FPGA Development for the LHCb Vertex Locator Upgrade

Nicholas Mead
8064141
School of Physics and Astronomy
University of Manchester

December 19, 2015

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur blandit purus ut lacus aliquam, a sodales ante sodales. Etiam a elit nunc. Mauris ipsum tellus, ullamcorper et arcu at, cursus malesuada elit. In tempus pellentesque nisi, vel egestas enim cursus tempus. Sed velit urna, luctus sed efficitur sed, laoreet vitae magna. Mauris elementum dignissim lacus vitae tempus. Curabitur laoreet molestie dictum. Donec sit amet auctor nisl.

Duis pellentesque euismod pellentesque. Praesent volutpat tincidunt eros, at faucibus tellus eleifend a. Quisque molestie sed ante sit amet sodales. Duis sed justo quam. Curabitur tellus felis, laoreet et bibendum a, posuere eget nisi. Donec suscipit lacinia porttitor. Aenean posuere sem nibh, et iaculis nisl faucibus eu. Donec ac posuere sapien. Aenean suscipit, nisi eget porttitor viverra, dui sapien vulputate lectus, ut dapibus purus orci nec arcu. Etiam placerat sapien non massa fringilla, et malesuada nibh hendrerit. Vestibulum et porttitor mi. Aliquam turpis velit, rutrum vitae erat at, scelerisque cursus lacus. Praesent libero urna, sodales efficitur eros id, sodales lacinia sem. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Contents

1 Scrambling Algorithms **1**

1.1 The Role of Scrambling Data in the VELO 1

1.2 Scrambler Options 1

1.3 Cross Checks 2

1.4 Algorithm Analysis 2

1.4.1 Messurements of the Algorithms 3

1.4.2 Statistical Predictions 4

1.4.3 Results of Analysis 7

1.5 Conclusion 8

References **9**

1 Scrambling Algorithms

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 linkes (per modual) are used to transfer the data from the front end VELO to the Data Aquizition FPGA (DAQ). When comunicating data digitaly, the transfering modual (TX) and the recieving modual (RX) must have syncrinised clocks. In these case, the (name form dataflow) is the TX, and the DAQ is teh RX. When achieving syncronised close, there are two main approunches:

I. Transmit the TX clock to the RX modual - used in I^2C and SPI communication.

II. Use bit-changes in the data to continuously synchronise the RX clock.

The former of these options, although the more convienient, is not appropriate for the VELO as it is suseptable to unforseens delays that could cause desyncronisation of the clocks to the data. The latter, while more invarient during delays, requires data with a high density of tranitions to reduce the likelyhood of a desyncronisation event. Becuase delays in the data are possible, the latter option has been selected.

1.1 The Role of Scrambling Data in the VELO

For the reasons described in Section 1, it is nessesary to ensure that the data has large density of transitions before being transmitted from the front-end detector to the DAQ modual. However, as the majority of super pixel hitmaps are empty, the data has a bais towards *'0'*s. This reduces the frequancy of transitions in the data - increasing the probability of a desyncronisation event. It is therefor nesseccary to scramble the data prior to transmition and descramble the data in the DAQ FPGA.

Scrambling and later descrambling the data is not a trivial exercise. The scrambleing (TX) modual and descrambling (RX) modual must use a sycronised *'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 when generating this *'key'*:

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

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

1.2 Scrambler Options

Three scrambling algorithums have been concidered:

Additive Scrambler

This algoritum is simple and easy to use, however has the drawback of time dependance. If a desyncronisation event occours, all subsequent data is rendered unrecoverable untill such time as a global reset signal is sent. Further adding to the drawbacks, if a data packet is not sent from the TX during any clock cycle the RX descrambler will still evolve its descramble key - the TX sccrambler, however, will not. This will ofcourse desyncronise the *'keys'*, and as before all subsequent data is lost.

Intermediate Scrambler

Deriving its name from being the second algorithum under concideration, the Intermediate Scrambler is a **multiplicative** algorithm. The *'key'* is generated from the current incoming frame and the previous frame. Therefor, in the event of desyncronisation, only two frames are lost before the *'key'* is automatically recovered. This is a significant improvment over the Additive Scrambler.

VeloPix Scrambler

Named as, at the time of the start of the project *(September 2015)*, this algorithum is the current preffered option by the VeloPix team; this too is a **multiplicative** algorithm. The *'key'* is, again like the Intermediate Scrambler, generaged from the currect and previous data frame. The VeloPix Scrambler differse from the Intermediate scrambler as it aims to more effeciently scramble the data.

1.3 Cross Checks

Ofcourse, the main prioritys when scrambling data is ensuring that the data in recoverable. For all three scramblers, the algorithum was sysnthesised in Quartus[1] and simulated in Modelsim[2]. The aim of sysnthesising and simulating the scramblers in these programs was to ensure that the design was physical in term of on-board logic gates, and to check that the scrambled data was recoverable.

Furthermore, a C++ simulation was created for the three scramblers. This simulated had two purposes: firstly the output of the C++ can checked against the Modelsim simulated cehck consistancy; secondly to simulate the scrambler over a much larger simple of data as Modelsim simulations are less time effecient. In attition to the cross checks, the C++ code allowed for the descrambling to be delayed by several frames. The aim of this was to simulate a desycronisation event. As expected, the additive scarmbler was unable to recover any data, however the intermediate and VeloPix scarmblers both recovered the *'key'* after two frames and start descrambling data.

1.4 Algorithm Analysis

Intuitively, one can assume that fully scrambled data will be indistinguisable from randomly generated data. For this reason, the three algorithm are not only tested against eachother and the pre-scrambled data but also randomly generated binary. The randomly

generated data was created using the Python *'random'* library, selecting a *'0'* or *'1'* with probubility 1/2 each. While the Python *'random'* library is only sudo-random, on the scale of this example (i.e. $>$ 100,000 frames), this is by far sufficient.

More mathematicaly rigorus, however, is to evaluade the system abstractly in the framework of statistical physics. In this abstraction, the ensemble is the 120 bit frame (with the header and parity removed); microstates are the particular form of the frames; and macroscopic quantities can be calculated by averaging a large number of frames (i.e. the desync data). For the analysis outlined in section 1.4.1, predictions will be made using these principles and outlined in section 1.4.2.

In the context of the statisical model, it is reasonable to concider the degree of *'scrambledness'* as entropy. Therefor a scrambled system can be assumed to one of maximum entropy; and from Boltzmans law,

$$S \sim ln(\Omega) \tag{1.1}$$

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

1.4.1 Messurements of the Algorithms

To compare the effecincy of the three algorithums in section 1.2, the algorithums where run over the same unput data and compared for the following measures:

Number of Transitions Per Frame
This meassure 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 hypethetical 20 bit frames,

a) 10101010101111111111
b) 10011001100110011001

both with 10 transitions, are concidered equaly. However, (b) is clearly a more suitable output for data transfer as (a) has a large probability of desyncronisated due to the long chains of *'1'*s. It is therefore also nessecary to evaluate the length of common bit chains within the scrambled data as shorter chains are more suitable for data transfer.

Total Bit Frequancy

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

1.4.2 Statistical Predictions

Number of Transitions Per Frame

Concider a particle in a symmetric, descrete time-dependent, two state system,

$$p_0(t) = p_1(t) = 0.5 \quad : \quad \forall\ t \in \mathbb{N} \tag{1.2}$$

At each time itteration,

$$p_{i\to j}(t) = p_{i\to i}(t) = 0.5 \quad : \quad i,j = [0\ 1], \quad \forall\ t \in \mathbb{N} \tag{1.3}$$

However, as $p_{1\to 0}(t)$ is equal in both probalility and importance to $p_{0\to 1}(t)$, the probability of a bit change shall herefore be refered to as $p_t(t)$.

Over a n step process, analogous to a n bit frame, the probalility distribution of the number of transitions N_t is given by Binomial statistics,

$$f(N_t) = \frac{n!}{N_t!(n-N_t)!}\ p^{N_t}\ (1-p)^{n-N_t} \tag{1.4}$$

Simplified for the special case $p = p_t = 0.5$,

$$f_t(N_t) = \frac{n!}{N_t!(n-N_t)!}\ (p_t)^n \tag{1.5}$$

For $n = 120$, we can calulate,

$$< N_t >^{Binomial} = \sum_{N_t=0}^{n-1} N_t\ f(N_t) = n\ p_t = 60 \tag{1.6}$$

$$\sigma_{N_t}^{Binomial} = \sqrt{n\ p_t^2} = 5.48 \tag{1.7}$$

Furthermore, when concidering the entropic argument in section 1.4 equation 1.1, the number of microstates corespoding to each macrostate N_t can be related to equation 1.5,

$$\Omega_t \sim \frac{n!}{N_t!(n-N_t)!} \tag{1.8}$$

$$< N_t >^{Entropic} = MAX[S_t] = MAX[\Omega_t] \tag{1.9}$$

This can be numerically solved,

$$< N_t >^{Entropic} = 60 \tag{1.10}$$

While the result of equation 1.10 does not contibute anything new, it is important as a *'sanity check'*. Because the system can be described as in section 1.4, it would indicated a problem in the theoretical framework if the result did not match.

Common Bit Chain Length

The probability of a chain of legth n is,

$$p_n = p_1(1-p_t)^n, \quad : \quad n \in \mathbb{N}, \quad n > 1 \tag{1.11}$$

where p_1 is the number of chains of lenght 1. As $p_1 = N_0(1-p_t)$, where N is the total number of chains,

$$\frac{N_n}{N_0} = (1-p_t)^n. \quad : \quad n \in \mathbb{N}, \quad n > 1 \tag{1.12}$$

Takeing the log of both sides,

$$log\left(\frac{N_n}{N_0}\right) = n\ log(1-p_t). \tag{1.13}$$

Therefor, for a graph of $log(count)$ against n for a large sample of data, the gradient would be $log(1-p_t)$. In this case, as $p_t = 0.5$,

$$log(1-p_t) = -0.30\ . \tag{1.14}$$

Total Bit Frequancy

A, the assymetry of *'1'*s and *'0'*s is defined as,

$$A = N_1 - N_0, \tag{1.15}$$

where N_1 and N_0 are the number of *'1'*s and *'0'*s respectively. We can concider the evolution of A with frame t of size n as a stockastic itterative map with zero deterministic growth[3],

$$A(nt + \Delta t) = A(nt) + \mathcal{N}(nt) \tag{1.16}$$

Where $\Delta t = \Delta frame \times n$ and $\mathcal{N}$ is an independant random variable picked from a gausian distribution. While $A(t) \in \mathbb{Z}$, in the limit of large nt we can approximate that A is continious.

If we concider the moments of A,

$$< A(t = M\ \Delta t) > = \sum_{m=0}^{M-1} \mathcal{N}(m\ n\ \Delta t), \tag{1.17}$$

$$\begin{aligned} < A(t = M\ \Delta t)^2 > &= \sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \mathcal{N}(m\ n\ \Delta t)\mathcal{N}(m'\ n\ \Delta t)\ \delta_{mm'} \\ &= \sum_{m=0}^{M-1} < \mathcal{N}(m\ n\ \Delta t)^2 > . \end{aligned} \tag{1.18}$$

Clearly, in Equation 1.17, $< A >= 0$. In Equation 1.18, we assume the variance is of form $(\Delta t)^\alpha$ [3]. Then,

$$< A(t = M\ \Delta t)^2 > = M(\Delta t)^\alpha. \tag{1.19}$$

Running the analysis over the frames $t = 0$ to t_f, the number of bits sampled is $M = t_f/\Delta t$. Substituting this into Equation 1.19,

$$< A(t = M\ \Delta t)^2 > = t_f\ (\Delta t)^{\alpha-1}. \tag{1.20}$$

Concidering the three cases of α:

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

With $\alpha = 1$,

$$< A(t = M\ \Delta t)^2 >= M(\Delta t). \tag{1.21}$$

And thus,

$$\sigma_A = \sqrt{< A^2 > - < A >^2} = \sqrt{< A^2 >} = \sqrt{\Delta t}. \tag{1.22}$$

1.4.3 Results of Analysis

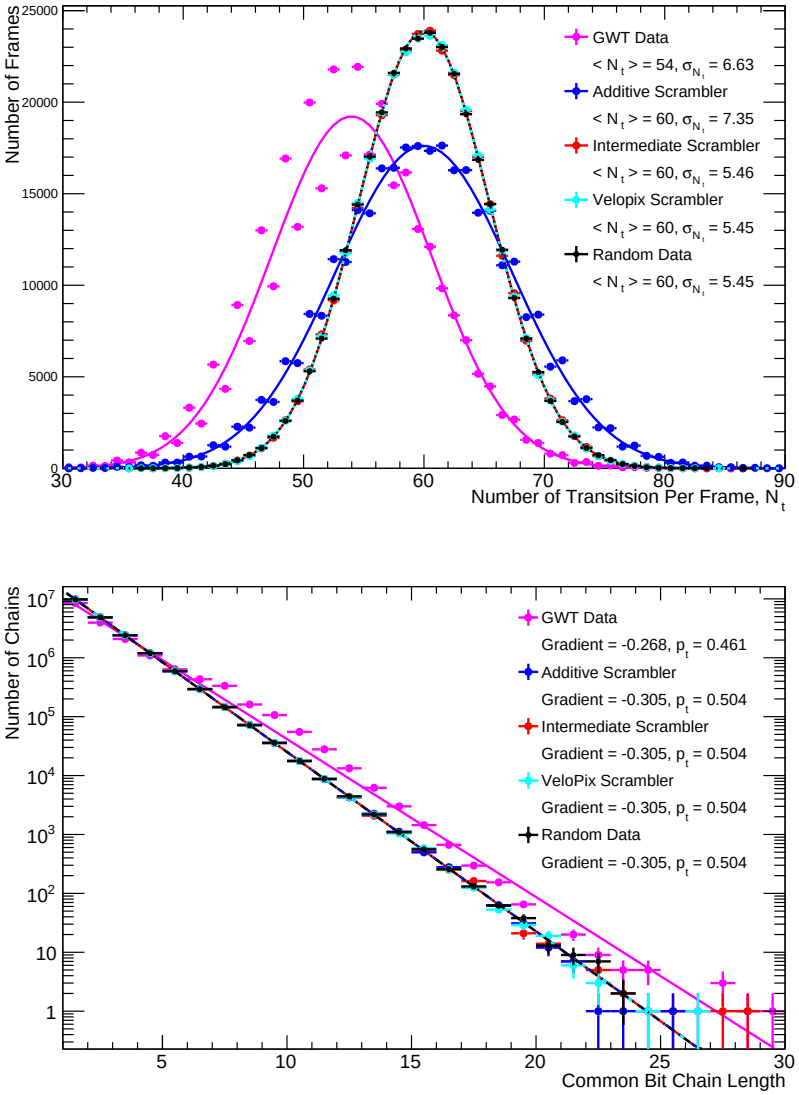

Figure 1.1: Results of the *'Number of Transitions Per Frame'* analysis (Top) and the *'Common Bit Chain Length'* analysis (Bottom). The results for the Random Data, Intermediate Scrambler and VeloPix Scrambler overlap for the *'Number of Transitions Per Frame'* analysis. The results for the Random Data, Additive Scrambler, Intermediate Scrambler and VeloPix Scrambler approximatly overlap for the *'Common Bit Chain Length'* analysis.

The results from the *'Number of Transitions Per Frame'* analysis, shown in Figure 1.1, show a strong corelation between the Intermediate and VeloPix Scramblers with the randomly generated data. These results are withing 1% agreement with the theoretical predictions for $< N_t >= 60$ and $\sigma_{N_t} = 5.48$, made in Section 1.4.2. The remarkable consistancy between the theoretical predictions and the randomly gernerated data provides confidance in both the theory, and the scrambled nature of the Intermediate and VeloPix scrambler outputs.

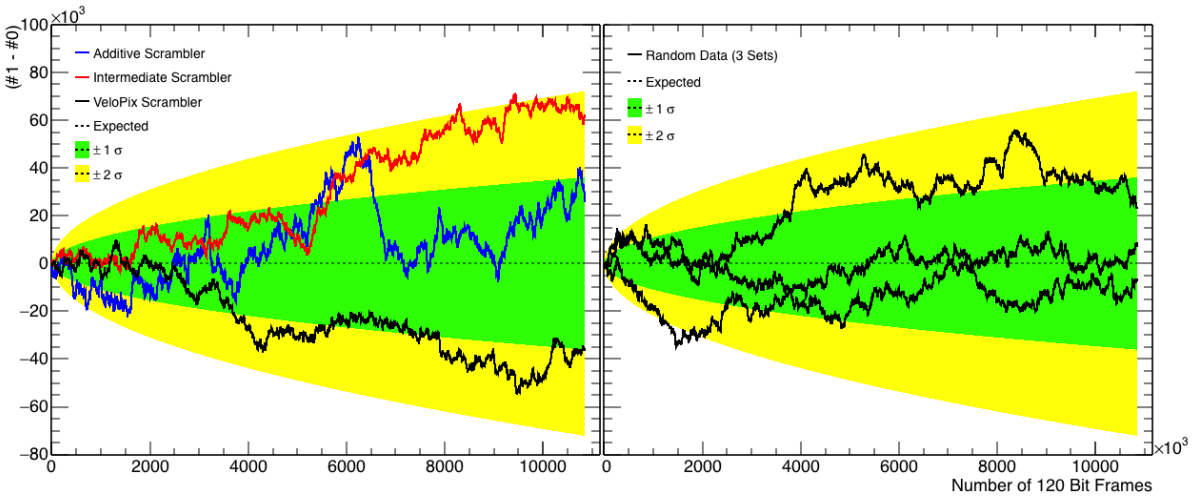

Figure 1.2: The results of the *'Total Bit Frequancy'* analysis.

1.5 Conclusion

References

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

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

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