Rapid Synchronization Recovery from Single Event Effects in the Large Hadron Collider

Anatoliy Martynyuk

A thesis

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Electrical Engineering

University of Washington

2022

Committee:

Scott Hauck

Shih-Chieh Hsu

Abstract

The Large Hadron Collider will undergo an upgrade to become the High Luminosity – Large Hadron Collider after a long shutdown in 2024. This upgrade will significantly increase the particle collision rate which will result in more adverse radiation effects, called single event effects, in the electronics close to the collisions. To reduce data lost after a functional upset, the synchronization mechanism must be improved. A new synchronization system is developed and compared with the original in term of performance and hardware resource utilization. The final product shows enormous benefits in performance, reducing blocks lost to nearly 1% of the original with a moderate increase in FPGA resources to accommodate the improvement. The necessary background, design, decisions, and results will be outlined and discussed in this thesis.

Table of Contents

1.0 Introduction

The Large Hadron Collider (LHC) is currently the largest and most advanced particle accelerator laboratory in the world. Sitting a hundred feet below the ground, the LHC spans the length of 27 km (16.5) miles in a ring at the European Organization of Nuclear Research (CERN)[citation]. This particle accelerator sits across the French and Swiss border, outside of the historic city of Geneva. Its main function is to accelerate atomic and subatomic particles close to the speed of light, collide these particles and then record the result of the collisions. While this seems like a crude way to study physics, this method reached a monumental breakthrough in 2012, with the verification of the existence of the Higgs Boson resulting in the completion of the standard model of physics[citation]. Since then, the LHC has been used to explore new physics, beyond the standard model it had been built to support.

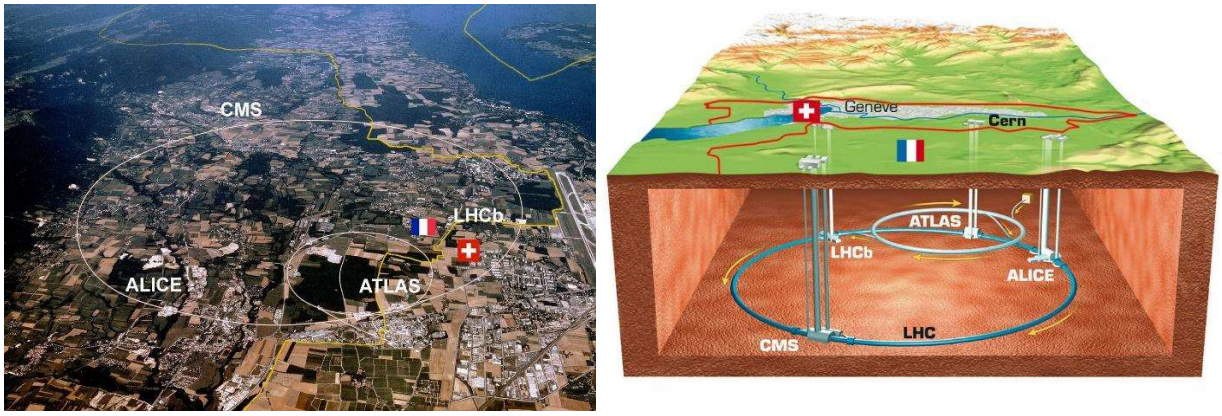


Figure X: Aerial view of where the LHC sits on the border of France and Switzerland. [citation]

The LHC, like other particle accelerators, utilizes powerful magnetic and electric fields and charged ions to reach high energies prior to collision. Hydrogen atoms are first sent through an electric field to be stripped of electrons, leaving behind positively charged protons. Being positively charged, protons will accelerate in electric fields and arc in magnetic fields, both of which are utilized to accelerate the protons in a circle. The proton steps through a series of acceleration stages through smaller rings with electrical and magnetic fields where it's joined with other protons into bunches of $1.1 * 10^{11} 1.1 * 10^{11}$ protons[citation]. The bunches, once at the correct energy level or speed, are split and injected into the LHC, with half going in each direction around the ring. The split bunches, called beams converge, and collide in one of four sites: ALICE, CMS, LHCb, and ATLAS. Each collision of energy generates mass which is recorded by sensor instrumentation in each of the collision sites, filtered, and sent through data acquisition modules to be stored and read[citation]. These collisions generate as much as 60 terabytes of information

every second, placing heavy demands on the sensing and filtering instruments within the collision sites [citation].

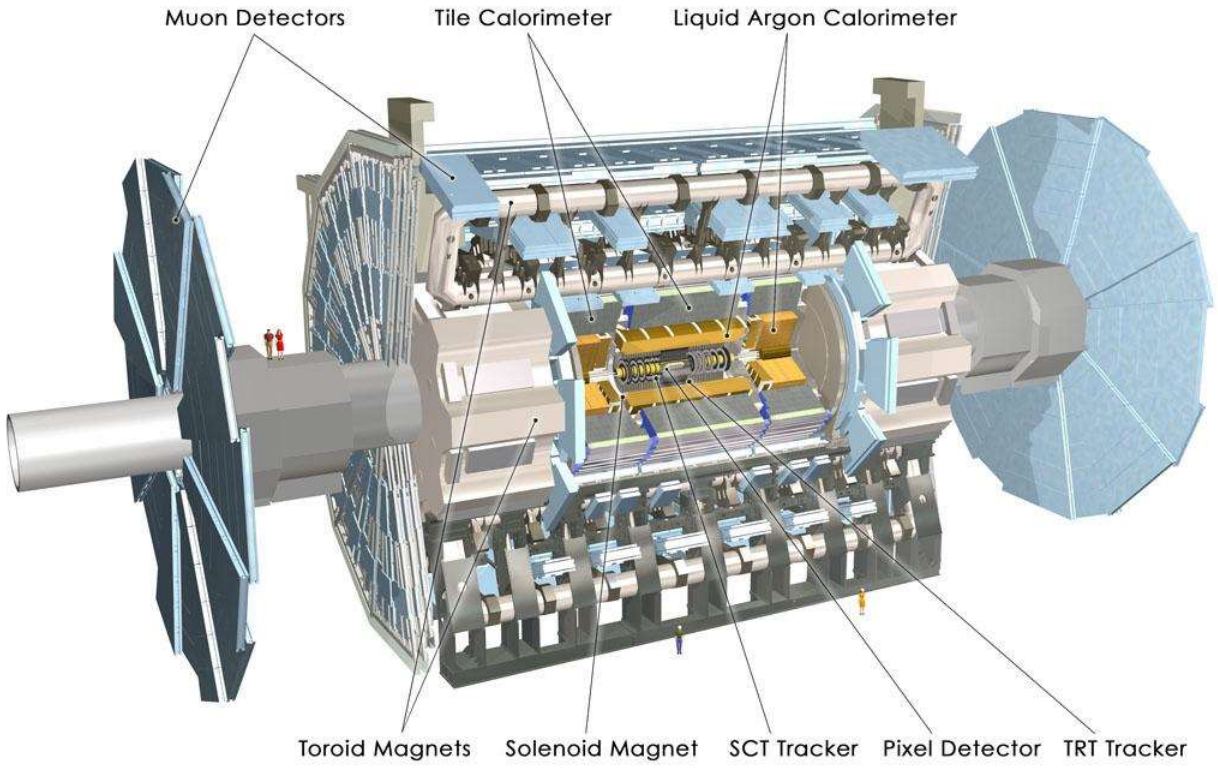


Figure X: ATLAS Pixel Detector [citation]

One of the collision sites at the LHC is the ATLAS (A Toroidal LHC ApparatuS). It is the largest general-purpose detector ever made for any collider and sits at 46 meters long, 28 meters in diameter, and weighs nearly 7000 tons[citation]. Figure 2 shows the detector to scale against people to demonstrate the size. The detector is built out of multiple layers or shells of detectors designed to record the trajectory, momentum, and energy of the individual particles seen during collisions. The four major components are the Muon Spectrometer, Magnet System, Calorimeters, and the Inner Detector. The Inner Detector is the innermost layer and is further broken down into its own layers with its own innermost layer being the pixel detector.

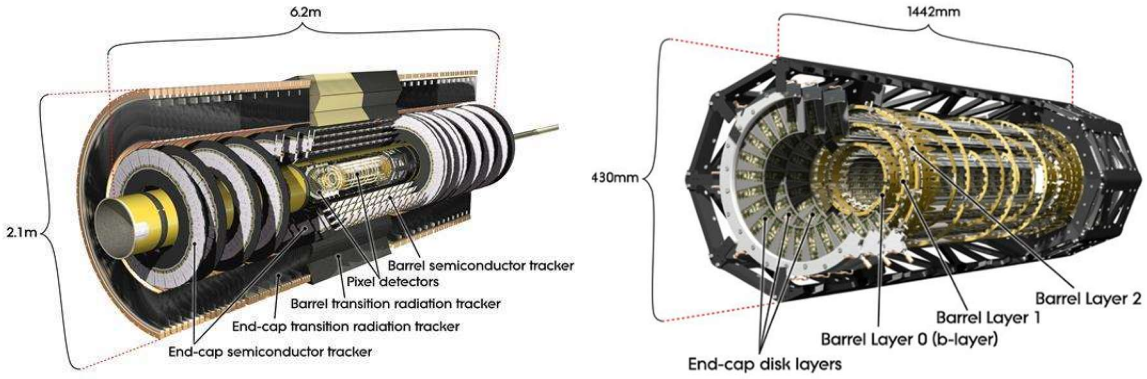


Figure X: The Inner Tracker (left) and within it is the Pixel Detector (right). [citation]

The Pixel Detector contains a silicon chip called the Front-End chip, whose basic operation is to convert charge from passing particles into a digital value[citation]. It also records the length of time this value was above a threshold value, referred to as the Time over Threshold (ToT). These values are bundled into data and passed through to a data acquisition (DAQ) module; in ATLAS's case this is the Yet Another Rapid Readout (YARR)[citation]. Unlike the Front-End chip, the DAQ modules are kept far away from the collision point, and further process the data before forwarding it to software databases for analytics.

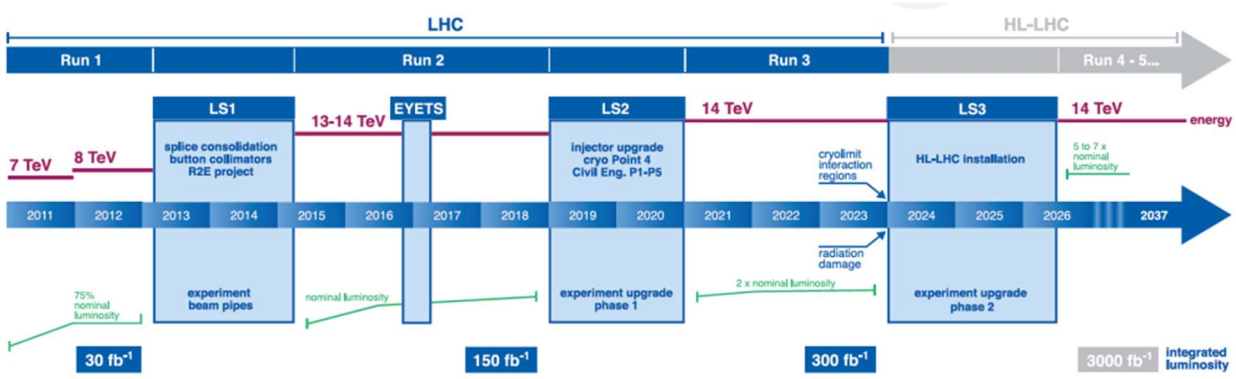


Figure X: LHC Shutdown and Luminosity timeline. [citation]

Since its opening in 2010, the LHC has undergone long shutdowns (LS) in 2013 and 2018 and is expected to undergo another in 2024. This upcoming LS aims to increase luminosity: the number of collisions per bunch by an order of magnitude, from 300 $fb^{-1}fb^{-1}$ to 3000. This new High Luminosity LHC (HL-LHC) will demand upgrades to pixel detectors within the ATLAS[citation]. Development for the inner tracker has been ongoing over the previous years, with the development of new silicon technology to support and tolerate the increases in occupancy, bandwidth, and radiation damage resulting from the increase in luminosity.

While the inner layer electronics are being updated to tolerate the increased radiation damage they'll experience, significantly more electrical anomalies are anticipated to appear within the Front-End chip. Toleration of these anomalies, as well as the mitigation of their consequences falls not only on the inner layer electronics but also on the upstream DAQs that not only filter and process the incoming data but will also need to detect and correct failures and corrupt data resulting from radiation effects on the electronics.

2.0 Single Event Effects

2.1 Background

Single event effects (SEEs) are a phenomenon resulting from interactions and collisions of high-energy particles with devices in integrated circuits. When a high energy particle passes through the silicon substrate of a device it generates charged particles along its path through a series of collisions. If the charges are generated at or near a transistor junction, the new charge can induce an upset in the transistor resulting in a change of state usually manifesting in a memory bit flip or a sudden spike in voltage or current. The particle itself can be charged, but usually, it is an uncharged particle, such as a neutron, which only begins generating this ionizing path after collision with a doped substrate[citation].

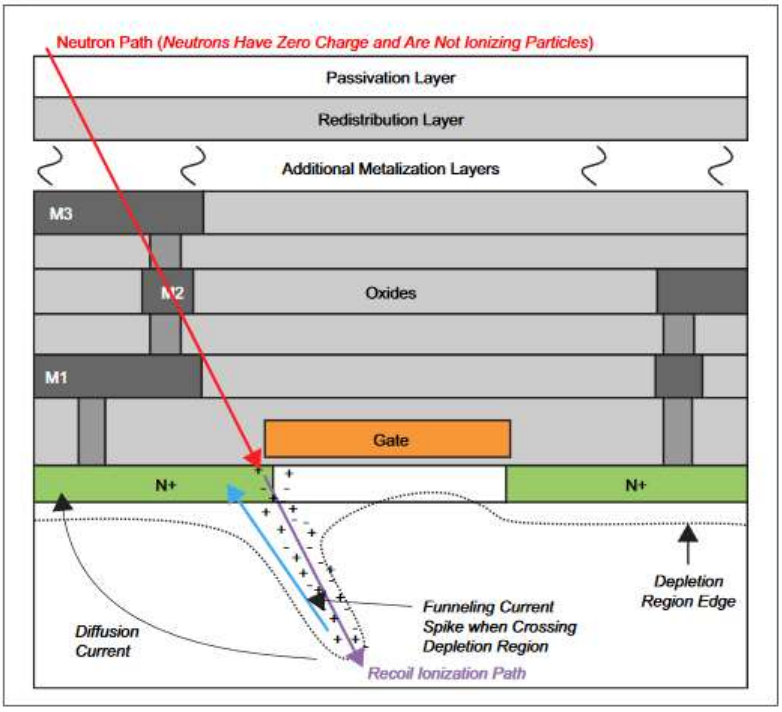


Figure X: High-Energy Neutron generating an ionizing path after collision. [citation]

SEEs are not a new problem in electronics but have been recorded as anomalies in electric equipment in nuclear testing from as early as 1954 [citation]. Additional instances of these apparently random errors and interruptions in otherwise perfectly operating ICs were observed in

space electronics. The first paper describing this phenomenon wasn't published until 1975 [citation].

These high-energy particles are present nearly everywhere. However, there are specific environments and sources where they become a genuine reliability concern. Galactic cosmic rays (GCR) from space are the most common source[citation]. GCR are made up of subatomic particles and light ions traveling at nearly light speed. While they might not necessarily directly strike ICs, they generate high energy neutrons through nuclear spallation resulting in air showers[citation]. If these neutrons retain energy, generally greater than 10 MeV, they can induce an SEE [citation]. The density or flux of these neutrons is dependent both on altitude and latitude. For example, a New Yorker would experience twice the neutron flux as someone in Singapore, whereas airplane passengers would experience 600 times the neutron flux the New Yorker would[citation]. Given that altitude has the greatest influence on the probability of an SEE occurring, mitigation and recovery are usually considered in space and aviation applications. Although the devices in the LHC are contained and shielded from atmospheric effects, the collisions the LHC is emulating generate an equivalent environment.

Single event effects are an umbrella term for several possible errors that can occur, categorized as soft and hard errors. Hard errors are errors that cause lasting or permanent damage, and generally can't be solved using logic techniques and will not be a focus of this paper. Soft errors are upsets to a device's operation but are self-correcting in time or are correctable. These can generally be described as one of two events: upsets and transients[citation].

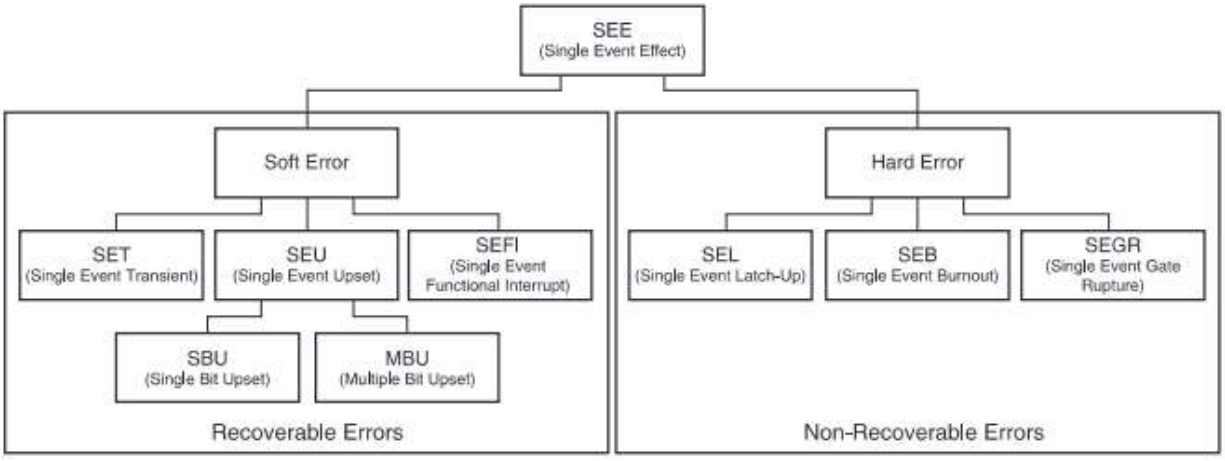


Figure X: SEE Categorizations[citation]

Single event upsets (SEUs) occur when high-energy particles impact transistors in memory cells, which result in an immediate bit flip. A single bit flip is referred to as an SBU (Single Bit Upset) and if multiple bits are affected it is referred to as an MBU (Multiple Bits Upset). If the bitflip occurs in a datapath that otherwise doesn't influence any system state registers, the flip propagates but ultimately is flushed out. However, if the bitflip modifies a state register, such as an FSM state, a control counter, or another register that doesn't naturally flush its contents, the

effects can be more severe up to requiring a system reset. Even worse, in FPGAs, is when the bitflip occurs within the configuration memory, resulting in LUT contents or routing to change, referred to as routing errors[citation].

Single event transients (SETs) result when high-energy particles impact a combinatorial path of an IC and result in voltage or current spikes on a wire. There are two instances where this results in an error. The first is when the transient occurs on any data or signal line leading to a clocking element. If the pulse width of the spike is wide enough and happens at the right time, the glitch can propagate through the circuit. The second is when the glitch appears on critical system signals such as a clock or a reset. Unless the signal trees for these are built to resist errors like this, SETs can result in extra transitions and unintended system resets. While transients don't directly influence memory cells the way SEEs do, their influence on data as well as on critical system wires can result in similar effects to SEUs or worse. When the consequences of either of the above noticeably freeze the regular function of the system, they are further categorized as SEFIs (Single Event Functional Interrupts) [citation].

SEEs are characterized using several metrics. Failures in time (FIT) is a measure of the rate at which events occur, where the unit of time is 10^9 hours. Mean time between failures (MTBF) is closely related but describes the average length of time a system is operational between individual functional upsets. The mean down time (MDT) is then the length of time for which a system interrupted or acted erroneously after an upset.

Since their discovery efforts have been made to mitigate, detect and or resolve single event effects. Solutions and recovery schemes exist to fit a variety of failures but come with tradeoffs. In mitigation or prevention systems the difference in FIT rates and mean time between failures are standard measures of efficacy. In recovery schemes, the detection time of a failure as well as the mean down time are the primary metrics. Both however also consider the costs of their implementation, including device performance and implications because of these added complexities, which often result in device resource, time to market, and monetary costs.

2.2 Triple Modular Redundancy

The simplest and most implemented hardware mitigation method is triple modular redundancy (TMR). TMR is a fault tolerance methodology that duplicates system units in anticipation of one failing. By having three duplicate copies of a component, usually a DFF or other memory storage, and a voter circuit, which forwards whichever value appears twice or more on its input lines, single errors can be automatically ignored (see Figure X). If any single DFF encounters a bit flip or other upset, the remaining two will hold a majority and will outweigh the single corrupt DFF. The corrupted DFF can eventually expect to be rewritten or refreshed to the correct value and will no longer be in contention with the other two.

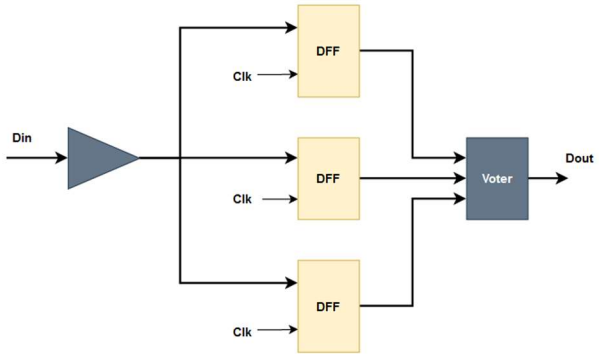


Figure X: Generic TMR (triple modular redundancy)

This version of TMR resolves SEUs, but SETs remain a threat. Since all three registers receive the same input and latch onto it at the same time, an SET can simultaneously corrupt all three of them. A relatively straightforward solution to this is just to add buffers or delays onto the clock input line of the DFFs. By setting the difference between each clock's arrival time to some time *t* (15ps in Figure X), the TMR would be able to mitigate any SET corruptions with glitch lengths of less than *t*.

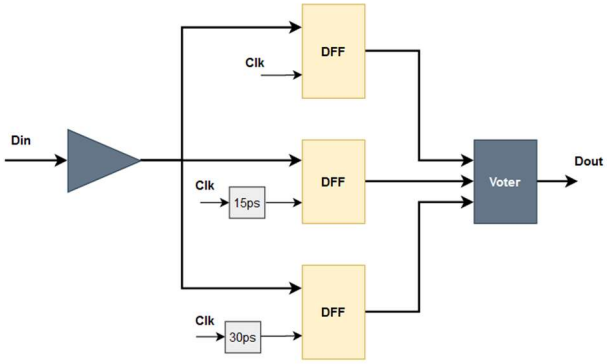


Figure X: TMR with delayed clocks to mitigate SET interrupts

In circumstances when the DFFs are updated infrequently, the TMR described above can simply delay a failure rather than mitigate it. In critical memory, failures of any kind, delayed or not, are unacceptable and so correction within the TMR registers is sometimes necessary. By adding a feedback loop (Figure X) from the voter output back into DFF input, the system can ensure not just mitigation but also correction, assuming only a single bit flips at a time.

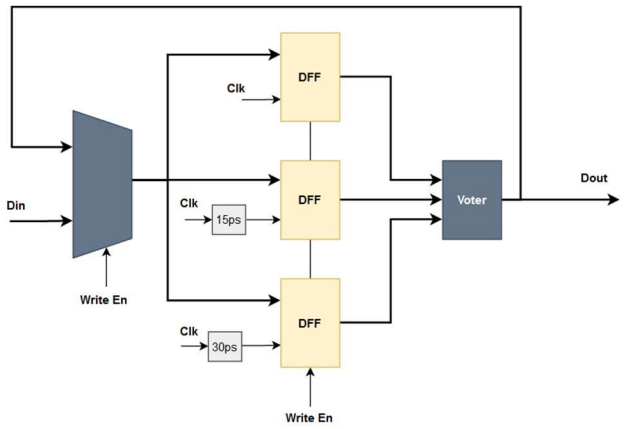


Figure X: TMR with voter feedback for correction

Additional measures could be explored to improve reliability in TMR. Spacing the duplicate memory components in hardware decreases the risk that two or more flip simultaneously to a multiple bit upset. Similarly, having the components wired to different branches of global signal trees prevents multiple duplicates from experiencing a transient effect. These both put constraints on the layout of the FPGA and may result in nonoptimal placement. Steps in the other direction can be taken as well. Triple modular redundancy can be substituted with dual modular redundancy in applications where mitigating upsets isn't as important as simply detecting them, or resource constraints limit the resources available for mitigation.

2.3 Application to the LHC DAQ

There are instances where none of the above is applicable, reliable, or perform at par with what a system requires. In these circumstances, application-specific detection, correction, and or mitigation systems must be developed from the ground up. Such is the case in DAQ systems of the LHC.

The circumstances of LHC DAQ don't allow for conventional SEE mitigation and recovery techniques. Unlike most electronics dealing with SEEs, the DAQs at the LHC experience very little radiation, practically none at all. This is because while the pixel readout chip is located directly at the collision site, the DAQ is bunkered 80 meters away and lies 100-meter underground[citation]. Instead, the DAQs inherit corrupted data from the pixel redout chips they communicate with and must resolve lasting communication interruptions that result from them.

The RD53B pixel redout itself is also hardened against radiation. However, current testing predicts that a significant number of effects are expected within chip memory. In protected and corrective memory an estimated 0.2 bit flips per hours whereas in the worst case as many as 230 flips per minute[citation]. Memory aside, the serialized data on the readout links have been shown to be especially susceptible to single event effects[citation]. It's this very sensitivity than the YARR DAQ aims to resolve through early detection and rapid recovery.

3.0 YARR Tx/Rx

3.1 Data Format and Protocol

To understand the SEE detection and recovery system, an understanding of the logic and data surrounding the transmitting end of the RD53 readout chip and receiving end of a DAQ is necessary.

At the highest level, the connection between the YARR DAQ and RD53 pixel readout chip is made up of one or multiple differential serial data connections. For simplicity, a simple case with just a single data line will be considered and is shown in Figure X. This connection has no control, handshaking, or reference clock signals but purely a single serial wire transmitting a bit at a time. Communicating serially without these additional signals poses several analog and logical challenges.

[Figure of an abstract block diagram of the communication]

To address these issues, communication uses a custom 64b/66b line encoding. The goals of the encoding are to support clock recovery, allow for data stream alignment, achieve DC balancing, and maintain standard transition densities and run length. Clock recovery was necessary in earlier iterations of the pixel readout chip but isn't any longer. Stream alignment allows the receiving chip to receive a continuous stream of 1s and 0s, yet still distinguish individual blocks within the stream. DC balance standards required that an even number 1s and 0s are transmitted over a data line. This is so that there is no bias of voltage which can have adverse effects on level detection thresholds[citation]. Transition density is the ratio of 0 to 1 or 1 to 0 transitions within the stream to the number of bits transmitted. Run length is the maximum number of consecutive 1s or consecutive 0s expected to be seen within the stream. 64b/66b line encoding does not enforce strict bounds on transition density or run length but imposes statistical bounds to expected density and run length.

In 64b/66b encoding, data blocks are 66b blocks and have two distinct elements. The two most significant bits in the blocks are the header, which is strictly a 01 or 10. This sets a maximum upper bound on run length and guarantees that one transition will occur every 66 bits. The remaining 64 bits are scrambled data. The data needs to be unscrambled on the receiver end to recover the original 64 bit payload. Although a custom variant was developed for the RD53 pixel readout, block codes, flow control and other features were borrowed from Xilinx's Aurora 64b/66b Protocol [citation].

3.2 Pixel Readout Tx Logic

The pixel readout chip can have multiple simultaneous data output streams; however, each stream utilizes identical logic to package and transmit the data. Each port is made up of 3 major blocks which transform the data from a parallel framed 66-bit block in the format given above, to a scrambled block serialized at high speeds across the communication link: the scrambler, the output gearbox, and the OSERDES.

[Image of a high level block diagram of the RD53 Tx Logic]

3.2.1 Tx Scrambler

To accomplish the DC balancing, transition density, and run length requirements set by the line encoding algorithm, the data being transmitted must be modified. Data stored in registers, and data produced from atomic collisions doesn't tend to already have the characteristics required. To transform it so that it does, the data is pushed through a scrambler. The scrambler takes a block of a set bit width and applies a function to it to produce a second block that is statistically likely to have a short run length, a high transition density and an equal number of 1s and 0s.

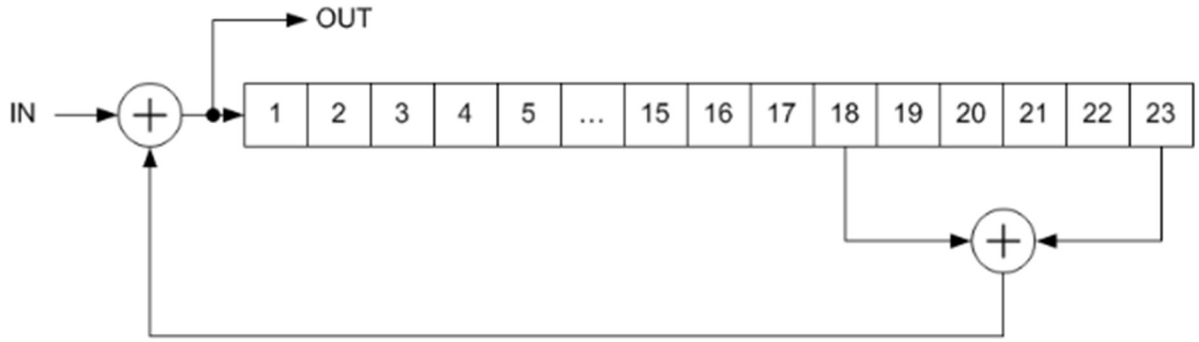


Figure X: [description] [citation]

Scramblers are described with a polynomial which tells you the tap values to use for feedback. Figure X shows an example scrambler with the function $1 + x^{18} + x^{23}$. While the scrambler looks like a serial operation, it can, with sufficient resources and time, evaluate a full 64-bit block simultaneously. One thing to note is that the operation has memory so that the generation of bits for the output depends on the previously generated bits and feeding the scrambler the same block continuously (for example an IDLE) will produce different outputs cycle by cycle.

The scrambler used by the RD53 pixel readout chip uses a $1 + x^{39} + x^{58}$ polynomial as per Xilinx documentation for its own IP 64b/66b protocol [citation]. This scrambler is multiplicative self-synchronizing meaning that this transmission scrambler and its receiver counterpart can be initiated at different times or with different state but still achieve synchronization. The scrambler in the RD53 receives a 64 block of payload data, which it transforms into 64 bits of scrambled data. This data is pre-appended with two header bits, which are not scrambled and passed forward to the gearbox.

Before moving to the gearbox, it may seem like scrambling the data to introduce more transitions will work counter the stream alignment requirement. With more transitions, how can the receiver be certain which transition is indicative of the header and which is simply well-scrambled data. However, since the scrambler produces essentially randomized data, each pair of bits, except the designated header, in the stream is 50% likely to not see a transition. Therefore, if we watch a position for *n* blocks, there is only $\frac{1}{2^n}$ probability of having a transition appear in all *n* blocks, unless of course it is the correct header.

[simple image depicting the gearbox – may not be necessary]

3.2.2 Tx Gearbox

Between the scrambler and the serializer is the gearbox. The gearbox receives the 2 bits of unscrambled header and 64 bits of scrambled data and packages them into 32-bit chunks to prepare them for the 32-bit serializer. The data is shifted out of the gearbox at twice the clock rate as is shifted in. Therefore, every input clock cycle, 66 bits are shifted into the gearbox and 64 bits are shifted out. As result of this input output size mismatch there is an accumulation of 2 bits every input cycle. To address this there is a pause every 33rd cycle to shift out the accumulated bits and allow the gearbox to “catch up”.

3.2.3 Tx Serializer

The final step in the output sequence is the serializer. The serializer shifts in 32 bit chunks and shifts out each serially, meaning a single bit at time. To accomplish this the serializer uses a faster clock as well as DDR data sampling to transfer data on both the rising and falling edges of the clock. The abstract behavior can be described as a parallel in, serial out register with a bit width of 32. Serializer parameters are given below.

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

Table X: [caption][citation]

3.3 DAQ Rx Logic

The Rx logic within the DAQ mirrors that of the pixel redout Tx. Each of the blocks used to implement the line encoding have counterparts to undo the encoding in the reverse order. Built into each block as well as some periphery, however, is additional logic to ensure that stream alignment is maintained. Just like with the Tx, multiple duplicate channels can simultaneously service multiple input lines, however for the simplest case we consider just a single channel.

[Image of the Rx Block Diagram]

3.3.1 Rx Deserializer

The receiving end deserializer is functionally very similar to the transmission serializer except that it performs the inverse operation. The deserializer receives serial data at a high rate which is accumulated into a 32-bit chunk and passed inward to the gearbox. Just like the serializer, the deserializer can be modeled as a serial in parallel out register. One key note is that the deserializer is able to rotate the positions of the bits in every 8 bits received. The 8-bit sets are then assembled into a 32-bit chunk. More on this feature is discussed in later sections. Just as in the Tx serializer, the deserializer's parameters are given below:

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

Table X: [caption][citation]

3.3.2 Rx Gearbox

Just like in the transmitter, the receiver gearbox's purpose is to perform a transformation between 32-bit chunks from the deserializer, back into the 66-bit RD53B block. In this case the reverse of the Tx side is accomplished. The gearbox shifts in two 32-bit chunks and shifts out a single 66-bit block along with a block valid signal every cycle. Although the behavior is similar to the Tx gearbox, understanding the Rx gearbox is vital to understanding the resynchronization scheme in place as well as the proposed replacement.

[Image for the functional behavior for the Rx gearbox]

The basic behavior of the gearbox is to shift stream data through a buffer. The deserializer produces and passes 32-bit chunks to the gearbox which is shifted into an internal 128-bit buffer every half cycle. On the side, a 66-bit block is shifted out every cycle, containing the original scrambled data and unscrambled header. Because of this 64-to-66-bit transform, 2 bits more are taken out of the buffer every cycle than shifted in.

[Image for the functional behavior for the Rx gearbox]

The 66-bit block on the output of the gearbox is selected by a sliding window. While the buffer is 128-bits wide, a select 66 bits are chosen every cycle for the output word. In the first cycle, bits 127 down to 62 of the buffer are chosen for the first 66-bit block. In the second cycle, bits 125 down to 60 are read out for the second 66-bit block. This pattern continues until bits 65 down to 0 are read out. The reason for this sliding window is because the gearbox has to make up for the 2 extra bits that are shifted out from the buffer than shifted in by adjusting the expected position of the next block.

[Image for the functional behavior for the Rx gearbox]

Every 33rd cycle there is a pause, and no 66-bit block is produced. Once the sliding window reaches bits 65 down to 0, there is no further it can go, and the gearbox has to wait for the buffer to fill up again through data being shifted in from the serializer. After the pause, the sliding window is reset to its initial position at bits 127 down to 62, where another 32 66-bit blocks are produced before the next pause.

3.3.3 Rx Descrambler

Of the 66-bit block produced by the gearbox, the two most significant bits, the unscrambled header, are checked for validity while the remaining 64 scrambled bits are passed through the descrambler. The Rx descrambler is the reverse of the Tx scrambler and therefore has identical properties to it. It is multiplicative and self-synchronizing and utilizes the same polynomial, $1 + x^{39} + x^{58}$, to derive the unscrambled data. A smaller example descrambler is shown in the figure below.

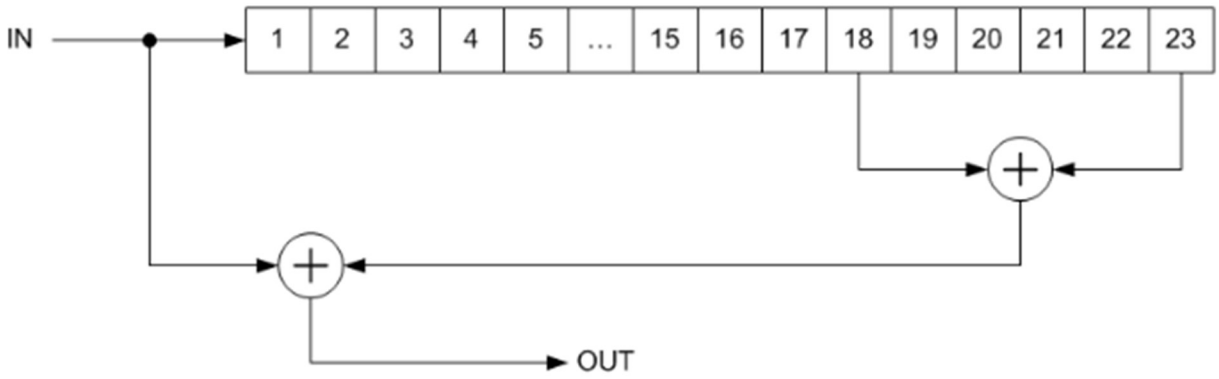


Figure X: [description] [citation]

4.0 Original Detection and Resynchronization Scheme

Built into the Rx logic of the DAQ is a set of control logic to verify and maintain stream alignment. Stream alignment is represented by ensuring that the two most significant bits of the 66-bit block read out of the gearbox have a transition within them, i.e. they are a valid 01 or 10 header. By shifting the stream relative to this reference point the Rx logic can scan through the data being streamed to find where this transition occurs consistently cycle by cycle. To verify the correct position has been found, 16 consecutive valid headers need to be seen. If an invalid header is seen at any moment after that the control logic must enter resynchronization and realign the header bits again.

In order to shift the stream data, the Rx logic utilizes a control logic FSM to allocate the "slipping" of the stream by leveraging features of the deserializer and in the gearbox and to determine when synchronization is obtained or broken.

4.1 Serializer Bitslip

The deserializer in the YARR DAQ is made up of a series of Xilinx primitives, including the ISERDESE2. Built into this primitive is a "bitslip module" whose original purpose was to lock onto a repeating pattern, called a training pattern, to ensure that the deserializer was ordering the received bits in the correct orientation. Since the slip effectively drops bits, the alignment control unit leverages this feature to shift the position in which the header bits appear. To keep up with the data rate required, the deserializer is operated in DDR mode, which has a somewhat complex bit slipping behavior. The bit slipping model can be depicted as a 15-bit buffer from which an 8 bit window decides the data that is read out.

[Image depicting the behavior of the bit slipping module]

Since the buffer available to the deserializer is limited in size, at most 7 bits can be dropped, and 8 header positions can be evaluated. Once all 8 are seen, the deserializer snaps back into its original position. Despite the back and forth slipping behavior of the DDR bit slip, the bit slipping generally shifts the data stream in a single direction, which results in alignment being recovered more quickly for stream bit drops due to SEEs over stream bit adds.

4.2 Gearbox Slip

8 bits of range is inadequate with block sizes of 66 bits, and so a second mechanism for slipping or shifting the data stream is required. The gearbox fills this role by taking advantage of its larger buffer where it can search a much greater range compared to the deserializer. The gearbox uses a sliding window which it uses to select 66 bits out of its buffer to produce a block. By allowing the alignment control FSM to actively move this window, the entire 128-bit buffer can be scanned over a series of cycles. Since the gearbox block selection window moves 2 bits each cycle to keep up with the uneven input out and output shifting, the control FSM can just freeze the window in place and prevent it from sliding. This has the equivalent effect of shifting the entire stream by two bits. For example, if the gearbox is incorrectly selecting bits 127 down to 62 in the first cycle, then it expects to incorrectly select bits 125 down to 60 in the second. However, if the gearbox instead selects bits 127 down to 62 in the second cycle, the gearbox will have moved the expected position of the header bits over by two bits.

[Image depicting the gearbox slip]

The gearbox slip has the benefit of having an unlimited bit range it can slip, given the time. However, it suffers from only being able to slip in a single direction, favoring stream bit drops, and slips two bits at a time, meaning that alone it would be unable to recover from a stream slip of 1 bit whether an add or a drop.

4.3 Block Sync FSM

The final piece and the one tying the last two together is the alignment control finite state machine (FSM) also referred to as the block sync FSM. This FSM has two purposes, to evaluate whether the system is synchronized or not and to allocates slipping to the two blocks capable of it. Both purposes are effectively achieved by tracking three counters: the gearbox data valid counter, the sync counter and the serdes bitslip counter.

[Pseudocode 1: Tolerance and synchronization requirement]

The gearbox data valid counter counters the number of gearbox data valid signals seen since the previous slip. The gearbox produces a data valid signal every time a 66-bit word is read out of the gearbox regardless of whether the header is considered valid or not. The gearbox counter’s primary purpose is to slow down the rate at which slips can occur. Following a slip, the

counter is reset, and it takes another 16 words, as well as an invalid header prior to seeing another slip.

The sync counter tracks the number of valid headers seen consecutively. The sync counter is incremented every time a 66-bit block from the gearbox has valid header bits and reset once an invalid header is seen, assuming the data valid counter has reached 16. To be considered synchronized with the pixel readout chip, the sync counter must see at least 32 valid headers. This is to ensure that an incorrect header position isn't locked on due to the statistical likelihood of an incorrect position appearing to have valid headers for several consecutive cycles.

[Pseudocode 2: Slipping Allocation]

When resynchronizing, the bit slip FSM is also responsible for allocating when bit slips happen. Whenever 16 blocks have been seen and an invalid header is read from the gearbox, the FSM signals that a bit slip is needed. It utilizes the serdes slip counter to set the ratio between serializer bit slips and gearbox slip and to determine when each should be executed. Since the serializer is capable of 8 slips before snapping back, this ratio is set to 8:1. This consequently results in 6 out of every 8 serializer bit slips to be redundant, since the gearbox slips only 2 bits.

4.4 Original Scheme Performance Evaluation and Results:

Evaluating the scheme comes down to determining the data lost as a result from interrupts and the speed of recovery from this error. As discussed previously, SEEs are described by the rate at which they occur (FIT), the average time between them (MTBF) and the average time for which a system is in recovery (MDT). Of the three, the first two are set by the pixel readout chip which is the source of errors. The MDT, mean downtime, is however set entirely by the response of the DAQ to an erroneous block, i.e., the resynchronization scheme. However, rather than evaluating the amount of time spent in resynchronization, which changes with the clock frequency, we measure by the number of blocks lost during resynchronization, which otherwise would have reached the DAQ and been processed. This value is portable to other systems which might have different clock frequencies, use a different size for their blocks or change other parameters of the system. It also has a much more practical implication for the consequences of an SEE occurring.

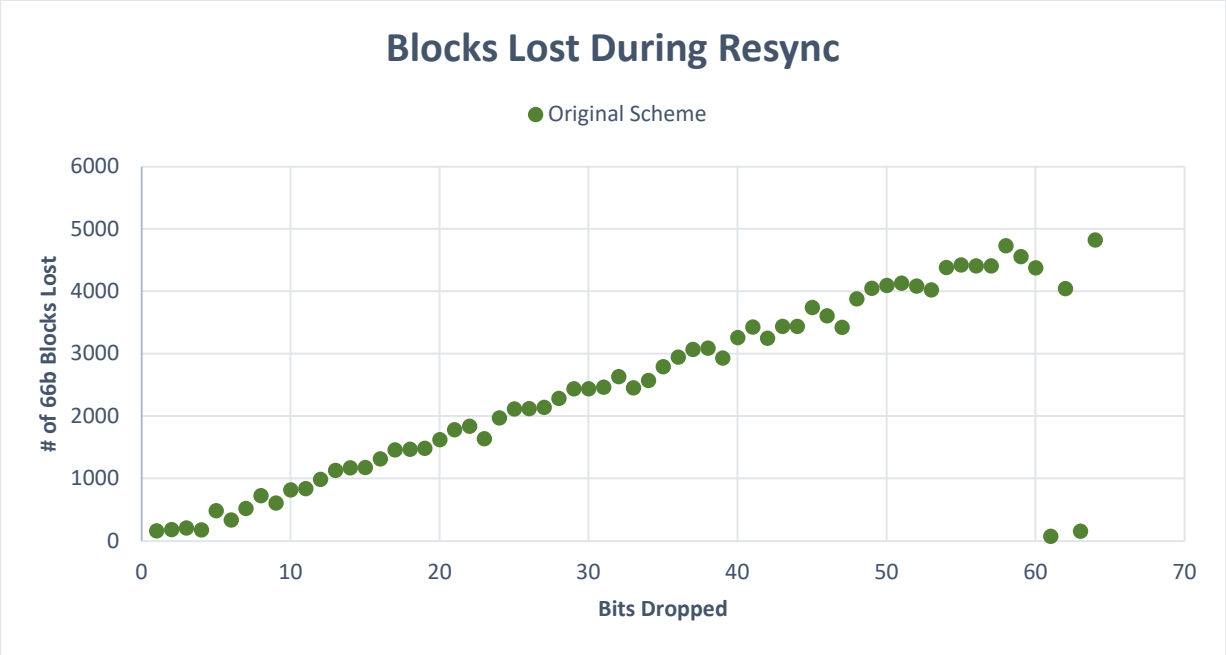
The primary measure for performance is a bit drop sweep. SEUs and SETs can be generally represented as either a bit flipped, a bit dropped, or a bit added to a block being transmitted to the DAQ. Bit flips will usually corrupt a data block without causing desynchronization, however when they do, they are equivalent to 66 bits dropped and a forced desynchronization because the header position doesn't move. A bit added to a data block results in the header bits of the following block

to appear a single bit later than the expected position and is identical to if 55 bits were dropped. Similarly, a bit dropped from a block results in the header bits appearing a bit earlier in the stream and are therefore equivalent to 55 bits added. Since an equivalency can be made to bit drops for each effect, the response to any of the effects can be described with the effective equivalent in terms of bits dropped.

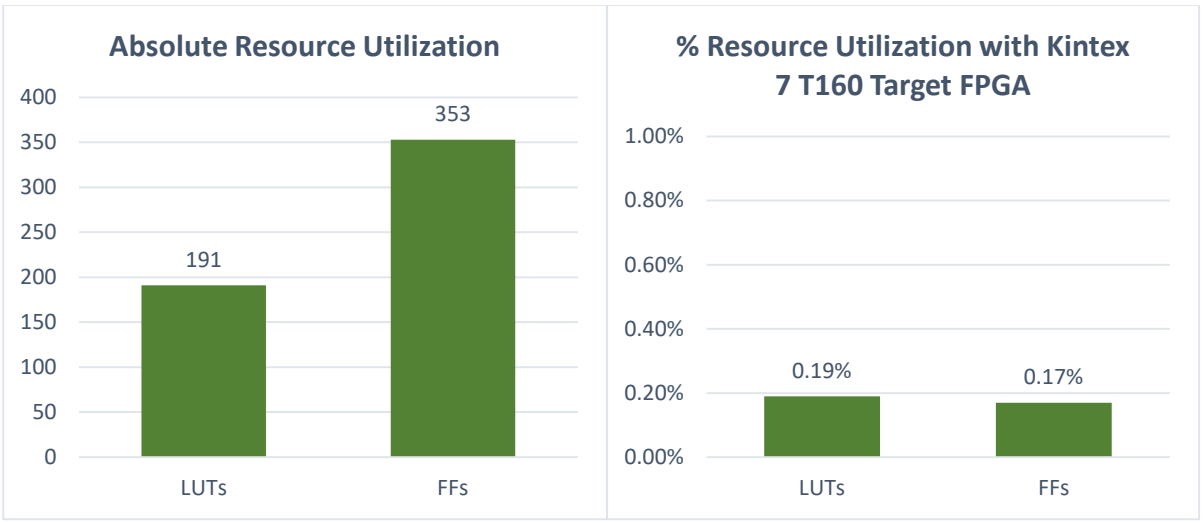
[image describing the above]

A bit drop sweep evaluates the response time of the resync scheme following a block where the *n* least significant bits of a block are removed. This value *n* is then swept from 1 to 65 to determine the resynchronization duration for any number of bits dropped, added, or flipped. Although bits can be dropped from or added to any point within the block, the new position of the header bits is the same regardless of the specific bits duplicated, removed, or flipped in the corrupt block. The only difference is whether the corrupt block is evaluated as valid or not by the system. In this sweep, the corrupt block is always considered lost, whether or not the DAQ system would have cleared it as a valid block, as well as any following invalid blocks which the system can still mistake for valid. This sweep is done purely in simulation utilizing Siemens Modelsim, an HDL verification software.

The primary measure of cost was the resource utilization of a Xilinx Kintex T160 FPGA. The primary constraints of the YARR DAQ design are the resources utilized as well as a 6.25 ns clock period. By targeting a Xilinx FPGA and using the Vivado design software passing timing constraints can be verified and resource utilization reports for different designs can be obtained and compared. The Kintex T160 FPGA was chosen because it is a likely candidate for the YARR DAQ used at the LHC. Since the resync scheme doesn't use any hard silicon resources, the comparison will be made purely with LUT and FF utilization.



The number of blocks lost while realigning the header bits under the original scheme is a linear trend against the number of bits dropped. This should come as no surprise, after all this scheme examines a single position at a time and checks position in a mostly unidirectional way. The exception is in bits 64 and 62, which are a consequence of the serializers bit slipping pattern. The nonuniformity or noise in the graph above comes down to random factors with the scrambling, where an incorrect position may be examined for longer than the minimum 16 bits because it appears to produce valid headers for several additional cycles. The amount of noise is expected to increase the more bits are examined and that appears to be expressed in the above. It should be noted that SYNC MAX was set to 16 so that there are a minimum of 16 cycles required before a new bit is checked or synchronization is declared which sits a minimum amount for blocks lost and plays an important role in the slope of the trend.



The resource utilization of the original system sets the bar for comparison against other resync schemes. The absolute resource utilization quantifies exactly the amount of LUTs and FFs used for the YARR RX, and is more portable to other FPGAs, however there may be variance if the FPGAs are produced through a different vender, are configured using a different software, or are simply a different generation within a vendor and use different primitives. The % utilization is a more intuitive quantification of the resources used but are specific to only the Kintex 7 T160 FPGA. Both above report the utilization for an entire channel rather than the logic directly related to the resync scheme, because the logic for resynchronization is distributed across multiple components.

5.0 Proposed Detection and Resynchronization Scheme

The evaluation and performance of the original recovery scheme left a lot to be desired. The two-driver alignment system is unnecessarily complicated and favors bit drops significantly more than bit adds. The allocation of bit slips between the two drivers is suboptimal. The individual header positions are checked for too long, and the criteria to gain and lose synchronization are ineffectual. A replacement recovery system developed improves on or removes each of these flaws and attains large performance boosts in most metrics.

That said, the replacement recovery scheme is constrained in the resources it can utilize. The RX logic described is only a small component of the DAQ, which is hosted on an FPGA with limited resources and a defined clock rate. While only a single channel is being considered, multiple channels are instantiated on each FPGA simultaneously and so resource utilizations need to be scaled to accurately represent utilization. Any new scheme would need to simultaneously maintain a low chip footprint and operate with a clock period of 6.25 ns.

5.1 Moving from Bit Slipping to Parallel Evaluation

The first issue addressed is the header alignment methodology. The original sync recovery scheme utilized two-bit slipping drivers, the gearbox and the serializer's bitslip module, to shift position of the data stream as seen by the DAQ. Since the DAQ only considers a single position where the header bits might be at a time, it must search 1 position at a time, and wait at each position for several blocks to stream through. Only after it has seen those blocks can it determine whether the header is there, at which point it declares synchronization or moves over a bit to check elsewhere. This method is slow and can be immediately improved by evaluating multiple possible header positions simultaneously. Checking n locations simultaneously should reduce the time to find the correct header location equally by a factor of n. If n is chosen so that every possible position can be evaluated simultaneously, the correct header position can be evaluated very quickly. It would only be limited by the time it takes to verify that all other positions are incorrect by counting the number of consecutive valid headers in each possible position. However, to accomplish this, the system must retain additional bits from the stream as the original scheme's 128-bit gearbox buffer stores just 1 possible header position in its worst-case state.

To track every possible header position, the replacement scheme expanded the gearbox buffer to allocate 67 bits for the evaluation of 66 possible header positions. 67 bits are required because each block is 66 bits wide, and the header is expected to appear in the top two bits of the block. Since the worst-case scenario has 65 bits dropped, or equivalently 1 bit added, not only do the top two bits of one block get considered, but also the 65 bits of the previous block, resulting in 67 bits checked. If 66 bits were dropped, that would be a whole block lost and the header would appear exactly where expected, albeit a block ahead of expected. The only caveat is that in the case

of a bit(s) add, the header would not appear in the window of possible positions, until the next block is shifted in.

[image demonstrating the above paragraph]

By tracking every bit where the header might appear, the serializer and gearbox can be simplified to their primary purposes. Since every possible position is known to the new resync scheme, bit slipping is no longer necessary. One benefit of this is that the blocks that had been responsible for bit slipping, the gearbox and serializer, can be simplified and reduced to their core functions. This has not only the benefit of simpler logic, easier maintainability, and more flexibility, but also removes some of the flaws associated with their slipping mechanisms. In the replacement scheme, the only purpose of the serializer is to transform a serial stream into 32-bit chunks. The only purpose of the gearbox is to transform 32-bit chunks into 66-bit blocks. This does however mean that a new aligner block must be developed to evaluate the header positions and guide the gearbox window to read the correct 66-bit block.

The naïve implementation of the aligner is to simultaneously evaluate each possible header position. A count of the consecutive valid could be maintained for each possible position and the position with the largest count would be determined to be the correct header position. While effective and fast, this method suffers two main hurdles. First, comparing 66 values and determining the greatest demands an enormous combinatorial path which is unachievable in the clock period required. To resolve this pipelining is necessary which complicated the design. The second is that the combinatorial resources, the pipelining costs associated results in an enormous number of resources utilized, which violates the second of the two constraints assumed. Instead, a tradeoff is made between performance and resource utilization, with the final replacement scheme still being significantly more attractive than the original scheme.

[image demonstrating the above paragraph?]

5.2 Six Seeker Aligner

The 6 header seeker aligner (HS6) utilizes fail-fast principles to minimize hardware costs while maintaining similar performance. If the aligner can quickly recognize that a bit position does not contain a header, it can ignore it and look elsewhere. With multiple seekers, each with a unique set of bit positions to search, the aligner can evaluate all 66 possible positions while only evaluating a few positions simultaneously.

This 6-seeker aligner starts resembling a partially parallel header search, with 6 positions evaluated, and 66 header positions available for evaluation. Just like in the naïve implementation a count of the number of consecutive valid headers seen is maintained for each position. However, as soon an invalid header is seen, the seeker changes the position it is evaluating. By distributing the 66 positions possible between the 6 seekers, each seeker rotates between 11 positions. Due to the randomness of the scrambler, every incorrect position still has a 50% chance of seeing what appears to be a valid header. However, the probability of seeing multiple accidentally valid headers drops exponentially and we can reasonably expect that a seeker will not usually get stuck at any incorrect position for more than a few blocks.

[image demonstrating the above paragraph]

While parallelizing the search is valuable, the fail-fast principle at work offers the most significant benefits to performance. A key feature of this search mechanism is that there is no tolerance whatsoever for invalid headers. As soon as a seeker reads a 11 or 00 it rotates to the next position it is responsible for. The original recovery scheme also had no tolerance, however, this only kicked into effect after having seen 16 blocks pass through, meaning that even if the first 16 showed invalid headers, the recovery scheme would not move on to search the next possible position until after the 16^{th} word was seen. The original scheme has the exact same probability of determining that the current position is invalid after the 17^{th} block passed through as the seeker has on its 1^{st} block.

Failing fast doesn't come without consequences, however. If the stream were prone to bit flips, where the correct header position doesn't change but an incorrect header is seen for one block, the scheme could be constantly pushed into recovery mode. Luckily that can be ignored for this application since the ratio of erroneous blocks to valid ones is vanishingly small. Further having a tolerance built-in would have additional negative effects on performance. The increased tolerance works against the fail-fast methodology used since allowing invalid headers to be seen along with consecutive valid ones increases the time spent evaluating an invalid position.

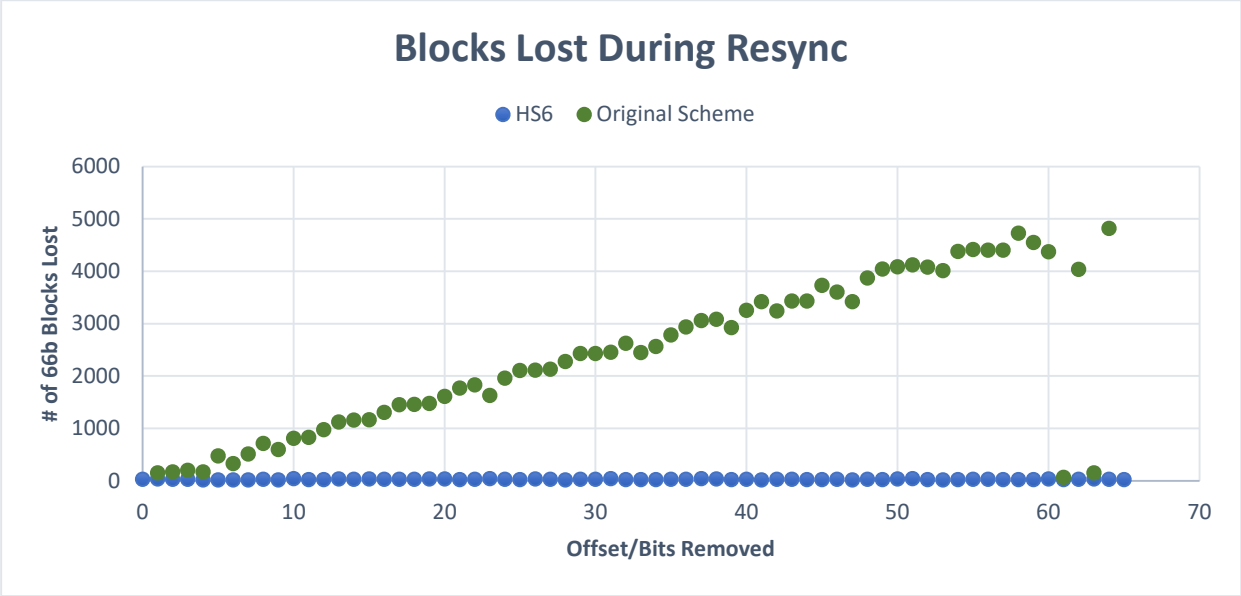
Any factor of 66 could have been used for the seeker count, however, 6 seekers provided the best performance-cost product among those tested. Since 66 positions encompass all possible header positions, a factor of 66 is used for the seeker count so that the positions can be split evenly among them. The most attractive counts were 11, 6, and 3 as they appeared to have the best balance of performance seen against resources utilized. To determine the best of the three, the minimum product of the blocks lost to recovery and resources utilized on an FPGA was used.

[table showing results of the above paragraph]

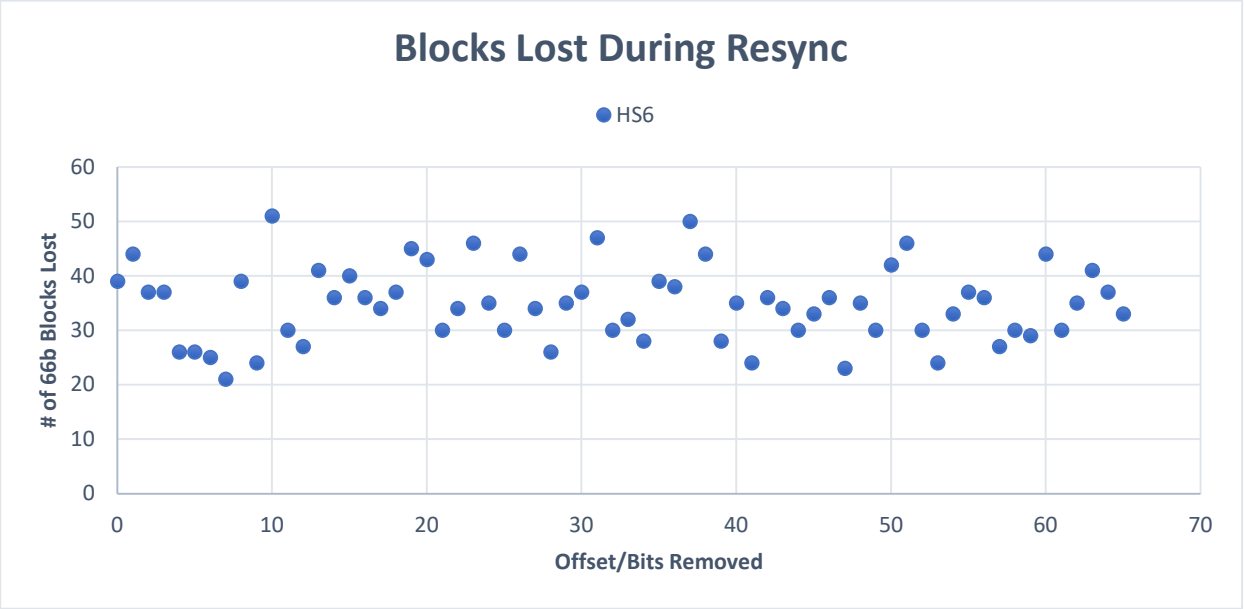
With bit slipping done away, and tolerance handled by the aligner, the responsibilities of the block sync FSM are reduced. This immediately resolved some of the major issues within the FSM, and, just like with the gearbox and serializer, improved the maintainability and testability of the logic. The remaining purpose of the block sync FSM is to determine whether the RX is synchronized. This is achieved by tracking the number of consecutive valid headers seen up until it reaches a configurable value SYNC MAX. This purpose of setting a minimum number of consecutive cycles is so that invalid data isn't labelled as valid if an incorrect position happens to see a few consecutive correct values. The value defaults to 16, and as a result guarantees that at least 16 blocks, aside from the first corrupted one, are lost during resynchronization. All other blocks lost are a result of the seekers attempting to find the correct position and having its consecutive valid count pass all incorrect position counts.

5.3 Proposed Scheme Performance and Results

The performance improvement made from the original resynch scheme to the proposed replacement is drastic, and a comparison of the two across the same sweep is shown below.



While the direct comparison is not especially informative of the header seeker 6 resynch scheme, it demonstrates just how effective the fail-fast and parallel search are to recovery performance. A separate performance graph for the header seeker 6 scheme is shown below.



The header seeker 6 recovery system differs in both scale and shape. The first and most obvious difference, illustrated in the comparison graph, is that the proposed replacement scheme operates with roughly 2 orders of magnitude fewer blocks dropped during resync. The primary reason for this is that the SYNC MAX constant was now a constant added to the blocks lost rather than a multiplier applied to the number of positions searched. The behavior of each sweep can be roughly described by the formulas:

$$Orig\ Blks\ Lost = (SYNC_{MAX} + AccidentalValid_i) * PositionsSearched_{\forall i} + AccidentalValid$$

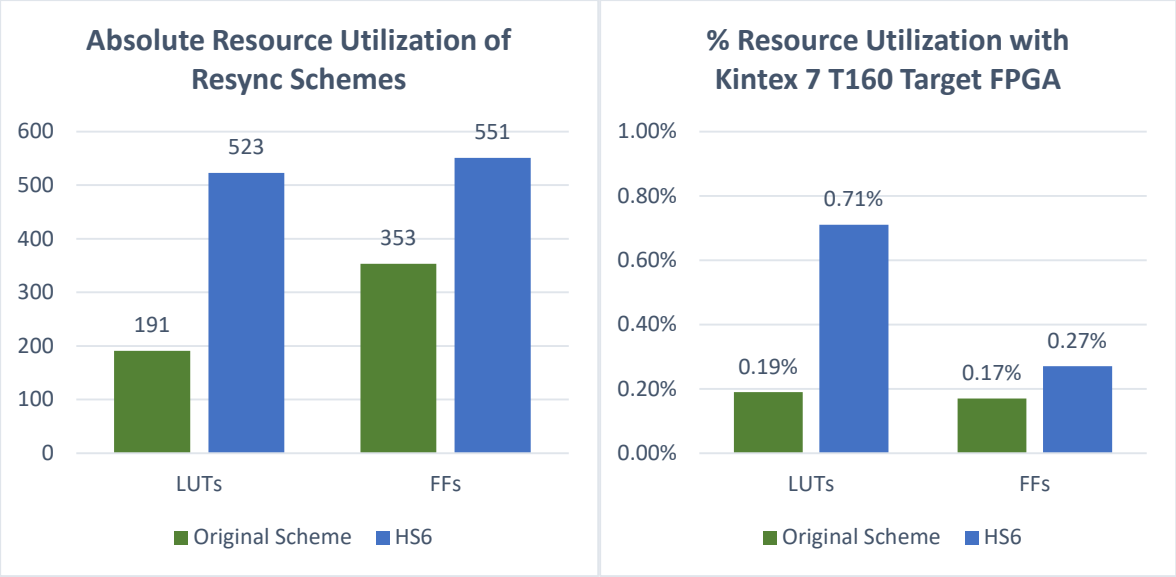
$$HS6\ Blks\ Lost = AccidentalValid_i * PositionsSearched_{\forall i} + SYNC_{MAX}$$

AccidentalValid is a random variable for each position rooted in the fact that headers will occasionally be counted as valid for some consecutive cycles even though they aren't because the scrambler randomized the data bits. For the original system, these are first a constant added and count as the time it takes for the system to realize it has lost sync. Then they also appear as a factor because each position searched can have a sequence of accidentally correct header values. In the header seeker 6 implementation, the aligner is constantly searching all bit positions whether the system is considered in sync or not which erases this term from the constants added. However, this is unavoidable when evaluating a position for whether it's a header or not and so it still factors in as a multiplicative term with the number of positions searched before finding the correct one.

The primary difference, as briefly discussed earlier, is how *SYNC_MAX* factors into each equation. In the original system it's a multiplicative term because it is applied to every position searched. Removing this term from the product and adding it back as just a constant is the greatest contributor to the performance of the HS6 scheme.

PositionsSeached also plays in even though it appears to be identical in each equation. For the original system, the worst-case value is 66. Being clever about what positions are searched first and last can result in the vast majority of headers being aligned within 7 positions checked and is accomplished by starting the search around bit 63 rather than 0 so that the neighboring 3 bits are checked first, which is where the vast majority of misalignments will be. In the HS6 scheme, *PositionsSearch* has a worst case of 11, which is brought on by parallelism. This factor of 6 improvement contributes significantly to the performance, but once again is overshadowed by the fail-fast principle.

The trend, having accounted for the noise, resembles a flat line. This might seem unexpected, considering we don't have full parallelism and the seekers have to sweep through 11 positions the same way the original system swept through 66. However, the difference is really the result of the more passive nature of the HS6 recovery scheme. In the original system, recovery only started when the system was found to be desynchronized and stopped when synchronization was achieved again. In HS6 however, the recovery scheme is always operating and so the positions being evaluated at the moment synchronization is lost is effectively random.



The resource utilization of the HS6 scheme is considerable relative to the original system, however its performance improvements dwarf this cost. The LUT count increased to over three and a half times the original utilization and solidified as the relevant cost in the FPGA over the FFs. While 0.71% still sounds like a negligible amount, each YARR DAQ can have multiple channels so this value would be multiplied, for example, by 4 for a four-channel DAQ. Even then however the total LUT utilization remains under 3% making it sufficiently tolerable except for the most logic dense devices. HS3, HS2 and even HS1 schemes can be considered for further reduction

in resource utilization, however their cost in terms of performance is more significant compared to the resources saved.

6.0 Conclusion

In anticipation of the upcoming HL-LHC upgrade, hardware and firmware is being upgraded to support the increase in luminosity. While the collision points are being upgraded to support the increase in data resulting from the upgrade, luminosity is correlated to radiation experienced and an increase in single event effects is expected. While the RD53 pixel readout chip will be the device experiencing these single event effects, it is anticipated to forward corrupted data in the form of bit flips, adds, and drops within its data block to DAQs such as the YARR. This thesis investigated the current recovery response scheme built into the YARR DAQ and proposes a replacement system to improve performance by reducing time in the form of blocks lost while unsynchronized.

Synchronization, or header alignment, is done by shifting the data stream so that two prefix header bits of each block appear where the RX expects to see them every 66 bits. When the headers move relative to the expected position, synchronization is lost, and the recovery system starts to search for them. The original system leveraged two bit slipping mechanisms and allocated their execution through an FSM to search for the header bits one position at a time. The replacement scheme, called header seeker 6 aligner (HS6), utilizes additional memory and parallel evaluation to accelerate the recovery process significantly at the expense of additional hardware resource utilization.

The performance and cost of each recovery scheme was evaluated through simulation software and FPGA configuration tools. When targeting a Kintex 7 T160 FPGA the original system had mean down time of 2406.5 blocks lost and utilized 0.19% of LUT resources for every operating channel. The HS6 scheme targeting the same FPGA had a mean down time of just 34.8 blocks but utilized 0.71% of LUT resources. This comparison from the original or the HS6 recovery scheme shows a decrease to 1.45% of blocks lost during resynchronization and an increase to 374% of resources used. While this increase in resources appears to be significant, in a four-channel DAQ only 2.84% of resources are allocated to the RX and recovery system. These results serve to allow users to determine which of the two schemes is a better fit for their design.

7.0 Future Work

As with most designs, the HS6 alignment and recovery scheme is only a base from which more improvements can be developed, and additional tradeoffs can be made.

There are multiple variations with different levels of parallelism to trade performance for hardware resources. As hinted throughout the paper, there are variations and alternatives which provide different tradeoffs in terms of performance and resource costs, as well as design maintainability, complexity, and other engineering costs. The sync recovery scheme chosen and designed in this paper, header seeker 6 alignment, was chosen because it produced the minimum blocks lost, resources utilized product, in other words it offered the best bang for buck deal. However, in other applications, resources could be significantly more expendable and lower down times more attractive, which would result in greater parallelization variations being a better fit such as HS11, HS22, HS33 or even full parallelization. The inverse could be true as well in which case less parallelization would be preferred.

A pair of parameters lightly discussed but not justified are the tolerance and synchronization requirements of the system. In this design the tolerance for the original system and HS6 is 0, meaning desynchronization occurs immediately after an invalid header is seen. The synchronization requirement is SYNC_MAX, which sets the required consecutive valid headers seen to consider the design synchronized and allow data blocks to pass further into the DAQ. Both of these parameters are rooted in statistics involving the likelihoods of an incorrect header position seeing multiple accidentally correct values in a row. A deeper evaluation of the statistics could result in an optimal value for each parameter, depending on the application. This is a non-trivial relative improvement for both schemes presented and comes at a negligible cost in hardware resources.

Both aforementioned improvements could be unified into a single parameterizable alignment unit. The logical similarities between each header seeker aligner variation are close enough that the design could be relatively straightforwardly be transformed to be easily configurable before being synthesized and loaded. This would allow the design to be simplified and easily portable to different DAQs and apply to their various constraints and tradeoff preferences.

8.0 Acknowledgements

I would like to thank all the brilliant people I have worked without throughout these years which led to this project as well as the help I received throughout it.

I would first and foremost like to thank Professor Scott Hauck and Professor Shih-Chieh Hsu as my advisors and mentors throughout my entire time at UW and especially during this thesis project. Scott Hauck was a professor of mine who through a single course convinced me not only to completely change the concentration of my Electrical Engineering degree but also to join a lab and strive for a masters. I had never expected to enjoy a subject as much as I did hardware design and am very fortunate to have fought my way into his competitive class in my very first quarter at UW. I remember on my first day I was even offered money for my seat in it. I offer my apologies to Shih-Chieh who I had first assumed was a student like me in the lab. His expertise in the physics and logistics should have tipped me off much sooner and I was always impressed as physics was a favorite subject of mine in my lower-division coursework. I also really appreciate the team they put together and continue to evolve that is the ACME Lab.

I would also like to thank Timon Heim, who has been an enormous help in this thesis project. He was the one who first proposed this project and has also been my go-to source for answering a myriad of questions and resolving confusion as I studied and developed the project. I also appreciate the resources and opportunities that were always available through him.

I would like to thank each person in the ACME Lab, past and present, who guided me with relatable experiences and student and industry perspectives. Special thanks go to Geoff Jones, who ultimately convinced me that VHDL is the superior language and has unintentionally imparted a significant amount of my FPGA knowledge through our various conversations. I'd also like to thank my predecessors and colleagues, past and present, Donovan Erickson, Matt Trahms, Lauren Choquer, Amelia Dumovic and Sanjukta Roychoudhury who I had close contact with throughout my time at ACME Lab and have been great friends.

Finally, I want to give a thanks to my friends and family who have motivated me to succeed in a relatively unconventional path and drove me to do my best. It means the world to hear that my relatively unconventional path is held in such high regard to you all. It's through your encouragement, support, and patience that I am able to accomplish what I have.

References

[1] ”CERN Website”, CERN, [Online], Available at https://home.cern/

[2] “CERN Atlas Website”, CERN, [Online], Available at https://atlas.cern/discover/detector

[3] McMahon, Stephen; Pontecorvo, Ludovico, “Technical Design Report for the ATLAS Inner Tracker Strip Detector”, CERN, April 1, 2017, [Online], Available at https://cds.cern.ch/record/2257755?ln=en

[4] “High Luminosty LHC Project”, CERN, [Online], Available at https://hilumilhc.web.cern.ch

[5] “The RD53B-ATLAS Pixel Readout Chip Manual v2.18”, CERN-RD53-PUB, January 2022

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

[7] “Development of an FPGA Emulator for the RD53B Chip”, N Mittal, S. C. Hsu, S. Hauck, 2020

[8] “YARR Documentation”, CERN, [Online], Available at https://yarr.web.cern.ch/yarr/

[9] “Single Event Effects (SEEs) in FPGAs ASICs, and Processors, Part I”, D. White, Xilinx, [Online] Available at https://www.eetimes.com/single-event-effects-sees-in-fpgas-asics-and-processors-part-i-impact-and-analysis/

[10] “Single Event Effects (SEEs) in FPGAs ASICs, and Processors, Part II”, D. White, Xilinx, [Online] Available at https://www.eetimes.com/single-event-effects-sees-in-fpgas-asics-and-processors-part-ii-mitigation/

[11] “Mitigating Single-Event Upsets”, J. Hussein, G. Swift, Xilinx, May 2015

[12] “Considerations Surround Single Event Effects in FPGAs, ASICs, and Processors”, D. White, Xilinx, March 2012

[13] “Introduction to Single-Event Upsets”, Altera, 2013

[14] “Neutron-Induced Single Event Upset (SEU) FAQ”, Microsemi, August 2011

[15] “RD53 SEU/SET Protection Scheme”, T. Hemperek, 2022, [Slide Deck], Available at https://indico.cern.ch/event/1121927/

[16] “SEU/SET Tests with Ions and Protons and Scaling to System Rates”, M. Menouni, 2022, [Slide Deck], Available at https://indico.cern.ch/event/1121927/

[17] “SEE Chip/System Estimates for RD53B/C”, CERN, 2022, [Slide Deck], Available at https://indico.cern.ch/event/1121927/

[18] “Aurora 64B/66B Protocol Specification v1.3”, Xilinx, October 2014, [Online], Available at https://docs.xilinx.com/v/u/en-US/aurora_64b66b_protocol_spec_sp011