

FPGA Development for the LHCb Vertex Locator Upgrade

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

May 12, 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 Flaging (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. Despite a bug being identified in the simulated data, the VHDL development is ongoing. A bubble sorting algorithm, implemented by the EIF system, is ready for testing. Once the EIF system is complete, it will be implemented in the master code for the low level interface.

Contents

1	Introduction	1
1.1	Field Programmable Gate Arrays	1
1.2	The Standard Model of Particle Physics	1
1.3	The LHCb Experiment	2
1.4	LHCb Upgrade	3
1.4.1	VELO Upgrade	3
1.4.2	Data Flow and Low Level Interface	4
2	Scrambler	5
2.1	Scrambler Options	6
2.2	Cross Checks	7
2.3	Algorithm Analysis	7
2.3.1	Measurements of the Algorithms	8
2.3.2	Statistical Predictions	9
2.3.3	Results of Analysis	11
2.4	Conclusion	13
3	Event Isolation Flagging	14
3.1	Module	15
3.2	Bubble Sorting	15
3.3	Isolation Checking	16
3.4	Data Train Overflow	16
3.5	Current Stage of Development	18
4	Future Development	19
5	Acknowledgments	19

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 synthesised 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. [?]

FPGAs are programmed in a variety of languages 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). Programming a circuit in VHDL requires the creation of entities, sometimes referred 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 entity's 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 compute signals between their 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,

$$\begin{aligned}
L_{GWL} = & \sum_f (\bar{\Psi}_f (i\gamma^\mu \partial_\mu - m_f) \Psi_f - e Q_f \bar{\Psi}_f \gamma^\mu \Psi_f A_\mu) + \frac{g}{\sqrt{2}} \sum_i (\bar{a}_L^i \gamma^\mu b_L^i W_\mu^+ + \bar{b}_L^i \gamma^\mu a_L^i W_\mu^-) \\
& + \frac{g}{2x_w} \sum_f \bar{\Psi}_f \gamma^\mu (I_f^3 - 2s_w^2 Q_f - I_6 e_f \gamma_5) \Psi_f Z_\mu - \frac{1}{4} |\partial_\mu A_v - \partial_v A_\mu - ie(W_\mu^- W_v^+ - W_\mu^+ W_v^-)|^2 \\
& - \frac{1}{2} |\partial_\mu W_v^+ - \partial_v W_\mu^+ - ie(W_\mu^+ A_v - W_v^+ A_\mu) + ig' c_w (W_\mu^+ Z_v - W_v^+ Z_\mu)|^2 \\
& - \frac{1}{4} |\partial_\mu Z_v - \partial_v Z_\mu + ig' c_w (W_\mu^- W_v^+ - W_\mu^+ W_v^-)|^2 - \frac{1}{2} M_\eta^2 \eta^2 - \frac{g M_\eta^2}{8 M_W} \eta^3 - \frac{g'^2 M_\eta^2}{32 M_W} \eta^4 \\
& + |M_W W_\mu^+ + \frac{g}{2} \eta W_\mu^+|^2 + \frac{1}{2} |\partial_\mu \eta + i M_Z Z_\mu + \frac{ig}{2c_w} \eta Z_\mu|^2 - \sum_f \frac{gm_f}{2M_W} \bar{\Psi}_f \Psi_f \eta. \quad (1.1)
\end{aligned}$$

The standard model shown in equation 1.1, is a quantum field theory that describes the fundamental particles and how they interact. While this document does not require or attempt a detailed understanding of the intricate detail of the standard model, the aim of many particle physics experiments is to verify, measure and expand the model. Despite

being the current best theory to explain particle interactions, the model is not complete. There are many undescribed 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. [?]

1.3 The LHCb Experiment

One experiment at the Large Hadron Collider is Large Hadron Collider beauty (LHCb). Located at intersection point 8, LHCb is designed to study rare particle physics phenomena, such as lepton flavour violation and CP violation. LHCb studies the decay modes of the B meson. These hadrons travel on the order of μms in the detector before decaying. As such, B meson decays can be identified by decay products that propagate from a secondary vertex.

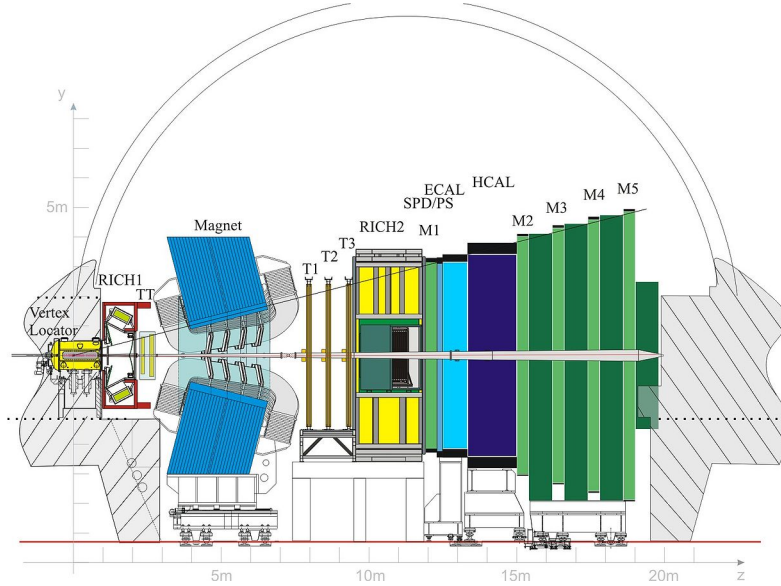


Figure 1.1: The LHCb Detector along the bending plane.

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 vertex³, or a secondary vertex⁴.

¹Radial distance from the beam pipe.

²Azimuthal angle from the beam pipe.

³The position at which the protons collided.

⁴The decay point of a short lived particle. i.e. B Meson.

The RICH detector, comprised of two subdetectors 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,

$$E^2 = M^2c^4 + p^2c^2. \quad (1.2)$$

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*' [?] 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 advance 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, the upgraded VELO will sit at 5.1 mm from the beam at the closest pixel [?]. The current VELO achieves 8 mm [?].

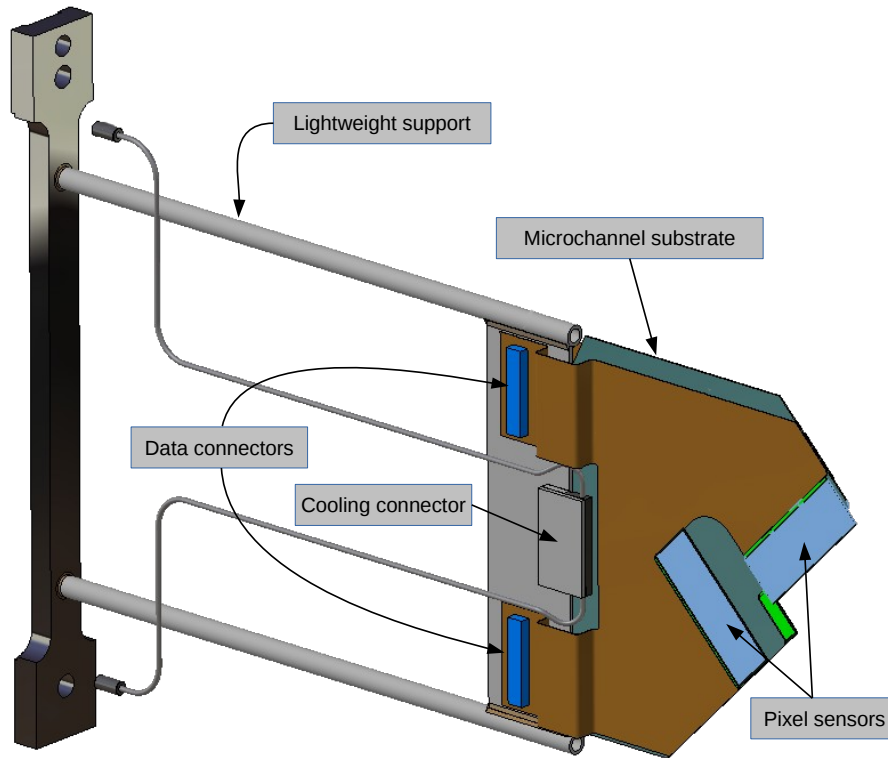


Figure 1.2: The current module design. Two sensors are shown, the remaining two are mounted to the rear face of the module to form two horizontal rows - covering the rightmost area of the module (as viewed in the figure).

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 [?], 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 microchannel substrate. [?]

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 beamline, at the closest point [?]. The foil is made of $250\mu m$ 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 hitmap.

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. Located on the DAQ

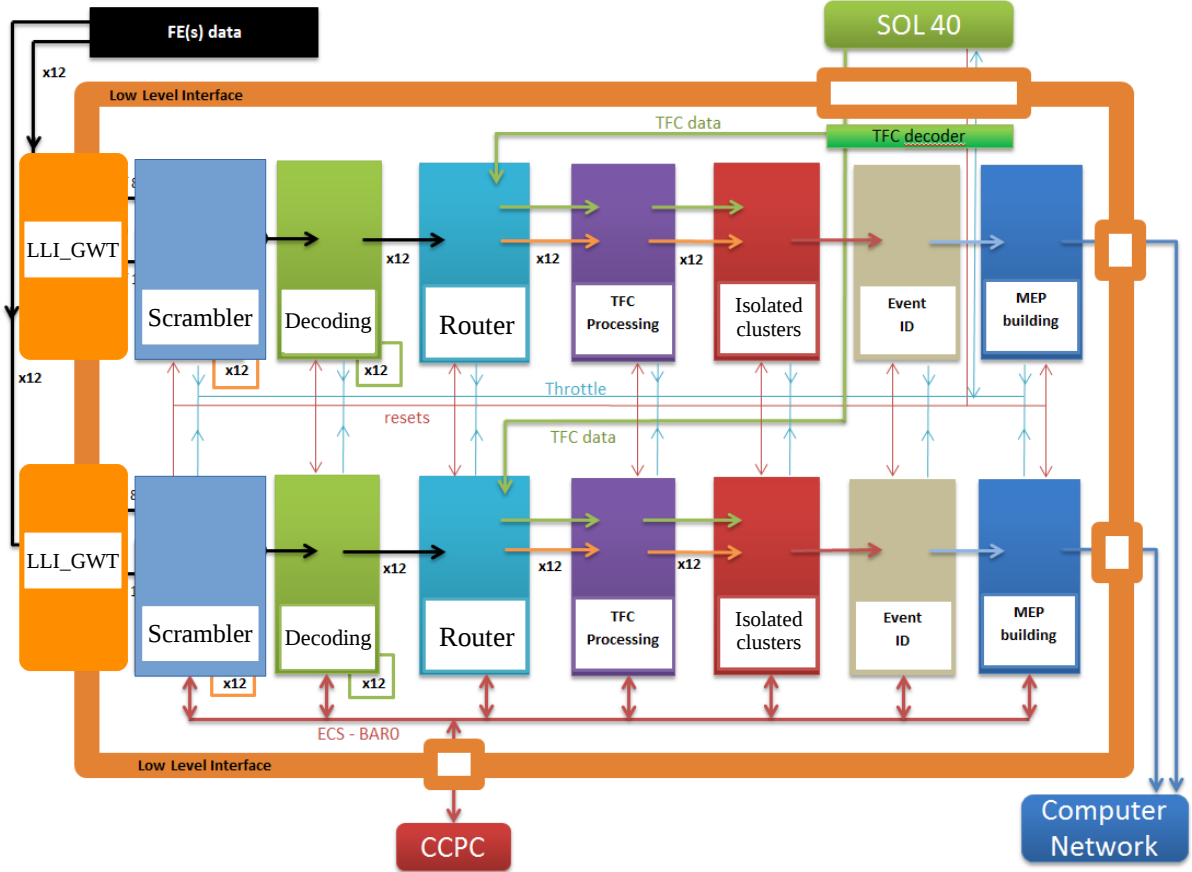


Figure 1.3: A Diagram showing the data flow in the low level interface.

FPGA is the Low Level Interface (LLI). The LLI is responsible for sorting the incoming data into time order and packaging the data in the correct form for the computer systems and to optimise output bandwidth. Other processes included in the LLI are descrambling and event isolation tagging. These processes will be discussed in more detail later in this document.

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 I²C and SPI communication.
- Use bit-changes in the data to continuously synchronise 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 hitmaps 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 con-

sideration, 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 [?] and simulated in Modelsim [?]. 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 desynchronisation event, in which the ‘*key*’ is lost. As expected, the Additive Scrambler was unable to recover any data post desynchronisation, however the intermediate and VeloPix scarmblers 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 ‘*scrambled-ness*’ analogous to entropy. This analogy is not dissimilar to the common interpretation of entropy as a measure of disorder. From Boltzman’s equation for entropy,

$$S \propto \ln(\Omega) \quad (2.1)$$

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 were 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 hitmaps being empty. Scrambled data, via entropic arguments, *should* show zero

bias eitherway. 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

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

$$p_0(t) = p_1(t) = 0.5 \quad : \quad \forall t \in \mathbb{N}, \quad (2.2)$$

At each time iteration,

$$p_{i \rightarrow j}(t) = 0.5 \quad : \quad i, j = [0 \ 1], \quad \forall t \in \mathbb{N}. \quad (2.3)$$

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

$$f(N_\tau) = \frac{n!}{N_\tau!(n - N_\tau)!} p^{N_\tau} (1 - p)^{n - N_\tau} \quad (2.4)$$

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

$$f_\tau(N_\tau) = \frac{n!}{N_\tau!(n - N_\tau)!} (p_\tau)^n \quad (2.5)$$

For $n = 120$, we can calculate,

$$\langle N_\tau \rangle^{Binomial} = \sum_{N_\tau=0}^{n-1} N_\tau f(N_\tau) = n p_\tau = 60 \quad (2.6)$$

$$\sigma_{N_\tau}^{Binomial} = \sqrt{n p_\tau^2} = 5.48 \quad (2.7)$$

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 2.5,

$$\Omega_\tau \sim \frac{n!}{N_\tau!(n - N_\tau)!} \quad (2.8)$$

$$\langle N_\tau \rangle^{Entropic} = MAX[S_\tau] = MAX[\Omega_\tau] \quad (2.9)$$

This can be numerically solved,

$$\langle N_\tau \rangle^{Entropic} = 60 \quad (2.10)$$

While the result of equation 2.10 does not contribute 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,

$$p_n = p_1(1 - p_\tau)^{n-1}, \quad : \quad n \in \mathbb{N}, \quad n > 1 \quad (2.11)$$

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,

$$\frac{N_n}{N_0} = (1 - p_\tau)^n, \quad : \quad n \in \mathbb{N}, \quad n > 1 \quad (2.12)$$

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

$$\begin{aligned} \log\left(\frac{N_n}{N_0}\right) &= n \log(1 - p_\tau), \\ \log(N_n) &= n \log(1 - p_\tau) + \log(N_0). \end{aligned} \quad (2.13)$$

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 $p_\tau = 0.5$,

$$\log(1 - p_\tau) = -0.30. \quad (2.14)$$

Bit Asymmetry

$A_{1,0}$, the asymmetry of ‘1’s and ‘0’s is defined as,

$$A_{1,0} = N_1 - N_0, \quad (2.15)$$

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 [?],

$$A_{1,0}(nt + n \Delta t) = A_{1,0}(nt) + \mathcal{N}(nt). \quad (2.16)$$

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 $A_{1,0}$,

$$\langle A_{1,0}(nt = M n \Delta t) \rangle = \sum_{m=0}^{M-1} \mathcal{N}(m n \Delta t), \quad (2.17)$$

$$\begin{aligned} \langle A_{1,0}(nt = M n \Delta t)^2 \rangle &= \sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \mathcal{N}(m n \Delta t) \mathcal{N}(m' n \Delta t) \delta_{mm'} \\ &= \sum_{m=0}^{M-1} \langle \mathcal{N}(m n \Delta t)^2 \rangle. \end{aligned} \quad (2.18)$$

Clearly, in Equation 2.17, $\langle A_{1,0} \rangle = 0$. In Equation 2.18, we assume the variance is of form $(n \Delta t)^\alpha$ [?]. Then,

$$\langle A_{1,0}(nt = M n \Delta t)^2 \rangle = M(n \Delta t)^\alpha. \quad (2.19)$$

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 2.19,

$$\langle A_{1,0}(nt = M n \Delta t)^2 \rangle = t_f (n \Delta t)^{\alpha-1}. \quad (2.20)$$

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

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

With $\alpha = 1$,

$$\langle A_{1,0}(nt = M n \Delta t)^2 \rangle = M(n \Delta t). \quad (2.21)$$

And thus,

$$\sigma_{A_{1,0}} = \sqrt{\langle A_{1,0}^2 \rangle - \langle A_{1,0} \rangle^2} = \sqrt{\langle A_{1,0}^2 \rangle} = \sqrt{n \Delta t}. \quad (2.22)$$

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 $\langle N_\tau \rangle = 60$ 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.

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

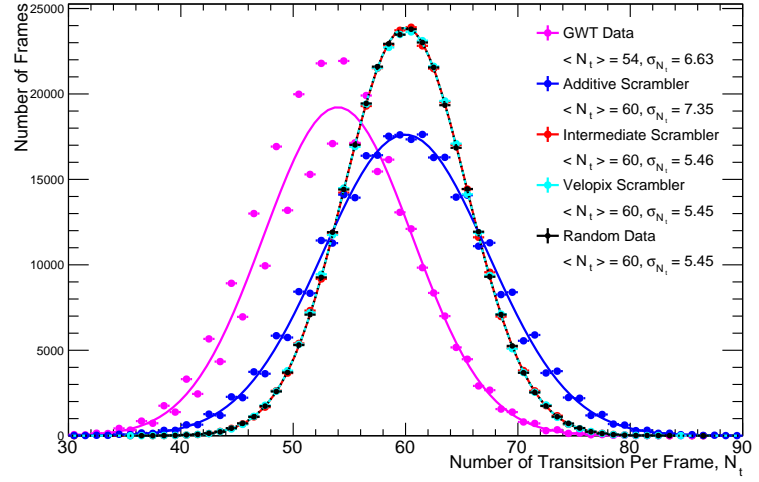
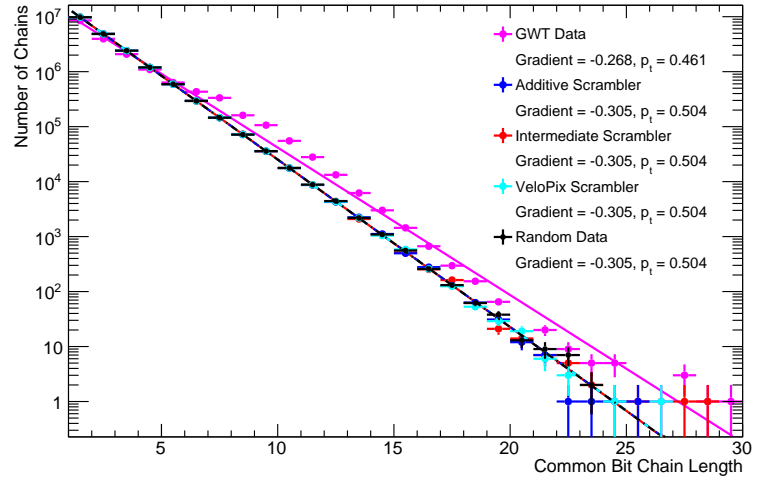


Figure 2.2: Results of the ‘Common Bit Chain Length’ analysis. The results for the Random Data, Additive Scrambler, Intermediate Scrambler and VeloPix Scrambler overlap.



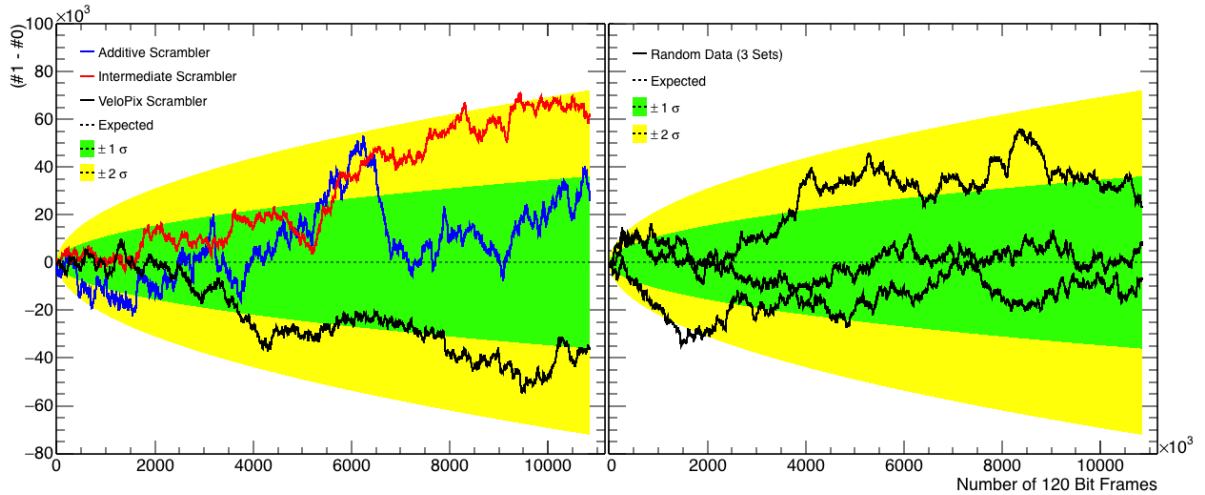


Figure 2.3: The results of the ‘*Bit Asymmetry*’ analysis.

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

	$\langle N_\tau \rangle$	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 algorithm analysis.

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*’ 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 inline 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 through 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 to 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 with 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 Module

The EIF data processing comprises of two main steps.

- Bubble sort the SPP's by row ID.
- Compare SPP row ID 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

As implied by the name, this entity passing data across the EIF system without data processing.

3.2 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).

The first step in EIF is to sort all the SPP's that correspond to the same bunch crossing (Henceforth referred to as a '*data train*') by their row in the ASIC.

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 3.1.

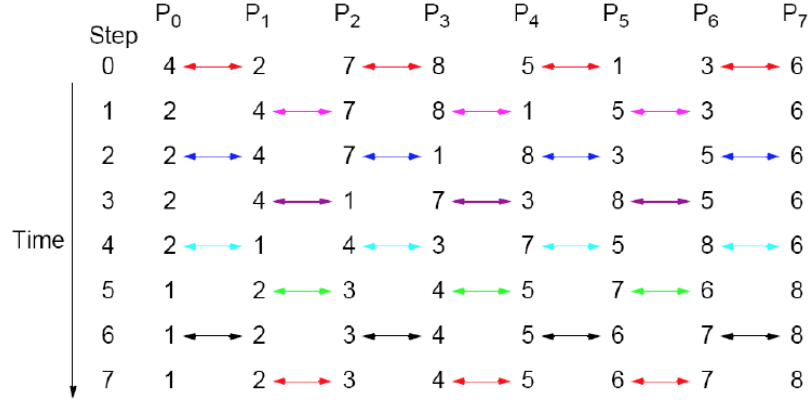


Figure 3.1: A diagram showing Bubble Sorting in an FPGA.

3.3 Isolation Checking

Once the data train is sorted by row, each SPP in the train can be compared against its adjacent SPP's. If the SPP is separated by more than 1 row to both adjacent SPP's, the event is isolated. The SPP is then stored as a 31 bit SPP, with the new bit added as the least significant bit (shifting the original SPP 1 bit in significance), with the new bit signalling 1 for isolated and 0 for non-isolated.

3.4 Data Train Overflow

One limitation of EIF in an FPGA is the limitation on resources. The logic systems are static in design and, as such, there is a natural need for a cap on the size of the data train that the EIF system can accept - specifically for the bubble sorting. Because of this limitation, the EIF system is required to implement an overflow system that will reject data trains above a pre-determined limit, and move them to the next step of the LLI without processing them. This system is also required to bypass data if a data train arrives at the EIF system before the previous train has been processed - preventing pile-up.

In order to investigate the limit needed for the overflow, the distribution of data train sizes was investigated. For each ASIC, a graph similar to those in Figure 3.2 can be created.

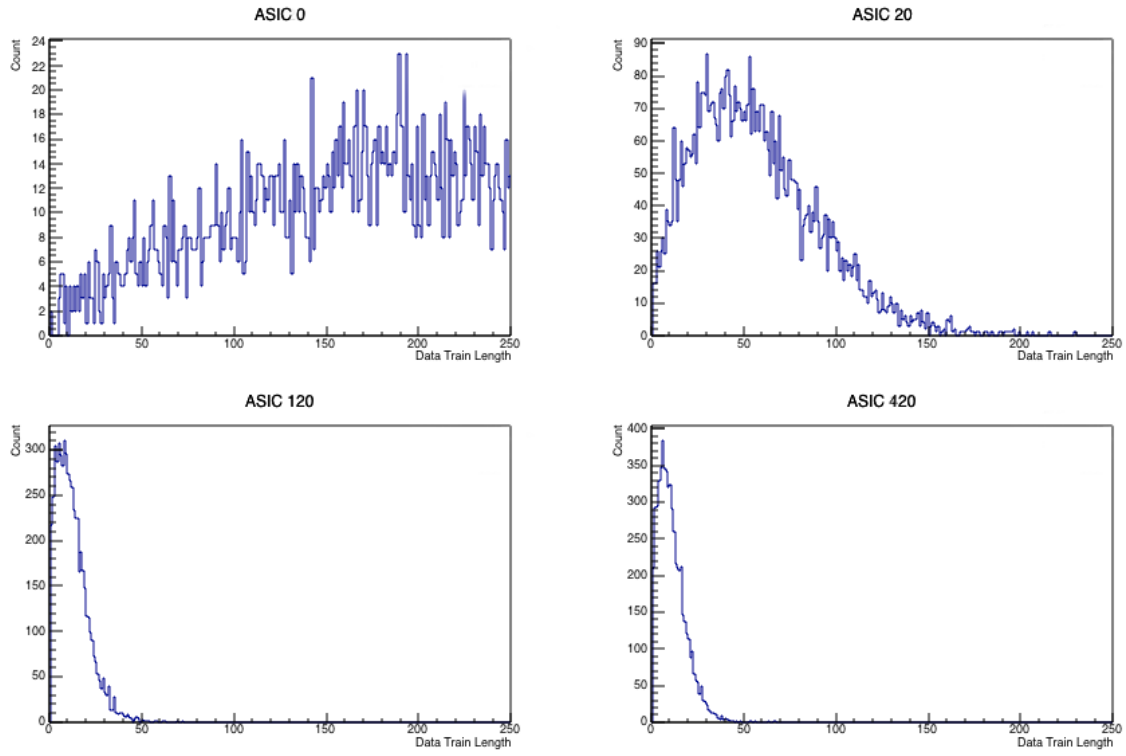


Figure 3.2: The data train length distribution of 4 ASIC chips.

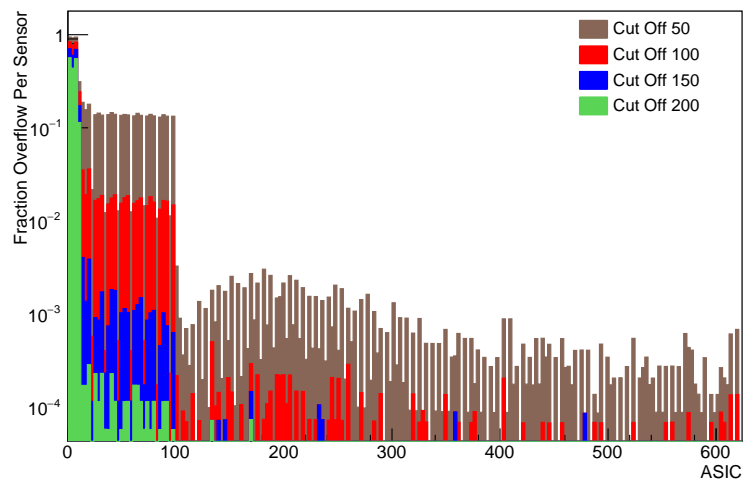


Figure 3.3: fraction of overflow data trains for four overflow limits.

More important, however, is the fraction of data trains over the bypass limit. For four hypothetical limits, the fraction of overflow data trains was calculated from the VELO simulated data, and is shown in Figure 3.3. This analysis, however, raises questions beyond that of the overflow limit. The ASICs below 100 show no similarity to those above 100.

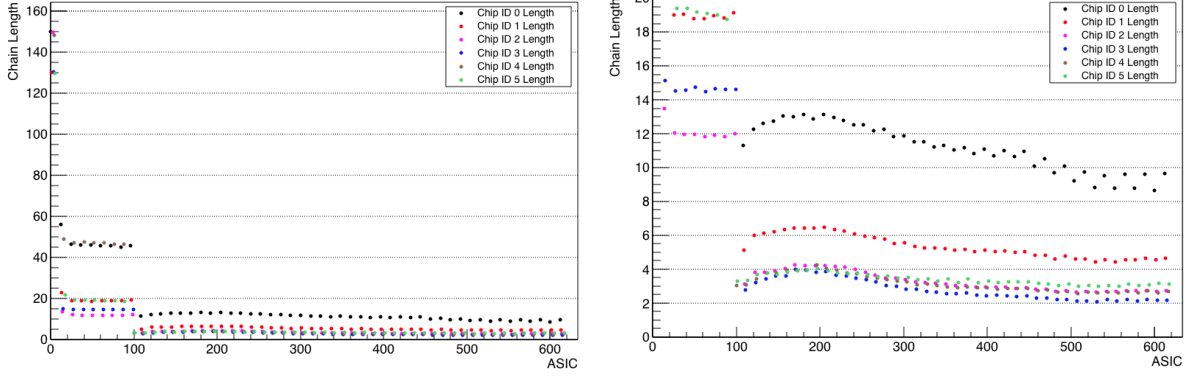


Figure 3.4: The mean data train length for each ASIC, coloured by the chip number.

Further investigations as to the structure of the simulated data is shown in Figure 3.4. Here the data is partitioned by the position of the ASIC chip in the module. From this we learn that large variance in the data (ASIC number > 100) is due to the ASICs position on the module. This result is expected as the ASICs closer to the beam line will detect more particle paths. However, this structure is not consistent across the ASICs pre and post 100. It can be concluded that the simulated data contains a *'bug'*. This *'bug'* is now being reviewed by the creators of the simulation. No further analysis can be continued on setting an overflow limit until this *'bug'* has been properly investigated.

3.5 Current Stage of Development

The EIF system is still currently in active VHDL development. The current developmental code is still in a stand alone format and not integrated with the master LLI code. Currently created and ready for stand alone testing, is a bubble sorting module with data in and out systems. The module consists of a top level control entity and a comparison/swap sorting entity. The control entity forms a feedback loop passing the output of the sorting entity back into its input at each step. At each step, the parity of comparison is changed (i.e. odd-even to even-odd).

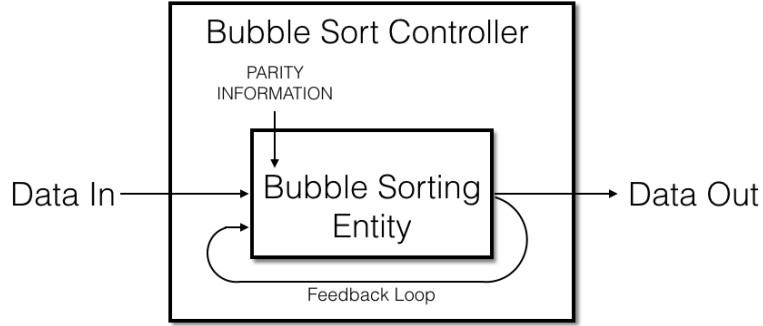


Figure 3.5: Data from for the developmental bubble sorting module.

This process continues until the input and output of the sorting module is identical for two subsequent steps. At this point the data is sorted and passed to the output. The data flow is more simply demonstrated in Figure 3.5.

Once testing of the bubble sorting is complete and the simulated data bug is fixed, the EIF will be expanded to include Isolation Flagging and an overflow, as discussed. Once the stand alone system is complete, it will be intergrated into the LLI master code and modified to comply with the LLI data management systems.

4 Future Development

Once the current areas of FPGA development discussed in this document are complete, focus can be directed on areas of the LLI still in active developemt. One such area is in the processing of incoming data from the optical links into frames in the DAQ, currently being worked on by a colleague in Manchester University. However, due to the nature of the development being spead over many European countries, reliably predicting the next step in development is a not possible.

Once the FPGA development is complete and the VELO upgrade is implemented in 2019, the more accurate VELO data will allow for greater statistical analysis and more significant results. This is of particular interest for rare processes, such as CP violation, which are largely dominated by statistical error with the current LCHb data.

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.