Developing Firmware and Algorithms for the Liquid Argon Signal Processor

(Peter) Xiangyuan Ma

University of Toronto,
Toronto Ontario Canada
McGill University,
Montreal, Quebec Canada

E-mail: peterxy.ma@gmail.com

Abstract: In this report we discuss the development of various firmware and algorithms for the digital electronics of the Liquid Argon Signal Processor (LASP), which is designed to measure and reconstruct the energy deposited into the Liquid Argon Calorimeter cells. We first examine the development of firmware for PATGEN, TTCGEN, 10Gbe Base R/KR network protocols. Then we explore the development of novel machine learning algorithms for optimal energy reconstruction given the digital current signals. Lastly, we investigate various hardware tests to examine the functionality of the future electronic boards.

Contents

1 Introduction

The ATLAS (A Toroidal LHC ApparatuS) detector is one of the four main particle detectors at the Large Hadron Collider (LHC), which is the world's largest and most powerful particle accelerator located at CERN (the European Organization for Nuclear Research) near Geneva, Switzerland.

The main purpose of the ATLAS detector is to observe and study the particles produced by high-energy proton-proton collisions in the LHC. This detector works by detecting various kinds of particles in 4 main sections.

Firstly, the innermost part of the ATLAS detector is the tracking system, which consists of semiconductor-based detectors like silicon strips and pixels. When charged particles pass through these sensors, they ionize the material, creating electron-hole pairs that generate electrical signals. These signals allow the tracking system to reconstruct the paths of charged particles, determining their momentum and charge.

Next we have the Electromagnetic Calorimeter (ECal) which is designed to measure the energy of electrons and photons. When these particles pass through the ECal, they interact with its dense material, producing electromagnetic showers. The energy of the original particle can be measured by the the ionization signal inside the calorimeter.

Then surround that we have the Hadronic Calorimeter (HCal) which is used to measure the energy of hadrons, such as protons, neutrons, and mesons. Hadrons interact strongly with matter and create hadronic showers when they pass through the HCal. The energy of the original particle is measured by the total amount of hadronic activity in this calorimeter.

Lastly, we have the Muon Spectrometer. Muons are particles that can penetrate through several layers of the detector due to their weak interactions with matter. The outermost part of the ATLAS detector is the muon spectrometer, which consists of large magnet systems and dedicated tracking chambers. It is used to precisely measure the trajectories and momenta of muons, which are essential for various physics analyses, including the discovery of the Higgs boson.

Together with these sub-detectors, the ATLAS detector provides a comprehensive view of the particles produced in LHC collisions, allowing physicists to explore and understand the fundamental constituents of matter and the forces that govern them.

1.1 Liquid Argon Calorimeter and Electronics

The Liquid Argon Calorimeter is specifically responsible for detecting and measuring the electromagnetic showers produced by electrons and photons as they interact with the detector's material. These showers are cascades of secondary particles generated when high-energy electrons or photons lose energy.

The active medium, liquid argon, is an ionizable material. When charged particles, such as electrons or positrons, enter the liquid argon they ionize the atoms along their path, producing charged particles (electrons and ions) within the LAr.

The ionization produced in the liquid argon generates electrical current and thus a measurable signal. These signals are collected by the readout electrodes in the gaps between the lead absorber plates. The readout electrodes are made of copper and are surrounded in the liquid argon. By measuring the ionization signals, the ECal can determine the energy deposited by the electrons and photons in the calorimeter.

The Liquid Argon Calorimeter employs a sampling technique, where the dense absorber material (lead) causes electromagnetic showers to develop, while the liquid argon serves as the active medium to detect and measure the energy of the showering particles. This method allows for accurate energy measurements while also containing the size of the calorimeter.

However, the electrical signals measured on the readout electrodes are weak and need to be amplified and shaped before further processing. Front-end electronics, located close to the calorimeter, perform this initial amplification and shaping of the signals.

After being amplified and shaped, the analog current signals are digitized. Analog-to-digital converters (ADCs) convert the continuous analog signals into digital values, which can be easily processed and recorded by digital systems. This entire process is handled by the Front End Boards (FEB). In the future the FEB will be upgraded to the FEB2.

Then, we transmit the data to the digital signal processor where the final step in the liquid argon signal processing is the reconstruction of the energy deposited by the particles in the calorimeter. To achieve such goal, we need to develop custom electronics to meet the demands for the future upgraded High Luminosity Large Hadron Collider, to handle the increased data rate. This new electronics, involve software, firmware and hardware developments on Field programmable Gate Arrays. We will spend the majority of our development and discussion on building and improving this component of the calorimeter called the Liquid Argon Siganl Processor (LASP).

1.2 High Luminosity Large Hadron Collider (HL-LHC)

The HL-LHC upgrade is aimed at significantly increasing the collision rate and the overall luminosity of the LHC by a factor of 5-10, which refers to the number of particle collisions occurring per unit of time and per unit of area. This increase in luminosity is crucial for enhancing the chances of discovering new particles or phenomena and for conducting more precise measurements of known particles. However this upgrade comes with many new challenges, of which pileup and new trigger systems are critically important motivators to upgrade the current LAr Calorimeters.

Pileup: Pileup refers to the phenomenon that occurs when multiple proton-proton collisions take place simultaneously in a single bunch crossing in a particle accelerator like the LHC. In other words, when two high-energy proton beams collide head-on, each bunch of protons contains a certain number of protons, and each proton can interact with another proton from the opposing beam. However, due to the high luminosity of the LHC, there's a possibility that multiple protons from one bunch can interact with protons from another bunch at the same time. This results in several independent interactions occurring in one bunch crossing.

Pileup becomes more significant as the luminosity of the accelerator increases, this in part is due to the higher number of interactions per bunch crossing on the order of 200 as opposed to 40 in the current LHC. High pileup levels can pose challenges to particle detectors and analysis because it becomes harder to distinguish the particles coming from different interactions. This can impact the accuracy of measurements and make it more difficult to identify rare or interesting events.

In terms of the new Trigger Systems, particle accelerators like the LHC generate a tremendous amount of data from each collision, and not all of this data can be processed and stored due to limitations in computing resources. To address this, trigger systems are employed. Trigger systems are specialized hardware and software designed to quickly decide which collision events are worth recording for further analysis.

The HL-LHC upgrade involves the development of new trigger systems to handle the increased number of interactions per bunch crossing. This means that it is more difficult to identify quickly the particles we want to have as a final state and hence the events to select. This is in turn a result of increased pileup. The new trigger systems aim to be more sophisticated and selective in choosing which events to record. Here are some aspects of the new trigger systems.

Together with other upgrade challenges we see that critical electronics such as those for the LAr need to be upgraded as well to match the new demands of the HL - LHC both with the new trigger scheme and for the high pileup as well upgrading the hardware to hold the more computationally intensive firmware. This new upgrade for the LAr is called the Liquid Argon Phase-II upgrade. Even more specifically we look to upgrade the Liquid Argon Signal Processor (LASP) to address these new challenges.

1.3 Overview

For the LASP upgrade there are few major components that need to be upgraded and developed. This falls under 3 big categories, firmware, energy reconstruction algorithms and functional tests. In the remaining sections we discuss 5 major projects working towards advancing each of these categories in various smaller projects.

1. **Firmware**: is the development of custom circuits on digital electronics to process incoming data. The LASP is primarily programmed by writing firmware on these FPGA (field programmable gate arrays). Below are few firmware projects that we have developed this summer.

   (a) PATGEN: stands for Pattern Generator and is a piece of firmware designed to generate FEB2 equivalent data internally without having to connect to a FEB2 board.

   (b) TTCGEN stands for the TTC (Trigger Timming Control) generator which generates internally TTC equivalent data without having to be connected to the outside world.

   (c) 10 GB BASE-KR is a new implementation of the 10GB network stack that uses the BASE-KR PHY layer instead of the original BASE-R PHY layer [1].

2. **Energy Reconstruction Algorithms** are algorithms on the LASP used to convert digital current measurements to estimations on energy. In this project we investigate the use of machine learning algorithms to help with energy reconstruction in high pileup environments.

3. **Functional Tests:** we also prototype circuits and software for reading current and voltage measurements on the LASP boards. These are designed to test the functionality of the manufactured hardware.

Together all these projects help contribute to the development of LASP board for the planned Phase II upgrades.

2 Background

2.1 FPGA's and Firmware

An FPGA, stands for Field-Programmable Gate Array, is a type of integrated circuit that is programmable and reconfigurable after manufacturing. Unlike traditional application-specific integrated circuits (ASICs) that are designed for specific tasks, FPGAs can be customized and adapted for various applications through hardware programming called firmware programming.

FPGA's offer attractive advantages for our LASP use case in that 1) Flexibility: They can be easily reprogrammed to implement different logic functions or algorithms, making them adaptable for various applications. 2) High performance: In certain scenarios, FPGAs can provide faster processing compared to software running on a general-purpose CPU. 3) Real-time processing: FPGAs are well-suited for real-time applications due to their ability to process data in parallel.

The principle of firmware programming is that it is a description of *circuits* rather than *code* like in a procedural program. By default signals are sent in parallel, but in order to implement procedural logic, clocked processes are vital. For every component we time the manipulation of signals to the cycles of a clock. This way one can reasonably program and synchronize the various parallel processes in their firmware. This clocked process can also be called a flipflop. With this principle in mind, one can leverage the power of these fabless boards for realtime processing!

The general cycle of development with firmware follows as such: 1) firstly we implement the algorithm in VHDL (Very High-Speed Integrated Circuit Hardware Description Language) code 2) we simulate the code in a testbench simulation to verify it works as intended 3) we write the component onto a development kit FPGA and finally we write to the LASP testboard and use signal tap to verify its functionality.

This summer we build upon the existing LASP firmware upgrade to implement more transparent testing components for future development.

2.2 Energy Reconstruction and the Optimal Filter

In addition of the firmware side of the LASP, the actual algorithm in reconstructing the digital signal is vital as well. The signal processor needs to take in ADC data and convert it into an estimation in the energies of an event. Currently, to do so we use the Optimal Filter (OF).

The OF is a linear filter. The filter works by taking a set of constant coefficients to which we then multiply and sum with the incoming ADC data. These static coefficients are found through minimizing a least squares fit between the model and the simulated data. This technique is currently favoured due to its interpretability, accuracy and efficient implementation on a FPGA.

Although this has proven to be successful, there are new challenges that are not present for the current ATLAS detector, namely pileup.

Pileup occurs when the collision rate is high, which is expected in the HL-LHC. In these scenarios, particles produced from single collision can still be present or interacting within the detector when subsequent collisions occur. These collisions occur at every 25. For the LAr, this is due to the delay caused by the time it takes for electrons to drift across the liquid argon gap creating a pileup of signal since. In other words the signal does not disappear fast enough before the next collision occurs, hence creating an increased amount of pileup.

This poses a problem for the OF since this super position of signal creates noise making inaccurate reconstruction of energy. Furthermore, even if we refit the optimal filter to the high pileup data expected for the HL-LHC the simple linear filter still is insufficient in recreating the original energy signal accurately due to the noisy low energy pileup.

Previously, attempts have been made to improve the linear filter through the use of various neural networks, and other non-linear filter techniques. This summer we investigated new approaches to this problem using machine learning beyond that of convolution neural networks and recurrent neural networks.

2.3 Hardware Boards and Testing

Lastly, regardless of the firmware and software engineering involved we also need to develop robust hardware to run the LASP on. We anticipate the hardware to eventually be manufactured by third-party vendors like Intel and physical links developed by members of the ATLAS collaboration.

However, to determine the functionality of these hardware boards we need to develop physical probes to test the hardware. One of the proposed ideas is to develop user friendly software to interface with measurement tools such that engineers on the ground can quickly determine the functionality of a given manufactured board. This allows for faster turnarounds.

3 Firmware

3.1 Introduction Firmware Programming and LASP Framework

Programming an FPGA involves designing and configuring the hardware logic within the FPGA to perform a specific task or function. These devices are versatile in that they allow one to define one's own digital logic circuits, making them suitable for a wide range of applications. The process involves several steps, from designing the logic using a hardware description language (HDL) to actually programming the FPGA device. We quickly walk through how this is typically done for the LASP project.

For the LASP project, the firmware is based in VHDL and tested, developed and deployed to Intel based silicone. Because of these restrictions we make use of Intel's Quartus Prime 22.4, Intel's IP Cores and their suite of simulation and programming tools to implement our design and our workflow.

Using these Intel provided software tools, the LASP project builds a framework to help accelerate the design and testing of these projects by connecting all various dependencies and sub repositories through various symbolic links. This means that when using the LASP repository, everything from simulations to compilations to even opening up quartus is done through MAKE commands which automatically pulls in dependencies from the larger LASP project.

On a macro-scale, the LASP project consists of many components responsible for various tasks in the digital signal processing pipeline. Each coloured box 1 represents a component of the LASP. These components live as sub-repositories within the LASP repository project.

As evident by the number of coloured boxes, this firmware is quite complex. When developing firmware we start with VHDL code of a component in isolation and slowly integrate complexity into a particular component of the project. We then simulate the newly created components and

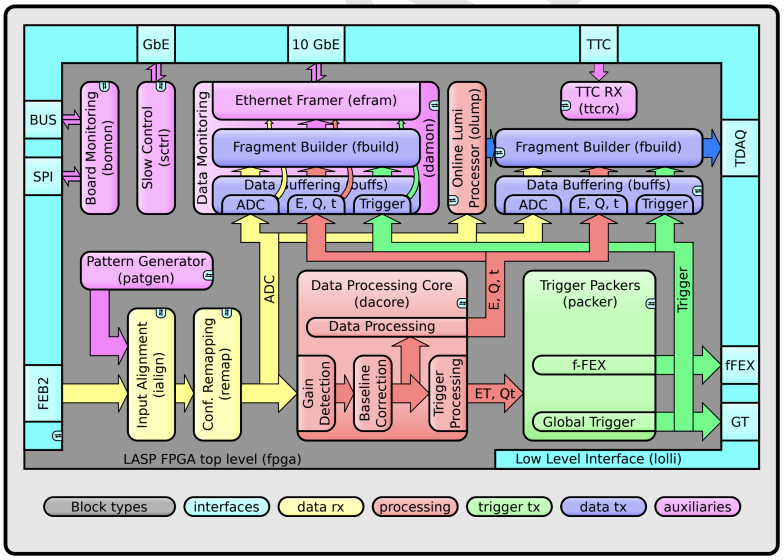

Figure 1. Diagram of all LASP Components

verify it's behaviour is correct. Beyond verifying by eye, we can verify using automated checks called UVVM (Universal VHDL Verification Methodology). UVVM checks are checked by the continuous integration (CI) in the LASP repository such that any merged code will need to pass these checks to be accepted into a master branch. Lastly, one would implement a compilation target that uses the component that we have created. This compilation target should also include signal tap which allows one to verify the internal signals of a particular component after writing it to an FPGA. This way it allows us to investigate the internal signals of the FPGA in realtime. The devices that we run on are the INTEL H-TILE STRATIX 10 DEVELOPMENT KITS with some of our work also having a LASP testboard (closer to real electronics deployed in the future) compilation target.

With this design, simulate, compile, test framework in mind, we move on to developing 3 components of the LASP, namely PATGEN in section 3.2, TTCGEN in section 3.3 and 10GBE BASE-KR network stack in section 3.4.

3.2 PATGEN

Pattern Generator (patgen) is a component designed to provide a FEB2-equivalent data generator [2]. This is an important component dedicated to testing and ensuring all subsequent components of the LASP is functioning properly with testable mock data.

To understand what kind of data PATGEN needs to mimic, we need to understand where the data comes from. The FEB2 stands for the Front End Board 2 where 2 is the indication for phase 2 upgrades [3]. The FEB2 houses the electronics that are mounted close to the detector.

These boards in particular receive the signals from the calorimeter cells to perform a fast analog processing, including amplification, shaping and a split into a high gain and a low gain [2]. Both gain are digitized by an ADC.

After the FEB2, the data is now a digital signal ready to be sent off detector optically. This is done through the lpGBT [4]. The LPGBT (Low Power GigaBit Transceiver) is a high-speed data

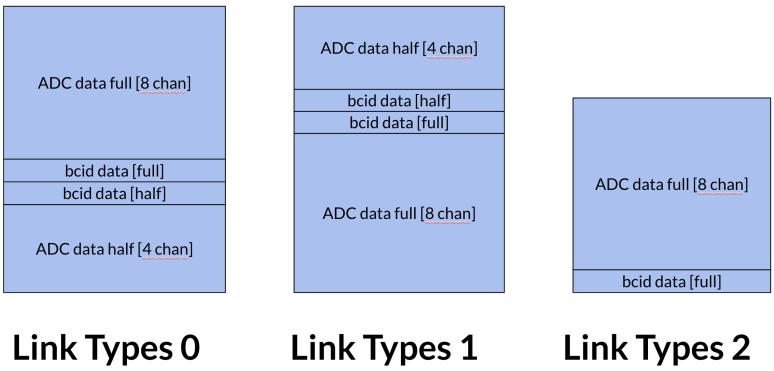

Figure 2. Example of link types 0,1, 2.

transmission and reception device made to provide reliable and high-speed data communication in harsh radiation environments. This connection has 14 channels of inputs unlike the 8 output channels of the FEB2.

However there is a complication, the mappings between the ADC chips with data coming out of the FEB2 versus that number of lpGBT boards is a mapping of 3 to 2. To support this there are 3 specific link types (named 0, 1, 2 [2]) to complete such an awkward map.

In link type 0, 1, it includes 1.5 of a complete FEB2 ADC data output with 2 BCID (bunch crossing id) signals for each FEB2 output. In link type 2 it contains just 1 ADC data output. Note that each FEB2 outputs 8 channels of ADC data, so a full FEB2 has 8 ADC channels while half has 4. More specifically we expect signals to look like the following incoming from the FEB2, see diagram 2.

Our goal is to mimic this design and make it modular to allow future development of more complex data payloads.

3.2.1 Requirements

The more explicit requirements of PATGEN are outlined in the firmware specifications document [2], but are repeated here for readers to easily follow.

1. The data provided by patgen must have the same structure as the data from the FEB2s
2. Patgen must be able to mimic the non-synchronization between the links (BCID shift).
3. Each of the 66 links shall have an independent source of data.
4. A controllable multiplexer shall select between the two possible data sources (patgen or lolli's FEB2 interface) for ialign for each link individually.
5. Patgen as a data source shall be start- and stoppable,
6. While running it shall provide data in an uninterrupted, continuous and circular manner
7. The data repetition shall be compatible with an orbit (3564 BCs). (If resources require, a repetition twice or any integer fractions of an orbit are tolerable

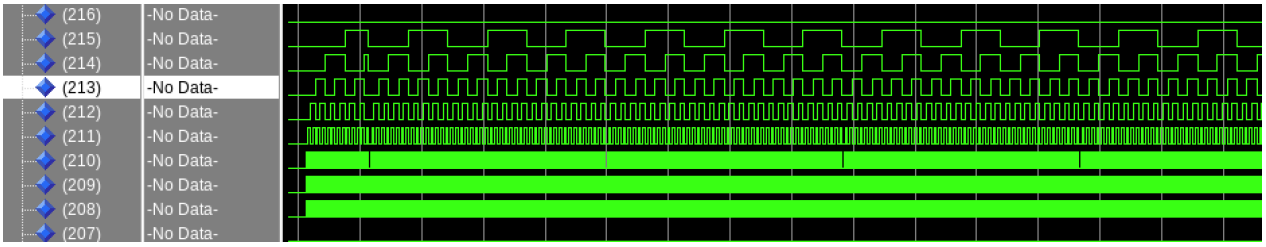

Figure 3. Example of an ADC channel implemented as a simple counter.

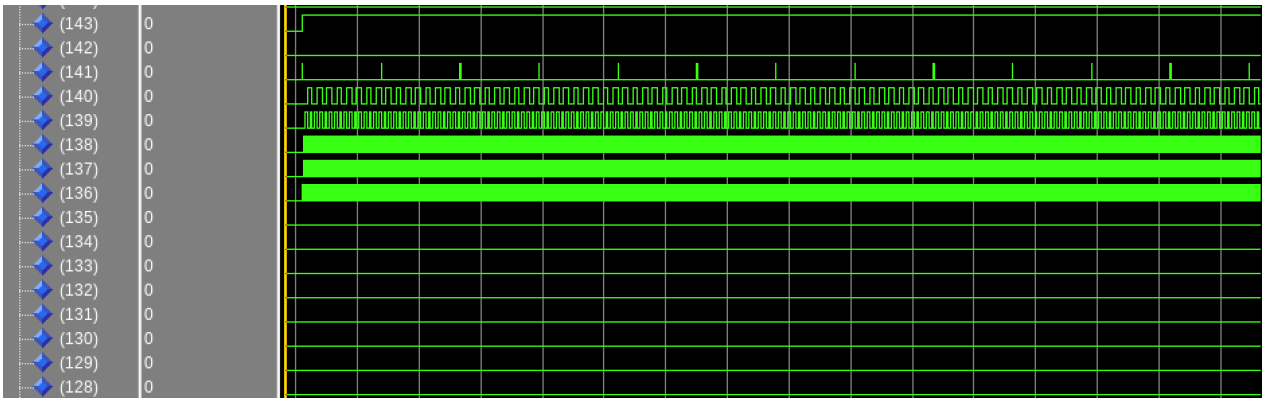

Figure 4. Example correctly formatted BCID info. Notice that the first bit is always 1, while the second bit is always 0 and the third bit is the BCR (bunch crossing reset) and the next five bits are the BCID followed by 8 bits of padded zeros.

3.2.2 Simulation

Firstly, to simulate PATGEN, we need to firstly generate a valid ADC data. For our simulation setup we will generate mock ADC data from a simple counter. We chose to use a simple counter as the mock ADC data because it makes it simple to debug future firmware components [2]. Below is a demonstration of utilizing a counter in figure 3.

Secondly we need to format the BCID information to the correct standards. We need the first bit to be 1 the second bit to be 0, the third bit to represent the BCR (bunch crossing reset) and the next 5 bits to represent the truncated BCID and the remain 8 bits to be padded with 0 [2]. Below is a demonstration of that in figure 4

Thirdly, we need to make sure all the link types are of the correct format. There are three link types, each representing the combination of full and half ADC data. Recall that an ADC data has 8 channels, and a half ADC has 4. We piece together a set of 12 channels based on this data to form the three link types. Below is a demonstration of one of those link type implementations in figure 5. Fourth, we have delay chains. One of the primary use cases for the PATGEN component is to use it to test the ialign module. The ialign module is used to align the inputs coming in from the FEB2 [2]. In this case we want to introduce artificial delays such that we can test the ialign module for such situation. Below we demonstrate the use of delay chains and subsequently the use

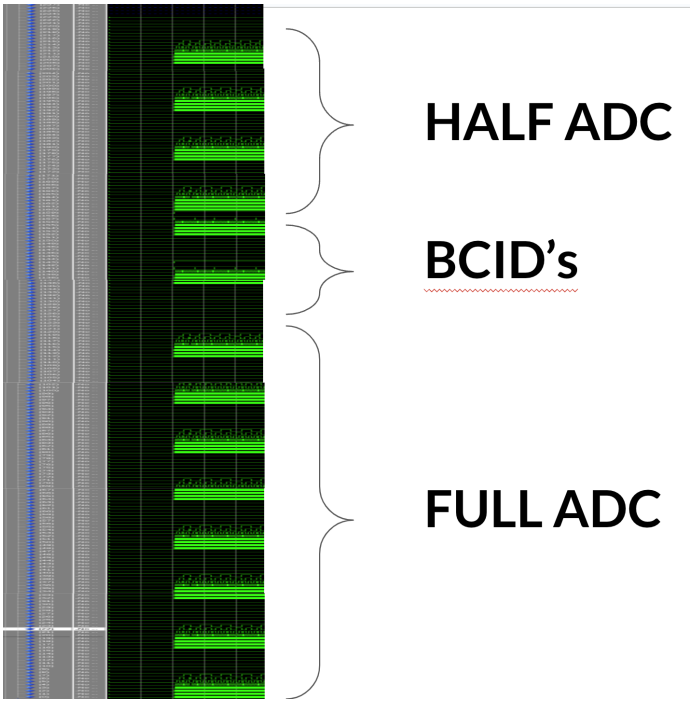

Figure 5. Example of linktype 0 which conforms to diagram 2. This appears to be flipped but keep in mind the simulation counts in reverse order.

of ialign with these delay chains. Fifth point is, we want to make sure that the ADC data orbits every fixed number of counts following the BCR [2]. This is because it makes debugging easier when observing the ADC data we can see it follows the same loop back. Last point, we have also implemented UVVM continuous integration checks for the PATGEN component. We check mainly for the regular counting of ADC and BCID data. We also count to keep track that the BCID loops back in the expected stated number of cycles.

3.2.3 Compilation

Finally having checked out the implementation in simulation we finally compile this to a particular target. This means writing to a particular FPGA and use a program called signal tap to check that the FPGA's internal signals are behaving in the correct fashion. Below in figure 8 we check signal tap on a compiled firmware and write it to the LASP testboard and observe the that the signals match that of the signals coming from the simulation. This verification is an indication that the written VHDL code is most likely correct.

3.2.4 Next Steps

The next step is to integrate PATGEN with other components in slice test beyond just the IALIGN component.

3.3 TTCGEN

The TTCGEN stands for TTC generator [2] . The TTC stands for Trigger Timing Control [3]. The TTC system provides the timing and trigger signals necessary to coordinate data acquisition and

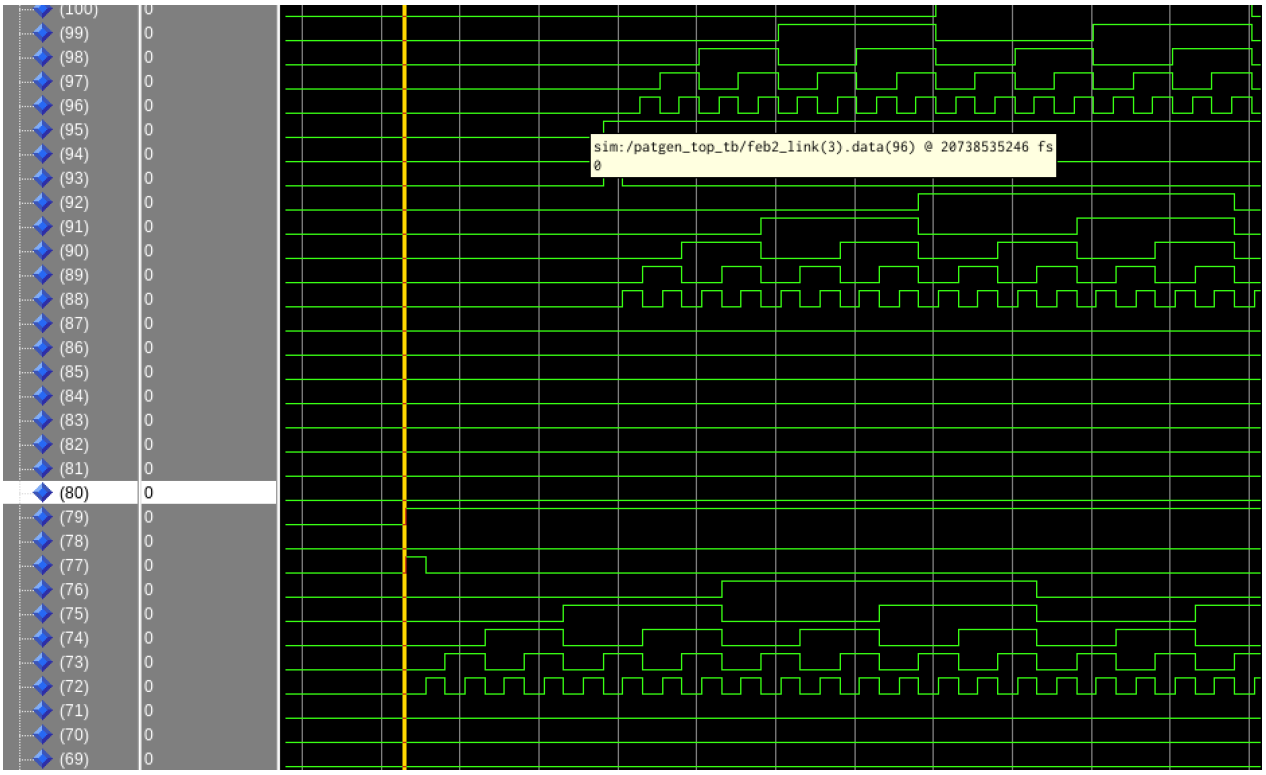

Figure 6. Example delay chains, with some signals coming earlier than others

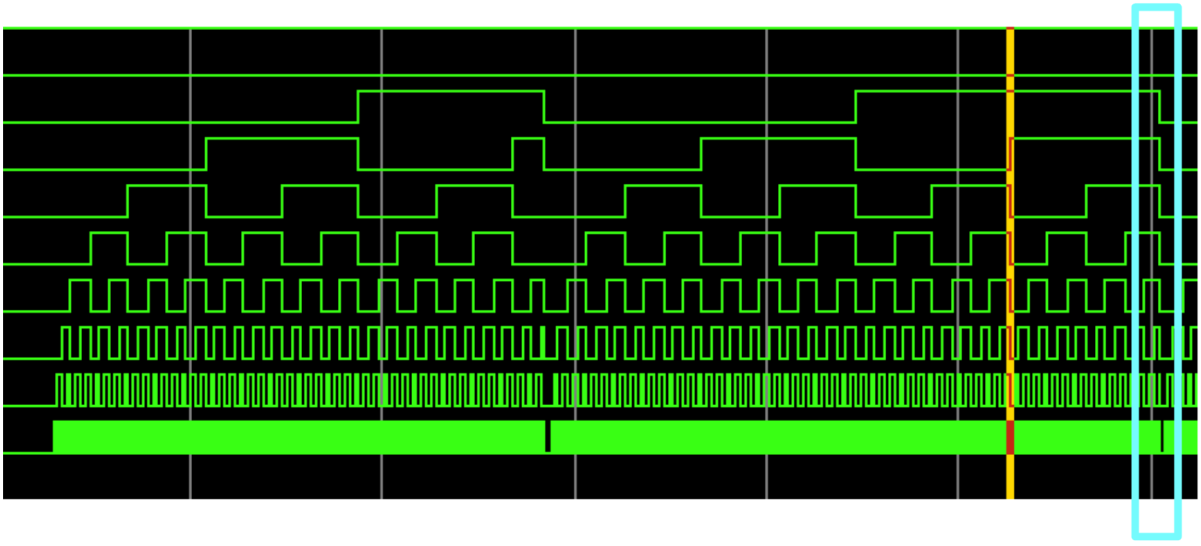

Figure 7. Example of an ADC channel having the correct fixed orbit highlighted by the last bit around the blue box.

Figure 8. Example of patgen running on a LASP Testboard.

trigger events in the ATLAS detector [3]. It ensures that data collection is synchronized across the various subsystems.

More specifically it will deliver the Level-0 trigger accept signal (L0a), which is based on coarse information about particle energy and position, to the LAr calorimeters [2] .

Along side the L0a, The LASP receives a payload of data all through a component called the TTCRX from the outside world about particular triggered events. Similar to PATGEN, to test the LASP in the future would require us to have connections to the TTC incoming data, however relying on being connected to the outside world to debug internal components is not optimal. Instead want to be able to generate this signal internally. To do so we created the TTCGEN component to satisfy this need.

3.3.1 Requirements

The more explicit requirements of TTCGEN are outlined here.

1. Must provide a valid L0a, L0ID, LBID, ttype and BCID signals. Please see TTCRX firmware specification manual [2] for details on these signals.
   (a) L0a must take the form of a BCID signal
   (b) L0ID must be a 38 bit counter keeping track of every L0a that occurs
   (c) LBID is the luminosity block id which is a counting the first bunch crossings.
   (d) ttype is the trigger types that can be outputed by the trigger system.
2. A trigger must be set at random times to mimic real trigger system
3. The trigger must be a light weight implementation.

However, since the current use of slice test does not make use of the ttype and the LBID signals we will set those to be zero for the time being. Furthermore, to simplify the development, we created a minimal viable piece of firmware and so the trigger does not occur randomly instead it occurs every set number of frequencies. In otherwords we address points number 1, 3 first.

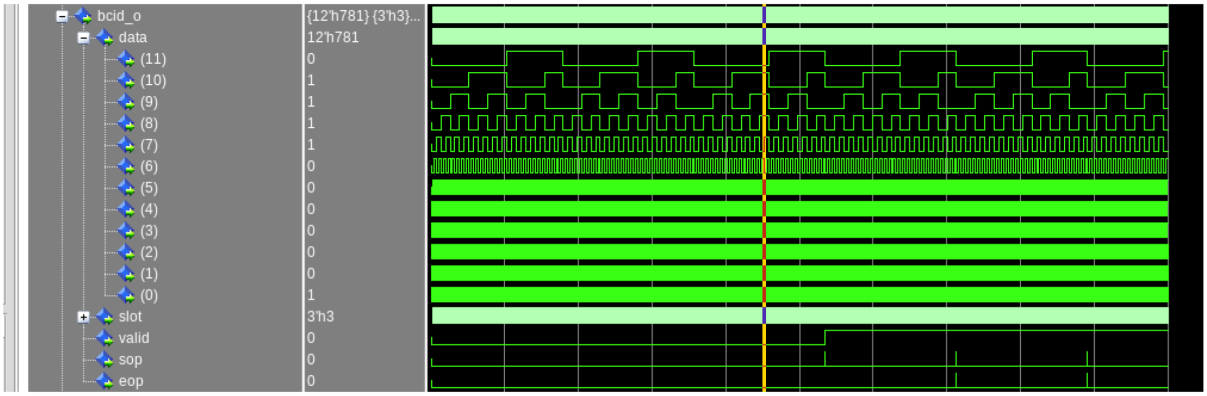

Figure 9. Example of TTCGEN bcid signal.

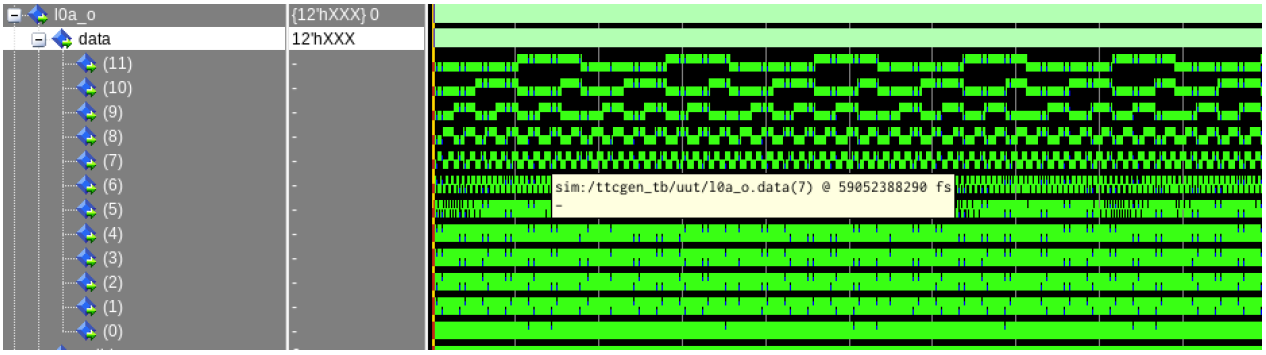

Figure 10. Example of TTCGEN L0a signal.

3.3.2 Simulation

For the simulation we want to first check that the BCID is of the correct format and is of the correct size and data shape. We expect the BCID to be a 12 bit counter increasing upwards with additional bits of for the valid, start of packet, end of packet and slot. Here we see figure 9 demonstrating that the BCID is in fact 12 bits long and that the data format contains the valid, eop (end of packet), sop (start of packet) signals matching the correct data format.

Secondly we want to check the TTCGEN is outputting the correct BCID's to trigger on. We expect the signals to be the same as the BCID signals except it contains regular splits of data where the trigger is not on. Here in figure 10 for demonstration purposes we trigger on every 10 cycles. We see that there are sections of the BCID where we loose data as expected since we are not triggering on all the data.

Lastly we want to check that L0ID is counting up for each of these triggers that occur. This is of 38 bit length data. We expect to see a simple counter. We demonstrate below that this is indeed the case 11.

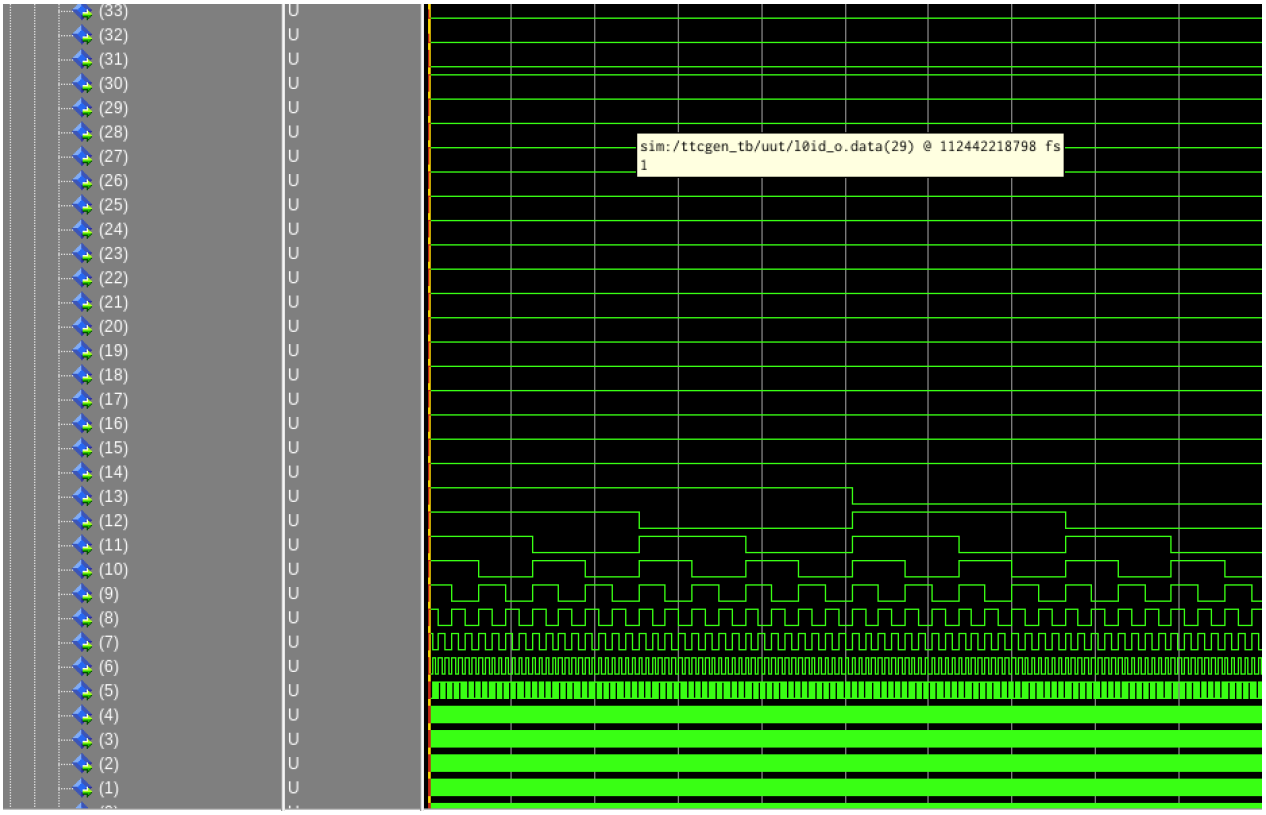

Figure 11. Example of TTCGEN L0ID signal. It is in fact counting upwards insych with the L0a signal.

3.3.3 Compilation

As described previously we also compiled to the devkits and the LASP testboard and have successfully completed both.

3.4 10GBE Base R/KR

Currently the connection between the 10 gigabit link between the Smart Rear Transition Module (SRTM) component. The SRTM unlike the current LASP uses a 10 gigabit BASE-KR network protocol in its network stack.

A network stack, also known as a networking protocol stack, is a layered set of software protocols and components that enable communication between devices over a network [1]. It's responsible for managing the flow of data between computers, servers, or any network-connected devices. The network stack follows a structured architecture, usually based on the OSI (Open Systems Interconnection) model or the TCP/IP model.

1. **Physical Layer**: This layer deals with the actual physical transmission of raw data bits over the communication medium, such as cables, wireless signals, and connectors. It defines characteristics like voltage levels, cable specifications, and physical topology.

2. **Data Link Layer**: Responsible for reliable point-to-point communication between directly connected nodes. It handles error detection, flow control, and framing (dividing data into frames). Ethernet and Wi-Fi protocols operate at this layer.

3. **Network Layer**: Focuses on routing packets of data from source to destination across multiple networks. It involves addressing, routing, and logical network topology. IP (Internet Protocol) operates here.

4. **Transport Layer**: Manages end-to-end communication and ensures reliable data delivery. It divides data into smaller segments, performs error checking, and handles flow control. TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) operate here.

5. **Session Layer**: Establishes, maintains, and terminates communication sessions between applications on different devices. It manages dialog control and synchronization between processes.

6. **Presentation Layer:** Handles data translation, compression, and encryption, ensuring that data sent by one application is understood by another. It also provides data formatting and code conversion.

7. **Application Layer**: The topmost layer interacts directly with user applications. It provides network services and application protocols for tasks such as file transfer (FTP), email (SMTP), web browsing (HTTP), and more.

To demonstrate the network stack here is a diagram depicting such a network stack in figure 12.

3.4.1 Simulation

For the simulation we took the existing network stack that has the BASE-R implementation of the network stack and attempt to use the base-KR implementation of the PHY layer. However we will only use one network stack and implement a loop back. Meaning the data sent to the transceiver TX will then be connected to the receiver thus looping back the data. The result appears as follows.

We see all the signals are high for the RX/TX serial data and their respective 'ready' signals. We also see that the various locks are all high except for the block_lock signal. We note that the result is mostly promising except for the single lock signal.

3.4.2 Next Steps

After contacting Intel engineers for assistance[1], one of the suggestions was to instantiate two network stacks instead of just one network stack. There is an apparent issue with the. However if one use KR in loopback one have to set the nonce bit for the AN/LT to come up properly.

[1]We reached out to peter.schepers@intel.com

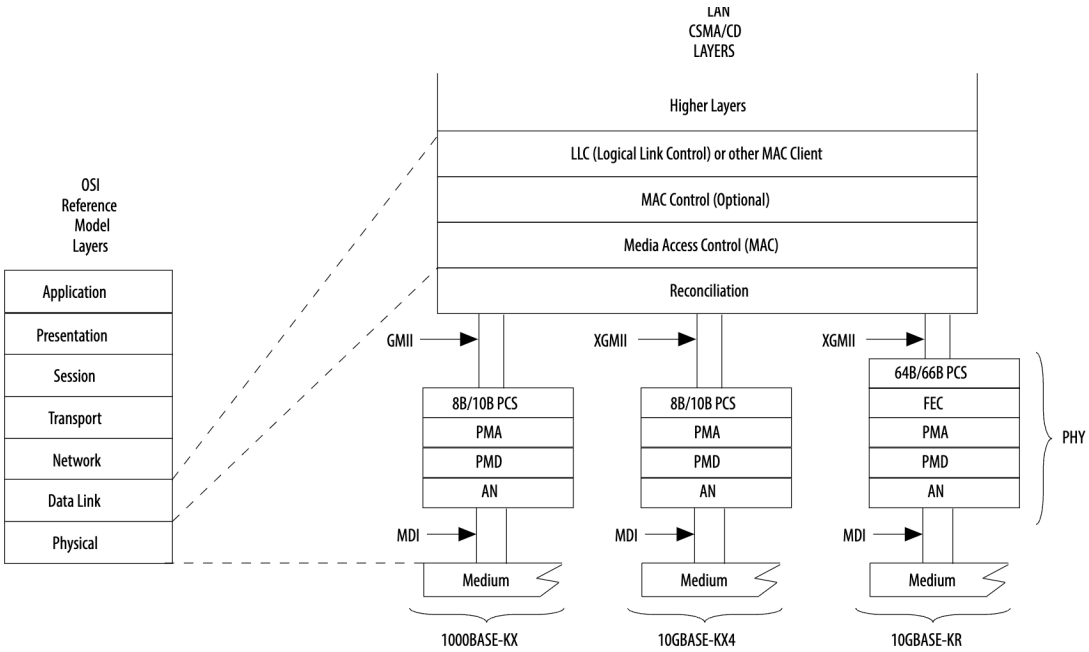

Figure 12. We see that the physical layer of the stack uses the base KR implementation that includes the MAC layers and the BASEKR layer.

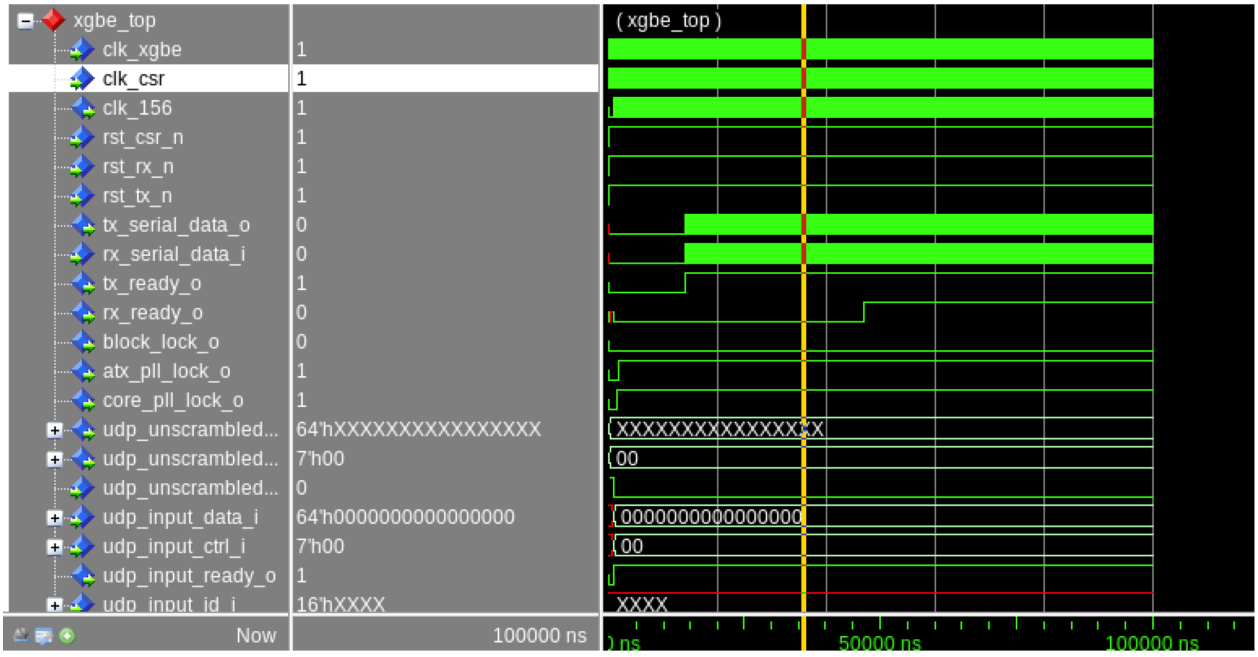

Figure 13. Example of BASE KR implementation of the network stack.

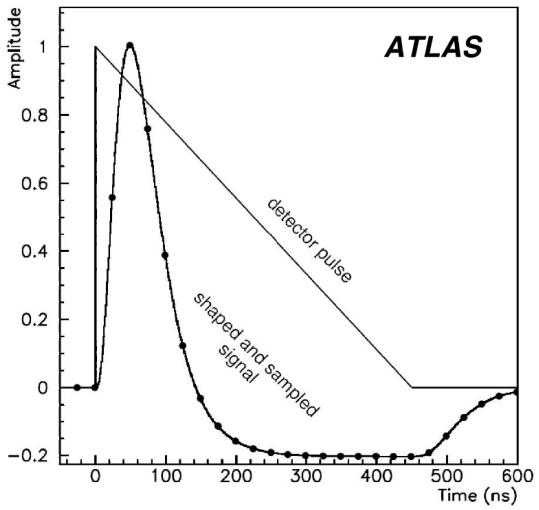

Figure 14. Example of the pulse shape that we wish to fit to [6].

4 Energy Reconstruction Algorithms

4.1 Optimal Filter

Calorimeters are detectors that measure the energy of particles by absorbing them and converting their energy into detectable signals. The optimal filter is responsible for accurately determining the energy of these particles based on the measured signals [5]. So here is a quick run down as to how the optimal filter works in energy reconstruction for calorimeters.

Firstly, calorimeters respond to incoming particles by generating electrical signals. The shape of the generated signal, known as the signal response function, depends on the type of particle and its energy. The optimal filter aims to find the best way to match the measured signals with the expected signal response [5].

Then we can look at deconvolution. We take the known shape of the energy deposition and fit the coefficents to match training data generated by a simulation. We use these optimal coefficents (a_{linear})for computing the actual shape of the pulse.

Lastly we get a weighted sum. The optimal filter performs a weighted sum of the measured signal values, where the weights are determined by the deconvolution process. The goal is to emphasize the parts of the measured signal that best match the expected signal response. Mathematically it looks like the following:

$$O_t = \left(\sum_j^n a_{\text{linear}} V_{t-j} \right) \tag{4.1}$$

As mentioned before, the choice of filter weights is determined mathematically to maximize the signal-to-noise ratio. This means that the optimal filter amplifies the parts of the signal that carry the most information about the particle's energy while suppressing noise and other unwanted effects. This gives it the characteristic shape.

The optimal filter technique is used to improve the accuracy of energy reconstruction in calorimeters by taking into account the characteristics of the detector's response and minimizing the impact of noise. This is crucial for particle physics experiments where precise energy measurements are necessary to study the properties of particles and phenomena. The technique requires a deep understanding of signal processing, detector physics, and mathematical optimization methods.

4.2 Motivation

The Optimal Filter is still robust in certain regimes of input data. Let us make use of the already simple model and add on to it small corrections. We fill in these small corrections by O which we model as a recurrent neural network.

However, since we know that we don't care about the energy reconstruction *everywhere* but rather we care about the energy reconstruction at peaks where there's high energy, we can selectively restrict the problem to just these special regimes. These regimes are controlled by the Γ "gate" value by multiplying this correction term by 0 or 1. This gives us the structure we see in the equation above.

4.3 Optimal Filter Neural Network Correction

Let us fill in the missing blanks. We will work with a window size of m with n optimal filter coefficients.

1. We let O take in $m > n$ samples of 'ADC' data
2. We compute Optimial Filter from the most recent values, this case we have n coefficents so we take the n most recent 'ADC' data points and feed into the 'OF' model. Note: We keep track of the most recent m 'OF' outputs in memory.
3. We take the most recent n 'OF' outputs and feed them into the gate model

Let us formalize this more.

1. Compute 'OF'
$$O_t = \left(\sum_j^n a_{\text{linear}} V_{t-j} \right)$$
2. Compute Gate, here we are using O as the saved m samples of O_t (optimial filter output)
$$\Gamma_t = \Gamma(O), \quad \Gamma : \mathbb{R}^m \to \mathbb{R}$$
3. Lastly we compute the Correction terms where $V_t \in \mathbb{R}^m$ is all the m samples of the 'ADC'
$$O_j = O(V_t)_j, \quad O : \mathbb{R}^m \to \mathbb{R}^n$$

Okay enough talk, implement this. Let us first check how the optimal filter preforms on high pileup data. See figure 15.

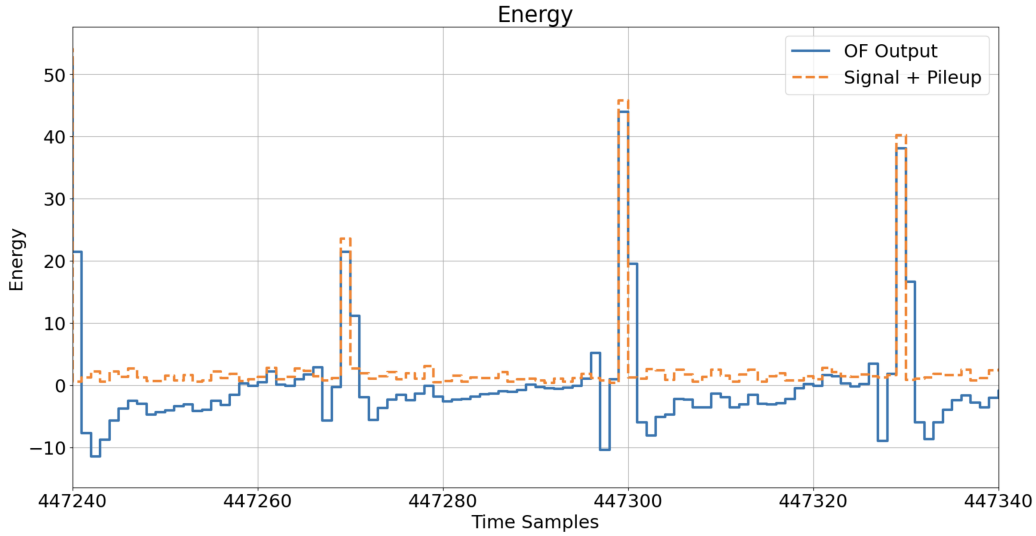

Figure 15. Example of optimal filter on high pileup data

Here we construct a neural network model that uses the optimal filter alongside a neural network correction term. However to train the neural network we compute minimize the variance of the error. This is because in general we do not care about a constant bias, we care only about minimizing the spread or the variation in the error. Thus we directly minimize such loss function. Consider y as the label, and we say p as the model including the optimal filter, and V as the input sample. And consider there to be n training samples then we have

$$\mathcal{L}(V,y) = \frac{\sum_i^n \left((p(V_i) - y_i) - \frac{1}{n}\sum_j^n p(V_j) - y_j) \right)^2}{n-1} \tag{4.2}$$

We minimize this using ADAM optimizer [7] and built the model using Keras [8]. The neural network model is a simple recurrent neural network with of 915 parameters. With 3 layers of 10 hidden units.

A more descriptive diagram can be found here in figure 16.

4.4 Results

The initial results appear to have better performance than the classical Optimal Filter by an order of 40% improvement in reconstruction error. There is a reduce in the standard deviation in the errors between true energy and predicted energy. This is reflected in the plot of distribution of error 17. This approach is notably better than classical pure ML techniques (see figure 18), because of its computational efficiency and improved accuracy. Since there is a gate, the feed forward correction term (the computationally expensive term) is only calculated intermittently where as for other approaches using Convolutional Neural Networks these expensive forward propagation terms are computed multiple times. The code can be found here repository.

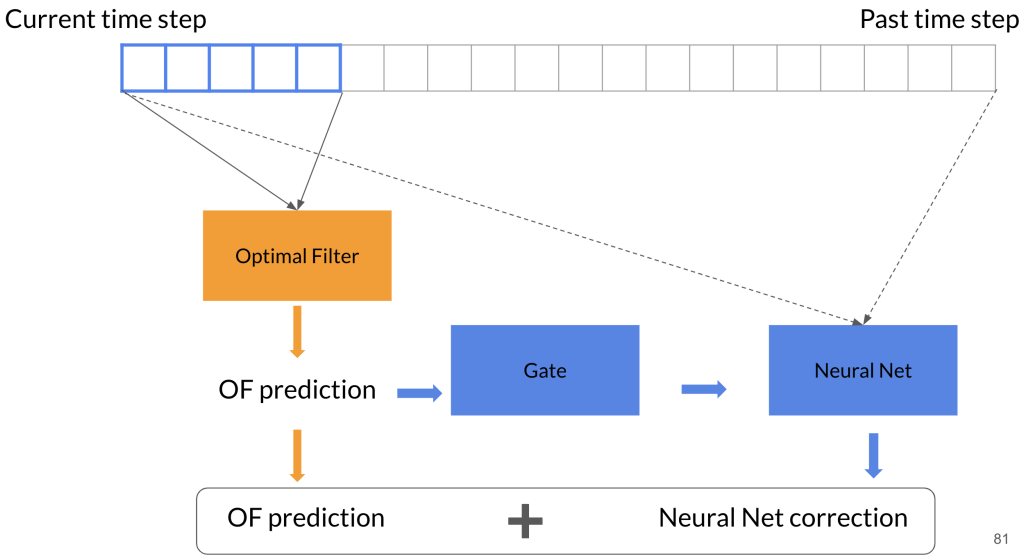

Figure 16. Example of neural network architecture. We see that the optimal filter takes in data as per usual, the output then gets fed into a gate which triggers the neural network correction term depending on its energy output. The results are then added together. Note that the neural network needs more data to be kept in memory on the order of 20 time samples, similar to a traditional convolutional neural network implementation.

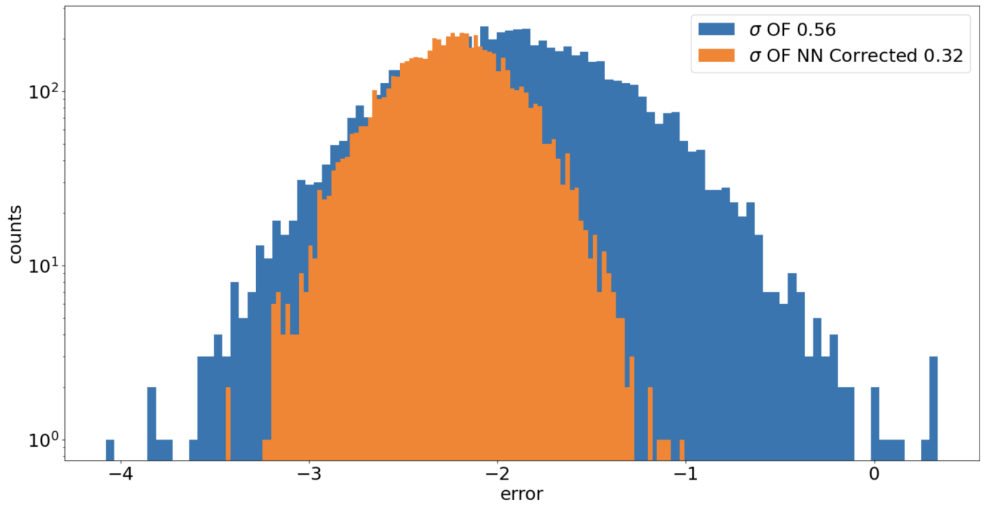

Figure 17. Histogram of computing true energy deposition minus predicted energy and we see that the spread or the standard deviation of the novel approach drastically improved the energy reconstruction error. The σ is the standard deviation of each error.

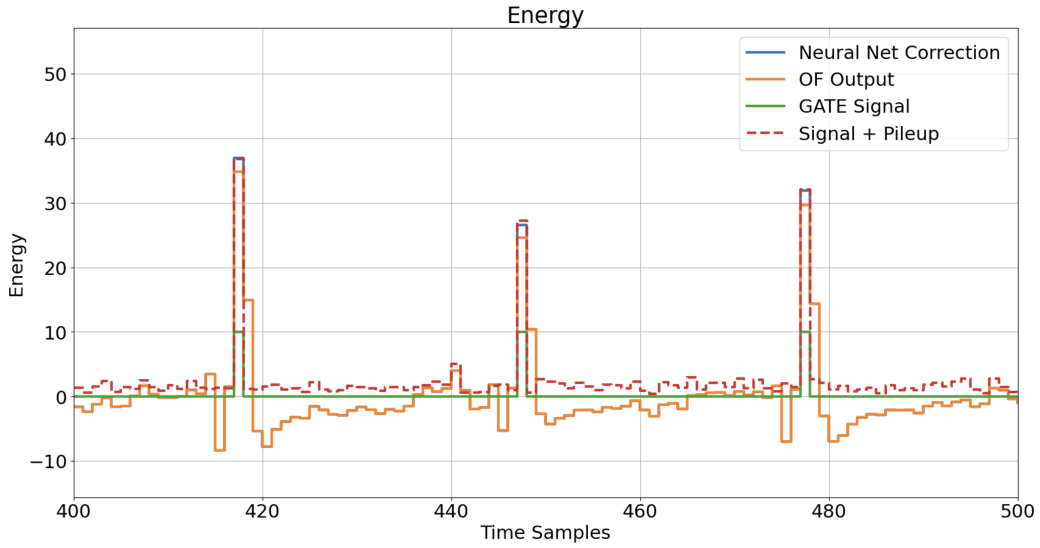

Figure 18. Visual check of the fit of our correction model. We see a noticeably better fit of the energy estimation than that of the traditional optimal filter.

5 Functional Tests

In this section we discuss the functional tests for the LASP board. One of the most important parts in the development is fabricating the physical board after designing all the firmware and the software to interface with the hardware components. However when building these boards from third-party suppliers it is up to us to help them determine the quality of their manufactured product and if these performances matches our expectations.

One of these performance tests is to test if the board is functional if at all immediately after they have been manufactured. This includes two key measurements, one of voltage and the second of the current. In this part of the project we discuss developing software interfaces and prototype circuits to conduct these tests.

5.1 National Instrument Data Acquisition

To measure the voltage or current using custom software defined tools we need to interface with the a data acquisition device. One such suitable device is the National Instrument (NI) cDAQ and its various modules used to control and acquire data.

For this project we use the special NIDAQMX python libaray to develop python scripts to measure and control and interface with the cDAQ, NI 9203 module for current measurements and the NI 9205 module for voltage measurements and NI 9481 for relay control.

5.2 Voltage Measurements

5.2.1 Hardware Setup

We first describe the steps for setting up the circuits and hardware tools required for preforming a simple voltage measurement on a known power supply unit (PSU).

1. Turn on the DAQ by plugging it in with the USB connector and its power cable.

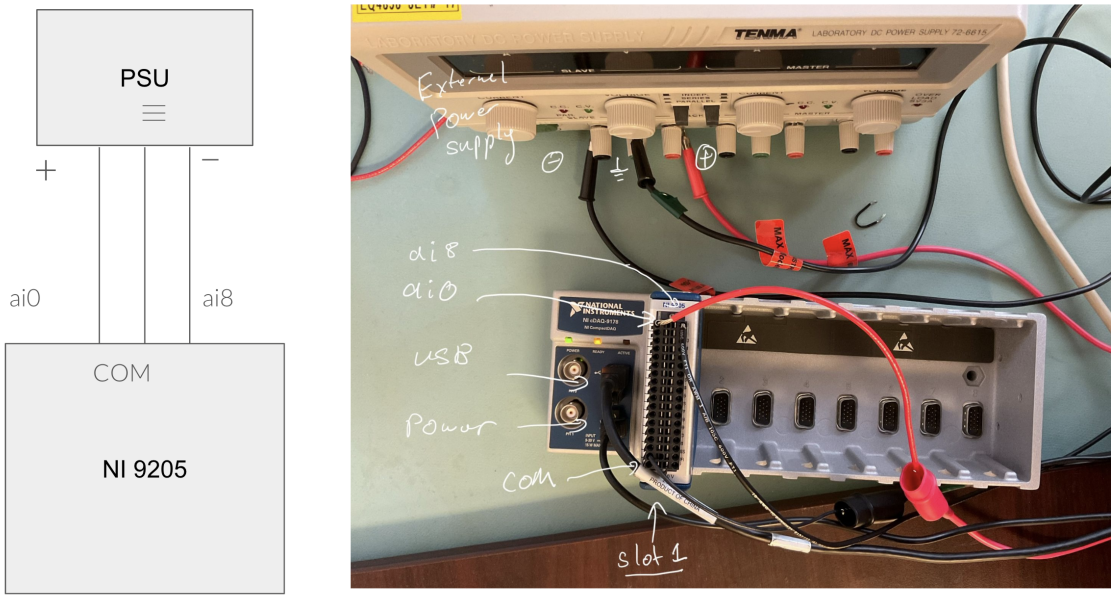

Figure 19. A graphical description of the hardware setup for our voltage measurement tests.

2. Take a power supply and route the positive leads to the ai0 channels. The negative leads come from the ai8 channels. (General Note: the negative leads come from the opposite end of the connectors)

3. We also want to ground the device so that our ground matches that of the power supply. Route a connection between the COM with the ground of the power supply. This makes the readings more accurate.

The circuit setup should look like the following description.

5.2.2 Software Setup

In the remaining parts of the project we developed the software components to interface with the NI cDAQ modules. Below is an example of our custom library interface.

```
from cDAQ import cDAQ_measurements

voltage_params = {"channel": ["cDAQ1mod1/ai0"], 'seconds': 20, 'sample_rate':
    1000}
test = cDAQ_measurements(voltage_param=voltage_params)
test.measure()
test.save_csv_voltage("voltage_measure")
test.plot_voltage("voltage_measure")
```

Listing 1. Example interface for reading voltage measurements on the cDAQ NI 9205 module

This interface is supported by the package provided by the national instrument. To make all this work, please follow the here.

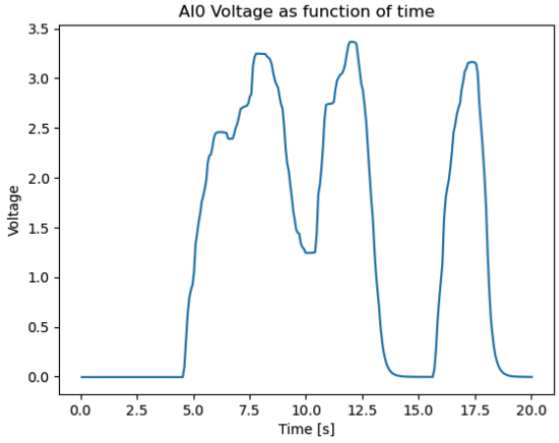

Figure 20. An example of voltage measurement conducted on a known power supply. Here we varied the voltage manually and see the recorded measurement reflect that manual variation.

Range	Accuracy at Full Scale	Random noise σ	Sensitivity
$\pm 10V$	6,230 μV	237 μV	96.0μV
$\pm 5V$	3,230 μV	121 μV	46.4μV
$\pm 1V$	692 μV	29 μV	10.4μV
$\pm 0.2V$	175 μV	15μV	4μV

Table 1. The accuracy of the voltage measurements using the NI 9205 module.

The resulting voltage measurements can be ploted as presented below in 21.

However it is important to also note about the accuracy of the voltage measurement for each gain in voltage that we choose to measure.

5.3 Current Measurements

5.3.1 Hardware Setup

Similar to previous section we describe the steps for setting up the circuits and hardware tools required for preforming a continuous live current measurement on a known power supply unit (PSU).

1. Turn on the DAQ by plugging it in with the USB connector and its power cable.

2. Take a power supply and route the positive leads to a potentiometer. We use the potentiometer to vary the current since the PSU cannot change the current.

3. We then connect potentiometer to the NI 9203 module in series putting the lead into ai0 and connecting ground to COM.

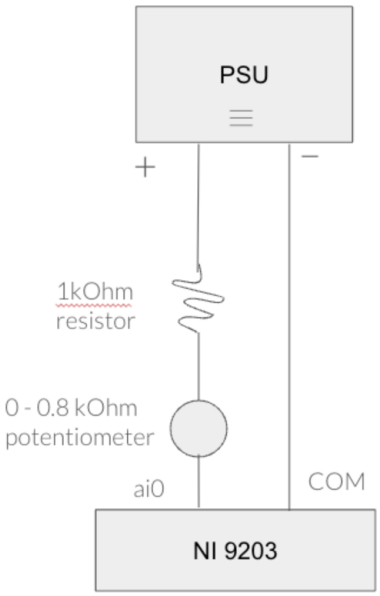

Figure 21. An example of voltage measurement conducted on a known power supply.

5.3.2 Software Setup

In the remaining parts of the project we developed the software components to interface that uses a live interface to display current measurements in a graphical interface. To do so run the following script. Please see here for a live demo of the GUI.

```
from cDAQ import cDAQ_measurements

current_params = {"channel": ["cDAQ1mod1/ai0"], 'seconds': 20, 'sample_rate':
    1000}
test = cDAQ_measurements(current_param=current_params)
seconds = 60
test.multichan_current_live(seconds, ds = 0.2)
```

Listing 2. Example interface for live reading current measurements on the cDAQ NI 9203 module for 20 seconds with a sample refresh rate of 0.2 seconds

One of the technical challenges when building such a GUI, is an issue with buffering data and multi-threaded data acquisition. Since a single thread is allowed to read the data, we must then store the data in some buffer or memory. One way is to have 2 concurrent threads, one thread to read in data and buffering it and one to generate the GUI. We choose to do this because the NI cDAQ does not allow concurrent reading on a single cDAQ we have issues in recording data, hence we have a small gap in the time it takes to buffer the data meaning we loose data taking time momentarily. To mitigate its effects we create a third concurrent process that listens to a global variable. This third thread buffers the data into memory whenever this global variable is updated. This leaves the first thread free to read data again without having to manage memory. This trick although unsafe with higher refresh rates on the order of 10ms offers a decent compromise in providing a usable GUI that's accurate at higher sampling rates.

6 Next Steps

We hope that the continued development of the firmware, hardware testing and algorithms development help move forward the develop of electronics for digital readouts on the LAr. Next steps would be to further integrate these changes into the LASP framework allowing for faster testing and debugging of future projects to come.

Acknowledgments

Special thanks to my supervisors and colleuges, Brigitte, Xingguo, Raphael, Sam, Nick, Maheyer and Kathrin for their help and guidance throughout the summer! I acknowledge that this work was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) Undergraduate Student Research Award and the travel costs was supported by the Institute of Particle Physics (Canada) (IPP). Thanks to the organizers of the CERN Summer Student Program and the IPP Summer Student Fellowship Program for this excellent opportunity. [2]

References

[1] Intel, *Intel® Stratix® 10 10GBASE-KR PHY IP Core User Guide*, .

[2] T.L.C. Group, *ATLAS LAr Calorimeter Phase-II Upgrade*, .

[3] ATLAS collaboration, *ATLAS Liquid Argon Calorimeter Phase-II Upgrade : Technical Design Report*, .

[4] N. Guettouche, S. Baron, S. Biereigel, D.H. Montesinos, S. Kulis, P.V. Leitao et al., *The lpGBT production testing system*, *Journal of Instrumentation* **17** (2022) C03040.

[5] C. Rey, *Optimal filtering of calorimeter signals*, *Thermochimica Acta* **184** (1991) 329.

[6] E. Fullana et al., *Optimal Filtering in the ATLAS Hadronic Tile Calorimeter*, .

[7] D.P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. 10.48550/ARXIV.1412.6980.

[8] F. Chollet et al., "Keras." `https://keras.io`, 2015.