FPGA Development for the LHCb Vertex Locator Upgrade

Nicholas Mead
Student Number: 8064141
School of Physics and Astronomy
University of Manchester

May 16, 2016

Abstract

This document discusses two areas of FPGA development for the LHCb VELO upgrade scheduled to coincide with LHC Long Shutdown 2 in 2019. The areas of development are the selection of a suitable scrambling algorithm and the continued work on the event isolation flagging.

The analysis for three scrambling algorithms, required for data transfer from the front end electronics to the Data Acquisition FPGA, was compared against the theoretical predictions of scrambled data. It was found that the currently implemented VeloPix scrambler is the optimum of the choices but also that an alternative multiplicative scrambler was suitable for computer simulations.

Event Isolations Flagging (EIF) system is intended to identify and flag the easier to re-construct events in order to reduce event pile-up in the computer network. The module is complete as a stand-alone project and ready for implementation into the master FPGA code. Only once the implementation is complete will the routing and resources be tested. Assuming testing is successful, the EIF module will be complete.

Contents

1 Introduction

1.1 Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are silicon based integrated circuit chips. Unlike conventional chips, where the circuit and logic gates are permanently synthesized from silicon transistors at manufacture time, the internal structure of an FPGA can be manipulated to the desired structure of the user. The advantage of FPGAs is the high data transfer rates and versatility they deliver, without the cost of manufacturing purpose built chips. As such, FPGAs are used extensively in the development of new technology and small batch size production. [1]

FPGAs are programmed in a variety of codes known as Hardware Description Languages (HDLs). One of the more common HDLs and the HDL used in this project is Very High Speed Integrated Circuit Hardware Description Languages (VHDL). Programing a circuit in VHDL requires the creation of entities, sometimes refereed to as modules. These entities can be thought of as a *'black boxes'* that compute logic on their inputs and thus form their outputs. The form of this logic computed by the entity is known as the entities' architecture.

When building a circuit larger than one simple function it is often necessary to use more than one entity. In VHDL entities can contain sub-entities within their architecture; entities that do this are known as top-level entities. Top level entities are often used to commute signals between there sub-entities and/or control how sub-entities operate.

1.2 The Standard Model of Particle Physics

Central to the modern study of particle physics is the standard model,

$$
\begin{aligned}
L_{GWL} = &\sum_f (\bar{\Psi}_f(i\gamma^\mu\partial\mu - m_f)\Psi_f - eQ_f\bar{\Psi}_f\gamma^\mu\Psi_f A_\mu) + \frac{g}{\sqrt{2}}\sum_i(\bar{a}_L^i\gamma^\mu b_L^i W_\mu^+ + \bar{b}_L^i\gamma^\mu a_L^i W_\mu^-) \\
&+ \frac{g}{2x_w}\sum_f \bar{\Psi}_f\gamma^\mu(I_f^3 - 2s_w^2 Q_f - I6e_f\gamma_5)\Psi_f Z_\mu - \frac{1}{4}|\partial_\mu A_v - \partial_v A_\mu - ie(W_\mu^- W_v^+ - W_\mu^+ W_v^-)|^2 \\
&- \frac{1}{2}|\partial_\mu W_v^+ - \partial_v W_\mu^+ - ie(W_\mu^+ A_v - W_v^+ A_\mu) + ig' c_w(W_\mu^+ Z_v - W_v^+ Z_\mu|^2 \\
&- \frac{1}{4}|\partial_\mu Z_v - \partial_v Z_\mu + ig' c_w(W_\mu^- W_v^+ - W_\mu^+ W_v^-)|^2 - \frac{1}{2}M_\eta^2\eta^2 - \frac{gM_\eta^2}{8M_W}\eta^3 - \frac{g'^2 M_\eta^2}{32M_W}\eta^4 \\
&+ |M_W W_\mu^+ + \frac{g}{2}\eta W_\mu^+|^2 + \frac{1}{2}|\partial_\mu\eta + iM_Z Z_\mu + \frac{ig}{2c_w}\eta Z_\mu|^2 - \sum_f \frac{gm_f}{2M_W}\bar{\Psi}_f\Psi_f\ \eta. \quad (1.1)
\end{aligned}
$$

The standard model shown in equation 1.1, is a quantum field theory that describes the fundamental particles and how they interact. The aim of this project and document is not to directly understand nor evaluate the standard model, instead to develop the technology used to verify, measure and expand it. Despite being the current best theory

to explain particle interactions, the model is not complete. There are many un-described phenomena, such as the matter domination in the universe, that require physics beyond the standard model in order to be described. To that end, major international efforts, namely in the form of the Large Hadron Collider, aim to gain further knowledge and understanding of the underlying physics of the universe. [2]

1.3 The LHCb Experiment

One experiment at the Large Hadron Collider (LHC) is Large Hadron Collider beauty (LHCb). Located at intersection point 8 of the LHC, LHCb is designed to study rare particle physics phenomena, such as lepton flavor violation and CP violation. LCHb studies the decay modes of the B meson. B mesons are short lived particles with a lifetime of $\sim 1.5ps$. Because of the time dilating effect of special relativity, these hadrons can travel up to a centimeter in the detector before decaying. As such, B meson decays can be identified by decay products that propagate from a secondary vertex.

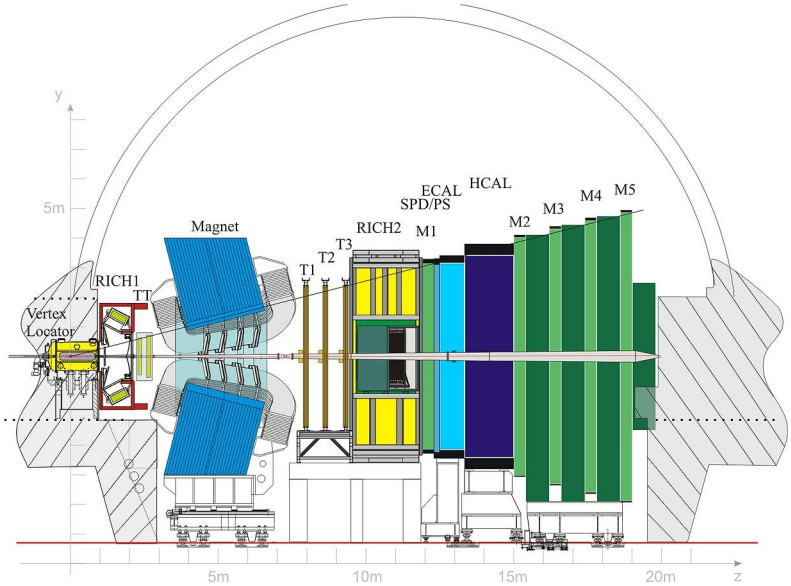

Figure 1.1: The LHCb Detector along the bending plane.

As B mesons are light (in comparison to many other particles studied in the LHC), the decay products are produced at a shallow angle relative to the beam pipe; this is the driving factor in the design of the experiment. LHCb is a single arm forward spectrometer. Surrounding the point of collision is the Vertex Locator (VELO), this high precision detector uses silicon strips to detect ionising particles as they propagate from a collision and provides the coordinates of the particle in terms of R[1] and ϕ[2]. By reconstructing the paths of particles back to the intersection point, it can be identified whether the particular decay particles are a product of the primary vertex[3], or a secondary vertex[4].

[1]Radial distance from the beam pipe.
[2]Azimuthal angle from the beam pipe.
[3]The position at which the protons collided.
[4]The decay point of a short lived particle. i.e. B Meson.

The RICH detector, comprised of two sub detectors located either side of the magnet, uses Cherenkov radiation to deduce the velocity of the particle. The silicon trackers, labeled TT and T1-3 in Figure 1.1, calculate the angle deflection by the magnet. By combining the velocity and angle of deflection, the mass, momentum and energy of the particles can be deduced from simple relativistic kinematics,

$$E^2 = M^2c^4 + p^2c^2. \tag{1.2}$$

The Muon detectors, labeled M1-5 in Figure 1.1, are designed to detect muon's the detector. This is of particular importance in LHCb as Muons can be easily misidentified as charged Pions, due to their similar mass. Pions and Muons are a common decay product of the interactions studied at LHCb, furthering the need to accurately differentiate Muons and Pions.

HCAL and ECAL, shown in Figure 1.1, are hadronic and electric calorimeters respectively. Both measure the total energy of incoming particles. As the calorimeters absorb the particles they detect, any leptonic particle reaching the M2-5 muon detectors can be assumed to be a muon.

1.4 LHCb Upgrade

With advancements in accelerator technology, the detectors must also advance in order to make best use of the accelerators. The LHC is scheduling to increase its luminosity during Long Shutdown 2 (LS2), and as such LHCb will have to cope with this greater luminosity. The front end electronics of LHCb implement a hardware trigger (later followed by a software trigger) and this is limited to a 1MHz maximum readout speed. Post LS2, LHCb will have to cope at a luminosity of $\mathcal{L} = 2.10^{33} cm^{-2} s^{-1}$; this is significantly greater than the current $\mathcal{L} = 4.10^{32} cm^{-2} s^{-1}$. A simple luminosity increase will not significantly reduce the uncertainty for some statistical error dominated channels. To achieve greater statistical significance, greater resolution of the VELO and a fully software based trigger is required. Detailed in the *'LHCb VELO Upgrade Technical Design Report'* [3] the main goals of the 2019 upgrade are as follows:

- Increase the luminosity to $\mathcal{L} = 2.10^{33} cm^{-2} s^{-1}$.
- Read data from the detector at the bunch crossing frequency, 40 Mhz.
- Convert to a purely software based trigger.

1.4.1 VELO Upgrade

Common with its predecessor, the upgraded VELO uses thin, retractable modules. The advantage of this approach is that during collisions, the modules can sit closer than otherwise possible to the beam line. The modules retract for the beam fill, avoiding the radiation damage from the wider fill beams. In order to gain greater resolution of

secondary vertices's, the upgraded VELO will sit at 5.1 mm from the beam at the closest pixel [3]. The current VELO achieves 8 mm [4].

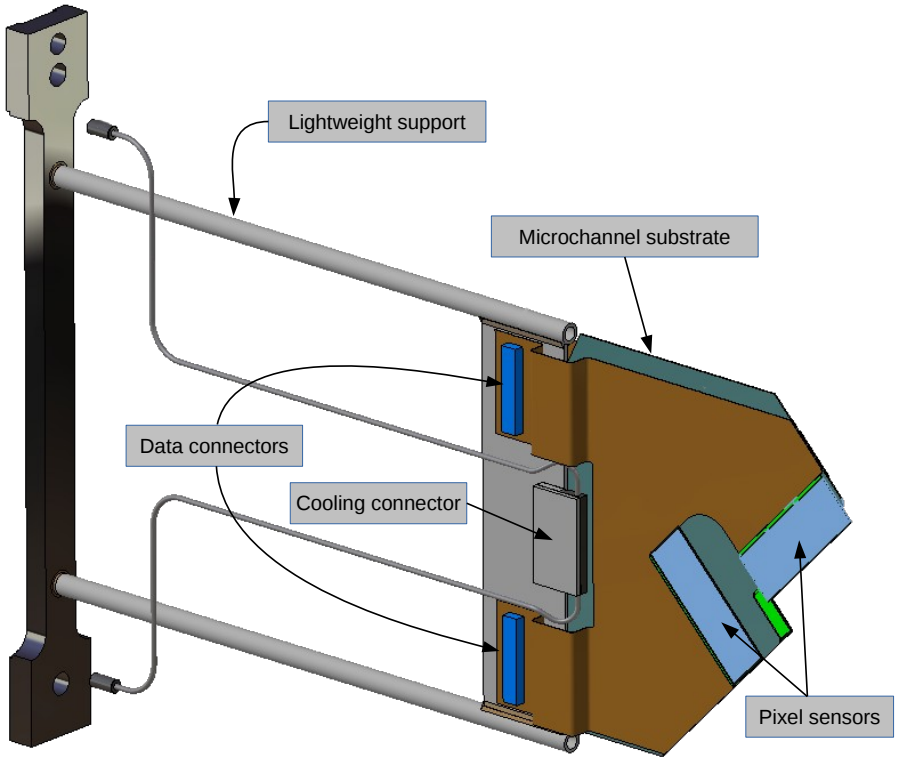

Figure 1.2: The current module design. Two sensors are shown, the remaining two are mounted to the rear face of the module to form two horizontal rows - covering the rightmost area of the module (as viewed in the figure).

As previously mentioned, the current VELO uses silicon strips to detect particles. The upgraded VELO, however, will use silicon pixels. These pixels, $55\mu m \times 55\mu m$ in size and $200\mu m$ thick [3], are arranged in a 256 wide square matrix on a ASIC chip. The pixels are arranged into groups of 8 to form a Super Pixel (SP). The ASIC chips are arranged in a row of 3 and bonded to a sensor. Each module has 4 sensors, 2 per side, as shown in Figure 1.2. The module is cooled by bi-phase CO_2 in micro-channels etched into the micro channel substrate. [3]

The VELO modules will operate in the LHC secondary vacuum. It is separated from the primary vacuum by RF foil that is 3.5mm from the beam-line, at the closest point [3]. The foil is made of 250 μm thick aluminum to reduce its interaction with the collision decay products.

1.4.2 Data Flow and Low Level Interface

FPGAs are used in the Data Acquisition (DAQ) modules for their speed and parallel processing capabilities. The DAQ, in its simplest form, is a series of optical links, a data processing FPGA and a PCIe port for data transfer to the VELO computer system.

The data from each SP is packaged in a 30 bit Super Pixel Packet (SPP). The SPP is comprised of a (from most to least significant bits) 9 bit Bunch Cross ID (BCID); 13 bit SPP location information (horizontal and vertical coordinates); 8 bit SP hit map.

A GWT serialiser in the ASIC forms a 128 bit *'frame'* comprising of a header (1010), four single bit parity flags and four SPPs. The parity flags indicate the parity of the four SPPs as a validation check for downstream processes. The data is then transmitted via an electrical to optical converter though optical fibers to the DAQ.

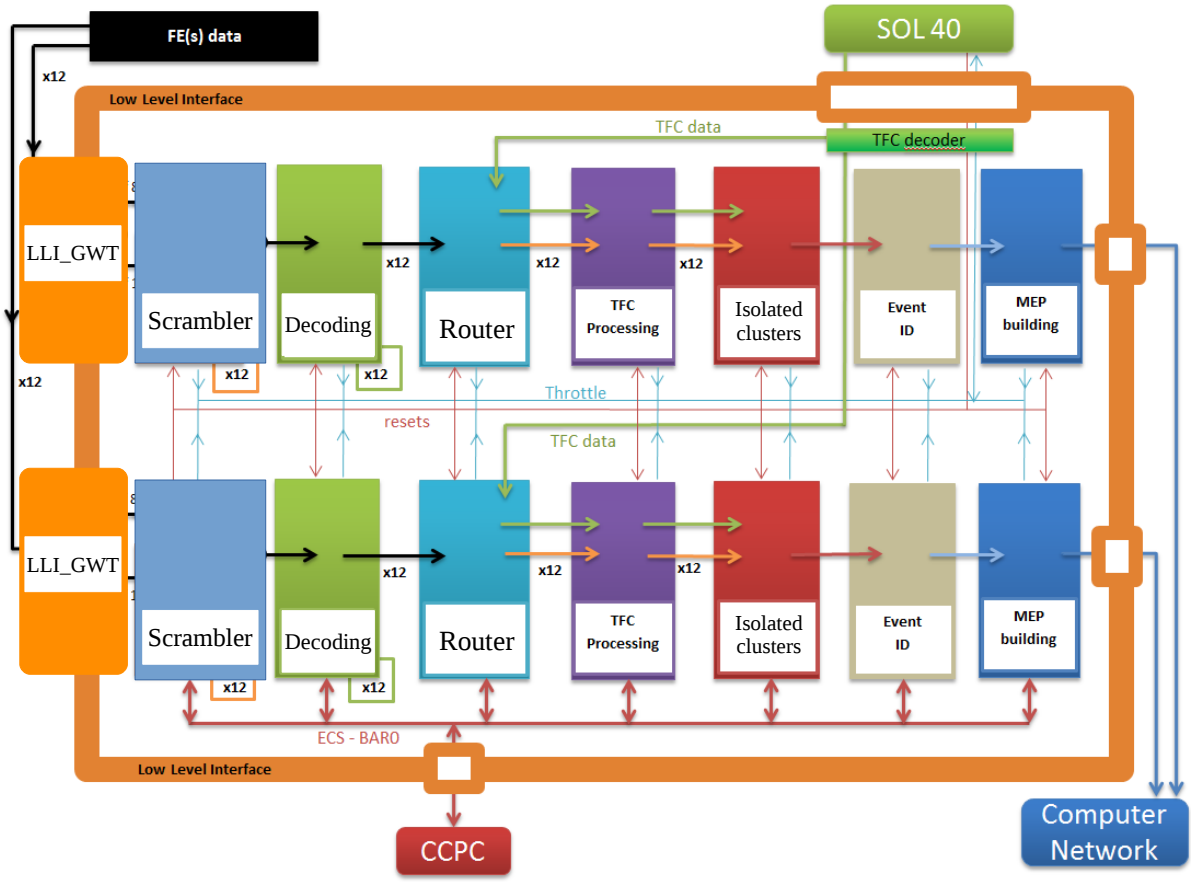

Figure 1.3: A Diagram showing the data flow in the low level interface.

Located on the DAQ FPGA is the Low Level Interface (LLI). The LLI is responsible for the construction of the data from the digital light signals from the optical links. The scrambler is necessary for descrambling the incomming data, this is discussed in further detail later in this document. The decoding block decodes the BCID of each SPP from gray code to binary. Next, the router sorts the data into time order. TFC or Time Fast Control processing is used to interface reset signals from the front end detector with the FPGA. Isolated Clusters, also referred to as Event Isolations Flagging, identifies SPPs that fully describe a track though the detector - again, this is further discussed in this document. Event ID and MEP building together build the processed data into computer compatible formats and allow the data to be matched to signals from other detectors in the experiment.

2 Scrambler

Due to radiation levels inside the detector chamber, the main data processing takes place in a concrete bunker away from the detector. To facilitate this, 20 optical links (per module) are used to transfer the data from the front end VELO to the DAQ FPGA. When communicating data digitally, the transferring module (TX) and the receiving module (RX) must have synchronised clocks. In this case, the GWT serialiser is the TX, and the DAQ is the RX. When achieving a synchronised clock, there are two main approaches:

- Transmit the TX clock with the data to the RX module - used in I^2C and SPI communication.
- Use bit-changes in the data to continuously synchronised the RX clock.

The former of these options, although widely used in conventional electronics, requires a finely tuned clock accounting for all possible delays. The latter, while negating cons of the former, requires data with a high density of transitions to reduce the likelihood of a desynchronisation event. Because delays in the data are possible, the latter option has been selected.

As mentioned, it is necessary to ensure that the data has a large density of transitions before being transmitted from the front-end detector to the DAQ module. However, as the majority of super pixel hit maps are empty, the data has a bias towards *'0'*s. This reduces the frequency of transitions in the data - increasing the probability of a desynchronisation event. It is therefore necessary to scramble the data prior to transmission and descramble the data in the LLI of the DAQ FPGA.

Scrambling and later descrambling the data is not a trivial exercise. The scrambling (TX) module and descrambling (RX) module must use a synchronised *'key'*, that is used in both the scrambling and descrambling processes. In the FPGA, the *'key'* is derived from the previous states of the data. There are two methods investigated for generating this *'key'*:

Additive The *'key'* is generated by evolving the previous *'key'* at each iteration of data using the incoming frame.

Multiplicative The *'key'* is generated from the previous n frames. (Here n is a variable specific to the algorithm).

2.1 Scrambler Options

Three scrambling algorithms have been considered:

Additive Scrambler

This scrambler was originally implemented and used two sets of two-input XOR logic gates. As the name implies, this scrambler used additive key generation which is dependent on all previous input frames since the last reset signal.

Intermediate Scrambler

Created by Karol Hennessy, and deriving its name arbitrarily from the order of consideration, this multiplicative scrambler combines the current and previous frames to generate the *'key'*. Therefore, in the event of desynchronisation, only two frames are lost before the *'key'* is automatically recovered. This feature alone is a significant improvement over the Additive Scrambler.

VeloPix Scrambler

This is the current implemented scrambling algorithm in the DAQ and VeloPix code. Like the Intermediate Scrambler, it uses multiplicative *'key'* generation. However, the VeloPix scrambler is compatible with further constraints enforced by the ASIC, including the number of combinational logic operations. The Intermediate Scrambler was design purely for simulation purposes and as such does not meet the additional ASIC constraints.

2.2 Cross Checks

The main priority when scrambling data, is ensuring that the data is recoverable. For all three scramblers, the algorithm was synthesised in Quartus [5] and simulated in Modelsim [6]. The aim of synthesising and simulating the scramblers in these programs was to ensure that the design was both physical in terms of on-board logic gates, and to check that the scrambled data was recoverable, respectively.

Furthermore, a C++ simulation was created for the three scramblers. This simulation had two main purposes; firstly to cross check the output of the C++ against the Modelsim simulations; secondly to simulate the scrambler over a much larger sample of data as Modelsim simulations are less time efficient. In addition to the cross checks, the C++ code allowed for the injection of a desychrnonisation event, in which the *'key'* is lost. As expected, the Additive Scrambler was unable to recover any data post desychrnonisation, however the intermediate and VeloPix scramblers both recovered the *'key'* after two frames and continued to decode data.

2.3 Algorithm Analysis

For analytical purposes, it is assumed that fully scrambled data is indistinguishable from randomly generated data. For this reason, the three algorithms are not only tested against each other and pre-scrambled simulated QWT data but also randomly generated binary. The randomly generated data was created using the Python *'random'* library, selecting a *'0'* or *'1'* with equal probability. While the Python *'random'* library is only pseudo-random, on the scale of this application (i.e. >> 100,000 frames), it is sufficient for these purposes.

A more mathematically rigorous approach, however, is to evaluate the system abstractly in the framework of statistical physics. In this abstraction, the 120 bit frame (with the header and parity removed) is considered an ensemble; microstates are the particular form of the frames; and macroscopic quantities can be calculated by averaging a large number of frames. For the analysis outlined in section 2.3.1, predictions will be made using these principles and outlined in section 2.3.2.

In the context of the statistical model, it is reasonable to consider the degree of *'scrambledness'* analogous to entropy. This analogy is not dissimilar to the common interpretation of entropy as a measure of disorder. From Boltzmann's equation for entropy,

$$S \propto ln(\Omega) \tag{2.1}$$

where Ω is the number of microstates associated with the macrostate, we learn that this state of maximum entropy is a macrostate with the maximum number of associated microstates.

The entropic argument of Equation 2.1 is not only mathematically founded. For a scramble algorithm to hold for all possible data sets, it must also be capable of outputting all possible permutations. As such, assuming all possible outputs are equally likely, the count of each macroscopic output will be proportional to the number of microstates associated.

2.3.1 Measurements of the Algorithms

To compare the efficiency of the three algorithms in section 2.1, the algorithms where run over the same input data and compared for the following measures:

Number of Transitions Per Frame
This measure counts the total number of bit transitions (i.e. $bit(n) \neq bit(n-1)$) in a 120 bit frame. The header and parity information was not included as they are not scrambled. This is an important test as one of the roles of the scrambler is to maximise the number of transitions.

Common Bit Chain Length
One of the downfalls of the 'Number of Transitions Per Frame' analysis is that the two hypothetical 20 bit frames,

a) 10101010101111111111,
b) 10011001100110011001,

both with 10 transitions, are considered to be equal. However, (b) is clearly a more suitable output for data transfer as (a) has a large probability of desynchronisation due to the long chains of *'1'*s in the rightmost bits. It is therefore also necessary to evaluate the length of common bit chains within the scrambled data as shorter chains are more suitable for data transfer.

Bit Asymmetry

Pre-scramble, the data had a large bias towards *'0'*s due to the majority of the hit maps being empty. Scrambled data, via entropic arguments, *should* show zero bias either way. Therefore, by investigating how the number of *'1'*s - *'0'*s evolves over many frames, any bias in the scrambler can be found.

2.3.2 Statistical Predictions

Number of Transitions Per Frame

The average number of transitions, $< N_\tau >$, is predicted to be,

$$< N_\tau > = \sum_{N_\tau=0}^{n-1} N_\tau \ f(N_\tau) = n \ p_\tau = 60, \tag{2.2}$$

with standard deviation,

$$\sigma_{N_\tau}^{Binomial} = \sqrt{n \ p_\tau^2} = 5.48, \tag{2.3}$$

where p_τ is the probability of the transition - i.e. any given bit being opposite to the previous bit.

Common Bit Chain Length

For a log graph of number of chains of length n, $log(N_n)$, against n for a large sample of data, the gradient would be $log(1 - p_\tau)$. In this case, as $p_\tau = 0.5$,

$$log(1 - p_\tau) = -0.30. \tag{2.4}$$

Bit Asymmetry

The average asymmetry of 1's and 0's, $A_{1,0}$, is expected to be zero.

The error on this value is calculated to be,

$$\sigma_{A_{1,0}} = \sqrt{< A_{1,0}^2 > - < A_{1,0} >^2} = \sqrt{< A_{1,0}^2 >} = \sqrt{n \ \Delta t}, \tag{2.5}$$

where $n\Delta t$ is the number of bits analysed.

Full derivations for the above quoted equations can be found in Appendix A

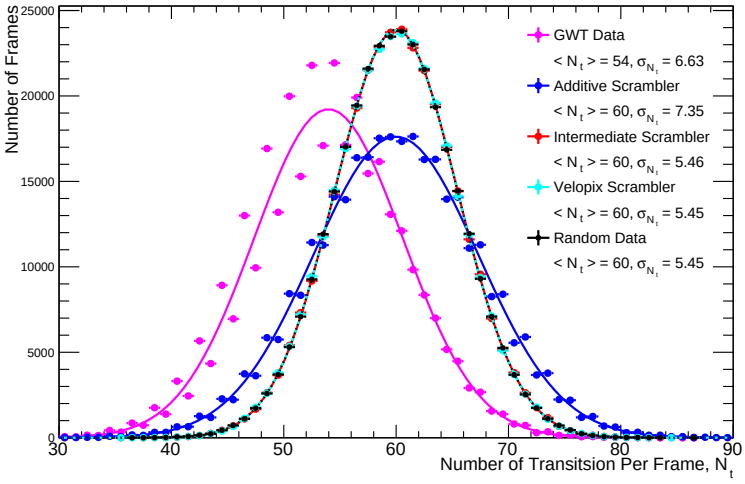

Figure 2.1: Results of the *'Number of Transitions Per Frame'* analysis. The results for the Random Data, Intermediate Scrambler and VeloPix Scrambler overlap.

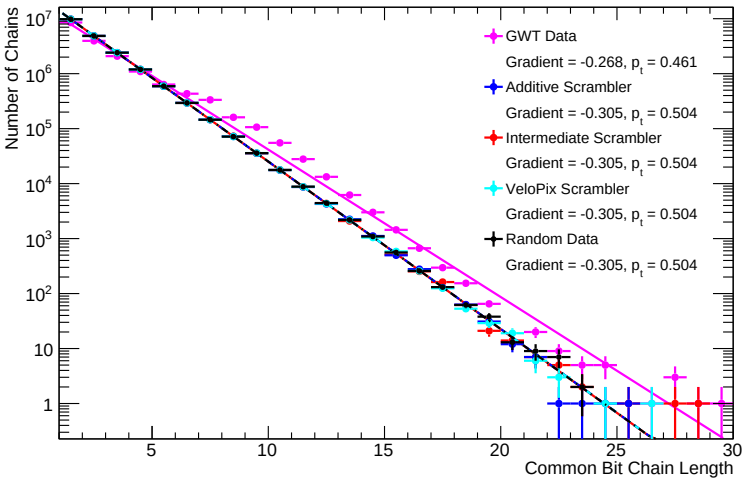

Figure 2.2: Results of the *'Common Bit Chain Length'* analysis. The results for the Random Data, Additive Scrambler, Intermediate Scrambler and VeloPix Scrambler overlap.

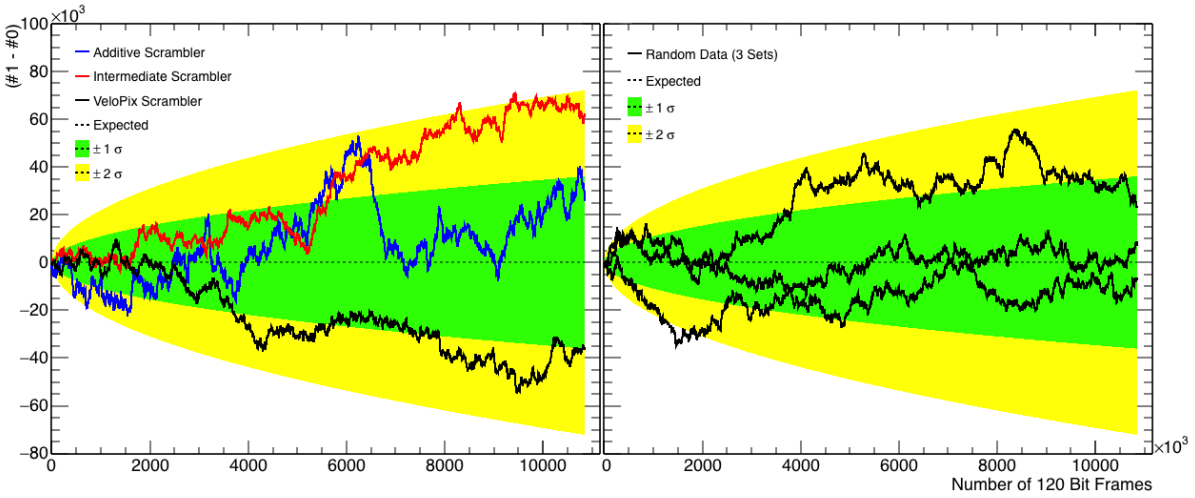

Figure 2.3: The results of the *'Bit Asymetry'* analysis.

2.3.3 Results of Analysis

The results from the *'Number of Transitions Per Frame'* analysis, shown in Figure 2.1, show a strong similarity between the Intermediate and VeloPix Scramblers with the randomly generated data. These results are within 1% agreement with the theoretical predictions for $< N_\tau >= 60$ and $\sigma_{N_\tau} = 5.48$, made in Section 2.3.2. The remarkable consistency between the theoretical predictions and the randomly generated data provides confidence in both the theory, and the scrambled nature of the Intermediate and VeloPix scrambler outputs.

For the *'Common Bit Chain Length'* analysis all three scramblers; the random data, and the theoretical predictions are consistent to within 1%. Comparing the results for the Additive Scrambler, it is shown that while the frequency of longer chains is consistent with random data, the variance of transitions is larger than predicted. Thus, the long and short trains are more locally clustered.

The *'Bit Asymmetry'* of each scrambler, shown in Figure 2.3, is consistent with the theoretical prediction. The deviation of $A_{1,0}$ for the predicted mean of 0 is fully consistent with stochastic noise. The random data also shows consistency. This gives confidence in the assumptions made in Section 2.3.2.

2.4 Conclusion

The consistency of random data and the theoretical predictions justifies the assumptions and approximations made in Section 2.3 and Section 2.3.2. Furthermore, the confirmation of the statistical model allows for accurate comparisons to be made from predicted values and their measured counterparts.

The Additive Scrambler, while consistent with the *'Chain Length'* and *'Bit Asymmetry'*

	$< N_\tau >$	σ_{N_τ}	Gradient	p_τ
GQT data	54	6.63	-0.268	0.460
Additive Scrambler	60	7.35	-0.305	0.504
Intermediate Scrambler	60	5.45	-0.305	0.504
Velopix Scrambler	60	5.46	-0.305	0.504
Random Data	60	5.45	-0.305	0.504
Theoretical Prediction	60	5.48	-0.3	0.5

Table 2.1: The combined results of the algorithum analysis.

analysis, has a variance in the transition frequency that leads to the conclusion that long and short chains are locally clustered. This is not ideal for data transfer. Many sequential long chains increase the probability of TX-RX clock desynchronisation. Furthermore, the additive scrambler will not recover from this loss of synchronisation, as the *'key'* will never be recovered without a common reset signal.

The Intermediate Scrambler produced an output consistent with random data. This makes the algorithm suitable for data transfer. As already mentioned, however, the scrambler is designed for computer simulation. As such, it is not suitable for implementation as it does not meet the addition requirements of the ASIC.

The VeloPix Scrambler, like the Intermediate Scrambler, produces a statistically scrambled output. Furthermore, the algorithm is in line with the additional requirements of the ASIC. As such, it is ideal for implementation, and hence is currently the choice algorithm for use in the 2019 VELO upgrade.

3 Event Isolation Flagging

One challenge of increasing the readout speed of the detector is processing the data that is produced. Because of this, any pre-processor functions executed in the FPGA reduces the load and processing time of the computer system. One area where the DAQ's FPGA will be used for this purpose is in Event Isolation Flagging (EIF).

When particles traverse the VELO there is a finite probability that they will pass though the boundary of two or more Super Pixels. This will cause multiple SPPs to be created for the same particle path. For such paths, the computer is required to search all SPP corresponding to the event, and identify SPP's that relate the the same particle track. However, as the computer doesn't have information on which SPP's do and do not have adjacent SPP's, the computer is required to search for neighboring SPP's for each SPP in the BCID. This is an inefficiency which can be solved withing the FPGA, before the

computer processes the data.

The aim of EIF is to identify the SPPs that completely describe the particles interaction with the module and flag such SPP's as isolated. The computer can then save time by negating the need to search for adjacent SPP's. By speeding up to the processing of such tracks, the pile-up of data is reduced. This thus reduces the probability of the computer needing to reject incoming data or for the detector to stop collecting data.

3.1 Router and MEP Interface

Retrieving and delivering BCID information withing the FPGA is a challenge in its self. In the router data is not processed in BCID order, instead information from many BCID's are processed simultaneously. Adding complication, the EIF-MEP interface must identically match that of the Router-IEF. This is because the IEF module is not a priority for implementation in the final production for the DAQ FPGA. Thus, the IEF module must be easily removable without the need for a significant re-design.

The decision was made to implement swinging buffers between the Router-EIF and EIF-MEP interfaces. Each swinging buffer consists of two sets of 16 RAMs, each containing the SPPs of 32 BCIDs and an additional count ram, that contains the number of SPPs in each BCID. During one swing cycle, one of the interfacing modules can write information to one set of RAMs in the buffer while the second module reads from the second set of Rams. This approach has two main benefits: firstly there can be no read-write conflicts where the EIF or MEP read data from a BCID before writing is complete; secondly the calculation deciding if a BCID can be processed before being overwritten is significantly simplified. In addition, the swinging buffer allows for the parallelization of EIF modules. This reduces the load on each EIF unit and prevents the BCIDs from being bypassed due to lack of processing time.

3.1.1 RAM Interfacing

Communicating with a RAM block, either to read (rd) or write (wr), requires the use of three signals.

Enable (en) signaling to the RAM to read or write.

Address (addr) identifying the address in the RAM to read or write from.

Data This signal is used to transfer the data into out of the RAM.

On the falling edge of the clock (clk), if the wr_en or rd_en signals are high, the RAM will write the wr_data signal to the specified address or output the data stored in the address to the rd_data signal respectively. The count RAM (ct) is the only exception as it does

not have a read enable. Instead the count RAM will update the data signal every falling edge to whatever address the ct_addr signal specifies.

The RAMs used in the FPGA do not have a large enough output bandwidth to output the whole of a BCID in one address. Each address in the RAM contains the information of 16 SPPs. Therefore, if a BCID contains greater than 16 SPPs its required to iterate over several addresses on the RAM and combine the data to form a complete set of SPPs. Similarly, when writing more than 16 SPPs one must write the data incrementally to adjacent addresses. One consequence of this is that the RAMS must be flashed clear on the buffer swing to prevent old data being used twice.

In addition to the 16 SPP max read/write speed, each BCID has only 32 address. Therefore a max of 512 SPPs can be stored in one BCID.

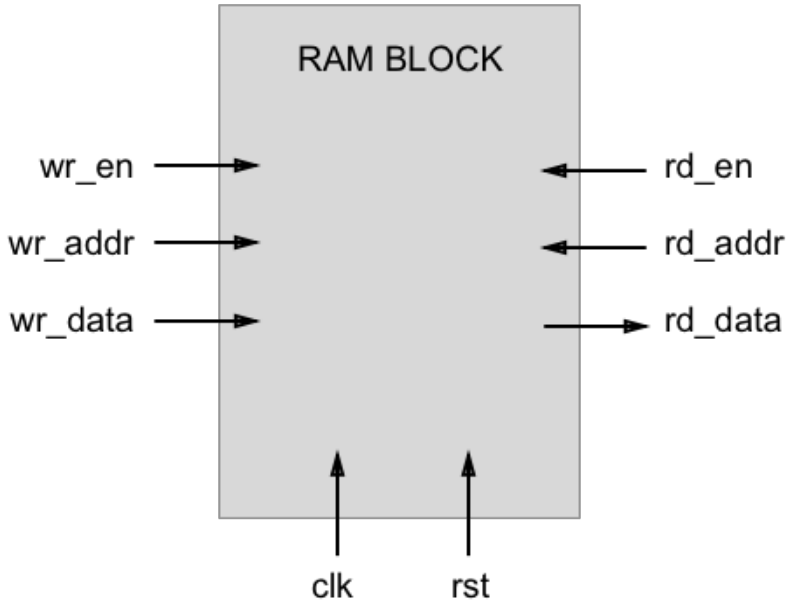

Figure 3.1: A diagram showing the signals required for a simple RAM block.

3.2 Top Level EIF Module

The EIF data processing comprises of two main steps.

- Bubble sort the SPP's by row ID.
- Compare SPP row ID to adjacent SPP's and compute if isolated.

One of the major constraints of the data processing is the max number of SPP's that the Bubble sort can accept. It is therefore necessary to include in a bypass system to the process that allows BCID's with too many SPP's to be passed onto the next stage

without being processed. Such BCID's will be processed in the same way for the rest of the FPGA and CPU processes, but will not benefit from the time saving property of the EIF.

As such the top level entity contains three child entities.

Active Controller
: This entity reads in the BCID's to an array of data processors and writes the results to the output ram.

Interface FIFO
: This First In First Out entity is written to by the Active Controller and is used to inform the Bypass Controller of which BCID's have and have not been bypassed.

Bypass Controller
: This entity passing data across the EIF system without data processing.

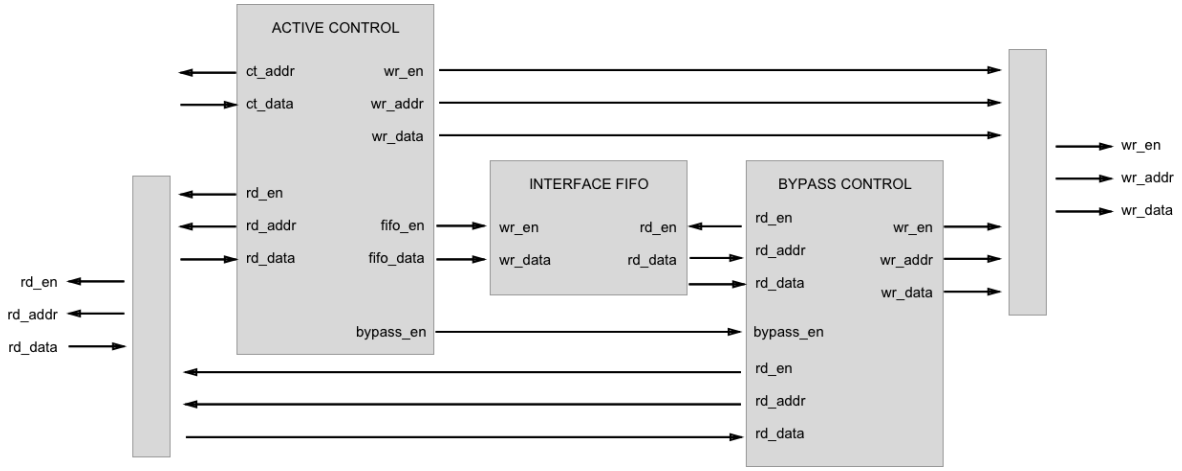

Figure 3.2: A diagram showing the signal arrangement of the top level EIF entity.

By monitoring the bypass_en signal, the EIF top level diagram can detect whether the Active Controller or Bypass Controller read and write signals need to be routed to the outputs. This can be seen in Figure 3.2 as the two slim grey rectangles at either side of the diagram. When the bypass_en signal is low, the Active Controller is linked with the RAMs. The signals pass though the EIF top level entity without delay. Conversely, when the bypass_en signal is high, the Bypass Controller signals pass the the RAMs.

3.3 Active Controller

At the beginning of a swinging buffer cycle, the active control checks if the first BCID both has data and can fit into the data processing units. Then, if the BCID is suitable for processing it is read into a data processor and a value of 0 is written to the Interface

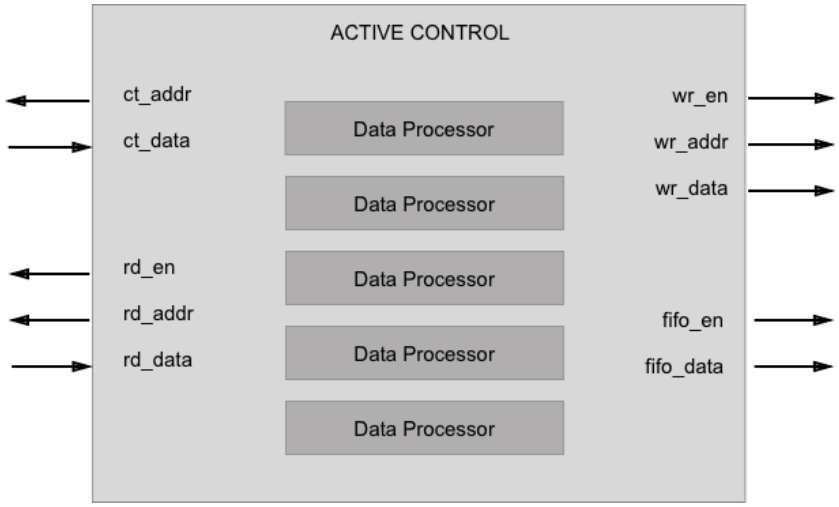

Figure 3.3: A diagram showing the structure of the Active Controller entity. This diagram shows the presence of five Data Processor entities, however any number of Data Processors may be used.

FIFO and the Active Controller moves to the next address. If the number of SPPs is too large or if it is zero, the number of SPPs is written to the Interface FIFO, and again the Active Controller movies to the next address.

Once the Active Controller has finished with the final address and all data processors have had their output written to the MEP Interface RAMs, the bypass_en signal is set high and the Active Controller waits until the next swinging buffer.

3.4 Data Processor

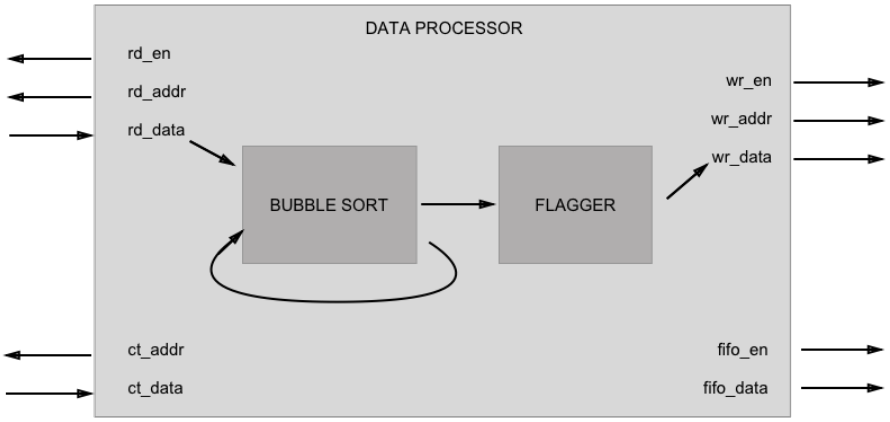

Figure 3.4: A diagram showing the structure of the Data Processor entity.

The data processor computes the sort algorithm, passes the sorted data to the flagging module and then outputs the data. Not shown in Figure 3.4 for simplicity, the data processor uses validation signals to communicate with the active controller to ensure it is not passed data too early and that the data is outputted correctly after the process.

In addition to the SPPs being passed into the Data Processor, the BCID and number of SPPs must also be given. The BCID is used later when writing the information to the MEP Interface RAM and the SPP count is used in the bubble sort process.

3.4.1 Bubble Sorting

Bubble sorting is the process of sequentially comparing adjacent elements of a list and swapping the element if necessary. For example, if sorting into ascending order and comparing the value 5 and 3, a swap would be needed. However, comparing 3 and 5 in the same example would not need a swap. When sorting an arbitrary list, we can define an 'even-odd' comparison as one comparing an even placed element with an odd one (i.e. element 4 with element 5). An 'odd-even' comparison is the logic opposite (i.e. element 5 with element 6).

Bubble sorting, when implemented in series processing, is a relatively slow sorting algorithm. At worst case, Bubble sorting requires n^2 iterations to complete the sort of a list of size n. However, FPGAs can easily parallel process. By making $\frac{n}{2}$ comparisons simultaneously (all even-odd or odd-even pairs), FPGA Bubble Sorting in the worst case scenario only requires n iterations. This is made clear in Figure 3.5.

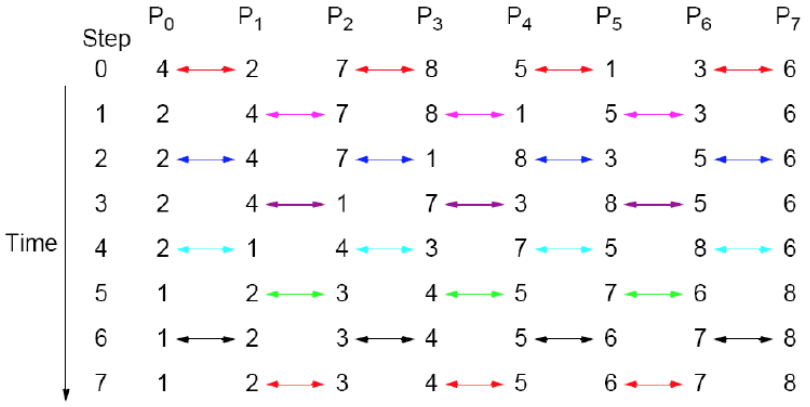

Figure 3.5: A diagram showing Bubble Sorting in an FPGA.

When bubble sorting in the data processor, the SPPs are cycled around a feedback loop until sorted. There are three approaches to calculating when the sort is complete:

Static:
Sorting for the worst case scenario assuming the max number of SPPs have been inputted.

Semi-Static:
Sorting for the worst case scenario for the actual number of SPPs inputted.

Dynamic:
Checking if swaps have been made after each iteration and classing the data as sorted once the odd and even comparisons sequentially make no swaps.

Intuitively, the semi-static method is superior to the static method, thus the static option is immediately rejected. There are pros and cons of the both the semi-static and dynamic options. The dynamic sort will save time on almost sorted data sets, however for short or heavily disordered datasets the dynamic sort isn't every efficient as it requires two cycles without comparison to exit. The semi-static method, while waisting time on almost sorted data, uses less resources than the dynamic method and for smalled data sets there is less opportunity to save time.

Deciding between the semi-static and dynamic sorting methods required further research. The two methods where simulated and compared on time saved by implementing a dynamic sort over semi-static. The data used was a time ordered simulated output of the VELO. If the dynamic sort took longer than the semi-static, it was given a time saved of zero, this is because the dynamic sort can be setup with a semi-static exit strategy.

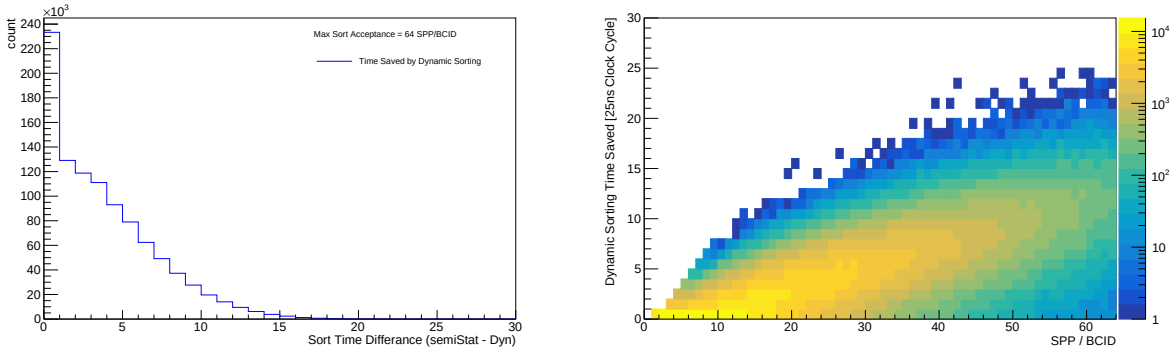

Figure 3.6: The time saved by implementing a dynamic sort method over a semi-static method. The left graph show absolute frequency of time saved. The right graph shows how the distribution of time saved relates the number of SPPs, plotted on a log(z) scale.

It is clear from Figure 3.6 that the dynamic sort method does not significantly reduce the sort time. For this reason, the decision has been made to implement a semi-static method. This will reduce the resources usage of the EIF module, while not significantly increasing the time required to sort the data.

3.4.2 Isolated SPP Tagging

Inside the Flagger entity shown in Figure 3.4, the flagger compares each SPP in the BCID against the adjacent SPPs. If the row ID of both adjacent SPPs is greater than 1 row different, the SPP is classed as isolated. This is shown by setting the most significant bit of the SPP to 1.

In the special case that the SPP is at the edge of the ASIC chip, it is never flagged as isolated. This is because other SPPs from adjacent sensors may also relate to the same track. However, as the two half modules are processed in different data channels, this deduction cannot me made until the data is combined in the CPU.

Furthermore, two isolated SPPs that happen by random chance to be in the row but different column is not be flagged. With the pixel density of the ASIC chip, this situation is considered too unlikely to warrant further sorting and checking. If two isolated SPPs are in the same or adjacent columns, the lack of isolation flag will not affect the CPU processing. The SPPs will be checked for neighboring SPPs and when none are found, treated the same as isolated SPPs. While this is less time efficient in the CPU the extra processing required to avoid such a situation would negate the benefits.

3.4.3 Max SPP Count Per BCID

In order to determine the max number of SPPs the data processor should accept without the need to bypass, the data was analysed to determine the fraction of data that would be accepted at different cutoff values. This analysis focuses on two conflicted ideals, the data processor must be able to processor the majority of the data while reducing the amount of resources required.

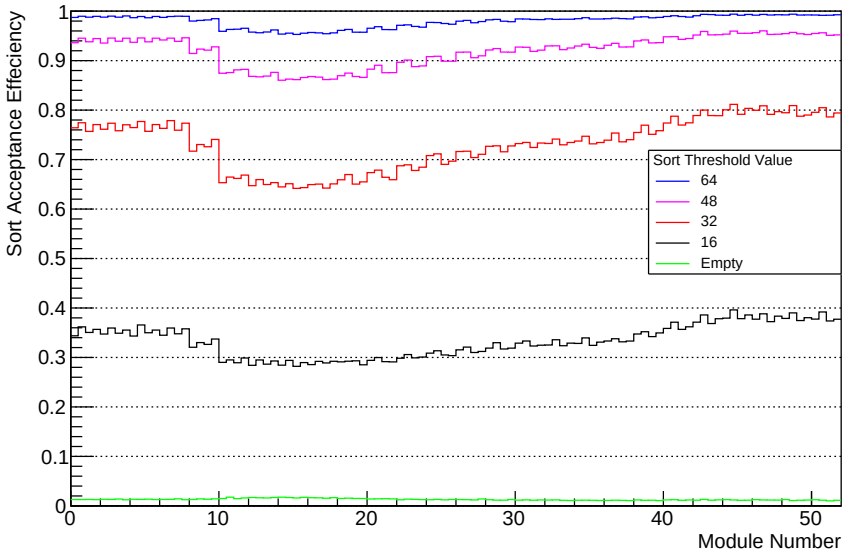

Figure 3.7: The fraction of BCID's accepted at different threshold values.

The data structure in Figure 3.7 is as expected. The modules around 15 are closest to the interaction point and thus have higher occupancy. It is therefore expected that these modules will accept less BCID's for data processing.

Furthermore, there is a diminishing return on the faction of data accepted. A threshold value above 64 would not yield a significant increase in the number of accepted BCID's.

The value of the sort acceptance, however, is easily modified in the VHDL code and can thus be fine turned once the true resources are calculated.

3.5 Bypass Controller

The Bypass controller is the simplest of the entities the EIF. Once the bypass_en signal is received as high by the Bypass Controller, the information from the Interface FIFO is read in one by one. If the value read in is zero, the corresponding BCID does not require bypassing. If the value is greater that zero, the corresponding BCID is read out of the Router Interface RAMs and wrote into the same address of the MEP Interface RAMs. This continues until the Interface FIFO is empty, at which point the Bypass Controller waits for its enable signal to rise back to high. At this point the swinging buffer will have swung, the Active Controller will have finished is process and it will be time for the Bypass Controller to begin its process again.

For each BCID that is bypassed, the time taken is only two cycles longer than the total time it would take to just read or write the data. This is because one clock cycles is used to read in from the FIFO and identify that the BCID requires a bypass, and one clock cycle is used to move the data from the read-in register in the Bypass Controller to the write-out register.

At no point does the Bypass controller contain the whole information for one BCID. This is because the read and write cycles happen synchronously. The only possible scenario for a hole BCID to be contained in the Bypass Controller is for the BCID to contain less than 32 SPPs, however considering Figure 3.7 it is highly likely that the sort acceptance will be much greater than this value.

3.6 Timing

One of the major advantages of the swinging buffer described in Section 3.1 is that the timing calculations are greatly simplified. With 512 BCID's, each occurring at a rate of 40MHz, we have a total of 2048 clock cycles at 160MHz before the swinging buffer. The Router is limited to writing 1 SPP per ram per clock cycle, meaning each ram can only ever contain 2048 SPPs at once. This, although a worst case scenario, is an average of 64 per BCID.

Consider a scenario where a a RAM contains 2048 SPPs, shared in 4 BCIDs of 512 each (The worst case bypass scenario). 32 clock cycles would be used in flagging the BCIDs for bypass and skipping (The 0 SPP BCIDs are still evaluated). 28 clock cycles are then used skipping the empty BCID's. Each BCID of 512 SPP will require 34 clock cycles to bypass.

The total time taken is thus,

$$32 + 28 + 4 \times 34 = 196. \tag{3.1}$$

One EIF modules could not bypass all 16 RAMs in this scenario as this would require 3136 clock cycles. However, two parallel EIF modules could bypass 8 RAMs each taking 1568 clock cycles.

In the opposite extreme, consider all 2048 spread evenly at 64 SPPs per BCID. Each BCID would require:

1 clock cycle to be identifiable for sorting.

4 clock cycles to be read into the processor.

64 clock cycles to be sorted.

1 clock cycle to be flagged.

4 clock cycles to be wrote onto the MEP interface RAM.

At 74 clock cycles per BCID, the total processing time would be,

$$74 \times 32 + 32 = 2400, \tag{3.2}$$

where the additional 32 clock cycles are from the bypass skipping each BCID.

2400 clock cycles is too many. Even though this scenario is very unlikely, the EIF must be able to process 1 RAM minimum in the worst case scenario to be viable for implementation.

The solution is to parallelise the data processors in the Active Controller. If 16 data processors are used, the first data processor will finish before the 17th BCID is ready for processing. The process will therefore need 32 iterations of 5 clock cycles to read in all the data, 65 cycles to sort and flag the last BCID and 4 to read out. This brings the total processing time for one RAM to,

$$32 \times 5 + 65 + 4 + 32 = 261. \tag{3.3}$$

In this case, 4 EIF modules could process 4 of the 16 RAMs each in 1044 clock cycles.

If the sort threshold is greater than 64, these numbers will be affected. Its worth noting that as the FPGA development is still active, that the values and limits may change. For this reason, one should make any solid conclusions from these calculations and instead view them at advisory.

3.7 Conclusion and Current Stage of Development

Currently, the EIF module is built in VHDL as a stand alone project and ready for testing with simulation data. The module, as described in Section 3.6, can process up to 4 RAMS with certainty though this assumes a sort threshold of 64 SPP.

Furthermore, the calculations are made without consideration of the LHC fill scheme. When the LHC ring is filled, not all bunches are able to be filled and therefore a guaranteed number of BCID will be empty. This is not consistent across swinging buffer however, and thus was not taken into consideration.

One of the major features of the project is the versatility of module designed. As many factors affecting the true values used in the project are still to date undetermined, the project is driven from a single list of constants. Altering the constant list allows the implementation engineer to easily scale the project to their needs without having to change any of the VHDL entity files.

4 Future Development

With the EIF module designed and ready for testing, the next stage of development is to integrate the EIF into the master code for the FPGA. From here, tools such as Quartus are able to design the signal routes within the FPGA. It is still unclear if this will be possible, however the only way to find out it to implement the EIF into the full FPGA code, compile and then test on the chip itself.

Limitations may be due to resources or routing of signals. One can hope, but never make any real deductions as to the feasibility of the project.

One of the reasons that information on EIF implementation is vague is because the FPGA currently planned to be used is not yet on the market. Therefore many of the necessary test are not possible at this point in time. Engineering prototypes are available, however the architecture of these chips are likely to differ from the production model. As unsatisfactory as one may find this result, the destiny of the project from here is still to be decided as such a decision not possible at this time.

5 Acknowledgments

I would like the acknowledge Pablo Rodriguez, Marco Gersabeck and Chris Parkes for their continued support and supervision. I also acknowledge Benjamin Jeffrey for his equal role in the Scrambler Analysis and Event Isolation Flagging. I acknowledge Toby Nonnenmacher also, for his equal role in the Scrambler Analysis. I further acknowledge Karol Hennessy for the creation of the simulated VELO data and the members of the LCHb collaboration for their inividual contributions to the LHCb project.

References

[1] "Toward FPGA-Enabled Scientific Computing". In: *Design Test of Computers, IEEE* 28.4 (July 2011), pp. 4–4. ISSN: 0740-7475. DOI: 10.1109/MDT.2011.91.

[2] Cern. *The Standard Model*. 2015. URL: http://home.cern/about/physics/standard-model (visited on 12/2015).

[3] LHCb Collaboration. *LHCb VELO Upgrade Technical Design Report*. Tech. rep. CERN-LHCC-2013-021. LHCB-TDR-013. Geneva: CERN, Nov. 2013. URL: https://cds.cern.ch/record/1624070.

[4] CERN. *LHCb VELO Project*. 2015. URL: http://lhcb-vd.web.cern.ch/lhcb-vd/html/project.htm.

[5] Altera. *Quartus Prime Software*. 2015. URL: https://www.altera.com/products/design-software/fpga-design/quartus-prime/overview.html (visited on 12/2015).

[6] Mentor Graphics. *ModelSim - Leading Simulation and Debugging*. 2015. URL: https://www.mentor.com/products/fpga/model/ (visited on 12/2015).

[7] Kurt Jacobs. *Stochastic Processes for Physicists - Understanding Noisy Systems*. Cambridge University Press, 2010. ISBN: 9780521765428.

A Statistical Derivations

Number of Transitions Per Frame

Consider a particle in a symmetric, discrete time-dependent, two state system,

$$p_0(t) = p_1(t) = 0.5 \quad : \quad \forall\ t \in \mathbb{N}, \tag{A.1}$$

At each time iteration,

$$p_{i \to j}(t) = 0.5 \quad : \quad i, j = [0\ 1], \quad \forall\ t \in \mathbb{N}. \tag{A.2}$$

However, assuming zero bias and detailed balance, as $p_{1\to0}(t)$ is equal in both probability and importance to $p_{0\to1}(t)$, the probability of a bit change shall henceforth be refereed to as $p_\tau(t)$.

Over an n step process, analogous to an n bit frame, the probability distribution of the number of transitions N_τ is given by Binomial statistics,

$$f(N_\tau) = \frac{n!}{N_\tau!(n - N_\tau)!}\ p^{N_\tau}\ (1-p)^{n-N_\tau} \tag{A.3}$$

Simplified for the special case $p = p_\tau = 0.5$,

$$f_\tau(N_\tau) = \frac{n!}{N_\tau!(n - N_\tau)!}\ (p_\tau)^n \tag{A.4}$$

For $n = 120$, we can calculate,

$$< N_\tau >^{Binomial} = \sum_{N_\tau=0}^{n-1} N_\tau\ f(N_\tau) = n\ p_\tau = 60 \tag{A.5}$$

$$\sigma_{N_\tau}^{Binomial} = \sqrt{n\ p_\tau^2} = 5.48 \tag{A.6}$$

Furthermore, when considering the entropic argument of equation 2.1, the number of microstates corresponding to each macrostate N_τ can be related to equation A.4,

$$\Omega_\tau \sim \frac{n!}{N_\tau!(n - N_\tau)!} \tag{A.7}$$

$$< N_\tau >^{Entropic} = MAX[S_\tau] = MAX[\Omega_\tau] \tag{A.8}$$

This can be numerically solved,

$$< N_\tau >^{Entropic} = 60 \tag{A.9}$$

While the result of equation A.9 does not contibute anything new, it is important as a *'sanity check'*. Because the system can be described as in section 2.3, it would indicate a problem in the theoretical framework if the result did not match.

Common Bit Chain Length

The probability of a chain of length n is,

$$p_n = p_1(1-p_\tau)^{n-1}, \quad : \quad n \in \mathbb{N}, \quad n > 1 \tag{A.10}$$

where p_1 is the number of chains of length 1. As $p_1 = N_0(1-p_\tau)$, where N_0 is the total number of chains,

$$\frac{N_n}{N_0} = (1-p_\tau)^n, \quad : \quad n \in \mathbb{N}, \quad n > 1 \tag{A.11}$$

where N_n is the number of chains of length n. Taking the logarithm of both sides,

$$\begin{aligned} log\left(\frac{N_n}{N_0}\right) &= n\ log(1-p_\tau), \\ log(N_n) &= n\ log(1-p_\tau) + log(N_0). \end{aligned} \tag{A.12}$$

Therefore, for a graph of $log(N_n)$ against n for a large sample of data, the gradient would be $log(1-p_\tau)$. In this case, as $p_\tau = 0.5$,

$$log(1-p_\tau) = -0.30\ . \tag{A.13}$$

Bit Asymmetry

$A_{1,0}$, the asymmetry of *'1'*s and *'0'*s is defined as,

$$A_{1,0} = N_1 - N_0, \tag{A.14}$$

where N_1 and N_0 are the number of *'1'*s and *'0'*s respectively. We can consider the evolution of $A_{1,0}$ with frame t of size n as a stochastic iterative map with zero deterministic growth [7],

$$A_{1,0}(nt + n\ \Delta t) = A_{1,0}(nt) + \mathcal{N}(nt). \tag{A.15}$$

Where $\mathcal{N}$ is an independant random variable picked from a gaussian distribution. While $A_{1,0}(t) \in \mathbb{Z}$, in the limit of large nt we can approximate that $A_{1,0}$ is continous.

If we consider the moments of $A_{1,0}$,

$$< A_{1,0}(nt = M\ n\ \Delta t) > = \sum_{m=0}^{M-1} \mathcal{N}(m\ n\ \Delta t), \tag{A.16}$$

$$\begin{aligned} < A_{1,0}(nt = M\ n\ \Delta t)^2 > &= \sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \mathcal{N}(m\ n\ \Delta t)\mathcal{N}(m'\ n\ \Delta t)\ \delta_{mm'} \\ &= \sum_{m=0}^{M-1} < \mathcal{N}(m\ n\ \Delta t)^2 > . \end{aligned} \tag{A.17}$$

Clearly, in Equation A.16, $< A_{1,0} >= 0$. In Equation A.17, we assume the variance is of form $(n\ \Delta t)^\alpha$ [7]. Then,

$$< A_{1,0}(nt = M\ n\ \Delta t)^2 > = M(n\ \Delta t)^\alpha. \tag{A.18}$$

Running the analysis over the frames $t = 0$ to t_f, the number of bits sampled is $M = t_f/n\ \Delta t$. Substituting this into Equation A.18,

$$< A_{1,0}(nt = M\ n\ \Delta t)^2 > = t_f\ (n\ \Delta t)^{\alpha-1}. \tag{A.19}$$

Considering the three cases of α in the approximation of continous $n\Delta t$:

- $\boldsymbol{\alpha > 1}$: Here $A_{1,0} \to 0$ as $\Delta t \to 0$.
- $\boldsymbol{\alpha < 1}$: Here $A_{1,0} \to \infty$ as $\Delta t \to 0$.
- $\boldsymbol{\alpha = 1}$: This is the only sensible choice.

With $\alpha = 1$,

$$< A_{1,0}(nt = M\ n\ \Delta t)^2 >= M(n\ \Delta t). \tag{A.20}$$

And thus,

$$\sigma_{A_{1,0}} = \sqrt{< A_{1,0}^2 > - < A_{1,0} >^2} = \sqrt{< A_{1,0}^2 >} = \sqrt{n\ \Delta t}. \tag{A.21}$$