC is currently achieving beam energies as high as 13TeV and luminosities of $2.2 \times 10^{34} cm^{-2} s^{-1}$ events [1]. The higher energies allow for the creation of more subatomic byproducts with each collision and the higher luminosities increase the number of collisions that occur per square centimeter every second. This results in more data for physicists to analyze. As Figure 1.1 shows the LHC is not a single monolithic circle, but several stages of loops of various sizes, each ramping up the energy of the beam on the way to the largest ring. Two particle beams are accelerated at nearly the speed of light in opposite directions and collide within the various detectors located around the primary ring. Figure 1.1 names these detectors and locates them on the ring. The different detectors serve distinct experimental purposes: the quite large ATLAS and CMS detectors are classified as general detectors, meaning essentially that they search for a wide range of physical phenomenon; the more specialized and relatively smaller ALICE and LHCb experiments that look for heavy ions and the relationship of matter versus antimatter respectively.

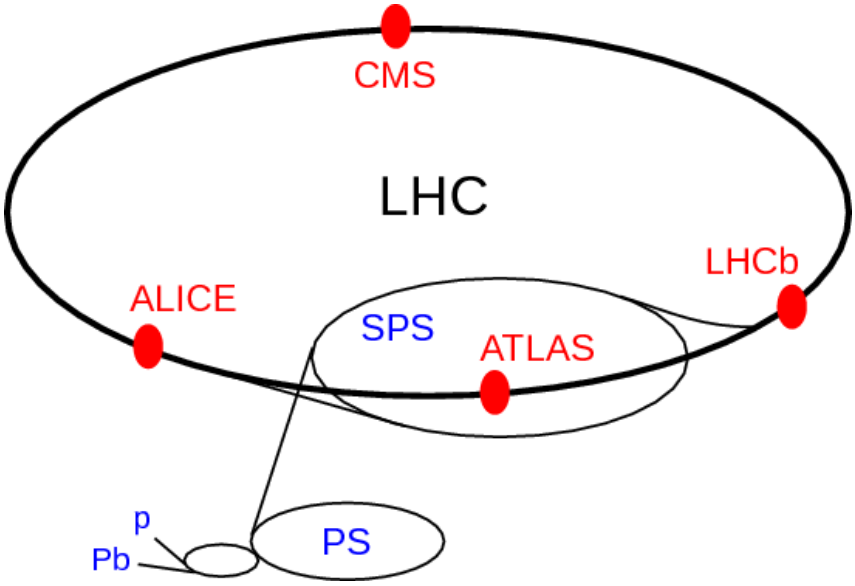


Figure 1.1: LHC Ring Topology [2]

ATLAS (which stands for A Toroidal LHC ApparatuS) is one of two general purpose detectors in the LHC, standing 25m high and weighing in at 7,000 tons and is shown in Figure 1.2. As a whole ATLAS is what is known as a 4π detector, meaning it has detector material completely surrounding the interaction point. Figure 1.2 enumerates the subdetectors of ATLAS that help achieve this structure: there are the tracker detectors which uses silicon sensors to record particle energies as they pass by, the Calorimeters which measure energy by absorbing it, the muon chambers which look to specifically measure the momentum of muons, and the large solenoid and toroidal magnets which allow for the measurement of particle momentum. The LHC beam pipe passes directly through the detector's center, colliding its beams every 25ns (an event know as a bunch crossing) causing energies and their corresponding particles to explode out in all directions. The aftermath of the collision event is then recorded by the various subdetectors. Physicists look at the tracks left behind as the particles traverse through the detector in order to search for new particles and understand phenomenon such as Dark Matter.

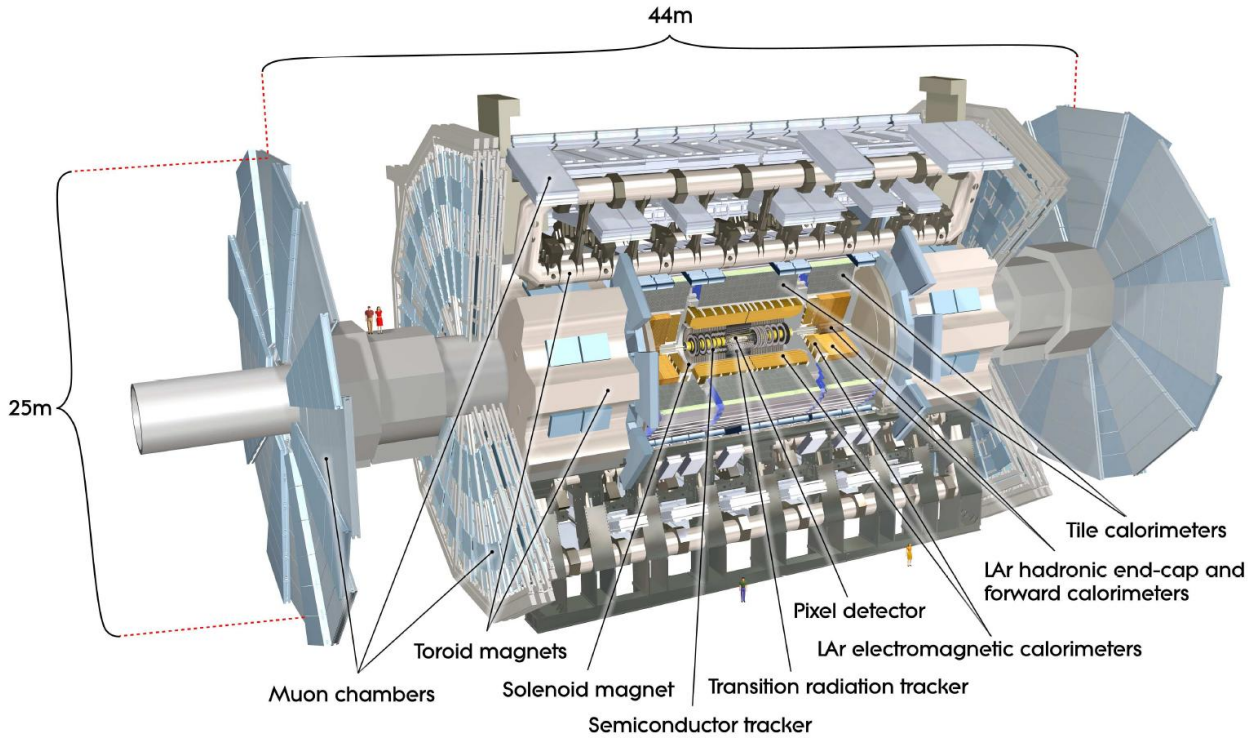


Figure 1.2: ATLAS and its Subdetectors [3]

The Pixel Detector, otherwise simply referred to as Pixel, is the innermost detector of the ATLAS Inner Tracker, and is therefore the closest to the beam interaction point. Pixel is concerned with catching high energy, quickly decaying particles and tracking their movements precisely as they cross the detector. In order to achieve this Pixel is made up of several layers equipped with large arrays of silicon sensors that surround the beam in a cylindrical fashion, as well as forward and backward endcap disk layers. The first three layers, along with the endcaps, can be seen in Figure 1.3, while Figure 1.4 shows the insertion of the new fourth layer. The size of Pixel with respect to ATLAS can be seen by comparing Figures 1.2 and 1.3. Cumulatively among all these layers Pixel has a total of 2192 Front-End modules and 92 million channels.

Table 1.1 enumerates the various layers of Pixel and the number of sensors they contain. The electronics on the actual detector are referred to as Front-End Electronics (FE). Pixel's FEs are composed of two parts: First the actual silicon sensors, a specially doped piece of silicon, which are excited by the electrical charge of the particles that cross over them, resulting in an electrical signal being created and its amplitude and duration recorded; second are the FE readout electronics which gather the electrical signals from the sensors at timing intervals with the granularity of a single bunch crossing and prepare them for off-detector readout by such actions as data packing and encoding.

Table 1.1: Enumeration of the Pixel Layers in order from innermost to outermost [1]

Layer Name	Staves	Modules	Pixels ($\times 10^6$)
Insertable B-Layer	14	448	12
B-Layer	22	286	13.2
Layer-1	38	494	22.8
Layer-2	52	676	31.2
Disks	48	288	13.2

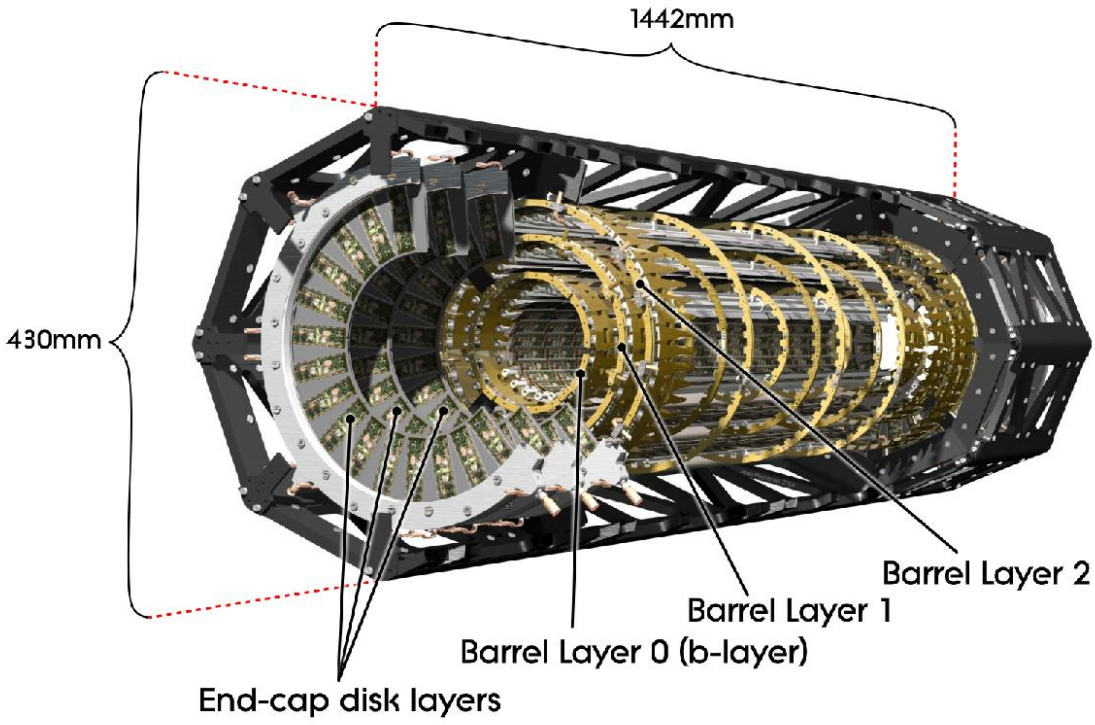


Figure 1.3: The Pixel Detector before the insertion of IBL [1]

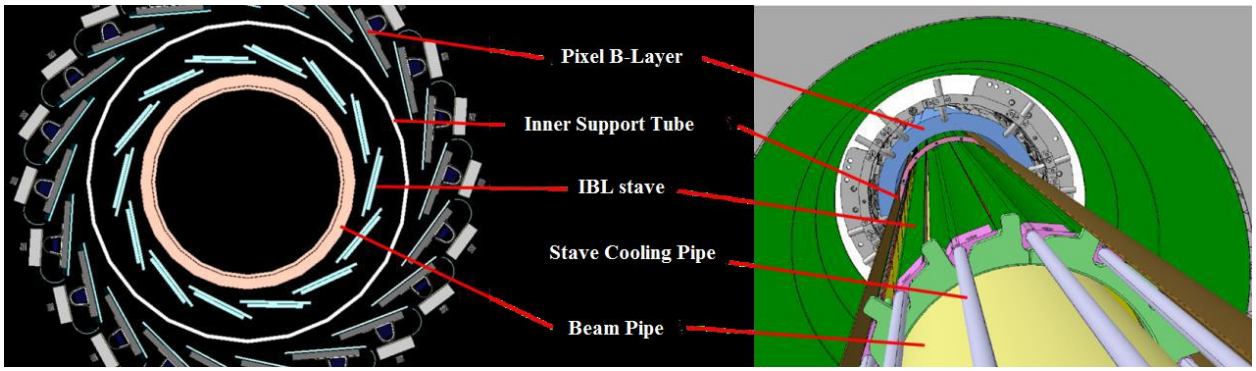


Figure 1.4: The IBL and Pixel B-Layers after IBL is inserted [4]

1.1 Introduction to the Data Acquisition System

ATLAS is a large and complex machine with many moving parts and subsystems working concurrently to make the detector work. Pixel is one subdetector with several subsystems that mirror the larger ATLAS systems, these include: DCS (Detector Control Systems), DQ (Data Quality), and DAQ (Data Acquisition). DCS is responsible for control of the electrical, optical, and cooling systems on the detector, ensuring that all FE modules are receiving the proper voltages and are operating with the correct temperatures and current draws. Data Acquisition is concerned with the coordinated readout of the data produced by the Front-End electronics after a collision event, and DQ is concerned with the quality of this readout data checking it for things like corruption and timing errors. All three of these systems work to create a fully operational high-energy particle detector.

The primary goal of the ATLAS-wide DAQ system is to coordinate the capture of a single event's occurrence across all subdetectors. The High-Level Trigger system (HLT) is responsible for managing this complicated timing. It does this by distributing a synchronizing pulse known as a Level-1 (or simply L1) trigger to all subdetector DAQs, which are responsible for dealing with the trigger timing latencies that occur in their individual DAQs. The frequency of the L1 trigger is important because it sets the data throughput speeds that all systems must be able to meet. If one system cannot then the entire detector must be slowed down, resulting in missed opportunities to collect valuable collision data. This signal is known as a busy and occurs when one subdetector has its event data pileup needing extra time to process its data. Currently the L1 Trigger rate has a maximum of 100kHz, giving 10us for data readout. Sometime after the trigger has been sent the HLT receives coarse event data back from the subdetectors via the Level-2 system and uses fast filtering algorithms to decide on the quality of the event and determine whether or not to accept the event and on which precise bunch crossing (BC) to send the next L1 trigger. Essentially ATLAS operates as a large camera, taking snapshots of the entire detector after collisions have occurred and capturing the energy and momentum left behind on the detector's sensors.

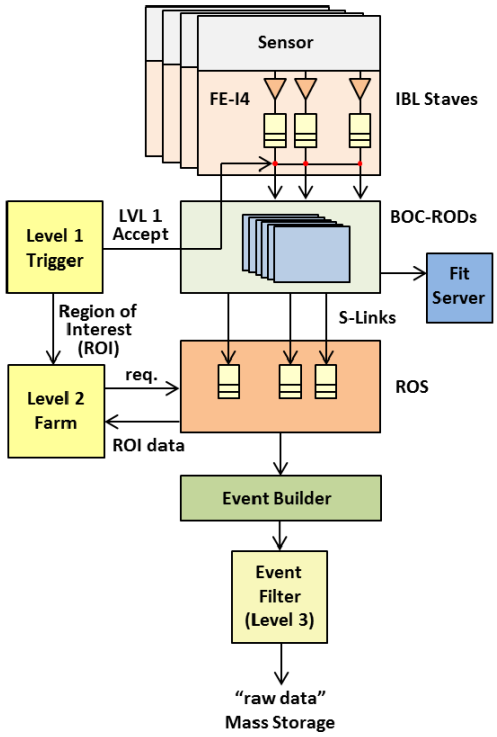


Figure 1.5: Coordination of ATLAS Trigger DAQ and IBL DAQ Systems [4]

Pixel's DAQ system is responsible for two main goals: distribute a trigger to all FEs, and readout the resulting data before the next trigger arrives in order to avoid event pileup. Here we will use the IBL layer as an example of Pixel DAQ; the other layers operate in a similar fashion with the only difference being the number of RODs and modules. IBL is made up of 14 staves with each stave playing host to 32 FEs, which for IBL are FEI4s so named because they are the fourth generation of Pixel Front-Ends. Each stave has a corresponding ROD (Readout Driver Card)-BOC (Back of Crate Card) pair which are responsible for its readout. All 14 ROD-BOC pairs are housed in a single VME crate which also contains the TIM (TTC Interface Manager). Figure 1.5 provides an example of this DAQ. We start in Figure 1.5 with the yellow blocks labeled Level 1 trigger. When a trigger is received from ATLAS's Timing and Trigger Control (TTC) System Pixel DAQ forwards it to the local crate TIM. The TIM then sends the trigger and corresponding event info to each ROD in the crate. The ROD then forwards the trigger down the Tx paths to the Front-Ends and stores the event information for future processing. Once an FE receives the trigger it begins to read out the data stored in its sensors and transfer the packaged data back to the ROD-BOC via the Rx data path. The ROD is then responsible for matching the raw data that was read out of the FE with the event information from the TIM. Finally the collated events are sent to the Level 2 computers, known as the Readout Subsystem (ROS), where they are examined and forwarded to both Level-3 permanent storage and back to ATLAS HLT.

1.2 Thesis Motivation and Outline

In this thesis we will discuss the work that was done over a period of just under two years. This work spanned several tasks of the ATLAS Pixel DAQ. Figure 1.6 shows the projected upgrade timeline for the LHC. The upgrade timeline follows a predictable pattern of the LHC increasing

its energy and luminosity and the experiments modifying their detectors in response. The primary reason for this cyclic behavior is data. As the LHC goes to higher luminosities (HL-LHC stands for high luminosity LHC) more and more collisions occur inside the experiments. This is ideal from a physics standpoint because experimental particle physicists are searching for rare phenomenon of nature. The more collisions that occur the more likely rare events, such as the detection of a Higgs Boson, are to be recorded. The drawback, though, is the amount of data created by such a high luminosity collider. For the data to be useful it must be readout using a DAQ system, and since the bandwidth of such a system is limited, so too is the amount of data it can process. This thesis looks at the development and modification of three major tasks of DAQ systems, allowing them to cope with the aforementioned problem.

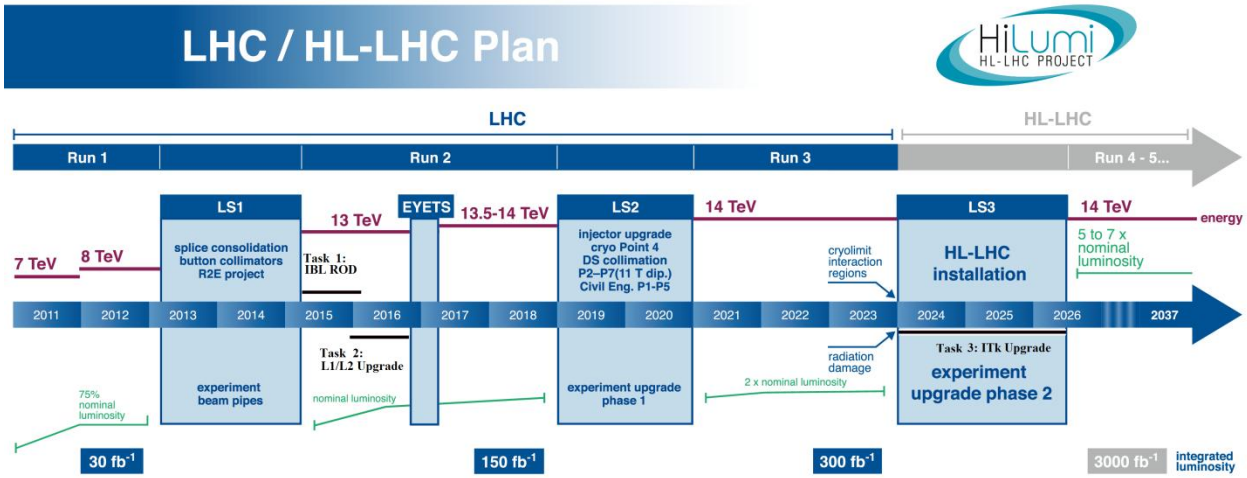


Figure 1.6: LHC Upgrade Timeline [1][5]

As the LHC began to increase both their energy and luminosity ATLAS took the step of placing an additional layer closer to the beam pipe. IBL was installed during the Long Shutdown 1 (LS1) as a response to the LHC's increase and as a result of the degrading performance of Pixel's original three layers. This required an enormous amount of effort which included the creation of a new DAQ system for IBL, which included a new ROD card. The IBL Technical Design Report [1] describes these reasons nicely and some will be enumerated here. First, is the effect of irradiation damage from the beam on the Pixel Detector and how it degrades Pixel's tracking performance. Radiation causes the electronics on Pixel's Front-Ends to fail; this renders the FE and all of the sensors it is responsible for useless. When this happens to a large number of modules on a given layer then the information about the collisions in that section of the detector are lost and tracking performance suffers. This is especially true of the B-Layer, which used to be the closest layer to the beam; IBL was inserted to recover some of the loss in tracking performance as well as to increase tracking precision due to its location close to the beam [1].

[1] fb stands for femtobarn which is 10^{-39} squared centimeters and is used to represent the number of events in a given surface area. Therefore $100fb^{-1}$ is equivalent to 100 events per femtobarn. Multiply by the number of femtobarns in the cross-sectional area to get the total number of events.

The second major reason for adding IBL is due to the increase in luminosity which, as we discussed previously, correlates to an increase in the amount of data created in Pixel. The large amounts of data created from high luminosity collisions is the cause of event pileup and high occupancy in Pixel's Layers [1]. This leads to readout inefficiencies in the detector and loss of data, which again means a degradation of tracking performance. The reasons for these inefficiencies are twofold: limited bandwidth in the Front-Ends, and limited bandwidth the DAQ. IBL confronts these problems by having lower occupancy which aids in maintaining tracking performance [1]. It also uses new FE technology (FEI4) and new DAQ technology as well (IBL ROD) both of which have increased bandwidth compared to the original Pixel Layers.

While the insertion of IBL is a welcomed addition to Pixel it is not the only tool that exists for mitigating the deterioration of the detector. The issues of irradiation damage and limited bandwidth in the Pixel Layers can also be solved with the upgrade of the DAQ system for Layers 1 and 2. During the course of Run 1 Layer 2 operated at a bandwidth of 40Mb/s while Layer one operated at 80Mb/s. However, both of these numbers are lower than the maximum achievable bandwidth of 160Mb/s, at which the B-Layer operates. Upgrading the readout of both Layers to the IBL ROD allowed for the exploitation of new technology on the card, specifically higher density FPGAs, which relieved some of the bandwidth strain that resulted from event pileup due to the increased energy and luminosity of the LHC. The combination of both IBL and the upgraded DAQ ensure that Pixel's tracking performance will be sustained throughout Run 2.

Though the previously mentioned upgrades were large tasks, taking many man hours to complete, they are small in comparison to the upcoming Inner Tracker (ITk) Upgrade. This upgrade will occur during the LHC LS3 in preparation for the HL-LHC, sometime around 2025. It will be a full revamp of the entire tracking system in ATLAS, from the detector and its DAQ to the triggering and power systems [6]. Many areas of Research and Development are needed in order to for the full project to be realized. A crucial are of focus is R&D for the Front-End Electronics as well as the DAQ readout system. The data based motivations of previous upgrades carry over into ITk with the addition of and an increase in the triggering frequency of the detector. Because the HL-LHC will create more data in the detector the FEs will need to be triggered at a higher rate to avoid event pileup. This places extra strain on the bandwidth capabilities of both the FEs and the DAQ. Research and Development must be done in order to find solutions to these and other problems faced by ITk.

In this thesis we will discuss the work that was done over a period of just under two years, spanning several tasks on the ATLAS Pixel DAQ. This thesis will start in Section 2 with DAQ development for the IBL; which was installed during Long-Shutdown 1 (LS1) and was part of the Pixel Detector's upgrade for LHC's Run 2. Next in Section 3 we will discuss the upgrade of Layers 1 and 2 of Pixel; older layers used in Run 1 whose DAQ hardware and firmware needed to be upgraded in order to cope with the increased demands of the LHC and ATLAS. Then in Section 4 we will move the development an integrated circuit FPGA emulator and next-generation DAQ for the ITk Upgrade. This will occur during the LHC LS3 in preparation for the High-Luminosity (HL) LHC, sometime around 2025. Finally, in Section 5 we will conclude with a look at what work remains to be done for moving forward with the ITk upgrade.

Section 2: The Readout Driver Card for the Insertable B-Layer

The Readout Driver Card (ROD) is for forming a Pixel event out of raw FE data and ATLAS event information making it the central piece of DAQ operation. These events created by the ROD will later be used by physicists to search for particles, dark matter, etc. For the RODs of the Insertable B-Layer (IBL) this importance is especially true due to IBL's location, only 3.3cm from the collision point. This means that IBL captures large amounts of data in a short period of time, putting extra pressure on the readout system. To cope with these demands the IBL ROD uses multiple FPGAs and a spatial architecture to handle data from many FEI4 modules in parallel. It allows for clock speeds up to 80MHz, double the achievable speed of the other 3-Layer's readout systems.

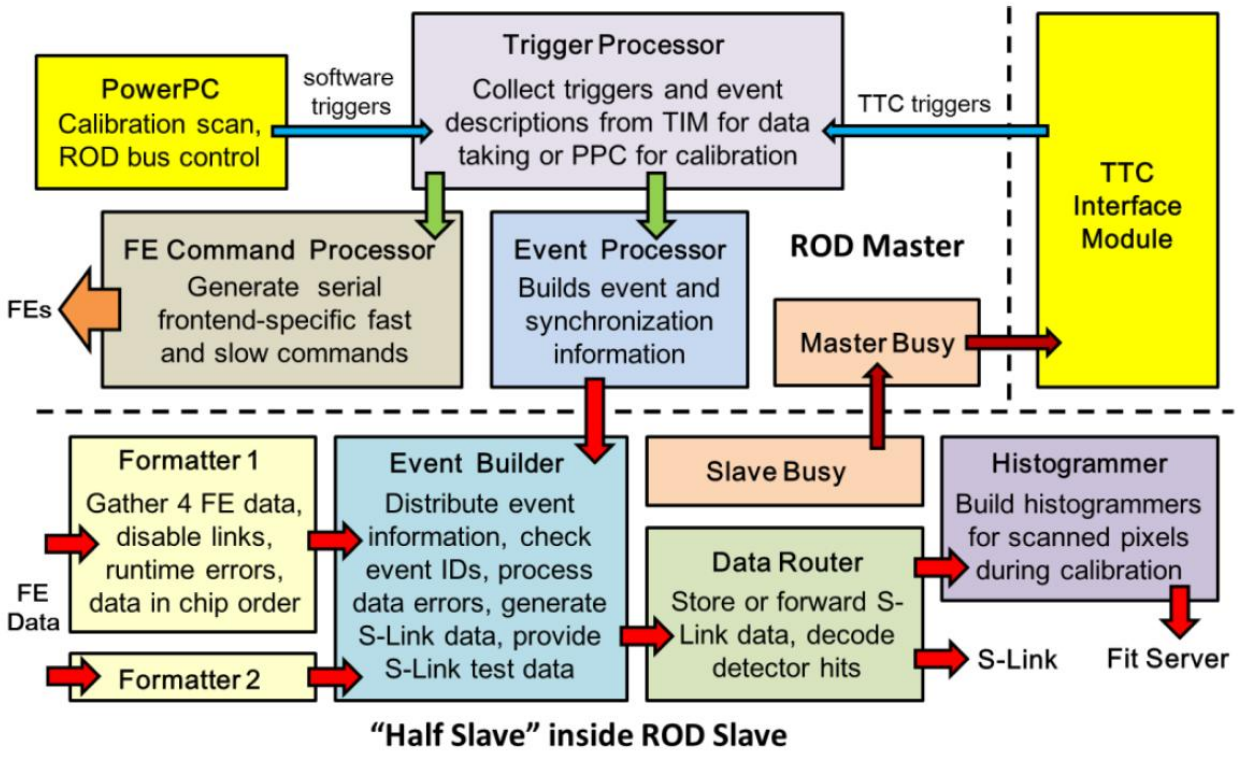


Figure 2.1: ROD Firmware control and data processing flow [4]

2.1: ROD System Architecture

The ROD itself is a large PCB card composed of four FPGAs, a DSP, SRAM, and a JTAG interface. It occupies a single slot inside a VME crate. The VME crate provides power to the ROD as well as allowing it to communicate with the BOC over a common backplane. The four FPGAs of the ROD facilitate all operations that occur on the board and are broken down into one Master, one PRM, and two Slave FPGAs. The Master FPGA is a Xilinx Virtex5-FX70T which has an embedded PowerPC processor and is in charge of all control operations. Figure 2.1 shows these various operations, which include: receiving triggers from the TIM, generating commands for the FEs, reporting busy to the TTC, and sending event info and action commands to the Slave FPGAs. The PRM (Program Reset Manager) FPGA is responsible for handling the reset and

bitstream programming of the Master and Slave. The two Slave FPGAs are Xilinx Spartan6-LX150 and are the datapath FPGAs in charge of raw data processing and event forming, as well as histogramming for calibration, illustrated in Figure 2.1. In the following sections we will discuss the main components of the Slave in more detail. Then we will move on to discuss improvements made and problems solved in preparation for and during Run 2 DAQ operation. For more detailed documentation on the roles of the Master and PRM please see [6].

2.2: ROD Slave Datapath

The ROD Slave datapath is composed of three main processing modules: the Formatter, Event Format Builder (EFB), and Router connected in respective order by variously sized FIFOs. The slave uses both a spatial and stream processing architecture passing data between its concurrently processing modules using both standard First-Word Fall-Through (FWFT) and Clock Domain Crossing (CDC) FIFOs. The datapath uses valid signals for forward processing, and FIFO full signals for backwards flow control. In data taking each Slave is responsible for processing the data from 16 Front-Ends and transferring this info to the ROS via two SLINK connections to the BOC.

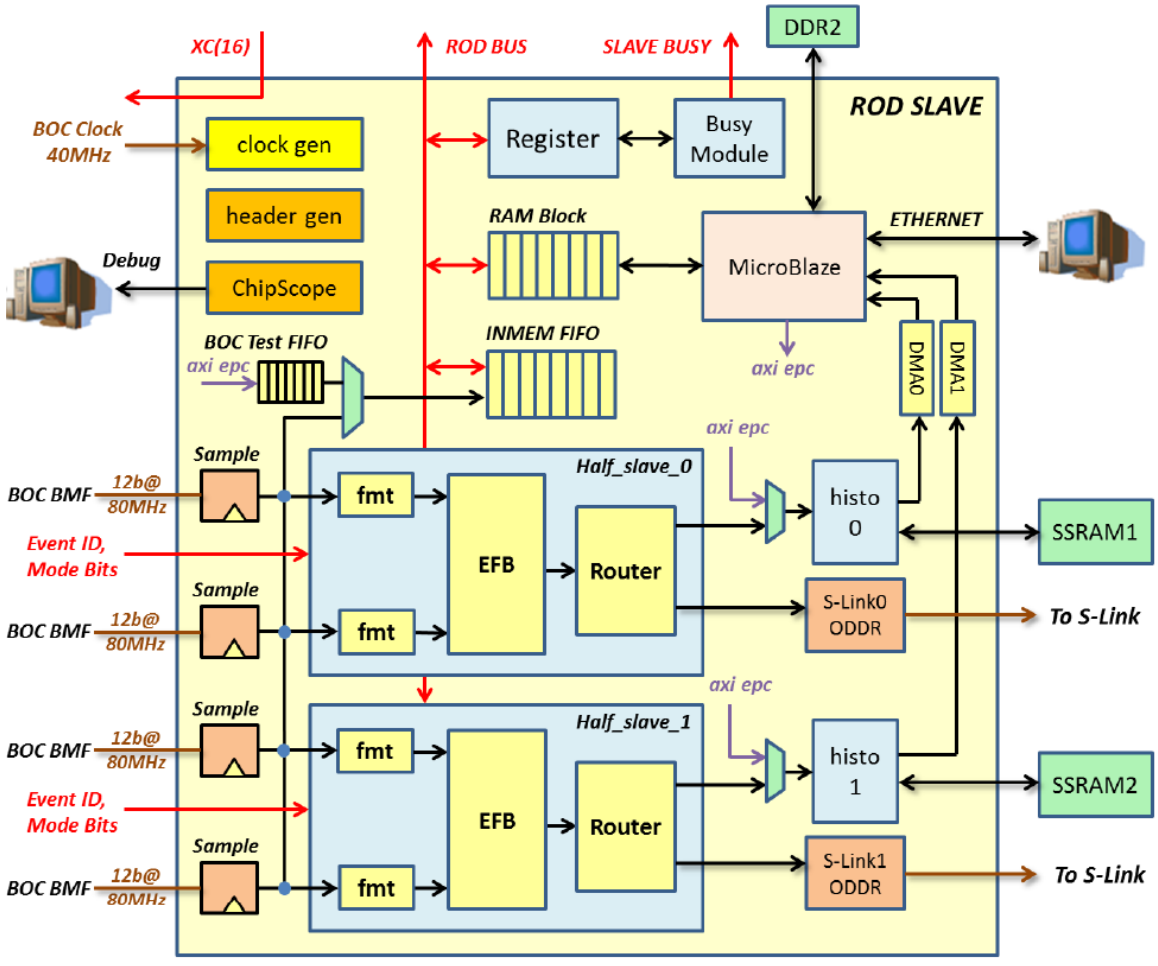


Figure 2.2: ROD Slave Diagram [4]

Figure 2.2 above shows the full block diagram for one ROD Slave FPGA. One Front-End on an IBL stave maps to a single link in a ROD formatter. There are 4 links per Formatter module (represented by BOC BMF in Figure 2.2), 2 Formatters per EFB, and two EFB/Router pairs per Slave, also known as a "half slave". Thus, one Spartan6 FPGA is responsible for data from 16 FEI4s, and a ROD for 32. These primary blocks are also supported by many secondary blocks also show in Figure 2.2. There is the Master/Slave "ROD" communication bus that is used to read and write from the large register set, both programmable and read only, that exists in the ROD. The bus and register file are used as the Hardware-Software interface in the Slave where the C++ code written for the PowerPC can be used to program and read the state of the Slave FPGAs. A MicroBlaze soft-core CPU is also present in the Slaves, where it is used to aid in the histogramming process for calibration. The busy reporting block alerts the Master of event pileup in the Slave and the IMEM FIFO acts like trace storage which aids in debugging. Finally there are the Integrated Logic Analyzer (ILA) cores for dynamic debugging via ChipScope available in the Spartans.

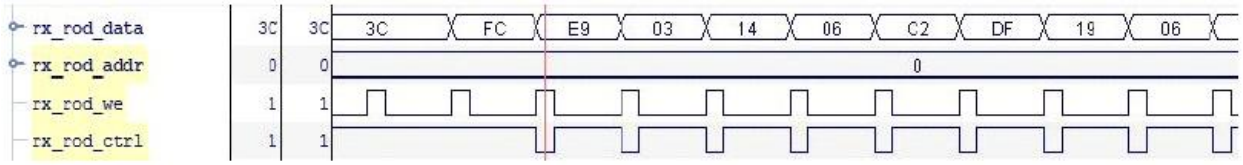


Figure 2.3: ROD-BOC bus transmitting a header packet [7]

The first major module in the Slave's datapath is the Formatter. Figure 2.4 shows the full layout of a single Formatter module from the demultiplexed bus, the link encoder and their corresponding FIFOs to the readout controller. The formatter is connected to the BOC via a custom parallel bus that travels over the backplane. The 12-bit bus, seen in Figure 2.4, includes 2 bits for address, 8 bits for data, and 1 bit each for write enable and control. The data from the BOC is time multiplexed on the bus and the address bits are used in the ROD Formatter in order to transfer data to the correct link. As the red line in Figure 2.3 shows, a byte of data is considered valid when the write enable signal is high and the control signal pulsed low. After the correct link destination has been chosen, and the data said to be valid, a link encoder module is used to format the data. It starts by forming 24-bit words from three consecutive 8-bit data transfers. A complete data transfer to link number 0 is shown in Figure 2.3.

After correct decoding of the bus there are four unique data packets that the link encoder submodule will create based upon the 24-bit data word received, they are: Data Header, Data Trailer, Data Hit, and Service Record. The bit definitions for each can be seen in Table 1 in Appendix A. Headers are the first item decoded from the data stream identify the Level-1 trigger associated with the incoming data. Hits are then formed as consecutive three 8-bit sequences that occur in between a header and a trailer, with the first two bytes representing row and column address of the sensor on FEI and the third being the Time Over Threshold (ToT) data (the actual information from the sensor). Finally a trailer arrives to close out the L1 trigger event. Service records are a special set of information sent from the FEI4 and alert the user of bit flips, overflows, etc. They can appear at any point and are packaged in the data stream like any other data word.

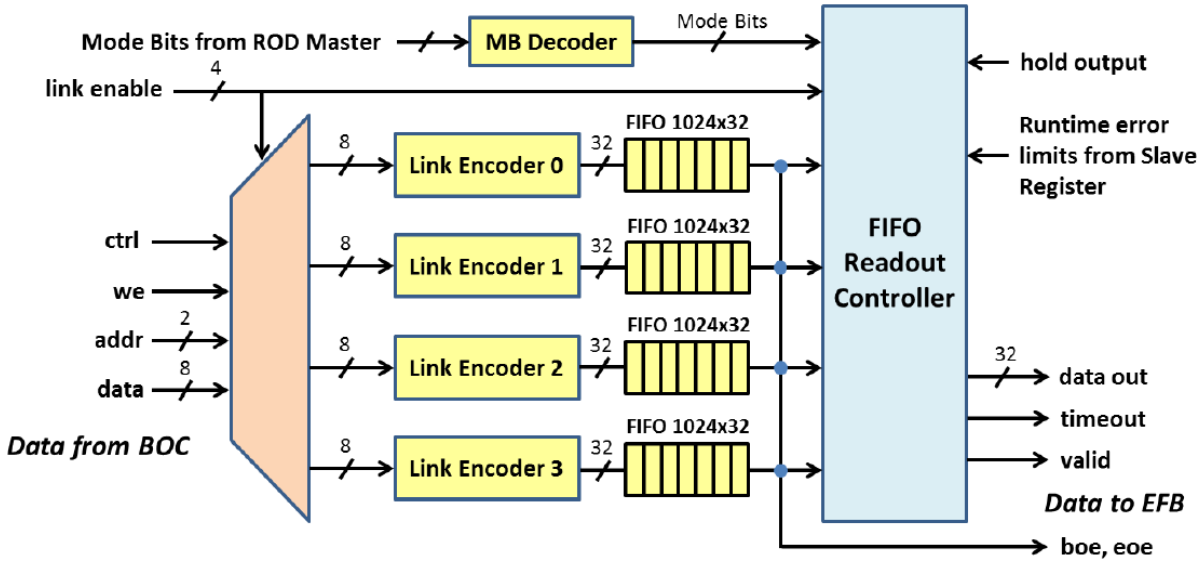


Figure 2.4: Block diagram of the Formatter module [4]

Once the data word is complete it is checked for both data integrity and flow control errors in the links and then stored in a corresponding link FIFO. Cumulatively these data make up what are referred to as data frames, with the header and trailer defining the edges of the frame. A data frame also has physical significance in that one frame corresponds to one bunch crossing, with the number of frames read out per L1 trigger being the number of BCs that data is taken from. A large state machine known as the FIFO Readout Controller (FRC) is then used to readout the link FIFOs in numerical order and forward the data to the EFB. While generally simple in its readout, the FRC does have the ability to check the number of header/trailer pairs, it sends ensuring that the correct number of data frames are sent to the EFB.

The EFB is the critical junction in the ROD Slave datapath because it is the module in which the raw FE data and ATLAS event information collide and are formed into a corresponding physics event. ATLAS event info is received from the Master FPGA over a special-purpose bus used only for communicating with the EFB. The received information is then decoded and stored into an event buffer, by the Event Data Decoder submodule, shown in Figure 2.5, where it waits to be read and attached to raw data. Once event data is present the EFB notifies the Formatters' FRC to send the Front-End data it is currently storing. An FSM in the EFB is used to synchronize the process of requesting data from the Formatter. It works with the FRC to ensure the correct numbers of data frames are sent.

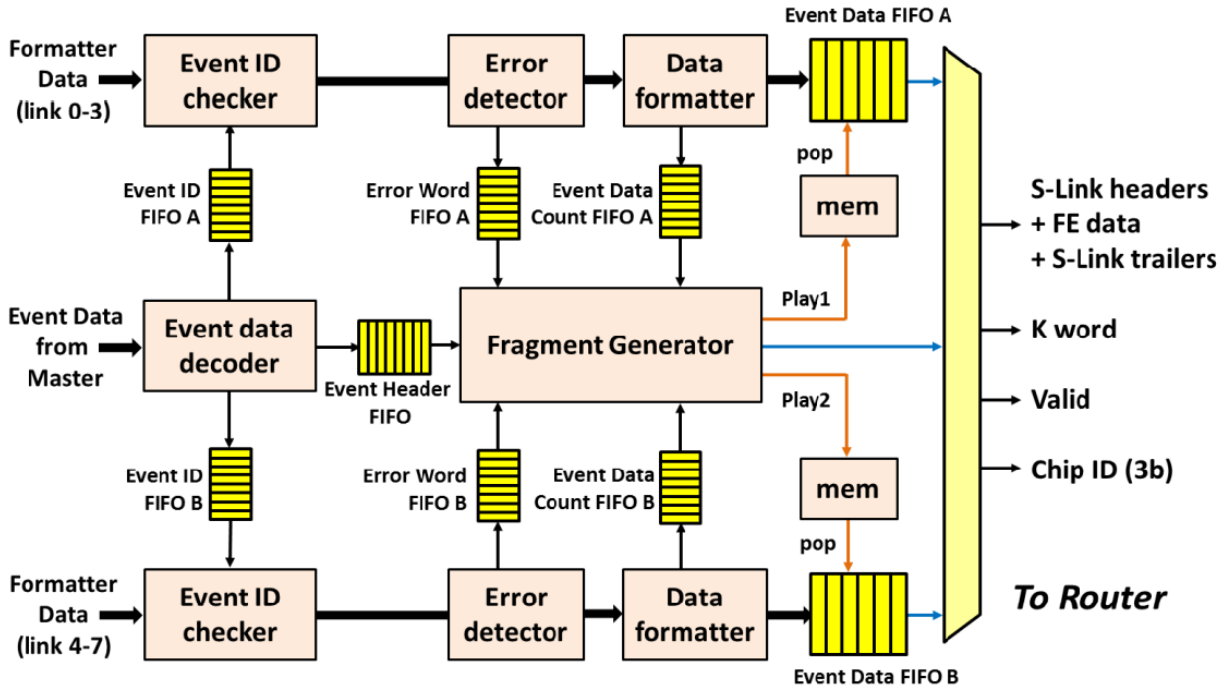


Figure 2.5: Block diagram of the Event Format Builder [4]

Since a single EFB is responsible for data from two Formatters (8 FEs) it contains two parallel datapaths for processing the data from each formatter simultaneously, shown as two replicated paths in Figure 2.5. When data is received from the Formatter it is first passed through the Event ID checker, where the BCID and L1ID information stored in the data headers is compared against the event information and an error is reported if a mismatch occurs. Next the data is sent to the Error Detector where the runtime errors that have been marked are logged in order to create an error summary which is included in the SLINK Trailer. Outputs of the Error Detector are passed to the Data Formatter which counts the number of packets it sees and stores them in a FIFO where they await further operation. Finally, the two paths are then merged using another FSM known as the Fragment Generator. The fragment generator packs the information received from the Formatters between an SLINK event header and trailer, which is created from event information and the Error Detector Block. The FIFOs storing the data from the two parallel paths are read out again in numerical order with Formatter 0 going first followed by Formatter 1.

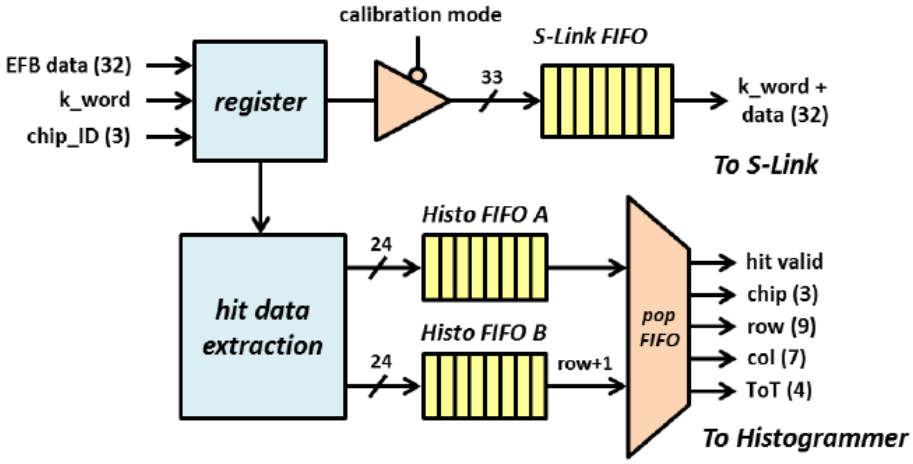


Figure 2.6: Block diagram of Router Module [4]

The final module of the ROD slave datapath is the Router. The Router has two different modes: calibration and data taking (this split can be observed in Figure 2.6). In data taking mode the data is simply transferred from one buffer to another with the second buffer being a CDC FIFO labeled in Figure 2.6 as the S-LINK FIFO. This FIFO is written to at 80MHz and transferring data back to the BOC at 40MHz. Flow control is a huge issue here because loss of data words, headers and trailers in particular, could corrupt the whole packet. To combat this, a backpressure signal is created that is the logical OR of three signals: SLINK down, SLINK off, and BOC FIFO full. If high no data is sent to the BOC and the backpressure propagates to the other modules risking pileup in the entire datapath. In calibration mode the data from the EFB is sliced up with the headers and trailers being thrown away and the link numbers, row, column, and ToT values being forwarded to the MicroBlaze. The data is stored in two separate FIFOs, which appear in Figure 2.6 as Histo FIFO A and B. A is for the first four bits of ToT (ToT 1) and B is for the last four bits (TOT 2). The MicroBlaze then does binning of the ToT values from each link and creates histograms that are used in calibrating the sensitivity of the readout sensors.

2.3: Enhancements of the ROD Slave Datapath

For the LHC Run 2 several modifications were made to the ROD firmware. The major changes that took place in the firmware are enumerated here with the purpose of providing clarity to the process of actively modifying DAQ firmware, as well as highlighting some key features of the IBL ROD firmware. These changes were influence by both dynamically occurring issues as well as lessons learned from the use of the original Pixel RODs in Run 1.

2.3.1: Enhanced Error Detection and Reporting

The first major improvement to the ROD firmware was the addition of runtime error detection and reporting in the link encoder block of the Formatter module. The upstream data quality monitoring software depends heavily on this information to know the correctness of the received data, and whether or not it can be used. Reported error data is also used in active DAQ operations as a feedback mechanism giving information about the status of the detector and its data taking. The primary goal of this enhancement was not only to report the errors but also to

enforce the frame packet structure of the data, that being Header → Data → Trailer, in order to prevent frame fragmentation. This is important because the following processing modules depend upon a correct frame structure to operate; a corrupt frame causes the downstream components to either produce a bad result or stall completely. All runtime error checks occur after the 24-bit data word has been assembled.

The task of error detection and reporting was divided into three parts: detection, reporting, and that (if possible) correction. Reporting of the errors typically takes two forms; the first is to mark an error is present in the frame trailer (these are then accumulated in the event trailer), and the second is to write to one of the Slave's registers, both as a single bit to mark the presence of the error and as a counter of the total number errors. In addition there are also three classes of runtime errors: corrupt data, timeout, and flood. A corrupted data error occurs if the bit fields of the 24-bit data word are out of bounds or incorrect for that given word type. It is also considered corrupted data if an unexpected data word occurs. Timeout errors are errors in which a needed data word, most likely the trailer, does not arrive in an allotted amount of time. Timeouts prevent the system from getting stuck on waiting for a data word that may never come, which will cause event pileup. Finally there is the flood class of runtime errors which occurs when too many of one data word type is sent continuously from the Front-End, and risks overrunning the data throughput capacity of the system.

The corresponding bit fields for the marked errors can be found in Appendix A. The descriptions of the errors are:

- **Readout Timeout:** Occurs if the FEI4 fails to produce all of its expected frames, or any data at all, after a programmable amount of time. The value of this timeout is programmable from software and is set by default to just over the maximum L1 trigger rate of 10us. If the timeout does occur pseudo-frames are generated and marked with the suffix 0xBAD.

- **Trailer Timeout:** Occurs when the trailer character 0xBC is never received by the link encoder. As a result the trailer error flag is set and a pseudo-trailer is generated by the link encoder after a programmable amount of time, with a current value of 1us. The data format of the pseudo-trailer is identical to that of a normal trailer, with the exception of the error flag being set.

- **Header-Trailer Limit:** Allows for a cap to be placed on the number of hits accepted from the FEI4 for a given frame. If the Formatter is currently receiving an event when the condition occurs the encoder will stop writing data to the FIFO until a trailer is detected and stored in the FIFO with the corresponding error bit set. The limit is again programmable, and is currently set to 840 (the maximum number of hits that can occur during a calibration of the Front-End).

- **Header Error:** When the first 24-bit word received by the link encoder does not contain the correct 8-bit MSB header qualifier 0xE9. This could signify sampling errors from the BOC. The error is marked with the suffix 0xBAD written in a pseudo-header which is

generated and written into the link FIFO, allowing data to continue to be taken. The corresponding error bit is also set in the frame's trailer.

- **Row/Column Error:** Another case of corrupted data, which is due to row and column values of a given hit being out of bounds of what is physically possible in the FEI4. This corresponds to a row value greater than 336 or a column value greater than 80. Upon occurrence the error flag is set and the data is passed to the FIFO. The data is passed rather than dumped so that the incorrect values can be investigated later and a possible source discovered, and their relative significance.

Along with these five tests, additional mechanisms were put in place to ensure that frame fragmentation was not allowed to occur and that the link encoder was never flooded with one specific data word. However, these types were not reported because of a lack of bits available in the trailer.

2.3.2: Enhanced Frame Handling

The second ROD firmware modification was the result of unexpected behavior from the FEI4 that was discovered during ATLAS data taking. Over the course of numerous LHC runs it was observed (thanks to the error reporting and detection techniques discussed earlier) that a significant number of IBL events had a myriad of errors (most notably L1ID and BCID mismatches) meaning that the data produced could not be used. After investigation of the offline data packets, and probing of the raw data coming into the ROD via Chipscope, it was revealed that the FEI4 was inserting its idle character (0x3C) unexpectedly between Hit packets that belonged to the same frame, or bunch crossing. This leads to a snowball effect inside the link encoder submodule. It starts with the link encoder misinterpreting the existence of a trailer character in the data stream causing the event to be closed by mistake. Now the next data word will be attached to the incorrect event, and so on until a reset occurs.

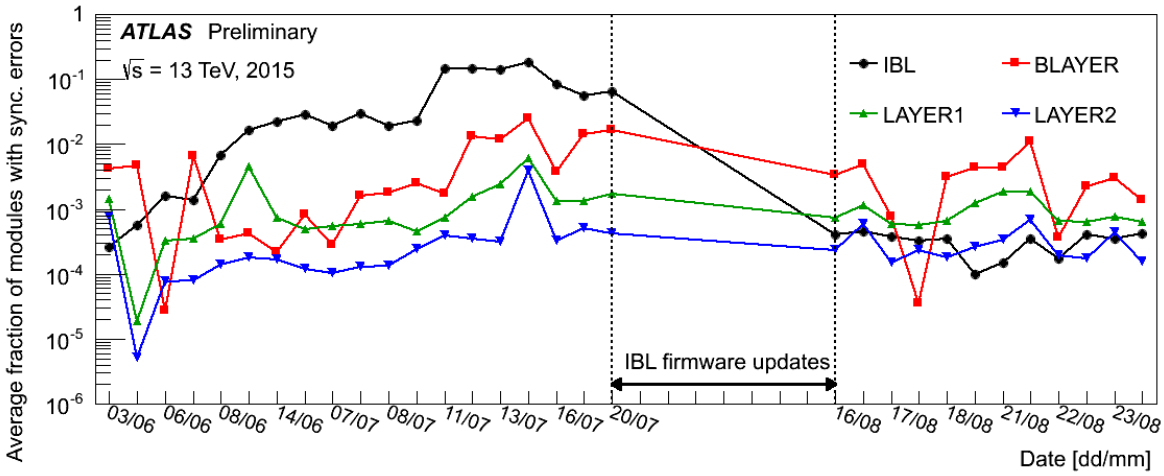


Figure 2.7: Graph showing the decrease in IBL desynchronization as a result of upgrades to the IBL firmware [8]

This was a rare occurrence for the FEI4, but if it happened even once during a single run all subsequent data taken during the run was forfeit, until the L1ID was reset in the FEI4 via an ECR. To prevent this cascading effect and loss of data the logic used by the link encoder to decode the incoming data needed to be modified. The first step was to work with the FEI4 designers to ensure the functionality of the FEI4 readout and possible data sequencing was completely understood. After this was done the link encoder code was modified to reflect this new understanding; this included changing how start and end of frame characters were interpreted. As a result this created the issue of how many 0x3C idles could be inserted before another packet was expected. The concern here being that waiting for too many idles before expecting a trailer would cause the system to stall and data overruns to occur. Through experimentation and trial and error the value of 3 to 5 idles was deemed appropriate, and a counter was used to terminate the frame. This was separate from the trailer timeout because it gave the logical indication of when to expect the trailer 0xBC, as opposed to just an end of frame character 0xBC. Figure 2.7 shows the slow rise in the number of synchronization errors and then a sharp decline over the course of a few runs. When the link encoder changes were finally integrated into the ROD firmware the number of errors in IBL data taking was seen to reduce drastically, by two orders of magnitude.

2.3.3: Enhanced FSM Synchronization

A FSM for synchronizing the decoding of event information in the EFB and data readout of the FRC was the third major change to the ROD firmware, and it had far reaching consequences. This update allowed for full and correct calibration of IBL to be possible, as well as also driving down the number of L1ID and BCID mismatches. The primary motivation for the addition of this synchronizing FSM was the need for the raw data and event info to match (the main purpose of the EFB and the ROD itself). The full FSM that was created can be seen in Figure 2.8. This figure shows the communication steps that need to take place between the EFB and the FRC in order to ensure the FE raw data is matched with the correct event information.

The first state is entered upon reset and is exited if at least one of the links in the two Formatters connected to the EFB is enabled, in all subsequent states if all the links are found to be disabled then the FSM returns to the reset state. From idle the next state is moved to if either an event is present or the first Formatter as a whole is disabled. The output of this state is a signal to the FRC prompting it to begin sending data. The same logic and output is applied to the Wait Count FIFO2 state, with the difference being it communicates with the second Formatter's FRC. The GenWait state signals the EFB's Fragment Generator that it can begin assembling the SLINK Header and expecting data from the Formatters. This state waits for an acknowledgement from the Fragment Generator confirming the process has begun. Finally in the WaitDone state the FSM waits for the Fragment Generator to say it has finished processing the current event and the FSM is free to start the readout process over again.

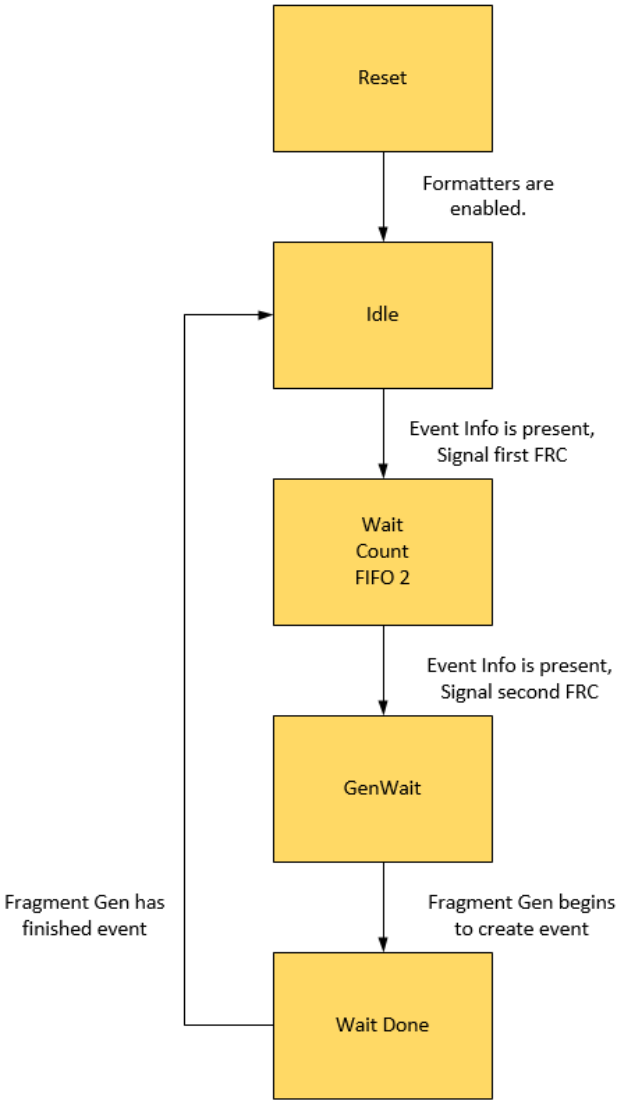


Figure 2.8: FSM used for synchronous EFB and FRC event readout

These changes had to be integrated during an active run and therefore incurred heavy testing using Chipscope and the FEI4 emulator on the BOC. In the end they resulted in both the ability to due complete calibration of the Front-End as well as reduced errors in IBL data taking.

Section 3: Upgrade of Layer-1/Layer-2 RODs

As previously shown in Table 1.1 Pixel is a 4-Layer detector. Layer-1 (L1), Layer-2 (L2), B-Layer, and the Endcap-Disks were used in Run 1, and the IBL was added for Run 2. At the start of Run 2 in March 2015 the three outermost layers of Pixel still used their original DAQ readout systems developed before the beginning of Run 1 in 2009. The original Pixel readout system mirrors the IBL DAQ described in Section 1.2 stages and functionality. A few key differences are the construction of the Front-End electronics and the FPGA architecture of the original ROD known as SiROD. The FEs for the outer three layers are composed of 16 FEI3s connected to an IC known as the Module Controller Chip (MCC). Figure 3.1 shows an original Pixel chip with the sensors and FEs connected and its relative size. The FEI3 contains both the Pixel sensors and a small amount of integrated digital electronics capable of reading data from the pixel columns and transferring it to the MCC. The MCC is then responsible for controlling link communication bandwidth by arbitrating which FEI3's data to send. The MCC is also in charge of encoding the data in its final packetized format. (The FEI4 does all of this work as a single monolithic IC bump bonded to the sensor). The readout driver card for the original pixel system was designed and used for not only the Pixel Layer but also the SCT (SemiConductor Tracker) Layer as well, another subdetector in the ATLAS Inner Tracker. SiROD stands for Silicon ROD and is composed of multiple FPGAs and DSPs on a single card. Because the PCB for SiROD was designed and developed back in 1999 it used FPGAs with significantly fewer LUTs compared to contemporary FPGAs. This caused the need to split the major aspects of the ROD datapath (Formatter, EFB, and Router) into separate FPGAs and then connect them through traces on the PCB. As a result the datapath processing was slowed down, due to slow clock speeds and significant transfer overheads; leading to events piling up which caused ATLAS to go busy. It also made SiROD more difficult to debug. The diagram in Figure 3.2 shows the logical connections of the readout chain in more detail and helps to visualize the hierarchy of 16 FEI3s communicating with a single MCC which is then responsible for a single communication link on a ROD.

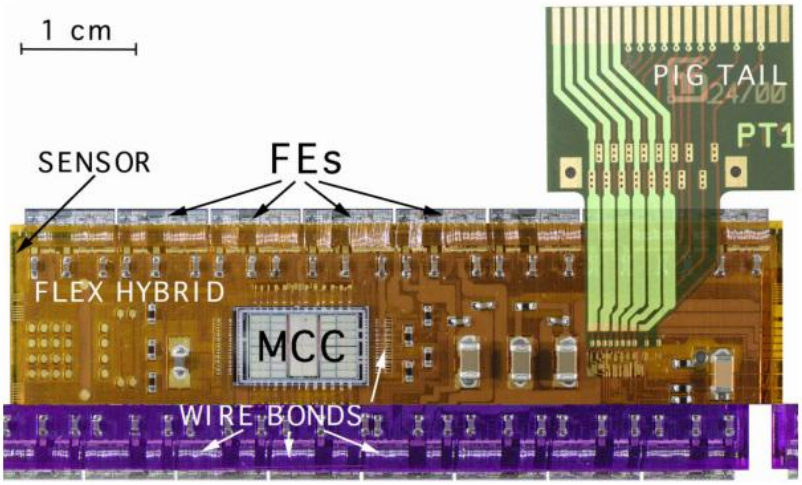


Figure 3.1: ATLAS Pixel Module for the outer three Layers [9]

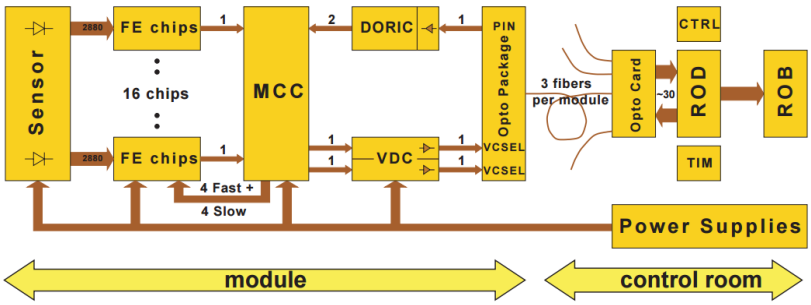


Figure 3.2: Original DAQ system architecture of the Pixel Detector [9]

A working group was assembled in order to upgrade the DAQ for Layers 1 and 2 using both the IBL ROD and BOC cards. Since the B-Layer was already operating at the maximum readout speed of the MCC it was not included in the upgrade. The three primary motivators for the upgrade were: 1) Higher bandwidth requirements due to increased luminosities and higher trigger rates. 2) Increased failure of modules due to radiation and other damage that require extensive monitoring. 3) The desire for a homogeneous and integrated Pixel readout system across the subdetector Layers. Each of these goals was able to be met by leveraging the superior FPGA technology on the newly created IBL ROD. The Table in Appendix B shows the expected link occupancy for the Pixel Layers in Run 2. It is clear from the table that if the link to ROD bandwidth was not improved data would be lost and inefficiency would suffer. This first goal was met due to the datapath speed of the ROD increasing from 40MHz on SiROD to 80MHz on IBL, which allowed the readout speed of the MCC to increase. The final two goals were met because of the increase of available resources in the later generation FPGAs used in IBL compared to the older one used in the SiROD. The increase in LUT resources allowed for the full datapaths of IBL and L1/L2 to exist in the same FPGA, along with additional space to add more sophisticated monitoring tools for the decaying layers.

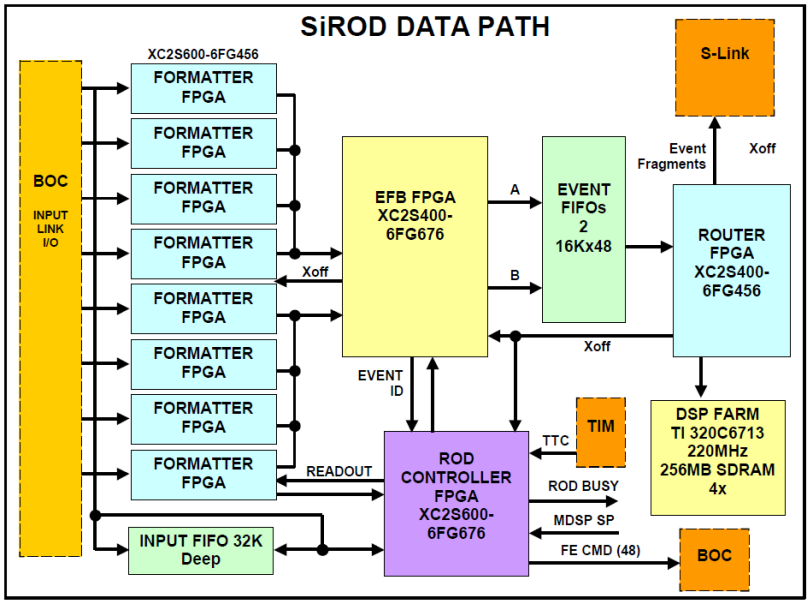


Figure 3.3: Datapath of the original ROD used in Pixel [10]

3.1: Datapath Module Modifications

The current IBL ROD Slave datapath required several modifications in order to handle L1/L2 readout. It needed to be compatible with both the readout procedure of the MCC and the data format of the FEI3, as well as Level-2 communication and DQ software processing. This necessitated careful alteration of the three major functional blocks of the Slave datapath. Goals of the datapath integration included: minimal modifications to current firmware to allow for firmware consistency for IBL and L1/L2 via a single source code base, reuse of FPGA resources, and consistency with the original SiROD's programming model so higher level software could also be reused. Because the IBL ROD was a derivative of the original SiROD these goals were reasonable to meet and the integration of the L1/L2 firmware datapath into the IBL firmware datapath was successful. A diagram of the original SiROD datapath is shown in Figure 3.3 and its similarities to IBL are immediately evident. The issue of multiple FGPAs is also clear. Currently at CERN the new RODs for Layer-2 have been installed and their official testing and integration is still ongoing. The Layer-1 upgrade is expected to be installed sometime in Summer 2016.

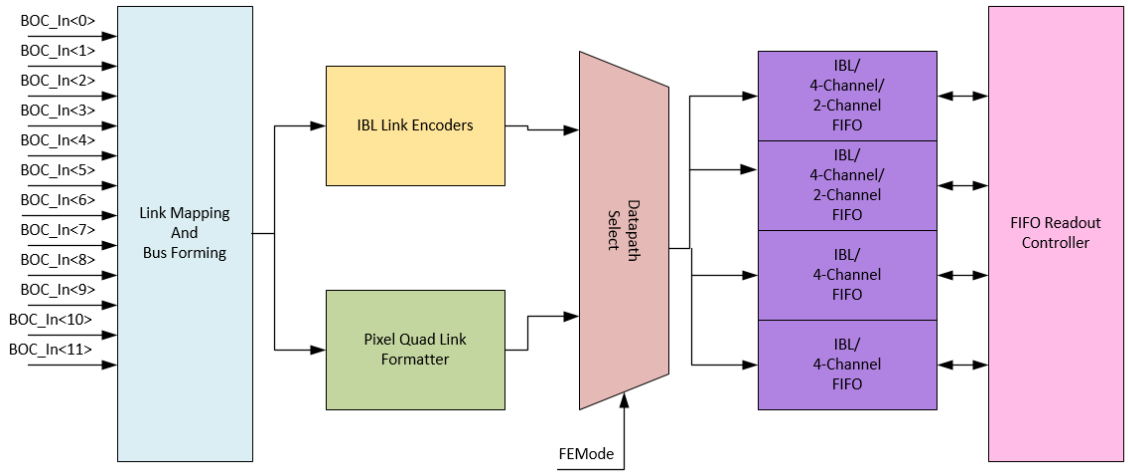


Figure 3.4: Formatter Datapath showing L1/L2 integrated with IBL

Modifications to the formatter took place first since the changes done here would affect what changes needed to be made in the subsequent modules. The first concern for the Formatter involved how to decode the serial data sent from the MCC. It was decided that the best solution was to recycle the serial link decoder from SiROD because it could be easily integrated and had shown to work effectively and without error throughout the full length of Run 1. Figure 3.4 shows the integration of the datapath starting with the BOC inputs fanning out to both encoders and then the multiplexor which decides with type of encoding to use; after the multiplexor in can be seen that the upstream treats both types equally. The link decoder for L1/L2, known as the Quad Link Formatter, operates in three different decoding modes: 4 MCCs at 40MHz, 2 MCCs at 80MHz, or 1 MCC at 160MHz; the operating mode is chosen by software through a programmable on-slave register. A diagram of the first two modes is shown in Figure 3.5, with the first mode using one link per QLF and the second pushing the streams from two links into a

single QLF. In the L1/L2 upgrade only cases one and two were ever under consideration since 160MHz operation is just for the B-Layer. The link mapping, functionality of the original SiROD, which allows for an arbitrary mapping between the BOC inputs and the inputs to the Quad Link Formatter, was also kept in place to provide greater flexibility.

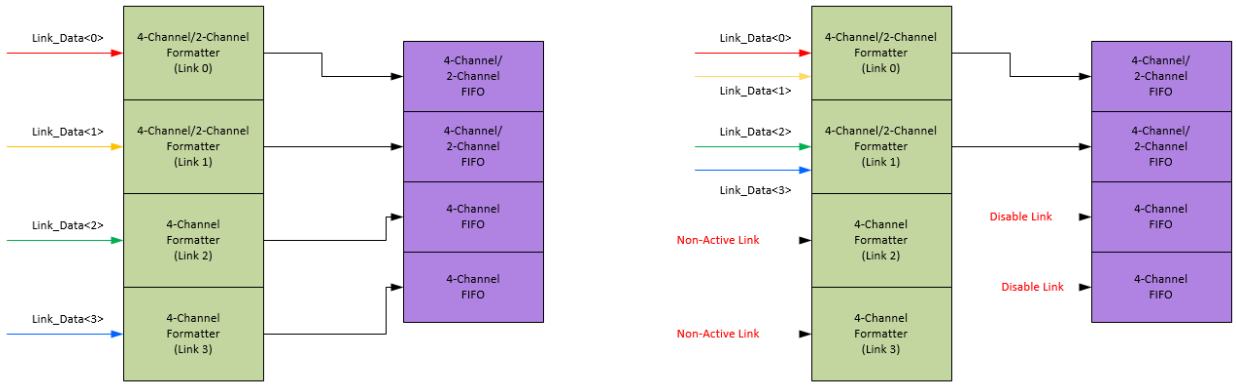


Figure 3.5: Formatting of 4-Channels at 40MHz and 2-Channels at 80MHz respectively.

Next the data words that were output from the link decoder had their bit fields modified according to Table 3.1 in order to more resemble IBL's. This would lead to less work being done in subsequent modules. Error checking and the FRC state machine were kept the same and did not require modification. While seemingly simple to flip only a few bit fields, great care must be taken, and meetings held across all parts of the DAQ and offline data monitors to ensure everyone is aware of and agrees upon the various changes.

Table 3.1: Original and Reformatted Formatter Output for L1/L2 [10]

Pixel Formatter Output Bits [31:0]

Name	Bits [31:0]
Header	001PxxxxxxxxAAAAMMMMLLLLBBBBBBBB
Trailer	010ZHVxxxxxxxxxxxxxxxxxxxxxxxxxx
Hit	100xFFFFTTTTTTTTxxxCCCCCRRRRRRRR
FE Flag Error (Old)	0000FFFFxxxxxxxxxxx11110FFFFEEEE
FE Flag Error (New)	0001FFFFxxx11111eeeeeeeeEEEEEEEE
Raw Data	011DDDDDDDDDDDDDDDDDDDDDDDDDDDD
Time Out Data	00100000000000000000000000000000

ReFormatted Pixel

Name	Bits [31:0]
Header	001nnnnnxxxxxxxxxxLLLLxxBBBBBBBB
Hit	100xFFFFTTTTTTTTxxCCCCCxRRRRRRRR
Trailer	010nnnnnEcPplbzhvxxxxxxMMMMxAAAA

Key:

A	BCID offset	B	BCID
C	Pixel Column	D	Raw Data
E	FE Error Code	e	MCC Error Code
F	FE Number	H/h	Header/Trailer Limit
L	L1ID	M	Skipped Events
P	Header Error	R	Pixel Row
T	ToT value	V/v	Row/Col Error
X	Don't Care	Z/z	Trailer Error
b	BCID Error	l	L1ID Error

The first aspect of the EFB that was inspected for differences was the event info received from the Master FPGA. The event information between both generations of RODs is identical so no changes were made in how the ROD parses and stores the event information. Raw data from the FRC uses the same FSM for synchronization as IBL and is again read into two parallel datapaths to accommodate the data from both Formatters. The only major difference is the slicing of the raw data in each of EFB submodules that comes in as it pertains to the bit field changes that occurred in the Formatter. Checks of the L1ID and BCID are still done in the same fashion as in IBL. The Error Recording block of the EFB had to be complete separated between IBL and L1/L2 because the MCC and FEI4 report different flags from their respective IC operations. The two paths were multiplexed to get the final output. The EFB for L1/L2 will also handle fewer modules, either six or seven at a time, compared to IBL's eight. The SLINK Headers and Trailers were also consistent between the successive generations of Pixel Layers. This meant that no changes had to be made to the Fragment Generator state machine.

The Router block underwent a significant change due to a requirement of the upstream readout: because legacy software on the Level-2 computers are only capable of handling 2 SLINKs per ROD. This meant the Router would need to compress the data from up to fourteen MCCs onto one SLINK output, causing some pressure to be applied to this section of the readout chain since the SLINK will still only operate at 40MHz, half the speed of the rest of the datapath. The data slicing for histogramming also required changing as a result of the bit field modifications. The MicroBlaze also required modifications to deal with the different ToT levels and the number of chips calibrated on a single MCC. These solutions were not developed in this work.

3.2: MCC Emulation and Datapath Testing

To confirm the success of the L1/L2 integration, and the correct operation of the datapath, both simulation and hardware tests were done. Both sets of tests relied on a MCC Emulator that was integrated and used as a built in self test to validate the upgraded datapath. This emulator was created by the designers of the original SiROD and ported to the IBL ROD where it was multiplexed with the BOC inputs on the Formatter Rx lines. Modifications were made to allow the emulator to intercept hardware triggers from the TIM along the new Tx path. Programmable registers were added to the Slave's register set, allowing the emulator to be turned on and off via a command line interface. Additional registers were created to control emulator functionality such as number of BC frames, number of hits per frame, and which MCC flags are present. The

hardware tests used a TIM to generate local triggers. When the emulator generated data ChipScope was used to spy the functionality of various aspects of the datapath to confirm correct operation. Unfortunately, at the time of the tests no high level software had been created for data quality nor Level-2 readout and processing.

Section 4: ITk DAQ & RD53 Emulator Development

The Inner-Tracker (ITk) Upgrade aims at completely overhauling the ATLAS Pixel detector and replacing it with a faster and simpler design capable of taking larger amounts of physics data. This upgrade is forecasted to take place in 2020 and correspond with the upgrade of the LHC to even higher energies and luminosities. ATLAS has responded by revamping their detector subsystems. The key consideration for experiments such as ATLAS is the ability to achieve higher overall trigger rates, implicitly improving the data readout speeds of all its sub-detectors, since in the end ATLAS's data taking capability is capped by that of its slowest sub-detector. Currently ITk is in the research and development phase and new possibilities are being discussed for everything from the sensor material and power cabling to DAQ and higher level software considerations. One subsection of the ITk upgrade involves investigation into what the next-generation front-end readout chip for Pixel should look like and what features it should contain. The current prototype for the IC is known as RD53. To assist in this effort an FPGA emulator of the RD53 integrated circuit has been developed, as well as a small DAQ core to communicate with the emulator and serve as a proof of concept for next-generation ITk DAQ systems.

The specification for the RD53 is in the beginning stages and is not fully complete. It is expected that once specification is finished it will take six months to one year to receive fabricated chips. This leaves an opening for an FPGA-based emulator to fill. The emulator will be available far in advance of the IC and provides ample opportunity for prototyping different functional aspects of RD53's digital blocks. With this in mind the project aimed at emulating a very specific (and most well-defined) aspect of the RD53. The IC's digital communication blocks. Implementing the digital communication blocks of RD53 on an FPGA allows for the testing of functionality under debate, such as different trigger encodings, hit data out encodings, and hit data output speeds. It also provides the opportunity for DAQ system researchers to have a device with which they can test their systems long before the actual chip is available.

4.1: RD53 Emulator Development

The RD53 FPGA emulator contains three major modular components: the Clock and Data Recovery (CDR) block, the Timing Trigger Control (TTC) word alignment block, and the word decode and output block. The high-level summary of RD53's purpose is to take in a serial TTC stream at 160Mb/s, decode its meaning, and respond accordingly. The three major modules listed create a digital communication shell inside of which other data processing logic can be inserted. The full block diagram for the RD53 emulator is shown Figure 4.1. Multiple clock domains are required to correctly emulate the functionality of RD53: the 160MHz domain is needed to process the incoming TTC serial stream, a 640MHz clock is required to do CDR on the TTC stream, and the 40MHz clock is needed to replicate the clock that occurs on the actual chip and synchronizes data processing (40MHz represents the bunch crossing time). The clock domain that each module functions in is represented by the different colors in Figure 4.1, with 640MHz, 160MHz, and 40MHz represented by green, blue, and orange respectively. An analog Phase-Locked Loop (PLL) macro-block on the FPGA is used to generate low-jitter versions of the first two clocks from a local 200MHz oscillator on the FPGA board. The recovery and creation of the 40MHz clock will be discussed in the next section.

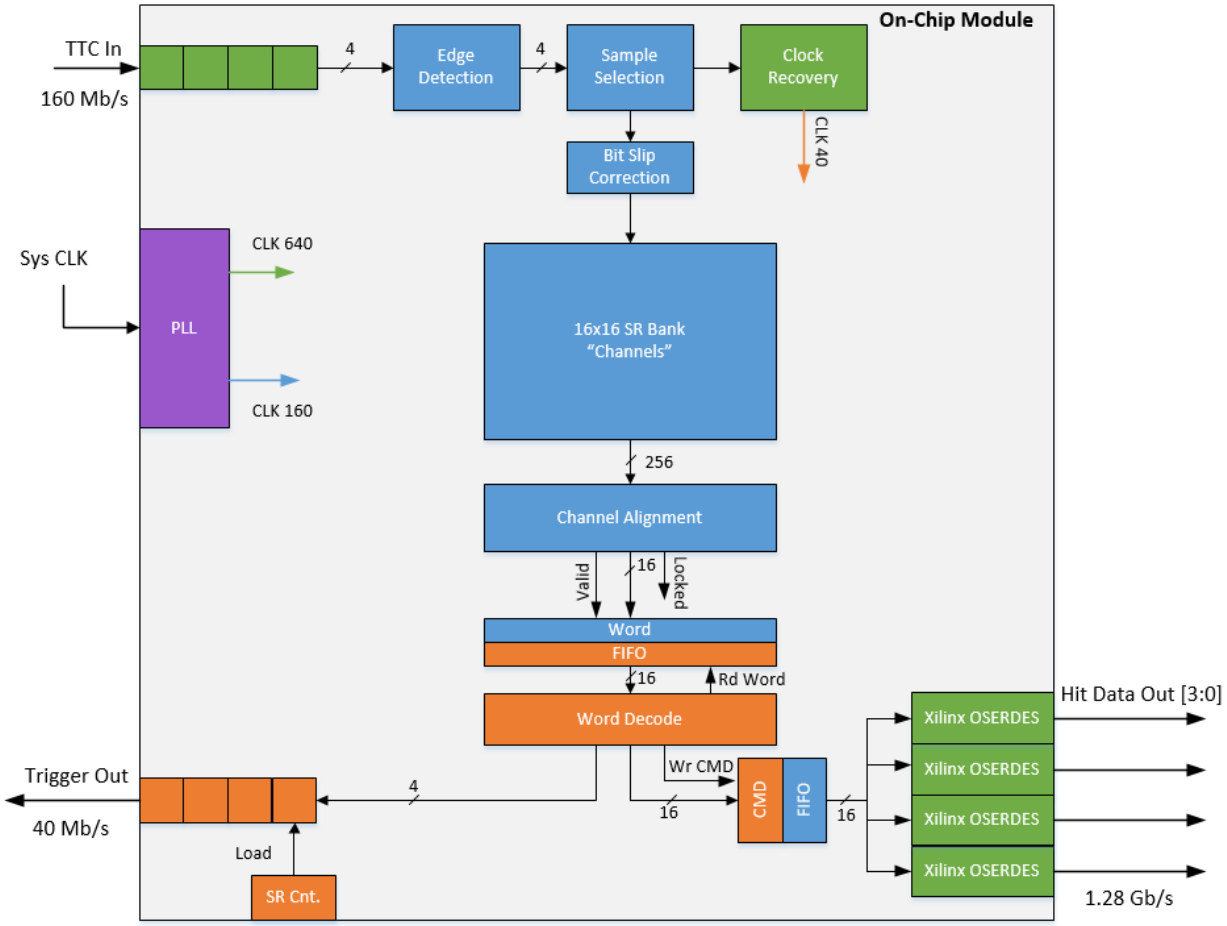


Figure 4.1: Block Diagram of RD53 Emulator

4.1.1: All Digital Clock and Data Recovery in an FPGA

The first aspect of the RD53's digital communication blocks that needed to be emulated was the clock and data recovery of the TTC input, an asynchronous 160Mb/s serial stream with 16-bit wide data words. An all-digital version of CDR is difficult because the incoming stream is purely data, meaning that it lacks a high number of level transitions, making its phase hard to discover (in the RD53 specification the number of consecutive bits sent without a transition is limited to 6 [11]). We cannot use custom analog Phase-Locked Loops (PLLs) to cleanly recover it, and the PLLs on the FPGA don't have this capability. This led the problem of CDR to be broken down into two parts: first is the recovery of the incoming asynchronous data into the local 160MHz clock domain, second is using the information from the data recovery to estimate the phase of the transmitting clock to within 90 degrees of the actual phase. This can be done because the speed of the incoming serial stream is known beforehand; meaning we can create a local clock of matching frequency, this not true of all CDR applications. However, there is still the problem that the receiving clock's phase will drift slowly with respect to the transmitting clock. We must rely on the presence of edges in the data stream to identify how much drift is occurring so that we may compensate for it. This is why we force the data stream to have at least one transition

every six cycles. Aspects of asynchronous data recovery in FPGAs had been worked out before, for example in Xilinx Note 225 [12], which was used as a reference for this application.

The initial stage of the data recovery involves oversampling the incoming data at 4 times the actual data rate, done in the 640MHz clock domain. By doing 4x oversampling we are essentially cutting the incoming data into pieces of 90 degrees of phase resolution. Each of the 90 degree phases is given a moniker of A, B, C, D from 0 to 270 respectively. Intuitively A, B, C, and D represent a set of four data samples taken during a single cycle of the local 160MHz clock. First the input is sampled in the 640MHz clock domain with each bit sampled stored in its own buffer. Next the oversampled data is delayed by one clock cycle in the 160MHz domain to remove any metastability that may occur around edge transitions, since the two clocks share the same phase it is essentially a two-bit synchronizer. Then the stable data set is fed to an edge detection logic block that looks for an edge transition in the oversampled data. The sample selection block then takes the information of when and where a transition occurred and chooses the best phase in which to sample the incoming data in order to record the correct value; typically, this means 180 degrees away from where the edge occurred. Both the edge detection and sample selection logic are done in the 160MHz domain. An example of the 4x oversampling can be seen a waveform diagram in Figure 4.2.

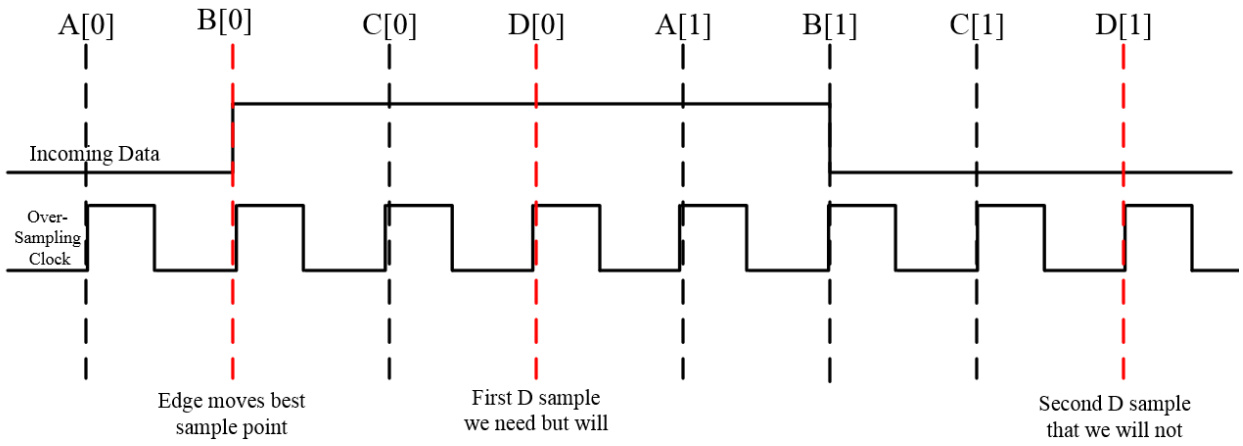


Figure 4.2: Example of bit-slip from A to D

Ideally you would have one, and only one, bit of valid data in the sample set every 160MHz clock cycle. In normal operation, once the set of four samples (A, B, C, D) has been collected they are written into four matching 4-bit shift registers, as seen in Figure 4.3. The valid output bit from the recovery operation is bit 2 from the shift register whose corresponding phase has been deemed the best current sample point. For example if C is our current best sampling point then DataC[0] from Figure 4.3 is the valid data bit for other components to use. However, it is not always the case that only a single bit, or that any bit, from the set is valid. A primary concern when doing this type of asynchronous data recovery is what is known as a bit slip. Bit slips occur when transitioning the best sample point from either the A phase to the D phase, or conversely from the D to the A phase. These two transitions cause, respectively, either an undersampling or oversampling of data that needs to be corrected.

As previously stated, in normal operation only the second bit from the top is valid for whichever shift register is currently designated as the best sampling phase. If, however, there is a bit slip that is changed. Figure 4.2 is an illustration of a bit slip from the A to D phase, which results in an undersampling of the incoming data. We can imagine starting off using A as the best sampling phase. Then in the sampling set denoted by index 0 we have an edge occur near the B phase. It is now in our best interest to switch from sampling in the A phase to sampling in the D phase. This is because using the D phase puts our sampling point closer to the middle incoming pulse. However, we can't acknowledge the switch to D until the sampling set one time step later, represented by the index 1. This causes us to miss the high pulse that occurs in Figure 4.2. The solution is, on the first 160MHz clock cycle where D is the best sample point, to take the top 2-bits of its shift register; which in this example would be DataD[1] and DataD[0] in Figure 4.3. The result is no loss of data from undersampling. The existence of this type of bit slip is why we extend the shift register by an extra bit, so that we can hold onto the value from the previous sample in case it is needed. The opposite is true for going from D to A, where you have sampled too much and must skip a cycle of output from a shift register by outputting no valid data.

The valid data bit, or bits, from the sampling set are written into a 17-bit shift register, shown in Figure 4.4, used to assemble a full 16-bit data word. In Figure 4.4 we see that the amount of data written in is monitored by a 2-bit control value that is aware of when a bit slip occurs. If these two control bits have a value of "11" then 2 bits of data are written, for "00" no bits are written, and in all other cases only 1 bit is written. The 17-bit shift register then multiplexes its parallel 16-bit output, and decides when to be valid, based upon if and when bit slip data is written in. If a shift register is 1-bit away from being valid and 2-bits get written in due to a bit slip it must output the top 16-bits and exclude the bottom bit while starting the shift registers counter over at zero. Other than this unique case the shift register operates as normal, outputting bits 15 down to 0 every 16 clock cycles.

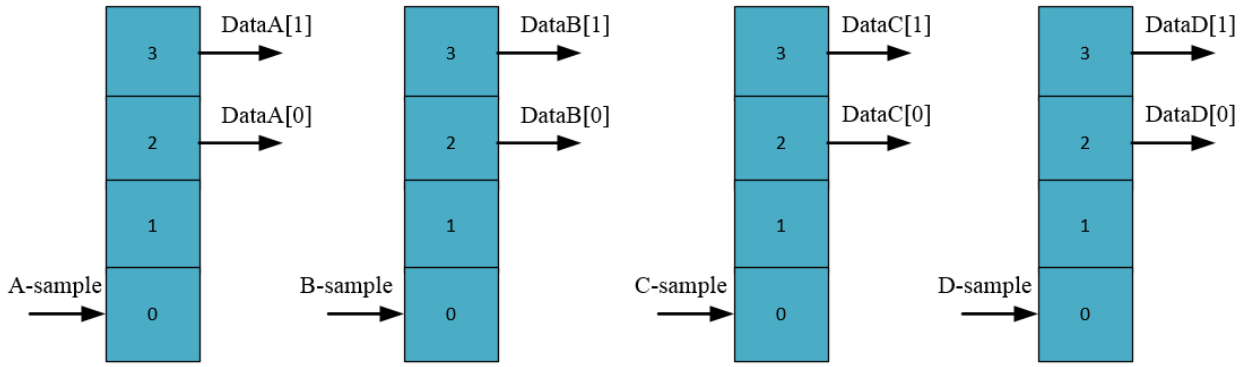


Figure 4.3: 4-bit Shift Registers for storing delayed samples

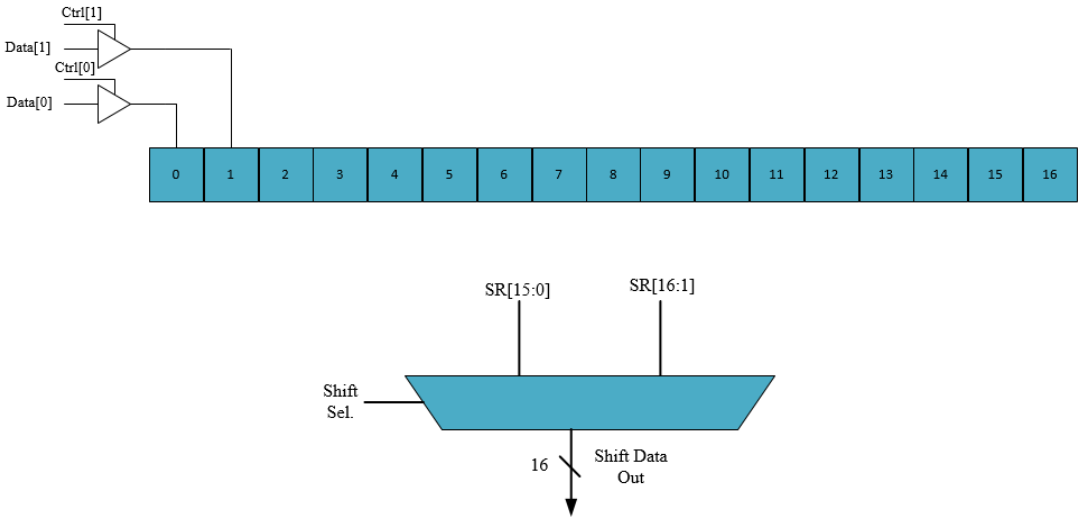


Figure 4.4: 17-bit shift register for received data words

Finally, there is the recovery of the clock. Since a precise analog PLL is not available to help us recover the transmitting clock phase of the incoming data, we must do the best we can to estimate the phase. To do this we use the 90 degrees of resolution obtained from 4x oversampling the data. Simply put in order to recover the clock's phase we observe where the edge of the incoming data occurred and chose the closest of our four available phases (A, B, C, and D) as the zero phase of the locally produced transmitting clock. This logic generated clock then gets divided down, using a simple counter, from 160MHz to 40MHz in order to produce the operating frequency of the internal RD53 components. While not the most accurate way to recover the phase of a clock, the jitter and the maximum 90 degrees of incorrect phase were deemed acceptable for the emulator project.

4.1.2: Channel Alignment

When transmitting 16-bit data words there are 16 possible "channels" in which the correct alignment of the data word could exist. The asynchronous receiver must have the ability to view all channels and select the correct one. In the RD53 emulator this is coordinated by a 16x16 bank of shift registers; one for each channel. There was an attempt to view all the channels through the use of only a single 16-bit shift register, but this proved to be difficult in the presence of bit slips. Each register is given the same values from the data recovery module on each cycle, but each has a different counter value from 1 to 16. Thus on every 160MHz clock one of the registers in the bank is valid.

To lock to a given channel the sync pattern must be detected in that channel's shift register. The sync pattern is a value (currently set to 0x817E) that is sent periodically to keep the transmission link alive. For a given channel to be considered "locked" to the transmitter it must have received this sync pattern for a specific number of valid data words; currently that number is set to 16. Once a channel reaches the locked state it can then pass on its data words for decoding and further processing. The simulation waveform for locking a channel can be seen in Figure 4.5, with the lock value of 16 and the subsequent valid of the next data word being shown. Since only

a single channel can be locked at any given time there is also a mechanism for unlocking a channel, that being if another channel sees a different number of sync patterns. This is the unlock number and is currently set to half the number required to lock a channel.

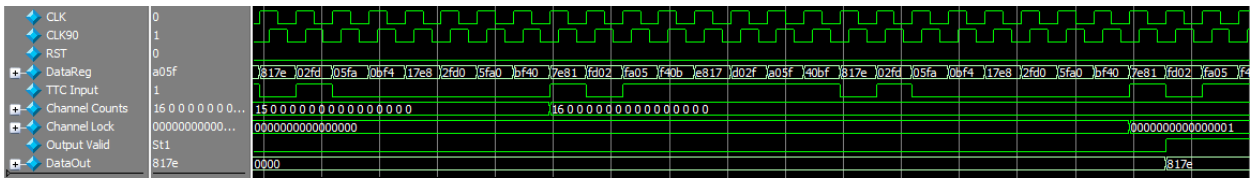


Figure 4.5: Simulation example of a channel becoming locked

As an example, consider a freshly reset system. After some time channel 2 has been observed to have received 16 sync patterns, channel 2 is then considered locked and its data words are passed on to the rest of the emulator. Once channel 2 becomes locked it also resets the sync pattern counters of all the other channels, while leaving its own intact. Now let's say that channel 2 has not observed a sync pattern in some time, but instead channel 3 has begun to receive the sync pattern in its register. Because channel 2 is no longer observing syncs its counter is stagnant and is not resetting the other channels. If channel 3 is able to accumulate enough sync patterns, and reaches the unlock value of 8, then channel 2's lock is wiped out, every channel has its sync counter reset, and whole process starts over. Currently there is no method of alerting the sync transmitter to this unlock occurrence so that action may be taken to avoid data loss.

4.1.3: Data Decode and Output

Once a channel is locked then its corresponding data words can begin to be decoded and their meaning understood. Currently there are two separate data words for decoding: trigger words, and command words. Because of their importance in the DAQ system the triggers have a unique encoding. At the current moment, however, there is no specified encoding for the triggers so a one-hot encoding was created for the purposes of testing. In the decoding system anything recognized as a trigger is decoded into its corresponding 4-bit trigger pattern and given to the trigger shift register for output. Everything else that is decoded is assumed to be a command and is written into the command word CDC FIFO. Commands are then transferred back out over the hit data bus via Xilinx OSERDES at 1.28Gb/s, with no special encoding given for the output. As RD53 matures in its development a specific encoding should become available. If the OSERDES are not sending a command, then they default to outputting the sync pattern. In future work commands will hopefully be interpreted and cause some internal stimulus in the emulator to output data over the hit data bus.

4.2: Development of a matching DAQ

A DAQ system was developed to communicate and test the RD53 emulator over the 160MHz TTC link. This DAQ has a core set of functionality that is likely to appear in all next-generation ITk DAQs because it matches the communication RD53 will obey. The core functional blocks of this DAQ are: the trigger processing module, the command and synchronizing modules, and the TTC word control FSM. The trigger processing block models the receiving of a hardware trigger from a local TIM and the command processor block models receiving a command from higher level software. The FSM then controls the coordination of this information being sent to the

Front-End. The DAQ module also uses a PLL to generate two clocks, 40MHz and 160MHz, which are also defined by their colors in Figure 4.6. As we will see in later sections many system settings were left open to programmability in order to test their impact on the DAQ system.

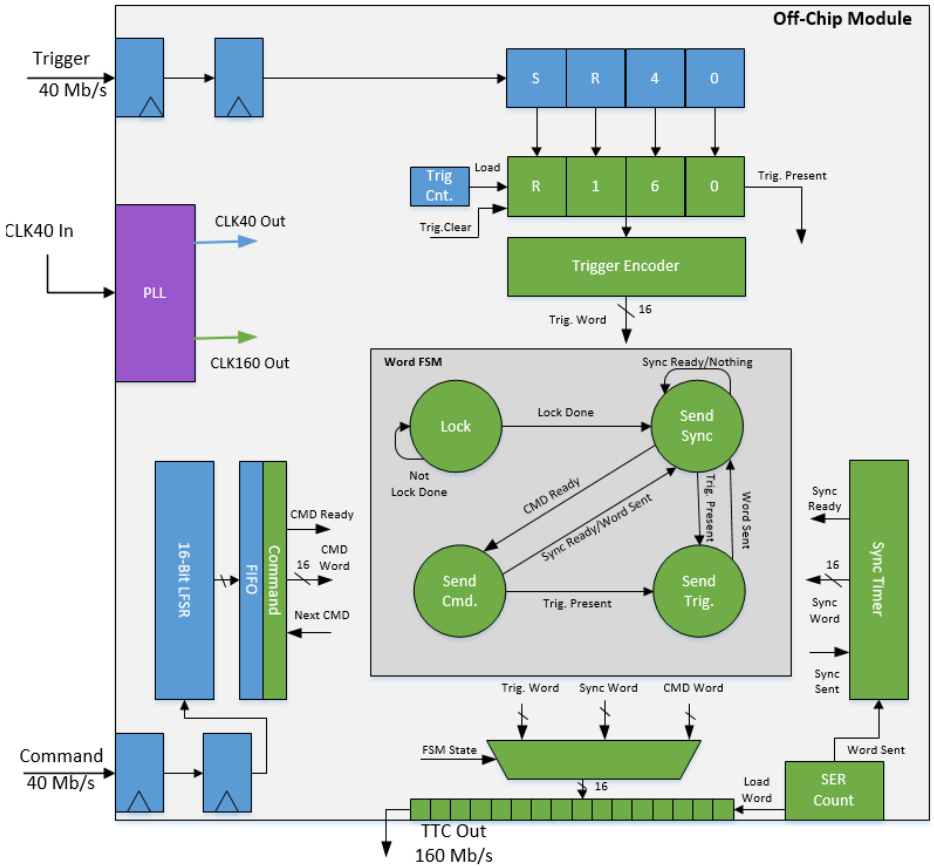


Figure 4.6: Block diagram of the DAQ system

4.2.1: The Trigger Processor

External triggers are captured asynchronously in the local 40MHz clock domain and passed through a 2-bit stabilizer. While it is true that the 40Mb/s could be generated by the same clock driving the PLL no such requirement is made in this DAQ system. Doing so does not hurt performance or skew testing, so it is treated as any other external signal. After synchronization trigger pulses are transferred into a 4-bit shift register on every 40MHz clock. An independently running trigger counter is then responsible for loading the trigger sequence into a 4-bit register in the 160MHz domain. The relationship between the two clocks, that they are derived from the same clock and one is a multiple of the other, here is important for two reasons: Firstly, no special cross clock domain techniques are used in passing data between the two because that would introduce added latency. This is an acceptable tactic here because the two clocks have a shared phase relationship. Second is the coordination between the trigger counter and the serializer counter responsible for outputting the 16-bit TTC word. While independent of each other, in the sense that there is no shared communication between them, they are coordinated based upon the relationship of their clocks and both start a new shift sequence, on the same phase. Figure 4.7 shows the Trigger processor in action and it priority in the system. The number

of clocks to output the trigger can be seen in Figure 4.7's waveform, it shows that after the trigger is in fact processed in the necessary amount of clock cycles to guarantee its immediate output.

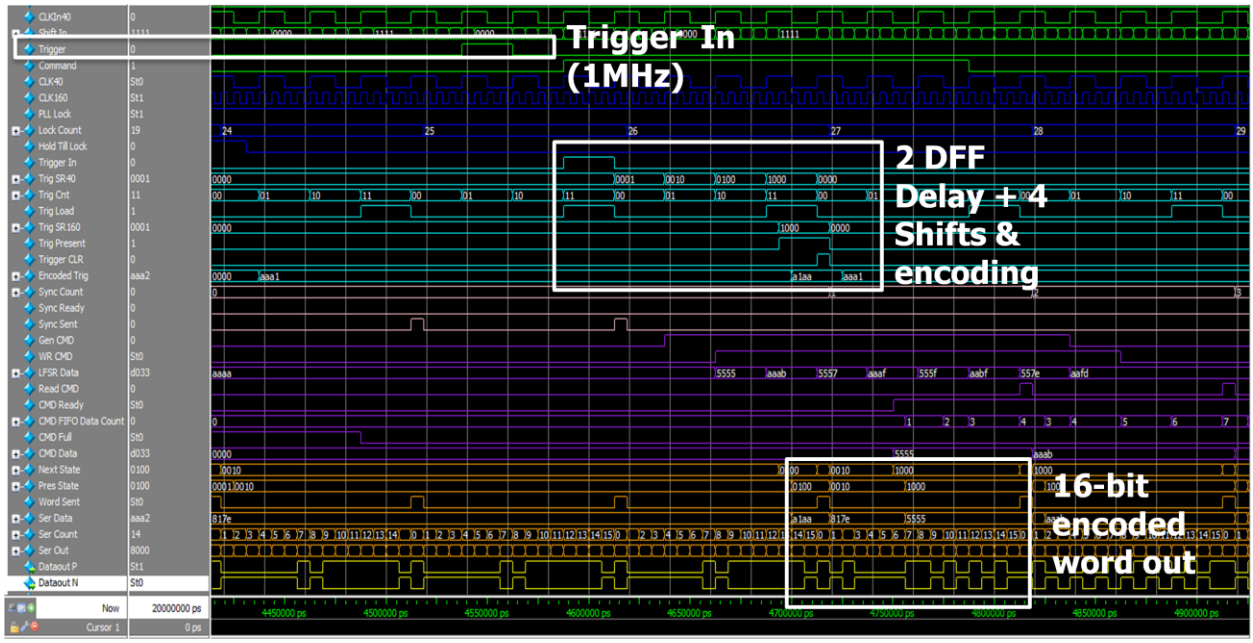


Figure 4.7: Simulation of a trigger processing timeline

After the trigger register is latched into the 160MHz domain the logic uses a one-hot pattern to encode the trigger sequence into a 16-bit word. The logic also detects if a trigger is present and if so alerts the control FSM. The whole process, from first shift to encoded trigger word ready to be sent out takes, only 14 cycles of the 160MHz clock. This fact is important because it guarantees that if a trigger is present it will be the next TTC word sent out after the current one is finished, giving it the lowest possible latency. Finally, after the encoded trigger is taken by the TTC for output the 4-bit register in the 160MHz domain is cleared so that the control FSM can transition away from the send trigger state.

4.2.2: Command Generator and Sync Timer

Apart from triggers the two other types of TTC words that the DAQ can send are command words and the previously mentioned sync pattern. Command pulses are input into the system in the same fashion as triggers, and for the same reasons they too are passed through a 2-bit stabilizer. Once synchronized the command pulse initiates the generation of a random 16-bit word from a Galois-type LFSR. This was the simplest solution at the moment because presently RD53 lacks any tangible commands that could be sent to the emulator. After the command word has been generated it is put into a CDC FIFO for storage and the control FSM is alerted via a valid/ready signal that there is a command available to send. If the FSM chooses to send the command, it must simply load it into the TTC shift out register and use the Next CMD signal to remove the command from the front of the FIFO.

The sync timer module exists in order to ensure that the predefined pattern of 0x871E is sent for the appropriate fraction of cumulative TTC words so as to keep the communication link locked. In the current system for testing the fraction of sync words that must be sent is 1/32, this has an effect upon the available TTC bandwidth that will be discussed later. For the majority of operation there are not triggers or commands in the priority queue waiting to be sent. Therefore, the sync pattern is constantly being transmitted and its timer never reaches the terminal value forcing a sync to be sent. However, when TTC bandwidth is limited, and many command and trigger words are contending for the output the sync must assert itself to the control FSM by setting and holding its sync ready signal high until its request has been met.

4.2.3: The TTC output word control FSM

As hinted at in previous sections the Control FSM is the center of the DAQ and controls which of the three word types gets sent out over the TTC link. Starting in the Lock state the FSM sends a preset number of sync patterns to give the emulator a large enough sample so that it can lock on to the correct channel, as described in Section 4.1.2. Currently the number of sync patterns sent from the lock state is set at 32, twice the number needed for an aligned channel to become locked. After Lock is finished the FSM transitions to being able to send either of the three word types, but enforces priority on which it chooses to send.

The priority order is simple: triggers have the highest precedence, followed by the sync pattern, and lastly the command words. Triggers have the highest priority in all readout systems because they are the catalyst for all data taking operation and need to be processed as soon as they are received. By giving them the highest priority it secures a fixed latency for their processing time. Sync is giving the second highest because, while not as important as triggers, its purpose of keeping the TTC communication channel in proper working order is more important than a command. Due to its default status it gets sent with greatest frequency of any of the three word types. Finally, while commands are important, they have no need to be processed in a specific amount of time, thus leading to their low priority status.

4.3: FPGA Emulator Hardware

The hardware chosen to emulate the RD53 is the Xilinx KC705 board, which can be seen in Figure 4.8. This board was chosen for several reasons. Chief among them are the FPGA as well as the myriad I/Os available on the board. The FPGA is a Xilinx Kintex-7, and in addition to containing enough LUTs to deploy several emulator instances together in a single chip, it also contains many hard macro blocks required for this project such as PLLs and the multi-gigabit transceivers (MGTs). The board itself contains two FPGA Mezzanine Connectors (FMCs), both a high pin count (HPC) and low pin count (LPC), which allows for the creation of breakout boards to interface with the FPGA. Many different types of breakout boards with various cabling have been suggested for the RD53 emulator, from VHDCI and RJ45, to DisplayPort. For this project a preliminary breakout PCB was designed using Altium as a prototype for such a board. The layout involved two DisplayPort connectors for a loopback test connected to the FMC port via LVDS pairs.

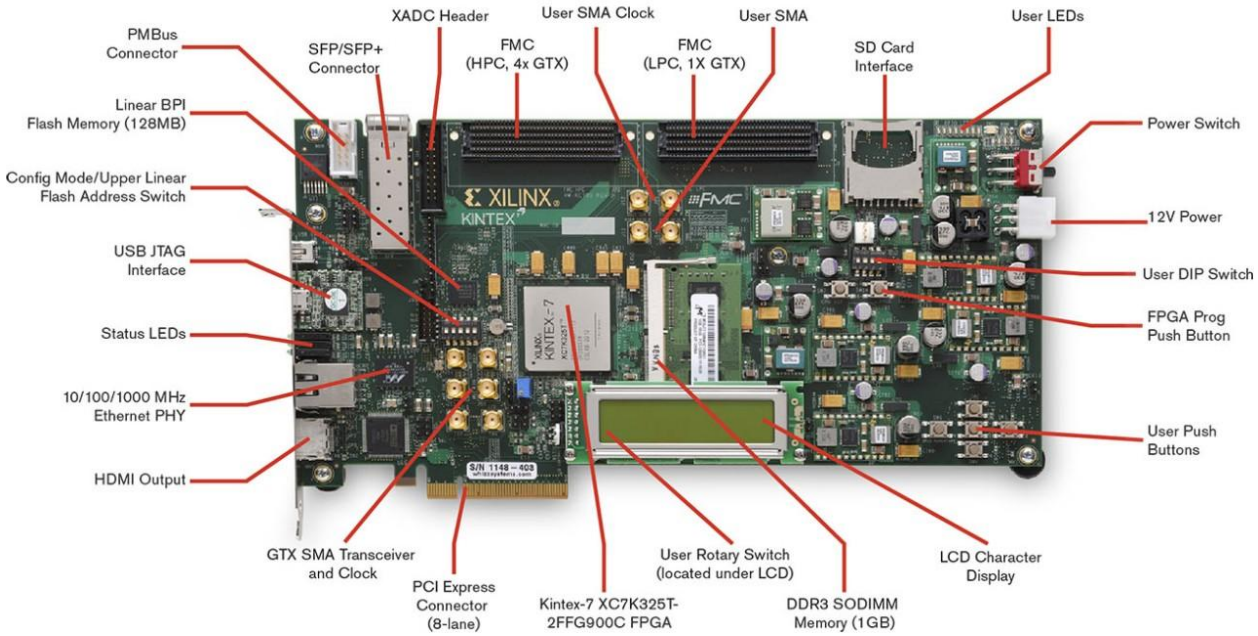


Figure 4.8: Xilinx KC705 board with key components labeled [13]

4.4: Trigger Latency and Command Bandwidth Tests

In addition to verifying the functionality of the DAQ/Emulator, initial tests were done to research the performance properties of the systems. The two tests that were performed were for fixed trigger latency and available command bandwidth. The fixed latency tests measured the number of bunch crossings, or 40MHz clocks, that it takes a trigger pulse to propagate from its starting point in Figure 4.4 of the DAQ to its final output in Figure 4.1 of the emulator. This timing will be important in ITk readout because the trigger has a latency interval in which to capture the correct data associated with a given bunch crossing. The lower the latency the quicker the trigger can get to the FE and process its data. In the DAQ/Emulator system fixed latency is guaranteed by two factors: the shift order being preserved in both the DAQ and Emulator trigger shift registers, and by the FSM control module in the DAQ granting highest priority to the trigger. For tests done in ModelSim the trigger was found to have a fixed latency of 22 BCs. While a good number, some of it is due to overhead as a result of FPGA emulation of RD53. Specifically the CDR blocks introduce an overhead of approximately 3 BCs.

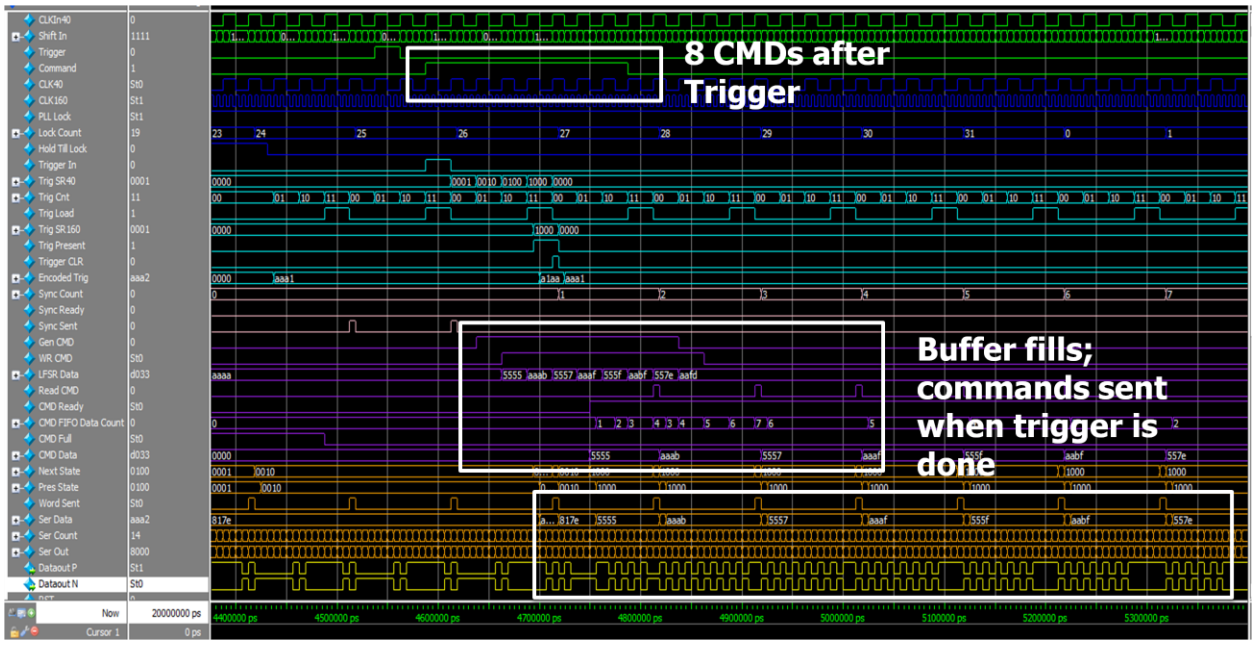


Figure 4.9: Simulation showing the command bandwidth tests

For the command bandwidth the investigation involves discovery of the number of command words that can be sent under a given set of trigger and sync conditions. Since the TTC link itself operates at 160Mb/s, and outputs 16-bit words, a max bandwidth of 10MHz exists as our upper bound. In terms of triggers we care about two factors: the trigger frequency and the input trigger pattern. The effects of trigger frequency are obvious; more triggers consume more TTC bandwidth. The pattern's affect is a little subtler. Imagine a pattern of two consecutive triggers. It's possible that a pattern of two consecutive triggers could be processed as one trigger word, appearing as bits 2 and 3 in the shift register. It is also possible that it gets split into two separate words that need to be sent; with the first being sent as bit 3 of the shift register in set 1 and the second as bit 0 in the next set. The sync consideration is also clear, the higher the fraction of sync words that need to be sent less bandwidth is available to send commands. In the ModelSim tests, as shown in Figure 4.9, the trigger was a single pulse with a frequency of 1MHz, and the sync fraction was left at 1/32. The result was a command bandwidth of 8MHz or 8 commands per trigger.

Section 5: Conclusion and Future Work

For the first two runs of the LHC the basic architecture of the Pixel DAQ system remained the same, with the Readout Driver Card standing at the center of operation. The three primary modules of the ROD were responsible for processing raw data from Pixel into physics events. This 3-block model of DAQ readout was left unchanged from Run 1 and into Run 2 staying stagnant for IBL and the upgrade of Layers 1 and 2. If we look back at the timeline in Figure 1.7 will we see that is nearly 15 years using the same readout architecture. With the coming ITk upgrade a new look will be needed for future DAQ systems, and the architecture of these readout models will have to be discovered through development and testing over the course of the next few years.

Future electronics work for the ITk upgrade will involve developing and assessing the validity of the next-generation DAQ systems, using both the RD53 emulator (presented in Section 4) and actual IC chip. These DAQs will be assessed on their ability to process large amounts of data created by the FEs and high trigger rates (300KHz - 1MHz), meaning high throughput architectures will need to be exploited on the readout FPGAs. Another parallel goal of the DAQs is efficient and faster calibration times. This means histogramming the data from millions of pixel sensors and moving it from a task that used to take hours to complete to hopefully one that takes only a few minutes. If achieved in a real system then the full detector would be able to be recalibrated more frequently, leading to more accurate physics and a better performing detector. Some solutions are already being tested in this area and involve fast FPGA data binning and high-speed communication over PCIe to a terminal running several simultaneous software threads for creating histograms. Finally, there is a push within the ITk community to make the next-generation DAQ system hot-pluggable in terms of PCB components used. They would like to develop a system that is not dependent upon specific version of FPGAs or other components. This would take advantage of the fact that when a newer faster commercial FPGA becomes commercially available it can be effortlessly integrated into the system and its benefits (such as faster clock speeds) be realized, a lesson learned from SiROD and the L1/L2 upgrade.

For the RD53 emulator and its DAQ specifically there are a few key enhancements and tests that can be done on a short timescale that will prove useful to the ITk community in assessing next-generation DAQ systems:

- **Programmable Register File:** The addition of a register file to the emulator would serve two purposes: First it would present an opportunity for simple read and write tests to show that a DAQ is able to communicate with the emulator. Second it will allow for the investigation of different command encodings which are an important consideration based upon exclusivity with trigger encodings and the need for a large number of commands.

- **Hit Data Emulator:** A mechanism that responds to received triggers on the emulator by outputting a programmable number of hits will be useful in testing bandwidth capabilities of future DAQ. While such an emulator would not be able to precisely capture the latencies that occur in readout of the actual silicon sensor it would be a useful first order

approximation. The emulator could even be tuned to create desired latencies to investigate exactly how much latency is tolerable.

- **Multiplexing of TTC:** Being able to multiplex a single Timing and Trigger Control interface to multiple chips would be useful in decreasing the number of cables going to the detector. While all FEs could use the same sync signal, a multiplexing strategy would need to be developed that distributes triggers equally to all chips but with addressable commands.

- **Multiplexing of Hit Data:** The multiplexing of the hit data from multiple FEs would also reduce the number of cables between the detector and counting room electronics. The two major concerns of hit data multiplexing would be the available bandwidth of the both the integrated circuits and the cabling as well as the asynchronous demultiplexing of the data in the off detector DAQ.

The implementation of these and other future enhancements will require continued collaboration with those at ATLAS ITk institutions, most notably with the SLAC RCE group, YARR group at LBNL, and the RD53 circuit designers. Collaboration will ensure development is being done that will help to further the upgrade's development.

Bibliography

[1] The ATLAS Collaboration, "ATLAS Insertable B-Layer Technical Design Report", CERN-LHCC-2010-013, https://cds.cern.ch/record/1291633.

[2] Kohn, Fabian, "Measurement of the charge asymmetry in top quark pair production *pp* collision data at $\sqrt{s} = 7$ TeV using the ATLAS detector", CERN-THESIS-2012-024, http://inspirehep.net/record/1103034.

[3] Ludwig-Maximilians-Universitat-Munchen, ATLAS Experiment, http://www.etp.physik.uni-muenchen.de/research/atlas/.

[4] Chen, Shaw-Pin, "Readout Driver Firmware Development for the ATLAS Insertable B-Layer", University of Washington, June 2014, http://www.ee.washington.edu/faculty/hauck/publications/BingThesis.pdf.

[5] http://hilumilhc.web.cern.ch/about/hl-lhc-project.

[6] The ATLAS Collaboration, "ATLAS Phase-II Upgrade Scoping Document", CERN-LHCC-2015-020, https://cds.cern.ch/record/2055248.

[7] The ATLAS Collaboration, "IBL ROD BOC Manual" (development draft version), https://espace.cern.ch/atlas-ibl/OffDecWG/Shared%20Documents/Forms/AllItems.aspx?RootFolder=%2Fatlas-ibl%2FOffDecWG%2FShared%20Documents%2FReadout%20System%2FIBL%20ROD&FolderCTID=0x01200026DA0CF8EC432F48938A1D26EF579130&View={952E30F4-2295-4C1C-9696-356CDBFF7724}.

[8] The ATLAS Collaboration, "Synchronization errors in the Pixel detector in Run 2", PIX-2015-003, https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/PIX-2015-003/ .

[9] Beccherle, Roberto, "The Readout Architecture of the ATLAS Pixel System", National Institute for Nuclear Physics, Genova, Italy, http://www.slac.stanford.edu/econf/C020909/rbpaper.pdf .

[10] Joseph, J. et al, "ATLAS Silicon ReadOut Driver (ROD) Users Manual", http://www-eng.lbl.gov/~jmjoseph/Atlas-SiROD/Manuals/usersManual-v164.pdf.

[11] The RD53 Collaboration, "RD53A Integrated Circuit Specifications", CERN-RD53-NOTE-15-001, August 29, 2015, https://cds.cern.ch/record/2113263.

[12] Sawyer, Nick, "Data to Clock Phase Alignment" XAPP225, February 18, 2009, http://www.xilinx.com/support/documentation/application_notes/xapp225.pdf .

[13] Xilinx Inc., Xilinx Kintex-7 FPGA KC705 Evaluation Kit, http://www.xilinx.com/products/boards-and-kits/ek-k7-kc705-g.html#hardware

[14] Flick, Tobias, “L1/L2 Review Introduction & HW Status”, University of Wuppertal, September 2015, https://indico.cern.ch/event/447710/contributions/1107382/attachments/1159576/1673191/Introduction.pdf

Acknowledgements

Over the two year span in which this work was done many people helped offered guidance and support making this thesis possible.

First I would like to thank my advisors Dr. Scott Hauck and Dr. Shih-Chieh Hsu. Their constant support and guidance in both the technical aspects of the project and the non-technical aspects of communication and organization were invaluable. Without them the opportunity to work on such a large and important project as the ATLAS experiment would not have been possible; nor would the side benefits of world travel. Experiences I am truly grateful to have had.

I would also like to thank Dr. Davide Falcheri, Shaw-Pin Chen, and Dr. John Joseph whose work on the IBL and Pixel DAQs was the foundation for much of this thesis. Through multiple conversations and collaborations with each I was able to build an understanding of Pixel DAQ firmware.

The work on the ITk upgrade described in this thesis was done in collaboration with Dr. Timon Heim and Dr. Maurice Garcia-Sciveres at Lawrence Berkeley National Laboratory. I would like to thank for both hosting me at LBNL as well as providing guidance and input on the RD53 emulator and DAQ.

In addition I would like to thank the numerous collaborators I worked with during my time at CERN: Dr. Karolos Potamianos, Dr. Laura Jeanty, Dr. Marcello Bindi, Luca Lama, Gabriele Balbi, Dr. Kerstin Lantzsch, Dr. Matin Kochain, Dr. Marius Wensing, Dr. Tobias Flick, Nick Dreyer, Dr. Kazuki Todome, and Dr. Federico Meloni. All of these individuals assisted me in understanding the operation of the Pixel DAQ system as well as the ATLAS experiment as a whole.

I must also thank my office mate Logan Adams for sharing his expertise of the Microsoft Office suite as well as providing insightful conversations and observational humor.

Finally I would like to thank my family: my mother Mary Mayer, my father Joe Mayer, my sister Elizabeth Mayer, and my brother-in-law Terrance Link. Their emotional and logistical support is the primary reason for my success in completing this thesis and obtaining a graduate degree.

Appendix A: Data Formats

A.1. ROD Formatter Data Words [7]

D(SubdetectorID) = 0x14, M(ModuleID)(1-0)=ROL ID (2 bits)

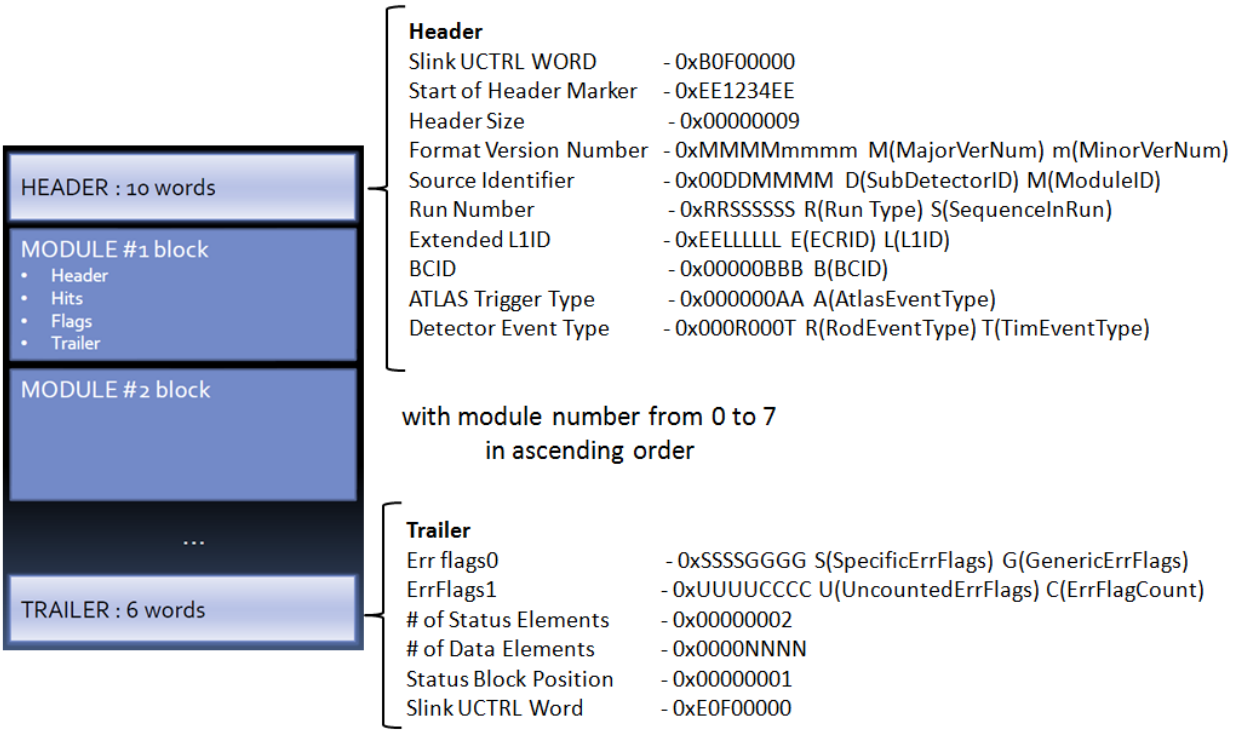
MODULE ## block	data_out[31:0] (to the router)
header	001nnnnnFLLLLLLLLLLLLLBBBBBBBBB
hit (long)	100nnnnnTTTTTTTTCCCCCCCRRRRRRRRR
hit (condensed mode)	101RRRRRTTTTTTTCCCCCCCRRRRRRRRR 1CCCRRRRRRRRTTTTTTTCCCCCCCRRRR 1TTTCCCCCCCRRRRRRRRTTTTTTTCCCC 111TTTTTTTCCCCCCCRRRRRRRRTTTTT
FE flag error	000nnnnnxSSSSSSxxxxxxxDDDDDDDDD
trailer	010nnnnnEcPplbzhvMMMMMMMMMBBBBB

n: link number
L: L1ID
T: ToT
R: row column
D: service code counter
c: condensed mode
M: number of skipped triggers
l/b: L1ID/BCID error
h: header/trailer limit error
F: FeI4B flag bit
B: BCID
C: hit column
S: service code
E: readout timeout error bit
P: link masked by PPC
p: preamble error (header error)
z: trailer timeout error
v: row/column error

A.2. Event Information Fields [7]

Event ID from the V5
word 0 : L1 ID[15:0]
word 1 : ECR ID[7:0] & L1 ID[23:16]
word 2 : BC ID[11:0] & RoL & BOC OK & TIM OK
word 3 : TT - ROD[5:0] & TIM[1:0] & ATLAS[7:0]
word 4 : Dynamic Mask 0 for Links [7: 0]
word 5 : Dynamic Mask 1 for Links [15: 8]
word 6 : Dynamic Mask 2 for Links [23:16]
word 7 : Dynamic Mask 3 for Links [31:24]

A.3. SLINK Event Packet Format [7]



Appendix B: Occupancy Tables

B.1. Expected MCC to ROD link occupancy for the outer three Layers and disks at 75kHz and 100kHz as well as 50ns and 25ns bunch crossing frequencies [14].

Link occupancy at 75 kHz L1 Trigger					
	μ	B-Layer	Layer 1	Layer 2	Disks
50 ns	37	39%	34%	52%	30%
25 ns; 13 TeV	25	35%	31%	48%	27%
	51	53%	59%	66%	39%
	76	71%	73%	111%	64%

Link occupancy at 100 kHz L1 Trigger					
	μ	B-Layer	Layer 1	Layer 2	Disks
50 ns	37	51%	45%	69%	40%
25 ns; 13 TeV	25	47%	42%	65%	37%
	51	71%	67%	88%	52%
	76	95%	97%	148%	75%

Note: The calculations for these numbers were obtained via simulation, and while close to the expected experimental values, are still a work in progress.