Design, Implementation and Commissioning of the Readout and Control Systems for the Mu3e Experiment

Dissertation
zur Erlangung des Grades
„Doktor der Naturwissenschaften“
am Fachbereich Physik, Mathematik und Informatik
der Johannes Gutenberg-Universität
in Mainz

Martin Müller
geboren in Boppard
Mainz, den 23. September 2024

1. Gutachter: Prof. Dr. Niklaus Berger
2. Gutachter: Prof. Dr. Sebastian Böser

Abstract

The Mu3e project is searching for the charged lepton flavour violating muon decay $\mu^+ \rightarrow \ e^+e^-e^+$. According to the standard model, this decay is suppressed to an unobservable level and detecting it would be a clear sign of new physics. In the absence of a detection, a more precise experimental limit of the branching ratio will impose tighter constraints on beyond standard model theories.

Mu3e phase I is pursuing a challenging branching ratio sensitivity goal of $2 \cdot 10^{-15}$. In order to reach this sensitivity, 10^8 muon decays per second will be observed by a barrel-shaped detector consisting of 2844 monolithic active pixel sensors and 8896 scintillator readout channels.

This thesis has contributed to the data acquisition (DAQ) system of the detector. The DAQ consists of multiple layers and is expected to face a data rate of 100 Gbit/s. It will process this data using a triggerless fast network of FPGAs and GPUs for online track reconstruction. This work has focussed on the lower layers of this system and discusses the development of FPGA hardware structures for readout, control and synchronisation of the Mu3e detector.

The functionality of the developed system has been demonstrated in two test runs and insights from these tests have been used to implement DAQ design improvements.

Zusammenfassung

Das Mu3e-Projekt zielt darauf ab, das experimentelle Limit für das Verzweigungsverhältnis des Leptonen-Flavour-verletzenden Zerfalls $\mu^+ \to e^+e^-e^+$ zu verbessern. Nach dem Standardmodell wird dieser Zerfall auf ein nicht nachweisbares Niveau unterdrückt, und sein Nachweis wäre ein eindeutiges Zeichen für neue Physik. Ohne einen Nachweis würden genauere Messungen des Verzweigungsverhältnisses Theorien jenseits des Standardmodells strengere Einschränkungen auferlegen.

Phase I des Mu3e-Projekts verfolgt ein Sensitivitätsziel für das Verzweigungsverhältnis von $2 \cdot 10^{-15}$. Um diese Sensitivität zu erreichen, werden pro Sekunde 10^8 Myon-Zerfälle von einem zylinderförmigen Detektor beobachtet, der aus 2844 monolithischen aktiven Pixeldetektoren und 8896 Szintillator-Auslesekanälen besteht.

Diese Arbeit hat zur Entwicklung des Datenerfassungssystems des Detektors beigetragen. Dieses besteht aus mehreren Lagen und wird voraussichtlich mit einer Datenrate von 100 Gbit/s konfrontiert. Die Daten werden mittels eines triggerlosen, schnellen Netzwerks von FPGAs und GPUs verarbeitet. Diese Arbeit konzentrierte sich auf die unteren Lagen dieses Systems und behandelt die Entwicklung von FPGA-Hardwarestrukturen zur Auslese, Steuerung und Synchronisation des Mu3e-Detektors.

Die Funktionalität des entwickelten Systems wurde in zwei Testläufen demonstriert, und die Erkenntnisse aus diesen Tests wurden genutzt um Designverbesserungen umzusetzen.

Contents

1

Introduction

The Standard Model of particle physics (SM) has predicted and explained many observations made by physicists over the last decades. Its success in doing so has pushed much of fundamental physics research towards the hunt for instances where the SM falls short in explaining a measurement. Phenomena such as dark matter, gravity and matter-antimatter asymmetry demonstrate the incompleteness of the SM and drive physics research to improve our understanding and description of nature.

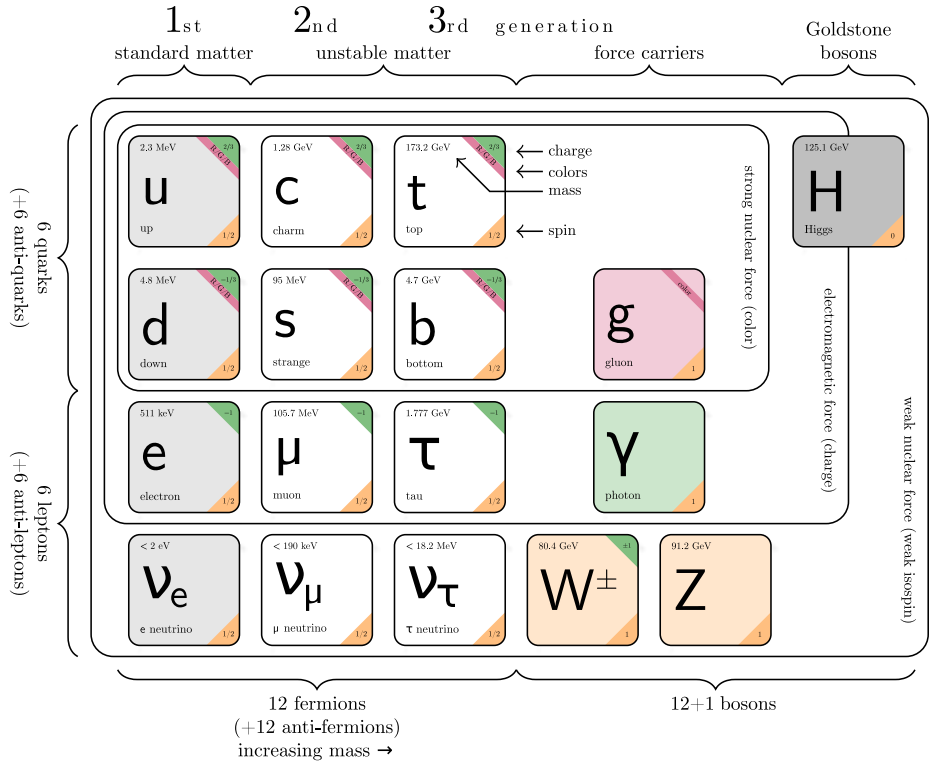


Figure 1.: Particles in the Standard Model. [1] adapted by [2].

The fundamental particles in the Standard Model are categorised into bosons and fermions. The fermions are further divided into quarks and leptons and come in three generations. The interactions between them are described with three fundamental forces, which are caused by the exchange of force carrier particles – the gauge bosons. The strong force bonds quarks together by gluon exchange and forms hadrons such as the proton or neutron. The photon mediates the electromagnetic force, which attracts

or repels charged particles from each other. The third fundamental force is carried by the W- and Z-boson and is called the weak force. The charged leptons (electron, muon and tau) interact via the electromagnetic and weak force, while their neutral counterparts (electron-, muon- and tau-neutrino) can only interact via the weak force. The last particle is the Higgs-boson, which was predicted by the SM and discovered in 2012 by the ATLAS [3] and CMS [4] collaborations.

1.1. Charged Lepton Flavour Violation

The three lepton generations come with the lepton flavour quatum numbers L_e, L_μ and L_τ. In the Standard Model with massless neutrinos, these numbers are conserved and the production of an electron, muon or tau always requires the creation of the antiparticle or the corresponding neutrino of the correct flavour.

Hints for an occasion where the SM disagrees with a measurement were observed by the g-2 experiment [5] in the anomalous magnetic moment of the muon. In addition to the g-2 tension, the observation of neutrino oscillations has recently further motivated the search for new physics in the lepton sector.

Neutrino oscillations have proven that neutrinos are not massless and that lepton flavour is not a conserved quantity. However, a violation of lepton flavour for charged leptons still remains to be observed. An obvious candidate to study charged lepton flavour violation (CLFV) is the muon since it is unstable and has a lifetime which is long enough to perform measurements. Almost 100 % of muons decay to an electron and a neutrino-antineutrino pair with a lifetime of $2.2 \cdot 10^{-6}$ s. A summary of the decay channels is shown in table 1.

Decay	Branching ratio	CLFV Experiment
$\mu^+ \to e^+ \nu_e \, \bar{\nu}_\mu$	$\approx 100\%$	
$\mu^+ \to e^+ \nu_e \, \bar{\nu}_\mu \, \gamma$	$(6.0 \pm 0.5) \cdot 10^{-8}$ [6]	
$\mu^+ \to e^+ \nu_e \, \bar{\nu}_\mu \, e^- e^+$	$(3.4 \pm 0.4) \cdot 10^{-5}$ [6]	
$\mu \to e\gamma$	$< 3.1 \cdot 10^{-13}$, 90% CL [7]	MEG II
$\mu \to eee$	$< 1.0 \cdot 10^{-12}$, 90% CL [8]	SINDRUM
$\mu N \to eN$	$< 7.0 \cdot 10^{-13}$, 90% CL [9]	SINDRUM II

Table 1.: Measured branching ratios or limits of muon decay channels.

The experiments MEG and SINDRUM have provided upper limits for the three lepton flavour-violating decays in table 1. The observation of neutrino oscillations in the Super-Kamiokande and other experiments [10] has, in principle, allowed these decays by extending the SM with neutrino mixing since it leads to the possibility of charged lepton flavour violations via processes as shown in figure 2. However, the branching ratios for these types of processes are estimated at $< 10^{-50}$ [11], which makes them experimentally unobservable in the Standard Model.

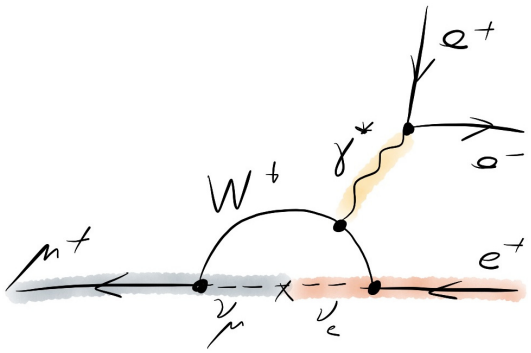


Figure 2.: SM diagram of $\mu \rightarrow 3e$ via neutrino mixing.

There are, however, beyond standard model (BSM) theories that predict an enhancement of charged lepton flavour violating decay channels, which makes an experimental search for them valuable in order to set exclusion limits or to launch further research in this field in case of an actual detection.

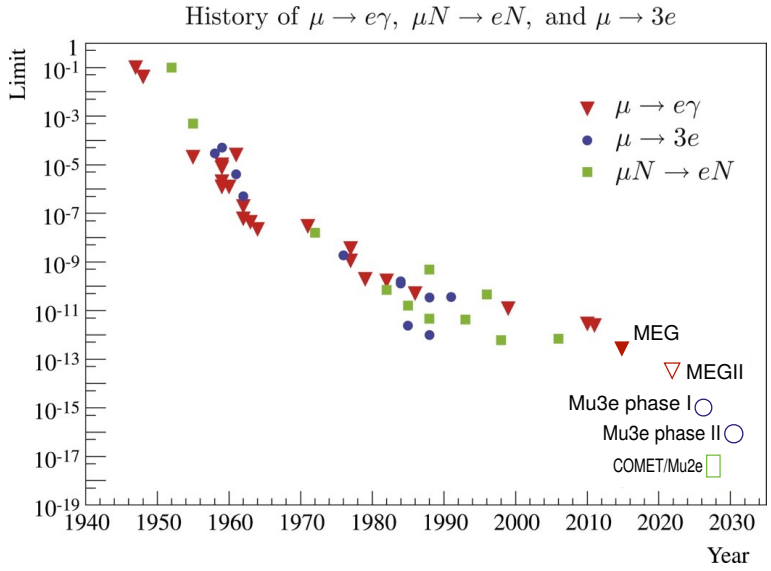


Figure 3.: History of charged lepton flavour violation searches. Adapted from [12].

Figure 3 summarises searches and planned searches for lepton flavour violating muon decays and the upper limits obtained by or predicted for these experiments. The most recent addition is the first data from the MEGII experiment, which improves the limit for $\mu \rightarrow e\gamma$ to $< 3.1 \cdot 10^{-13}$ at a 90% confidence level and is expected to reach a sensitivity of $< 6.0 \cdot 10^{-14}$ once the full statistics has been analysed [7].

It is important to note that the motivation for the experimental efforts in these three decay channels cannot be summarised into a single general search for charged lepton flavour violation. New physics models have not been able to predict absolute

rates for individual decay channels. However, several models can estimate a relative rate [13]. Examples can be found in [13, p. 38]. Values such as

$$\frac{BR(\mu \to eee)}{BR(\mu \to e\gamma)} \tag{1}$$

receive model-dependent predictions. Therefore, improvements in all CLFV channels are necessary to fully utilise the results of each individual experiment. Should one of the channels shown in figure 3 observe a non-zero branching ratio, it will be possible to make model-dependent predictions for the others. A detection of $\mu \to e\gamma$, for example, is only able to rule out some of these theories if the currently reached sensitivity on $\mu \to eee$ or $\mu N \to eN$ allows it. Limiting the searches to only one channel is, for some models, equivalent to orders of magnitude in required sensitivity increase before a detection would be observed. It is technically less challenging and probably also cheaper to improve the measurements for all three channels than to improve the sensitivity of one of them by another three orders of magnitude. It is, therefore, unreasonable to focus on just one decay channel.

1.2. The Mu3e Experiment

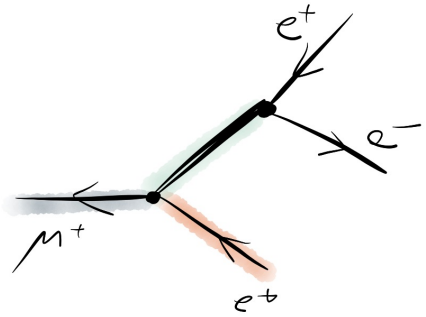


Figure 4.: BSM diagram of a Mu3e decay at tree level.

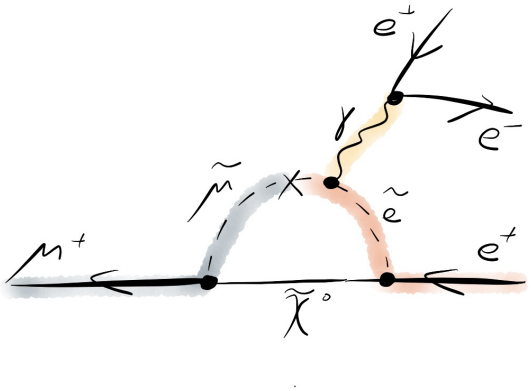


Figure 5.: BSM diagram of a Mu3e decay in a loop containing SUSY particles.

The Mu3e experiment intends to either detect the $\mu^+ \to e^+e^-e^+$ decay or to improve the upper limit for the branching ratio. Currently, the best limit for this decay was measured by the SINDRUM collaboration in 1988 at a value of $< 1 \cdot 10^{-12}$. Mu3e is pursuing a sensitivity goal of $2 \cdot 10^{-15}$ in the first phase of the experiment and is planning for a second phase with an improvement towards 10^{-16}.

Due to the unobservable small size of the branching ratios of decays with a neutrino mixing and the non-existence of other standard model contributions, the decay

channel $\mu^+ \to e^+e^-e^+$ represents a very clean environment to search for physics processes beyond the SM. Hence, the precision of any experiment searching for it will only be limited by the detection efficiency for this decay, the false-positive rate from background processes and the amount of muon decays observed.

The high number of required muons will be produced by the high-intensity proton accelerator (HIPA) at the Paul Scherrer Institute (PSI). The protons are directed to a target where they produce positive pions. These pions decay at rest on the surface of the target and produce muons via

$$\pi^+ \to \mu^+ \nu_\mu \tag{2}$$

Due to momentum and energy conservation, these muons have a kinetic energy of 29.79 MeV:

$$E_\mu = m_\pi c^2 - p_\nu c = \sqrt{(m_\mu c^2)^2 + (p_\mu c)^2}, \quad p_\nu = p_\mu \tag{3}$$

$$\to p_\mu = \frac{(m_\pi^2 - m_\mu^2)c}{2m_\pi} = 29.79 \frac{\text{MeV}}{c} \tag{4}$$

A part of this energy is lost due to interactions in the target, which causes the energy distribution of the muons that leave the target to peak at around 28 MeV [14]. They are then collected by a secondary beamline and guided to the Mu3e experiment, where they are stopped in a target and decay at rest. At the target, a muon decay rate of 10^8 muons/s in phase I and $2 \cdot 10^9$ muons/s in phase II is expected.

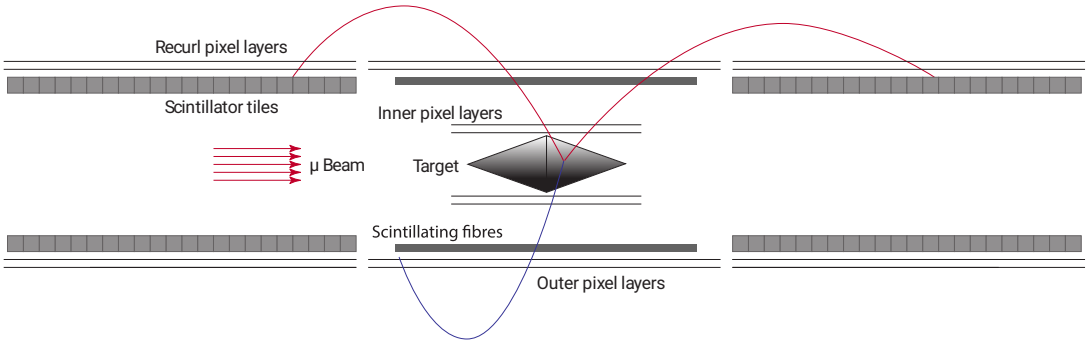


Figure 6.: Schematic of the Mu3e detector.

The target is surrounded by a barrel-shaped detector with multiple layers and is located in a homogenous 1 T magnetic field oriented along the beam axis. Charged decay products enter a helical trajectory and are, in an optimal case, detected by four layers of pixel sensors when they leave the instrumented area and by two additional pixel layers upon reentry. In addition to the pixel sensors, scintillating fibres and scintillating tiles are placed in the regions shown in figure 6 to provide a more precise time measurement. The three segments in this figure will in the following be called the upstream recurl station, the central station and the downstream recurl station. Solid angle coverage of the target is limited by the beam cross-section and the space

needed for other infrastructure.

Since no observable SM process contributes to $\mu^+ \to e^+e^-e^+$, the background consists exclusively of false-positive detections. These can result from internal conversion decays $\mu^+ \to\ e^+\ e^-e^+\nu_e\ \bar{\nu}_\mu$ with particularly low neutrino energy or from accidental overlaps of two $\mu^+ \to\ e^+\nu_e\ \bar{\nu}_\mu$ decays with an electron from Bhabha scattering.

The internal conversion background can be suppressed by a track reconstruction, which is precise enough to identify the missing momentum carried away by the neutrinos. This requires a pixel detector with a high granularity and a good knowledge of the magnetic field along the particle trajectories. Additionally, it puts the detector on a material budget since multiple scattering will influence the accuracy of the momentum measurement. This is a critical aspect since the muons decay at rest and their decay products will be limited to a low momentum range.

The accidental overlaps can be suppressed by a precise vertex reconstruction. In contrast to the internal conversion background, the observed trajectories do not originate from the same decay in this case. Resolving the difference of these decays either in location or time will allow a background suppression. However, the probability of accidental overlaps scales with the beam rate. They will therefore become more problematic in phase II, where an increase in muon rate is planned.

In summary, Mu3e needs a thin detector with a good momentum and time resolution and the ability to process high detection rates. $\mu^+ \to e^+e^-e^+$ candidate events will need to be selected based on their origin in the same vertex and the appropiate kinematics where the three detected decay products add up to a muon decaying at rest.

$$\sum_i \vec{p}_i = 0\ , \qquad \sum_i E_i = m_\mu \tag{5}$$

Details on the detector will follow in the next chapter. The collaboration has performed geant4 simulations using a realistic model of this detector, which produces the reconstruction results shown in figure 7 and leads to the expected phase I sensitivity of $2 \cdot 10^{-15}$.

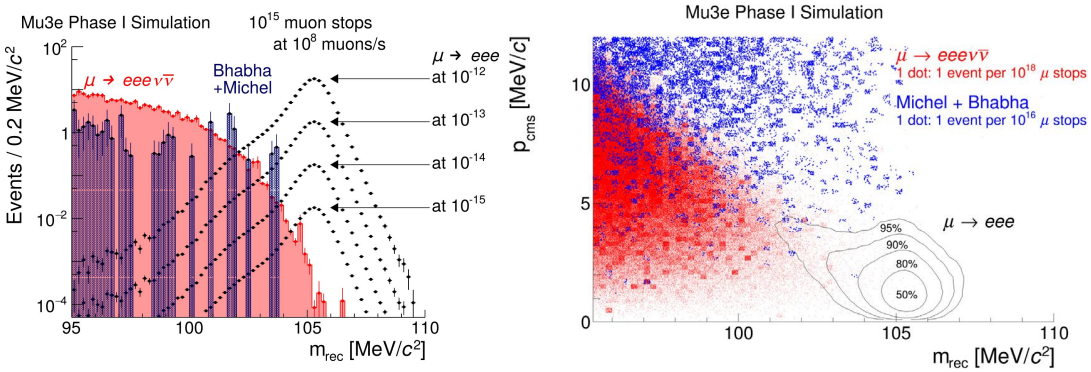


Figure 7.: Reconstructed invariant mass for simulated signal and background events in Mu3e phase I. Taken from [15].

2

The Mu3e Detector

This chapter will briefly summarise the design of the Mu3e detector and introduce some of the components involved in the Mu3e experiment. Especially the MuPix and MuTrig chips discussed in sections 2.2.1 and 2.3.1 will be relevant for the next chapters since they are the entry point into the Mu3e data acquisition and the following work in this thesis.

2.1. Beam, Target and Magnet

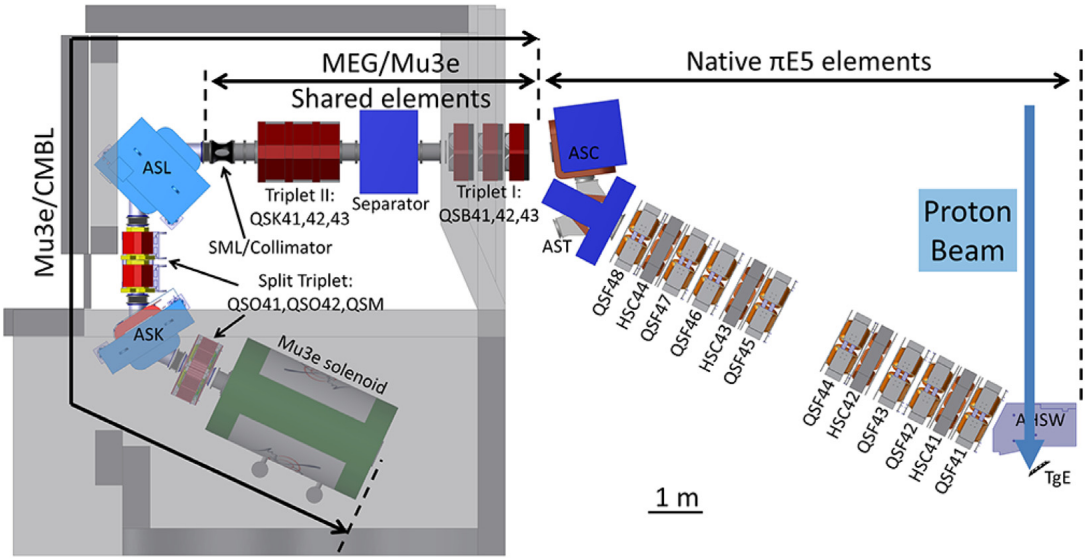


Figure 8.: CAD model of the Mu3e beamline at PSI [15]

The protons from HIPA are directed to a carbon target (TgE in figure 8), where they produce positive pions. These pions decay at rest on the surface of the target and produce muons, which are collected by a set of focusing magnets and guided through a secondary beamline to the Mu3e magnet. At the end of this arrangement, a rate of 10^8 muons/s is stopped on a hollow double-cone mylar target with a diameter of 3.8 cm and a length of 10 cm. The target is mounted in a detector cage placed in the centre of a 31-ton superconducting solenoid, which provides a homogeneous 1 T magnetic field in a 1 m diameter bore with a length of 2.7 m [15].

Similar to the target, a part of the beampipe and detector components are mounted on a moveable cage, which can be inserted or extracted from the Mu3e Magnet.

2.2. Tracking Detector

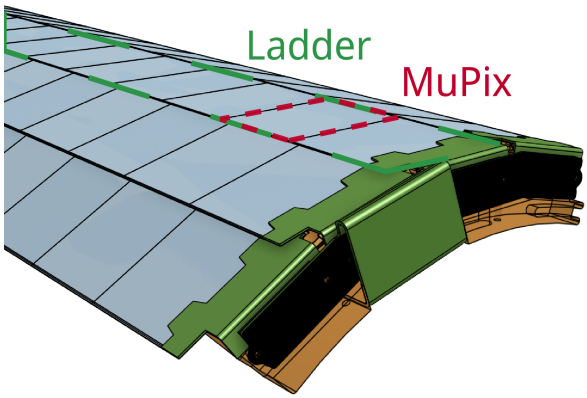


Figure 9.: CAD model of a outer pixel module. Taken from [15].

The six tracking layers are built from MuPix pixel sensors developed within the Mu3e collaboration. To minimise the used material, they are glued and bonded on a polyimide foil, which includes aluminium traces for data readout, control lines and power[1]. One of these foils with an array of MuPix sensors glued onto them is called a Mupix ladder. Multiple ladders are used to build a layer of the tracking detector. The sides of the ladders overlap with the neighbouring ladders to achieve full coverage without gaps. This is shown in figure 9. The ladders in the outer tracking layers in all three detector stations will consist of 17-18 Mupix sensors. Ladders in the inner two layers of the central station are shorter and contain only six sensors.

The sensors are cooled by a 50 g/s [16] helium flow between and around the tracking layers. Using air or other gases for cooling would introduce too much material into the trajectory of the decay products and would further limit the momentum resolution due to scattering effects.

Power, readout and control lines are provided from the ends of each ladder. One half of the chips are serviced from the left and the other half from the right end. This is necessary since the space on the foils does not allow the routing of all connections to one side. However, it also comes with the consequence that a large fraction of the sensor connections must be made at the interconnects between the central station and the two recurl stations. This area becomes quite challenging in terms of space since the available volume also needs to be shared with the helium distribution for the inner tracking layers and following components of the timing detectors.

[1] These will also be called High-Density-Interconnect (HDI) flexprints in the following parts of the thesis

2.2.1. The Mupix Sensor

A pixel sensor consists of junctions of p-doped and n-doped semiconductor material. Conduction electrons from the n-doped material will combine with holes in the p-doped material and vice versa. The region between the n-doped and p-doped material is then free of charge carriers and an electric field from the p-doped to the n-doped material forms.

Any particles interacting in this depletion zone will form electron-hole pairs and lead to a measurable current through the p-n-junction, which the readout electronics of the pixel detector can measure. A further increase in the size of the depletion zone can be gained if a high reverse voltage is applied to the p-n junction. This increases the sensitivity due to the larger active area and also the speed and time resolution because of a faster charge collection.

For the Mu3e detector these aspects had to be implemented on a tight material budget. Therefore, the sensor needs to be thin. Thin pixel sensors existed previously and achieved a low material budget by including the readout electronics into the active detection volume. The Mu3e collaboration has developed a new sensor which combines this with the fast charge collection of sensors with a reverse bias voltage. The resulting High-Voltage Monolithic Active Pixel Sensor (HV-MAPS) is the MuPix [17], which is used to build the Mu3e tracking detector.

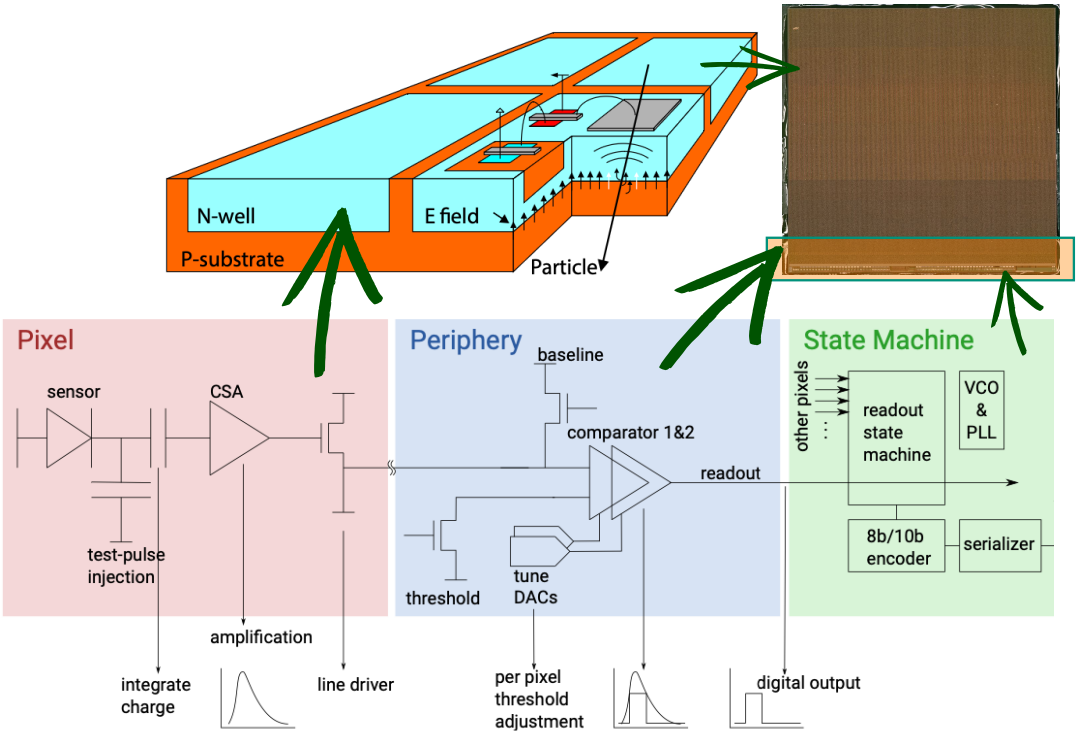


Figure 10.: Sketch of the electronics in a MuPix sensor. Adapted from [17].

A MuPix sensor has a size of 2x2 cm and consists of 256x250 individual pixels. It can be thinned down to a thickness of 50 μm and allows the application of a reverse bias voltage to fully deplete the sensor. It includes an amplification stage and line driver inside of each pixel, which is used to drive signals towards a periphery (see figure 10). During ladder construction, the chip's periphery is oriented in a way that overlaps with the active area of a sensor on the neighbouring ladder.

In the periphery, the signal is compared against an adjustable threshold voltage. Pulses which exceed the threshold are recognised as particle detections and further processed in the digital section of the sensor.

A second adjustable threshold is used to gather information about the shape of the pulse. Once the pulse drops below this second threshold, another signal is sent towards the digital sensor parts and can be used to calculate a time-over-threshold (ToT) value, which is a measure for the pulse amplitude and the deposited energy.

The baseline for both of these thresholds can be set globally for all 256x250 pixels on the sensor. Pixel-individual adjustments are then possible using a 3-bit value for both thresholds. These values are called tune-DACs[2] and allow a calibration of the threshold voltage for each pixel. This is relevant to have a handle on impurities in the semiconductor and other imperfections in the production process, which might change the response of single pixels relative to the rest of the chip.

In addition to the six bits for threshold tuning, another bit is added, which allows the pixel to be turned off. Therefore, each of the 256x250 pixels in a MuPix has a 7-bit value for calibration. The complete Mu3e tracking detector will consist of 2844 MuPix sensors. The pixel threshold tune calibrations will, therefore, contain about 1.3 Gb of data in addition to settings for the sensor-global threshold and other configuration parameters.

The MuPix development took multiple sensor versions. The current version (MuPix 11) fulfils the requirements for Mu3e in the form of a time resolution of < 20 ns and an efficiency above 99 % [18]. It is currently used to build the tracking detector. Further sensor design details can be found in [19], [20] and [17].

2.3. Timing Detectors

The scintillating tile and scintillating fibre detectors are present in the Mu3e design to further increase the time resolution, which can be used to suppress the background from accidental overlaps discussed in section 1.2.

Both of these sub-detectors consist of a scintillator material, which gets excited into a higher energy level by particle interactions. During the de-excitation, light with visible wavelengths is emitted. The emitted light is then internally reflected and can be detected by silicon photomultipliers attached to the material.

[2] DAC: Digital to Analog Converter

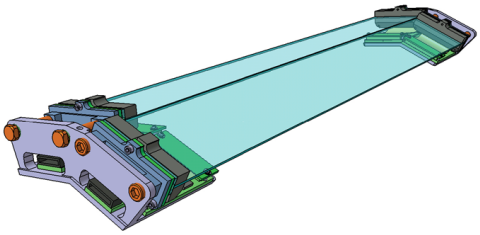


Figure 11.: CAD image of a scintillating fibre module with two ribbons. Taken from [15].

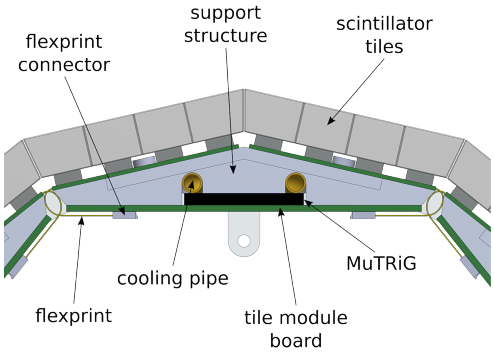


Figure 12.: Schematic cross section of a scintillating tile module. Taken from [15].

The tile detector consists of 5824 small scintillator blocks, which are mounted below the tracking layers in the upstream and downstream recurl stations. Each station contains seven of the tile detector modules shown in figure 12. A module consists of a long PCB equipped with several submodules which contain a 4x4 tile matrix. They form a full barrel layer between the beampipe and the Mupix sensors. Each 5̃x5 mm tile is wrapped into a reflective foil to prevent light from entering into neighbouring tiles.

The scintillating fibre detector is located between the second and third tracking layer in the central detector station. Six of the modules shown in figure 11 form a cylindrical shape around the target. The modules contain two ribbons with three layers of scintillating fibres.

The tile detector is not constrained by a material budget. Once a particle has crossed all six tracking layers and hits the tile detector in the recurl station, the data for the momentum reconstruction has already been recorded and the material budget becomes irrelevant. However, the fibre detector is affected by the material budget constraints. The fibres have to be read out from both ends, which adds to the issue of limited space availability at the station junctions.

Both timing detectors are coupled to silicon photomultipliers, which are read out by MuTrig ASICs. Similar to the MuPix, the MuTrig was also developed by the collaboration for the Mu3e experiment.

2.3.1. Timing Detector Readout and the MuTrig ASIC

The MuTrig is a silicon photomultiplier readout chip capable of acquiring data from 32 channels with an overall event rate of up to 38 MHz. Individual events can be resolved with 50 ps timing bins [15]. For the tile detector, an average time resolution of 46.8 ± 7.6 ps was measured. The fibre detector reaches around 250 ps.

These improvements over the pixel time resolution are used to suppress accidental backgrounds. In the case of the scintillating fibre detector, they also enable particle

identification for decay products with low z-momentum. The pixel tracker is not able to identify particles without momentum along the beam axis since it cannot distinguish between clockwise and counter-clockwise trajectories. When combined with the scintillating fibre detector, identification as a positron or electron is possible via flight time.

2.4. Overview and Infrastructure

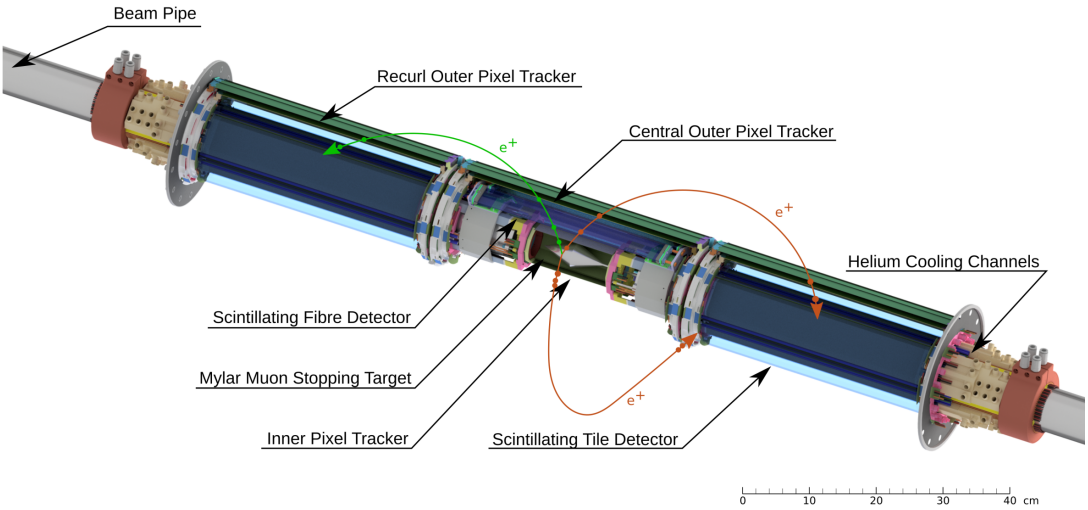


Figure 13.: CAD model of the Mu3e detector. Image taken from [21].

The dimensions of the instrumented area are shown in figure 13. Each station has a length of about 35 cm and a width of 18 cm. The remaining space along the magnet bore diameter is filled with helium but otherwise empty[3] to allow decay products to recurl into the upstream or downstream station.

At both ends of the instrumented area the diameter of the structure is increased to provide space for infrastructure. Before that point, all cooling, power and cabling for the central station and the centre half of both recurl station pixel layers needs to be guided through the gap between the beampipe and the tile detector. This is shown in figure 14. To increase space efficiency, the power is provided through a copper layer glued to the beam pipe.

Large parts of this thesis have contributed to the development of components which are directly attached at the upstream and downstream ends of the detector and are responsible for readout and configuration of the MuPix and MuTrig sensors. The space constraints shown here and the previously discussed constraints on a MuPix ladder have driven the development of a configuration method in [20], which requires a minimal amount of connections. The control and readout components have to

[3] With the exception of a mounting cage.

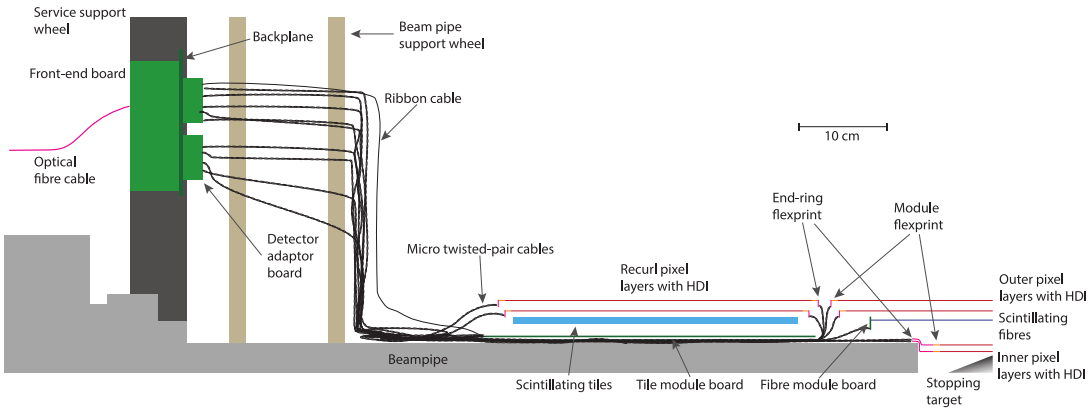


Figure 14.: Sketch of a quarter detector cross-section along the beam axis. Taken from [22].

provide the other end of this method and the infrastructure to process and move data from particle detections from the detector to the outside parts of the readout system.

The expected data rate for the first phase of the experiment was estimated at 100 Gbit/s [23], which requires the use of fast data processing devices. The next chapter will introduce the principles of these devices and the methods used to develop the readout and control system.

3

Digital Electronics

A significant part of this thesis contributes to the design and implementation of the data acquisition and control systems of the Mu3e experiment. In the following chapters, detailed discussions will be provided regarding the challenges, design decisions and functionality of these systems. In this chapter, the foundations on which they are built will be introduced.

3.1. General Concepts

All electronic circuits or data signals can be classified as either analogue, digital or a mixture thereof. In an analogue data signal, the information is carried by a voltage, current or any other physical quantity that can be measured by a receiver. Effects that influence this quantity can negatively affect the accuracy of the information transfer or the functionality of a circuit, for example external fields coupling into a voltage signal.

Digital signals on the other hand use a defined set of discrete levels of the carrying quantity. These levels are defined by the used signal standard in the form of ranges of a quantity, which devices built according to this standard have to identify as the signal level they belong to. Since the information is encoded into the used level rather than the exact measured value, digital signals are less prone to disturbances because only effects that change the signal enough to be identified as a different level actually have an effect.

Most digital circuits and signals use a signal standard with a binary data representation with the logic levels 0/false and 1/true. If the higher carrying quantity corresponds to the true level then the signal is called "active high"; otherwise, it is called "active low". There are some examples of digital signal standards where the data is not encoded in a two-state system, and we will encounter such an example in section 5.13.1.

Processing binary data in an electronic circuit requires a set of components such as flip-flops, logic gates and lookup tables. All of these can be constructed from transistors within different kinds of technologies and devices, which will be discussed later in this chapter. In the following it will be assumed that these basic components are known to the reader. Detailed introductions can be found in [24].

3.1.1. Clocked circuits

A clocked digital circuit consists of a set of registers and logic gates, and any data processing or data movement is done in discrete time steps[1]. To implement these time steps, a clock signal is distributed to all registers, which ensures that the output is synchronised to the clock edge if the input to the register is also synchronised to the clock edge (which is the case if the input is also driven by another register supplied with the same reference clock). A processing step in such a circuit is then the propagation of the output of a set of registers through an arbitrary complicated network of logic gates to the input of another set of registers (Figure 15). The output of the destination register is again the source of a logic path and can connect to further registers or also to the input of the original source register. A clocked digital circuit (for example a CPU) is then the collection of registers and their interconnects through logic gates combined with a few signal lines which leave or enter the circuit and connect it to the outside world.

This kind of system has very desirable properties. All changes take place right after a clock pulse, and the system transits into a new stable state based on the state of the system right before the clock pulse. This principle allows to construct logic sequences and is the foundation that CPUs, other digital ASICs, in general digital electronics and also FPGAs are built upon.

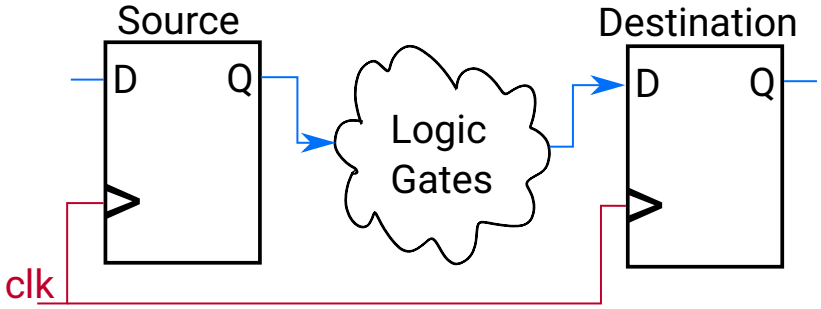


Figure 15.: Schematic depiction of the path that data (blue) has to travel from one register to the next one within a cycle of a clock (red)

In order to ensure the intended functionality of a circuit, all contained propagation paths of a signal from a source register to a destination register need to adhere to certain timing requirements at the input of the destination register with respect to the exact arrival time of the clock edge. These requirements and the consequences of violations thereof are discussed in section 3.3.1. The achievement of meeting these requirements for all paths in a circuit is commonly referred to as *timing closure*.

A collection of registers interconnected in this way and driven from the same clock forms a *clock domain* of a digital circuit. Should a circuit only be driven from a single clock, then it consists of just a single clock domain. If multiple clocks are involved, each one of them drives a separate clock domain. Transitions of signals between

[1] Which are for almost all cases identical to a clock cycle.

clock domains should generally be kept at a minimum since they introduce additional problems. These problems and methods to reduce them will be discussed in sections 3.3.1.4 and 3.7.4. However, transitions between clock domains are a fundamentally unsolvable problem, and these methods only reduce failure probability. Any circuit with more than one driving clock therefore intrinsically has a probability to fail.

An important quantity for a digital circuit is the clock frequency at which it can be operated. In the example of a CPU, this frequency directly corresponds to the number of instructions that this CPU can execute per second and is, therefore, responsible for the speed at which the computer can run calculations. For other kinds of circuits, the clock frequency has a similar meaning, and we will encounter many examples of this during this thesis. The limiting factor for the clock frequency of a given circuit is the longest travel time of a signal between a source and a destination register. This travel time is influenced by a variety of factors.

One of these factors is the amount of logic gates a signal has to travel through. This part can be influenced by the circuit designer and needs to be taken into account when the circuit is supposed to run at a particular frequency, which means that the design of a digital circuit for a higher frequency can be different from the implementation of the same functionality for a lower frequency. These design differences will be discussed with examples in section 3.7.6.

Another essential factor for the maximal clock frequency is the chosen technology to implement the circuit design. As discussed, logic gates and registers can be constructed from transistors. For this reason, a higher transistor density is beneficial for the maximal clock frequency since the travel time of a signal depends on the distance it has to travel.

Environmental conditions, for example temperatures, can also influence the maximal frequency of a design and restrict the operation of a device to specific temperature ranges if the intention is to reach a particular frequency. In particle physics, environmental conditions can also include increased radiation levels, which can lead to events where a bit in a digital design is flipped by the deposited energy of a particle interaction with the electronics ("single-event upset"). Such errors can be reduced by redundancy in the design.

3.2. Hardware Description Languages (HDLs)

Designing a digital circuit is either done by manually drawing connections and placing components or by using a hardware description language (HDL). Hardware description languages such as Verilog or VHDL use text files to describe connections and components and were originally developed to document the behaviour of digital circuits. Now, they are used to develop digital circuits. Different tools are available which perform logic simplification of HDL code and then map the resulting logic onto specific component libraries. Depending on the technology that the design should be implemented in, the components used to do so will be different since components that

are available in one technology might not be available in another one[2]. This process is called synthesis and results in a technology-dependent netlist that describes the components and interconnects of a digital circuit.

After the synthesis, the tools will perform placement and routing to assign the components of the netlist to actual locations in a design while adhering to the requirements mentioned before, which will be discussed in 3.3.1. In addition to the possibility to synthesise HDL into hardware, they are also useful to track design changes in version control systems since the fact that they are text-based allows the use of tools like git.

```
1 process(i_clk)
2 begin
3 if rising_edge(i_clk) then
4         d1 <= i_d;
5 end if;
6 end process;
7 o_q <= i_d and not d1;
```

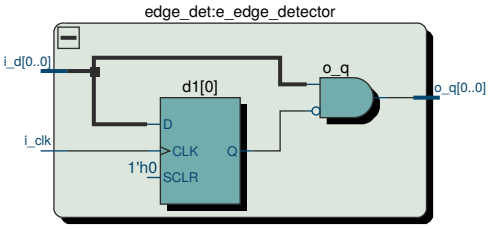


Figure 16.: Example of an HDL language describing an edge detector

3.2.1. Functional Simulation of VHDL

Apart from the steps mentioned above, which are effectively the way to design an actual device, VHDL can also be used for the simulation and verification of a digital circuit. A simulation will sequentially execute the statements in a design on a CPU and emulate the non-sequential nature of a digital circuit by using a number of zero-time time steps ($\delta-$cycles) for each actual clock period in order to resolve signal dependencies within that clock cycle. Since simulations are not actually implementing a circuit but simulating it as a sequence of commands on a CPU, they can take very long to execute for larger designs.

During such a simulation, a design usually must be exposed to external inputs to verify the circuit's output. For most reasonably complicated circuits, these external inputs (or "test-vectors") will only cover a small fraction of all possible scenarios that the circuit might be exposed to - especially since the output of the circuit might not only depend on a single input but also on all other inputs since the start of the simulation as well as the starting conditions. It can, therefore, be difficult to conclude from a simulation that the design will function properly under real conditions since the amount of test cases in the simulation was limited. For sequential circuits treated as a "black-box" it was shown in [25] that verification of a certain behaviour is fundamentally not possible if only the inputs and outputs are observed.

[2] see sections about ASICs (3.5) and FPGAs (3.6).

Furthermore, functional simulations will always only cover the part of the design which is actually simulated. In reality, the proper operation of a circuit and many of the error conditions depend on factors outside of the reach of functional simulations, such as signal integrity, clock jitter, signal inversions or asynchronous clock domains, where logic will typically not have a realistic transition into other domains during the simulation.

Instead of simulating a design and checking the output against the test cases, it is also possible in some cases to formally prove that a design will behave according to a predefined set of conditions. Some methods prove via induction that some properties are valid for all following time steps given a set of assumptions, some perform equivalence checking against reference circuits and there are also other methods [26] which pose interesting questions in computer science. Commercial tools for formal verification are also available [27]. We will not encounter those during this thesis due to the lack of practicability for a project of the size of the Mu3e DAQ.

3.3. Timing

The propagation of signals through interconnects, logic elements and registers is not an instantaneous process. In order to ensure the proper operation of a circuit, it is therefore necessary to account for and arrange the expected delays accordingly. Design tools usually handle this for large and complicated circuits since it needs to be done for all contained components. However, the designer of the circuit has to make sure that all register-to-register paths are short enough in terms of the amount of logic levels and travelled distance to allow the tools to achieve the delay necessary for the desired clock frequency.

3.3.1. Timing requirements

Timing requirements are given relative to the arrival time of the edge of the clock which is used to sample the signal (latch edge) at each destination register of a logic path. The signal that is supposed to be latched into the register with this clock edge generally has to be stable for some time before and some time after the arrival of the edge. These time frames are called setup-time t_{su} and hold-time t_{hold} and depend on the specific device and operating conditions.

3.3.1.1. Setup-Time

The required setup-time t_{su} and a register-internal delay t_{DQ} (time from the input D to the output Q of a register) effectively put an upper limit on the time available for a signal to propagate from the destination to the source register since the overall available time is given by the clock period T:

$$t_{prop} < T - t_{DQ} - t_{su}. \tag{6}$$

Which means that logic inserted between the source and destination register is limited to the amount of logic levels that the signal can travel through within the time t_{prop}. Therefore, fast circuits with a low clock period T have to restrict themselves to simpler logic steps for each clock cycle. In contrast, slower circuits implemented in the same technology can perform more complicated calculations within a single cycle.

This already introduces a very important concept: Assume a complicated calculation that can be computed within one cycle of frequency f_1 on a particular device. The same calculation might not be able to run with a frequency $f_2 > f_1$ on the same device due to the restriction above. However, if the calculation can be separated ("pipelined") into two steps, where each step is executed in its own clock cycle, operation at f_2 might be possible. The pipelined circuit will use more resources since an intermediate result needs to be stored between the two processing steps and the latency of one calculation will increase to $2 \cdot \frac{1}{f_2}$, but the number of calculations per unit of time will increase from f_1 to f_2 if the circuit can be designed in a way that both steps can be used at the same time for two different instances of the same function. In summary, latency and resources were traded against speed. Tradeoffs of this kind are one of the main tools of the circuit designer to stay within the timing requirements of the device, but they can also be used to affect other properties, as will be discussed in section 3.7.6.

Logic placed physically further apart from each other needs longer signal lines between the different components, which will increase the necessary propagation time t_{prop} for signals between them and, therefore, reduce the maximal speed of the circuit. Since the transistor density on a device is always limited, only a specific number of logic components and registers can be present in a given area. If the output of a register connects to several destinations, then the number of destinations (also called Fan-out) will influence the propagation time t_{prop} because not all of them can be placed close to the source register. In addition, the capacitance of the output wire of the source register will be larger, which requires more charge and therefore more time to change the logic state.

The same concept applies to the input of a register or logic component. A large number of signal sources (Fan-in) of a destination register cannot all be placed close to this specific destination. Registers with high Fan-in and/or high Fan-out can, therefore, be a limiting factor for the operating frequency of a circuit and have to be taken into account during the design. This becomes especially problematic when high Fan-in/out registers are supposed to connect to/from registers that also have a high Fan-in/out. When a timing-critical signal path connects registers A and B, it is not enough to look at this specific path; logic before A and after B and the general availability of space in this logic area must also be considered.

Additional constraints to the placement of logic can also arise from the connection to external components from a device. A data connection from some external component will always arrive at a particular physical location on a device, which will constrain the connection logic to this particular region for high frequency circuits.

3.3.1.2. Hold-time

As mentioned above, the hold-time t_{hold} of a register is the time that a signal is required to be stable after the arrival of a clock edge. In contrast to the setup-time, this does require a minimal travel time of any data between two registers since the data is not allowed to change at the destination register for a time t_{hold} after the clock edge. This is different from the concept of maximal propagation-time t_{prop} from section 3.3.1.1 in the sense that for logic which connects multiple sources to the same destination, the minimal source to destination path is relevant, while eq. (6) sets limits on the maximal source to destination path. In between these two boundaries – after the hold time and before the setup time of a register – the data applied to the input can be contaminated with random state changes because the different sources will have different path lengths to the destination and can, therefore, change the output of the combinatorial logic to the destination multiple times before stabilizing on the final result (Figure 17). The sum of the minimal propagation time t_{cont}, which will cause

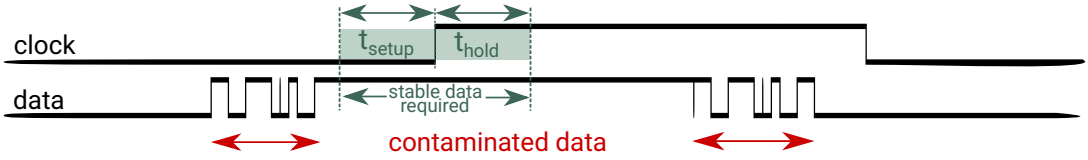


Figure 17.: The input data to a register is not necessarily changing state in a single transition. A stable signal is only needed for the timing requirements around the clock edge.

the input data to a register to be contaminated, and the register internal delay t_{DQ} therefore needs to exceed the hold time t_{hold}.

$$t_{cont} > t_{hold} - t_{DQ}. \tag{7}$$

Until this point, the assumption was made that source and destination registers receive the same clock signal at exactly the same time. This is of course not true, since the clock signal will also experience delays. The difference between the arrival time of a clock at two different locations is called clock skew and can also introduce setup and hold-time violations, since the hold- and setup-time requirements on the data are always relative to the arrival of the clock at the register.

In order to minimize this effect, clocks in some devices are, for example, distributed in tree-like clock distribution networks as shown in figure 18 to ensure a similar arrival time for a clock at all registers.

Even if this kind of clock structure is implemented, clock skew can still be present due to imperfections in the distribution network or in some other scenarios, for example in cases where there is a phase shift between the source and destination clock or where the source and destination are not within the same clock distribution network.

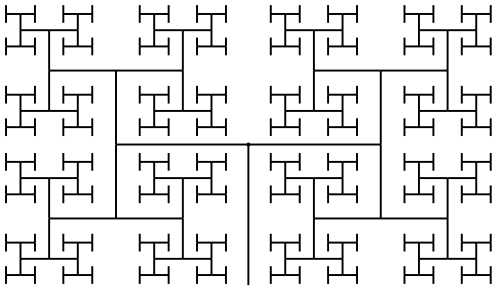


Figure 18.: Example for a clock distribution network to minimize clock skew

3.3.1.3. Resets, Recovery- and Removal-time

The assumption for many digital circuits is that they will start operation from a specific predefined state. Starting a circuit from a state where all contained registers have a random value can cause the circuit to produce unintended results since the starting condition might be a state that is not reachable with the intended logic structure. In such a situation, other invalid states might be reached from this state, and the correct function of the circuit might be compromised.

In order to avoid this, reset signals are used to set registers into a defined starting condition. This is especially relevant during the power-up of a device because this process can lead to undefined register states but is also used to restart processes or functionalities from a defined point in a logic sequence. Since many components and logic circuits require a reset signal, they will often also have separate connections for them, independent of the regular logic inputs and similar to the clock distribution discussed above.

Nevertheless, reset signals are still logic inputs and, as such, still have timing requirements relative to the clock[3], similar to the setup- and hold-time discussed in the last section. The difference is that for a reset signal, applying the reset cannot cause relevant timing violations since the circuit will not perform any actions afterwards because it is held in the reset state for the following clock cycles. The relevant part is the release of the reset, where the circuit is started again. The timing requirements discussed before still apply, but only in one direction – release of the reset signal.

The recovery-time is a different name for the setup-time requirement of the release of the reset signal relative to the first active clock cycle after the release. The reset signal needs to be stable for this time before the arrival of the first clock edge which is supposed to be outside of the reset state.

Removal-time is the same concept as hold-time, applied to the removal of the reset only. The reset signal needs to be stable for this time after the first clock edge outside of the reset state.

[3]There are some common misconceptions around the concept of asynchronous reset signals. They will be discussed in section 3.7.5. This statement also holds for asynchronous resets depending on the component manufacturer's definition of *asynchronous*.

3.3.1.4. Metastability

The last sections discussed the timing requirements that data arriving at registers needs to fulfill relative to the clock arriving at these registers. Now, the consequences of violating those requirements will be discussed. Violations should normally be avoided by designing the circuit around the requirements, but as we will see in section 3.7.4, there are some situations where avoiding them is fundamentally not possible.

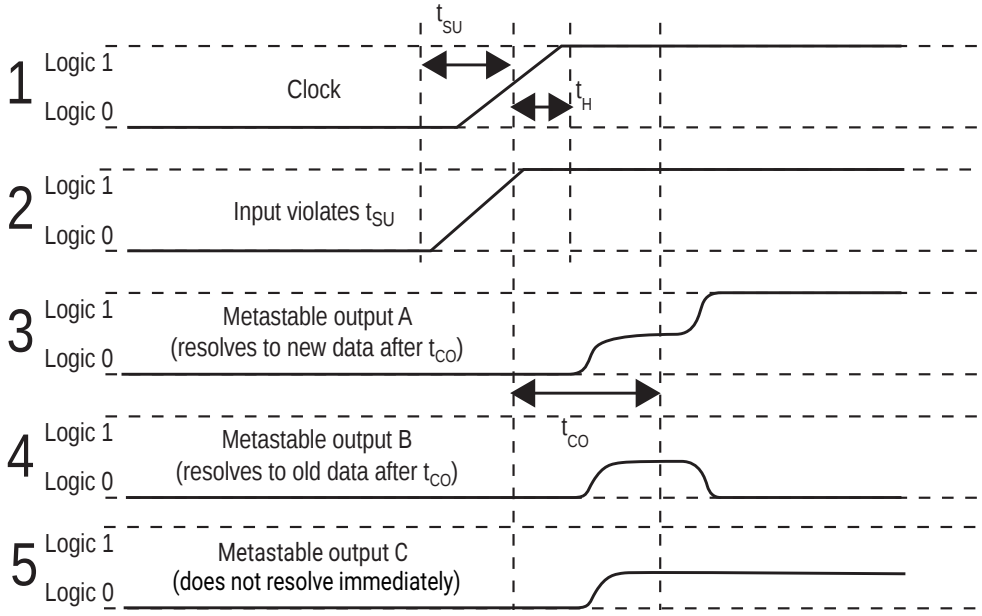


Figure 19.: The input in line 2 violates the setup time t_{SU} relative to the clock edge in line 1. Possible results are a metastable state that resolves to Logic 1 outside of the time window t_{CO} (line 3) or a metastable state that resolves to Logic 0 (line 4). In a very unlikely case, the output can also stay in a metastable state for a longer time period (line 5). Adapted from [28].

In regular operation, the output voltage of a register after a clock-to-output delay time t_{CO} after the clock edge is on a defined logic level of 0 or 1. If one of the timing requirements of the register is violated (line 2 in figure 19), the time t_{CO} after which it needs to have flipped its output to the new value cannot be guaranteed anymore. The output voltage can be somewhere between the two logic levels in a metastable state after the time t_{CO}. This metastable state usually quickly resolves into one of the defined logic levels. For small violations of the setup-time, the output is more likely to resolve into the logic level that was intended with this transition (line 3). For larger violations, the output will most likely resolve to the previous logic level (line 4). The exact timing of the input signal determines the result and the output of the register is not predictable in such a case.

In this situation, the time needed to resolve to a defined state is also not predictable and it can, in theory, stay in a metastable state indefinitely. In practice,

small fluctuations in the supply voltage will prevent that from happening. However, this unpredictability of the output timing can cause further timing violations since the output data of the register will itself connect to other registers, which again have timing requirements regarding the arrival time of that data. An input timing violation at a register can, therefore, propagate to the following registers, even if the path between them would fulfill the nominal requirements for the propagation time.

Timing violations have therefore to some degree the ability to propagate through a system and cause failures in the following logic which can be larger than a single undefined output of a register. A single undefined output of a register can – depending on the logic – of course also be a larger problem in itself, without considering this propagation.

The treatment of situations where these metastable states cannot be avoided is usually a question of managing failure probability [28]. More about these cases will follow in section 3.7.4.

3.4. Recurring Structures

This section will discuss some circuit structures that will occur in many different designs since they implement commonly needed functionalities such as data storage or clock frequency changes. Some of these structures will differ from the concept of clocked source and destination registers with logic gates in between them that was shown in figure 15 of section 3.1.1. The designer of a circuit can write down some of these structures in an HDL, but due to their widespread use, they are typically provided as finished parts to be included in a design. If the implementation of one of these structures is already intended for a specific device or technology, the timing considerations described in the last section within this structure have already been taken care of by those who designed it. In terms of timing, a user typically only needs to care about connections to and from these entities.

3.4.1. Memory Structures

Probably the most commonly used structure like this is memory in many different variants. One has to distinguish between volatile and non-volatile memory. Volatile memory only keeps the information contained in it while the device is powered, whereas non-volatile memory will also store the information without electrical power. Specifics about non-volatile memory will not become relevant during this thesis and will, therefore, not be discussed here.

Volatile memory loses the data stored in it once the power to the device is cut. It is again divided into the two categories of static and dynamic memory, each again with several subcategories. Dynamic memory (DRAM) stores the information in the charge of capacitors, which requires constant updating cycles in order to counteract the capacitor discharge and to preserve the information. Static memory (SRAM) stores information in inverter-pairs and does therefore not require these updating cycles, but comes with a speed and space disadvantage when compared with dynamic memory.

3.4.1.1. RAM

A Random access memory (RAM) is a type of memory where contents can be accessed in any order. Independent of the implementation as static or dynamic memory, the interfacing to the simpler types of this kind of memory will usually be similar and consist of a data-input and -output, an address, a clock and a read- and write-request signal. The read-request signal might be dropped in some cases. There are also memories which have a second set of these signals acting independently of the first one. Those are called dual port RAM.

Figure 20 shows an example transaction with a RAM. The exact timing properties of a memory can differ between different implementations. For example, the latency between the application of the *rdreq* signal and the arrival of data on the data_out port in figure 20 might take more clock cycles in other RAM implementations.

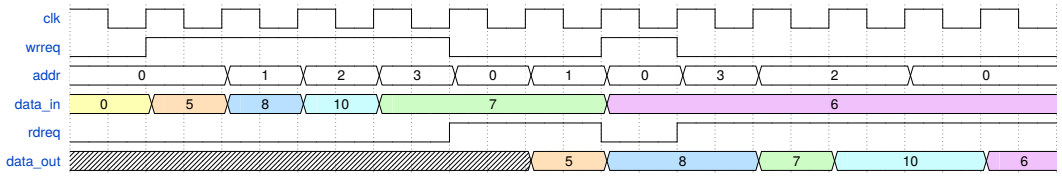


Figure 20.: Wave diagram of write and read transactions on a standard RAM interface. The *wrreq* signal writes the content of *data_in* into the memory location specified by *addr*. The memory content can then be accessed using *rdreq* and *addr*.

For some memory implementations, the latency might not even be a constant number of clock cycles and therefore require an additional output signal such as „data_ready“ or „data_valid“ in its interface to signal the arrival of the requested data on the data_out port. There can be an impact on the available bandwidth in these cases since the read side of the memory interface has to wait for the arrival of the data.

This leads to modifications to the standard interface shown in figure 20 in order to maintain a high read bandwidth. These modifications can then, for example, introduce the concept of burst transactions, where data is not read or written word by word but in larger blocks of data. The implications of this will become relevant to system performance in section 4.4.4 of this thesis and will be discussed in more detail there.

Another concept that can arise from dealing with higher latency memory is caching, where a faster and smaller memory is connected as a kind of buffer for frequently accessed addresses between the actual memory and the entity that wants to interface with it. Sometimes this is even done in multiple cache levels with increasing memory latency. This is another example of a situation where tradeoffs between different system properties can be made, an idea that was already introduced earlier.

The size (number of bits) of the address signal together with the size of the data ports determines the maximal size of the memory. Since many systems operate with instructions or data-words with sizes in powers of 2, the memories within them will

usually also have these sizes as width of their data-in and -ouput ports (64-bit memory, 32-bit memory, etc.). As we will see later in this thesis, finished memory components might for this reason be only available in certain sizes because others are not economical. This means that a design that in theory only requires a certain amount of memory space can in some cases need more physical resources than that because it cannot fully utilize them.

3.4.1.2. FIFOs

A First In-First Out (FIFO) memory is a type of memory where the order in which data can be read is the same order in which the data was written. The access to a random piece of the data is not possible without reading all of the data. Conceptually a FIFO can be built from a RAM by steering the address ports of the RAM accordingly. The interface of a FIFO does therefore not contain address ports (see figure 21). Additionally, signals can be added to give information about the filling level of the FIFO. Since data is always read in the same order in which it was written, the FIFO can become full. In a RAM data would be overwritten and it therefore does not have a filling level unless the rest of the logic defines one.

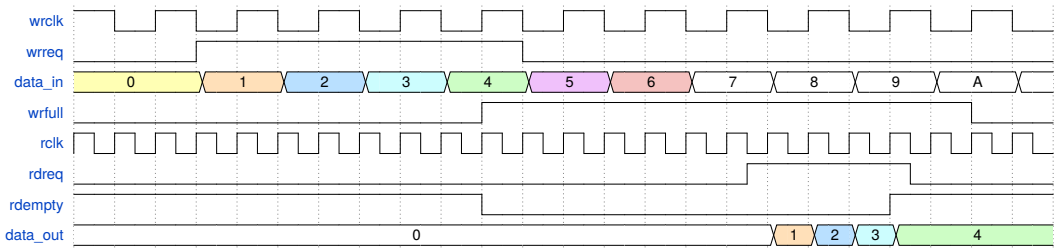


Figure 21.: Wave diagram of write and read transactions on a standard FIFO interface. Compared to figure 20, the *addr* port is missing and the data is accessed in the same order as it was written with the *wrreq* signal.

FIFOs where the read and the write side are clocked by different clocks are called dual clock FIFOs. They are a useful tool for clock domain transitions, which will be discussed in section 3.7.4. The same can be done with dual port RAM if the two ports are driven by different domains.

3.4.1.3. Shift-Registers

A shift-register is a chain of registers where the contained data bits can be pushed to the next register in the chain with a clock and enable signal (see figure 22). They can be used as a type of memory to store data, but we will encounter variants of it during this thesis that implement also other functionalities.

One of these variants is a linear feedback shift-register (LFSR), which can be used to generate pseudo-random outputs. In an LFSR the input bit of the first register in the chain is produced by tapping into the outputs of specific registers in the chain

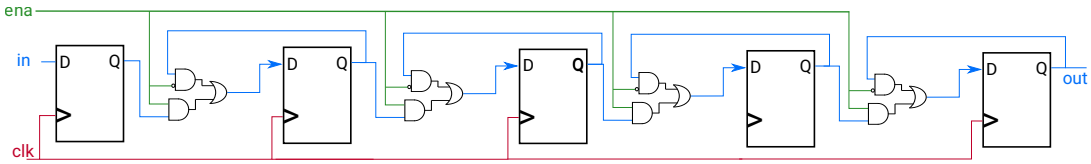


Figure 22.: Schematic of a shift register. When the *ena* signal is set to '1', the contents of each register will propagate to the next one on a rising edge of *clk*.

and combining them with XOR or XNOR feedback gates, the result of which is then connected to the input of the first register.

For a shift register of length N it is possible to produce an LFSR that will cycle through $2^N - 1$ different states by choosing the correct number and location of these so-called taps of the feedback. The number and location of taps for different N can be found in [29]. It is not possible to reach all 2^N possible states, since the state where all registers of the chain hold a value of 1 will also produce 1 in the XNOR feedback and therefore not reach any other state from there. The same situation applies to the all-zero state of a shift register with an XOR feedback.

In addition to generating a pseudo-random sequence of outputs, LFSRs can be used to implement very fast counters. As discussed earlier in section 3.3.1.1, the maximal speed at which a circuit can operate is related to the Fan-in and Fan-out of the registers in a circuit. For the implementation of an N-bit long binary counter, the most significant bit will have a Fan-in of at least N, since the state of all lower bits determines what happens to the highest bit if the counter increases by 1. This can be the limiting factor for the clock frequency at which the counter can operate.

An alternative is to use an LSFR instead of a counter, which can usually operate at higher frequencies since registers only have to connect to their neighbours in the register chain and therefore have a lower Fan-in and Fan-out. This method comes with the two drawbacks that the counter is now encoded in pseudo-random values that will likely need to be converted back into a different representation at some point and that the repetition length of the counter decreases to $2^N - 1$, while a binary counter would repeat after 2^N cycles. This difference can create complications as we will see in section 4.8.

Other, less common shift register variants and their application will be discussed when they appear in the system design[4].

3.4.2. PLLs

Phase locked loops (PLLs) are used to derive a clock with a potentially different frequency from a reference clock. In contrast to most other topics in this chapter, they are usually analog components.

The phase and frequency of the reference clock clk_{ref} of a phase locked loop (PLL) is compared to a clock generated by a voltage controlled oscillator (VCO) in the phase-

[4]Sections 4.8, 4.6 and 4.6.3

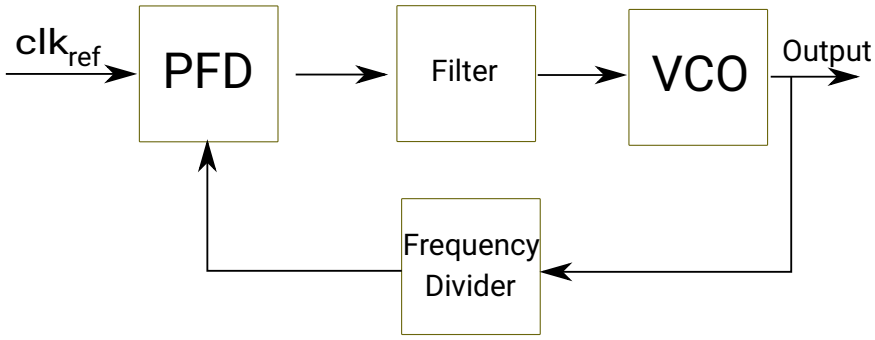


Figure 23.: Simplified schematic of a phase locked loop.

frequency detector (PFD). This component generates an output that determines if the VCO frequency should run faster or slower, which is then converted in a filter or charge pump to an input voltage to the VCO. The output of the VCO is the output clock of the PLL.

The frequency of the PLL can be adjusted by inserting a clock divider between the VCO output and the PFD. Also non-integer relations of the frequency of the reference clock clk_{ref} to the output frequency are possible if the setting of the frequency divider is dynamically changed between two or more values. This is then called a „fractional“ PLL (fPLL).

The frequency divider divides, for example, a specific fraction of clock cycles to frequency x, the others to frequency y and the mean of the resulting input voltage to the VCO will cause it to run with a frequency z, which has a non-integer relation to the frequency of clk_{ref} [30].

3.5. ASICs

Implementing complicated digital circuits is often not done with discrete electronic components, since other methods can offer better performance due to higher transistor density[5], lower production cost per piece and a lower space and power usage. One of these methods are application specific integrated circuits (ASICs). ASICs are produced from silicon wafers. A photolithographic process is used in multiple layers to create electronic components and interconnects in a semiconductor material.

ASICs can have digital and analog components. The digital parts of an ASIC can be described with an HDL and can then be mapped to the component libraries of the production process by software tools. Analog parts of an ASIC are not described by HDLs and have to be drawn. The created structure sizes determine the achievable transistor density and, therefore, the maximal operation frequency. The custom ASICs used for Mu3e are based on a 180 nm process. Modern processors, for example, are produced with structure sizes of below 10 nm.

[5] see Section 3.1.1.

The initial costs for an ASIC can be quite high and will increase significantly with lower structure sizes. In order to avoid a part of the initial costs, wafers can be shared between different projects, which was also the case during the development phase of the MuTrig and MuPix ASICs in Mu3e.

Once an ASIC is produced, its functionality is fixed and can only be altered in ways that were foreseen as configurable settings during the development process. Any changes or bug fixes require the production of a new ASIC with all the delays and costs that come with it.

3.6. FPGAs

A field programmable gate array (FPGA) is conceptually the same thing as an ASIC, with the difference that an FPGA has enough configuration options to change its behaviour in order to implement basically arbitrary HDL code. It is, therefore, not application-specific like an ASIC but can be reprogrammed in the field. This comes at the cost of lower performance since the components on an FPGA are not tailored to a specific application. Also, the option of reconfiguring itself requires storage elements and logic for the configuration, which require space on the device. This space is then not available for logic elements, which decreases the usable logic density and, consequently, the performance of an FPGA compared to an ASIC of the same structure size.

The advantages of an FPGA over an ASIC are the lower costs for a small number of units, the possibility of updating the design, and the faster development cycle. Bug-fixes or changes to a logic circuit on an FPGA can be implemented in a matter of hours compared to many months for an ASIC. Verification and simulations are essential for both ways of implementation. However, on an FPGA, it is feasible to experiment a bit with a design without running extensive tests and simulations beforehand.

3.6.1. FPGA Architecture

This section's discussion of FPGA architecture will primarily follow the architecture of intel FPGAs, since most FPGAs used during this thesis are produced by intel. The ArriaV FPGA will be used as an example. In general, it consists of Logic Array Blocks (LABs), Memory Blocks and Digital Signal Processing (DSP) blocks.

Conceptually, LABs are general-purpose blocks containing registers and configurable lookup tables. Most of the area of the FGPA consists of them and most of the HDL code written by the designer will be implemented using this type of block. Memory blocks are used to implement any kind of mass data storage, as discussed in section 3.4.1. DSP blocks are specialised on the efficient implementation of arithmetic functions.

Figure 24 shows the schematic distribution of blocks on an ArriaV FPGA. Other FPGAs will consist of similar building blocks but will differ in the exact structure, size, number and arrangement of them. They might also include different types (for example a full processor block) that are not available on this specific FPGA.

The exact structure of the blocks in an FPGA is relevant, since the tools have to map HDL code written by the designer onto the blocks available on the target FPGA, and the number and type of blocks needed to do so determines the space that a design requires on the FPGA. HDL source code can, therefore, be optimised to fit a specific FPGA's resources efficiently. For the designs in Mu3e, this will be an especially important topic for memory blocks.

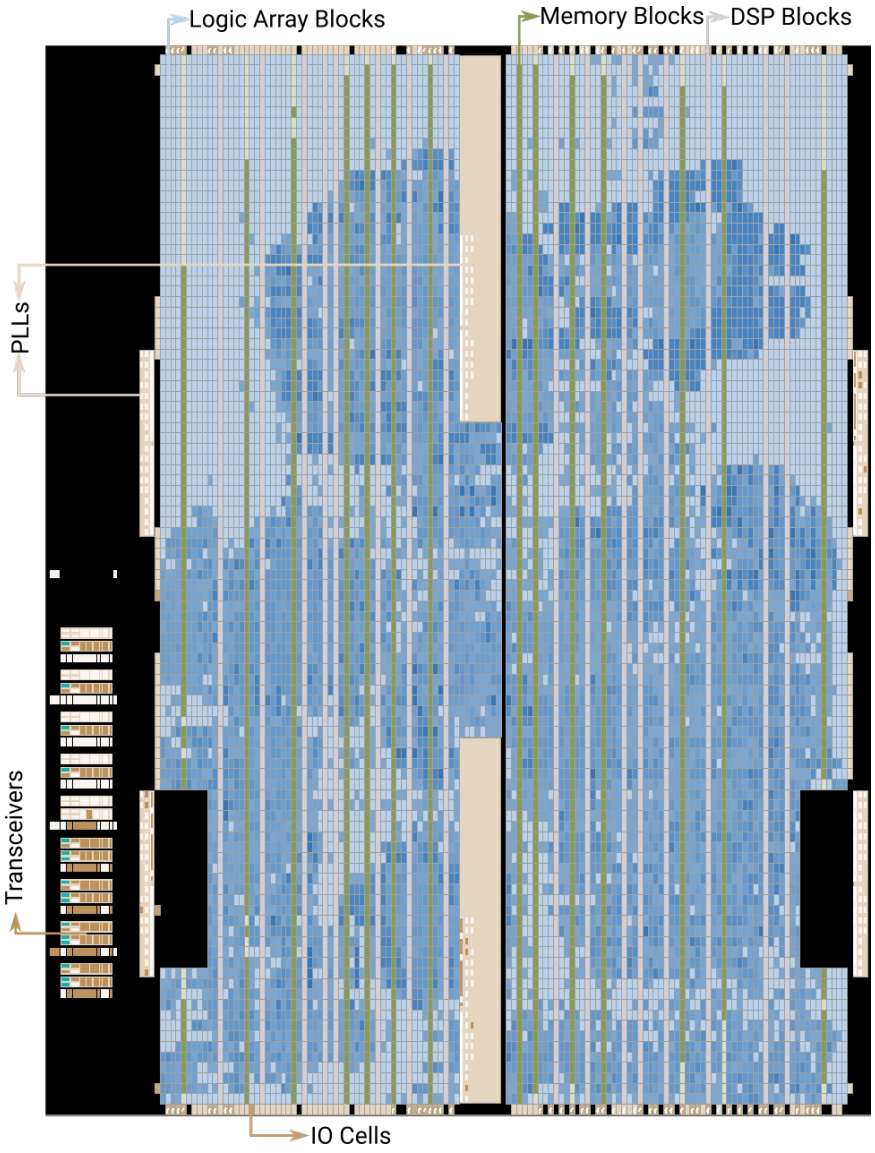


Figure 24.: Building blocks of an ArriaV FPGA. DSP, Memory and Logic Array blocks are freely configurable by the designer. Other parts of the FPGA, such as transceivers or PLLs, have a specific predefined function and offer only limited configurability.

Logic Array Block (LAB)

A LAB of an ArriaV FPGA consists of ten adaptive logic modules (ALMs), which contain a configurable lookup table (LUT), two adders, and four registers each. The registers drive the output of the cell. The output can then travel through one or more lookup tables of other ALM blocks before the signal is registered again.

Depending on the configuration by the user, only a part of the elements in figure 25 might be used. In a long combinatorial path, for example, it might be necessary to combine the lookup tables (LUTs) of multiple ALMs – bypassing their registers – to achieve the desired logic for a register of another ALM.

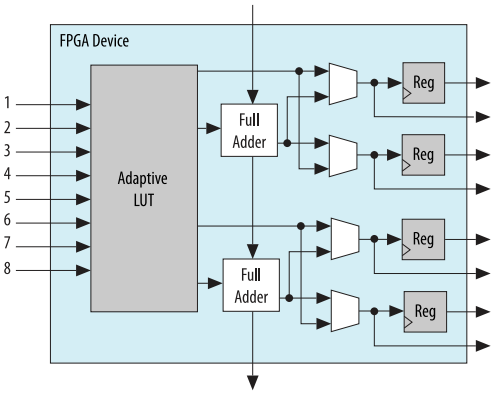


Figure 25.: Schematic of an ALM [31]

Although the LUT in figure 25 has eight inputs, it can only implement a full 6-input LUT[6] and is implemented as a fracturable ALM, which means it can be split into two smaller, independent LUTs in various ways[7]. According to Intel, this is more efficient in terms of space on the FPGA compared to single register ALMs [32].

Memory and MLAB Blocks

Memory structures are very frequently needed in digital designs. For that reason, FPGAs include dedicated memory blocks that are designed to implement memory structures efficiently. It is possible to implement the same functionality in LABs, but specific hardware for memory allows for much higher speed and area efficiency since the additional logic to implement arbitrary configurable functions is not needed. The memory on the ArriaV example device comes in M10K blocks, where each block can contain a maximum of 10240 bits of information[8].

It is important to note that this capacity depends on the combination of width and depth of the memory structure (size of *data_in* and *addr* in section 3.4.1.1). The maximum value of 10240 bits per block can only be reached in specific width/depth combinations. Uncommon width/depth combinations will lead to a significantly lower capacity per block.

Another important factor is that M10K blocks cannot be shared between independent structures. Implementing two random access memories in M10K blocks will

[6] A configuration as 7-input LUT is also possible, but does only support a subset of all logic combinations.

[7] Two 4-input LUTs, two 5-input LUTs with two shared inputs, other combinations with shared inputs [32]. A more detailed example of a LUT implementation can be found in appendix A.1.

[8] Details can be found in [31], p.21. The amount of read/write ports also plays a role.

always use at least two blocks, even if the capacity of one block would be sufficient to implement both memories. Implementing a very small memory in M10K will also use a full M10K, making it unavailable for other design parts. In order to efficiently use the memory resources of an FPGA, the design needs to be written in a way to ensure high memory utilisation.

In general, small memory structures and uncommon width/depth combinations will lead to lower memory block utilisation and should be avoided. Since „avoiding small memory structures“ is not always feasible, Intel has implemented a special type of block to implement small memories. These MLAB (Memory LAB) blocks are technically normal LAB blocks (see section above) with the added functionality to be convertible into small 640-bit memory blocks. The interesting thing about that is that a LAB block only contains 20 registers (2 per ALM, 10 ALMs per LAB), but can save 640 bits of information in MLAB configuration. The exact technical details of these blocks are not public, but it is presumably realised by repurposing the LUT mask SRAM cells that would usually be used to hold the configuration of the ten 6-input LUTs in a LAB block. A full 6-input LUT requires $2^6 = 64$ configuration bits in the LUT mask, adding up to 640 bits for a complete LAB, which matches the maximal capacity of an MLAB block.

Not every LAB can be used as MLAB, but MLABs still make up a significant fraction of the total memory capacity of the FPGA. For the ArriaV used in this thesis, about 10% of the memory capacity is generated by MLABs.

MLABs should not be used to construct larger memories since they are spread across the FPGA and combining large amounts of them will, therefore, have disadvantages in terms of possible operation frequency. However, for smaller memories they can contribute to high memory utilisation by saving M10K blocks.

DSP Blocks

DSP Blocks implement common digital signal processing functions, such as multiplications. Standard LABs could also be configured to serve the same function, but dedicated hardware allows for more efficient implementations. The designs that will be discussed in this thesis do not contain a lot of arithmetic functions. DSP blocks are, therefore, not very relevant during this project and form unused space on the FPGAs in Mu3e. However, FPGAs are not designed for a specific project but for a wide range of applications, where some of them profit from faster DSP functions.

IP-cores and Hard-IPs

Intellectual Property-cores (IP-cores) are entities within an FPGA design provided by the device manufacturer or licensed by a third party, sometimes with a few configuration options for the user. They can either be embedded into the FPGA fabric during the production process (hard-IPs) and provide a higher performance due to their predefined, hardcoded function (similar to DSP or memory blocks) or be constructed of the existing resources of the FPGA during the design synthesis (soft-IPs).

A typical example is a CPU on an FPGA. While it is possible to write a CPU in an HDL and afterwards compile software for this CPU and load both onto an FPGA, having a CPU in an FPGA design is such a widely used concept that the manufacturer of the FPGA provides a finished solution for it. In the case of the Mu3e project, most of the FGPAs will use the Intel NIOSII processor, built from LAB and memory blocks available on the FPGA. FPGAs can also include hard-IP processors, which are capable of achieving much higher operation frequencies than the NIOSII used in Mu3e.

Hard-IPs can provide fast input-output (IO) capabilities that would be technically difficult to achieve in the fully configurable environment of LAB cells. PCIe and fast ethernet connections fall into this category.

Another task of hard-IPs is to provide access to analogue capabilities that might be embedded in the FPGA. Examples of such a case are PLLs (section 3.4.2) or analog-to-digital converters.

Clocking and Insertion-Delay

The difficulty with clock distribution within an FPGA's architecture is similar to the discussion of clock skew in section 3.3.1.2. The clock signal needs to arrive at all destinations with minimal differences in the arrival time. Clock networks similar to figure 18 in the earlier section are used to accomplish that task. The problem here is that the amount of clock routing resources is limited and fixed once the FPGA is produced.

In order to counteract this issue, the example device that we are looking at (ArriaV) uses a hierarchical structure with global, regional and periphery clock networks. The global clock network distributes clocks that must reach all parts of the FPGA. More clock routing resources are available for clocks that only need to get to a specific area of the FPGA. These regional clock networks are arranged in quadrants in the four corners of the FPGA. Additional periphery clock networks drive clocks across the FPGA in horizontal or vertical direction and are intended to drive signals in or out of the FPGA [33].

These clock networks ensure the low skew distribution of signals and are not exclusively used by clocks. Other high fan-out signals like resets or clock enable signals can also make use of them, which effectively creates a way to circumvent the placement issues that would normally come with a high fan-out[9].

However, doing so creates another potential problem, which is illustrated in figure 26. One register ($clk2$ *Source*) creates the signal $clk2$ via a clock divider. A second register right next to it drives data to a destination, which, for the sake of simplicity of this example, is also physically located next to the other two registers and is clocked by the signal $clk2$. The path of the data between the source and destination register is short. If $clk2$ is to be routed through a global clock network then the path of $clk2$ to the destination register can be quite large, since it first needs to be routed to the insertion point and then go through the whole network before arriving at its

[9] discussed in section 3.3.1.1

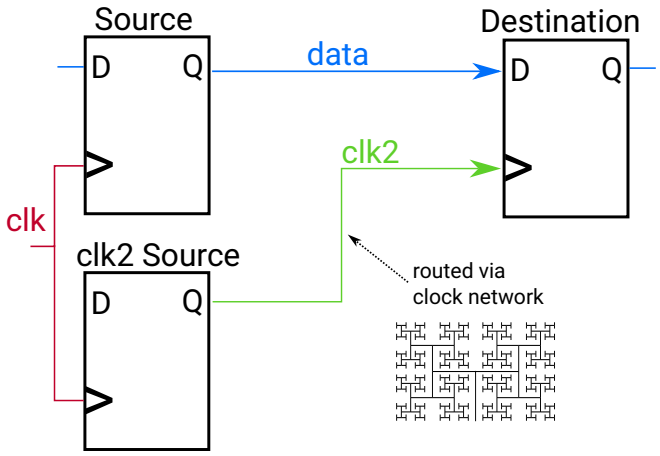


Figure 26.: Schematic illustration of insertion delay. The datapath between the source and destination register can be significantly shorter than the path of the clock to the destination register since the clock might be routed through a clock network.

original location again. This so-called insertion delay of the signal *clk*2 into the clock network can cause a hold-time violation on the destination register, and the tools will need to add wire to the path of the data in order to match the insertion delay of the clock.

Therefore, clocks and resets on FPGAs are different from other signals on the FGPA and should not be mixed with them. Clocks should not be used as signals and signals should not be used as clocks. Mixing them connects two areas of propagation delay with each other that are not supposed to be connected. While the use of statements such as *rising_edge* on random non-clock signals is legal in VHDL, it is not to be recommended for FPGA designs since significant addition of wire can be necessary to match the delay of a clock network. The situation for ASICs or FPGAs with a fundamentally different architecture can be different.

Inference

A piece of HDL code that is intended to be implemented on an FPGA needs to be mapped to the available hardware blocks of the target FPGA discussed previously, not just in terms of physical location but also in terms of the type of block used. A FIFO memory, for example, can be implemented via an IP-configurator tool, where the designer can specifically ask for an implementation using M10K or MLAB blocks. The same FIFO memory can be created by writing down an HDL description of a FIFO memory, which the compilation tool identifies as a memory structure and implements using memory blocks depending on the compilation settings for memory inference. The second option is preferred for portability between manufacturers since it does not rely on their specific IP-generation tools.

In the HDL code, directives can be provided to the compilation tool to force or avoid implementation with a specific type of block. Not every structure identified by the tool to be implementable within memory blocks actually benefits from doing so. It can be helpful to implement these structures in LABs, for example in cases where memory resources in an FPGA design are almost depleted.

3.7. Advanced Timing Topics

When a firmware design has to be compiled for a specific target FPGA, the task of the compilation tool is to map the HDL code onto the existing device resources and to achieve timing closure by fulfilling all timing requirements for all connections within the design. These requirements have been discussed previously and this section will extend on topics related to fulfilling them.

First of all, the requirements of setup, hold, recovery, and removal time depend not only on the previously mentioned placement and the resulting connection distance between resources but also on external factors, such as temperature, supply voltage and device manufacturing variations.

Manufacturing variations might be again categorised into different speed grades. An FPGA will undergo a series of tests after production and will be rated for a certain speed depending on the test results. This speed rating is part of the part number and affects the price at which the FPGA is sold. The compilation tool is then instructed to achieve timing closure for a particular speed grade of that device. However, this rating still contains a range of different device behaviours, and timing analysis will usually consider the worst possible (slowest) FPGA that would still have been rated into this category as well as the fastest FPGA in the category. This leads to a slow and fast timing model that has to be considered.

In addition, all FGPAs, regardless of their speed rating, will slightly change their timing behaviour at different operation temperatures. The temperature can be controlled by the user, but will necessarily move within some range depending on the cooling system used for the FPGA, resulting in the need for a low- and high-temperature timing model.

These two circumstances lead to four combinations of conditions, which will be used as the timing models by the compilation tool in a multi-corner timing analysis. All four corner cases need to meet the requirements for setup, hold, recovery, and removal time in order to ensure that the design will function properly with the used speed grade of the FPGA at the expected operation temperature range.

3.7.1. Slack

The available overhead time before a timing requirement (for example the setup time) is violated is called the slack. Figure 27 illustrates this with an example of a setup slack. The other timing requirements have their own slack values for each connection.

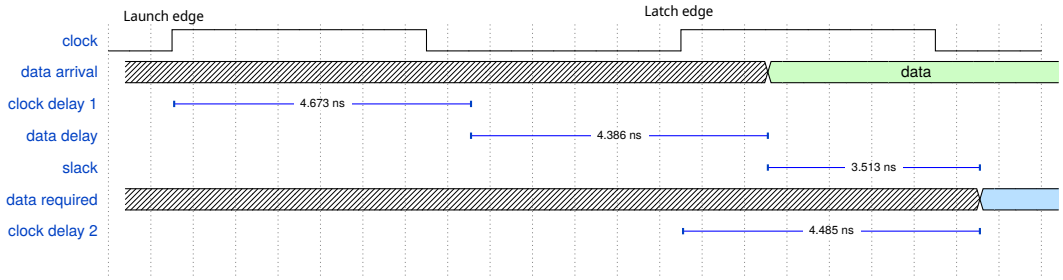


Figure 27.: Example diagram of a setup slack. From the launch edge of the clock the clock needs the time *clock delay* 1 to travel through the clock network to the source register, then the data from that register needs the time *data delay* to arrive at the destination register, where the data is required when the clock arrives there (*clock delay* 2). The difference between the data required and data arrival time is the setup slack (ignoring things like clock uncertainties for simplification).

When slack values are positive, the timing requirement is met. Negative slack values represent timing violations. During the compilation, a form of maximisation of all slack values takes place, and the results are analysed afterwards using the timing models for the four corner cases discussed earlier. The result is a distribution of slack values for each corner for each requirement of setup, hold, recovery, and removal time for each clock domain in the design. An example for a setup slack distribution in the slow high-temperature (85 °C) corner is shown in figure 28.

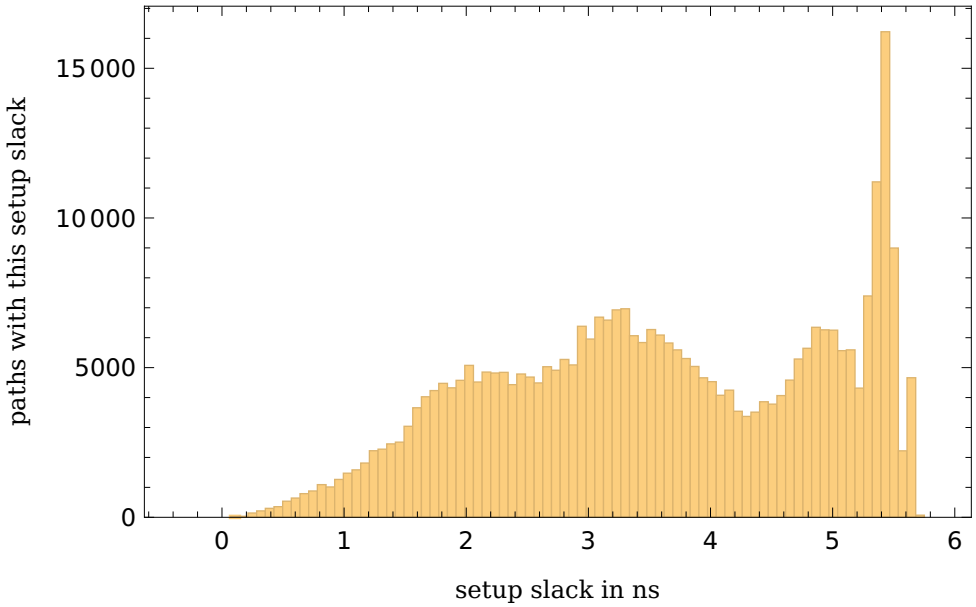


Figure 28.: Example of a setup slack histogram with only positive slacks.

All of these histograms need to contain only positive values in order to ensure that the design will function properly. Although the histogram contains relevant information about the general criticality of the timing situation in the design, the really relevant part is the lowest entry. A single negative entry represents a timing violation and can lead to unintended behaviour of the design. Since this needs to be avoided in general, it is not relevant how much unintended behaviour is expected and, therefore, also not how many negative slacks exist. However, if negative slacks exist, it is still useful to look at the amount in order to find the proper solutions. A design with hundreds of negative slacks is likely not fixable by changing only the single path with the minimal slack.

It is important to note that the slack values discussed here are always a direct result of Fan-in, Fan-out and the resulting placement of registers, which was discussed in section 3.3.1.1. Changing logic paths in order to ensure positive slacks is, therefore, a question of reducing Fan-in and -out of the registers within it, relocating into areas with lower resource usage and pipelining the logic into multiple cycles[10].

These changes do not necessarily need to focus on the most critical path (the lowest entry in figure 28) to achieve something. In general, the paths towards the left end of the histogram tend to be the more difficult ones in terms of timing. If the critical path is conceptually difficult to pipeline, it is possible that changes to other paths on the left end of the slack histogram will free up enough resources to close timing of the critical path, even if the logic of the critical path itself was unchanged. Similarly, if the critical path is a connection between the source register A and destination register B, it can be useful to investigate the logic paths which have A as a destination or B as source. Changing those can loosen constraints on A & B's position, which can again open a possibility to close timing for the critical path between A and B without changing the logic inside it.

Another reason for failed timing closure can be the lack of appropriate resources. A memory instantiated automatically by inference from HDL that usually uses an M10K block for the implementation could fall back to LAB cells when no more M10K blocks are available. The same applies to routing resources: The compilation will automatically promote some signals into the clock distribution networks, which will be fully filled at some point. It can therefore be beneficial to specify which parts of the design are implemented with which resource type. Changes of these specifications can change the paths which receive the lowest slack values. Even if the resource specification does not solve the timing violation, it can be used to move the problem into a part of the design where other solutions might exist.

In section 3.6.1 it was mentioned that clocks and signal routing resources should not be mixed. There is a case where there might be a point for doing so. If a particular signal is responsible for timing issues that are otherwise difficult to solve, promoting that signal to clock routing resources will radically change the timing behaviour of the signal. It is not given that the situation will improve, but the problem will change, which can be part of a solution process.

[10] already discussed in section 3.3.1.1.

3.7.2. Duplication and Merging

Depending on the logic, a Fan-out reduction can be achived by duplication of registers. While pipelining is mainly aimed at reducing Fan-in by inserting additional clock cycles, duplication aims at reducing the Fan-out by connecting a part of the original Fan-out to an identical copy of the register instead. Additional resources are required in this case to produce the copy and the logic path leading to the copy. Multiple copies are possible to further lower the Fan-out of the register.

This method is in some cases automatically applied by the compilation tool, but can also be made explicit within the HDL source. Explicit duplication can have advantages since the distribution of the Fan-out accross the copies of the register can be specified in a way that represents the design structure. For a register C that connects towards two otherwise independent entities A and B, the register can be copied in a way where this independence is preserved by connecting one copy C_1 of the register to entity A and the other copy C_2 to entity B. Other ways of splitting the Fan-out of the register during the copy will in some form further tie the physical location of A and B together since they are both driven from the same source register. We will encounter a slightly more complicated example of this idea during the design discussion in section 4.4.4, where this idea is conceptually combined with multiple pipelining steps between C_1 and A in order to completly disentangle any dependence on the location of B.

The reverse process – merging of registers – is also sometimes automatically performed by the compilation tool when logic duplication is detected in the HDL source. However, in the version of the tool used during this thesis these merges seemed to target lower resource usage without much consideration of timing implications of the merge. An attribute can be assigned to signals to prohibit this behaviour, which is necessary, especially during intentional explicit duplication.

3.7.3. Optimisation Process and Seed Variation

It is also important to keep in mind that the slack distribution shown in figure 28 above is a result of a large optimisation process during the compilation, which maximises the minimal slack. This can mean that the path with the minimal slack at the time when the optimisation efforts stop is not actually the reason for failed timing closure. The reason might be in other paths with low slack values. Also, this optimisation is unlikely to find the best possible value for the minimal slack. There are settings within the compilation tools which configure how much effort is supposed to go into optimisation. These settings can influence the result but will also increase compilation time. A firmware compilation which takes $\mathcal{O}(hours)$ is not unusual.

Since the optimisation is unlikely to find the global maximum, the result will depend on the starting seed of this process, which can be configured in the compilation tools. The influence of the starting seed on the compilation result can be quite significant. Figure 29 shows distributions of the minimal setup slack for different FPGA designs that we will encounter during this thesis. The figure was created by always taking the lowest entry (minimal setup slack) of the previously shown histogram in figure 28 and

recreating that histogram 200 times with different starting seeds for each compilation of the designs A, B, C and D.

These are real designs, but meant only as an example of the influence of seed variation at this point, which is why they are labelled with A, B, C and D here[11]. Clock domains were not considered and only the domain with the lowest minimal setup slack was entered into the histogram. Slacks are calculated for the slow 85°C timing corner. Clock frequencies reach up to 156.25 MHz for A and B, 250 MHz for C and 50 MHz for D.

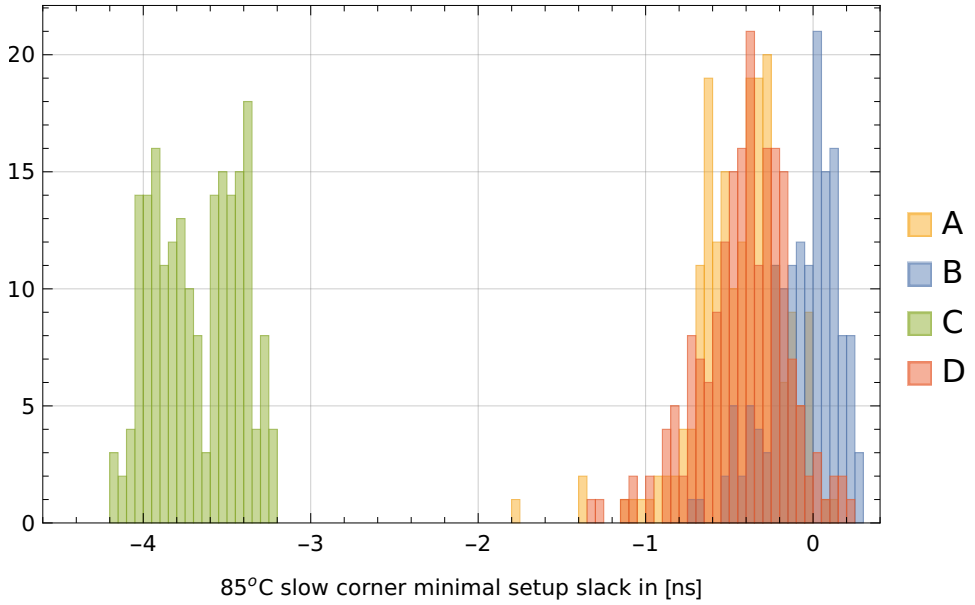


Figure 29.: Minimal setup slack variation for different example designs

Version	Mean minimal setup slack [ns]	σ_{setup} [ns]	Timing Closure probability	Max. frequency [MHz]
A	−0.44	±0.27	1 %	156.25
B	−0.06	±0.21	48 %	156.25
C	−3.68	±0.25	0 %	250
D	−0.41	±0.26	5 %	50

Table 2.: Minimal setup slack variation for different example designs

The resulting minimal setup slack distributions show standard deviations between 0.21 and 0.27 ns. Example C contained a mistake in the design for the version that was used for these measurements and, therefore, shows very significant negative slack

[11] A refers to the scifi FEB, B the mupix FEB, C the SWB in the DDR4 version, C the Max10 firmware. Details about the content and function will follow in later chapters. All on firmware compiled with quartus 18.1 on standard settings.

values. The other three examples also have negative minimal slack values on average, but the distributions are also reaching into positive values. This means that for a compilation of these designs, there is a certain chance that they will achieve timing closure, depending on the fraction of seeds that result in positive slack values. The highest observed difference between the worst and best setup slack is at 1.9 ns, for example A, which is almost 30 % of the lowest clock period of that design. Consequently, the timing situation within a design can only be judged adequately by looking at the compilation results of more than one seed value since single seed results can deviate significantly from the average expected outcome.

Ensuring that every compilation of an FPGA design will meet all timing requirements regardless of the used seed is not always desirable. As long as there are seeds where timing is closed, the task is to find one of those seeds, which is a question of compute time. Shifting the minimal setup slack distribution to the right will lead to lower compute time in order to find a working seed, but will also come with a design effort to make this shift and usually also with a higher resource usage on the FGPA. A tradeoff between these two issues needs to be made. For the examples listed above, A and C were considered to require action in order to increase timing closure probability. Example D is a smaller, therefore fast compiling FPGA design where finding a working seed did not require a lot of compute time. For Example B, the timing closure probability was considered sufficient.

The values and distributions from above were all considering the setup slacks for the slow 85°C corner. For each compilation there is a result for all four corners and for all four slack values. The setup slack in the slow 85°C corner is just the value which was the most likely to violate any requirements during the firmware development of this thesis.

The right side of figure 30 shows a comparison between the slow and fast corner for the minimal setup slack at 85°C. Examples A, B and D looked quite similar in the slow corner, which was already shown in the histogram of figure 29. However, A and D never fail timing in the fast corner, while B seems to be able to fail the fast and slow corners independently. This is somewhat unexpected for setup slacks since it shows that a design can meet the setup timing requirements on a particularly slow chip but, at the same time and with the same seed and temperature, fail to achieve this on a faster FPGA. This is possible since the setup slack depends not just on the travel time of a signal from one register to another but also on the arrival time of the clock at these two registers. The example also shows that the way in which timing requirements are not met depends on the design: The designs A and B, for example, are implemented on identical hardware, but with differing HDL source codes.

The left side of Figure 30 compares the two temperature cases in the slow corner. In general, a correlation between the minimal setup slack result at 85°C with the result at 0°C can be observed, with a higher slack at 85°C usually also leading to a higher slack at 0°C. However, design D seems to perform significantly better at lower temperatures compared to the other examples.

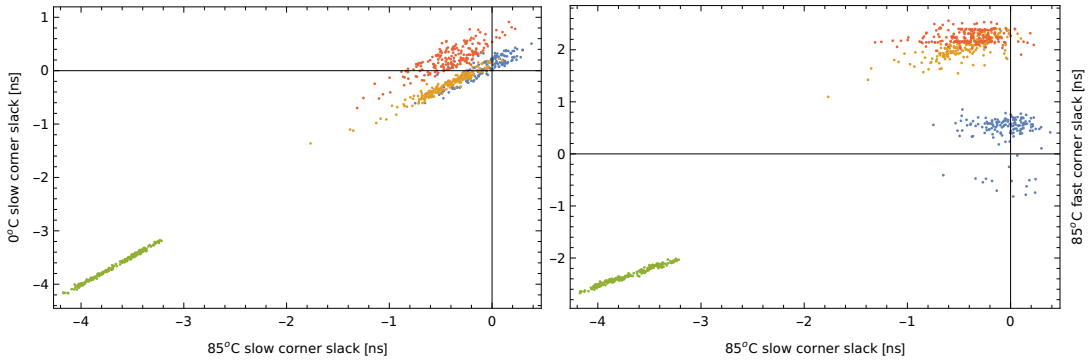


Figure 30.: Comparison of the minimal setup slack between different timing corners under seed variation for the example designs A (yellow), B (blue), C (green) and D (orange).

Higher performance at lower temperatures is an expected behaviour for silicon chips. The transistors that make up an FPGA are able to switch faster at lower temperatures since lower temperatures increase the carrier mobility in the material and consequently increase the drain currents. Therefore, the question is not why design D performs better at lower temperatures, but why the other examples do not seem to benefit as much from this effect.

Figure 31 contains a closer look into figure 30 for example B. For most of the seeds the minimal setup slack is higher for lower temperatures and the design is expected to perform better at low temperatures on average. However, a small minority of seeds leads to results where the maximal frequency at which the compiled design could be operated is actually higher at a temperature of 85°C. This is surprising since it means that less cooling of the FPGA can make it faster.

Intel states that these results are possible in section 5.3 within the design optimisation chapter of [34]: "In addition, designs targeting newer device families (with smaller process geometry) do not always present the slowest circuit performance at the highest operating temperature. The temperature at which the circuit is slowest depends on the selected device, the design, and the compilation results".

It is important to mention here that design examples A, B and C use FPGAs built with a 28 nm process [31], while example D uses an FPGA with a 55 nm process geometry [35]. The reason for the influence of process scale on the direction of the temperature dependence is a change in the supply voltage V_{GS} to threshold voltage V_{th} ratio for the transistors of newer (smaller) process geometries [36]. The drain currents depend on the difference $V_{GS} - V_{th}$ ("gate drive"), which increases with a rising temperature due to a negative temperature coefficient of V_{th} [24]. This effect becomes dominant over the change in carrier mobility at smaller process scales and leads to a reversed temperature dependence for newer devices [36]. This is likely the reason why the minimal setup slack for the 28 nm examples is less affected by temperature change compared to example D at 55 nm.

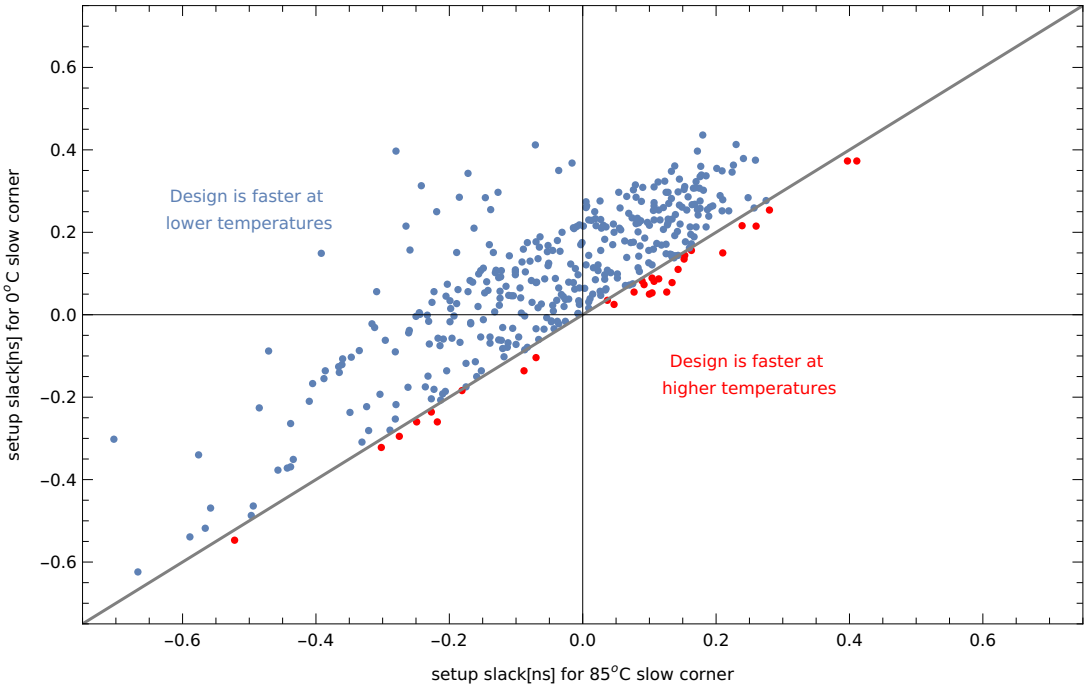


Figure 31.: Minimal setup slack results for two different temperature corners.

Hold Time

Similar comparisons can be made for the results of minimal hold times, looking now exclusively at the results of example B using the same dataset from above.

In contrast to the setup slack discussed previously, the hold time tends to be more critical for the two fast corners. This is expected since the earlier arrival of signals at their destination register will decrease the available hold time slack. For the same reason, hold time slacks on average benefit from higher temperatures (see figure 32).

Interestingly, the minimal hold time distributions for high temperatures seem to split into two separate peaks, indicating that the optimisation process ends up in separate groups of local maxima, which are somehow clearly distinguishable from each other. Figure 33 shows the correlation of the two temperature cases. With a few exceptions, the hold slacks are lower for the 0°C corner. The separation of the 85°C slack peaks becomes larger for lower hold slacks in a 0°C corner. This could be an artefact of the optimisation process or a hint that some minimal hold slack paths are differently affected by temperature than others.

If the distribution would significantly reach into negative values, it could be useful for the design process to analyse the logic paths which usually make up the left and right peaks. However, this was not considered here since the hold slacks are positive, and no correlations between these minimal hold slack peaks and the setup slack could be found in this specific dataset.

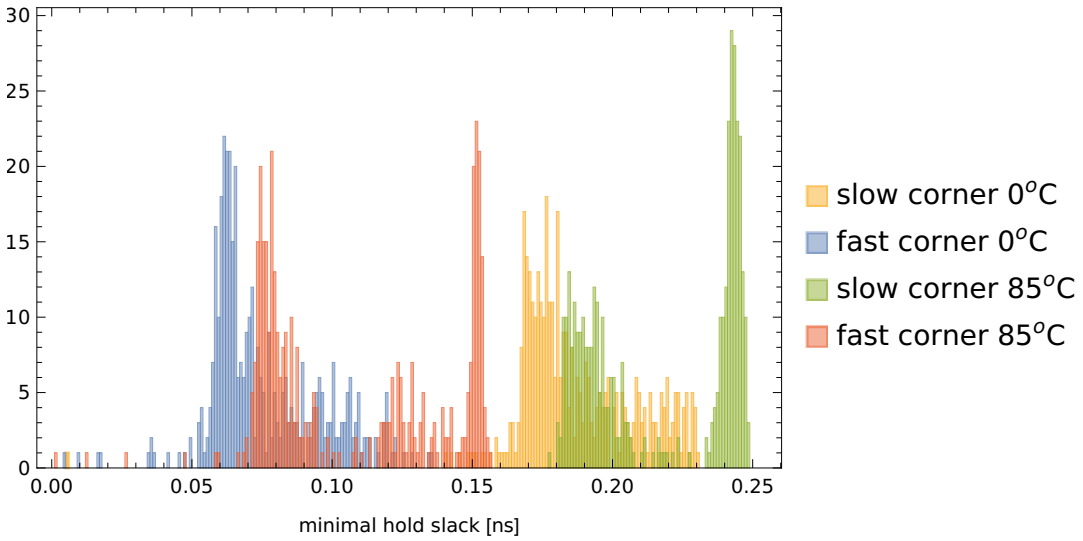


Figure 32.: Minimal hold time distributions for all four timing corners using example design B.

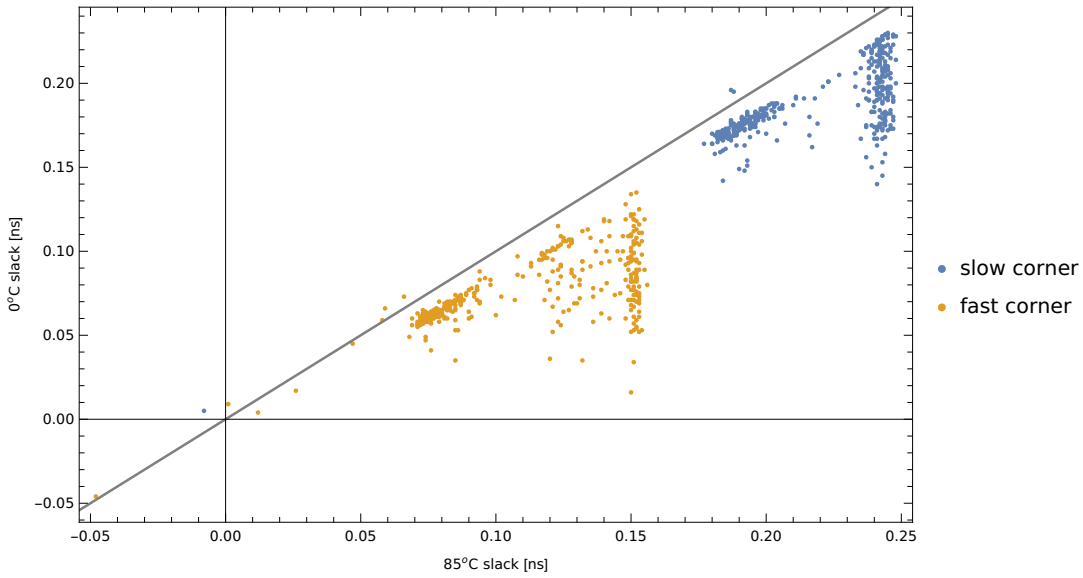


Figure 33.: Minimal hold slack results for two different temperature corners.

To summarise the topic of seed variation, it is a useful tool to achieve timing closure by repeating compilations where the resulting slacks are only slightly negative. It is also useful for investigating timing problems since it gives statistical information about the expected outcome of a compilation rather than the result from a single sample. As seen before, the spread of possible outcomes can be quite significant. It is possible to utilise seed variation in a way that frees up other resources since not every seed

is required to achieve timing closure. This comes, however, at the cost of computing time, and a tradeoff between these resources needs to be made.

In some cases, structures might arise in the slack distributions, which could contain usable information about the timing issues in the design. The four timing analysis corners exist for a reason, a design can fail them independently, and it is not sufficient to check only the corner which is suspected to show the most critical slack values. Furthermore, it cannot be assumed that the best performance is present at the lowest operation temperature since the optimal value will depend on the design and the seed used for the compilation.

3.7.4. Clock Domain Transitions

FPGA designs will often contain more than one clock. The reason can be that interfaces to other components run at different speeds and must be driven by different clocks. Also, parts of the design itself might be written for lower frequencies and could be unable to operate at higher ones, which might be required to achieve bandwidth targets in other parts of the design.

Transitions of signals between different clocks are called clock domain transitions or clock domain crossings (CDCs). For two unrelated clocks, it is impossible to enforce timing constraints between their two domains. Even if the frequencies of the clocks are nominally the same, the two oscillators driving them will have slight frequency variations and the resulting clocks will never be identical and the position of their edges relative to each other will not be constant. Therefore, the timing requirements for paths between them cannot be ensured and will be violated for at least one path. This is a fundamental issue that cannot be avoided, and CDC methods can only minimise the risk.

The risks here are data corruption and metastability issues. A single-bit signal between two clock domains transitioning from 0 to 1 will, at some point, also transition from 0 to 1 in the destination clock domain. The exact cycle where that happens is not predictable and the sampling register might enter a metastable state. To mitigate the risk of that metastable state propagating and causing further issues, additional registers can be placed as a shift register before the signal is used in the destination clock domain. This is called a synchronisation chain and reduces the chances of metastable outputs, since all registers in the chain would have to enter a metastable state in order to violate timing at the end of the chain. This is unlikely and the chances depend on the number of registers. The tools will detect these structures and calculate a value called Mean Time Between Failures (MTBF) for the design based on the failure probabilities of these synchronisers. The exact cycle where the transition happens in the destination clock domain remains unpredictable.

Since the cycle of the transition remains unpredictable, multiple-bit signals are at risk of data corruption. A 32-bit word D1 transitioning to the word D2 could be sampled in the destination domain as the sequence D1, D3, D2, where each bit of D3 consists of a random choice between the according bits in D1 and D2. This is possible since each of the 32 bits will have a separate path in the FPGA, which will all differ in

their exact delay to the destination. The relative phase of the clock there is unknown, so a bit might be sampled before or after the transition at random.

There are options to avoid data corruption. A constant multiple-bit word can be safely sampled by other clock domains. Therefore, a 32-bit register can be set to the value that should be transmitted, then waiting for an appropriate time can ensure that change has arrived at all destinations[12] and then a single-bit signal can be sent into the new domain using the method above to indicate that the register can now be sampled. Another single-bit signal would have to be sent back to indicate that the register was now sampled and the next data word can be applied. This handshake operation limits the bandwidth but avoids data corruption.

If the bandwidth should not be limited, a dual clock FIFO or RAM can be used. Data is written by one clock domain and read by the other one. For the internal control signals, handshakes are used, but full bandwidth is available for the user. The memory or FIFO can be small depending on the application but will need resources. Since memory comes in blocks, this is one situation where the MLAB cells from section 3.6.1 are useful since they allow a smaller memory partitioning.

Two unrelated clocks with nominally identical frequency will slightly differ in their actual frequency. If data has to be transmitted every cycle between two 100 MHz clocks then one direction will cause overflows in the synchronisation logic, since one of the clocks will run slightly faster and the other one cannot process the additional data. A form of backpressure needs to be in place in order to avoid data loss. The same is the case if the destination domain runs on a slower clock than the source.

CDCs between related Clocks

Until now, the discussion was about transitions between unrelated clocks. If the clocks are related, the question is if the phase relationship between the two domains is known at compile-time. If not, the CDC has to be treated similarly to unrelated clocks. If it is known, then timing can, in theory, be enforced, but the requirements can become stricter compared to transitions within the same domain. An examle is shown in figure 34: For transitions from $clock1$ to $clock3$, timing requirements are similar to transitions within $clock3$ since the distance from a rising edge of $clock1$ to the next rising edge of $clock3$ is always identical to a cycle of $clock3$. However, CDCs between related clocks can become arbitrarily complicated. Sending data from $clock2$ to $clock3$ has three possible relationships (a, b and c) towards the next rising edge. All of these possibilities need to fulfil timing requirements. At the point where this becomes too complicated, it is easier to treat the clocks as unrelated and deploy the CDC-methods from above.

Considering that CDCs always come with further complications, additional clock domains should only be introduced when necessary. Organising the design in functional blocks and a clear separation of the clock domains is needed in order to minimise the connection points and, thus, the amount of CDCs needed between them. This is

[12] This implies an assumption of how long that will take.

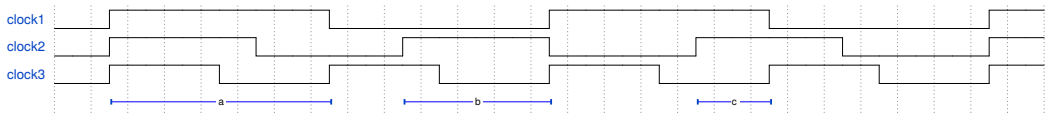


Figure 34.: Example of related clocks.

especially important since simulations cannot properly model unrelated clocks. Therefore, CDCs are an area where the result of simulations of a design can differ from the behaviour in actual hardware. For this reason, clock domains should be considered during the planning of the design of the firmware structure in order to make the necessary transitions in places where they cause the least amount of issues.

Ignoring CDCs

In some situations, CDCs can be ignored for specific signals. For example, before the Mu3e detector takes data, some settings have to be applied in the firmware, which are then constant during data taking. Such signals can assumed to be constant and can be ignored for CDCs and timing analysis. It is also possible to encode signals in a way to make ignoring CDCs viable. This is done via gray encoding, which is a different way to represent binary numbers, where adjacent numbers always only differ by a single bit. With this encoding data corruption from an ignored CDC will only result in an off-by-one error (assuming that the skew between all bits is less than the clock period). This is acceptable for many applications, such as counters, temperature measurements and others.

Since the tools usually assume all clocks to be related, they need to be explicitly instructed to ignore timing constraints for paths leading to a synchronisation chain or paths which are supposed to be ignored. Such paths are called false paths and, generally, all paths between unrelated clock domains should be flagged as such[13]. However, false path assignments can also be used within the same clock domain. Doing so is technically not accurate but can be used to remove timing issues, for example on a signal that is practically constant and has a very large fanout. Situations can exist where ignoring the timing of that signal is a safe approach.

During this thesis, we will encounter a CDC situation in section 4.4.2 where the unpredictability of the clock cycle where the signal is sampled in the destination domain is unacceptable. All methods shown here inherently include this unpredictability. Avoiding it is challenging but possible in specific situations. This will be discussed in section 4.4.2 for the specific design challenge in the Mu3e readout system.

[13] If they are not flagged as such, the tools will try to achieve timing closure on them and often fail to do so. It can be useful to intentionally off-tune clock frequencies in the timing constraints slightly (e.g., from 50 MHz to 49.99 MHz), which will ensure the tools to fail on these specific paths, which in turn allows to generate a list of paths between domains which are not set as false.

3.7.5. Resets

With the application of a reset signal, a digital system is moved into a defined state. This is used either as initialisation after the power-up of the device, in order to recover from a state of error or to restart some procedure. The usual purpose of the reset here is not to bring something to a defined state, but to start something from a defined state. Therefore, the release of the reset is the actually relevant part. The timing requirements (recovery and removal time) were discussed in section 3.3.1.3.

One can, of course, also construct some cases where not the removal, but the application of a reset is essential as well. We will not encounter such cases so they will not be further discussed here.

Logic cells in the FPGA, IPs and most VHDL entities will provide a reset input port. Reset inputs can be either synchronous or asynchronous inputs. Sometimes, both will be provided. The reason resets are discussed in this section in detail is because there are common misconceptions about the meaning of asynchronous resets, when resets are necessary and when not and their relationship to default values.

Synchronous resets behave similarly to all other signals in a design. Even though the LAB cells of the FPGAs used in this thesis provide separate inputs for synchronous resets, they are not used as such just because the HDL source code states it. Resets will be implemented by the tools in whatever way that efficiently implements the behaviour stated by the HDL source code. This might be done via dedicated reset pins or the normal data input path. Previously in this chapter, a schematic of an ALM cell was shown (figure 25). A more detailed view reveals that the synchronous reset port is just connected with the data input before the register using logic of the form $reg = (data\_in \ \& \ !\text{SCLR})$. The synchronous reset (SCLR) is, therefore, just part of the datapath. A detailed schematic can be found in appendix A.2.

Asynchronous resets force the tools to use the asynchronous input of the logic cell. Asynchronous means that the reset is able to change the output of the register without the presence of a clock edge. It does not mean that there are no timing requirements relative to clock edges should they be present. That is a very important difference since the consequence is that truly asynchronous resets are only valid if no clocks are present. With the presence of clocks, asynchronous resets have to follow the requirements for recovery and removal time.

This also applies to IP blocks. If an IP presents an "async reset" port to the user, it means that setup and hold time requirements do not apply to this port. This does not include recovery and removal time, which still apply in this case and – if the rules are followed strictly – forbid the use of a true asynchronous reset.

In order to avoid this problem, asynchronous resets can be synchronised into the clock domain of their target, which effectively turns them into synchronised resets and allows the tools to time them correctly. This can be done individually for each clock domain.

Only a few cases will actually produce reset signals that are not synchronous with any clock. This includes a human pushing a button or turning on the FPGA. How to cleanly turn on an FPGA is a difficult question since many things which are granted

during normal operation might not be in this case. Clocks could be unstable, power distribution across the FPGA might not be stable or clocks might be completely missing and registers could be in random states. How designs in Mu3e manoeuvre out of this situation will be discussed in 4.11.1.

Not using Resets

Resets are not necessary for every signal in a design. If a signal is not reset then it might be in an unknown state. If the logic does not require the signal to be in a known state from the beginning, then this is not an issue.

However, leaving out unnecessary resets does not immediately save resources. Not resetting signals within the typical $if(reset =$'1'$)..elsif(rising\_edge(clk))$ statement will lead to usage of the reset as clock enable input of the resulting registers since the signal needs to be kept constant during reset.

There are cases where using a reset is a mistake. Sometimes, using a reset can alter the design significantly in unintended ways. For example a piece of HDL code that describes memory where – without a reset – that memory is going to be efficiently implemented in M10K blocks of the FGPA via inference. If a reset is used to explicitly set the memory content, an implementation using M10K blocks is not possible anymore since they do not provide a way to set their entire content to a specific value with one operation[14]. In this case the synthesis will implement something fundamentally different based on normal logic cells and waste resources.

3.7.6. Trading between Resources

Various elements of an FPGA or properties of a design can be considered a resource. This includes the actual hardware resources of the FGPA, such as logic cells, memory blocks, routing lines and hard-IPs but also attributes of the design like clock frequency, bandwidth, latency, available slack or compile-time.

Trading between these resources is an essential part of FGPA firmware design. If the bandwidth of some process has to be increased that can either be done by increasing the frequency and spending available slack on the problem or, for example, by spending more logic resources in order to parallelise the process. If additional slack is not available in the design, a higher compile time might be accepted to find seeds with better minimal slack values. Another option would be to pipeline the process into multiple steps and increasing the latency to allow for a higher frequency and, therefore, bandwidth. It might also be possible to use different hardware resources (Memory or DSP blocks) to improve the process. These resources could already be in use, and other parts of the design might need to be changed in order to make them available.

[14] A default value in the HDL source code can be used to achived the same behaviour on power-up without a reset. Default values for logic cells and memory blocks are synthesisable in modern tools [37]. This is only applicable to power-up conditions.

Achieving timing closure is just a result of available slack. Similar to increasing bandwidth, available slack can be increased by spending other resources on it. The example that was made before was pipelining a kind of calculation into multiple operations processed in separate clock cycles. But the idea of pipelining applies also to topics regarding design structure.

The following example is actually implemented within Mu3e. The shown setup slack comparisons are the results of seed variations on modified versions of Mu3e firmware. Specific details and reasons for the chosen implementation will follow in a later chapter[15].

Consider an example where a set of $4^{j_{max}}$ entities, which are scattered randomly all over the design, is supposed to be accessed by a single controller entity. The amount of connections needed to do so is too large for the desired frequency and the minimal setup slack histogram shows a very low probability for timing closure (blue histogram in figure 36).

Assume we have a tree-like structure with multiple levels $j \in \{0; 1; 2; ... \ j_{max}\}$. Each level of the tree consists of 4^j individual nodes $N(i, j)$, starting from a root node $N(0, 0)$ and ending at the entities $N(0...4^j - 1, j_{max})$ that are supposed to be accessed. Every node $N(i, j)$ connects to 4 nodes on the next level within the tree $N(4i...4i+3, j+1)$. A node is now registering the control information from the upper level in four registers and connects these four registers to the four following nodes, which creates the structure in figure 35 and still provides a connection between the controller entity at the root of the tree and the $4^{j_{max}}$ entities at the end.

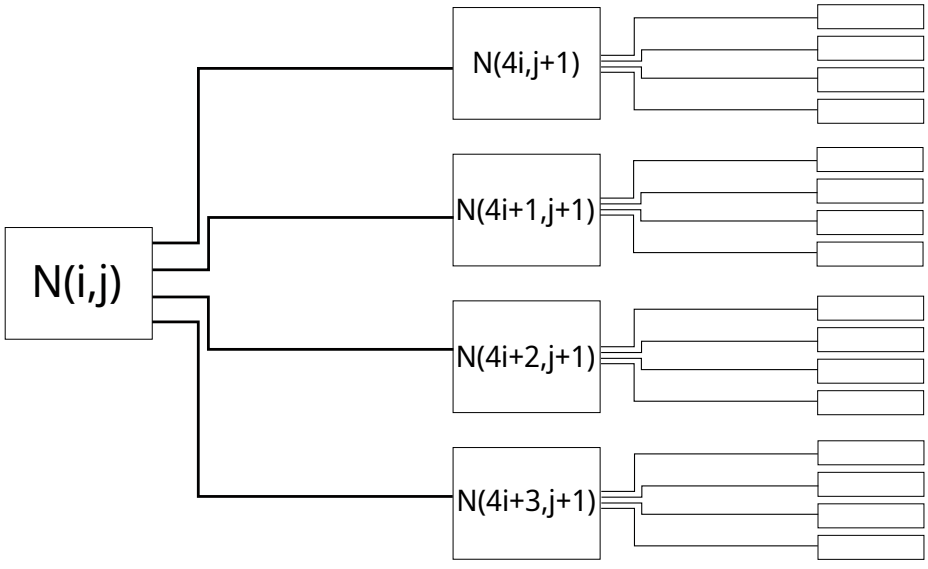


Figure 35.: Tree structure with each node connecting to four nodes on the lower level of the tree.

[15] In section 4.4.4, the section will refer to this example again.

This introduces a latency of j_{max} clock cycles for the control information to reach the destination but limits the fanout to four within the tree since only four copies of the signal have to be created at each node. Accepting a higher access latency and logic usage has been traded for better timing performance. The results can be seen in figure 36. The mean minimal setup slack has improved by 0.24 ns and the timing closure probability has increased from 1 % to 48 %[16]. An alternative, slightly different implementation of the idea can be found in appendix A.3.

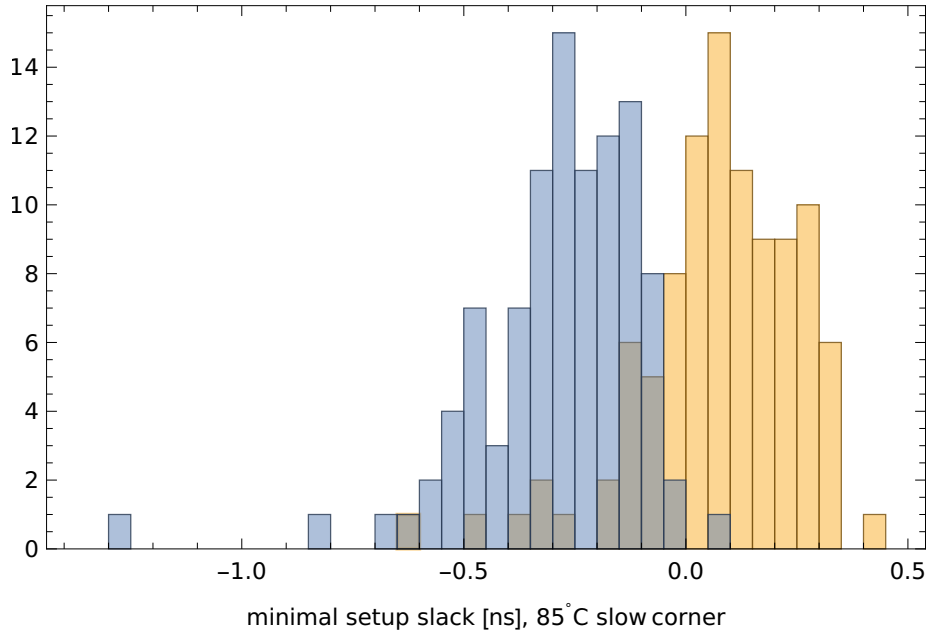


Figure 36.: Minimal setup slack histograms for the version without added latency (blue) and the reference measurement (yellow)

The extent of the improvement depends on the other parts of the design. For the blue histogram, the majority of the minimal setup slacks came from paths related to the the controller entity. After the introduction of the latency the controller entity is not the most timing-critical part of the design anymore, and the minimal slack histogram is driven by other design parts.

This was one example of a resource trade in order to achieve timing closure. The same effect could have been accomplished by sacrificing bandwidth or generally lowering the clock frequency. Alternatives will be discussed when the actual issue is explained in the appropriate part of the thesis.

Apart from closing timing, another thing which was realised here is structurally decoupling the entities from each other. As a result of the tree-structure the end of

[16]The design which includes this improvement is a version of Mu3e firmware and was already used as example B during the discussion of seed variation. It was modified in order to generate the dataset without the latency tradeoff.

the tree should now be able to spread more easily across the design. However, this is more difficult to quantify than slack improvements.

3.8. Data transmission

Fast data transmission is essential to modern data acquisition systems in particle physics. The aim for increasingly precise measurements comes with an increase in the required statistics, leading to new challenges for data acquisition systems to process the acquired data.

Processing data does not just include analysing it. Moving data out of the detector to the places where it can be analysed comes with its own challenges, such as the required bandwidth, spatial constraints or the operating conditions on the inside of particle detectors.

Moving data is not an exclusive problem of particle physics, and many of the techniques, encodings, and protocols that we will encounter are standardised. This section will discuss the parts of these topics that are used within the Mu3e data acquisition system.

3.8.1. Challenges for Data transmission

The quality of a digital signal can be degraded by different factors. The signal rise time, which is essential for high-speed signals, can be limited by capacitive loads on a cable if the source is not able to recharge the cable fast enough. They can also cause signal reflections, which can be described by the equation [38].

$$R(\omega) = \frac{Z_L(\omega) - Z_0(\omega)}{Z_L(\omega) + Z_0(\omega)} \tag{8}$$

Where $R(\omega)$ is the fraction of the signal that is reflected back towards the source, $Z_0(\omega)$ is the characteristic impedance of the transmission line and $Z_L(\omega)$ is the load impedance.

In order to avoid degradation of the signal integrity, the characteristic impedance of the transmission cable needs to match the load impedance at the termination of the line. This is a question of cable geometry and material or, in the case of Printed Circuit Board (PCB) traces, trace width and distance to the ground plane [38].

Unintended capacitive or magnetic coupling between transmission lines can lead to an effect called crosstalk, where a signal from a transmission line is picked up by a neighbouring connection. Similar to reflections, this can negatively impact signal quality and cause transmission errors.

When sending digital signals over large distances, signal attenuation due to the resistivity of the conductor has to be taken into account. A frequency-dependent attenuation is introduced by the skin effect, which is explained in figure 37.

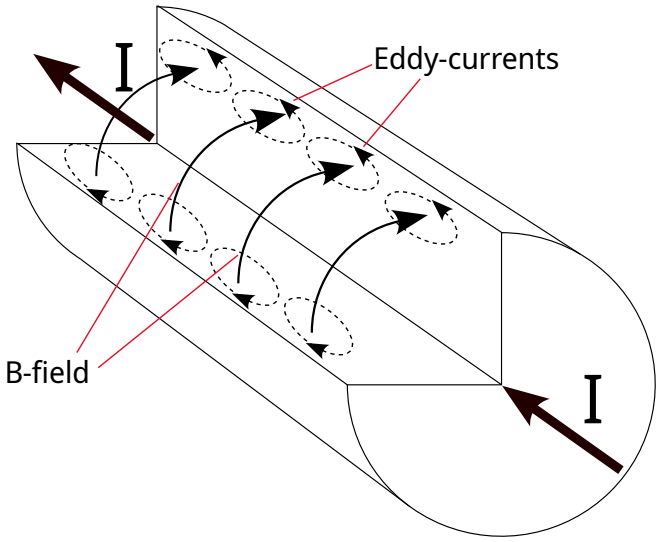


Figure 37.: Skin effect created by eddy currents in a conduction wire.
Figure was adapted from [39].

A magnetic field is created by the signal going through the conduction wire.

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{J} + \mu_0 \epsilon_0 \frac{\partial \mathbf{E}}{\partial t} \tag{9}$$

As the signal transmission proceeds, the magnetic field will alternate and cause eddy-currents in the material.

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{10}$$

These induced currents will add up to an increase in the overall current close to the surface and a decrease closer to the core of the conductor. The effective cross-section, which is used to transmit current through the conductor, is decreased and the result of this is an apparent increase in resistivity [39]. The eddy currents depend on the frequency at which the signal changes. The increase in resistivity consequently also depends on the frequency with the resulting effect of a frequency-dependent attenuation. For this reason, the distance over which signals can be transmitted through a conductor depends on their frequency and will decrease for faster signals.

3.8.2. Signal Standards

Data can be transmitted with a serial or parallel connection. Serial data transmission is using a single transmission line and sends the bits of each data-word consecutively. The word has to be serialised at the transmitter and deserialised at the receiver. When a sequence of words is sent, a form of protocol has to be in place in order to identify the word boundaries for deserialisation at the receiver.

Parallel data transmission uses multiple lines instead and can send multiple bits of the same data-word at the same time. The disadvantage of parallel connections is

the need for more physical wires and possible delay variations between them. Within an FPGA these variations can be precisely calculated and do not cause a problem. However, for inter-device communication, delay variations will create issues once they reach a magnitude similar to a clock period of the connection. Within the Mu3e data acquisition, serial connections are used for most high-speed signals.

Electrical data connections in the Mu3e DAQ will use the Low Voltage Differential Signaling (LVDS) signal standard. LVDS is a differential signalling standard where the transmitter sinks or drives a specified amount of current from both lines. The current is then converted into a voltage via a resistor between the differential pair at the receiver end of the line.

Signal distortions that appear on both differential lines identically cancel each other since the voltage drop over the resistor at the receiver remains unchanged. For this reason, LVDS signalling has the advantage of suppressing common-mode interference and ground noise [24]. It is also, to some extend, robust against differences in ground potentials between source and destination, which represent a problem for non-differential voltage-based signal standards.

3.8.3. Protocols and Encodings

Standardised data protocols specify the rules for the exchange of data between transmitter and receiver. Often, multiple of these protocols and encodings will be stacked upon each other in layers that serve different purposes. A very prominent example is internet communication: If an HTTP transaction is supposed to take place, the data is transmitted via a TCP connection. TCP forms a connection for the exchange and adds control information in packet headers, which contain information about source and destination ports and ways to acknowledge and request retransmission of data. The TCP data is then again enclosed in an IPv4 packet, specifying source and destination IP address and general routing information, followed by another enclosure that contains information about the MAC-addresses. Finally, the entire packet is encoded in a way that allows for the detection of transmission errors before being serialised and physically sent over a connection that follows some signal standard.

The Mu3e readout system defines Mu3e-specific protocols. Those will be discussed within the sections where they are used. Standardised protocols and encodings used in the Mu3e DAQ will be discussed here.

Gray Code

The reasons for using Gray-encoding were already discussed in the clock domain transition section 3.7.4. The objective is to provide a different representation of binary numbers such that adjacent numbers only differ by a single bit in their gray-encoded binary representation. As discussed in section 3.7.4, this is useful for clock domain crossings with parallel data.

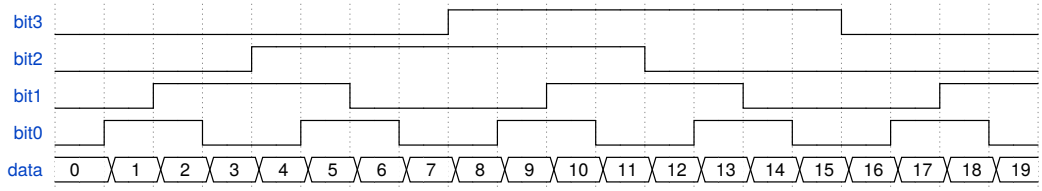


Figure 38.: Wave diagram showing the lowest four bits of a gray encoded counter.

8b/10b Encoding

8b10b encoding uses 10-bit words to represent 8-bit words. The additional two bits are used to accomplish a few beneficial properties for the serial transmission of the encoded data. Since the number of available codes is four times higher than the number of possible data words, a DC balance can be enforced by alternating disparity (difference between 0's and 1's). Multiple codes can exist to represent the same 8-bit word and they are chosen in a way to ensure DC balance, which is required to be limited to ± 2 bits at all times. This is desirable since, on average, no current is transmitted over an 8b10b encoded connection [40].

In addition to this, all codes containing six or more sequential 1s or 0s are forbidden. The resulting frequent logic transitions allow to recover the transmitter clock from the serial data stream. This enables the transmission of data without a separate line for the clock since the receiver can recover it from the data.

With these requirements, it is also possible to detect transmission errors since some codes are invalid and some combinations would violate disparity requirements. All single bit-flip errors can be detected, which can be used to gain information about link quality. Out of the 2^{10} possibilities, 12 codes remain after the restrictions above are implemented. They exist in a positive and negative disparity and are shown in table 3.

Name	Hex-Value	10b code with positive disparity
K.28.0	1C	1100001011
K.28.1	3C	1100000110
K.28.2	5C	1100001010
K.28.3	7C	1100001100
K.28.4	9C	1100001101
K.28.5	BC	1100000101
K.28.6	DC	1100001001
K.28.7	FC	1100000111
K.23.7	F7	0001010111
K.27.7	FB	0010010111
K.29.7	FD	0100010111
K.30.7	FE	1000010111

Table 3.: The 12 8b10b comma symbols [40].

These 12 remaining codes are called comma symbols or comma words and can be used for flow and link control commands. Within Mu3e, for example, they will be used to indicate the start and end of packets or runs.

The sequence 1100000, which is present in the symbols K.28.1 and K.28.5, is a unique sequence of bits that cannot otherwise appear at any point in the serial data. Therefore, this sequence becomes an important tool since it is the only possibility for the receiver to unambiguously recover the word boundaries, which is necessary to decode the 8b10b datastream back into the correct sequence of 8-bit words. As a consequence, K.28.1 or K.28.5 have to actually appear in the datastream at some point in order to allow the receiver to perform this alignment of word boundaries.

Cyclic Redundancy Check (CRC)

Another method for error detection are CRC codes. They are a fixed-length binary number which is calculated for a data packet and appended to it. The receiver can then use the same method on the received data, calculate the CRC code again and verify the data by comparing it to the CRC received from the transmitter. If they do not match, it indicates that an error has occurred during transmission. The calculation of CRC codes can be implemented in an FGPA and will be used in the Mu3e readout system error detection. While CRCs are highly effective in detecting errors, they do not provide error correction. The theory behind CRC and error-correcting codes can be found in [41].

SPI and I2C

Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I2C) are two serial communication protocols intended for short-distance communication, often to communicate at a low speed with periphery components on the same PCB.

SPI consists of an SPI-master, which is the entity controlling the interface, and one or more SPI-slaves, which are the entities controlled by the master. Four data lines are needed to make a full SPI connection: MOSI (Master Out Slave In), MISO (Master In Slave Out), SCLK (clock) and CS (Chip Select).

An SPI-slave can be implemented as a simple shift register, with MOSI connected to the input of the first register and MISO connected to the last register in the chain. The clock is per default inactive and only sends clock edges for every bit in MOSI that has to be transmitted. The chip select signal is needed to identify the device and acts either as an enable signal in the shift register or as a load signal that copies the content of the shift register into the register that actually controls the device. A chip select line is, therefore, needed for each slave connected to a master. MOSI can be shared and is only needed once, and MISO can be left out if data reading is not required.

I2C serves a similar purpose but is address-based. It only requires the two lines SDL (serial data line) and SCL (serial clock line). An I2C device is targeted by transmitting a start bit, the address of the device, followed by a sequence of data and a stop bit

over the SDL line. Similar to SPI, the clock is per default inactive and only activates during data transmission. The I2C data line can be driven by the master and the slave, so the same line that is used to send data to a device is also used for replies and acknowledge signals. The clock line is always driven by the master [24].

SPI is simpler to implement in hardware, especially on the slave side, since it only requires a shift register, but I2C is more flexible and requires fewer physical lines to the target. Mu3e will use both protocols within the readout system.

3.8.4. Avalon and AXI

Some protocols are specifically designed for on-chip signal transmission. On an FPGA those can be used to interconnect different design components or hard-IPs. A few of the intel-IPs that will be used during this thesis on FPGAs use the proprietary Avalon interface [42]. For Xilinx FPGAs the equivalent is called AXI, which is developed by ARM [43].

Avalon interfaces come in multiple variants which serve various use cases. The only one relevant for this work is the Avalon memory mapped interface (Avalon-MM), which provides address-based read/write transactions between the connected components. AXI and the other Avalon variants will not be discussed here. The section will only introduce the specific configuration of Avalon-MM that appears to a limited extent in the Mu3e data acquisition. The majority of the address based read/write connections in Mu3e are not based on Avalon and the following introduction is important to understand the decision against Avalon-MM in a later chapter.

The signals shown in figure 39 are needed for a minimal avalon-MM interface for a read and write transfer between host (master) and agent (slave).

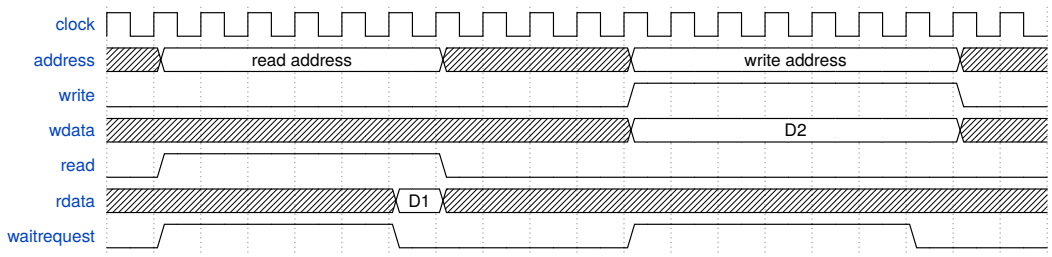


Figure 39.: Example wave diagram of an Avalon read and write transfer.

To issue a read, the host applies the read address and read signals. The agent has two options to respond: Either it has to immediately raise the waitrequest flag or provide the data D1 for that address at the rdata line. The waitrequest can be used to stall the host until the agent is able to provide the required data. The host is required to hold the read address and read signal until the waitrequest signal is released, at which point rdata is sampled and the next action can occur.

A write transaction has a similar structure. The host applies the address, write signal and write data, and has to hold those until the waitrequest signal from the agent is de-asserted.

With the signals provided here, a transaction speed of one read/write per clock cycle is only possible if no waitrequest signal is raised during the process. In order to provide an option for pipelined reads and writes, additional optional signals and settings can be used. The *waitrequestAllowance* setting allows the host to proceed with transactions for a specified amount of cycles, even if the waitrequest was asserted by the agent. When this is used, the agent needs to be able to manage this situation. In combination with this setting, a signal called *readdatavalid* can be used to provide a way for the agent to send data back to the host without de-asserting the waitrequest signal.

System integrator tools are available, which connect multiple hosts and agents within the same interface and automatically assign address offsets for the agent addresses. An example for such a system with multiple hosts and agents is shown in figure 40.

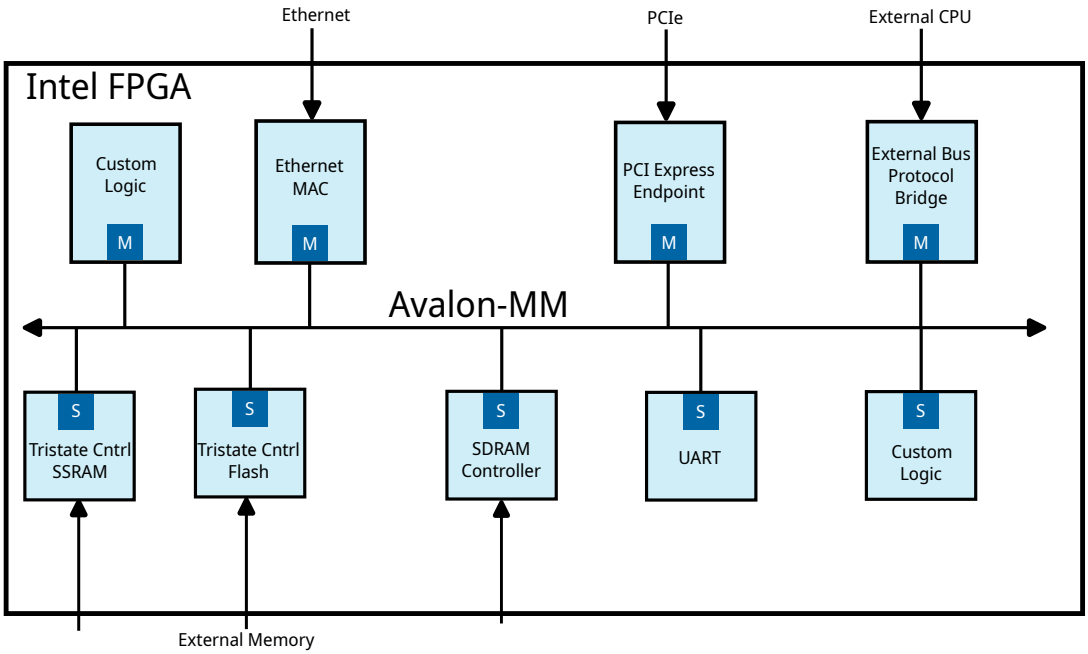


Figure 40.: Example for an Avalon-MM Interface connected to multiple hosts (M) and agents (S). Graphic was adapted from [42].

In these systems, a method of host-agent arbitration becomes necessary, and the hosts that lose the arbitration process will be stalled via the waitrequest signal until a transmission for them is possible. Host/agent arbitration is, in these cases, implemented by the system integrator tool and not exposed to the user. Therefore, latencies can be unpredictable: When the custom logic host in the top left corner of figure 40 wants to read from one of the agents, then the latency of the response depends on the state of that particular agent and the read/write processes of the other hosts in the system.

3.8.5. Transceivers

Transceivers are components used for receiving and transmitting serial data at high rates (above 1 Gbit/s). On intel FPGAs, transceivers are implemented as hard-IP cores. These cores are hard-wired to specific pins of the FPGA, meaning that the serial input and output signals cannot be routed freely through the FPGA's user logic. In principle, every IO-pin of an FPGA can be used to transmit a serial data stream by connecting an according logic in the firmware of the FPGA, but only hard-wired transceivers are able to achieve bandwidths in the order of 10 Gbit/s.

The hard-wired IPs location also constrains the point at which the parallel data has to be provided to the IP. For the FPGAs used in this thesis, this is usually one of the sides of the chip.

Intel transceivers are divided into two main blocks: The Physical Medium Attachment (PMA) and the Physical Coding Sublayer (PCS). The following explanations of these blocks are based on reference [44].

PMA Block

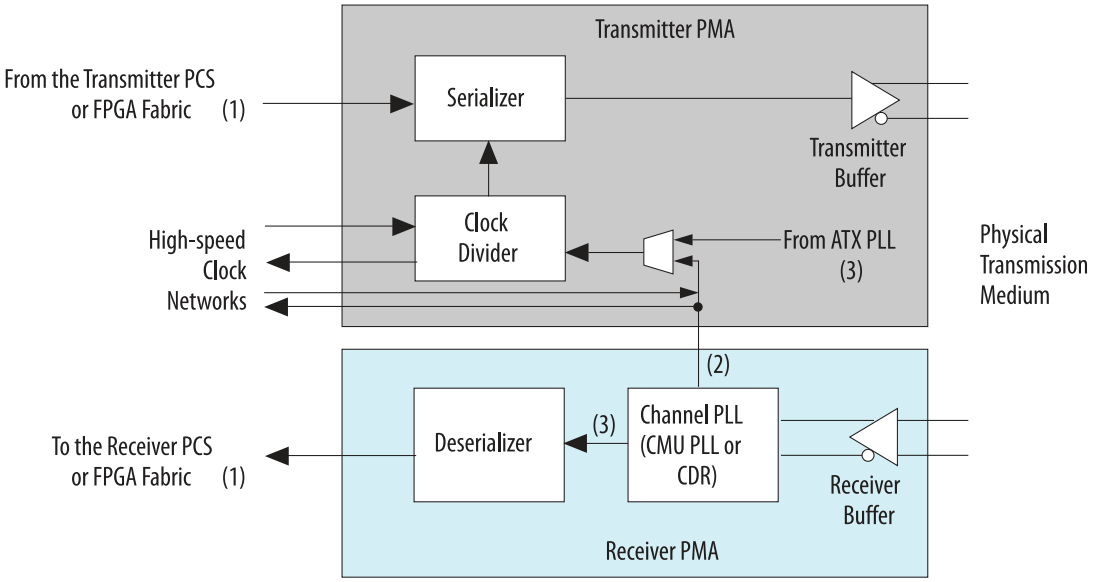


Figure 41.: Diagram of a PMA block of an ArriaV transceiver [44]

The physical medium attachment (PMA) block of the transceiver connects directly to the physical output or input pin of the FPGA. Incoming data is fed into the channel PLL, which is located in the transceiver PMA. This PLL recovers a synchronised clock from the data stream at the serial bit frequency if enough transitions are present. This can be ensured by the previously discussed 8b10b encoding scheme or similar methods.

The recovered clock is then utilised by the deserialiser to convert the serial data into a parallel data signal, which is then forwarded to the physical coding sublayer (PCS)

block. Initially, the PMA aligns the parallel data word randomly[17]. The recovered clock is also sent to the transmitter PMA, where it is divided to match the frequency of the parallel data. This clock also drives the serialiser in the transmitter PMA in certain configurations.

PCS Block

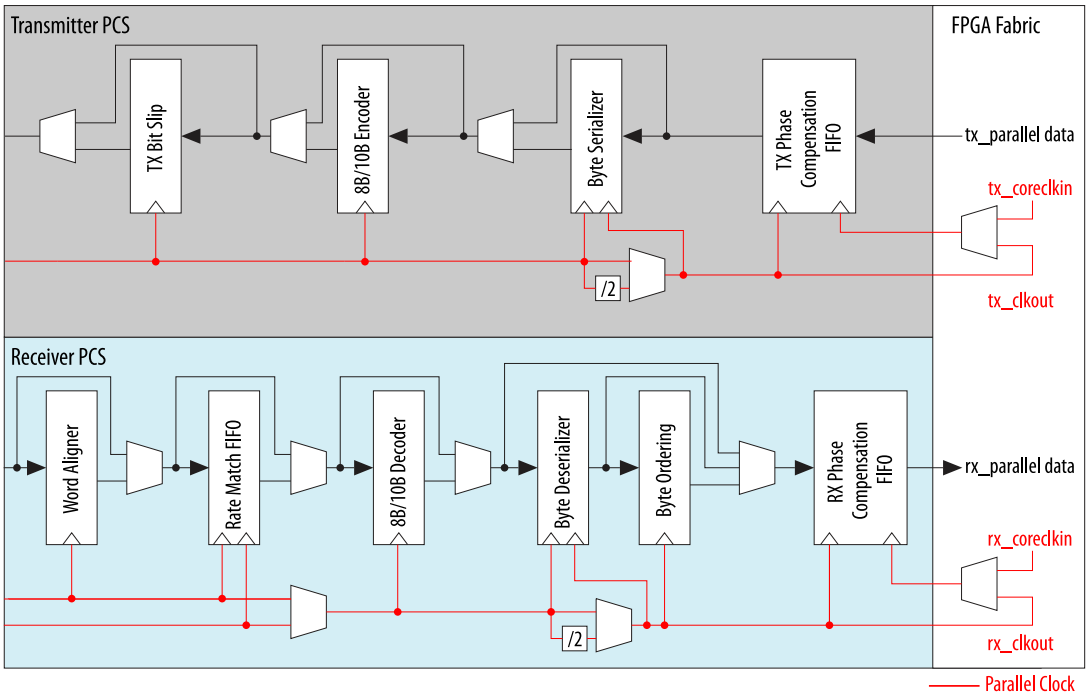


Figure 42.: Diagram of a PCS block of an ArriaV transceiver. Adapted from [44]. The left side is connected to the PMA block of the previous section.

The PCS block contains digital processing logic for the received data before it is given to the user in the FPGA fabric. Initially, data from the PMA block is word-aligned using control characters as specified in table 3, followed by 8b/10b decoding. For some applications where a very wide interface is wanted, the rate match FIFO and byte ordering blocks are needed and convert the output of the word aligner into a wider parallel signal.

In cases where the user clock is different from the recovered data clock, a phase compensation FIFO is necessary to feed data into the FPGA fabric. However, if the user clock frequency is slightly lower than that of the transmitting device, there is a risk of data loss due to the limited size of the FIFOs.

The transmitter PCS is much simpler compared to the receiver. Data from the FPGA fabric is written to a phase compensation FIFO, which is read with the divided

[17] At least for standard configurations of the transceiver. More about this will follow later.

clock from the channel PLL in the PMA. The bytes are serialised in cases with very wide interfaces and then 8b10b encoded and sent to the PMA block.

3.8.6. PCIe and DMA

Peripheral component interconnect express (PCI express) is a high-speed interface standard used for connections to different peripherals on the motherboard of a PC (graphic cards for example). Some FPGA boards can act as such a peripheral of the PC motherboard.

PCIe is, in contrast to its older version PCI, not a bus. Each component is connected to the PCIe network with a set of differential wires. Each pair of wires (receive and transmit) is called a PCIe lane. The size of a PCIe interface can vary and consists of a maximum of 16 (x16) lanes[18]. PCIe slots with more lanes are able to provide a higher bandwidth to the component. The FPGA boards used in Mu3e implement PCIe version 3.0, where each lane provides 8 Gbit/s serial data rate [45], adding up to a theoretical limit of 128 Gbit/s. Actual data transfer cannot reach this speed since it includes protocol overheads.

During the boot process of the PC, a negotiation of the capabilities of the connected PCIe devices will take place, including a potential speed reduction of faster components to older PCIe versions to ensure compatibility. Another parameter which might be negotiated is the maximal payload for PCIe packets (packet format is shown in table 4). Since PCIe is a network which can contain multiple bridges, the maximal payload will fall back to the lowest supported payload in the connection chain.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R 0	Fmt 0x2			Type 0x00				R 0	TC 0			R 0				TD 0	EP 0	Attr 0		R 0		Length 0x001										DW 0
Requester ID 0x0000																Tag (unused) 0x00												Last BE 0x0		1st BE 0xf		DW 1
Address 0x3f6bfc10																														R 0		DW 2
Data DW 0 0x12345678																																DW 3

Table 4.: Fields of an example PCIe write request. Table from [46] adapted by [47].

The FPGA designs in this thesis hardcode the PCIe version to 3 and payload to 256 bytes and will, therefore, only work if all components of the slot in the motherboard also support them. This is especially important for the payload since standard consumer motherboards will often only have one slot with the full capabilities, which is usually intended for a graphics card. A Mu3e PCIe card inserted into a slot with lower capabilities will not automatically negotiate a lower maximal payload.

Each PCIe device hosts a set of base address registers (BARs), which can be accessed by location (slot) of the PCIe device. During PCIe device enumeration, when the machine is started, the BARs will be used to gain information about the size and

[18]The standard allows up to 32 lanes but this is usually not implemented.

number of resources that the PCIe device provides. In the case of the FPGAs in Mu3e, four BARs are implemented and correspond to four memories.

After that, the OS will assign a range of addresses to each resource and inform the PCIe device of this decision by writing the starting address of this range into the corresponding base address register [46] [48]. „Base address“ is in this case just a different name for the starting address of a PCIe device's memory resource. The addresses following the base address correspond to the content of the memory up to the size previously requested in the enumeration process.

Physically available memory within a computer system is scattered across components like multiple different RAMs or a Disk/SSD. The host computer's memory management system will map these available resources into a continuous virtual memory, which will be used by programs ([23], Figure 43).

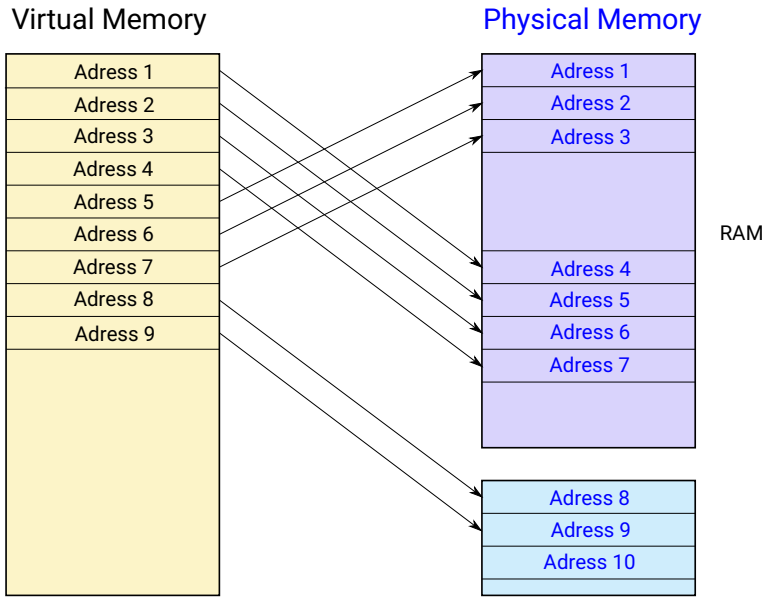


Figure 43.: Address mapping between virtual memory (used by programs) and physical memory. Figure from [23]

The same applies to the memory of PCIe devices. The memory corresponding to the base address registers of a PCIe device can, therefore, be accessed by software. The four BARs on the FPGAs in Mu3e are used for read/write registers and read/write memory. This is just an organisational decision and defines the direction of data flow. Read memory and read registers are only read from the PC's side and written from the FPGA. Write memory and write registers are only read from the FPGA and written by the software on the PC. From the PCIe point of view, there is no difference between these four resources.

DMA

In the section above, it was explained how software running on the CPU of the PC can write into memory that is physically located on an FPGA connected via PCIe. The FPGA's memory was assigned a starting address and size, which can be accessed by the CPU by sending PCIe read/write commands.

Direct Memory Access (DMA) is, in principle, the same process in reverse. PCIe is a network that connects CPU, RAM, GPU, and other components and provides the CPU with a way to access the RAM. The FPGA is now also a component in the PCIe network and, as such, can access the RAM. The issue is that under normal circumstances, the CPU is the only component accessing the RAM, and the memory management systems of the OS do not expect other actors.

This point has to be solved by the DMA-driver. Locations within the CPU's memory have to be reserved, and they need to be communicated to the FPGA's PCIe firmware. An issue that arises here is that the FPGA does not have the resources to store infinitely many of these locations. If the free areas within the CPU's memory are too scattered, it might be difficult to find large enough free continuous address spaces. A reboot of the computer can be required in these cases.

If the memory for the DMA can be reserved, the FPGA can send PCIe packets and write to these memory addresses. The CPU will need to copy this data into other locations to make space for new data transfers. Some organisational questions arise here about how exactly this can be used to process and analyse a continuous datastream generated by a particle detector, but these questions will be answered in a later chapter.

A PCIe driver on the PC, a DMA engine and PCIe control block on the FPGA were implemented by [23] and other members of the Mu3e collaboration for the use in the Mu3e experiment.

3.8.7. Fibre optic data transmission

The high data rates produced by modern particle physics experiments require data transmission methods with accordingly high bandwidth. For short distances, LVDS connections or systems like PCIe are sufficient. Long distances require alternative methods such as fibre optics.

In fibre optic data transmission lines, light pulses are used to transmit the data. The transmitter receives an electrical signal and converts it into light pulses, which are coupled into an optical fibre. On the other end of the fibre, a receiver diode converts the light pulses back into an electric signal.

This has advantages over long distances compared to copper connections. Fibre optic cables are unaffected by electromagnetic interference and can achieve higher switching frequencies since there is no longer a copper cable to be recharged. It also helps with the decoupling of ground potentials. Because there is no electrical connection, differing ground potentials between the transmitter and receiver do not matter.

However, fibre optic systems come at a higher cost than electrical transceivers and the hardware required at both ends of the fibre will require more physical space and power than their copper-based counterparts. There are also limitations for the transmission distance of data through optical fibres. The obvious factor influencing this is attenuation. Light will be lost by imperfect connections at every point where fibre is connected to another one, by cable impurities and potentially dust on their connectors. When the signal is attenuated too much, the receiver will not be able to identify it anymore.

The less obvious factor influencing the maximal transmission distance is signal propagation in multimode fibres: Within an optical fibre, light can travel along different modes or pathways to the receiver. Each of these modes has its own signal propagation time, which it needs to reach the receiver. For a multimode fibre, the transmitter will inject the signal into multiple modes at once and the receiver will sum up the signal from all modes at the other end. The signal edges of the received signal will be dispersed since this sum contains components with mismatched propagation times. This effect is called multimode dispersion [39] and will make the signal unusable when the dispersion reaches a magnitude of the distance between individual bits in the signal.

The range of an optical connection is, therefore, limited by the frequency of the signal and the amount of modal dispersion, which results in a bandwidth per inverse distance measured in [MHz · km]. Different optical cable standards have different dispersion properties and are shown in table 5 [49]. As examle, an OM3 cable is able to transmit a 10 Gbit/s signal over a distance of roughly $\frac{2000\ \text{MHz·km}}{10000\ \text{MHz}} = 200$ m.

Cable Type	Bandwidth [MHz·km]
OM1	200
OM2	500
OM3	2000
OM4	4700

Table 5.: Bandwidth for different OM standards for optical fibres. Data from [49].

There are ideas for using the individual transmission modes of a multimode fibre independently to establish multiple connections at the same time. However, multiplexing different modes is a technical challenge and there is some crosstalk between the modes during propagation in the cable [50]. This is not available as consumer electronics.

Single-mode fibres are used for larger distances. They are manufactured in a way that creates only a single transmission mode and are for this reason less affected by modal dispersion. The disadvantages are a higher price for the cables and transceiver modules and lower tolerances for the alignment of fibres at connection points [49].

4

Mu3e Data Acquisition System (DAQ)

The sensors discussed in chapter 2 produce the data required to identify and reconstruct muon decay events. Due to the very high muon decay rate, the amount of collected data will be in the order of 100 Gbit per second, which makes permanent storage of that data impractical. Therefore, a pre-selection needs to take place during data taking. The pre-selection will necessarily have to select data from individual muon decays with kinematics that fall into the signal region. The decay can only be reconstructed if information on all decay products is available in the reconstruction software. Even on a powerful machine or GPU, the reconstruction software used for the simulations in the introduction chapter will not be able to process this amount of data in real time.

Parallelisation across multiple machines is therefore needed to perform this task, and the natural way of parallelising a lot of muon reconstructions is to distribute them across multiple machines in a way that each machine has to reconstruct only a part of all muon decays. For the data acquisition system, this means that for each individual moun decay, there needs to be a way to ensure that all data corresponding to this particular decay is delivered to the same machine in real time.

This statement has a lot of implications for the DAQ. First of all, it implies that the datastream in the DAQ needs to be sorted in time at some point since time information is the only way to separate moun decays from each other before reconstruction takes place. As described in 2.2.1 and 2.3.1, the output of both used ASICs is inherently not time sorted, which makes the creation of a time-sorted datastream that can be easily distributed across multiple reconstruction sites one of the tasks of the DAQ system.

Creating time slices of the data in order to give them to the different reconstruction sites also makes it necessary that the detectors involved in producing the data operate on the same time basis. To ensure that, all sensors need to be supplied with the same clock signal and a reference point in the form of a reset to synchronise.

The system to supply this clock & reset is also necessary for the reconstruction itself since precise time information can be used to resolve pile-up of multiple muon decays and to identify the charge of low z-momentum decay particles by time of flight (ToF) methods. The detector with the best time resolution reaches a resolution of 70 ps. Therefore, the synchronisation across the whole detector and the overall precision of the clock and reset system has to be smaller than 70 ps in order to preserve the timing precision when combining individual detections of multiple detector components for the reconstruction.

In addition to the supply of a timing reference and the readout and preprocessing of the detector data, the DAQ system also has to provide many other kinds of services,

such as configuration and tuning for MuTrig and MuPix ASICs at a sufficient speed, live monitoring of different properties of the system, the capability to inject data for test purposes, start/stop procedures for data runs, emergency procedures to protect the system from events like overheating and various other things.

This chapter will discuss in detail the tasks and challenges for the data acquisition system and the design decisions taken to implement solutions for them. Some of these aspects were already partially addressed by the author in the master thesis preceding this work [51]. Sections related to those aspects have been expanded and updated with developments made during this thesis.

4.1. Overview

The Mu3e data acquisition system is structured into four layers. The detector ASICs discussed in the previous chapter represent the lowest layer in this system. The next layer is made up of 114 custom-designed "frontend boards" (FEBs) located within the service wheels on the sides of the Mu3e detector. These boards, still within the magnet and helium atmosphere, manage all communication to and from the detector ASICs, and receive data from them via up to 36 1.25 Gbit/s LVDS links per FEB. Of the 114 FEBs, 88 are allocated to the pixel detector, 14 to the tiles, and 12 to the fibre detector.

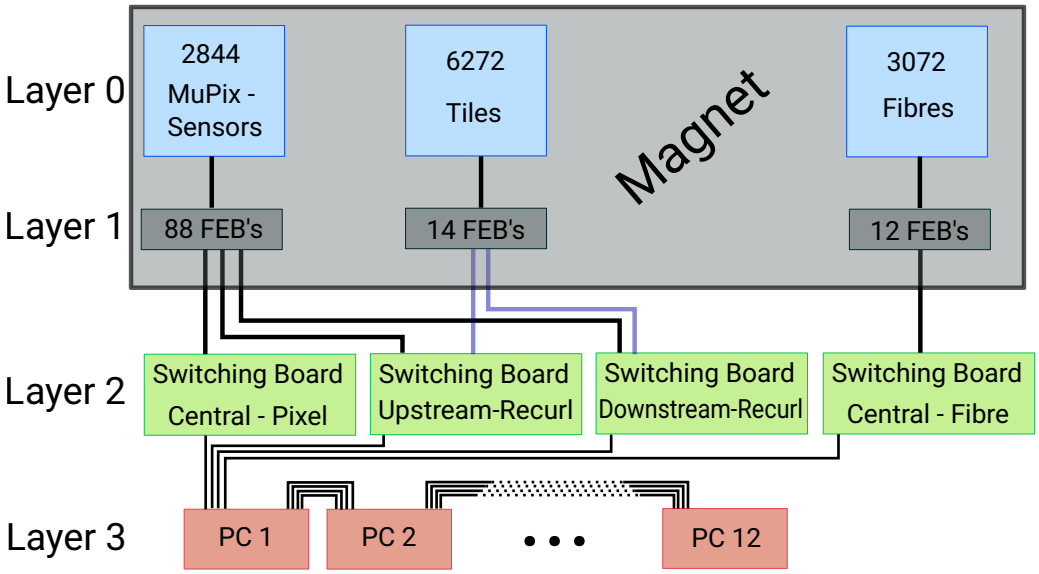


Figure 44.: Overview of the layers of the Mu3e data acquisition.

Each frontend board connects to one of four switching boards outside of the magnet through optical fibres operating at a data rate of 6.25 Gbit/s. One switching board receives all the data from the upstream, and the other one receives all the data from the downstream recurl station. The data from the central station is divided into pixel

and fibre data with a dedicated switching board for each detector type in this station.

The last layer consists of a daisy chain of PCs hosting one receiver board and one graphics card each. The first PC in the chain links to all switching boards via 16 optical fibres, each with a bandwidth of 10 Gbit/s. All following PCs in the chain have a connection with the same bandwidth towards the next one.

The overall data rate from all subdetectors is expected to be at around 100 Gbit/s in phase I [23] and 1 Tbit/s in phase II of the Mu3e experiment. The system does not contain a hardware trigger signal. Therefore, all the data needs to be streamed to the highest layer.

4.1.1. Layer 1 – Frontend Boards

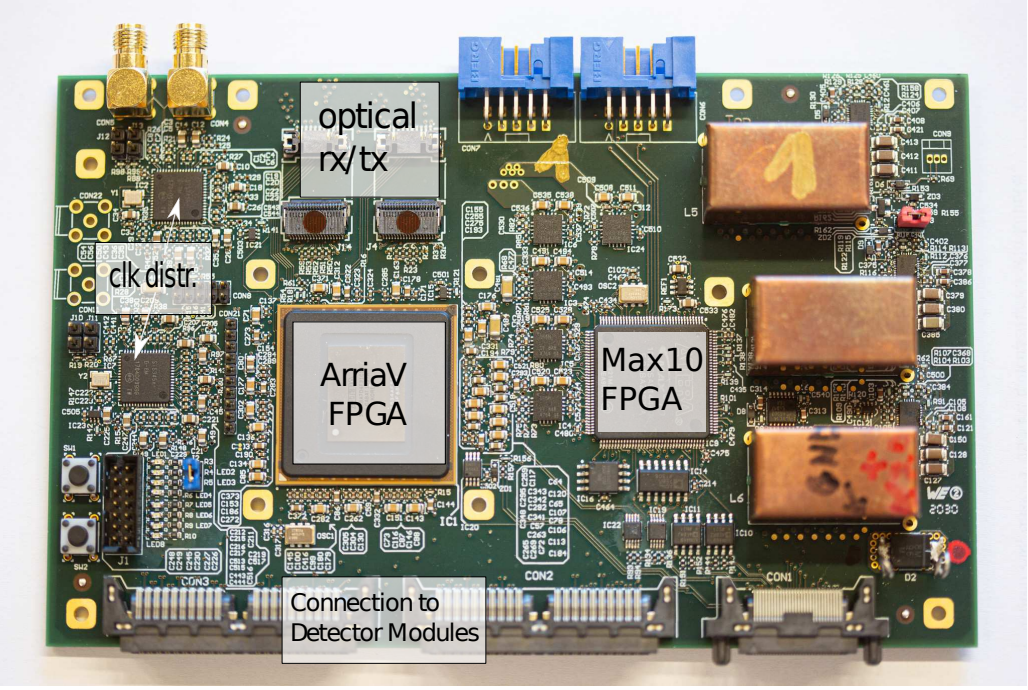


Figure 45.: Picture of the frontend board (FEB).

Layer 1 of the data acquisition is built from 114 frontend boards (FEBs). They were developed in the Mu3e collaboration in order to combine all the required functionalities with the tight available space at their intended position. Their main component is an ArriaV FPGA, version 5AGXBA7D4F31C5, with 91680 ALMs, roughly 14 Mb of M10K and 1.5 Mb of MLAB memory.

It communicates with the system's upper levels via optical transceivers running at 6.25 Gbit/s. The downwards connection to the detector modules is done electrically and consists of various control signals and up to 36 1.25 Gbit/s LVDS pairs to receive data.

The task of the ArriaV is to process, sort, and repackage the data from the detector modules and provide the necessary control interfaces to operate them. The detailed issues will be discussed in this chapter.

Apart from the upwards and downwards datapath, the FEB is also responsible for clock and reset distribution to the detector and contains dedicated hardware to implement this.

Since the ArriaV does not contain non-volatile memory, its configuration will be lost when the power is turned off. This is the main reason for the smaller 2nd FPGA (MAX10) on the FEB, which configures the ArriaV from flash memory.

The FEBs slide into the 60 slots of the circular backplane PCBs within the service support wheel shown in figure 46. Cooling is provided by water flowing through the aluminium parts of the wheel. These are then connected via copper heat pipes to cooling blocks mounted on the FEBs. The backside of the circular PCB uses a similar sliding and cooling mechanism to connect detector adapter boards (DABs), which connect to the cables of the actual detector modules.

The backplane PCB provides power to the FEB and a connection between FEB and DAB. The square aluminium plates on the left and right sides are the patch panels for the optical data cables. From there, the cables are routed out of the magnet and over a distance of $\mathcal{O}(50\text{ m})$ to the server racks where the rest of the DAQ system is located.

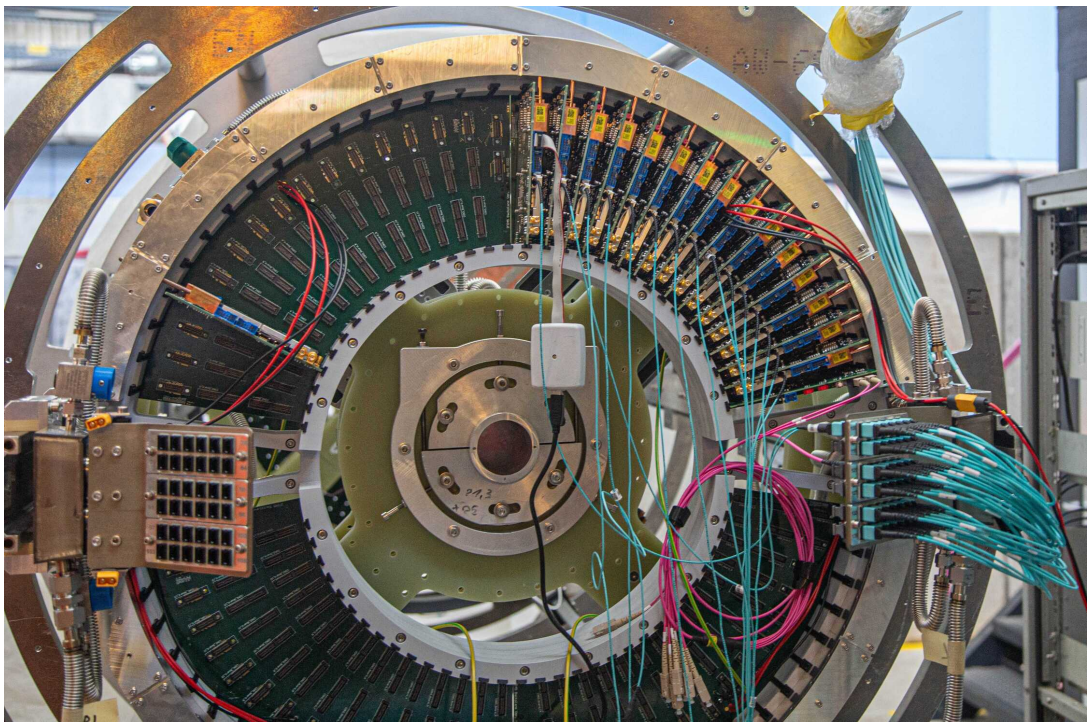


Figure 46.: Picture of the service support wheel [52]. An identical wheel is located on the other side of the detector.

The firmware used for the Max10 FPGA can be identical for all FEBs in Mu3e. This is not possible for the firmware of the ArriaV since different versions are needed depending on the sub-detector type to which the particular FEB is connecting.

In order to sufficiently integrate all three subdetectors into the same system, it is crucial that they use common infrastructure, protocols and methods as much as reasonably possible in the Mu3e DAQ. For this reason, the ArriaV firmware will be divided into two blocks: A common part of the firmware, which will be used by all three sub-detector types and a sub-detector specific part. The common part will implement things related to the board and communication infrastructure and manage the connection to the higher layers of the DAQ system.

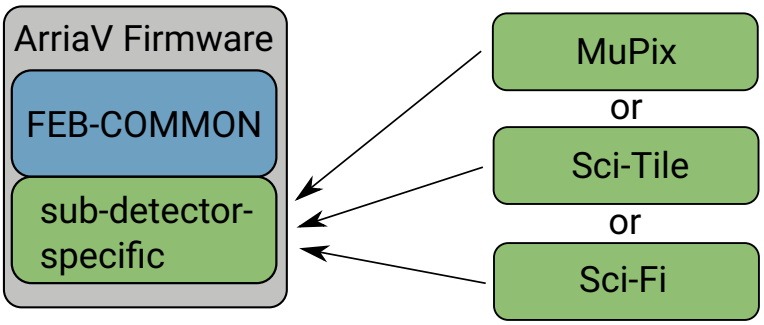


Figure 47.: Division of the FEB firmware into a common and a detector-specific part.

The sub-detector specific part implements the circuits necessary for the operation and readout of the Mupix or MuTrig ASIC. The Sci-Fi and Sci-Tile blocks will be similar since they both connect to MuTrig chips. The reasons for them not being identical will be discussed later.

This separation ensures that starting from the FEB-common block, the control and readout of data is similar for all three detectors in Mu3e. It also simplifies development since the FEB-common firmware only needs to be implemented once and can be reused for the three FEB flavours[1].

4.1.2. Layer 2 – Switching Boards

The four switching boards (SWB) are hosted in four switching servers outside of the beam area. They connect to the other end of the optical fibres from the FEBs according to the assignment in figure 44. Therefore, there will be an upstream and downstream SWB, a central pixel, and a central scifi SWB.

The board was originally developed by the LHCb collaboration under the name PCIe40 [53] and is used by Mu3e due to its optical input and output capabilities. It contains eight Avago Minipods [54], giving it the possibility to connect to 8x12 optical fibres, each of which could be operated at 10 Gbit/s.

[1] There are actually a lot more than three flavours of FEB firmware, but they do not provide a meaningful contribution to this section. Other firmware versions will be introduced later if they become relevant.

Similar to the FEB, the main component is again an FPGA, this time an intel Arria10. It receives the data from all the connected frontend boards and repackages it into a data stream that can be sent onwards to the third layer of the DAQ. It is also the place where the control datapath for detector operation is connected to software via a PCIe interface. Therefore, the third layer of the DAQ is no longer in contact with detector operation details.

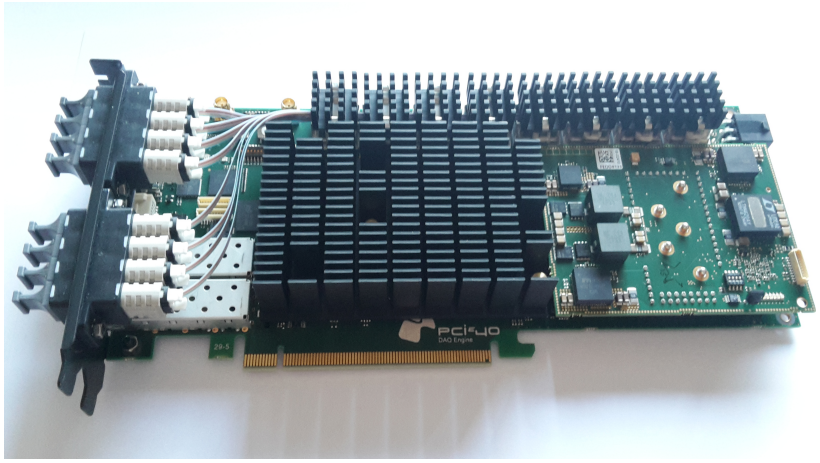


Figure 48.: Picture of the PCIe40 board developed by the LHCb collaboration.

4.1.3. Layer 3 – Farm PCs

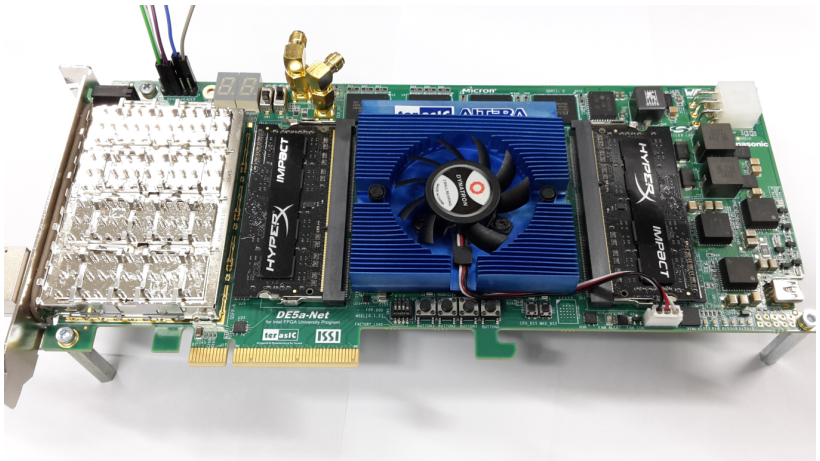


Figure 49.: Picture of the DE5a-Net development board, which is used as an optical receiver inside of each farm PC in the Mu3e DAQ.

When the data arrives at the last layer of the data acquisition, it is received by a chain of commercially available DE5a-Net development boards hosted in separate

servers.

Every board in the chain sends time slices of data via a direct memory access (DMA, section 3.8.6) to a GPU where a tracking algorithm is used to make a selection decision on that time frame based on the expected kinematics for a $\mu \to 3e$ decay. Selected time frames are sent to long-term data storage and will be used for offline analysis. Other frames are discarded.

Each board also forwards all data to the next server in the chain and flags the processed time slices. The procedure above is performed by each server in the chain until all time slices are flagged as processed. It was shown that 12 PCs equipped with a Nvidia GTX1080Ti graphics card are able to achieve this for the expected data rates in the first phase of the experiment [23]. With newer graphic cards available at the start of the experiment, this number can likely be reduced [55].

The complete layer three of the DAQ will also be referred to as farm PCs, filter farm or GPU farm in the following parts of this thesis.

4.1.4. Structure of this chapter

The discussion of challenges and their solutions in the DAQ system has to follow a path through the system, explaining issues as they appear along that path.

However, the choice of that path is not obvious, and the Mu3e DAQ was not written in that way. It started with bits and pieces, which have developed over time into a first functional system, which was then further optimised and changed. This thesis does not aim to discuss the history of this process but to present the solutions that came out of it and are implemented today.

In order to do so, a path has been chosen that aims to minimise places where references to later sections are needed to understand the discussion. This path does not directly follow the flow of data from the detectors to the analysis since this would create too many of these forward references.

We will start with an introduction to the software framework, continue with the connection between layers one and two and discuss how the software can communicate with frontend boards (FEBs). This will be followed by a discussion of the common parts of the FEB, which will include the reset and clock distribution, slowcontrol and general architectural aspects.

After this has been introduced, the discussion will continue with data flowing towards the pixel detector and everything that is necessary to enable the MuPix chips to take data. This section will range from layers two to zero and use many of the previously discussed aspects.

Once this has been achieved, we will turn around and follow the data path from the MuPix to the common part of FEB firmware and highlight the differences to the scifi and tile detector before continuing through layers 2 and 3 until the analysis.

A few parts of the system are necessary to operate it, but not strictly necessary to understand the previous discussion. Those aspects will be addressed separately at the end.

4.2. MIDAS

MIDAS (Maximum Integrated Data Acquisition System) is the software framework used to build the Mu3e DAQ. It has been continuously developed at PSI and TRIUMF since 1993 [56].

Controlling an experiment with MIDAS conceptually consists of three parts. The MIDAS frontends are independent software programs that control the detector's hardware components or functionalities[2]. In Mu3e, for example, there is a MIDAS frontend for the power supplies, the magnet control, one for each switching board and many others.

These MIDAS frontends can operate on any device in the network. There is no requirement for them to run on physically the same machine. For example, the MIDAS frontends for the switching boards will, like the boards themselves, be operated by the server in which the board is located.

All MIDAS frontends can read and write to a common online database (ODB) located on a machine in the network. The ODB can contain any kind of information and is structured like a folder system. ODB variables are accessed by name. There is also the possibility of setting up watch functions on specific variables or parts of the ODB. Any change to one of these ODB entries will then trigger a user-defined function in the according frontend. This gives the individual frontends a way of communicating via the ODB.

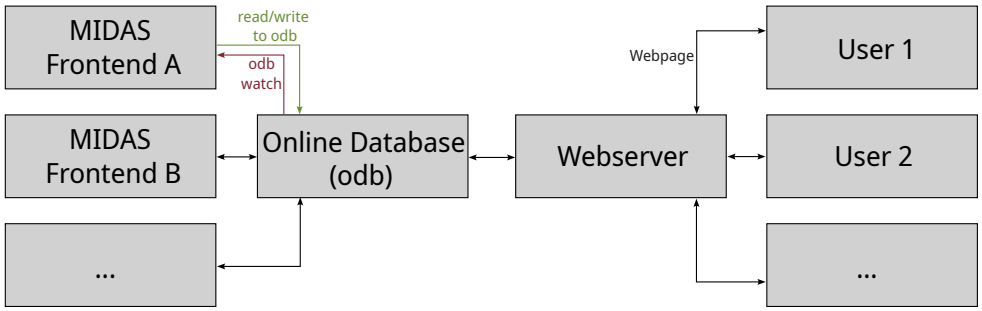


Figure 50.: Controlling an experiment with MIDAS.

The third component is a web server, usually hosted on the same machine as the online database. The web server provides users with access to the ODB via web pages. The web pages can be written specifically for the experiment and can be used to implement any desired user interface with JavaScript and HTML.

The user can interact with the webpage and change ODB values via the web server. Changing ODB values can trigger watch functions in the MIDAS frontends, which can perform actions within the detector.

MIDAS also provides additional functionalities which are often needed by physics experiments. Run starts and stops are predefined separate procedures that can trigger

[2] MIDAS frontends have nothing to do with the *frontend board*. FEBs will be controlled by MIDAS frontends, but usage of the term *frontend* for both is just a coincidence.

a sequence of events within the connected frontends.

All values in the ODB can be connected to a history system, which automatically creates a time series of this value and can, for example, be used to display the temperature of a detector changing over time. Alarms can be created on ODB values and trigger frontend functions. For example, a temperature rising above a certain value can be set as an alarm and cause some action.

4.2.1. Detector data

Large volume data readout within the MIDAS system is separated from the previously described control and monitoring functionalities. Detector data is read out by the MIDAS frontends but not written into the ODB. Instead, it is written into the MIDAS event buffer, where it can be accessed by data loggers or other software. Data loggers write the received data to disk.

The data format in the resulting file and within the event buffer is called a MIDAS bank. A bank consists of a header and data. The header contains information such as the serial number of the event, the bank name, Unix time at creation and bank size. Precise formatting information can be found in [56].

Multiple bank names can be defined, referring to independent streams of data. Offline analysis can loop over the file created by the data logger and search for the bank names relevant to the intended analysis.

4.3. Frontend Board Communication

The connection between the frontend boards (FEBs) and switching boards (SWBs) is the only high-speed communication interface to the inside of the Mu3e magnet. It is implemented using pairs of OM3 optical fibres operated in both directions on a serial bit-frequency of 6.25 Gibt/s. This allows for complete electrical decoupling of the detectors from DAQ components outside of the magnet since all communication is based on optical fibres[3]. Except for power, there are no electrical connections to the detectors, which is beneficial for grounding schemes since it avoids ground loops.

4.3.1. FEB Communication Protocol

The standard integer size on the machines connected outside of the magnet is 32-bit. In order to simplify the software implementation there, large parts of the FEB firmware are also based on 32-bit protocols, including the parallel interface width of the optical connection to the SWB. From the FEB firmware point of view there is no particular reason for a 32-bit architecture.

Every byte in each 32-bit word transmitted between SWB and FEB is 8b/10b encoded before transmission, which increases the actual parallel word width to 40

[3] There are also alternative communication channels with very limited bandwidth. They will be discussed later.

bits, effectively resulting in a $\frac{6.25\ \text{Gb/s}}{40\ \text{b}} = 156.25$ Hz transmission frequency for 32-bit words per connection.

Most transmissions between SWB and FEB will be organised in packets. The packet structure is similar for both directions and is shown in table 6. It consists of a preamble, the payload and a trailer. For the lowest bytes of preamble and trailer 8b10b comma words (table 3 in section 3.8.3) are used. They are necessary to identify the start and end of a packet unambiguously.

31 30 29 28 27 26	25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	
TYPE	SC	FPGA ID	K28.5	Preamble
Word 0				Payload
Word 1				
Word 2				
...				
Trailer Payload			K28.4	Trailer

Table 6.: Structure of a data packet between SWB and FEB.

The payload is unable to produce a similar bit sequence since K28.5 and K28.4 are unique in their encoded 10-bit representation. Their 8-bit representation can be identical to the contents of the payload, but the 8b10b encoding/decoding logic will have additional input/output bits to identify comma symbols. The idea of this is shown in figure 51.

The serial data is received from the optical fibre, aligned and deserialised into 40-bit words which are then 8b10b decoded. The result is a 32-bit word and a 4-bit datak indicator, which is used to show which of the 4 bytes in the 32-bit word was an 8b10b comma symbol in the 40-bit representation. The datak indicator is also used for byte alignment in order to ensure the position of the comma symbol in bits 7-0 of table 6.

Figure 51.: Concept of 8b10b decoding and deserialisation.

The transmitting side uses the same concept in reverse without the need for alignment. The logic controlling the transmission and protocol handling is provided with the 32-bit data word and 4-bit datak indicator for the receiver and transmitter.

In addition to the comma word K28.5, the preamble contains fields named FGPA ID, SC and TYPE. The FPGA ID is the 16-bit identifier of the FEB where this packet is headed to or coming from. At the time of writing this thesis, this is used as a

cross-check for correct cabling and does not have significant routing purposes since the protocol is only used for point-to-point connections between SWBs and FEBs.

Should the architecture of the Mu3e DAQ change in the future, the FPGA ID can be used for routing purposes without breaking compatibility with existing firmware. When inserted into the service support wheel, the FEB is informed about the ID via position identifier bits on the backplane PCB. For test systems, the ID is configured from the software[4].

The TYPE field specifies the type of the packet and substructure to expect in the payload. The specified types are listed in table 7. Each subdetector has a type for hit data and debug data. The hit data type is only sent from the FEB to the SWB and contains information such as timestamp and location of particle detections. The BERT and debug types are only used during development.

All control and monitoring information between the SWB and the FEB is exchanged via the *SlowControl* type. The name slow control has historical reasons and does not imply anything about the packet's transmission speed. Slowcontrol packets are transmitted with the same frequency as the other types.

When no packets are currently in tranmsission, the link is required to continuously send K28.5. This also allows for gaps within the tranmsission of a packet: If the firmware is unable to send an N-Word packet within N clock cycles, it can fill the resulting gaps with K28.5 without breaking the protocol.

6-bit ID	**TYPE**	comment
111010	MuPix	Hit information from pixel detector
111000	Scifi	Hit information from scintillating fibres
110100	Tile	Hit information from scintillating tiles
000111	SlowControl	All control and monitoring data
000000	idle	-
111011	MuPix debug	general-purpose MuPix debug data
111001	Scifi debug	general-purpose Scifi debug data
110101	Tile debug	general-purpose Tile debug data
000010	BERT	Bit error rate test

Table 7.: TYPE identifiers in the SWB-FEB protocol.

The payload of slowcontrol and hit data packets follows a specified lower level protocol. The details for data packets will follow later since they are irrelevant for two-way communication between SWB and FEB.

Communication with a FEB is based on addresses on which read/write actions can be performed. Every setting and status information on the FEB is assigned an address under which it can be accessed. A setting on the FEB is, for example, a register controlling the FPGA-ID in DAQ test-systems (as mentioned above). Slowcontrol

[4]How things on the FEB are configured from the software is the purpose of this section and will follow soon.

packets can then be used to read or change the register using its address. Such registers will be referred to as slowcontrol registers or sc-registers during this thesis.

Slowcontrol registers can behave differently when read/write actions target their address. For example, a register connected to a temperature sensor will not react at all to a write transaction. A sc-register connected to the read port of a FIFO will also not react to write transactions, but in addition consecutive reads from the same address will give different results until the FIFO is empty. Should the sc-register be connected to the write port of the FIFO instead, it will not be possible to read useful information from it. We will encounter many of these cases where sc-registers do not behave like content behind a memory address. The software parts need to be written accordingly since reading multiple times from the same memory location would otherwise be removed by the software compiler during optimisation.

The substructure of a slowcontrol packet from the SWB towards the FEB is shown in table 8. It would be inefficient to issue a separate slowcontrol packet for each register access. As we will see in the following chapters, it is quite common that multiple consecutive register addresses have to be read or written from software together. For this reason, the slowcontrol packet specifies the start address of a transaction in the second word of the packet, followed by either the length of the requested data or the length of the data followed by the data to be written. The FEB firmware is then required to read the data from registers *start address* until *start address* + *length* or to write data to registers *start address* until *start address*+*length*. The identification as a read or write packet is done via bit 0 of the SC field in the preamble. Write packets where *length* does not match the amount of data words are invalid.

31 30 29 28 27 26	25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	
000111	SC	FPGA ID	K28.5	preamble
- $\bar{M}$ $\bar{S}$	$\bar{T}$ $\bar{R}$	start address		
-		length		read
-		length		write
data				
data				
-			K28.4	Trailer

Table 8.: Structure of a slow control packet heading towards a FEB.

The upper bit of the SC field sets the address to non-incrementing. This means that the packet will result in a total of *length* read/write actions on the same address. This is, for example, useful to read/write data from or to a FIFO. The four possible combinations for SC are shown below:

- SC = 00: incrementing read
- SC = 01: incrementing write
- SC = 10: non-incrementing read
- SC = 11: non-incrementing read

The frontend boards have to reply to each received slowcontrol packet. The format is shown in table 9. A read command must be answered with a packet containing the identical start address and length, followed by the read data. A write command needs to be acknowledged by start address and length.

31 30 29 28 27 26	25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	
000111	SC	FPGA ID		K28.5	preamble
-		start address			
-		1	length		write ack
-		1	length		read ack
data					
data					
-				K28.4	Trailer

Table 9.: Structure of a slow control packet heading towards an SWB.

When the SWB needs to issue the same write command to all connected FEBs, the same message can be broadcast to all FEBs simultaneously. Acknowledge packages would create problems in this case since the SWB would have to process them for all connected FEBs. The acknowledge for write operations is primarily a system integrity check and actually not needed for many applications if the system is operating error-free, which is why acknowledge packages can be suppressed by setting the $\bar{R}$ bit in table 8. The same location in all acknowledge packages is also set to 1 since we do not expect another reply from the SWB in these cases.

Broadcasting is, therefore, a more efficient communication mode if all connected FEBs should be addressed. If that is not the case, subgroups of FEBs can be defined in the upper bits of the second word of table 8. At the time of writing this thesis, the subgroups follow the type of sub-detector specific firmware present in the ArriaV FPGA. The bits $\bar{M}$, $\bar{S}$ and $\bar{T}$ are used to instruct the according type of FEB to ignore everything in this message. If $\bar{M}$ is set to 1, all MuPix FEBs will ignore the transaction. The same applies to scifi ($\bar{S}$) and tile ($\bar{T}$) FEBs. $\bar{M}$, $\bar{S}$ and $\bar{T}$ are different from $\bar{R}$ in the sense that $\bar{R}$ only causes suppression of the acknowledge package, while $\bar{M}$, $\bar{S}$ and $\bar{T}$ prohibit the execution of the read/write action. The reason for the inversion of $\bar{M}$, $\bar{S}$, $\bar{T}$ and $\bar{R}$ is backwards compatibility with a previous version of the protocol.

In rare scenarios, broadcasting can also be useful for read transactions. For example, in a situation where a FIFO on multiple FEBs should be read empty, but the content

is not of interest.

The protocol shown here is used on the optical fibres between the FEBs inside the Mu3e magnet and the SWBs in the server room. Parts of it were developed for Mu3e during this thesis. An older version of it was already shown in [51]. The next section will discuss the firmware implementation for this communication link on the side of the SWB. The other side requires a detour into FEB firmware architecture first and will follow afterwards in section 4.4.4.

4.3.2. SWB Slowcontrol implementation

The slowcontrol system of the Mu3e DAQ consists of the firmware and software components outside of the magnet and firmware blocks on the frontend boards in the service support wheels of the Mu3e cage. As previously introduced, these two parts are connected with a pair of optical fibres for each FEB and are operated at 6.25 Gbit/s with a 32-bit interface width. This section will discuss the parts on the SWB where the packets from the previous section need to be assembled and directed towards the correct FEB. Afterwards, the reply from the FEB needs to be processed. The speed at which this can be done is essential for the efficient operation of the Mu3e detector.

Section 3.8.6 of the previous chapter introduced PCIe communication and the function of PCIe base address registers (BARs). As already mentioned there, four base address registers are implemented for the SWBs in Mu3e and assigned to the resources read/write registers and read/write memory. These will now be used to build the SWB end of the Mu3e slowcontrol system. Details on BARs and PCIe communication have been discussed in section 3.8.6.

The essential firmware components for this end of the slowcontrol system are shown in figure 52. Conceptually, the software writes the read or write packet into the PCIe write memory and the packet's length into a predefined location in the write registers. A toggle of an enable register in the write regs will start a process in the slowcontrol main (SC main) entity, which reads the packet from write memory and directs it to the correct transceiver where it is 8b10b encoded, serialised and sent to the FEB.

The FEB will reply with the read data or an acknowledge package. After deserialisation and decoding, the datastream will first go through a data demerger entity, where these slowcontrol packages from the FEB are filtered from other packets on the connection, such as data from the Mu3e detectors.

A secondary slowcontrol entity will then process the slowcontrol packets from the FEB and write them into the PCIe read memory. The address at which the last packet was written is reported in a predefined location of the PCIe read registers, which allows the software to find and read the reply package from the PCIe read memory.

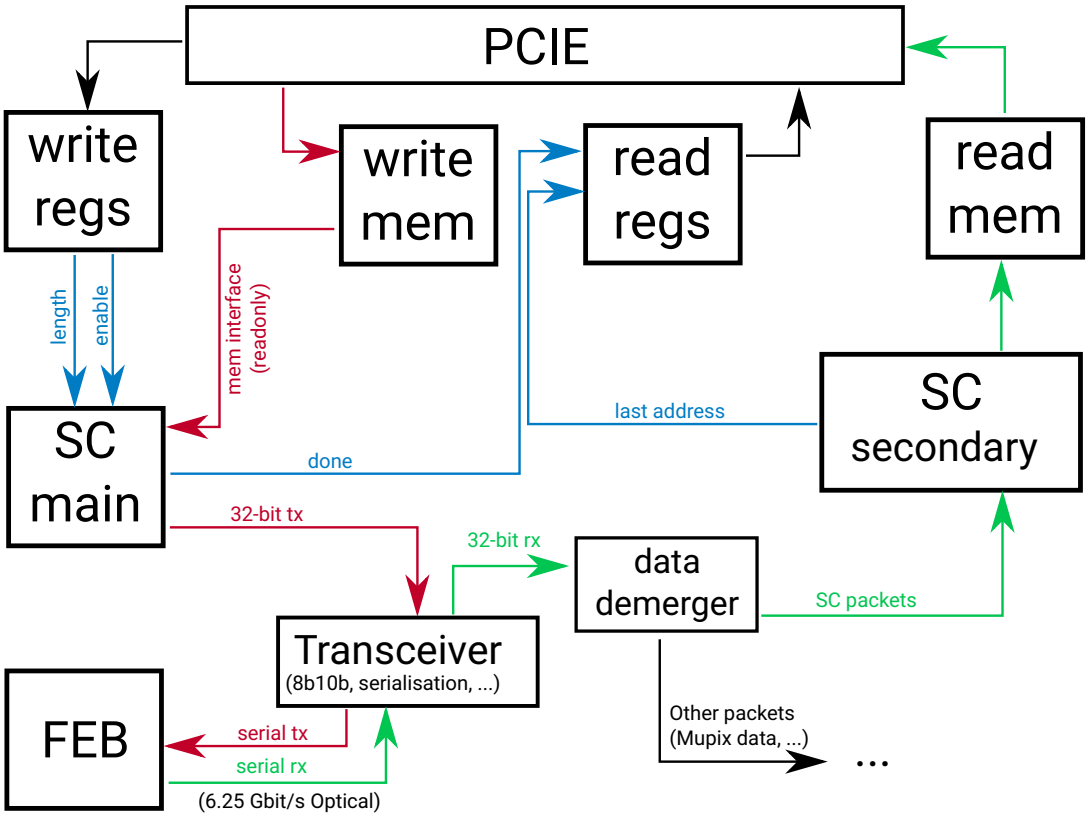


Figure 52.: Schematic of SC firmware on the SWB. Slowcontrol information flowing towards the FEB is displayed in red, and the response in green. Control signals which are directly relevant for slowcontrol transmissions are shown in blue. Other control signals and clock domains have been left out for simplification.

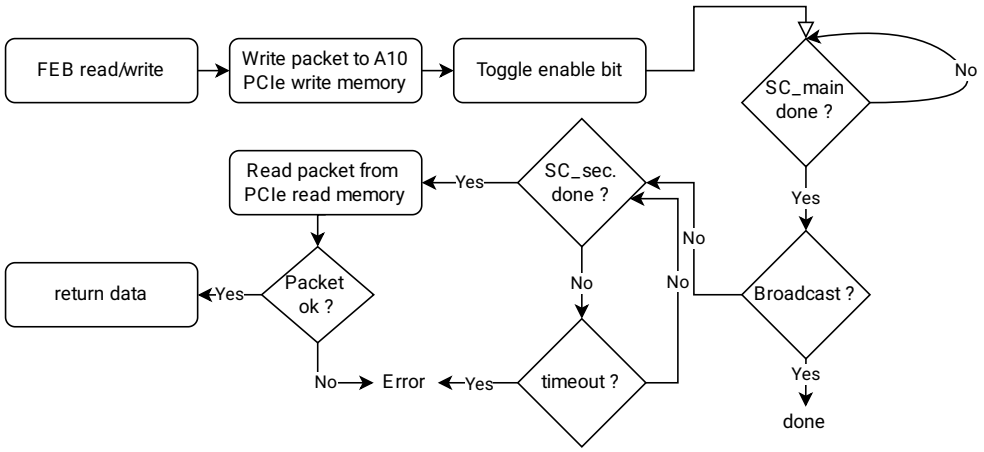


Figure 53.: Software flowchart of a slowcontrol transaction.

The software procedure is shown as a flowchart in figure 53. After the enable bit has been toggled, the software will poll the done register in the PCIe read regs until the slowcontrol main entity has sent the entire packet towards the transceiver. If the packet was broadcasted to all FEBs, no reply is expected, and the transaction is finished. Otherwise, the software will start polling the last address of the secondary slowcontrol entity until a reply packet is reported in the PCIe read memory. This packet is then acquired from there and checked for protocol integrity. In case of a read action, the data read from the FEB is afterwards returned to the part of the software that requested it.

If the FEB fails to reply within a specific timeframe, the software will issue a timeout error. This is detected by the polling time of the secondary slowcontrol entity. Timeouts usually occur when the physical connection to the FEB does not exist. This is a frequent error during the commissioning of a Mu3e DAQ system and can be introduced by many reasons[5].

The only other way to produce an error is when the packet read by the software does not comply with the protocol specification. Since the state of some components after receiving misformatted packets cannot be guaranteed, a single error is often followed by further errors. In both cases, no action will be taken to rectify the situation. Transmission errors between the FEB and SWB are considered unacceptable and require the user's intervention. The system will not automatically attempt corrections by resetting all involved entities.

4.3.2.1. FEB routing and Broadcasting

The routing of an SC packet to the correct FEB is implemented via the FPGA ID field in table 8. The SWB will read the packet from the PCIe write memory and convert the FPGA ID into a one-hot encoded bit vector, which is used to direct the packet to the transceiver connected to the targeted FEB. At the time of writing this thesis, the FPGA ID is the integer position of the FEB within that bit vector. A special broadcasting FPGA ID of $0x2F$ marks all FEBs as targets and activates all bits in the routing vector.

During the development of the Mu3e DAQ, the FPGA ID was determined by the output of the SWB to which the FEB is connected. The actual FEB ID in Mu3e will be determined via the position of the FEB within the service support wheel. The issue with this is that many existing test setups and variations of the Mu3e DAQ do not have their FEBs connected to the final backplane of the Mu3e detector. Therefore, the FEBs in these systems do not receive an FPGA ID via that backplane.

The current solution is that the FPGA ID in table 8 is not required to match the FPGA ID in table 9. The SWB will target FEBs based on the index of the optical output cable. The FEB on the other end of that cable will not use the received FPGA ID and will consider any received packet as valid. The response packet from the FEB is assigned an FPGA ID as specified by location on the backplane PCB. This allows

[5] cabling mistakes, FEB not powered, FEB firmware not configured, ...

for automated checks of correct cabling since both IDs are visible to the SWB.

However, other implementations are possible and will likely be considered in the future to avoid additional mapping requirements. One option would be to broadcast all slowcontrol messages from the SWB generally and have the FEBs react only if their backplane address matches the received FPGA ID. This removes the need for an FPGA ID on the side of the SWB and would be a simple solution for the actual Mu3e detector but create issues with other system variations. An advantage of this idea is that the cabling from the SWB to the FEB does also not require mapping anymore. Any output of the SWB could be connected to any FEB in this case. Achieving the same for the SWB inputs would be more difficult since the detector data and other packets coming from the FEB must also be considered here.

The other option would be to remove the FPGA ID on the frontend board side and only use IDs assigned by the software. This was in operation for early versions of the Mu3e DAQ but is likely not a suitable option since it removes the possibility of verifying the FEB location without physical access to the detector.

For the same reason, the previously discussed broadcasting subgroups (bits $\bar{M}$, $\bar{S}$ and $\bar{T}$ of table 8) are not purely implemented via software. It would be possible to write the one-hot encoded routing bit vector on the SWB directly from the software and thus broadcast messages to arbitrary combinations of FEBs. It was decided not to implement this. Instead, all broadcast messages are always sent to all FEBs, and the FEBs decide to which subgroup they belong to.

At the moment, this is done for $\bar{M}$, $\bar{S}$ and $\bar{T}$ based on the subdetector-specific firmware present on the ArriaV, but in principle, it could be hardcoded in the FEB firmware to backplane IDs belonging to specific detector parts. Broadcasting to all MuPix FEBs, for example, is then not broadcasting to all FEBs with a MuPix firmware but to all FEBs connected to the actual MuPix detector slots in the backplane, which is much less error-prone than the software configuration of the SWB. For systems where this is only steered by software, a wrongly connected cable could lead to damage since situations where a FEB with scintillating fibre firmware is connected to a pixel sensor can easily be created[6] (Firmware upload to the FEB is also done via the slowcontrol system, as we will see later).

4.3.2.2. Bandwidth and Latency Optimisation

The schematic in figure 52 was simplified and left out a few important aspects. First of all, the PCIe interface operates at a frequency of 250 MHz, while the interface to the FEBs operates at 156.25 MHz (6.25 GHz with 8b10b encoded 32-bit words). This requires a clock domain transition in the system, which is located between the SC main and transceiver entity of figure 52. In addition, the SC main entity has to serve more than just one transceiver with slowcontrol packets. In a previous version of this system, the timing requirements of the distribution of SC packets to the transceivers were solved by significantly slowing down the SC main entity, which became the

[6] This issue was actually created during test runs.

limiting factor for the downwards slowcontrol bandwidth.

Different modifications of the system were then investigated to improve bandwidth and latency. Figure 54 illustrates the latency of a slowcontrol transaction against the size of the slowcontrol packet for different versions of the SWB firm- and software. The bandwidth can be calculated by dividing the number of words in the packet by the latency required to transmit it.

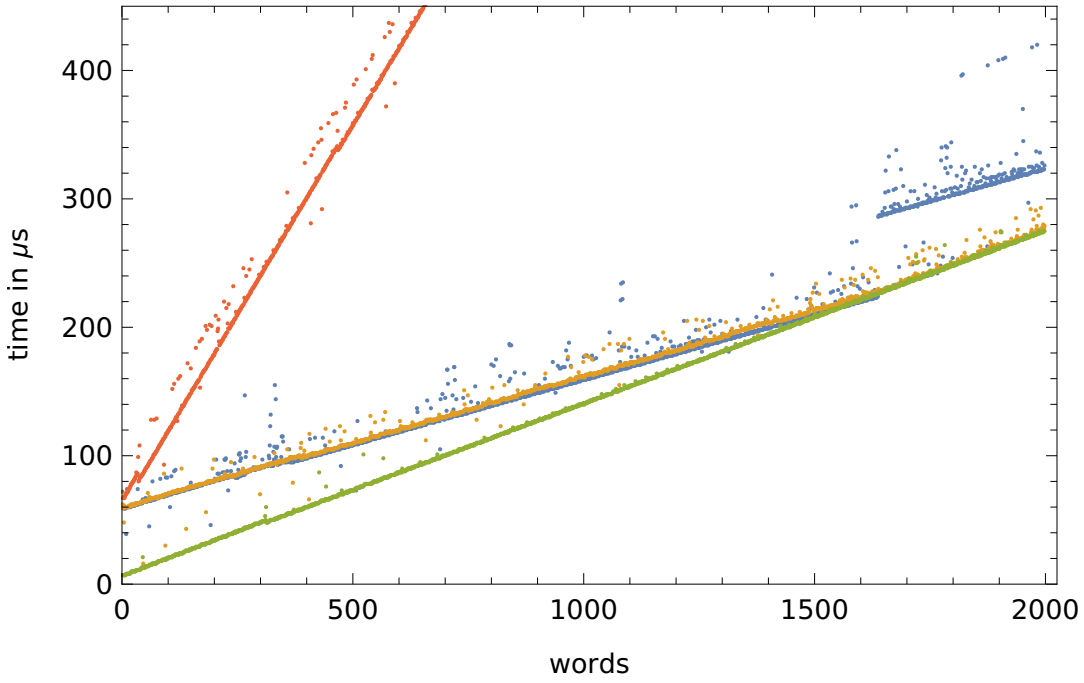


Figure 54.: Different versions of the SWB slowcontrol system and their response latencies based on slowcontrol packet size. Red is a read packet. Blue, orange and green are write packets.

The starting point for the optimisation of write packets was the blue line in figure 54, which immediately raises the question about the reasons for the 60 μs y-axis offset, the jump in latency at a packet size of around 1600 words and the slope between these points.

The reason for the jump at a packet size of 1600 words is the polling of the *done* signal of the SC main entity: The data is written into the PCIe write memory, and the enable bit is toggled by the software (see figure 53). Immediately afterwards, the software will read the *done* signal from the PCIe read regs.

This will return false if the SC main entity did not have enough time to distribute and read the packet from the PCIe read memory. That is more likely for large packets, and it happened at around 1600 words on the machine that this was tested on. These cases will always take longer since at least one additional transaction is required.

However, in the case of this measurement, the software implementation is also

relevant since it was written in a way where a context switch was introduced and left the resumption of the polling thread to the OS scheduler. When exactly the execution of the thread is resumed is not predictable on a scale of a few μs and leads to the jump in latency and the further scattering of measurement points towards higher latency values.

To avoid this issue, the software was changed, and a FIFO was introduced at the SC main entity's output. This allows the SC main entity to write data words at the full 250 MHz frequency into the buffer FIFO and report ready towards the software while data is distributed to the transceivers from the FIFO at a lower speed.

The frontend board will reply with an acknowledge packet once the transaction header is received and will not wait for the full message. For long write messages, the acknowledge packet will be in the PCIe read memory before the full data packet has left the SWB towards the FEB. The software can then continue with filling the next slowcontrol packet into the PCIe write memory. For large sequences of write actions, this effectively allows two slowcontrol packets to be processed at the same time and increases the bandwidth towards the detector.

Collisions between these two packets are not possible since the introduced FIFO provides backpressure to the SC main entity. The FEB not waiting for the full message is also not a concern since transmission errors between FEB and SWB are, as previously mentioned, unacceptable. Therefore, if the connection is unstable, it does not matter which exact packets are affected by this.

The y-axis offset resulted from a similar scheduling issue regarding the toggle of the enable signal for the SC main entity, and the slope between this and the jump point matches the speed limitation of the previous version of the SC main entity. Resolving the scheduling issues and introducing the output FIFO results in the green line in figure 54. For single-word writes, the latency improved by a factor of ten compared to the previous version.

The latency has a fixed offset of 6.70 μs and increases linearly with packet size by 0.1342 μs per word, resulting in a bandwidth of 238 Mbit/s. The exact values are machine and load-dependent. The measurements here were done for individual write packets without the parallelisation explained above on one of the actual Mu3e switching servers and without other packets transmitted between FEB and SWB. As we will see later, latency on the side of the FEB firmware can be neglected if no other packets are present.

Read packet latency shows a steeper increase with packet size. However, their optimisation was irrelevant for the following parts of this thesis, and they have not presented a bottleneck for any procedure in Mu3e at this point. Write transactions have been – and still are after the optimisation presented here – a bottleneck for a topic that will follow in section 4.6.3. The slope at which write latency increases seems to be the speed at which the PCIe driver can copy words into the PCIe write memory. Changing this would, therefore, require larger adjustments in the system, which are currently not justified.

This section has discussed the SWB end of communication with the components in the Mu3e experiment. In addition to their use here, the PCIe read and write registers

are also used for control functions within the SWB firmware that are unrelated to the slowcontrol system. Every control signal on the SWB is assigned an address there and can be accessed by software similarly to the *length*, *enable*, *done* and *last address* signals introduced here. As mentioned in the introduction of this chapter, the other firmware parts on the switching board will be explained once the discussion returns to the SWB with detector data. The FEB side of the slowcontrol system is discussed in the following section after an introduction to FEB infrastructure aspects.

4.4. Common FEB firmware

The common FEB firmware is the part of the firmware for the ArriaV FPGA on the frontend board which does not change depending on the type of connected subdetector. Technically the FEB also conatins a Max10 FPGA which does also not change with the type of subdetector but the term *common firmware* will only reffer to the ArriaV parts of the system.

These include the processing of packets to and from the SWB firmware discussed in the previous section. Most parts of the FEB slowcontrol system are located in this part of the firmware. However, the actual endpoints, such as slowcontrol registers or memory resources, are located in all parts of the FEB firmware since all parts require control and monitoring functionalities. The slowcontrol system and packet handling will be discussed in sections 4.4.3 and 4.4.4.

The other important task of the common FEB firmware is detector synchronisation. The MuPix and MuTrig ASICs are only able to provide accurate relative timing information for particle detections if all ASICs in the Mu3e DAQ system operate on the same time basis. In order to achieve this, a clock and a reset signal have to be provided to all parts of Mu3e and they need to be sufficiently synchronised across all detector components. The implications of this for the FEB firmware are explained in section 4.4.2.

Further tasks of the common FEB firmware include general operational aspects which are not directly related to detector readout or configuration. However, they are still very relevant since they provide backup and safety mechanisms, a boot sequence and other important functionalities. Since those topics are not immediately necessary for detector operation, they will be postponed to section 4.11.

4.4.1. Clock Domains of the ArriaV FEB Firmware

The ArriaV FPGA on the FEB operates – for the most part – on the three clocks shown in table 10. The first clock domain of 156.25 MHz is operating at the link frequency of the connection to the SWB. This is a result of the maximal transmission frequency of the ArriaV FPGA (6.25 Gbit/s) and the choice of an 8b10b-encoded interface width of 32-bit between the FEB and the SWB. All communication with the SWB naturally happens at that frequency.

The extent of this clock domain has then been increased to include all of the slow-control system, monitoring and ASIC configuration tasks. This is a natural choice

since these areas need frequent bidirectional communication with the SWB and a crossing to slower domains would limit the available bandwidth to do so. However, other domains of the firmware also need access to monitoring and control information from the SWB. The necessary domain transition into the 156.25 MHz domain happens at the endpoints of the slowcontrol system at each individual access point. The reasons and possible alternatives for this decision will be explained in section 4.4.4.4.

156.25 MHz	125 MHz	50 MHz
communication to the SWB	ASIC readout, decoding	board infrastructure
slowcontrol system	detector datapath	safety and backup sys.
ASIC configuration	sorting	boot procedure
system monitoring	detector synchronisation	disconnected operation
	timestamping	NIOS processor

Table 10.: Important clock domains of the ArriaV FPGA on the frontend board and the system areas that they are operating.

The 125 MHz domain is responsible for hit data coming from the detector ASICs. The MuTrig and MuPix were designed to send data words to the FEB with a frequency of 125 MHz. Therefore, the decoding and first processing steps also happen at this frequency. At some point this needs to be translated into the 156.25 MHz domain for transmission to the SWB.

Apart from data processing, the 125 MHz clock also serves as a global time reference for the Mu3e detector and is responsible for timestamping and detector synchronisation processes. Strictly speaking, multiple 125 MHz domains exist which are all based on the same clock but with unknown shifts relative to each other. This results in issues for detector synchronisation, which will be discussed in detail in section 4.4.2.

The 50 MHz clock domain is driven by a quartz oscillator located on the FEB. All other clocks are externally provided to the FEB in some way. Consequently, the clock is used in the firmware to provide operations that should function even when the FEB is completely disconnected from the rest of the DAQ system. Safety and backup systems, as well as the boot procedure, fall into this category. Additionally, the domain operates an embedded softcore processor (NIOS).

4.4.2. Clock and Reset Distribution

With the exception of the 50 MHz domain, the reference clock for the FEB is provided by the clock and reset distribution system. The critical task of this system is to synchronise all detector components of Mu3e. The time resolution of these components is an important factor for particle track reconstruction since a better time resolution reduces the number of hits that reconstruction algorithms have to search through, which is highly relevant at the particle rates expected for Mu3e. Sufficient time resolution also enables the use of time of flight (ToF) measurements, for example,

to distinguish clockwise and counterclockwise tracks in the central Mu3e station and identify them as positrons or electrons.

To make use of the time resolution of the Mu3e detectors, the timing information for each individual particle detection has to be measured relative to a global time reference. Providing this global time reference with the required accuracy is what we will refer to as synchronisation. Since the best time resolution in Mu3e is at around 70 ps, the synchronisation accuracy needs to reach values below 70 ps in order to preserve this resolution relative to other detector components. This is the task of the clock and reset distribution system.

Parts of the system have already been introduced by the author in the masterthesis [51] preceding this work. This section will draw from there and include updates on recent developments.

The Mu3e clock and reset system distributes a clock and a reset signal. These will be used to run and reset counters in the detector ASICs and on the ArriaV FPGA on the FEB. The counter is then used as a time reference (*timestamp*) for particle detections and other events in the DAQ.

On the detector ASICs (MuPix and MuTrig), the counter has a limited amount of bits and will occasionally overflow. The time information lost there needs to be added again as the data flows through the system. This will be explained once the discussion follows the path of the data starting from section 4.7.

The advantage of providing the time reference as a clock and reset signal is that the precision of the synchronisation can depend on the precision of the clock signal instead of the precision of the reset. This is achieved by resynchronising the reset signal to the clock at each receiver. The reset signal is then not the actual reset of the counter or parts of the detector but is instead used to identify the clock edge on which the actual reset to these components is issued. As long as the distributed reset always identifies the same clock edge in the complete system, the precision of the actual reset is a result of the precision of that clock edge.

Precise, low-jitter clock systems can be built from commercial off-the-shelf components. Similarly to the clock distribution networks in FPGAs (see section 3.3.1.2), a low clock skew between different detector parts can be achieved by using the same cable lengths for all clock lines, which is the approach taken by Mu3e.

Equivalently to the connection to the SWB, the clock and reset distribution to the FEBs also uses optical fibres to remove any grounding considerations from the system. The clock is an optical 125 MHz signal and the reset is a 8b10b encoded 1.25 Gbit/s link with an interface witdh of 8 bit. The parallel frequency of the reset link is identical to the 125 MHz clock line and synchronised to it since both are generated from the same device.

The clock and reset distribution lines originate at the so-called clock box (figure 55), which is located outside of the experimental area in the Mu3e server room. It contains a GENESYS2 board with a Kintex-7 Xilinx FPGA, where the clock and reset link is generated. The MIDAS DAQ system can control the GENESYS board via an ethernet connection. It is connected to the clock distribution board, generating 144 active optical copies of the reset and clock line.

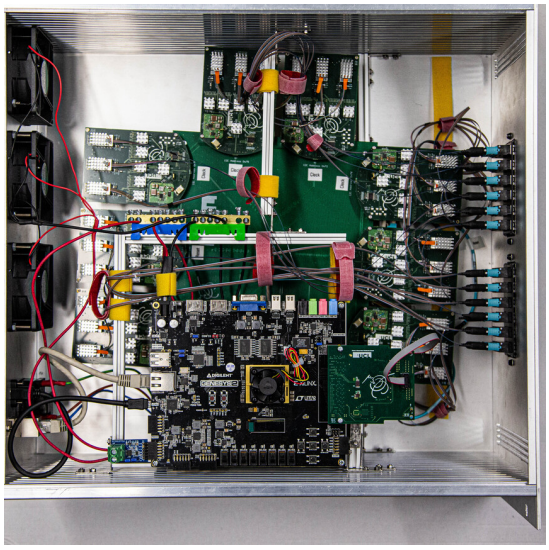


Figure 55.: Picture of the clock distribution box.

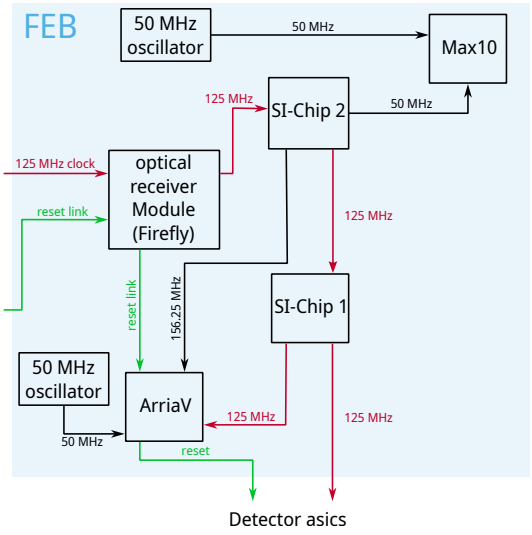


Figure 56.: Used clock and reset lines on the FEB.

Those are then individually connected to each FEB by two about 100 m long optical fibres and received on the FEB by a bidirectional ECUO-B04 firefly receiver module [57]. From there, the 125 MHz clock is forwarded through two clock distribution chips [58] to the ArriaV FPGA and the detector ASICs. Along the way, multiple other clocks are derived from the 125 MHz reference clock. The reset link is forwarded directly to the ArriaV FPGA, resynchronised there to the 125 MHz reference and used to drive the actual reset signal of the detector ASICs. The 125 MHz clock for the detector ASICs is not provided from the ArriaV since using dedicated clock distribution components results in a lower clock jitter. A measurement of an FPGA output jitter is shown in figure 57. The used clock chips are rated with an output jitter of 90 fs [58].

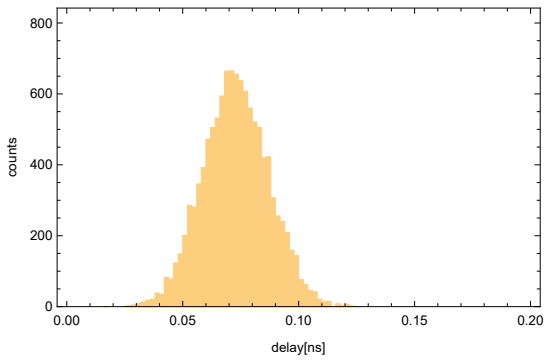


Figure 57.: Example of jitter on an FPGA output signal. The delay was measured against a reference clock. Jitter contributions from the oscilloscope are negligible.

Layers 2 and 3 of the DAQ are also provided with the same optical 125 MHz reference clock. However, since individual particle detections have already been assigned a global timestamp at this point, no reset needs to be provided and the reference clock is only used to drive the communication interfaces with the FEBs. A PCB with the dimensions of a PCIe card to receive the clock in the servers of DAQ layers 2 and 3 was developed in [59].

4.4.2.1. Reset Protocol

As mentioned before, the reset line between the clock distribution box and the FEB is implemented as an 8b10b encoded 1.25 Gbit link. The naive alternative to this would be to use a single-bit signal with the two states "reset active" and "reset not active". From a design point of view, this route was not taken since the implementation as a data link gives the opportunity to use the reset link for more than just one type of timing-critical operation. From a technical point of view, it turned out later that implementation as a simple two-state reset wire was never possible in the first place since the used optical transceivers are unable to transmit a constant 1 or constant 0 towards the frontend board[7].

Therefore, the reset was implemented using 8-bit commands, which can be used to cycle through a set of possible states of the FEBs. These 8-bit commands and their payloads are controlled by a MIDAS frontend and sent to the clock and reset distribution board via ethernet. The possible FEB states and the commands to change between them are shown in tables 11 and 12.

State	Comment
Normal operation	
`Idle`	
`Run Prepare`	
`Sync`	Resets active
`Running`	
`Terminating`	
Hard Resets	
`Reset`	
Others	
`Out of DAQ`	

Table 11.: FEB system states [60]

Command	Code	Payload
Run Prepare	0x10	32 bit run
Sync	0x11	-
Start Run	0x12	-
End Run	0x13	-
Abort Run	0x14	-
Reset	0x30	16 bit mask
Stop Reset	0x31	16 bit mask
Enable	0x32	
Disable	0x33	
Address	0x40	16 bit addr

Table 12.: Reset link protocol [60]

In a normal run sequence, the FEB cycles through the states `idle`, `run prepare`, `sync`, `running` and `terminating` before the `idle` state is reached again. During

[7] This was observed during the development of a system variation of the Mu3e DAQ and will be briefly discussed in chapter 5.

the `run prepare` state, all FEBs have to acknowledge that they are ready to start a new data-taking run[8]. When this is the case, the MIDAS software will send out the sync command, which instructs the FEBs to raise the resets of the connected detector ASICs until the `sync` state is exited with a start run signal. With the arrival of this signal, all timestamp counters in Mu3e are supposed to start synchronised counting.

A run is stopped with the end run signal. The frontend boards will move into the terminating state and continue processing detector data until the controlling entity receives permission to end the run from other firmware components on the ArriaV. This will return the FEB back into the `idle` state.

The additional states `Out of DAQ` and `reset` are not used during regular operation. The `Out of DAQ` state can only be entered from the idle state and allows to take a FEB out of the data acquisition system. FEBs in this state do not participate in data acquisition runs but are still powered and can be reinserted into the DAQ with the enable command. The `reset` state is used to hard-reset parts of the FEB firmware and is unrelated to the task of detector synchronisation. Which parts of the FEB firmware are reset in this state can be controlled via the payload.

All commands in table 12 can also be transmitted with a target address. This requires the address command and the FPGA ID of the frontend board to be transmitted right in front of the intended command.

Implementation details on the state controller which operates these FEB state transitions can be found in the masterthesis preceding this work [51]. The states `Sync test` and `Link test` mentioned there are not used in Mu3e anymore.

4.4.2.2. FEB Reset Resynchronisation

When the reset signal arrives at the receiver on the ArriaV FPGA, the receiver recovers a clock from the serial input data, divides this clock down to the appropriate frequency of the parallelised output signal and uses it to drive the parallel output data, which is then decoded according to the protocol explained in the last section. The reset line operates at 1.25 Gbit/s and has an 8b/10b decoded 8-bit wide interface. The recovered parallel clock is, therefore, running at 125 MHz. This clock is identical to the 125 MHz reference clock since both signals come from the Kintex-7 FPGA in the clock distribution box. However, there can be a phase shift between them. The issue is that this phase shift is initially unknown.

The phase shift is a combination of two factors. The first one is a possible cable length difference between the distribution of the 125 MHz clock and the reset to the ArriaV FPGA. For a correctly built Mu3e DAQ, this is only the difference in trace length on the FEB PCB since the optical cables are supposed to have an identical length. The other factor is that the receiver has ten possibilities for recovering the 125 MHz clock since there are ten rising edges of the fast 1.25 GHz clock available to which the phase of the slow 125 MHz recovered clock could be aligned to.

The result is the existence of two 125 MHz domains. One is the reference domain

[8] How they do that will be discussed later.

on which large parts of the FEB firmware operate and from which the reset needs to be driven to the detector ASICs. The second 125 MHz domain was recovered from the reset link, has one of ten possible phase relations and contains the state machine which provides the FEB state which is supposed to control the reset.

The problem is that the FEB state needs to go from there into the reference domain and none of the previously discussed methods for clock domain crossings (CDCs, section 3.7.4) can be used to do it. CDC methods inherently do not guarantee the exact cycle in which the data arrives in the destination clock domain. For example, the FEB state could be converted into a one-hot encoded state vector and a synchronisation chain could be used to move this to the reference domain. For each individual bit of the vector the synchronisation chain only decreases the possibility of metastable results at the destination and does that by adding more possibilities for the contained registers to resolve their output in either direction.

If the source and destination domain have a phase relation where the timing requirements for the first register in the synchronisation chain are fulfilled, the arrival cycle at the destination domain is deterministic and the synchronisation chain does not really serve a purpose. In the case where timing requirements for the first register are violated, the synchronisation chain does serve a purpose, but the cycle of arrival at the destination is not deterministic.

The timing of state changes of all FEBs in Mu3e needs to be deterministic in order to reset and synchronise the detector components. Therefore, we need to fulfill the timing requirements in the new clock domain without knowing the phase relation of source and destination clock. Since this is impossible, information about the phase relation needs to be acquired and needs to be used to enforce timing requirements.

The first step to do so is to implement a method to measure the phase relation in firmware. The second step is to produce a fixed phase relation of the recovered clock relative to the word boundaries as they arrive on the FEB via the serial reset line. In a last step, the reference clock can be shifted to a phase relation relative to the recovered clock where the timing requirements are fulfilled.

4.4.2.3. Phase measurement

A phase measurement entity was implemented to measure the phase between the recovered 125 MHz clock and the global reference clock in the ArriaV FPGA. The idea was already introduced in the masterthesis [51] preceding this work. The entity makes use of the 50 MHz clock from the oscillator on the FEB shown in figure 56. This clock is running independently from the 125 MHz recovered and reference clock domains. Slight frequency variations will randomly distribute the 50 MHz clock edges relative to both 125 MHz domains since the 50 MHz clock is produced by a separate oscillator and does not originate from the clock distribution system. These imperfections of the 50 MHz oscillator can be used to measure the phase relation.

On each rising edge of the 50 MHz clock, the two 125 MHz clocks will be sampled and compared. If their current value is not equal to the value of the other clock, a counter is increased (blue regions in figure 58). This is done for a measurement time

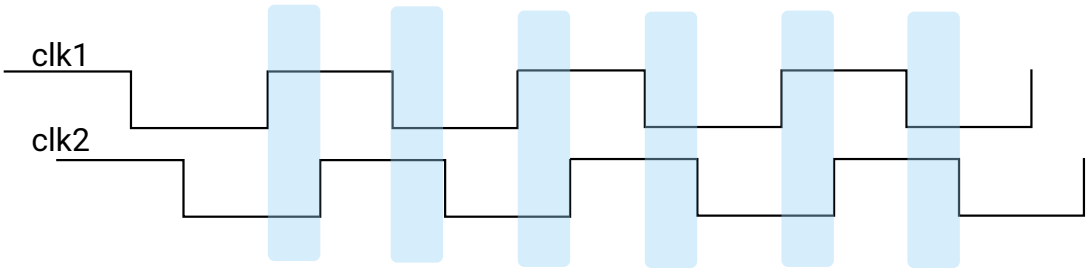


Figure 58.: Working principle of the phase measurement between clk1 and clk2.

T and then the value of the counter is compared to the total amount of rising edges of the 50 MHz clock in the time T. The phase difference between the two 125 MHz clocks clk1 and clk2 can then be calculated using:

$$\text{phase difference} = \frac{counts}{T \cdot f} \cdot \pi \tag{11}$$

where f is the frequency of the clock from the independent oscillator (50 MHz).

The results of test measurements in hardware are shown in figure 59. With the description given above, the entity is only able to measure the absolute value of the delay, therefore the y-axis contains only positive values. In principle, the resolution should only be limited by the measurement time, but in reality jitter, rise times and other things can have an influence on it.

The measurements in figure 59 form a plateau near a 4 ns delay on the x-axis. A part of the explanation for this is an increased duty cycle which was observed for the recovered clock in the setup where the measurement was performed. A simulation of the entity for different duty cycles confirms this effect on the measurement (figure 61). In addition to the duty cycle simulation also a simulation with jitter was written, which is shown in figure 60.

This entity can be used to determine the phase relationship between the two 125 MHz clocks. The method disregards all FPGA design principles from chapter 3. The 125 MHz clocks are used as logic signal inputs of ALM blocks and all timing requirements are completely ignored since there is no relation to the 50 MHz sampling clock. However, equation 11 still holds, and the resulting counts have to be interpreted as a statistical result. It is important to introduce a synchronisation chain between the count-up decision and the actual counter to avoid metastable counting. Otherwise, the counter can show quite erratic behaviour[9].

[9]For example, reach values of $counts > T \cdot f$ in the measurement time T

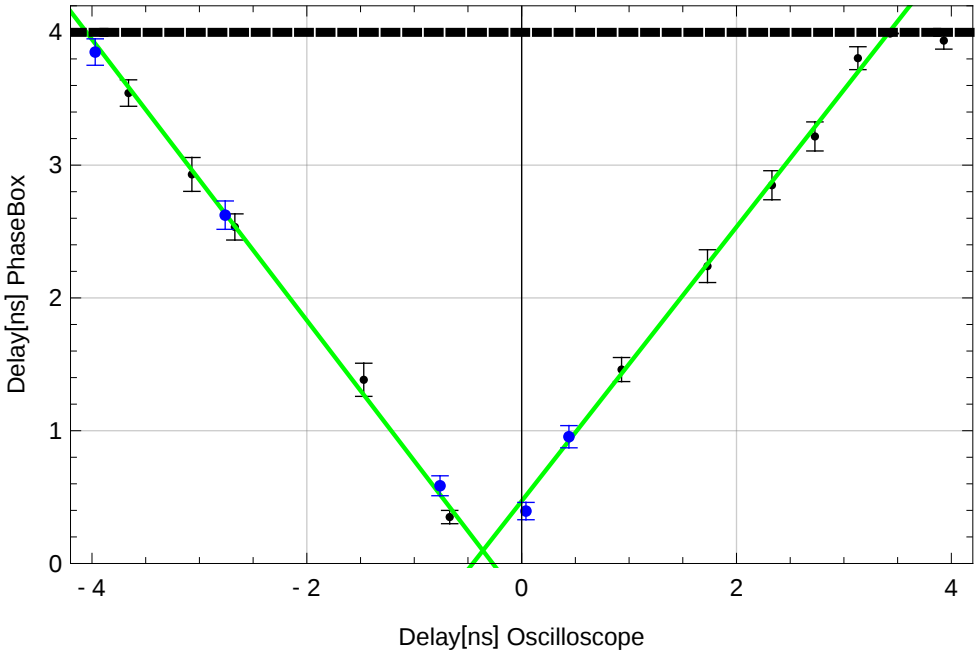


Figure 59.: Measurement results from the phase entity. On the x-axis the delay between a reference clock and the global clock measured by an oscilloscope is shown. The y-axis shows the mean of 15 delay measurements with the phase entity for a 50 MHz free running clock and a measurement time of 2^{26} counts. The point of intersection with the x-axis is of no meaning, since this was moved to 0 ns by cable lengths of the reference signal. Previously shown in [51].

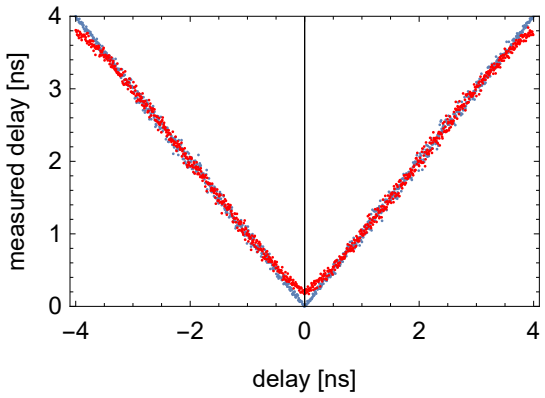


Figure 60.: Simulation of the phase measurement entity without jitter (blue) and with a 50 ps jitter on one of the input clocks(red). Previously shown in [51].

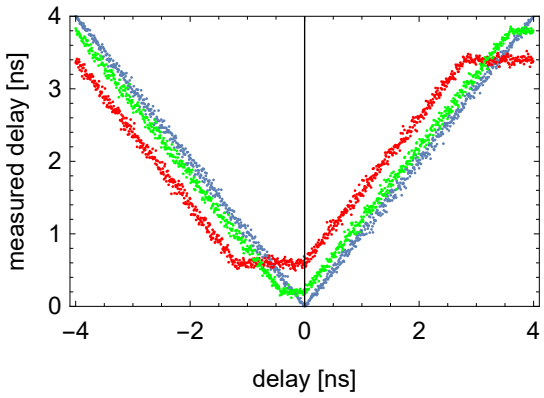


Figure 61.: Simulation of the phase measurement entity with different duty cycles of one input clock. 50 % (blue), 55 %(green), 65 %(red). The second input clock was on a fixed duty cycle of 50 % for all simulations. Previously shown in [51].

4.4.2.4. Ensuring a fixed phase relation

The ability to measure the phase relation alone is not useful when the result is still a random choice of one of ten possibilities for every restart of the FEB receiver. It is possible to remove this ambiguity when the receiver is operated correctly. The receiver has to be driven from the 125 MHz reference clock and the initialisation and reset procedure needs to precisely follow the instructions in [61]. All receiver resets, including the asynchronous ones, must be driven synchronously to the 125 MHz reference clock. Then, the word alignment has to be driven externally by user logic operating at the same reference frequency. After this procedure, the clock recovered by the receiver will have a fixed phase relation of relative to the word boundaries as they arrive on the FEB via the serial reset line.

The current version of the FEB uses LVDS receivers for the reset line. An older version used the fast transceiver IPs, where a similar way to enforce fixed phase relations exists. However, the issue remains the same. FEB state changes will have a fixed phase relation to the word boundaries of the serial input but not all FEBs will neccessarily realign these changes to the same 125 MHz reference clock cycle.

4.4.2.5. Enforcing timing requirements

Section 4.4.2.3 showed how to measure the phase relationship and the last section explained how to produce a deterministic phase relationship. The phase between the two 125 MHz domains is now reproducible and known. Therefore, timing requirements can be enforced between them with the goal of avoiding phase relationships where the synchronisation of the FEB state into the global 125 MHz reference domain can flip both ways and does not end up on a deterministic cycle.

The method to do that is based on finding the jump point in the phase relationship. The clock chips on the FEB shown in figure 56 of section 4.4.2 can be used to gradually shift the phase of the 125 MHz reference clock. At some point, FEB state changes will necessarily jump to a different clock edge of the reference clock. The phase at which FEB state changes can randomly jump between two edges is a phase at which the timing requirements of the first register in the global clock domain are violated. The current approach is to find that point and rotate the clock phase by 180° from there using the clock distribution chips.

Outside of the magnet and with physical access to the FEB, the jump point can be found with an oscilloscope. The result there can be translated to the FEBs in the magnet by using the phase measurement entity. In principle this has to be done only a single time once the final FEB firmware is connected in the detector cage with the final cabling. However, the FEB firmware will likely change a couple of times during Mu3e operation which might slightly change the optimal phase setting.

There are methods to fix the location of individual logic elements or entire design sections of the FPGA in order to predefine their timing behaviour. These methods were not discussed in the digital electronics chapter since there are only two occasions in the entire Mu3e DAQ where this could be useful for user HDL code. This is one of

them. It might be possible to predefine the exact implementation and location of reset resynchronisation parts of the FEB firmware in order to avoid changes in the optimal phase shift arising from compilation results for different FEB firmware versions. This idea has not been further pursued since its relevance is questionable at the moment.

4.4.2.6. Alternatives

Distribution of a clock and synchronisation of detector components is not a Mu3e-exclusive problem but is an important topic in all particle physics experiments. Many of them use high speed optical transceivers in their readout systems since the produced data rates often require such solutions. However, commercially available high-speed data transmission components are often designed for telecommunication purposes where deterministic and known latencies do not play as much of a role as in particle physics experiments.

For that reason, other experiments have implemented solutions for detector synchronisation issues similar to the ones discussed here. For example, in the white rabbit project [62] at CERN, synchronisation is achieved by measuring the delay of data packets which are sent in both directions between components. The timestamp and delay of these packets can then be used to synchronise timestamp counters.

Another approach is to use clock duty cycle modulation [63], which is the intention for the DAQ of the g-2/EDM experiment at JPARC [64]. The rising edge of the clock signal is used as time reference and the position of the falling edge is used to encode user instructions (figure 62).

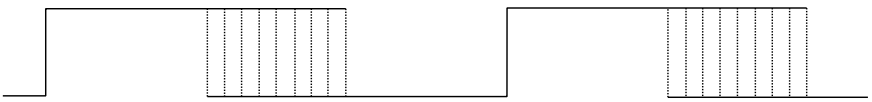


Figure 62.: Clock duty cycle modulation.

In such a system the issue of resynchronising the reset to the received clock does not exist since it is always clear to which clock edge a reset received via a duty cycle modulation belongs to.

There are also experiments which take a similar approach to Mu3e and use special configurations of existing transceivers for synchronisation. For example the GBT Project at CERN [65], which is using fixed latency transceivers [66].

The solution found for the Mu3e DAQ relies more on identical cable length than other ideas. However, this is not an issue for Mu3e since distances from detector components to the clock distribution are all quite similar due to the limited size of the Mu3e detector. If necessary, cable length differences could also be calibrated away by shifting the reference clock.

This section concludes the discussion of clock and reset distribution in Mu3e. The FEB state can be changed synchronised for all FEBs in the Mu3e DAQ, which will be used in the following sections to steer the actual detector ASIC resets and other operations such as run starts or stops.

4.4.3. Data merging

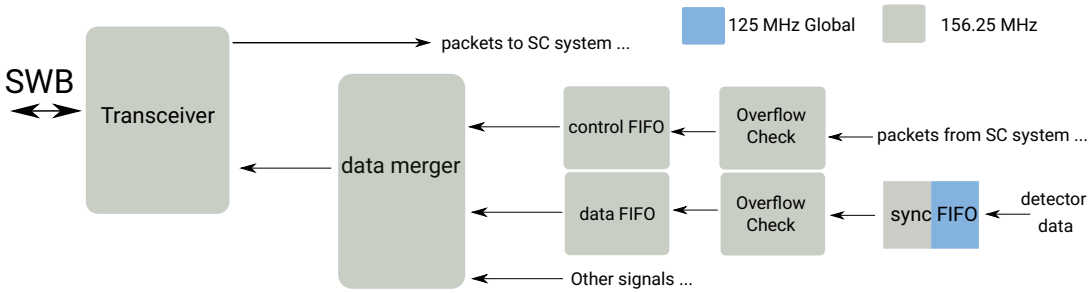


Figure 63.: Block diagram of the firmware components involved in the merging of data.

The optical connection from the SWB to the FEB consists of two directions. The direction towards the FEB is exclusively used by the previously discussed slowcontrol packets. For the other direction, mutliple types of data have to be transmitted over the same optical cable. The two most common types there are the data packets from the detector specific firmware and the slowcontrol packets responding to read/write commands from the SWB.

A data merger entity is connected to the outgoing optical link of the FEB to organise these different packet types which have to be transmitted. Data packets from the detetor datapath and the slowcontrol system are written into the data and control FIFOs in front of the merger entity where they wait for their transaction.

The packets in these FIFOs only contain the payload from table 6 in section 4.3.1. When a packet is read by the merger, the preamble and the trailer are added during transmission. In addition to the 32-bit width of the data words, 4 control bits are added to the width of the FIFO, which are used to identify the start and end of the packet in the FIFO.

The transmission of a packet from one of the FIFOs to the SWB can start as soon as the start of packet marker is seen in the 4 additional control bits. The data merger will then keep reading from the chosen FIFO until the end of packet marker is received. Should the FIFO run empty before receiving the end of the packet, idle words will be transmitted to the SWB to fill the gaps in the datastream until the next word of the packet arrives at the FIFO output. When the end of packet marker is received, the packet transmitted towards the SWB will be closed with a trailer according to the previously discussed protocol and a new FIFO will be selected for transmission.

The prioritisation of packets depends on a priority setting and the FEB state from table 11. During the states `Idle`, `Run Prepare` and `Out of DAQ`, only packets from the slowcontrol FIFO will be accepted. During the states `Sync` and `Reset`, no new packets are accepted, but transmissions which are ongoing when the state is entered will be finished. When the FEB is in the state `Running` or `Terminating`, a priority setting decides which of the two FIFOs is selected first. The other FIFO is then only selected once the FIFO with priority is empty.

On the write side of the two buffer FIFOs an overflow check is performed. When

the FIFO is filled at a level of 90% or more, incoming packets will be thrown away. The decision to throw a packet away is always made at the first word of the packet and then valid until the packet ends. This was implemented as a precaution against packet corruption.

However, since the detector datapaths are operated at a lower frequency of 125 MHz compared to the 156.25 MHz domain of the transmission towards the SWB, packet corruption should only be possible if the slowcontrol system is able to completely occupy the remaining bandwidth of $(156.25\ \text{MHz} - 125\ \text{MHz}) \cdot 32\ \text{bit} = 1\ \text{Gbit/s}$. As we have seen in section 4.3.2.2, the slowcontrol system has a bandwidth limitation on the SWB firmware of 238 Mbit/s and should, therefore, not be able to cause bandwidth issues on the FEB for extended time periods. On short timescales, the FEB bandwidth might be insufficient, but this can be resolved with an adequate size of the buffer FIFOs in front of the merger.

4.4.3.1. Run Control

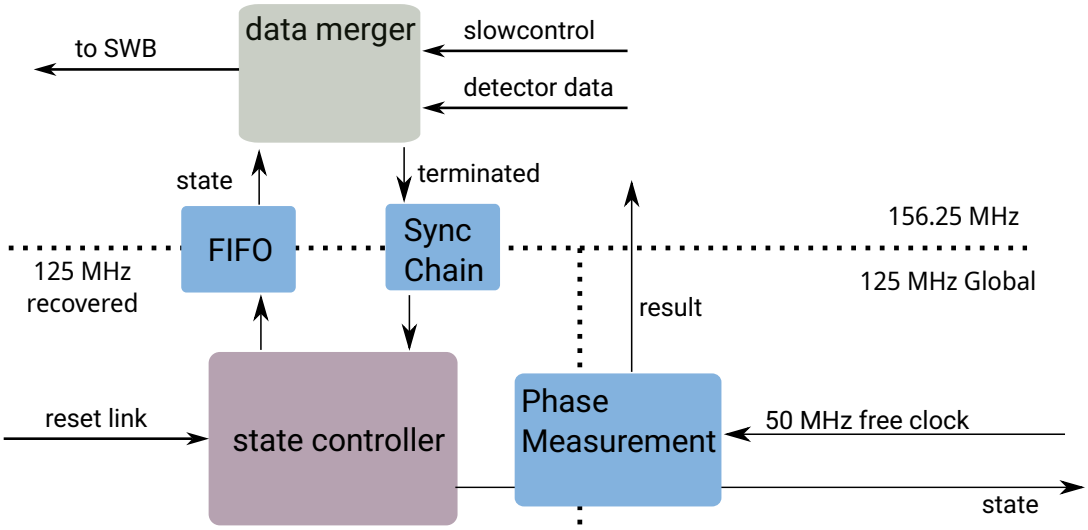


Figure 64.: Run control and reset distribution.

In addition to the task of merging slowcontrol and detector data packets into a single datastream, the merger also plays a role in the run start and stop procedure. It is involved there by sending run control signals to the SWB. These are single 32-bit words containing a unique, otherwise unused 8b10b comma word and a 24-bit payload. They can be sent completely independent of the previously introduced protocol and will be inserted by the merger between other packets. On the side of the SWB, these run control words will be removed and processed separately from the datastream in the data demerger shown in figure 52 of section 4.3.2.

Also involved in the run start and stop process are the clock and reset system from section 4.4.2 and the switching board firmware. Those two components are controlled

by two MIDAS frontends (see section 4.2) called crfe and switch_fe. A normal run start and stop procedure in Mu3e goes through the following steps:

1. The run start in MIDAS triggers a function in the switch_fe, which requests a run start command from the crfe via a watched ODB variable (as described for communication between MIDAS frontends in section 4.2).

2. The crfe sends the run start command, including the run number N, to the clock and reset distribution box via ethernet, which forwards this over the optical reset line to all FEBs (section 4.4.2.1).

3. The state controller on the FEB firmware (figure 64) decodes the command and changes the local run number to N and the FEB state into `Run Prepare` (section 4.4.2.1).

4. The data merger acknowledges or denies readiness to start run N based on information from the subdetector specific firmware. It does that by sending a run control word, as mentioned above, with the run number as a payload.

5. The SWB firmware creates a list of FEBs which have correctly acknowledged the start of a run with the number N.

6. The switch_fe reads this list and compares it against the expected list of FEBs from the ODB. If the lists match, the switch_fe concludes its run start function.

7. The crfe sends the command sync to the FEBs, which change their state into `Sync` and raise the reset signal for the detector ASICs.

8. The crfe sends the command start run to the FEBs, which change their state into `Running` and release the reset of the detector ASICs.

9. Data taking until the crfe issues the stop run command to the FEBs.

10. When the FEB receives the stop run command, it changes its state to `Terminating`, stops accepting new data at the connection to the detector ASICs, but continues to send packets towards the SWB until all the data in the pipeline has been processed.

11. The data merger acknowledges the run end with another run control signal and indicates towards the state controller that FEB state can be changed back into `idle`, which is then executed by the state controller.

12. The SWB records all the FEBs which have stopped the run. This is then read by the switch_fe which allows the MIDAS software components to conclude the run.

Other run control signals are used to indicate protocol errors to the SWB. For example, when a data packet is not closed by the slowcontrol system or detector

firmware after a predefined amount of cycles, the data merger will close the packet on its own and send a timeout run control signal to the SWB. This can be processed on the SWB since it is always possible to identify this error signal due to the usage of a 8b10b comma word.

4.4.4. FEB slowcontrol

The FEB slowcontrol system is the part of the FEB firmware which receives slowcontrol packets from the SWB. As mentioned before, communication with a FEB is based on addresses on which read/write actions can be performed. The slowcontrol system has to execute the read or write instruction contained in received sc packets and has to send a reply packet towards the control FIFO in front of the data merger in figure 63. From there, reply packets will be forwarded to the SWB and integrate into the previous discussion at the data-demerger entity in figure 52 of section 4.3.2.

4.4.4.1. Slowcontrol Receiver

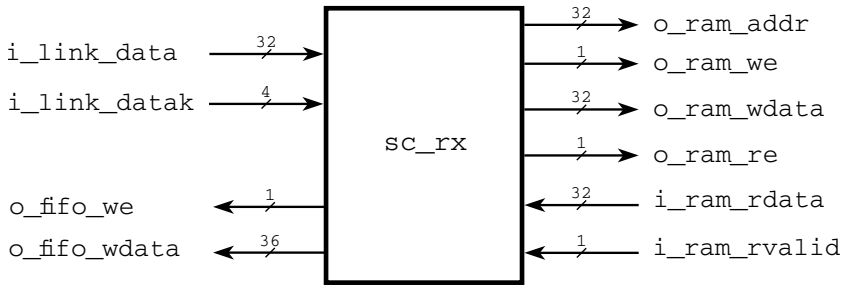


Figure 65.: Ports of the sc_rx entity.

In order to implement this, the incoming slowcontrol packets from the SWB link are first converted into an avalon agent (section 3.8.4) compatible interface by the slowcontrol receiver (sc_rx) entity. A write packet for example will apply the write enable signal (o_ram_we) and cycle through the target address range while setting o_ram_wdata to the according write data contained in the sc packet.

For non-incrementing writes, the output address will be held constant instead. Therefore, the differentiation between incrementing and non-incrementing packets is also resolved here and is not relevant anymore for other firmware components.

Read packets will act similar on the output address, but apply the read enable (o_ram_re) signal instead. The entity will wait for the arrival of the reply data and construct the reply packet to be send towards the control FIFO of the data merger.

The arrival of each data word is indicated by the downstream component with the valid signal (i_ram_rvalid). The writing of reply data to the merger FIFO is delayed accordingly if the read data does not arrive at the sc_rx entity in an uninterrupted sequence. However, the data merger might already begin transmission of the reply packet towards the SWB with the available data. In this case, a delay of the valid signal

at the sc_rx entity can cause transmission gaps in the FEB/SWB communication, which will be automatically filled by the merger with idle words.

In consequence, data packets from the detector readout in the other merger FIFO need to wait until the sc_rx entity has received the valid signal for all of the data requested by the slowcontrol read packet. This can become an issue. Buffering of packets is, in principle, the function of the merger FIFOs. However, buffering of detector data is problematic since the detector ASICs will produce significant amounts of data during operation and the FEB does not have the resources to store all of it for extended time periods. Buffering of slowcontrol packets is much simpler since the software on the SWB will need the reply packet before it issues another instruction[10].

This could lead to packet loss on the detector readout path, which, as we will see later, has to be avoided for the DAQ layer 2 firmware to operate properly. Therefore, the next section will discuss how the sc_rx entity can be provided with an uninterrupted reply datastream.

4.4.4.2. Slowcontrol RAM

The slowcontrol RAM (sc_RAM) entity provides an access interface to the range of FEB registers and other resources, which steer and monitor all functionalities on the frontend board and the connected detector components.

Towards the components on the left side of figure 66, the entity provides multiple avalon agent compatible memory interfaces. The entity in itself does not contain random access memory but connects to many components on the right side of figure 66, the aggregate of which acts like a random access memory. As we have discussed previously, not all addresses in this constructed RAM will behave exactly like memory. Some addresses might not be writeable, others might be the write port of a FIFO or might cause a shutdown of the detector whenever they are accessed. The action depends on whatever the components on the right are configured to do when their slowcontrol addresses are read or written.

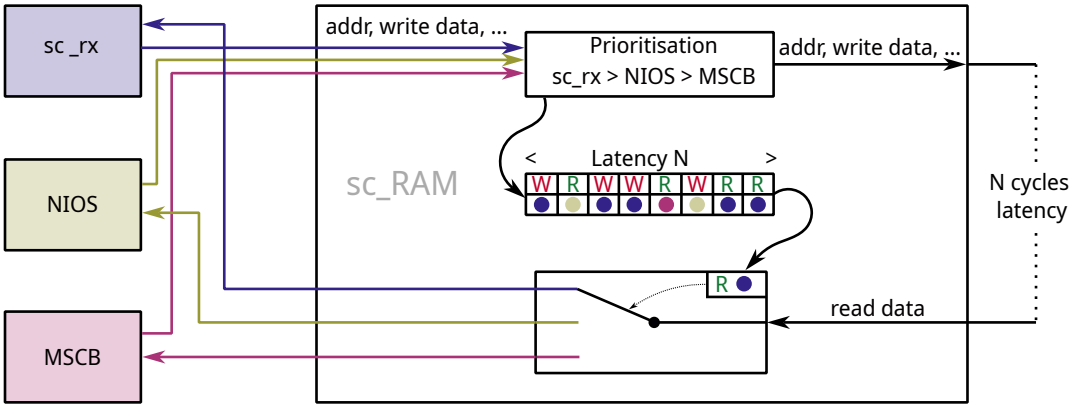


Figure 66.: Concept of the sc_RAM entity

[10]With a couple of exceptions which have been discussed in section 4.3.1

Read and write commands from the avalon compatible hosts on the left are prioritised and forwarded to components on the right. Those are required to reply to read commands within a latency of exactly N cycles. Since the latency is exactly defined, every clock cycle can be used to send commands towards the left. The reply to a command will arrive back at the sc_RAM entity N cycles later. There, a shift register with length N contains an identifier for the host and type of command (read or write) and allows the sc_RAM entity to keep track of the instructions and route the reply data towards the correct host after N cycles.

The sc_rx entity from the previous section is one of the hosts connected to a memory interface on the left. As discussed in the last section, the sc_rx entity only implements a valid signal and does not implement the usual avalon waitrequest. Therefore, it is not possible to slow it down from the side of the sc_RAM.

This is intentional and does not cause a problem since the sc_rx entity has the highest priority in the sc_RAM entity. Other hosts implement the avalon waitrequest signal and can be stalled in their current transaction at any time and indefinitely if it is needed to serve the sc_rx entity. This ensures that the sc_rx entity can always be provided with the full write and read bandwidth with a fixed read latency.

This has the important consequence that there is no backpressure needed towards the SWB soft- and firmware. A instruction can be sent from the SWB to the FEB at any time and the FEB will have the resources to execute it, independent of all other operations.

For the other hosts, reply times can vary depending on currently ongoing interactions with the SWB. The hosts NIOS and MSCB shown in figure 66 will be discussed in the next section.

4.4.4.3. NIOS and MSCB

The NIOS is a processor provided by Intel as softcore IP for the use on their FPGAs. It is used on the frontend boards to allow them to run software programs. The NIOS has the same access possibilities through the sc_RAM entity as the MIDAS software via the optical communication. It can read and write all slowcontrol addresses on the FEB.

Apart from the possibility to execute software on the FEB, this has the advantage that it can operate independently of MIDAS. This is useful in situations where the corresponding MIDAS frontends are not running or are unable to reach the FEB. For example, a user can operate all FEB functionalities via a USB cable connected to a standalone FEB outside of the Mu3e DAQ. The NIOS on the FEB operates this terminal and allows direct user interaction without involvement of other software or hardware. This is not relevant during operation of the actual Mu3e detector, but very useful for development and debugging.

However, the NIOS operates at a frequency of 50 MHz and the speed at which it can perform operations on the FEB is not comparable to the actual slowcontrol system. Other applications of the NIOS in Mu3e will follow in sections 4.4.4.6 and 4.11.

MSCB

The MIDAS SlowControl Bus (MSCB) interface is used to communicate with the FEB via the backplane in the service support wheel. It is intended as a backup solution for the optical connection to the SWB. If the optical connection fails for some reason, it is still necessary to get some critical information in and out of the detector.

MSCB is a slow serial bus which is fully integrated into the MIDAS slow control system. There is an optical MSCB bridge from the service support wheel to the outside of the Mu3e magnet, where the connection to MIDAS is made via an ethernet adapter. Apart from MIDAS, no additional software is needed to operate it.

An MSCB device contains a list of variables which can be accessed directly via MIDAS. Each device will self-document these variables into the MIDAS online data base upon connection. This means that all available variables of this device, their names, current values, length and also their unit are transmitted to MIDAS when the node is initially connected to MSCB and these informations can be shown and edited in the MIDAS web interface without any device specific driver software on the PC. Further information about the MSCB protocol and devices can be found in [67].

There are two independent MSCB connections for the frontend boards in Mu3e. One of them is made between the Max10 FPGA and the backplane. This is the actually necessary backup solution. It is important in the context of firmware upload, which is a separate infrastructure problem that is not relevant for the current line of discussion. It is therefore postponed to sections 4.11.3 and 4.11.4.

The other MSCB connection is the one referred to by the entity on the left of figure 66 and it connects the ArriaV FPGA with the backplane. Originally, the MSCB line was operated completely in software by the NIOS. This functionality was removed since it required too much memory resources in the NIOS[11].

A firmware implementation of an MSCB device was written and tested in [68]. This could in principle be used to implement the MSCB host for the sc_RAM entity in figure 66. However, this part doesn't exist yet since a backup communication with the ArriaV on the FEB has not become relevant at this moment. Should this be implemented, it will likely just use the signals of the interface to the sc_RAM entity as MSCB variables[12]. This would provide a very slow but full second communication channel to the FEB with identical functionalities as the optical slowcontrol.

[11] In the sense that the amount of memory blocks which need to be assigned to the NIOS processor for it to be able to operate MSCB is too large. It is possible to operate MSCB in software running on a NIOS (that is actually done on the Max10) but the ArriaV does not have memory resources available to do so since they are required by the rest of the firmware. The reasons for the tight memory situation will become clear in later sections.

[12] Using the actual FEB slowcontrol registers seems difficult since there are too many of them.

4.4.4.4. Slowcontrol Tree

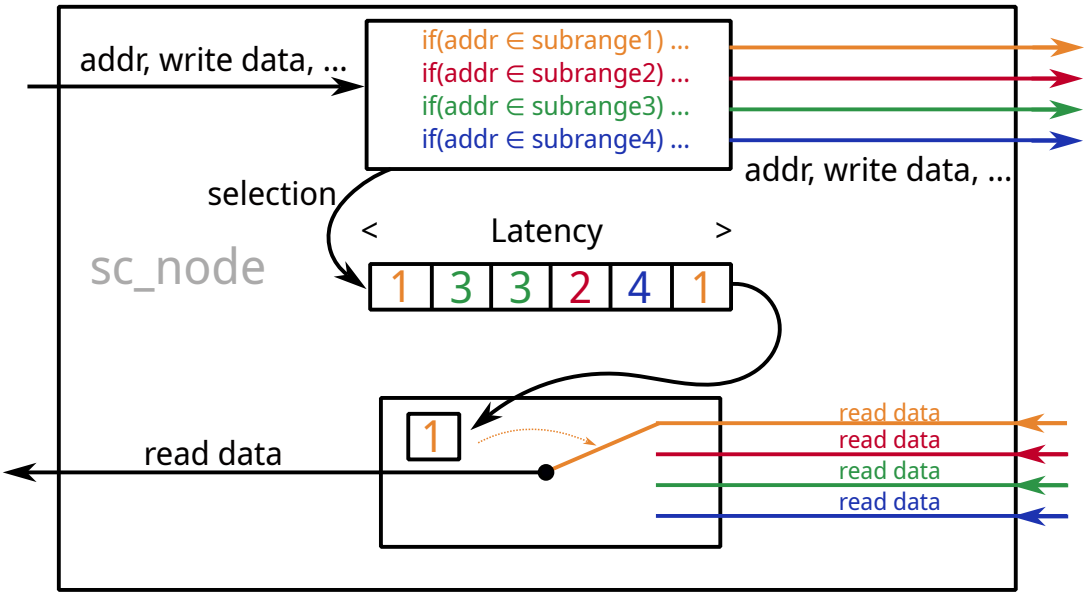


Figure 67.: Implementation concept of a slowcontrol node.

The connection between the left side of figure 66 and actual slowcontrol resources is done through the slowcontrol tree. This component was already used as an example for timing closure solutions in the digital electronics chapter (section 3.7.6).

The different slowcontrol endpoints – such as slowcontrol registers, FIFOs, memories and other resources with a slowcontrol address – are scattered randomly all over the design of the FEB firmware. They are now supposed to be accessed by the sc_RAM entity at a frequency of 156.25 MHz.

In the earlier development stages, direct access was possible here. As the system continued to grow, more and more slowcontrol resources were added until timing closure was not possible anymore at this frequency. The access latency shown in figure 66 was introduced to buy more time. The gained additional clock cycles are used to divide slowcontrol commands into individual target address subranges (top part of figure 67). Each split into address subranges costs one cycle of latency and the corresponding logic resources but can also reduce the number of target addresses in each subrange by a factor of four. This method was then further explored since it only introduces latency and does not reduce available slowcontrol bandwidth.

Each split is done in a sc_node entity. Multiple of these entities connected together form a tree-like structure as shown in figure 35 in chapter 3. In addition to a reduction of the fan-out, the split into address subranges might also allow the fixing of certain bits in the address signal if the range is defined correctly. If these bits of the address are identified as constant by the compiler that can have further positive timing effects.

In general, the subranges are defined in a way to serve logical sections of the FEB

firmware. A very early split – for example – is the split between detector specific and common FEB firmware. The detector specific address subrange will then further split into detector control components and detector readout components. This is intended to allow better physical separation between slowcontrol endpoints. Firmware for certain functions tends to occupy a compact space when mapped to the actual resources on the FPGA. Using individual slowcontrol subranges for these functionalities should allow the slowcontrol tree to also form a tree-like structure in terms of physical location on the FPGA, which can reduce signal distances. Appendix B.1 shows the distribution of slowcontrol endpoints across the ArriaV FPGA. It can be seen there that endpoints are able to spread over the complete chip independently without forming a single slowcontrol cluster.

The return path for the read data uses a similar concept as the sc_RAM entity. Each SC node needs to know its level in the tree and can use that to calculate the read latency at compile time. The deepest nodes will have a read latency of one cycle, which will increase towards the sc_RAM entity. The read latency at the sc_RAM entity results from the number of levels in the tree. Similar to the implementation there, the sc nodes will keep track of the selected subrange in a shift register and will connect the according return line exactly after the latency period. Every clock cycle can be used, and the full bandwidth is available for every address in the tree.

It is possible to configure the tree with differing depths. In a case where one subrange has a latency different from another subrange of the same sc_node, the sc_node can be configured to delay the return data by the difference. All these delay and latency calculations are done automatically at compile time. The information provided to each instance of an sc node is just the depth of the following tree for each connected subrange.

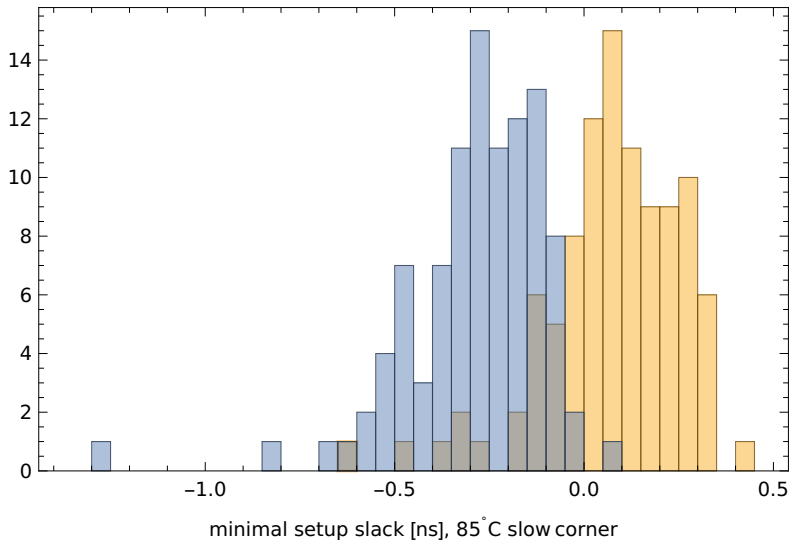


Figure 68.: Minimal setup slack histograms for before (blue) and after (yellow) the introduction of the slowcontrol tree.

The introduction of the tree structure has led to significant timing improvements. Figure 68 was already shown in the digital electronics chapter and was actually displaying the difference in setup slack histograms after the implementation of the slowcontrol tree and the read latency. Without these changes, timing closure is highly unlikely and the FEB will probably not function correctly. Further details on this measurement and minimal setup slack histograms can be found in chapter 3. The new timing limits, which shape the yellow histogram in figure 68, are unrelated to the slowcontrol system and will be discussed at a later point.

The choice of splitting at each node into a maximum of four address subranges was based on the logic elements on the ArriaV FPGA having four signal inputs. Whether this number is actually the best choice in terms of resource usage and gained timing performance was not further investigated since the result already sufficiently achieves timing closure. This variable can be used as a starting point if additional optimisation is needed in the future.

This concludes the discussion of how the software control system is able to communicate with the frontend boards and detector components. We will now turn towards the content of this communication after a discussion of alternative architectures.

4.4.4.5. Alternatives

There are several decisions in the shown slowcontrol architecture where alternative solutions are possible. A few of those were discussed within the collaboration and the arguments for and against other solutions will be shown here.

The first point of discussion is the commitment to a fixed latency system. It would, in principle, be possible to introduce a waitrequest signal in the entire slowcontrol tree and remove the requirement for a fixed latency. This would have a few advantages. The slowcontrol nodes would not need to know their position in the tree, which reduces the possibility of user error when the tree is put together. When the individual latencies in the current system are not set correctly, collisions on the data return line can occur and data can be lost. Even worse, data can be interpreted as a response to a different read request.

The other advantage concerns clock domain transitions. In the version of the system presented here, domain transitions are not possible since they are incompatible with the fixed latency concept[13]. The slowcontrol endpoints need to ensure a safe transition to the 156.25 MHz domain at the place where they connect to the slowcontrol tree.

The use of a waitrequest or data-valid signal in the entire slowcontrol tree would make clock domain transitions within the tree possible, which removes the need for the individual endpoints to implement a CDC. However, many of the currently used single-word slowcontrol registers are safe to use in a false path configuration and slowcontrol addresses which end in a FIFO or memory are anyways less of a concern since one can use a dual clock FIFO or dual clock memory there.

[13] At least for unrelated clocks. For related clocks, one could think of a configuration where the required latency in the 156.25 MHz domain is matched exactly.

The first consequence of using a wait request would be additional resource usage. The slowcontrol nodes cannot base the selection of the return data on a fixed delay relative to the cycle where they send out the request anymore. This means that each reply necessarily needs a signal for the address which propagates with it back to the root of the tree. Otherwise it is not possible to match the read data back to the original request. This is not a concern in the current system since the order in which replies arrive at each point in the tree is identical to the request order due to the fixed latency requirement.

The other consequence is that a situation is created where it needs to be possible to slow down the sc_rx entity similarly to the NIOS since the time of arrival of reply data cannot be predicted. The issue is that the sc_rx entity potentially receives a stream of data at 156.25 MHz. The only point in the system where one could stop or delay this is all the way up at the slowcontrol main entity in the SWB firmware. It would require some form of backpressure protocol to be transmitted from the FEB to the SWB to achieve this. One could also try to buffer data on the FEB instead of implementing backpressure tp the SWB. However, this idea is based on the hope that the buffer will be large enough and was discarded for this reason[14].

Overall, concepts which preserve the bandwidth of the optical connection but allow CDCs within the slowcontrol system cause too many issues which impact the guaranteed delivery of slowcontrol commands. Write commands have a bandwidth problem that would require backpressure to the SWB. Read commands have a reply data collision and ordering issue on the FEB. The fixed latency system described here ensures packet delivery for read and write commands and CDC implementations at the slowcontrol endpoints seem necessary for this at the moment.

Another alternative idea was to base the complete slowcontrol system on an avalon memory mapped interface managed by the intel system integrator tool. Each slowcontrol endpoint would be implemented as an avalon agent and the sc_rx entity would act as an avalon host. This idea comes in principle with similar concerns for write commands as the one above. The read command issues would likely not be present here.

The reason why this was not further pursued is that an implementation based on the system integrator tool would affect the portability of the firmware. At the moment, it seems likely that parts of the firmware developed for Mu3e are going to be used in other projects. A few examples will be shown in chapter 5. A few of these projects will use different FPGAs for their DAQ system, including FPGAs from different vendors. The current implementation is entirely written in user-HDL code and can be easily copied to any other FPGA. A commitment to intel-specific tools would impact this portability. Another argument for a pure HDL implementation is the higher educational value for new students compared to a specific version of a vendor-specific system integrator tool.

[14] Memory usage on the FEB is also quite high and does not allow a very large buffer here.

Packets without Request

The alternatives above are all concerned with the implementation details of the protocol introduced at the beginning of this chapter. This protocol is based on a request and reply idea. A large fraction of slowcontrol communication with the FEBs is the readback of monitoring information. This information is usually read by the SWB software in regular time intervals and updated in the MIDAS ODB.

In principle, it would be possible for the FEB to send these monitoring updates automatically without receiving a request from the SWB. For example, the NIOS could regularly collect monitoring information from different slowcontrol endpoints in the tree and assemble a monitoring update packet with a predefined format. This could then be sent to the SWB where some firmware could be implemented to receive and process these packets. However, individual read and write requests would still be needed for other functionalities and currently there is no clear advantage in the introduction of unrequested slowcontrol packets.

4.4.4.6. NIOS RPC calls

A remote procedure call (RPC) can be used in the current slowcontrol system to trigger software functions on the frontend board NIOS from the MIDAS software. A part of the NIOS processor memory is connected with an address subrange in the slowcontrol tree discussed above. This memory can be accessed by MIDAS via the slowcontrol and, similarly, by the NIOS processor through the connection at the sc_RAM entity. To the NIOS, this appears as a normal part of its memory since the NIOS system internally also uses avalon memory mapped connections.

To trigger a function on the NIOS, the MIDAS frontend can write an instruction to a predefined memory location. Writing this memory location causes an interrupt signal to be sent to the NIOS by the FEB firmware, which instructs the NIOS to interrupt the currently executing code and process the instruction by MIDAS instead. Payload for this instruction has to be written by MIDAS into other memory locations first.

For the return of data from the NIOS to MIDAS, it has to be written into locations in the same memory subrange. Software on the MIDAS side can then poll these addresses and receive an answer from the NIOS.

The speed at which these kinds of transactions can happen is not very high since the NIOS operates on a 50 MHz clock. In most cases this is not the optimal way of steering FEB functionalities since the software on the NIOS in itself also just operates on the slowcontrol RAM again, which can also be accessed directly from the MIDAS software.

4.5. Mutrig configuration

Each MuTrig chip is configured via a single spi shift register with a length of about 3000 bits. This register contains the thresholds and all other settings for the MuTrig

ASIC. The data for this register is sent from the MIDAS ODB via an rp call to the frontend board NIOS. From there, the NIOS operates the spi line from software and writes the received data to the MuTrig ASIC.

At some point during the development of the Mu3e DAQ, both the MuPix and MuTrig configurations were based on rp calls to the NIOS. The MuPix configuration firmware has moved towards a much faster but significantly more complicated configuration architecture, which is steered from the SWB software without the involvement of the NIOS processor. However, the configuration of the MuTrig is still using RPC's to the NIOS at the moment but is likely to change to a direct configuration method due to speed considerations. At the time of writing this thesis, MuTrig chips have not been operated in large numbers in the same system. Configuration speed issues have, therefore, not played a large role at the moment.

4.6. Mupix configuration

Configuring a mupix is significantly more complicated than a MuTrig configuration. However, the underlying concept is the same. Shift registers contain the configuration for various settings within the ASIC. In a MuPix chip, six individual shift registers are used to control different functions. Three of them control global settings, such as the global thresholds, amplifier settings or digital settings for the readout statemachine. Two shift registers access the tune values (TDACs, discussed in section 2.2.1) for each individual pixel on the MuPix ASIC. The last shift register is used for characterisation purposes.

The registers for the global settings are comparable in their combined size to a MuTrig configuration. As mentioned before, NIOS rp calls were previously used to write them. This quickly becomes impractical once the number of MuPix chips in the system increases since the speed of the NIOS is limited and there is no option for parallelisation.

The tune values (TDACs) contain more than 1000 times more data than the global MuPix settings since tune values are provided per pixel. Overall, a MuPix configuration has a size of roughly 0.5 Mb, which adds up to a total of about 1.5 Gbit of configuration data for the complete pixel detector. The challenge is to provide a system which can efficiently deliver this data to the MuPix sensors, especially since tuning procedures to find optimal combinations of global settings and TDAC values will involve iterative searches.

The solution to this problem will involve firmware blocks, which are specifically designed to allow for parallel and independent MuPix configuration and are less reliant on software components. We will first discuss the interfaces provided by the MuPix chip and how they can be used to upload configuration data. Then, the firmware solution for the three global configuration registers will be presented. The design for the tune value upload will follow in section 4.6.3 and will use large parts of the implementation for the global registers.

4.6.1. Mupix Control Interfaces

The sensor side of the configuration upload was developed in [20] with the goal of providing a fast interface with minimal pin count. The pin count is relevant for Mu3e since the HDI flexprints on which the MuPix sensor will be glued have a limited amount of space. All signals from or to the pixel sensor need to be connected from the upstream or downstream direction. The space on the flexprints is, therefore, shared between data readout lines, power and configuration. This puts constraints on the number of MuPix sensors which can be used on the same flexprint and causes the need for a minimal pin count configuration interface.

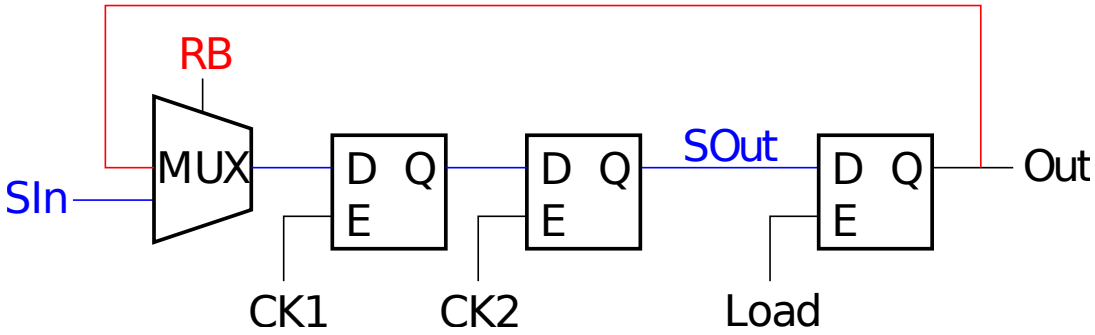


Figure 69.: A single cell in a MuPix shift register [20].

The six control shift registers on the MuPix are implemented as a chain of the cells shown in figure 69. The SIn signal always connects to the $SOut$ signal of the previous cell in the chain. Moving data through the shift register requires an alternating sequence of edges on the two clock signals $CK1$ and $CK2$. Once the data is in the correct place, the *load* signal is applied, which copies the content of $SOut$ to the Out signal. The Out signal then drives the configuration bit to actual components on the MuPix. Therefore, no change of the configuration takes place during the time when data is moved through the shift register. Only the load signal actually applies the current content of the shift register to the hardware.

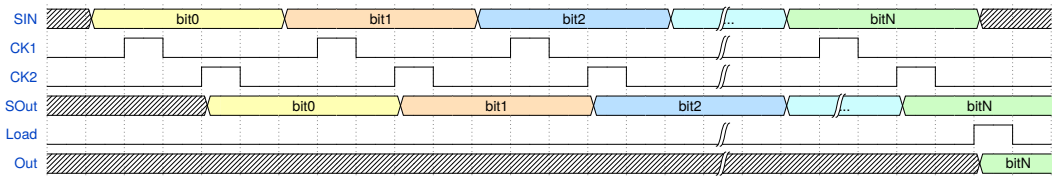


Figure 70.: Control signals for a single cell in a Mupix shift register. The output signal, which represents the actual configuration, is only changed when the load signal is applied.

The readback signal RB in figure 69 can be used to temporarily replace SIn with the output signal. This creates the possibility to return the configuration to components

at the end of the shift register by copying it into the space between the $CK1$ and $CK2$ latches and moving it towards the end with a sequence of $CK1$ and $CK2$ edges.

The amount of cells is not equal for all shift registers and ranges between 80 and 896. As shown above, there are five control signals which need to be supplied to each of the six implementations. The *Sin*, $CK1$, $CK2$ and *Load* signals exist individually and the *RB* signal is shared between them. This sums up to 25 signals. Four additional control signals will be explained later.

The firmware needs to use these 29 control signals to upload the configuration to the MuPix. There are three options to do so. The first option is to have a direct register communication using 29 individual lines connected from the FPGA to the MuPix sensor. This is a feature that was used for previous MuPix versions during the development phase and is not feasible for the actual implementation of the detector since it requires too many connections. Therefore, the following sections will not discuss this option further.

The second option is an SPI connection. This is implemented by connecting the 29 control signals with a separate 30-bit long SPI shift register[15]. This can then be written from the FPGA using an SPI data line, a clock line and a load/chip select signal. However, writing a single bit into one of the six actual configuration registers requires the firmware to completely write the 30-bit SPI register six times since the signals $CK1$, $CK2$ and *Load* must be applied and removed individually. The SPI connection on the ArriaV FPGA was usually operated at a slow-down factor of 8 relative to the 125 MHz reference clock, which together adds up to 1440 reference clock cycles to write a single configuration bit to the MuPix.

SPI configuration via NIOS RPC's was used for the test and development setups before the beginning of this thesis. The speed issues of the previously discussed RPC system and the steering of the SPI connection by the NIOS software added more delay to the 1440 reference clock cycles. Even if an ideal scenario is assumed and delays from all other sources are ignored, writing the full 0.5 Mbit of configuration data to the MuPix via SPI would require 720 million cycles and roughly 6 seconds. Without parallelisation, this adds up to 5 hours of configuration time for the 3000 MuPix sensors in Mu3e. As mentioned, tuning of MuPix sensors will be an iterative process and likely require hundreds of these configuration cycles. It is clear that this is not viable for the operation of the Mu3e detector and a parallel configuration architecture is needed.

Parallelisation has to be implemented on multiple levels. The highest and easiest level of parallelisation comes from the separate switching servers. Three of the four switching servers operate MuPix sensors and they can configure MuPix sensors independently of each other. The next level is the firmware on the frontend boards. It needs to be able to configure the connected MuPix sensors simultaneously. That rules out the use of sequential NIOS software and RPC's for this task.

The lowest parallelisation level is at the same time the third option of accessing the 29 control signals from above. A statemachine in the mupix can automatically

[15]The shift register would only need to be 29 bits long. It is 30 bits long.

perform the $CK1$ and $CK2$ cycles and other things. Instead of sending each action in figure 70 to the 30-bit SPI shift register, a 6-bit command with a 4-bit address and a 54-bit payload is sent to a statemachine on the MuPix, which then performs the sequence in figure 70. We will call these commands the MuPix slowcontrol protocol. They can be found in appendix B.2.

All MuPix sensors on a half ladder are connected to a single MuPix slowcontrol line. The address is used to identify the targeted chip on the ladder. The statemachine on the MuPix compares the received address against hardwired address pads, which depend on the ladder position. This saves additional lines on the HDI flexprint compared to SPI and also enables another level of parallelisation since multiple statemachines on the same HDI flexprint can be in the process of pushing their payload into the configuration registers at the same time.

Once the statemachine on a MuPix receives a command with the instruction to write the 54-bit payload into one of the six configuration registers, the same MuPix will not be able to receive other commands until the statemachine has finished this task. This results in a configuration deadtime.

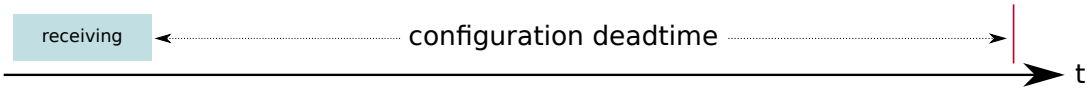


Figure 71.: Configuration deadtime in the MuPix protocol. A command and payload is received during the time marked in blue. Afterwards the chip is unresponsive until it comes out of its deadtime at the red line.

At the point where a chip comes out of the deadtime, no other transaction is allowed to be in progress. Otherwise, is could be falsely interpreted. However, if the transmission of a command is completely inside or completely outside of the deadtime of all other chips on the same ladder, a transmission is possible. Consequently, it is possible for the FEB to have a configuration conversation with multiple chips simultaneously. The issue is that every conversation with a MuPix will cause a dead point some time later where no conversation with any MuPix is allowed to be ongoing. This is shown in figure 72.

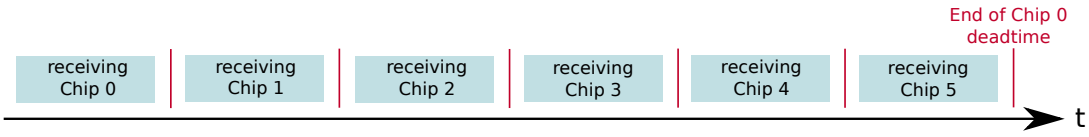


Figure 72.: Optimal communication in the MuPix protocol.

The task of the firmware is then to arrange communication with all chips on a ladder in a way where nothing collides with the end of any chip deadtime. The optimal solution will conceptually look like the graphic in figure 72. The length of the configuration deadtime is large enough to send commands to five other chips in the meantime. This is a result of the 54-bit payload and the amount of cycles it takes

for the statemachine to push this into one of the configuration registers according to figure 70.

The firmware developed during this thesis implements this by buffering configuration data from the switching servers on the FEB to ensure that data for all sensors is always available in order to apply the efficient spacing shown above. Due to a bug in a previous version of the MuPix sensor, this was the first time that the Mu3e protocol was put into operation in a DAQ system.

The concept shown in figure 72 is difficult to implement in pure software since it would require a just-in-time delivery of configuration data to all 3000 MuPix sensors in the system. The margins to avoid the end of deadtime points are small, and pure software cannot provide such timing precision. Options for a just-in-time delivery from the SWB firmware were considered during development but later abandoned in favour of a buffer-based implementation on the frontend board. The next section will discuss this implementation and show how it is used for the global configuration parts of the MuPix. The firmware for the tune values will follow afterwards since it will come with some additional complications.

4.6.2. Global Configuration

Each MuPix has its own slowcontrol address for global configuration. The global configuration for each chip is initially located in the MIDAS ODB, where it can be accessed by the MIDAS switching board frontend, which writes it to the corresponding slowcontrol address on the FEB. The range of slowcontrol addresses which belong to the global MuPix configuration is connected to the MuPix config splitter entity. This entity is one of the endpoints of the previously discussed slowcontrol tree on the FEB and receives the data from MIDAS through all the steps introduced in earlier sections. The splitter entity splits the arriving data into three separate 32-bit wide data streams for each MuPix. These three separate datastreams belong to the three global configuration registers on the MuPix sensor shown in table 13.

Name	Length	Content
BIAS	210	power and voltage settings
CONF	90	digital settings for statemachines
VDAC	80	global threshold voltages

Table 13.: The three global configuration shift registers on the MuPix.

The data for BIAS, CONF and VDAC is then written into a copy of the MuPix shift registers on the FEB. The FEB essentially mirrors the shift registers of all MuPix sensors that it connects to (figure 73). The global configuration arrives at the frontend board as one slowcontrol packet for each MuPix with a frequency of 156.25 MHz. In order to match this speed, the data is written into the mirrored shift registers on the FEB with 32 bits at a time. Therefore, the shift register on the FEB shifts by 32 positions on a write operation instead of just one position.

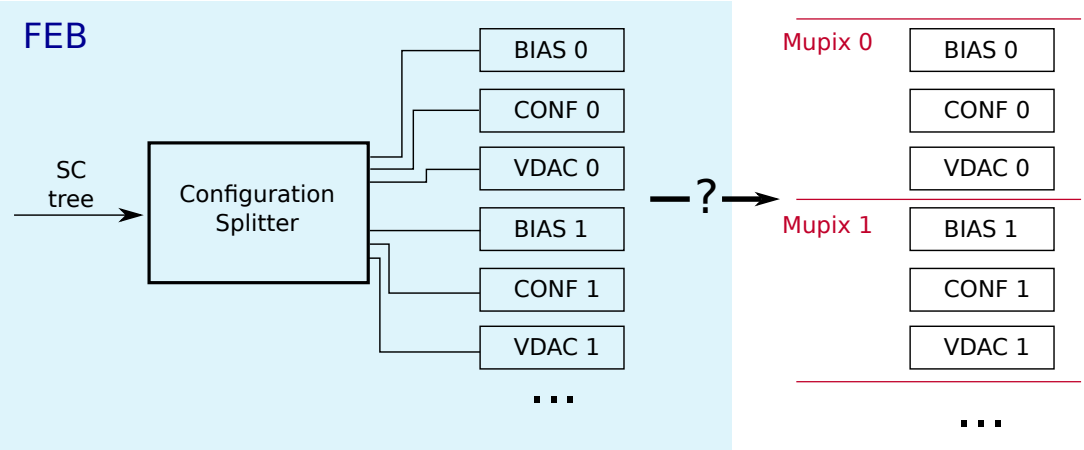


Figure 73.: The FEB holds a copy of the configuration registers of all connected MuPix sensors.

The new state S of the shift register at position i after write cycle n is calculated by:

$$S_i[n+1] = \begin{cases} \text{Input}_i[n] \text{ from conf. splitter}, & \text{if } i < 32 \\ S_{i-32}[n], & \text{otherwise} \end{cases} \tag{12}$$

There are now two possibilities implemented to ship the data from the mirror registers on the FEB to the actual shift registers on the MuPix sensors. The first one is the MuPix slowcontrol protocol, which, as previously discussed, sends data to MuPix sensors using commands with a 54-bit payload. Therefore, the mirror shift registers on the FEB are additionally wired up for a 54-bit shift operation.

$$S_i[n+1] = S_{i-54}[n] \tag{13}$$

Once a mirror register has been filled from the configuration splitter, this provides a read port with the correct interface width for the MuPix protocol.

Similarly, a single-bit shift operation

$$S_i[n+1] = S_{i-1}[n] \tag{14}$$

was implemented, which allows an SPI configuration option. As shown earlier, SPI moves data into the MuPix one bit at a time for each configuration register. Adding another shift operation here provides exactly the type of interface which would be needed by an optimal SPI configuration entity: One bit out of each configuration register in order to serve BIAS, CONF and VDAC at the same time from the 30-bit SPI register on the MuPix.

As an additional improvement, the two ends of the mirror shift registers on the FEB were connected to each other. This effectively forms a circular shift register with one write port and two read ports. Each write or read operation rotates the contents counterclockwise by the interface width of the operation. This is shown in figure 74.

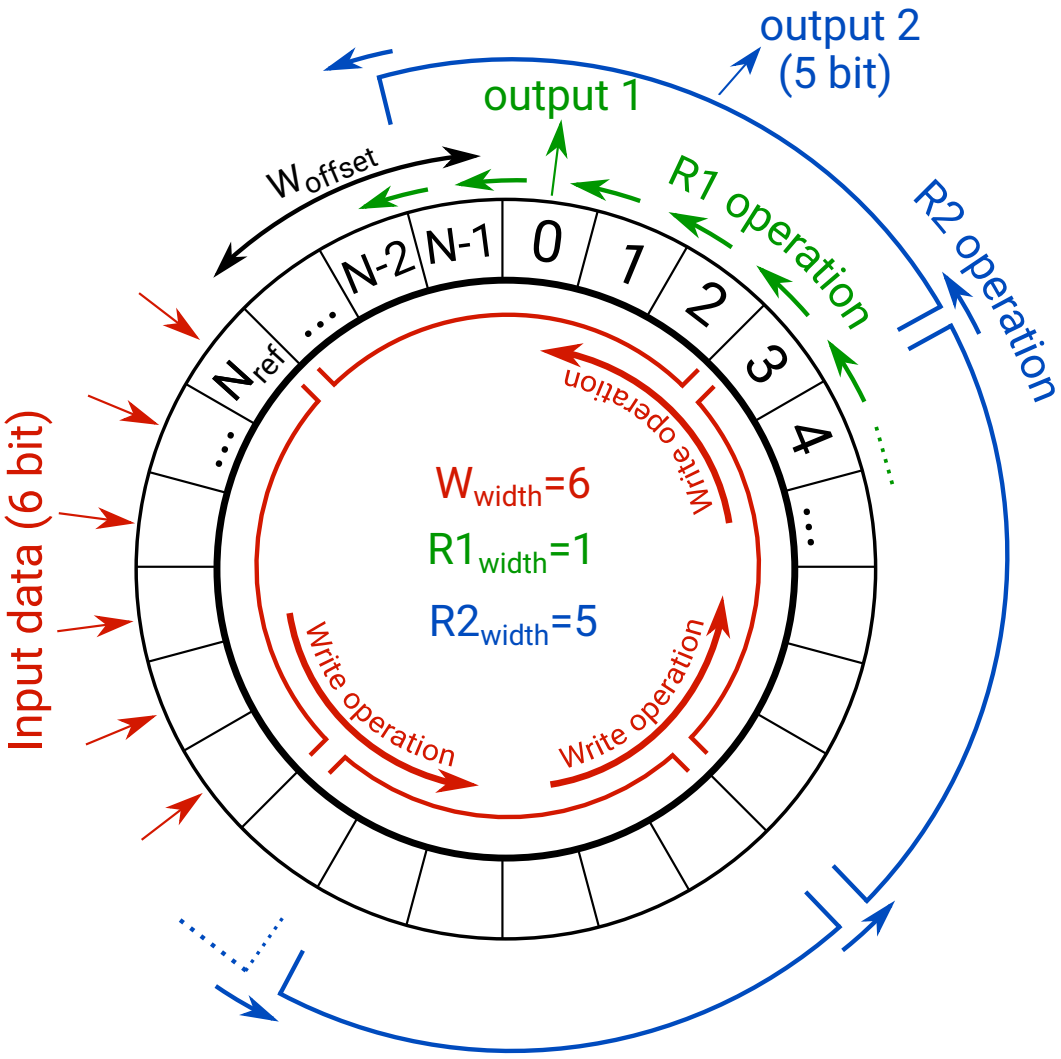


Figure 74.: Concept of the configuration buffer shift register on the frontend board using an example with a 6-bit wide write port and two read ports with widths 1 and 5. The write operation is marked in red. Data is written to the buffer on the 6 fixed indices $N_{ref}, ..., N_{ref} - 5$ of the shift register. Each write there rotates the other contents counterclockwise by 6 registers. The read operations R1 (green) and R2 (blue) rotate the contents by 1 bit for R1 and 5 bits for R2. The indices for the output ports have to be calculated relative to N_{ref}. For a 1-bit wide read port the index at which the output port is located will always be 0. Other read port indices (such as R2 in this example) will shift counterclockwise, starting from 0, by the amount necessary to fit an integer amount of reads into the index range until N_{ref}. The amount of bits W_{offset} between index N_{ref} and 0 is the maximum of the same idea applied to all three operations.

The advantage of connecting the two ends together is that the FEB firmware does still possess a copy of the configuration data after it has been shipped to the MuPix sensors. At the moment, there are no use-cases for this implemented. However, there are a couple of interesting possibilities. The FEB could, for example, perform an automatic readback of the configuration from the MuPix sensors and compare the received data at any time to the configuration, which should be inside the MuPix at this moment. This would not need to involve software since all the data is still available on the FEB. Another possibility would be to implement a repeat function, which could be broadcast to all MuPix FEBs and could instruct them to repeat the last MuPix configuration upload using data from the previous configuration.

On the data readout side, it might also be useful at some point for the FEB firmware to know the configuration settings for each connected MuPix. Either for the application of corrections or readout mode-dependent decoding of sensor data.

4.6.2.1. SPI Configuration

The SPI configuration is the simpler one of the two options to ship data from the mirror registers on the FEB to the MuPix. However, it is significantly slower compared to the Mu3e protocol option and is only used in some test setups. When SPI configuration is enabled via a slowcontrol register, an SPI entity for each SPI line will select a MuPix sensor and read 1 bit of configuration data from the corresponding BIAS, CONF, and VDAC mirrors on the FEB. These three bits of data will then be shipped to the MuPix by writing the 30-bit spi register six times according to the method shown earlier in figure 70. The load signal is automatically applied once the content of a mirror register on the FEB was fully read by the SPI entity.

4.6.2.2. Mu3e Protocol Configuration

For the Mu3e protocol configuration, one could, in principle, also just select any mirror register, send data from it to the MuPix and wait for the configuration deadtime before sending any other command to any other MuPix. However, the optimal version was implemented, which makes use of the configuration deadtime to send data to other MuPix sensors. The firmware to do so consists of three entity types for each configuration interface. A ticker entity provides the red lines in figure 75.

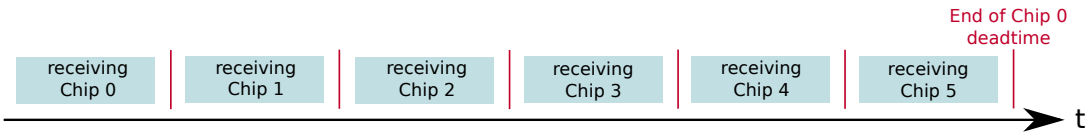


Figure 75.: Optimal communication in the MuPix protocol.

Instead of tracking and calculating the end of deadtime point for each MuPix on the interface, the possible positions of the red lines are defined by the ticker entity. All other Mu3e protocol entities on the FEB align their operations relative to the

ticks from this entity. The alignment of commands and spacing of the ticks matches the expected deadtime behaviour of the statemachine on the MuPix, which ensures that the actual deadtime endpoints coincide with the ticker. Therefore, the only thing that the other entities have to ensure in terms of timing is that they do not send two commands to the same MuPix within six ticks and that every command is properly aligned relative to a tick.

Every connected MuPix has an instance of a command assembler entity on the FEB. This entity checks the mirror registers for available data and decides which command and payload should be sent next to this specific MuPix. It also keeps track of the current position within each shift register in order to be able to issue the load commands at the correct moments. It signals towards the following parts of the firmware when a command is ready. When a command is picked up, it receives an acknowledgement and will prepare the next command for the MuPix. This command will not be made available for collection until the command assembler has seen six ticks. This ensures compliance with the configuration deadtime of this particular MuPix.

Each Mu3e MuPix slowcontrol connection is steered by a line controller entity. This entity chooses, at random, one of the commands flagged as ready and acknowledges it to the appropriate command assembler. After that, the command is serialised and sent out to the MuPix sensors with the correct alignment relative to the last tick. Therefore, the line controller prevents any collisions of commands with the end of deadtime points.

The ticker exists only as a single instance. The line controller exists once for every interface and the command assembler has one instance for each MuPix. Together, these three entities are able to operate every MuPix configuration at the same time, independently and at the highest possible efficiency. The limit of parallelisation is given by the amount of MuPix sensors connected to the same configuration interface. If too many MuPix sensors are connected, it is impossible to talk to all of them within the deadtime of a command. Accordingly, the configuration time will increase in these cases but still operate at the highest speed allowed by the protocol.

The software on the switching server writes a slowcontrol packet with the configuration data to the address and FEB belonging to the targeted MuPix sensor. There is no SPI or Mu3e protocol overhead in this packet since the entire protocol is produced by the firmware on the FEB. The speed limitation for the configuration of the Mu3e pixel detector is, therefore, given by the downwards bandwidth of the slowcontrol system. This is why the discussion earlier in this chapter in section 4.3.2.2 is important. Starting from the SWB firmware, the full bandwidth is available all the way through the FEB and the slowcontrol tree to the mirror registers of the MuPix sensors. At the time of writing this thesis, the limitation is the speed at which the software can push data to the SWB firmware. Speed improvements in other areas are unlikely since they already implement a solution which is able to reach the optimal link utilisation.

The next section will discuss the upload of tune values (TDACs) to the MuPix sensors. Speed considerations play a larger role there since they contain a lot more data than global MuPix settings. The tune value configuration will use the already existing entities shown here but will introduce a few additions and modifications.

4.6.3. TDAC Configuration

Tune values are used on the MuPix for a per-pixel adjustment of threshold values. In contrast to the global settings from the last section, the TDAC values are not stored directly in one of the six configuration shift registers. Instead, the configuration shift registers are only used to write the tune values into their actual location. Two of the remaining configuration shift registers are used for this task.

One of them, the TDAC shift register, contains a set of tune values and the other one, the COL register, defines the location in the MuPix matrix to which this set should be written. The routing and addressing of individual pixels on the MuPix is done via their column and row positions in the matrix. However, the digital indexing of columns and rows on the sensor is not identical to the physical column and row position of each pixel. This is intentional and was introduced in [20] in order to mitigate crosstalk effects. If the digital and physical addresses were identical, a particle that crosses between two pixels and causes a signal in both of them would look similar to crosstalk between neighbouring signal lines. Therefore, the routing of signals on the sensor was implemented in a way which does not resemble physical pixel locations. This allows the analysis to distinguish crosstalk from charge sharing and clustering effects.

In physical coordinates, the MuPix contains 256x250 pixels. In the digital representation, there are 512 columns and 128 rows. Each digital row address contains all the pixels of two neighbouring physical row addresses. The 2x250 = 500 pixels in such a double row are indexed using numbers from 0 to 511. Therefore, some of the 512 digital indices do not connect to one of the 500 physical pixels. The indexing function was changed between MuPix versions 10 and 11 but did not follow physical ordering in either of these versions in order to implement the crosstalk mitigation.

As introduced in section 2.2.1, each pixel contains 7 TDAC bits. The TDAC control shift register is used to access a specific one of these bits for all 512 pixel addresses in a double row. The COL register defines the bit and the row index to which the TDAC register connects to. This is shown in figure 76. For the example there, a single 1 is written to the fourth position of the COL register and all other COL register values are set to 0. This causes a connection of the TDAC register to the parts marked in green in figure 76.

In order to write tune values to the MuPix, the TDAC configuration firmware has to position a single 1 in the COL register, which is otherwise filled with zeros to select a row and a bit within that row. Then, the firmware has to write the according tune data to the TDAC register, which always contains only a single bit out of the 7-bit tune value for 500 pixels in the section selected by the COL register. This then has to be repeated for 128x7 selections until all the tune data is written.

Processing of tune data in software needs to happen in physical coordinates at some point. Since the ordering in which the MuPix would require bits of the tune values can appear quite chaotic and would require conversion functions in the software, it was decided to completely hide the existence of the digital address mapping from any software components. This includes the readout part, which will follow later in this

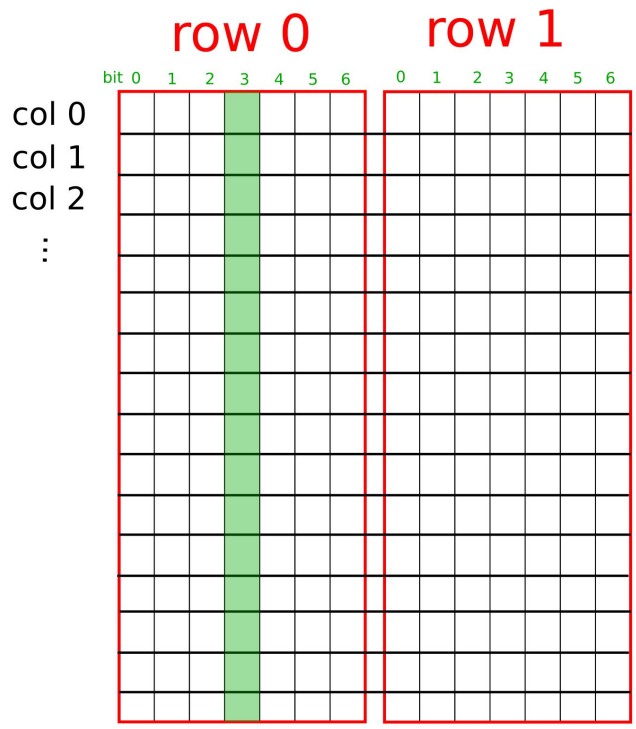


Figure 76.: TDAC register access to the tune values in digital coordinates. The part marked in green is accessed at the same time from the TDAC shift register. The horizontal location of the green part is defined by the COL register.

chapter, but also the tune value configuration. Therefore, the TDAC configuration firmware implements the mapping described above and no other DAQ parts need to know that it exists.

The first step of implementing this is to remove the column index conversion for the green area in figure 76. This is very simple to implement in the existing firmware framework since it only requires a minimal change of the mirror register idea from the previous section. Similar to the implementation for the three registers with global MuPix configuration, the TDAC configuration register will be mirrored on the FEB side. Writing data from there to the MuPix then uses the same infrastructure described in the previous section. The command assembler will include the TDAC mirror in the same way as the mirror registers for BIAS, CONF, and VDAC. Instead of issuing only the load command once the shift register was fully written, the command assembler will also relocate the selection in the COL register on the MuPix. This happens automatically whenever the TDAC mirror on the FEB was fully written and does not require any external input to the command assembler.

The digital column indexing required by the MuPix is then produced from the data in physical order by a minimal change of the mirror register. In the previous section, mirror registers were introduced as circular shift registers, where the content was

rotated counterclockwise by the width of the read or write operation. For the TDAC mirror register, the write operation does not rotate the contents counterclockwise. Instead, it rotates the contents counterclockwise along a permutation σ of the normal ordering.

$$S_i[n+1] = S_{\sigma(i-1)}[n] \tag{15}$$

The permutation σ can be chosen in a way where it produces the conversion between digital and physical pixel addresses. Therefore, TDAC bits are shifted into the mirror shift register using their physical ordering and are shifted out of it with digital ordering. This conversion does not use any additional resources since the only change is in the cabling order of a shift register which already exists. Per construction, there is no better place to do this since any other solution will either use CPU time or FPGA resources.

However, the data which needs to be written into the TDAC mirror register still contains only a single TDAC bit out of all pixels in the current row. This will be solved in the next section.

4.6.3.1. TDAC Configuration Memory

Transmitting data in larger slowcontrol packets towards the FEB is more efficient than transmitting individual words since individual words introduce additional delays due to protocol overheads and response latencies. This is especially true before the optimisation shown in section 4.3.2.2. Consequently, it is beneficial to introduce an additional buffer on the FEB in front of the TDAC mirror register. The three global mirror registers for each MuPix are large enough to hold the complete global configuration. For the tune values, this situation is not reachable since each MuPix has about 0.5 Mb of tune data. However, buffering on the FEB will still improve configuration speed since it allows the software to send tune data out in larger blocks.

Tune data is buffered on the FEB in the TDAC configuration memory. This is a single large RAM implemented in M10K blocks, which is managed by a TDAC memory controller entity. This memory controller is responsible for storing arriving data in the RAM and also serving it at the correct moments to the TDAC mirror registers.

Conceptually, the memory controller divides the RAM into a configurable number of sections (in the following called pages). The software has a slowcontrol address for each MuPix to which it writes tune data. When the write request arrives at the memory controller, it will select one or more empty pages and write the received tune data into them. For each page, the controller saves the ID of the MuPix for which the data was intended and the order of pages in which the data has arrived.

The controller also provides a slowcontrol register where the software can read the amount of free space in the memory in the form of an integer number of pages. Therefore, the software always knows how much tune data it can send to the FEB. The target chip does not matter for this since the memory controller will save the chip ID for each received page of tune values. This allows the buffer to be shared dynamically

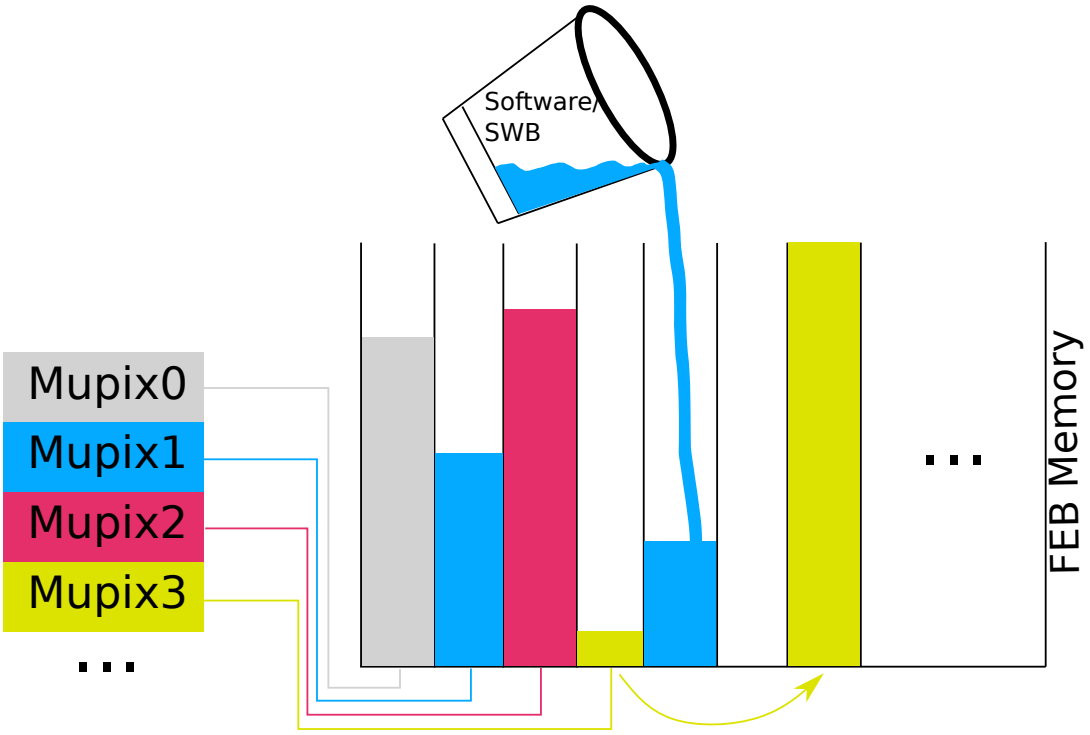


Figure 77.: Conceptual idea of the TDAC memory. The memory is divided into a set of compartments (pages), which can be individually refilled by the sofware via the SWB and the slowcontrol system. The memory controller keeps track of the content of each page and provides always the correct page towards the TDAC mirror registers for each MuPix.

between all MuPix sensors. If the software only wants to configure one of the connected MuPix sensors, the complete memory can be used for this one. Otherwise, pages will be assigned on the fly to MuPix sensors as the data arrives from the SWB.

The TDAC mirror shift registers are connected to the output side of the memory controller. Whenever one of the mirror registers becomes empty, the memory controller will start searching for the next data in its page registry. When the matching chip and page index is found it will read through the contents of the corresponding page and use it to refill the TDAC mirror with the correct data. Should the next tune data for that particular MuPix not be in the memory at that moment, it will be used once the required data arrives from the SWB.

The tune values are located in the buffer memory in the same format as they have arrived from the SWB. A series of 32-bit words where each of the four bytes in the word is the tune data for one pixel. However, as introduced above, only one bit out of each byte is required for a single refill of the mirror register. Therefore, the memory controller will read each page seven times before it is declared empty again. These seven reads are used to gather the seven tune bits for each pixel. Consequently, every

clock cycle during the read process only provides four bits for the refill of the TDAC mirror register. This is implemented by simply reducing the write input width of the mirror register to four and does not have any other consequences.

Once a page has been read seven times, it will be declared empty and is deleted from the controller's page registry. The slowcontrol register containing the amount of free pages is increased by one and the memory section can be used for new data again.

4.6.3.2. TDAC Configuration Speed Tests

The last section concludes the discussion of all Mu3e DAQ parts necessary for MuPix configuration. As mentioned before, the speed limit is introduced by the time it takes the software to push the data to the FEB via the slowcontrol system. However, this is influenced by the size of the TDAC configuration buffer since a larger buffer will reduce protocol overhead and the amount of roundtrip latencies.

Strictly speaking, that is only the case for a large amount of MuPix sensors. For a single sensor, the bandwidth of the software towards the FEB is larger than the allowed speed of the Mu3e protocol from the FEB to a single MuPix sensor. Therefore, the time it takes to configure the tune values depends on the amount of MuPix chips in the system.

At the time of writing this thesis, the full system with almost 3000 MuPix chips was not available for a tune value upload speed test. Instead, an alternative method was employed. This method involved using a DAQ system with a single FEB and simulating other FEBs with a wait time. The SWB software reads the amount of free pages from the one existing FEB and sends data to fill all of them. It then waits for a time T before reading the amount of free pages again, repeating this process until all MuPix sensors of that FEB are configured. This sequence is then repeated for different wait times T. The result is shown in figure 78.

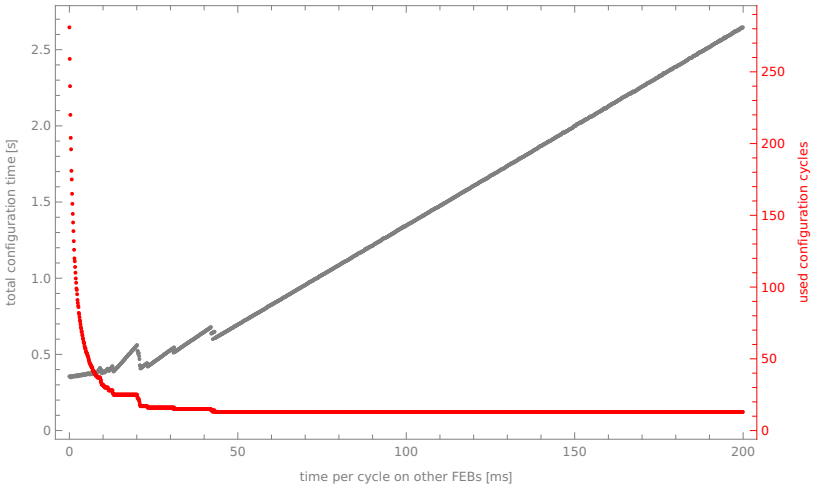


Figure 78.: Raw data of the TDAC upload speed measurement.

T simulates the time that the software would have spent on interactions with other FEBs if they existed. After the time T, the software returns to the existing FEB. For large values of T, the number of needed configuration cycles is low since all TDAC pages on the FEB buffer are empty when the software returns to the existing FEB. For lower values of T, not all TDAC pages are empty when the software returns. Therefore, less data can be written to the FEB for each cycle and more configuration cycles are required to complete the TDAC upload. This is shown in red in the plot above.

This measurement can then be used to estimate the configuration time for any number of MuPix chips. The first step is to calculate the fraction of time that the software was spending on the existing FEB by subtracting T times the number of configuration cycles from the total configuration time. The total configuration time is then divided by this number, which results in an estimate for the number of FEBs that the software could serve simultaneously, assuming that it always spends the same amount of time for each FEB and configuration cycle.

The variation of T produces then a relation of total configuration time against the amount of FEBs. That can be translated into a configuration time against the number of MuPix chips by multiplying the amount of FEBs with the number of MuPix sensors per FEB. The result is shown in figure 79 for different versions of the TDAC upload firmware.

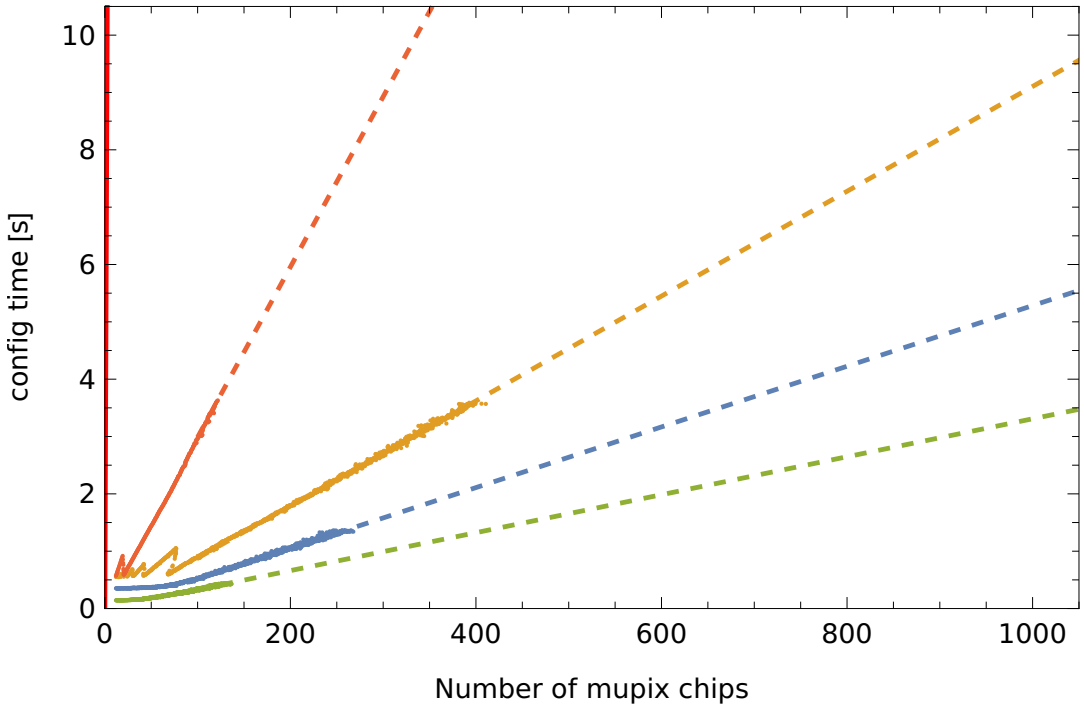


Figure 79.: Configuration time needed for different amounts of MuPix sensors and firmware versions.

The blue line in figure 79 represents the currently implemented version of the TDAC firmware. The two orange measurements are versions with a smaller buffer memory and without the optimisation of the SWB firmware shown in section 4.3.2.2. The green measurement is only achievable if the TDAC upload memory is increased with memory resources of other firmware parts of the FEB. Therefore, this will not be implemented for the final FEB firmware. The almost vertical red line is the reference before the introduction of the TDAC upload firmware.

For the first hundred MuPix sensors, the configuration time for the blue version is almost constant. In this region, a full parallelisation is possible and the Mu3e protocol between the FEB and the MuPix sensors limits the speed. For a larger number of MuPix chips in the system, the software's speed limitation is reached, causing a linear increase in time. About four seconds are needed for the configuration of 1000 MuPix sensors. This should also be roughly the achievable configuration time for the full Mu3e pixel detector since the 3000 chips there are distributed across three independent switching servers.

The pixel-individual thresholds are determined by three settings: The global threshold, a pixel-individual tune value which modifies the global threshold and a global dynamic range setting which defines by how much a single tune value step changes the threshold. An automated tuning procedure is still to be implemented by the collaboration. It will have to involve iterative search methods with many configuration cycles. After each configuration cycle some amount of data will have to be taken to calculate the next step of configurations. The reduction of configuration time to a total of four seconds for the entire Mu3e pixel detector is an essential part of this process. Previous configuration methods would have been unusable since a single configuration cycle would have required multiple hours for the amount of MuPix sensors in Mu3e.

4.6.4. Initialisation and Reset

It was already mentioned previously that there is not much space for additional signal traces on the flexprints to which the MuPix sensors are glued. For this reason, the reset signal to the MuPix is also provided through the Mu3e slowcontrol protocol. A reset of a MuPix sensor is achieved by driving a high signal into the Mu3e slowcontrol link for an extended time period. If the slowcontrol link to the sensors is high for more than 512 cycles of the 125 MHz reference clock, then the signal will act as a reset. Timestamp counters and other digital components on the MuPix will start operating once that signal is released. This includes the statemachine which interprets the Mu3e slowcontrol protocol. It is, therefore, necessary to issue a reset before any slowcontrol communication with the sensor can be started. Otherwise the alignment of the slowcontrol statemachine on the MuPix with the machinery on the FEB cannot be guaranteed and will likely not operate correctly.

This creates a problem. For the synchronised operation of the Mu3e detector during a physics run, the reset of the MuPix is provided by the clock and reset system discussed in section 4.4.2. As discussed there, this has to operate on the 125 MHz global

reference clock domain. However, the configuration firmware on the FEB from the last sections operates on the 156.25 MHz clock of the optical link to the SWB. The issue is that the configuration firmware requires a reset in its own clock domain to achieve alignment with the MuPix statemachine before it can send commands to the MuPix. The consequence is that the control line to the MuPix sensor needs to be handed over between the 125 MHz and 156.25 MHz clock domains whenever the system state changes between physics data taking and ongoing MuPix configuration. Configuration during a physics data run is not possible in the current architecture since the required reset in the 156.25 MHz domain would displace timestamp synchronisation for the remainder of the run.

Alternatively one could think about a CDC to the 125 MHz domain within the configuration firmware. For the tune values, the CDC could be easily implemented in the TDAC RAM by using a dual clock memory version and modifying the memory controller. However, for the global MuPix settings the earliest point where a transition is possible is at the read port of the mirror shift registers. It would likely be neccessary to use some sort of manual handshake there since these registers exist three times for every MuPix which makes it a very wide and resource-intensive interface for any memory based CDC. It could be possible to circumvent this by introducing a CDC FIFO earlier in front of the configuration splitter. Since the global parts of the MuPix configuration are not particularly large, this FIFO could be chosen large enough to buffer the difference between the two clock domains. However, currently it does not seem like a problem to separate physics data taking from MuPix configuration. Consequently, these ideas were not further explored.

This section concludes the discussion of the concepts for control, monitoring, and configuration of the detectors in Mu3e. A few independent aspects have been outsourced to section 4.11. The following sections will discuss the readout path of physics data. They will make use of the previously described systems for any monitoring or control functions and will merge and demerge with the control data flow according to sections 4.4.3 and 4.3.2.

4.7. MuPix Readout

The MuPix sends serialised 8b10b encoded hit data packages at a data rate of 1.25 Gbit/s over three separate LVDS output links. Each of these links is responsible for a vertical section of the sensor. A fourth LVDS output is configurable and can either be used to send data from all three sensor sections over one link or it can copy one of the three separate outputs.

The fourth link is used for the outer layers of the Mu3e pixel detector. On the outer layers, the space constraints for traces on the HDI flexprints do not allow the use of three individual LVDS data readout traces for each MuPix. On the inner pixel detector layers, the individual links are used. Therefore, the output bandwidth per sensor on the inner layers is larger. The sensors there will also receive more particle hits per area since they are located closer to the target.

The LVDS signals are routed out of the detector area via a series of connectors and adapters followed by a distance of $\mathcal{O}(1\,\mathrm{m})$ of custom-build micro twisted pair cables before arriving at the service support wheel. There, the signals are received by LVDS repeater chips on an adapter board before being sent to the FEB on the other side of the backplane PCB.

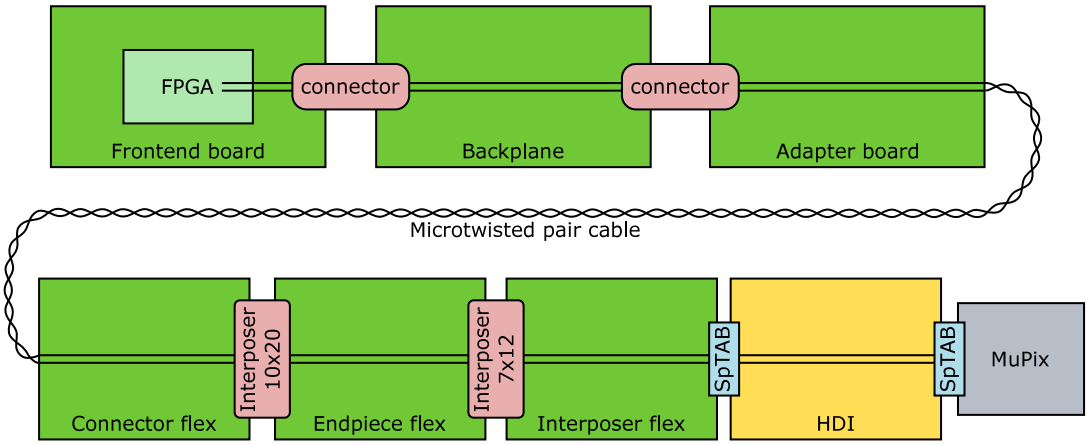


Figure 80.: Signal path between MuPix chip and the FEB. Taken from [15].

Achieving an error-free transmission over this connection chain is not a trivial task. It is not just influenced by the connections shown in figure 80 but also by settings and manufacturing variations of the connected MuPix chips. The frontend board firmware has to provide the tools to monitor signal integrity and tune the MuPix settings accordingly.

Signal integrity is measured on the FEB by counting invalid 8b10b codes, realignment cycles of the LVDS receiver and 8b10b-disparity errors. Signals from the MuPix sensors are deserialised in an LVDS receiver and sent to a data decoding entity for each input link. This entity performs 8b10b decoding and searches for the alignment pattern by shifting the deserialisation in the receiver. Once alignment is achieved, it is locked, and a large number of invalid codes are required before it can be moved again. This allows to distinguish errors introduced in the chain from the MuPix to the FEB from errors introduced by clock variations on the MuPix: A bad connection between the MuPix and FEB, which occasionally produces an 8b10b error, should be unable to shift the alignment of word boundaries relative to the reference clock on the FEB.

The three counters for each link are connected to the slowcontrol system and are regularly read by the SWB software and written into the ODB for monitoring in MIDAS. These values can then also be used for automated scans of relevant MuPix settings.

From the data decoder, each data stream is then forwarded to a data unpacker entity, where the data packets from the MuPix are deconstructed into hits with column, row and timestamp information. Additionally, the timestamps are gray-decoded. The next step is to condense the up to 36 data streams into a lower number of interfaces. This

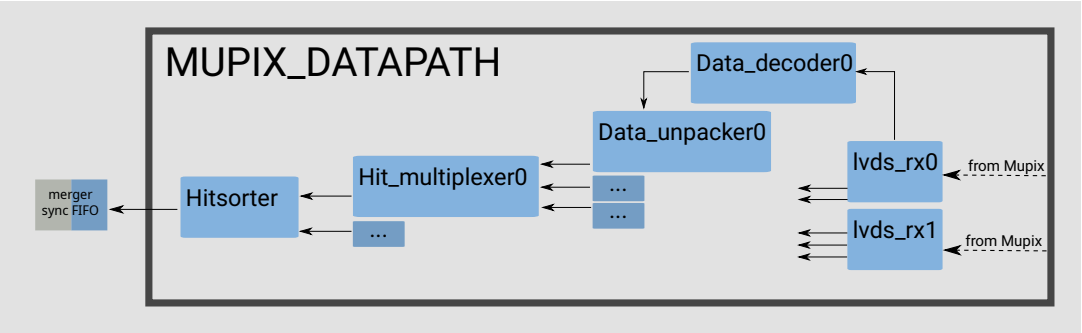


Figure 81.: Structure of the MuPix datapath on the FEB.

is possible since the statemachine on the MuPix is only able to produce one hit every four cycles. The three vertical sections of each MuPix can, therefore, be condensed into a single datastream on the FEB, which reduces the maximal number of interfaces to twelve.

4.7.1. Hitsorter

These twelve interfaces are then fed into a single hitsorter entity, where the hits of all twelve inputs are sorted in time and merged into a single output stream. This stream is required to be sorted in time by some of the following readout components. The concept behind the sorter entity was first introduced in [2] and then finalised in [69]. It was not developed in this thesis and will be shown here for completeness of the datapath. The idea is to store the incoming column and row information in a memory address determined by the timestamp of the hit. This memory can then be read in order, which results in a sorted output. However, the implementation of this idea is a bit more complicated.

The first complication is that more than one hit with the same timestamp can be received. For this reason, the timestamp cannot be directly used as a memory address since multiple memory locations are required for the same timestamp. Another issue is that one needs to keep track of the locations which are filled with hit data. Both of these complications are solved with a second memory, which contains the number of hits in each timestamp slot. This is shown in figure 82.

When a hit arrives at the sorter, the counter memory location (green) at the timestamp of the hit (blue) is read. The column and row is then written into the according memory slot (red), and the entry in the counter memory is increased by one.

The counter memory can then be used to construct the output datastream by reading through the counter memory in order and sending the according content from the main memory. This needs to happen with some separation with respect to the writing process. To achieve this, a sliding read and write window of addresses is defined. The write side is only allowed to receive timestamps within its current timestamp window. Other hits are thrown away. Similarly, the read side needs to operate outside

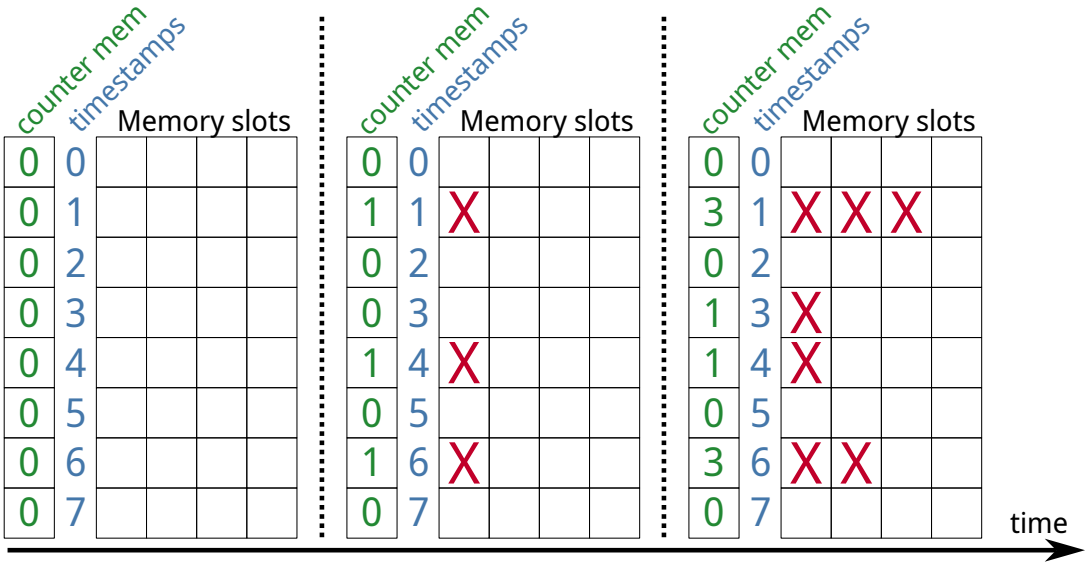


Figure 82.: Write procedure of the hitsorter.

of this timestamp window and needs to throw away hits if it cannot keep up with the movement of the write window.

The size of the write window needs to be large enough to cover the expected range of timestamps that could arrive from the MuPix sensors. The FEB and all connected MuPix sensors are synchronised at each run start. Therefore, particle detections on the MuPix sensors are assigned a timestamp that is always identical to the current reference timestamp on the FEB. The time range that it can take the MuPix to deliver this hit to the FEB needs to be fully covered by the write window in the sorter. Consequently, the borders of this time window are defined as a fixed offset relative to the reference timestamp on the FEB. Hits arriving in time and out of time are counted for each input link and these counters can be monitored from the software via the slowcontrol system[16]. Shifts of the write window can also be configured from a slowcontrol register.

The depth of the MuPix hitsorter covers 11 timestamp bits for each input line and the main memory contains 16 slots for each possible timestamp and sensor. This is one of the main reasons why memory blocks on the ArriaV FGPA are a rare resource for the other firmware components. During this thesis, an attempt was made to implement the counter memories in MLAB cells since they currently only utilise about 51 % of the space in their M10K blocks. This attempt failed due to the lower timing performance of MLAB cells. At the time of writing this thesis, the connections of the memories in the sorter are the most timing critical paths in the MuPix FEB firmware and make up most of the minimal slacks shown in the histograms in sections 3.7.6 and 4.4.4.4 after the optimisations discussed there.

[16]The sorter is one of the slowcontrol tree endpoints discussed in section 4.4.4.4.

4.7.2. Mupix Data Packets

Each roundtrip of the read pointer in the sorter memory produces a data packet. Since the sorting depth contains eleven timestamp bits, that will happen on average every 2^{11} clock cycles. The packet is then sent towards the data merger in the common part of the FEB firmware and follows the previously introduced 36-bit interface. It arrives in the common firmware part via the synchronisation FIFO in figure 63 of section 4.4.3, which translates it into the 156.25 MHz clock domain. Starting from there, it integrates into the previously described common firmware parts and is merged with run control and slowcontrol data packets to be sent to the SWB via the optical fibres.

The protocol follows the description in table 6 at the beginning of this chapter but adds a MuPix-specific substructure. This substructure is shown in table 14. After the preamble, a header followed by a sequence of sub-headers and mupix hit data is transmitted and the packet is closed with a trailer as defined earlier in this chapter.

The header contains a continuously increasing header identification number (H-ID)[17]. It is followed by a subheader and hit counter of the previous data packet and a reference timestamp. The reference timestamp is the timestamp counter on the FEB at the creation time of the packet. That is not necessarily identical to the timestamp to which the contained data belongs to, which can be calculated by multiplying the header ID with 2^{11}.

The header is followed by a sequence of sub-headers with their contained data. Each sub-header provides eight more timestamp bits and overflow information from the previous sub-header. The hits in each sub-header deliver the rest of the timestamp bits and the location of the hit in the detector.

It is necessary to split the individual timestamp bits into different header levels to reduce bandwidth usage. Sending the full timestamp information for every single hit is not efficient since the higher bits of the timestamp counter will be identical for every transmission. Therefore, the total timestamp of each hit is provided in different stages. The lowest bits are transmitted in the individual 32-bit hit words and some higher bits are transmitted in a sub-header. The sub-header is only sent when the time information provided there changes. The highest bits of the timestamp can be constructed from the header of the packet.

These packets flow through the common firmware parts described earlier and are further processed once they arrive on the SWB in section 4.9.

4.8. Mutrig readout

Conceptually, the MuTrig readout works similarly to the MuPix readout from the previous section. Data is also received by 8b10b encoded 1.25 Gbit/s LVDS links and is also fed into a variation of the sorter, which produces packets similar to the one shown in table 14. The only difference there is the packet ID in the preamble and the

[17] The header counter in the second word is identical to the lower bits of the H-ID. This has historical reasons and was kept for backwards compatibility [70].

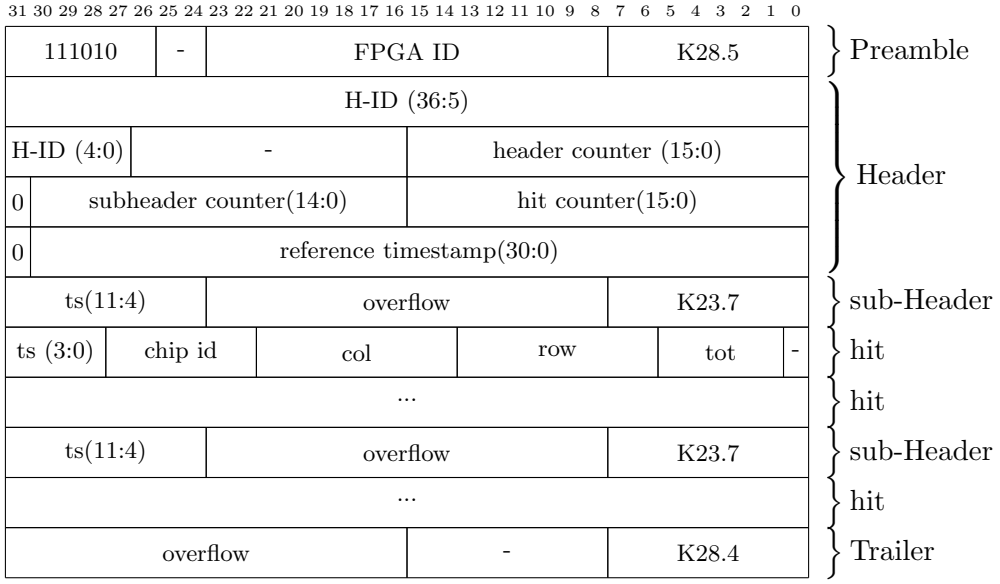


Table 14.: Structure of a mupix data packet between the FEB and SWB.

information contained in each hit. Mutrig hits will contain a channel ID instead of column and row information and will provide more timestamp bits due to their higher time resolution.

However, the timestamp measurements provided by the MuTrig require some further processing before they can be sent into the hitsorter. The MuPix timestamp is gray encoded, which is solved by a simple conversion function implemented in the unpacker entity. The MuTrig timestamp is a pseudo-random number that comes out of a linear feedback shift register (LFSR) on the MuTrig. The reason for this implementation is a higher maximal frequency. The maximal speed at which a circuit can operate is related to the Fan-in and Fan-out of the registers within it. The highest bit of a counter – also a gray encoded counter – will have a Fan-in of at least all the lower bits. An LSFR can operate at higher frequencies since registers only have to connect to their neighbours in the register chain.

As discussed in section 3.4.1.3, LFSRs only have $2^N - 1$ possible states since an LFSR where all registers are set to one is stuck at this output. Therefore, the overflow of an LFSR counter will happen one cycle earlier than the overflow of a binary counter with the same amount of bits. The consequence is that there is no direct conversion between the content of the LFSR and the according binary counter value. Such a conversion is only possible if the number of overflow events of the LFSR counter is known. Therefore, the number of overflow events since the reset has to be calculated on the FEB and has to be used before the sorter to convert the time information of the MuTrig. This thesis was not involved in the development of this firmware. Further details can be found in [71].

Another difference between the MuPix and MuTrig readout is introduced in the FEB firmware for the scintillating fibre detector. The FEBs connected to the fibre detector are expected to produce a higher data rate than any other FEBs in the DAQ. These FEBs are, therefore, equipped with a second optical output link, which doubles the bandwidth towards the SWB. The firmware for the second link is equivalent to the first one. A second instance of hitsorter and data merger is used and all other components and protocols are applied accordingly. The slowcontrol packet input of the secondary merger is unused.

4.9. Switching Board

Once the data packets arrive at the SWB, they are first separated from run control and slowcontrol packages in the data demerger. This entity uses the packet ID for this purpose and was already mentioned in figure 52 of section 4.3.2. Large parts of the datapath following that entity were developed in [71] and will only be briefly summarised here.

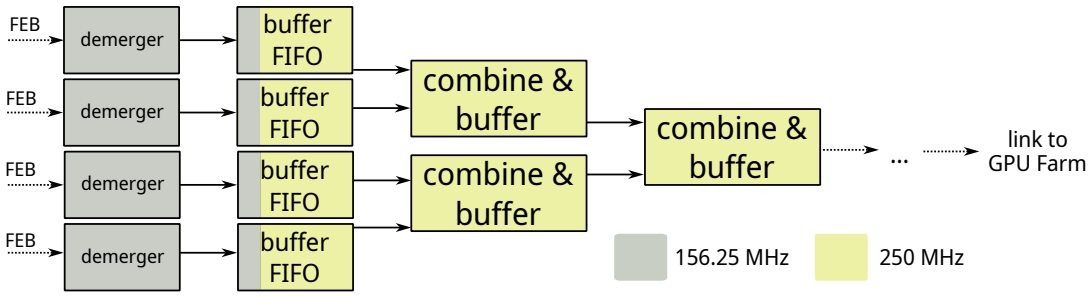


Figure 83.: Simplified structure of the SWB datastream merging.

After the separation from other packets, the data packets from MuPix, sciFi or sciTile FEBs are buffered in FIFOs and transit into a faster 250 MHz clock domain. This CDC happens because the optical link between the SWB and the GPU farm is operated at 10 Gbits/s and the parallel frequency needs to follow accordingly. The increase in the optical transmission speed is possible since the Arria10 FPGA on the switching board supports faster transceiver speeds than the ArriaV on the FEB.

The task of the SWB, as already mentioned in the introduction of this chapter, is to combine the time-sorted data streams from all connected FEBs into one time-sorted datastream to the GPU farm. This is implemented in a tree structure where each node combines two data streams. This is repeated until all streams are merged. Each combination circuit reads hits from both inputs, merges them according to their timestamp into a time-sorted output stream and buffers that in another FIFO. When no data is available on one of the inputs, the circuit has to wait until it is available to be able to compare the timestamps of both inputs. This is the reason why packet loss or even the loss of a single subheader is not acceptable in the Mu3e DAQ since missing data will stop the data flow for the entire system at the stream merging on

the SWB.

As a consequence of the merging, the chip ID in MuPix or MuTrig packets needs to be modified. The packet format between the FEB and the SWB has enough bits reserved to identify the chip for this specific FEB. When the streams are combined, additional information is needed to identify the source of a particle detection since the chip IDs are not unique anymore. This mapping to a new addressing scheme is also done on the SWB.

From the SWB, the re-packaged data is sent via 8b10b encoded 10 Gbit/s optical lines to the GPU filter farm.

4.10. GPU Farm

On the first machine of the GPU farm, all the data from all detectors in Mu3e is received by a single commercial board with an Arria10 FPGA. The four switching boards in the previous DAQ layer send a continuous, time-sorted data stream with a total of 16 optical connections opertaing at 10 Gbit/s.

The Arria10 selects a time frame of the incoming data and writes a copy of it to a DDR4 memory on the same PCB. The selected frame is flagged as processed. The received datastream is then – otherwise unchanged – forwarded to the next server of the GPU farm. This is repeated until all time frames are flagged as processed.

On each Arria10 board, the detections of the pixel sensors of the central station are converted into physical coordinates and sent to the main memory of the server via direct memory access (DMA). From there, the data is accessed by a tracking algorithm on the GPU, which makes a selection decision after reconstruction of the decay vertices based on the expected kinematics for a $\mu \to 3e$ decay.

If a frame was selected, another DMA transfer is used to copy the data from the DDR4 buffer on the Arria10 board to the server's main memory. This includes then also the data of the remaining parts of the pixel detector in the upstream or downstream recurl station and the data from the scintillating detectors. The selected frame is received by the CPU, processed and sent to the MIDAS data collection server via an ethernet connection. The selected frames from all GPU farm machines are accumulated there and written to long-term storage for offline analysis. The complete process is shown in figure 84.

It was shown in [23] that 12 PCs equipped with Nvidia GTX1080Ti graphics cards can process the expected data rates in the first phase of the experiment. Newer work on the online selection [72] will likely be able to reduce that number, also because graphics cards with a higher performance are available now. Further details on the firmware implementation can be found in [71] and [47].

A previous version of the concept for the layer 3 DAQ was removing selected time frames from the stream instead of just flagging them as processed. The change towards flagging was made to add the possibility of other selection algorithms at the end of the Mu3e filter farm. The full data is still available there and more machines with alternative selections can be added. This could be used at some point, for example, for

a GPU-based cosmic selection for alignment purposes in Mu3e or for other searches ([73], [74]) using the Mu3e detector.

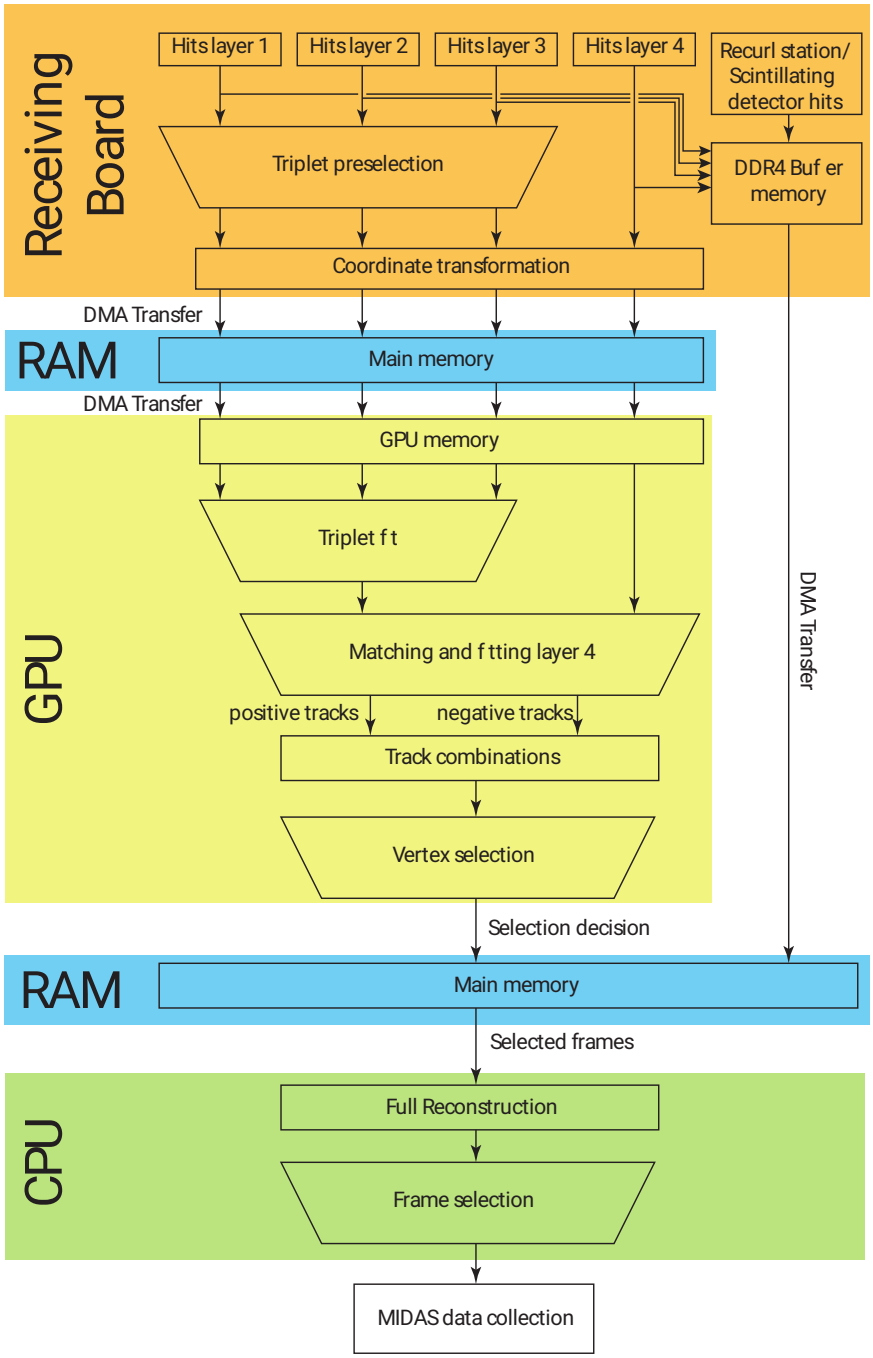


Figure 84.: Data flow in the online reconstruction. Taken from [22].

4.11. Other Aspects of the Mu3e DAQ

This section is intended to cover aspects of the Mu3e DAQ which are relevant to operate it but have not been necessary to understand the previous discussions.

4.11.1. FEB Boot Sequence

When the FEB is turned on, the firmware on the ArriaV and parts of the clock distribution between components on the FEB are in an undefined state and will not operate properly immediately. The FEB has to follow a certain sequence to put its components into operation since some components might depend on others. This sequence has to be predefined and has to proceed automatically after power-on since there is no connection to steer this process externally at this point. The FEB is located inside the Mu3e magnet, and any communication with the outside requires some parts of the FEB to function properly.

The first thing that starts operating on the FEB is the Max10 FPGA. The firmware there is automatically loaded from non-volatile memory included on the chip. The firmware waits for an internal PLL to lock onto the 50 MHz clock provided by a quartz oscillator and uses this lock signal to reset its firmware blocks. This is the start of the boot procedure. Once the Max10 is in operation, its task is to start up and configure the main ArriaV FPGA on the FEB. Ensuring that the ArriaV operates is why the Max10 is present on the frontend board.

The Max10 loads a firmware image from flash memory on the FEB and uses it to configure the ArriaV FPGA. Only the 50 MHz domain on the ArriaV (table 10) is operational after configuration since it is supplied by another external quartz oscillator. The other clocks are supplied by two clock chips on the FEB (SI-1 and SI-2, shown in figure 56), which must be configured before they drive the correct clock signal. The NIOS configures these chips with the settings saved in the firmware image via an SPI connection operated in the 50 MHz domain, starting with SI-2. SI-1 is configured a few seconds later since it is driven from SI-2 and requires a stable input signal. An earlier configuration of SI-2 does not achieve a defined phase of the 125 MHz clock for all components and breaks synchronisation between the individual FEBs in the DAQ.

Once both clock chips are configured, all clocks on the FEB will be available. Starting from there, the NIOS can steer the slowcontrol system in the 156.25 MHz domain from the connection shown earlier in figure 66. This is then used to reset and properly start the other parts of the ArriaV firmware and to move the FEB state into idle. Before the clocks are stable, the receivers of the reset link on the FEB can misinterpret the input stream and change the FEB state randomly.

Following the boot of the frontend boards in the magnet, a connection with the MIDAS software can be established. When the software is started, it will read the list of the expected frontend boards from the MIDAS ODB and iterate over them with a slowcontrol ping. A slowcontrol ping is an SC writerequest to a specific not writeable address which is used to check if the connection exists. If the ping was successful, first

setup informations are exchanged between the MIDAS ODB and the frontend board[18] and the system is ready for user operation.

It would be possible to extend this procedure by a cabling check in the future. The slowcontrol port on the SWB could be compared against the backplane ID read from the FEB. This would allow the automated detection of any wrongly connected cables, which might be a valuable addition, especially once more parts of the DAQ are assembled at PSI.

4.11.2. FEB Shutdown and Overheat Protection

A shutdown function was implemented on the FEB following an incident that will be discussed in a later chapter. The intention of the shutdown function is to reduce the power consumption of the FEB quickly. It can be triggered via the slowcontrol system but is intended to be used as an automated overheat protection.

The temperature of the FEB can be monitored by an internal IP on the ArriaV FPGA and via measurements of an ADC on the Max10. If the temperature increases above a hardcoded limit, the Max10 will send a shutdown signal to the ArriaV. When the ArriaV receives the shutdown signal, it will turn off the 156.25 MHz and 125 MHz clocks by permanently raising the resets of the clock chips. The 50 MHz clock also gets disconnected on the ArriaV and the optical transceiver module is instructed to turn off.

Turning off the clocks removes most switching processes in the FPGA. Switching a register state requires power. Therefore, the power consumption is reduced when registers do not change their state anymore. Some of the power drawn by the FEB is used by the adapter boards on the other side of the backplane. These components are not affected by a shutdown. On a standalone FEB a shutdown causes the power consumption to drop by more than 50 %. It is not foreseen to recover from the resulting state except by a power cycle of the FEB.

4.11.3. FEB Firmware upload

Once the FEBs are mounted at the service support wheel and the detector is inserted into the Mu3e magnet, there is no longer any physical access to change the firmware manually. During normal operation, the firmware image is loaded from the flash memory by the Max10 FPGA, as discussed in the boot sequence. A change of the firmware image on the flash memory is only viable via the communication channels provided by the Mu3e DAQ at this point.

Physical access to the FEB would require partial disassembly of the experiment and, in the case of the upstream side, the movement of a heavy beamline quadrupole magnet. Additionally, the helium atmosphere will be lost and must be replenished after reassembly.

[18] The git hash of the firmware is read to the ODB and information about the amount and location of connected ASICs is written to the FEB.

This amount of work is not reasonable for a firmware change. Therefore, the firmware image can be uploaded via the slowcontrol system to the ArriaV FPGA, which forwards it via an SPI connection to the Max10, where it is written to the external flash memory.

4.11.4. Backup communication via the Backplane

If any transmission error or interruption of the mentioned write process of a firmware image to the flash memory occurs, the image on the flash is unusable and the FEB will not be functional after a reboot.

There are two more lines of defense before the FEB is not accessible from the outside anymore. The first one is a backup image on the flash memory. The flash is large enough to hold two firmware images. Which one is loaded at boot can be selected in the Max10 firmware via a pin on the connector to the backplane. An MSCB node controls the signal on the backplane. The idea of the backup image is to always just overwrite one of the images on the flash memory during firmware upload. If anything fails during that upload, it is possible to switch to the backup emergency image and use this one to write the broken image again. Should this fail a second time, the FEB will not be reachable via the SWB slowcontrol system anymore.

In this case, it is foreseen to have a backup communication channel via MSCB on the backplane to be able to write a functional firmware image to the flash memory. MSCB was discussed earlier in this chapter in section 4.4.4.3. However, this method will require a significant amount of time for the upload process due to the low speed of the MSCB interface.

4.12. Generators and injection points

Data generators are used in different locations in the Mu3e DAQ in order to generate fake data for system tests without operating the full Mu3e detector with a beam at PSI. Depending on the location at which this fake data can be injected, the generators provide a different level of realism. For example, on the last layer of the DAQ, the collaboration performed multiple test runs where data frames from the geant4 simulation were converted into frames of the DAQ to test the reconstruction and analysis flow.

A similar concept allows running the software parts of the system without the presence of any hardware components. A simulated, digital version of the SWB can be used, which actually does provide some simulated slowcontrol functionalities.

With actual hardware present, data generators in the firmware can be used to produce a fake datastream. The highest hardware data generator is located before the PCIe interface. In the lower layers of the DAQ, there are additional generators, so the components there can also be tested.

On the MuPix FEB, a single multi-purpose generator is built from linear feedback shift registers (LFSRs). A first LFSR is used to generate hits with a pseudo-random

column, row and timestamp in each clock cycle. The output data rate can be configured from the slowcontrol system. In order to make the output timing patterns more realistic, additional LFSRs are used to decide for each clock cycle whether or not the generated hit is actually sent to the output of the generator.

The rate is adjusted from the slowcontrol by changing the conditions for which the secondary LFSRs will allow the output. These conditions can be made more or less likely, which defines the average output rate. This method implements an adjustable and deterministic average output rate but also produces a pseudo-random time structure of the outgoing hits, which is more realistic than a strictly regular output pattern. The pseudo-random output time structure is, of course, also regular but on a much larger timescale. Providing realistic generator data is essential for system tests. It is conceptually of similar importance as finding good test vectors for firmware simulations: the simulations will only discover edge-case errors in the firmware if the tests are realistic enough to produce them.

The stream of output hits can then be injected either at the input or output of the hitsorter. Additionally, the possibility was added to inject single hits with a column, row and timestamp written from the slowcontrol system.

It is also possible to configure the MuPix and MuTrig ASICs to inject fake signals. For the MuTrig, this feature is actually used for calibration purposes. The pixel sensor does currently not plan on using this functionality. It would be controlled via the sixth configuration shift register on the MuPix, which was intentionally ignored during the discussion of the configuration upload firmware. However, it is still accessible via a modification which will be shown in the next chapter, but it is very inefficient to do so since the system was not optimised for it.

4.13. DAQ Summary

This chapter summarises the author's contributions to the data acquisition and control systems of the Mu3e experiment. A design for an FPGA-based multi-layer DAQ system was presented, which is capable of processing the expected data rates and provides the infrastructure for configuration and monitoring of the Mu3e detector. The author's work has focused on the lower layers of the system, while the design for the upper layers was mainly produced in [71].

Unless mentioned otherwise, the components discussed for the lower DAQ levels have been implemented and are operational. Parts of the system were verified by the author against the detector ASICs using functional HDL simulations. The seed variation methods shown in chapter 3 were used to evaluate the timing behaviour of the resulting FPGA designs. Test runs will be discussed in chapter 6.

5

System Variations

The development of DAQ system elements shown in the previous chapter represents the primary contribution of this work to the Mu3e experiment. However, the construction process of a detector system does not just involve the development of the data acquisition system but will also require variations of the DAQ system for applications outside the context of the final Mu3e detector. Such applications are, for example, the development and construction of Mu3e detector elements and their quality control test procedures. Additionally, parts of the Mu3e system will be used with some modifications in other experiments. This chapter will cover the contributions of this thesis to DAQ system variations.

5.1. Dummy FEB

A particularly simple modification is the introduction of the dummy FEB firmware. The dummy version of the FEB removes the detector-specific part from the ArriaV and only leaves the common firmware parts. Since the communication and infrastructure aspects of the DAQ are exclusively handled in the common part, the FEB will still function in the DAQ system but will not be able to read out any detector.

This is particularly useful for DAQ infrastructure tests since the resource usage of the dummy version is much lower compared to the other FEB flavours. It does, therefore, not come with their timing issues and allows faster development cycles due to the lower compile time.

It is also safe to connect this version to any of the three subdetectors. Normally, a MuPix FEB firmware accidentally uploaded to a FEB connected with a scintillating fibre detector might cause damage since the input and output pin distributions on the backplane are not completely compatible. The dummy FEB version does not drive any of these pins on the backplane and can, therefore, be used in both cases. For this reason, a tested version of the dummy firmware will be used as the emergency image, which was mentioned in section 4.11.4. Any infrastructure-related functionalities are included in the common firmware. Therefore, the dummy firmware is also able to perform a firmware upload to the flash memory. It is likely that the dummy FEB firmware will be used as the default image for any FEBs not currently attached to a detector in order to prevent any accidental damage once they become connected to something again.

5.2. Operation without DAQ Layer 3

Developing, testing and characterising detector components has been a consistently ongoing task for many members of the Mu3e collaboration in the last few years. Sensor characterisations or component tests often require a DAQ. The system used for this purpose is based on the Mu3e DAQ but not tailored towards the reconstruction of $\mu \to 3e$ muon decays. The GPU selection in the third DAQ layer is not useful in these scenarios. The requirements shift from $\mu \to 3e$ reconstruction towards the ability to write a high bandwidth to disk in more portable systems, which are available in higher numbers and can be deployed at multiple institutes participating in Mu3e.

To achieve this, the third layer of the Mu3e DAQ is dropped and only a single SWB mounted in a desktop-PC is used. Additionally, the hardware used for the SWB layer is replaced by a commercial DE5a-Net development board since the PCIe40 – which is nominally used there – is produced by the LHCb collaboration and is not available in large numbers. Porting the firmware to this new hardware is not a concern since it uses the same Arria10 FPGA and is also used for the receiver boards in layer 3 of the Mu3e DAQ.

The firmware on the SWB replacement combines the slowcontrol functionalities of the nominal SWB with a new readout method. Instead of sending data to the third layer for the GPU reconstruction, the firmware builds MIDAS events and sends them to the MIDAS buffers via DMA. Building the event according to the protocol defined by MIDAS saves CPU time and can increase the reachable bandwidth. The development of the MIDAS event builder firmware was done in [71] and is discussed in more detail there.

Regarding hardware, the other Mu3e DAQ component with limited availability is the clock and reset distribution system. The clock and reset box shown in figure 55 only exists twice and needs to be replaced with another solution here. The options for this will be shown in sections 5.3 and 5.4.

On the side of the frontend board, only minimal firmware modifications are necessary. Depending on the test system, the number of sensors and the connector type used for them changes, which produces a variety of FEB versions which mostly only differ in their pinout configuration towards the detector adapters. Mupix setup versions which use the SPI configuration method instead of the new protocol do not require any additional firmware change since the configuration upload was written fully compatible with SPI. This was discussed in section 4.6 of the previous chapter.

5.3. Other Reset Options

The standard reset link controls state changes of the FEB via an 8b10b encoded 1.25 Gbit/s optical link from the clock and reset box shown earlier in figure 55. Alternatives to this system either have to produce this optical signal or implement another method for state changes on the FEB.

The first concept is implemented on the SWB replacement from section 5.2 by

emulating four reset lines with one of the optical output modules of this board. The commands are identical to the normal reset system and are controlled via the PCIe interface. Since they are sent simultaneously on all four outputs, this method provides synchronisation between the connected FEBs equivalent to the actual clock and reset box. However, the number of FEBs is limited to four.

The other option is to control state changes on the FEB without using an optical input signal. This method bypasses and disables the state controller on the FEB and uses write requests to predefined slowcontrol registers to induce state changes. State changes are still acknowledged by the data merger as defined in section 4.4.2. When this option is used, the SWB software will automatically bypass any actions from the clock and reset software and send broadcast SC commands instead to accomplish the requested state change. Since the NIOS is also connected to the slowcontrol system on the FEB, a state change can also be requested manually by a user connected to the USB terminal.

The slowcontrol-based reset replacement does not ensure synchronisation of all connected FEBs since it is distributed in the 156.25 MHz domain. However, the resulting time deviations on the FEB will likely be limited to one or two 125 MHz reference cycles since the signals are broadcasted from the SWB and not sent individually to each FEB.

5.4. Other Clocking Options

Similarly to the reset link, an optical clock signal can be emulated by the SWB, which is the standard method used for Mu3e DAQ setups outside of the Mu3e magnet. Alternatively, the 125 MHz reference clock can be provided electrically via an LVDS pair of cables to the FEB. In this case, the configuration of the second clock distribution chip needs to be changed to use the electrical input as the reference clock.

Another option is to drive the second clock distribution chip from an oscillator on the FEB, which allows the FEB to operate disconnected from other system components. User interactions are only possible via the USB terminal in these cases, but most functionalities can still be used via the connection of the NIOS to the slowcontrol system. High-speed data readout is not available since a stable connection from the FEB to the SWB cannot be assembled due to the missing reference clock between them.

5.5. Zero Suppression

During normal Mu3e operation, the GPU selection algorithms are used to reduce incoming data to an amount that can be written onto a disk for offline analysis. For the system variations in test and quality control setups, this data reduction step is not present and the system will aim to write all received data to disk.

The issue with this is the resulting protocol overhead. Section 4.7.2 defined the structure of data packets from the FEB to the SWB. A data sub-header has to be

sent on average every 16th cycle in this protocol. With the presence of hits, the concept of sub-headers saves bandwidth since it avoids repetitions of timestamp information. Without any hits, they are still needed in the full system to form a time-sorted stream for the GPU farm. This is not necessary anymore for most systems which do not include the GPU farm. Empty sub-headers are just unnecessary data in these cases.

A single FEBs datapath sends 32-bit data words out with a frequency of 125 MHz. On average, every 16th cycle is a subheader. Therefore, a single FEB produces at least 31.25 MB/s which are written to disk if the system is operated without layer 3. That is a significant fraction of the usual bandwidth to a hard drive and with 3-4 more FEBs the system would be completely occupied with writing empty data frames, even without any detected hits.

For this reason, zero suppression options were introduced to improve data storage efficiency and analysis speed by eliminating the storage of unnecessary data. An entity for zero suppression is currently connected to the data output of each data demerger on the SWB. For a previous version, it was connected on the FEB side of the optical link, which does not make a difference in this case. When the sub-header zero suppression is activated it will receive a sub-header word and store it until either a hit or the next sub-header word arrives. When no hit arrives between the stored sub-header and the next sub-header, the word will be discarded.

A header zero suppression was built on top of this idea. When enabled, a header will be stored until the first sub-header is seen, which indicates that the packet contains hits since the sub-header was not removed by the previous sub-header zero suppression. When no sub-header arrives, the complete packet will be discarded.

The advantage of a zero-suppressed readout is obvious. The resulting file sizes are smaller and only contain actual particle detections. Additionally, the number of FEBs that can be read out simultaneously has increased. However, the concept of the time alignment tree [71] on the SWB does not work anymore since the subheaders are missing and an alternative, sequential readout of the buffer FIFOs needs to be used.

5.6. Reference Trigger Inputs

When the Mu3e DAQ has to be used together with other detector systems, these other components either have to be fully integrated into the DAQ or time markers have to be put into the Mu3e datastream to be able to reference events in the separate system.

These reference inputs or trigger inputs replace one of the usual sensor datapaths on the frontend board. When a reference input signal arrives, a hit is created and inserted to one of the inputs of the sorter entity in section 4.7.1. The timestamp of this hit is created from a 125 MHz counter on the FEB. Since this counter runs synchronised to the other detectors in the Mu3e DAQ, it can be used in the analysis to correlate events from the Mu3e system with the external input.

In this version of the system, the timing precision of the reference input is limited by the 125 MHz counter frequency on the FEB. Further improvements were implemented with the intent to increase the precision of externally provided time references. This

became relevant in some situations where multiple external references were present in the Mu3e system or references to the scintillating detectors had to be made. For the analysis of relations with the timing of MuPix hits, these improvements would not have been necessary since the time resolution of a MuPix is already worse than sampling with a 125 MHz clock.

The first step to increase the reference timing precision is to add a faster reference counter on the FEB. The 125 MHz counter is still providing the timestamp bits used by the sorter in order to ensure proper propagation of the hit through the rest of the Mu3e DAQ. However, the bits that are normally used for column, row, and tot in the MuPix hit are replaced with more precise timing measurements from the faster counter. These locations in the hit are never modified again by the DAQ and will end up unchanged in the output file.

The reference timing precision in this modification is limited by the fastest clock that the FEB can use in internal user logic, which is 625 MHz due to the device limitations of the ArriaV FPGA.

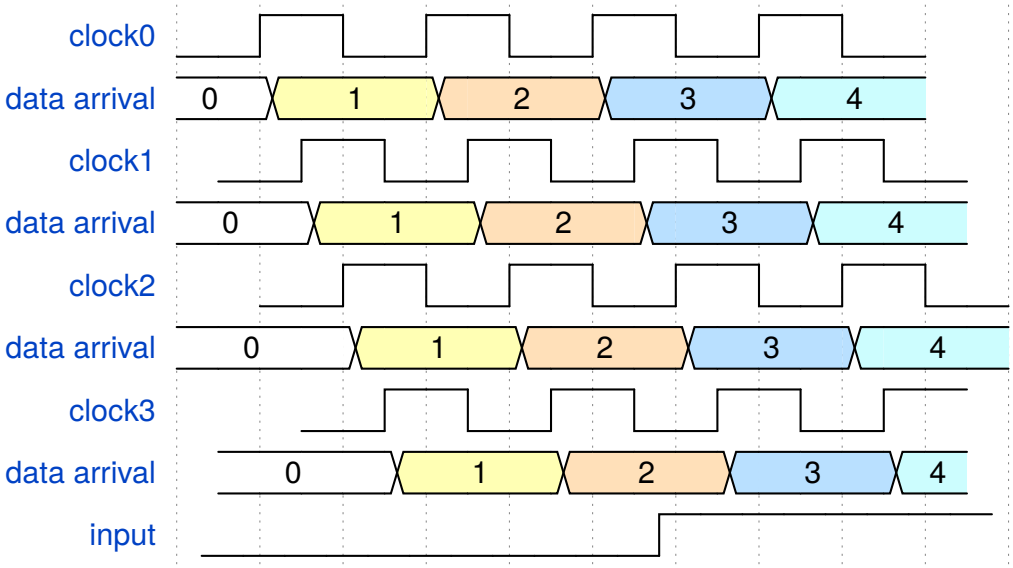


Figure 85.: Usage of clock shifts to increase timing precision. The clocks 0 and 1 will register the arrival of the input in cycle 3, clocks 2 and 3 will measure it already in cycle 2. The average of these measurements is more precise than each individual one.

A further improvement is possible by the method shown in figure 85. Multiple fast clock domains are created with shifted sampling points. Sometimes not all clock domains will register the reference input at the same counter value. The average of these counter values can be used to improve the timing precision since the number of domains which registered the input in cycle N vs the number of domains which registered it in cycle $N + 1$ contains information about the arrival time of the input signal.

This modification of the Mu3e DAQ was, for example, used for time-of-flight particle identification between e^+, μ^+ and π^+ in [75] or as a cosmic trigger reference in section 6.2 of this thesis.

5.7. Timestamp Replacement

Using a similar idea to the previous section, the timestamps of particle detections can be replaced with the FEB 125 MHz counter value at the time of arrival of the hit on the FEB. This concept was first used during a testbeam, where tuning issues of the involved sensors did not allow them to deliver reasonable timing information. An option controlled via an SC register was introduced to replace the time information with the arrival time. This reduced the timing precision significantly but allowed this specific testbeam to proceed with the measurement.

Although this idea came about in a distress situation, this is a feature that could be used for further studies in the future since it allows an analysis of delivery times between a hit on the MuPix or MuTrig and the arrival on the FEB. These delivery times are relevant for the size and position of the hit acceptance window in the Mu3e hitsorter and will depend on the total hit rate on the sensor. One possibility for such a test would be to copy the output of a MuPix sensor on the FEB to a datapath which was foreseen for another sensor and replace the timestamp with the arrival timestamps for this copy. This should allow an analysis of delivery time depending on the rate and position of the hit on the sensor[1].

5.8. QC histogramming

For quality control (QC) procedures it is sometimes relevant to record the response of a MuPix sensor accurately under increased noise conditions. This poses a challenge for the type of system discussed in section 5.2 since the readout method there can be saturated, which leads to the loss of hit information. For these types of tests, the actually relevant information is the noise level for each pixel and not the recording of each individual hit on the sensor. For this reason, a quality control histogramming was connected on the SWB, which allows snapshots of the hit distribution map for a specific MuPix sensor to be taken. This histogramming entity is dynamically configured to connect to a specific MuPix and FEB and will count the hits for each pixel for a given duration. Data loss is not possible for this method since the hits are counted in hardware and not transferred to the software first. After the hits have been counted for some time the histogram can be read out via the slowcontrol system. This provides a reliable way of measuring noise levels unaffected by most readout limitations.

[1] It is probably also possible to simulate this instead by using the design files.

5.9. Operation without DAQ Layer 1

Other projects are looking into the use of MuPix sensors for beam monitoring purposes. Either as part of a detector system or as a tool during development. Some are already using MuPix sensors ([76], [75]), and others are planning for modifications of the readout hardware. One of these projects ([77]) plans to operate the MuPix sensor without the use of a frontend board by moving the frontend board firmware shown in the previous chapter almost completely to the Arria10 on the SWB. The MuPix would then be operated via optical cables directly from the FPGA there. This will form another system variation of the Mu3e DAQ. Apart from porting firmware components, the control interface of the MuPix shown in section 4.6 will need to be modified since the optical modules are not able to drive constant values. Developments in this direction will be available in [77].

5.10. Manual MuPix Commands and Readback

Commands to the MuPix do not just upload the global and TDAC configuration as shown in section 4.5. They can also be used to set up the injections from section 4.12 or to read a configuration back to the FEB. Additionally, the MuPix includes an ADC, which can be configured to measure different voltages on the sensor. The ADC functionality is steered by a separate control command. Injections are controlled via the sixth configuration register, which was ignored during the discussion in section 4.5.

These options are currently not heavily used for sensors in a Mu3e DAQ system, and it is questionable whether they will be used at all during the normal operation of the Mu3e detector. For this reason, the slowcontrol architecture on the FEB was not optimised for these functionalities. However, they are still reachable by sending 64-bit commands to the MuPix manually.

Every connected MuPix has an instance of a command assembler entity on the FEB, as discussed in section 4.6.2.2. This entity decides which command should be sent and reads the payload data from the mirror registers. The command assembler entities are equipped with an additional 64-bit input for manual commands. A command applied to this input will only be considered in the decision when none of the mirror registers, including the TDAC and load cycles, are currently able to provide one. If that is the case, then the command from the manual input will be selected and provided for pick-up to the lane controller. This implementation ensures that manual commands also automatically adhere to the tick alignment, configuration deadtimes and the interleaving command order on the slowcontrol line of the ladder.

The manual inputs are exposed to the software via the slowcontrol system. Each input is connected to two slowcontrol addresses and is written whenever a write command from the SWB is targeting this address. Apart from the timing aspects above, the Mu3e protocol is not produced by the firmware in this case and has to come from the software.

Optimisations to this method should be considered if one of these features requires frequent use during the operation of Mu3e at some point. In the current system an efficient use of them is not foreseen.

During the development of the MuPix configuration system the method shown here was considered as an alternative to buffering data on the FEB. The SWB could buffer the data instead and automatically write slowcontrol write commands to the manual input ports of the command assembler entities without a direct involvement of the software. This would remove the issue of just-in-time delivery, which was mentioned in the MuPix configuration section, since the precise timing of commands would still be taken care of by the FEB. Should the resource situation on the ArriaV become worse in the future, then this is something to be considered again.

For the SPI configuration method a similar solution was implemented by providing a manual software access port to the SPI line. However, this is even less efficient than manual 64-bit commands.

The reply from the MuPix to these ADC or configuration readback commands is sent via the data output lines. The reply occupies a specific location in the output protocol and can be distinguished from hit data. For each MuPix, a small FIFO was inserted in the datapath which collects these reply words. The read port of the FIFO is connected to a slowcontrol address and can be read from the software. It is currently not foreseen to automatically forward this reply data to the higher layers of the DAQ. As mentioned above, this should be further optimised if these features are required frequently in the future.

The readback of configuration data is implemented via the *RB* line in figure 86. This signal can be controlled via a separate command and copies the data from the output latch to the latch controlled by $CK1$. Afterwards, the data can be shifted towards the end of each configuration register. Data which is shifted out at the end is sent towards the FEB and then also written to the FIFO there. The software has to keep track of the order of replies expected in the readback FIFO.

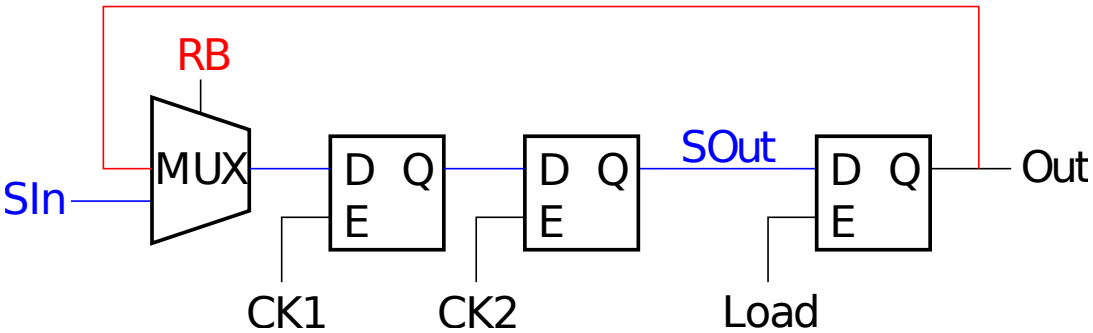


Figure 86.: A single cell in a MuPix configuration shift register. Taken from [20].

In principle, it would be possible to automatically compare the returned data for the three global configuration registers with the data that is still left over from the write process in the mirror registers. This could be used to build a fully automatic

write validation function. This is possible since the mirror registers are implemented as circular shift registers, which still contain a copy of the configuration data after the write process.

However, the idea of automated write validation has not been explored further in this thesis. An unstable configuration line that leads to incorrect settings on the MuPix can also easily produce an unstable data output line. However, the data output line can also be unstable with the correct settings. The readback of configuration data via this output line can then introduce additional mismatches compared to the content of the mirror registers on the FEB. Therefore, the validation idea can only work in cases where the configuration is received correctly on the MuPix. It can also fail in these cases if the output link is unstable despite a correct configuration, which somewhat contradicts the concept of a validation.

5.11. Mupix Gating

The P2 experiment [78][79] is planning to measure the weak mixing angle $\sin^2\theta_w$ for a low momentum transfer using electron-proton scattering of a 150 μA electron beam. The P2 tracking detector will be built using a variation of the MuPix sensor. The expected electron rate of 0.1 THz [78] does not allow a fully streaming DAQ concept as in Mu3e. The data rate needs to be reduced by adding a gate signal on a sensor level to enable the MuPix for some time period and disable it again.

Gating a MuPix sensor is, in principle, possible by uploading a new configuration. A first configuration contains settings where the threshold or some other setting is changed to disable hit detection, the standard configuration is uploaded for data taking and then the configuration is changed back to the original one. The issue with this idea is that it takes time to upload a configuration to the MuPix. The gate would be open for at least this time period.

A faster gating method for the MuPix was developed during this thesis and is shown in figure 87. The idea is to repurpose the three latches in figure 86 and the readback function from the previous section. The signals $CK1$, $CK2$, RB and *Load* in a MuPix register cell can be steered individually with the SPI interface or the direct output pins. This can be used to store and switch between two different sets of configuration bits in the MuPix.

In a first step, configuration 1 (config 1 in figure 87) is fully written and loaded. Then, the second configuration is written but not loaded into the last latch. The RB signal is applied and configuration 1 is copied back into the first latch with a pulse on $CK1$. This is the starting position before the gate is opened. Configuration 1 contains settings which disable the sensor.

From this state, it is now possible to switch the positions of config-bit 1 and config-bit 2 by just four pulses on *Load*, $CK2$, followed by RB and $CK1$. The *Load* pulse overwrites the content of the third latch with config-bit 2, which enables the MuPix. Then, the disabling configuration is moved into the second latch with a pulse on $CK2$ and the enabling configuration is copied back into the first latch with a pulse on

RB and *CK*1. At this point, another *Load* signal would close the gate again. The contents of the first and second configuration have switched places and the process can be repeated again. The idea was tested using the SPI interface.

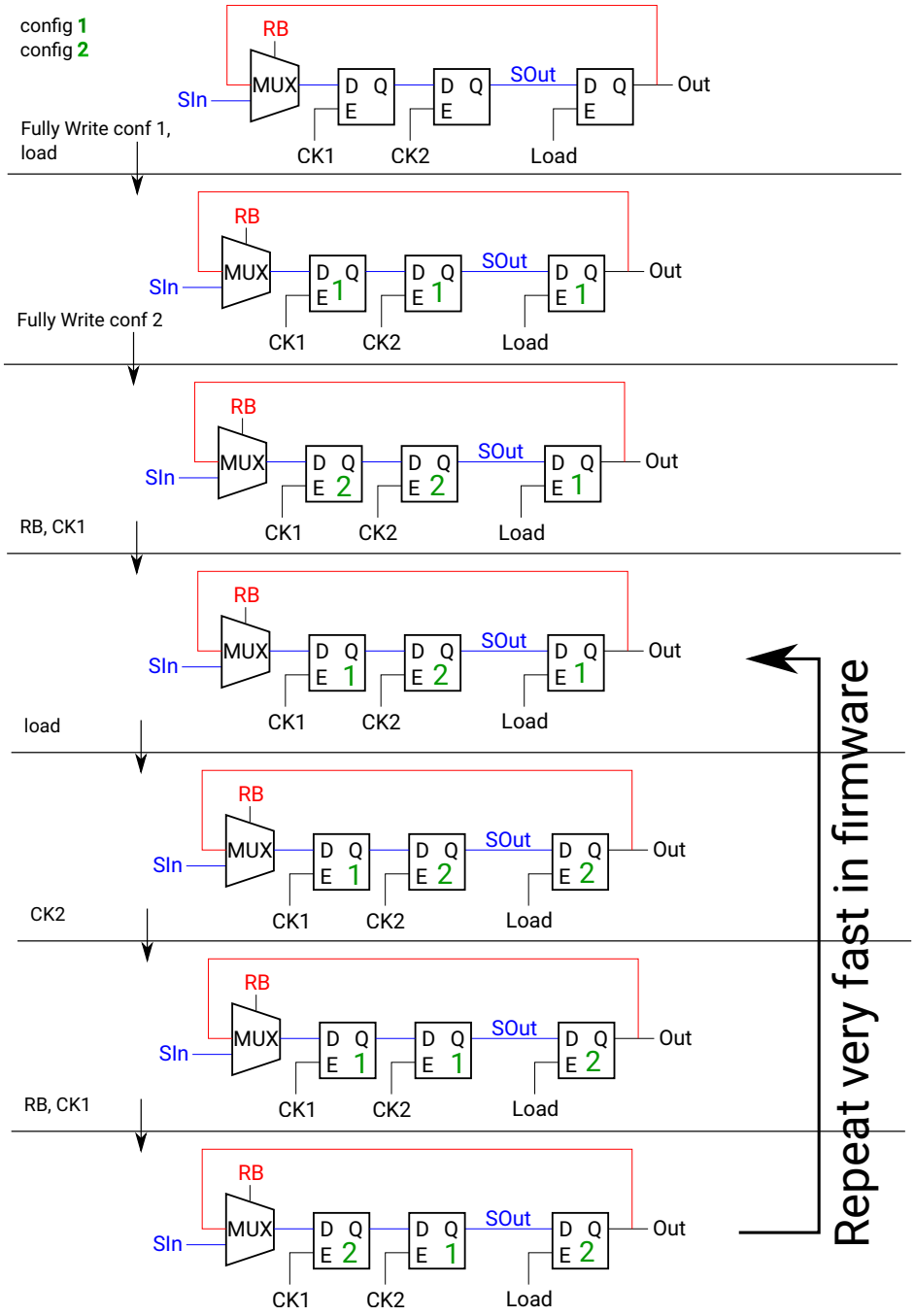


Figure 87.: Gating concept for a Mupix 11.

This is not the intended design purpose of a MuPix cell, but it allows switching between two configurations in just a few clock cycles. The drawback is that the SPI interface or direct input pads must be used to control this process. The slowcontrol statemachine on the MuPix would be perfectly able to do this on command, but the idea came up after version 11 of the MuPix was already produced. This is the disadvantage of ASICs: once they are produced, they cannot be changed anymore. The MuPix 11 fulfils all requirements of the Mu3e experiment, and therefore, it seems that version 11 will be the final version of the sensor. Should this not be the case, the addition of a „Exchange configurations“ command in the Mu3e protocol could be considered.

Another aspect to consider is the stability of the output link during configuration. It was observed under lab conditions that a configuration can be changed without causing 8b10b errors, but generally, the link stability seems to be negatively affected during a configuration process. To which degree that would apply to the idea in figure 87 is an open question. The P2 experiment has opted for the production of a separate MuPix variation which includes an input pad for a gate signal.

5.12. Temporary System Splits

The preparations for the test runs of the next chapter have made it clear that during tuning and other test operations of the Mu3e detector, it is not always beneficial to have all subdetectors connected in the same DAQ system. During run preparations, the different detector groups have to perform tasks specific to their subdetector system, which can collide in terms of run starts and stops or required readout modes.

For following runs, it should, therefore, be foreseen to split the Mu3e DAQ into four SWB sections. Each SWB server could temporarily act as the MIDAS host system and use reset commands via the slowcontrol for state changes of the connected FEBs. This would bypass the clock and reset system and would also not require the emulated signals discussed in section 5.3.

Operating the Mu3e DAQ in four separate sections has the advantage that the people working on the subdetectors can work independently on their detectors and do not have to share a single DAQ at all times.

5.13. Alignment

After the construction of the Mu3e detector, the absolute position of each pixel on each MuPix sensor is only known to the accuracy of the expected assembly tolerances. Therefore, an alignment must occur to reach the targeted spatial resolution. As discussed in [80], large parts of this alignment process can be based on tracks from muon decays. However, some movements and deformations of the detector will not be detectable in this way and represent weak modes of track-based alignment. One of these weak modes is shown in figure 88.

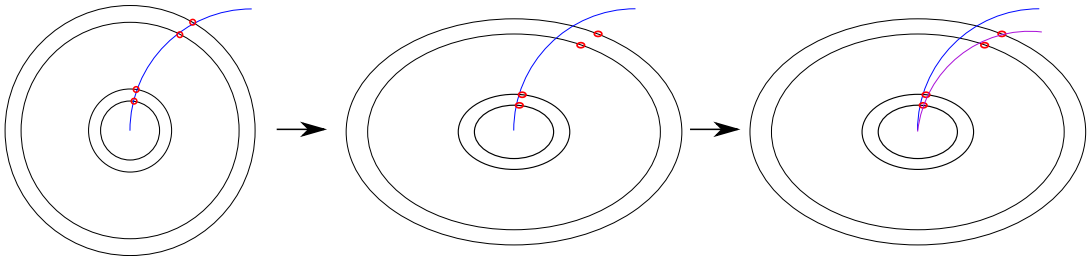


Figure 88.: Illustration of a deformed Mu3e barrel. The actual track (blue) will be reconstructed as the purple track if the reconstruction software is unaware of the deformation. The χ^2 of the reconstructed track can be unaffected. Image taken from [80].

The deformation shown in the figure above is not detectable by reconstructing the blue track since a track with a different momentum will lead to the same χ^2. More of these weak modes, for example relative station movements, are listed in [80]. The discussion there came to the conclusion that external measurements and the trajectories of cosmic muons are needed to correct these weak modes.

The need for external measurements and cosmic muon tracks results in the requirement of additional functions in the DAQ system. The reconstruction and selection of timeframes is optimised for muons decaying via $\mu \to 3e$ and does currently not include the identification of high-energetic cosmic muons. Additionally, a readout method for the external measurements needs to be provided.

5.13.1. Camera Alignment

The proposed solution for the external alignment measurements is a camera system mounted on the detector cage [15]. The cameras will be mounted in a way where they can see each other and reference points on the detector module. This will allow them to calculate relative movements. First measurements with a prototype of the system have been done in [81] and will be continued in [82].

For the data acquisition, the task is to read the images produced by the cameras and move them out of the detector. It is planned to use the existing infrastructure in the form of a frontend board for this task. Other collaboration members have already started the development of the necessary firmware, but it is not completed and is currently not operated on a frontend board. The idea for the implementation is to consider the cameras as another type of sub-detector and replace the sub-detector-specific part of the FEB firmware with the camera readout. Whether the camera data will be read out via the slowcontrol system or sent as detector data packets is still to be decided. However, the readout of the cameras will operate separately from other detector parts and will not need synchronisation or other hardware-level interactions with detector data. Further work is needed to integrate the camera readout into the DAQ [83].

5.13.2. Cosmic Alignment and Cosmic Trigger

In principle cosmic tracks are already recorded with the DAQ system presented so far. The GPU farm will select time frames with a $\mu \to 3e$ signature. These selected time frames will include any cosmic particle trajectories that happened to arrive in this time window. However, this means that only a small fraction of the cosmic muons will be available in the offline data. Higher efficiencies for cosmic muon detection require an additional online selection which searches for cosmic muon tracks. Multiple implementation options are currently considered. One of these options was investigated during this thesis and will be discussed below.

The idea is to build a hardware cosmic trigger on the Arria10 FPGAs of the receiver boards in the GPU farm using only the pixel data of the outer two tracking layers. When a cosmic particle passes through the Mu3e detector, it is likely to hit both outer tracking layers twice. There are some inactive areas, so not every cosmic particle will produce four hits, but we will not consider these cases here. Each point where the particle enters or exits one of the outer tracking barrels will be located on a different MuPix sensor. Therefore, a cosmic is likely to hit at least four of the 2844 MuPix sensors in the system.

For each data frame, the sensor IDs which have recorded a hit can be marked in a 2844-bit vector (top of figure 89) on the Arria10 FPGA. A single four-hit pattern can now be tested by calculating the logic AND of the according four registers in the vector. Multiple tests of four-hit patterns can be combined by a logic OR afterwards. Overall, there are

$$\binom{2844}{4} = 2.72 \cdot 10^{12} \tag{16}$$

possibilities of 4 hit patterns. However, the track of a high energetic cosmic muon is approximately a straight line and will not be able to produce most of these patterns. The fact that straight lines can only produce a limited amount of four-hit patterns can now be used to build a trigger decision using the method explained above and shown in figure 89: A pattern in the 2844-bit hit vector is tested by combining the four register outputs with a logic AND, followed by a logic OR to combine the output with all other tested patterns.

The next questions are how many of these pattern checks can be implemented with the available resources and is that enough to result in a sufficient cosmic detection efficiency. The 2844-bit vector will require the same amount of registers on the FPGA and is negligible in terms of resource usage. Each pattern will require a single ALM to calculate the AND of the four registers. The following combination in an OR logic will require a number of ALMs which depends on the number N of patterns which should be combined. In the Arria10 FPGA, each ALM can be configured as a 6-input OR element [84]. Therefore, the first layer of OR-combinations will need N/6 ALMs, the second layer will need $N/6^2$ ALMs and so on. Overall, the structure will require

$$N + \frac{N}{6} + \frac{N}{6^2} + \frac{N}{6^3} + \frac{N}{6^4} + \cdots = N \sum_{i=0}^{i=\infty} 6^{-i} = 1.2\ N \tag{17}$$

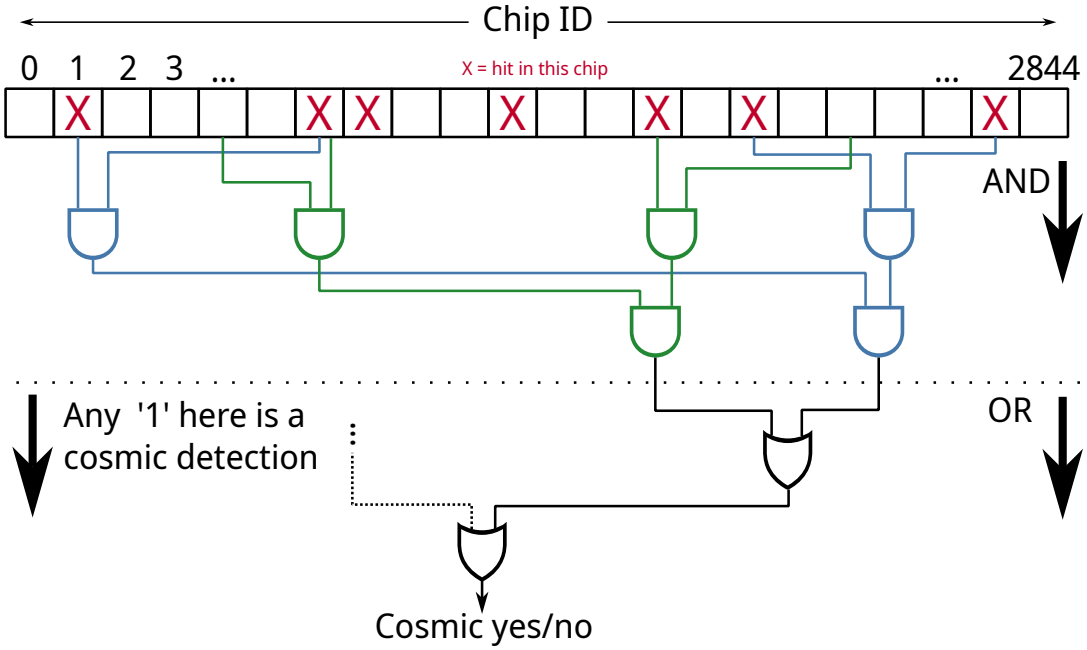


Figure 89.: The conceptual idea for a firmware cosmic trigger. In the upper part of the figure, the pattern detection logic is connected to the chip ID vector. For simplification, only two patterns (blue and green) are shown here. In the lower part of the figure, an OR of all detection patterns leads to the trigger decision.

ALM blocks. Each Arria10 in the GPU farm has about $4 \cdot 10^5$ ALMs and could maybe test $2 \cdot 10^5$ patterns. It is not possible to use all of the ALMs for pattern tests since the other firmware components will also require resources. Combining all FPGAs in the GPU farm, a million pattern tests seem to be reachable[2]. Combining FPGAs of the different farm servers is not an issue since it does not matter on which one the cosmic is detected.

The Mu3e geant4 simulation in the cosmic-only mode was used to generate this set of patterns and to test the detection efficiency. Whenever a cosmic produced four hits in the pixel tracker in a chip combination that had not previously occurred in the simulation, it was added to the pattern set. This was repeated until the set had a size of 1.15 million unique patterns. During the creation of this set, it was regularly tested against a reference simulation file with a different seed to measure the expected cosmic detection efficiency. The result is shown in figure 90.

For the $1 \cdot 10^6$ pattern tests, which could be implemented with the available resources, a detection efficiency of 85 % is reached. However, the same pattern set was then also tested in the standard Mu3e simulation with the nominal phase I muon rate and accepted 46 % of the frames as false positives there. A false-positive percentage of

[2]This is not an exact number since the number of farm servers is currently not final.

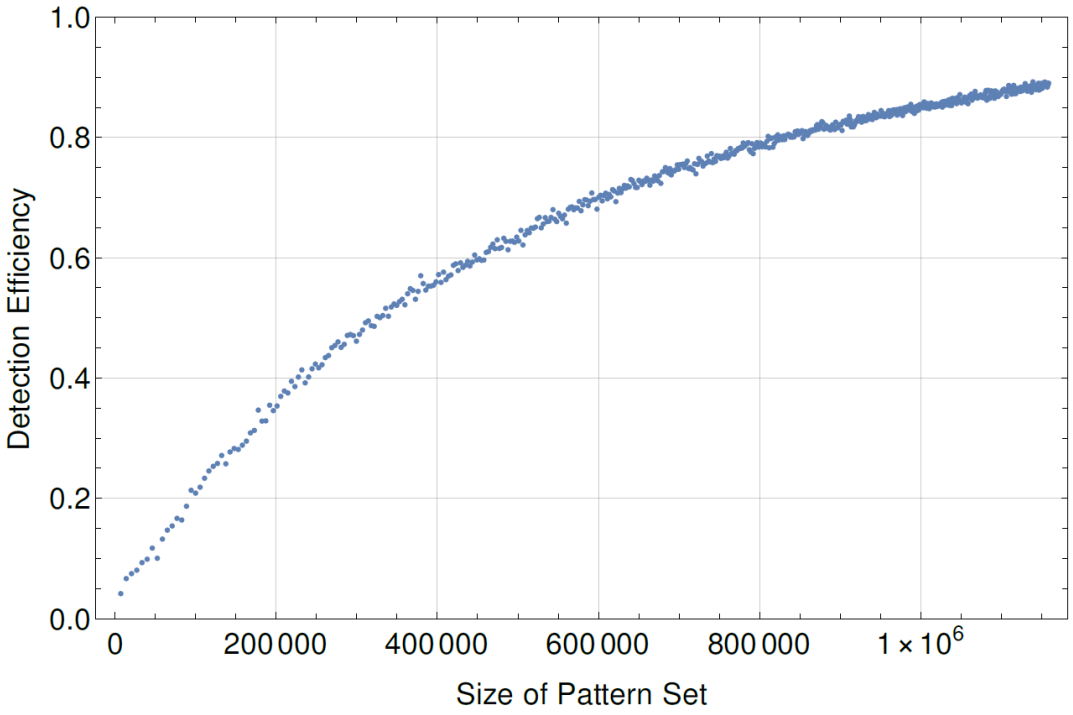


Figure 90.: Detection efficiency of simulated cosmics for different sizes of the pattern set.

46 % is too large for this method to be viable on its own. Improvements might be possible by a reduction of the frame length or an increase of the 2844-bit vector by using sensor quadrants or even smaller partitioning instead of the sensor ID. This would decrease the false positive rate but also require more patterns to reach the same cosmic detection efficiency. The resources for this step are not available on the farm FPGAs.

The unchanged concept is able to reduce the data rate by 54 % while keeping most of the frames with cosmic muons. This could already be used for a preselection in combination with other cosmic selection methods.

One of these other methods would be to run a cosmic reconstruction on the GPU, conceptually similar to the normal Mu3e track reconstruction. The issue with this idea is that the standard reconstruction is based only on the central station's tracker data. A cosmic reconstruction would need to consider data from all three stations, which would increase the hit combinatorics and require the FPGA to convert all tracker data to physical coordinates before sending them to the GPU. The additional GPU server(s) for this task could be added at the end of the standard GPU daisy chain.

Another option that was considered by the collaboration was the addition of large

scintillators above and below the Mu3e detector or magnet. Limited available space complicates such an addition. The readout of scintillators for a cosmic trigger would be included in the DAQ according to section 5.6. The data frames containing a cosmic trigger could be flagged and read out at the first farm server.

Further studies on cosmic pattern training can be found in [85]. The discussion there also includes a potential use of associative memories for a cosmic trigger.

5.14. Phase II

The high-intensity muon beamline (HIMB) upgrade at PSI is estimated to increase the muon stopping rate on the Mu3e target to $2 \cdot 10^9$ μ/s [12]. This second phase of the Mu3e experiment will require some changes not just in the detector but also in the DAQ. An order of magnitude increase in data rate will run into a bottleneck at the input to the first farm server, which is currently limited to 160 Gbit/s due to the available optical input bandwidth of the FPGA receiver board.

If the processing concept of timeslices on single GPUs should be kept, the data of each timeslice still needs to be served to a single server. Doing so either requires a receiver board with a larger optical bandwidth or a form of preselection of the time slices before they are transmitted to the GPU farm.

A preselection would break the idea of a single daisy chain of GPU servers since some frames would be missing in the datastream. However, it could be possible to keep this concept by distributing time slices into multiple GPU daisy chains. The optical output bandwidth of the SWBs is not fully utilised in the phase I design and could be used to serve additional GPU server chains. Once the existing SWBs outputs are fully used it could be possible to implement a second set of SWBs by passively splitting the optical cables from the FEBs. This second set of SWBs would double the available output bandwidth and could serve more GPU chains. However, the total number of GPU servers will likely not increase by an order of magnitude since changes in the detector hardware will aim for a better timing precision in the tracker [12], which would reduce the computation requirements since it reduces the amount of hits which need to be considered for track reconstruction.

An alternative would be to increase the optical bandwidth of each GPU server. This does not necessarily require the use of a receiver board with more optical bandwidth. It could, for example, also be implemented by mounting more than one receiver board in each farm server. However, these are just ideas at the moment and the architecture of the Mu3e phase II DAQ will need further investigation. It will likely be possible to reuse large parts of the phase I DAQ developments.

6

Test Runs

The design and development of the data acquisition do not end once the experiment starts operation. The practical experiences and real issues during the experiment's lifetime provide input to the design of the DAQ and will result in adjustments to the system over time. A perfect DAQ on day one is not likely. The situation for other parts of the Mu3e detector is similar: practical operation of the detector is likely to unveil some aspects that were not considered during the initial design phase.

The Mu3e collaboration has conducted two test runs with the intent to gain such operation experience with a prototype of the Mu3e detector. The goal of these test runs was not sensor characterisation or physics results but to provide input and proof of concepts for constructing the actual Mu3e detector. For the prototype system, final hardware was used wherever possible in order to construct something that is close to the final design.

This chapter will discuss these two test runs in terms of their consequences for the design of the Mu3e DAQ. The test runs also provided an opportunity to validate the DAQ under real conditions. This is different from a DAQ performance test. A functional DAQ is not a test result but a consecquence of designing the system correctly. The test runs have been used to find bugs or aspects which had not yet been considered.

6.1. Integration Run 2021

The first test run was performed in 2021 with a prototype of the inner vertex tracker mounted in the actual Mu3e magnet with beam on the final target. Mu3e timing detectors could not be operated due to hardware issues. The DAQ consisted of ten frontend boards connected via optical cables to a single SWB and the final clock and reset distribution in the Mu3e server room. The variation discussed in section 5.2 was used since the GPU farm did not exist at that time. Only a fraction of the data from the FEBs was read out on the SWB since the data rate exceeded the SWB server's disk write speed.

6.1.1. The Mu3e Vertex Detector Prototype

The construction of the Mu3e pixel tracker and also the construction of the vertex prototype used here are discussed in detail in [16]. 108 MuPix sensors are arranged in two barrel-shaped layers around the target. Groups of six sensors are mounted on U-shaped PCBs instead of flexprints since the flexprints were not available in 2021.

The dimensions are slightly larger than the nominal vertex tracker design and the tracker has some gaps between the pixel ladders. The reason for this is the height of the connectors on the ladder PCB, which does not allow the overlap planned in the final design [16]. A Comparison is shown in figure 91.

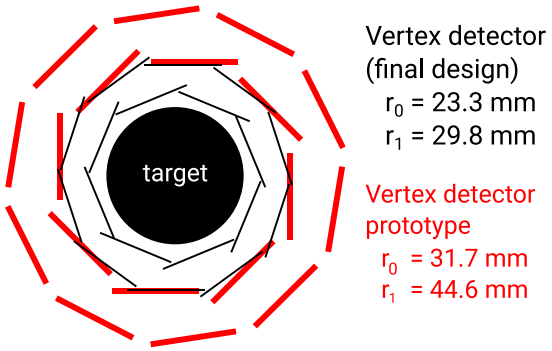


Figure 91.: Comparison between the vertex detector prototype and the final geometry. Adapted from [16].

The PCBs with the MuPix sensors are mounted on a 3D-printed structure, which is attached to the beam pipe. A kapton foil is wrapped around the detector to form a structure which allows a controlled helium flow for cooling purposes. The helium is provided through normal plastic tubes since the upstream and downstream recurl stations are not present and do not introduce the space constraints that would normally exist there in the final detector design. Similarly, power and data cables do also not follow the final design and use more space. A picture of the setup is shown in figure 92.

The pixel sensors are connected to the frontend boards via an SPI line for configuration and three LVDS data lines for each MuPix. The Mu3e slowcontrol interface could not be used since the detector was built with version 10 of the MuPix sensor which included a design flaw breaking the configuration possibility via the Mu3e protocol. The cables used for these connections will be different in the actual detector. As visible in figure 92, the blue cables which connect one side of the detector prototype with the FEBs in the service support wheel would require too much space when one considers that those only read out about 4 % of the final tracker in this image.

The module mounted on top of the vertex detector is a scintillating fibre ribbon. It was initially intended to be operated together with the vertex tracker. This was not possible due to a hardware issue with the SciFi readout. The entire circular cage shown in figure 92 was then inserted into the Mu3e magnet and connected to the outside via feed-throughs in the magnet doors.

The included ladders went through a first step of testing before and after being mounted in the vertex tracker. The test results are available in [16] and showed that a maximum of 83 sensors would be able to fully operate. The other chips showed different types of defects, such as shorts or visible cracks in the sensor. The number of working chips was reduced to 52 during data taking for various reasons, including 8b10b errors and the loss of an entire FEB section.

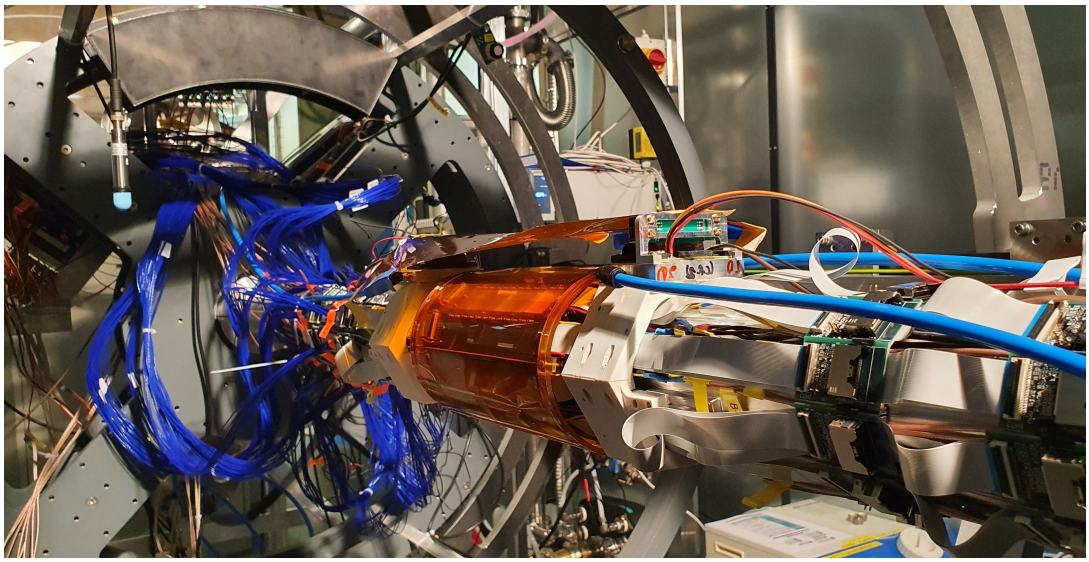


Figure 92.: Image of the Detector used for the test runs.

6.1.2. Results

The analysis of the integration run data has already been published in [86], [71] and [16]. In summary, it was possible to find spatial correlations between MuPix sensors and to reconstruct the target position. However, the results relevant for this thesis are the consequences to be drawn for the DAQ design. These will be discussed here. Some of the decisions in the previous chapters result from the things learned from the integration run. These cases will give references to the according parts of this thesis.

As already mentioned above, only a fraction of the MuPix sensors were fully operational. Some of these sensors were already broken before mounting the detector. Ensuring a functional detector requires chip tests on every step of the construction process: On the level of individual chips, ladders, modules, stations and finally for the complete pixel detector. Multiple versions of the simplified and portable DAQ discussed in sections 5.2 – 5.4 have been adapted and distributed to the different institutes involved in pixel detector construction to be able to perform these quality control tests. Test procedures have been developed in [87].

Additional chips were unusable after the mounting of the detector due to 8b10b errors and the loss of an entire FEB. The issue of 8b10b errors requires thorough quality control tests beforehand, as mentioned above. It is also necessary to provide monitoring possibilities in the DAQ in order to distinguish cable- and transmission-induced 8b10b errors from sensor issues. This is the reason to monitor disparity, realignment cycles and invalid codes separately in section 4.7. 8b10b errors can also be influenced by the tuning of driver settings on the MuPix, which will require the development of automated tuning procedures in the future. These will benefit from the configuration speed improvements made in section 4.6.

The problem of losing an entire FEB is solved by the redundant firmware upload from section 4.11.3 and the secondary emergency image (section 5.1) on the flash memory. The redundant firmware upload was already planned before the run but was not implemented at that time. Therefore, physical access with a USB cable was needed to revive this particular FEB. The reason why it initially lost its ability to boot after a power cycle is not clear. Should such cases happen regularly, they will need further investigation.

The access to the lost FEB led to another incident. The detector was completely powered off for a night and the detector atmosphere was opened the following day. The detector volume is located inside of the superconducting Mu3e magnet. Small insulation leaks to the cooling of the magnet can cause the detector volume to cool down even when all other systems – including the detector cooling – are turned off. Opening the detector volume in such a case causes the helium to exchange with air and allows humidity to enter the system. The temperature on the inside was below the dew point, which caused condensation to form on the detectors. It seems like no significant damage occurred, but it could have easily led to the loss of detector components if they were powered during this incident.

Another unplanned temperature extreme occurred when the FEBs were accidentally left powered on without the water cooling. The measured temperature increased from 20°C to 64°C on a sensor in the service support wheel. The final temperature on the FEBs is unknown but likely much higher. These events show that some critical values of the system have to be constantly monitored and alarms have to be set for too high and too low values. When communicating these alarms to the users, it is important to rank their criticality: A lost data packet in the DAQ is a very different level of problem than condensation water in the detector. Alarm structure and messaging to the users will have to evolve further once the experiment operates. A push-notification option was added to MIDAS to make very critical alarms directly visible to the user. Additionally, the automated FEB shutdown in section 4.11.2 was introduced after the incident. As discussed there, the shutdown will decrease heat production. This would not have prevented the accident, but it is everything that the FEB can do on its own for self-protection in such a case.

Another issue regarding FEB powering occurred whenever the detector was turned on. It turns out that the FEBs boot sequence from section 4.11.1 causes some power spikes when the NIOS boots and the SI-chips are configured. During the integration run, a hardware interlock was connected to the power drawn by the FEBs and would turn them off whenever it exceeded a limit. Measurements outside of the magnet showed that it was possible to reduce these power spikes by changing the boot sequence. However, doing so in the magnet was complicated since the boot procedure could only be changed by a firmware upload. Firmware uploads require the FEB to be powered. This effectively created a situation where a safety mechanism works fine when the detector is operating, but it also prevents the detector from being turned on. Therefore, safety mechanisms and also the topic of alarms from above depend on the state of the detector and on the things that users are currently doing with it.

Constructing and testing the different detector parts before the run took a significant

amount of time and people from each subsystem. Also, the limited availability of DAQ resources sometimes caused a bottleneck for progress in the setup process. Separated operation of the three subdetectors and operation outside of the beam area would have been required to completely avoid any collisions between the tuning and testing of detectors and ongoing infrastructure activities at the magnet. A method for separated detector operation was proposed in section 5.12 and a staging area close to the Mu3e server room was reserved for following runs to be able to operate the detectors outside of the beam area. The staging area is supposed to prevent the continuous assembly and disassembly of standalone DAQ systems whenever some test of a detector component is required. Centralising the infrastructure at the staging area allows a connection of all parts to the actual DAQ resources in the Mu3e server room instead of multiple smaller systems. These smaller systems have been discussed in 5.2 and are useful for tests at the involved institutes. However, for tests at the actual magnet cage at PSI, multiple of these systems turned out to produce too much clutter in terms of organisation and cables.

The data analysis after the run was difficult since the DAQ was operating with a few flaws. The most notable one was introduced by a disabled gray decoder in the data unpacker block of section 4.7 in the MuPix FEB firmware. The MuPix timestamps are gray encoded and need to be decoded before they are send into the hitsorter. The gray decoder in the MuPix firmware which was supposed to do that was not turned on for the integration run. This was a mistake and the option to turn it on or off was removed afterwards.

As a consequence of this mistake, the data was not sorted in time in the hitsorter but received an additional mixing by being sorted along the gray timestamp. Some data was lost in this process due to the limited size of the time acceptance window discussed in section 4.7.1. Some additional bugs were later found in the sorter which produced doubled subheaders and similar issues.

These issues propagated into the SWB firmware, where the time alignment tree did not function properly and had to be replaced with a sequential readout of FEB frames. As a result, the data which did end up on disk was not really the intended series of time slices of the entire detector data but could more be described as random parts of it with some kind of sorting along an incorrect timestamp. The issue with that is that the amount of data which can be saved is limited by the speed of a hard drive. For properly working time slices this would not have been a problem since all detector data would have been available for this time period.

Fortunately, the lower bits in a gray counter are still the most likely ones to flip when the counter is increased by one. This can be seen in figure 38 of section 3.8.3. Therefore, hits with timestamps which are close to each other have an increased probability to be close to each other when their gray-timestamp is interpreted as a normal binary counter by the sorter. Untangling this situation and additional DAQ bugs has been a mayor task after the run. However, it was possible to do so, which led to the correlation and tracking results published in [86], [71] and [16].

Further consequences for the DAQ include the need for online data quality monitoring. During the integration run the online monitoring functions, for example for

correlation histograms or pixel hitmaps, had speed issues and were not able to process larger amounts of data. Some live event monitoring is going to be needed to identify problems quickly. Which exact parameters will be required there and how they can be presented efficiently to the users will have to evolve over time as the system is used.

In summary, the integration run has provided a lot of input for the DAQ development and also for other aspects such as detector construction and cooling. The following section will discuss the cosmic run 2022, which reused large parts of the integration run system.

6.2. Cosmic Run 2022

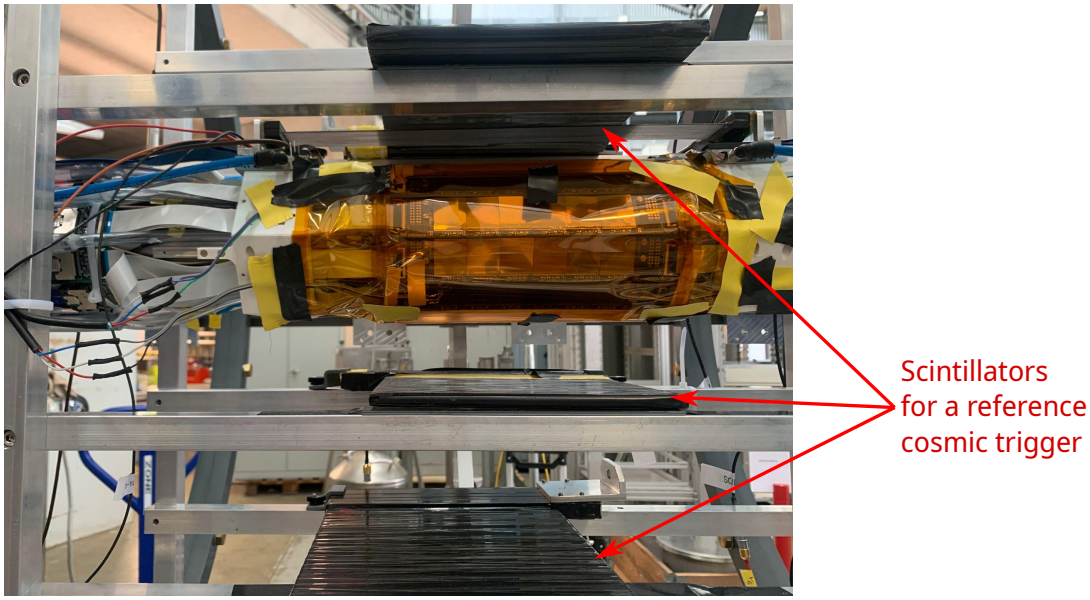


Figure 93.: Image of the scintillators for the reference trigger.

The cosmic run in 2022 used the same vertex tracker from the previous integration run. The ladders on it were rearranged to achieve a larger overlap between the working sensors of the two layers. Additionally, scintillator paddles were added above and below the vertex tracker to provide a reference trigger for cosmic muons (figure 93). The coincidence of the three scintillators was created in an external crate and then fed into a frontend board using the method discussed in section 5.6. The scintillating fibre ribbon, which was not functional in the previous integration run, was replaced.

The setup was operated without a beam and without a target outside of the magnet at the staging area. Relying on cosmic muons affects how the detector has to be tuned to allow reasonable tests. In particular, the acceptable noise limits in the pixel sensors are much lower compared to the integration run. In the integration run, the noise levels were insignificant compared to the rate of muon decays. For the cosmic run, the pixel detector had to be brought into an almost noise-free state.

Noise-free in this context also includes that no 8b10b errors on the LVDS links between FEBs and MuPix sensors are acceptable. Even small error rates there can produce fake hits on the FEBs, which can saturate the DAQ since the GPU selection is absent. The tuning process of the pixel sensors required multiple days of work since it was not fully automated. In addition to the sensors that were already known to be broken from the integration run, multiple functioning sensors had to be removed from the DAQ because no settings could be found to reduce the LVDS 8b10b error rates to zero. In the end, only 27 out of the 108 Mupix sensors were able to contribute to the measurements in the cosmic run. For the final experiment, the quality control process is supposed to filter the problematic sensors out before they are mounted on the detector.

Another few days of the tuning process were spent uploading TDAC values. At the point where the LVDS links of all remaining sensors operated properly, noise data was taken and used to turn off individual noisy pixels on each sensor. This process had to be repeated multiple times. The reason for this is that the noise was high enough to saturate the DAQ. Once the noisy pixels were turned off, other noisy pixels appeared which were not previously visible due to DAQ saturation. These then again saturated the DAQ, and this process was repeated multiple times before an almost noise-free operation was reached.

A single upload of TDAC values to turn off pixels required 28 minutes, which made this a time-consuming process. It is important to note that the process here was not a tuning process adjusting individual pixel thresholds but was just turning off pixels. An actual tuning process will have more free parameters and will require significantly more upload cycles. Afterwards, more effort was put into optimising the tune value upload since it was clear that this would not be viable for almost 3000 sensors. This then leads to the system discussed in section 4.6.3, which is expected to configure the entire Mu3e pixel detector in about four seconds. However, more work is still required to develop and automate a tuning procedure. How to deal with DAQ saturation during this tuning procedure is a question that might also need a solution in the future. For a small number of sensors, the firmware QC histogram introduced in section 5.8 can be used, but that is not going to be applicable for the entire detector due to resource limitations on the SWB. The rate of each pixel may need to be extrapolated from the fraction of time slices which were recorded in the DAQ.

The zero suppression from section 5.5 was introduced during the cosmic run to reduce protocol overheads. It is impossible to read out the complete data otherwise since every FEB would produce 31.25 MB/s of empty frames.

The cosmic run obviously acquired less statistics compared to the integration run from the previous section. However, the mistakes made there were not repeated for the cosmic run, which resulted in much cleaner data and accurate timestamp information. This is relevant for the DAQ since it can be used to validate the synchronisation between the components.

6.2.1. Pixel Detector and Reference Trigger Coincidences

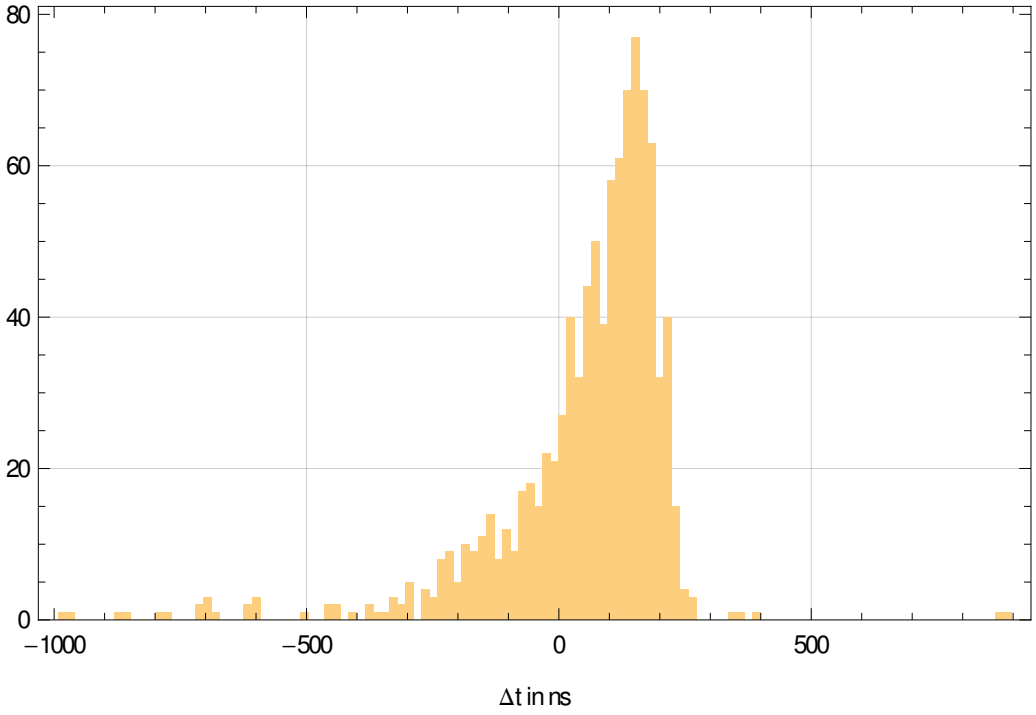


Figure 94.: Sum of all coincidences between MuPix timestamps and the reference cosmic trigger.

The largest amount of cosmic data for the pixel detector was accumulated during a 15 hour long, unattended overnight run, which was remote-controlled from Mainz. Remote control of the Mu3e experiment from the participating institutes is also the intention for the final detector operation. Remote control options and DAQ system stability are important aspects since they determine how many people the operation of the Mu3e detector is going to require. The cosmic run in 2022 was the first time that this was actually exercised. In summary, remote controlling via MIDAS worked without issues and the DAQ was able to run stable for 15 hours without any user interventions.

The accumulated data was then compared against the reference triggers received from the scintillator coincidence. The result is shown in figure 94. It shows that the DAQ system ran synchronised over 15 hours and that $\mathcal{O}(1000)$ hits of cosmic particles were recorded in coincidence with the scintillators.

It is important to mention that the histogram shown there is the sum of all coincidences in all MuPix sensors. The same data for the individual chips is shown in figure 95. As visible there, the shape of the time distribution is different for each sensor. Additionally, they do not provide the nominal time resolution of roughly 20 ns [15]. This is expected since the sensors were not properly tuned for timing

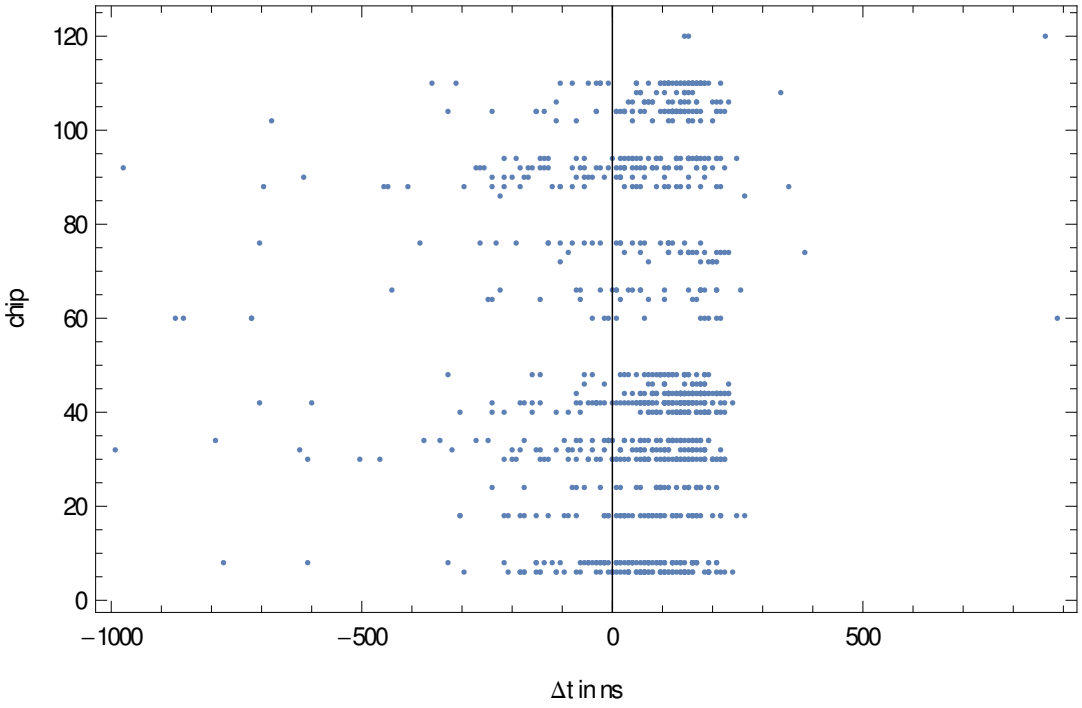


Figure 95.: Coincidences between MuPix timestamps and the reference cosmic trigger.

precision and since the applied reverse-bias voltage was reduced for some sensors. The tuning process described above was only concerned with link stability and noise. For timing precision, other parameters have to be adjusted and a timewalk correction needs to be made. This is also the reason for the tail towards negative values in figures 94 and 95 since the timestamp of late hits will need to be corrected using the time over threshold (ToT) measurement. The correction for this data can be found in [71].

These coincidences were then used to search for particle tracks using only the time information. Due to the many non-functional sensors, the probability of hitting three or more independent sensors is quite low. For the 15-hour measurement, only a handful of 4-hit events were found. Those were then provided to [88], where it is shown that they indeed also turn out as a straight line in terms of geometry.

An example of such a 4-hit track is shown in figure 96. In this graphic, only the 58 sensors that were operational during the integration run are shown. In the cosmic run, only the 27 sensors in figure 95 delivered data, which significantly reduces the possible parameter space for 4-hit tracks in coincidence with the external scintillators.

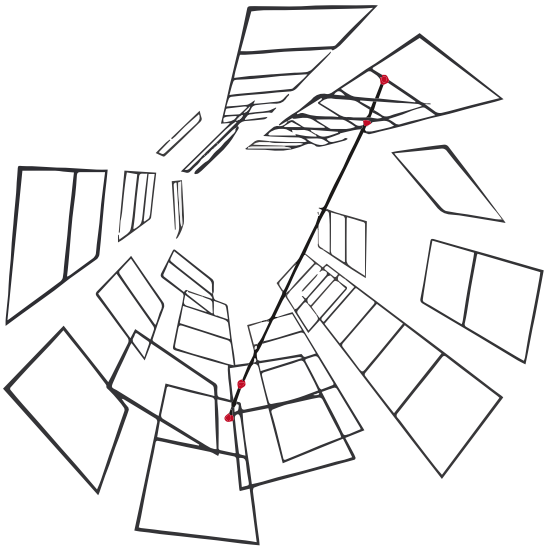


Figure 96.: Example for a 4-hit cosmic track. Image adapted from [88].

6.2.2. SciFi Detector and Reference Trigger Coincidences

Similarly to the MuPix coincidence, the timestamp received from the SciFi detector was compared against the reference trigger timestamp. The result is shown in figure 97. The width of this distribution can be attributed to the precision of the reference trigger. The logic used there is unable to match the time resolution of the SciFi detector. Therefore, the distribution width in figure 97 is determined by the reference signal instead of the scintillating fibre ribbon.

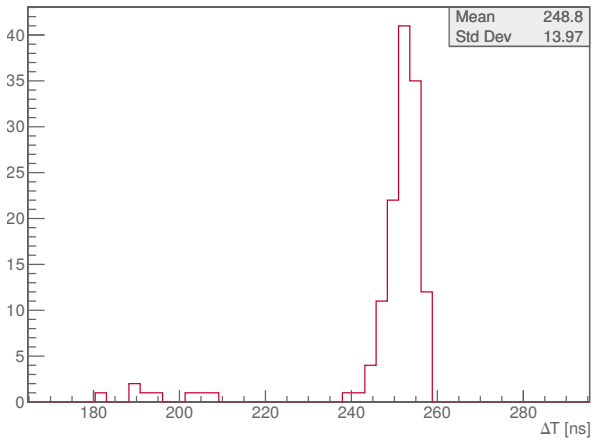


Figure 97.: Coincidences between SciFi timestamps and the reference cosmic trigger.

7

Conclusion and Outlook

The Mu3e project is aiming to improve the upper experimental limit for the branching ratio of the charged lepton flavour violating decay $\mu^+ \to e^+e^-e^+$. According to the standard model, this decay is not observable and is suppressed to a level of below 10^{-54}. Experimental observations of this decay would provide a clear signature of BSM physics. Without the detection of such an event, the improvement in the branching ratio measurement will further constrain BSM models.

Mu3e phase I is pursuing a challenging branching ratio sensitivity goal of $2 \cdot 10^{-15}$. A planned second phase of the experiment is supposed to further increase this sensitivity to 10^{-16}. This would improve the current limit measured by the SINDRUM collaboration in 1988 by four orders of magnitude.

Reaching these sensitivity goals requires a detector system able to analyse large amounts of muon decays. The collaboration has developed this system in recent years and this thesis has contributed to the design, implementation and commissioning of the data acquisition.

The phase I Mu3e detector will observe 10^8 muon decays per second with six layers of MuPix HV-MAP tracking sensors. A scintillating fibre and a scintillating tile detector improve the time resolution of the tracks. These detectors will produce a data rate of about 100 Gbit/s in phase I and 1 Tbit/s in phase II.

A multi-layer DAQ system based on field programmable gate arrays has been developed to process this data. The work in this thesis has focussed on the lower layers of this system and has additionally provided infrastructure for communication with detector components.

Setup- and hold-time violations of the used firmware components have been investigated using a seed variation approach to gain reliable information about the timing effects of firmware adjustments. This technique has been utilised to find a solution for read and write access to control ports and registers in the lower DAQ layers. The implemented solution makes use of a tree-shaped structure and allows the use of the full available bandwidth for communication between layers one and two of the DAQ.

Firmware structures have been implemented to integrate the different types of readout paths into the DAQ framework and to allow a shared connection for the control and detector data. The thesis has also contributed to the pixel-specific readout path and has led to the development of a configuration system which is expected to be able to configure the entire Mu3e pixel detector within four seconds.

The thesis has made contributions to the development of a clock and reset system to provide a synchronisation method to the detector. A method was developed to measure phase shifts between a recovered and reference clock domain, which can be

used to ensure detector synchronisation.

The DAQ system was tested and validated in two test runs, which also provided further input in form of design changes and additional functionalities for subsequent Mu3e operations. It was shown that the implemented synchronisation system works and is stable for longer time periods. The stability of the DAQ system has been sufficient, and remote operation of the detector was possible without user intervention.

Further developments and DAQ adjustments will have to follow in the future and will be based on the experiences gained during the test runs. The fast configuration upload introduced in this thesis enables the development of automated tuning procedures, which will be necessary for the efficient operation of the Mu3e detector.

Before operating the final detector, various studies will have to be conducted. The developed phase shift measurement method has to be used to adjust clock shifts in the DAQ for a proper reset synchronisation. Another study will have to compare the timestamps received by the detectors with their arrival timestamps on the first FPGA in order to configure shifts of acceptance windows in the firmware datapath correctly.

Additionally, an alignment has to be performed. Parts of the alignment will likely be based on tracks from cosmic muons. The DAQ must find these tracks online. The concept which was investigated in this thesis will not be able to do so due to limited hardware resources. Therefore, further developments are required to integrate a cosmic trigger in the DAQ.

Further work is also required for the camera alignment system's readout firmware. The major missing component there is communication between the camera sensor and the FPGA. Once the image data has reached the FPGA, the existing infrastructure developed in this thesis can be used to provide it to the alignment software.

A number of other projects will use detectors developed for the Mu3e experiment. Parts of the DAQ shown here will be adjusted for these projects. Some of these adjustments have been started during this work. Especially the increase of the data rate in Mu3e phase II is going to require architectural changes in the data acquisition. However, the phase II DAQ will likely be based upon the developments presented in this thesis.

Appendices

A

Further Details on FPGAs and ASICs

A.1. Lookup Table

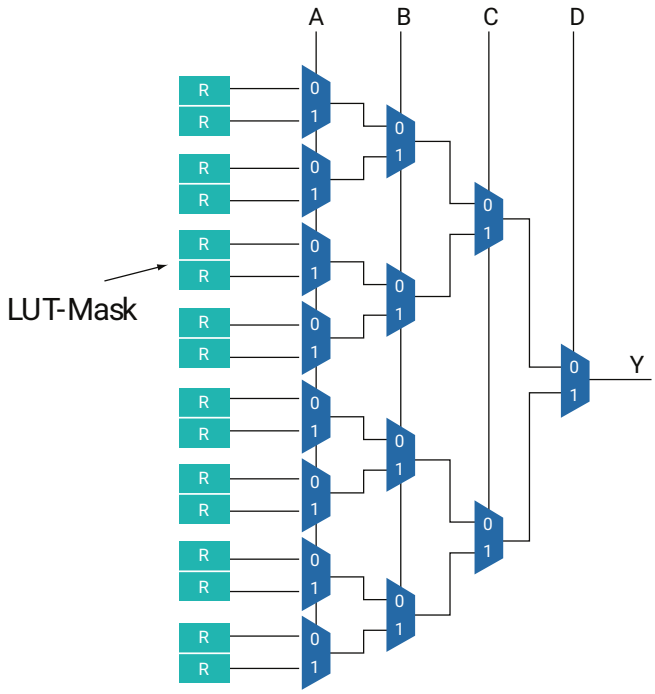
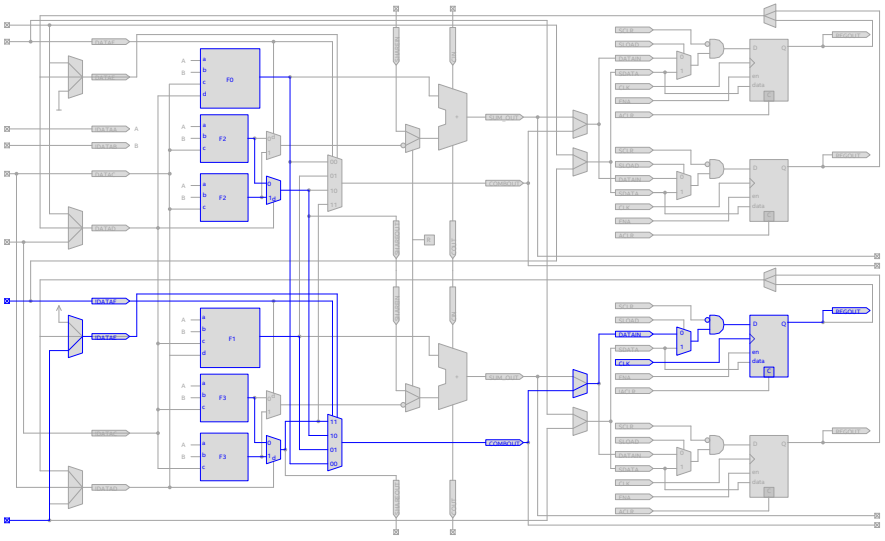


Figure 98.: Block diagram of a lookup table. The LUT-Mask is stored in static random access memory (SRAM) cells and configures the logic for the calculation of Y from the inputs A, B, C and D. A LUT with 2^n entries in the LUT-Mask can implement any function of n inputs. The lookup table (LUT) shown here can therefore implement any boolean function of A, B, C and D [32].

A.2. Detailed Schematic of an ALM cell



A.3. Alternative Implementation of Tree Structure

An alternative implementation of the tree structure discussed in section 3.7.6 and 4.4.4 is a version where each node of the tree only has one output register connecting to the four following nodes instead of four copies of that register. The conceptual difference is shown in figure 99. This situation can be created accidentally if the compilation tool identifies the four output registers as identical and automatically merges them. This shows the importance of implementation directives since the automatic decisions of the tools are not always beneficial for timing.

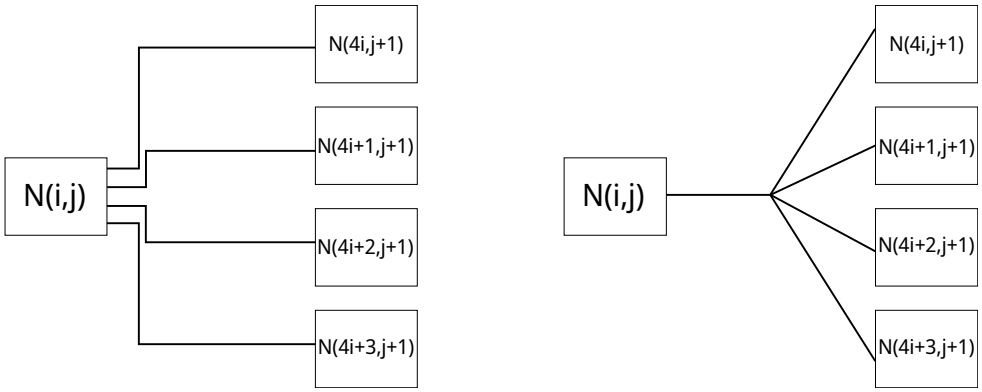


Figure 99.: Comparison of tree structures. The concept on the left was already shown in section 3.7.6.

The concept on the right increases the fanout compared to the idea on the left but saves register resources. Compared to the reference without the tree structure, the mean minimal setup slack improves by 0.18 ns and the timing closure probability from 1 % to 26 %. The concept with four individual registers (as discussed in section 3.7.6) showed a slack improvement of 0.24 ns and 48 % timing closure.

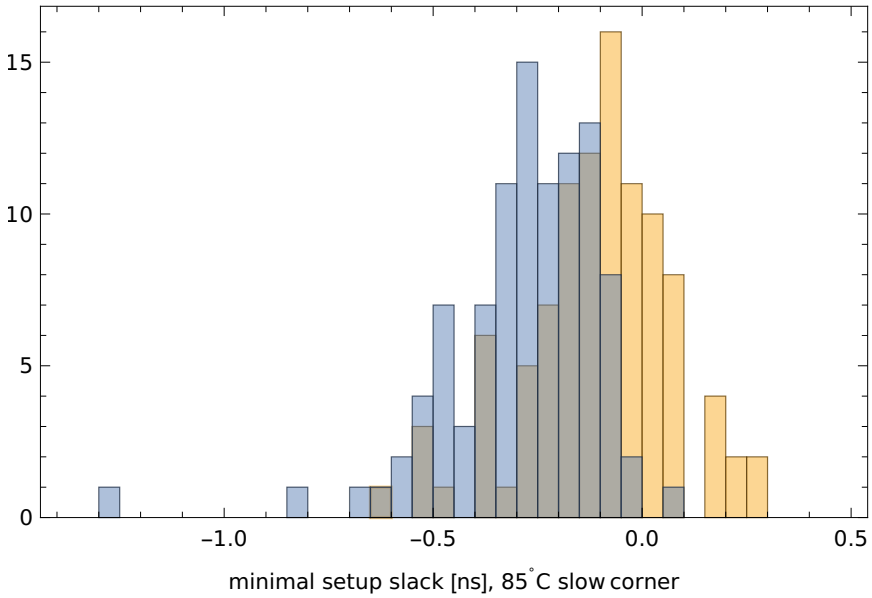


Figure 100.: Timing improvement of alternative tree structure. Blue is the original reference, yellow the alternative tree.

B

Mu3e Firmware Implementation Details

B.1. Distribution of slowcontrol endpoints on the FEB

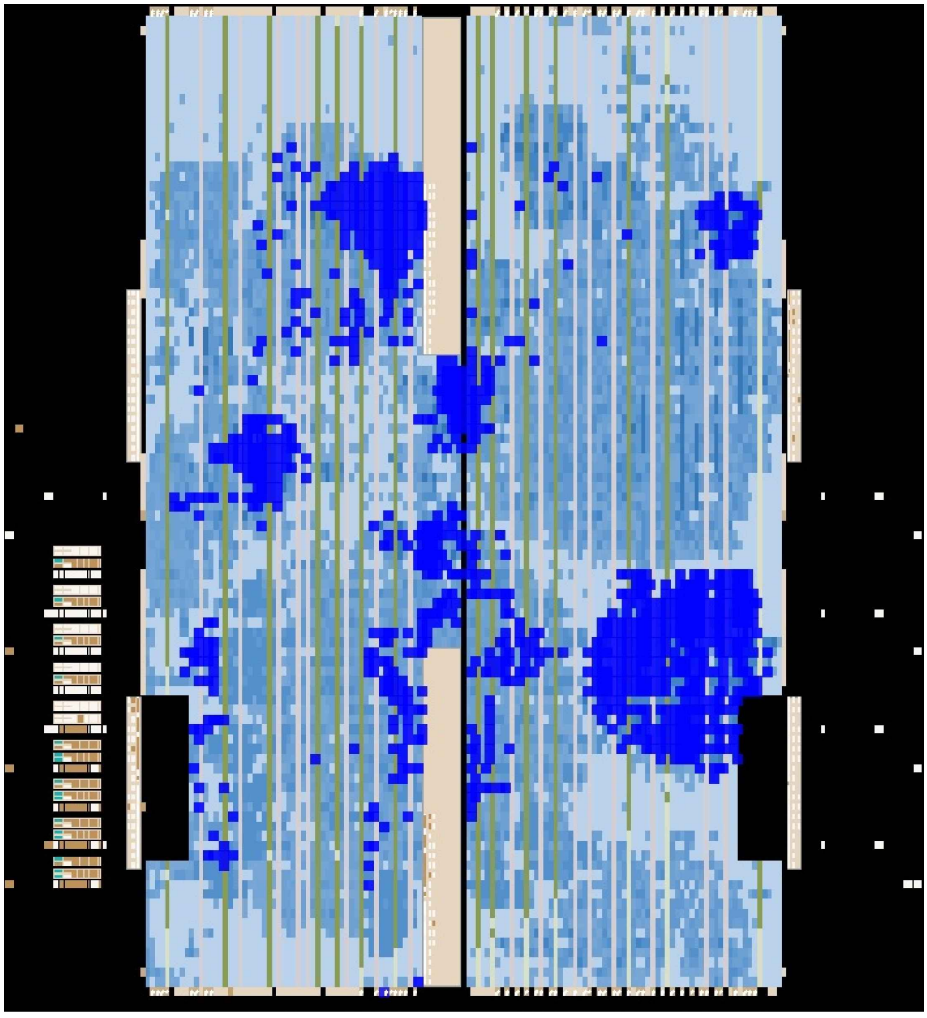


Figure 101.: Distribution of slowcontrol endpoints on the FEB.

B.2. Mupix Slowcontrol Protocol

Command	Bit-identifier	Effect
WriteDacRegister	111000	Write Bias DAC register
LoadDacRegister	000111	Load Bias DAC register
WriteConfRegister	110100	Write Config register
LoadConfRegister	001011	Load Config register
WriteVDACRegister	110010	Write VDAC register
LoadVDACRegister	010011	Load VDAC register
WriteColRegister	110001	Write PixelRam Write register
LoadColRegister	100011	Load PixelRam Write register
WriteTestRegister	101100	Write Test register
LoadTestRegister	001101	Load Test register
WriteTDACRegister	101010	Write TDAC register
LoadTDACRegister	010101	Load TDAC register
ReadbackDACs	100110	Readback for all registers
ReadbackTDACs	100101	Readback for the pixel bits
SteerADC	100000	Steer the ADC
Inject	001000	Trigger an injection
ResetBiasBlocks	000010	Reset the registers
ShiftColRegisterbyOne	111101	SHIFT-BY-ONE
SyncReset[64bit]		Synchronous Reset

Table 15.: Mupix Slowcontrol Protocol. Copied from [20].

List of Figures

List of Tables

List of Abbreviations

ADC analog-to-digital converter

ALM adaptive logic module

ASIC application specific integrated circuit

BAR base address register

BERT bit error rate test

BSM beyond standard model

CDR clock data recovery

CPU central processing unit

CMOS complementary metal-oxide semiconductor

DAC digital-to-analog converter

DAQ data aquisition

DMA direct memory access

DPA dynamic phase alignment

FIFO first in/first out buffer

FPGA field programmable gate array

GPU graphics processing unit

GUI graphical user interface

HDL hardware description language

HIPA high intensity proton accelerator

HSMC high speed mezzanine card

HV-MAPS high voltage monolithic active pixel sensors

I2C inter-integrated circuit

I/O input/output

IP interlectual property

JTAG joint test action group

LSB least significant bit

LUT lookup table

LVDS low voltage differential signaling

MAP monolithic active pixel

MSB most significant bit

MTBF mean time between failures

PCB printed circuit board

PCI express peripheral component interconnect express

PCS physical coding sublayer

PLL phase locked loop

PMA physical medium attachment

PSI Paul Scherrer Institute

QSFP quad small formfactor pluggable

RAM random access memory

ROM read only memory

SFP small formfactor pluggable

SM Standard Model

SPI serial periphal interface

SRAM static random access memory

TCL tool command language

UART universal asynchronous receiver-transmitter

VCO voltage controlled oscillator

Bibliography

[1] D. Galbraith and C. Burgard.
Standard Model of Physics. CERN Webfest, 2012.
http://davidgalbraith.org/portfolio/ux-standard-model-of-the-standard-model.

[2] A.-K. Perrevoort.
Sensitivity Studies on New Physics in the Mu3e Experiment and Development of Firmware for the Front-End of the Mu3e Pixel Detector.
PhD. Thesis, Heidelberg University, 2018.
https://www.psi.ch/sites/default/files/import/mu3e/ThesesEN/DissertationPerrevoort.pdf.

[3] G. Aad et al.
Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC.
Phys. Lett., B716:1–29, 2012.

[4] S. Chatrchyan et al.
Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC.
Phys. Lett., B716:30–61, 2012.

[5] G. Venanzoni et al.
New results from the muon g-2 experiment, 2023.
arXiv:2311.08282 [hep-ex].

[6] S. Navas et al.
Review of particle physics.
Phys. Rev. D, 110(3):030001, 2024.

[7] K. Afanaciev et al.
A search for $\mu^+ \to e^+\gamma$ with the first dataset of the meg ii experiment.
Eur. Phys. J. C, 84(3):216, 2024.

[8] U. Bellgardt et al.
Search for the Decay mu+ → e+ e+ e-.
Nucl. Phys., B299:1–6, 1988.

[9] W. Bertl et al.
A search for μ-e conversion in muonic gold.
The European Physical Journal C - Particles and Fields, 47(2):337–346, Aug 2006.

[10] Y. Fukuda et al.
Evidence for oscillation of atmospheric neutrinos.
Phys. Rev. Lett., 81:1562–1567, 1998

.

[11] A. Blondel et al.
Research Proposal for an Experiment to Search for the Decay $\mu \to eee$.
ArXiv e-prints, January 2013.

[12] M. Aiba et al.
Science Case for the new High-Intensity Muon Beams HIMB at PSI.
arXiv:2111.05788 [hep-ex], 2021.

[13] G. Signorelli L. Calibbi.
Charged Lepton Flavour Violation: An Experimental and Theoretical Introduction, 2017.
arXiv:1709.00294 [hep-ph].

[14] A. Blondel et al.
Research Proposal for an Experiment to Search for the Decay $\mu \to eee$.
2013.

[15] K. Arndt et al.
Technical design of the phase I Mu3e experiment.
Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 1014:165679, 2021.

[16] T. Rudzki.
The Mu3e vertex detector - construction, cooling, and first prototype operation.
PhD. Thesis, Heidelberg University, 2022.
https://www.psi.ch/de/mu3e/theses.

[17] I. Perić.
A novel monolithic pixelated particle detector implemented in high-voltage CMOS technology.
Nucl. Instrum. Meth., A582:876–885, 2007.

[18] T. Rudzki, H. Augustin, D. M. Immig, R. Kolb, and L. Mandok.
An ultra-light helium cooled pixel detector for the mu3e experiment, 2023.

[19] A. Weber.
Development of Integrated Circuits and Smart Sensors for Particle Detection in Physics Experiments and Particle Therapy.
PhD. Thesis, Heidelberg University, 2021.
http://www.ub.uni-heidelberg.de/archiv/30650.

[20] H. Augustin.
Development of a Novel Slow Control Interface and Suppression of Signal Line Crosstalk Enabling HV-MAPS as Sensor Technology for Mu3e.
PhD. Thesis, Heidelberg University, 2021.

[21] C. M. Perez and L. Vigani.
Searching for the Muon Decay to Three Electrons with the Mu3e Experiment.
Universe, 7(11):420, 2021
.

[22] H. Augustin et al.
The mu3e data acquisition.
IEEE Transactions on Nuclear Science, 68(8):1833–1840, August 2021.

[23] D. vom Bruch.
Pixel Sensor Evaluation and Online Event Selection for the Mu3e Experiment.
PhD. Thesis, Heidelberg University, 2017.
https://www.psi.ch/sites/default/files/import/mu3e/ThesesEN/DissertationVomBruch.pdf.

[24] P. Horowitz and W. Hill.
The art of electronics; 3rd ed.
Cambridge University Press, Cambridge, 2015.

[25] E. F. Moore.
Gedanken-Experiments on Sequential Machines, pages 129–154.
Princeton University Press, Princeton, 1956.

[26] R. E. Bryant and J. H. Kukula.
Formal Methods for Functional Verification, pages 3–15.
Springer US, Boston, MA, 2003.

[27] R. J. Punnoose, R. C. Armstrong, M. H. Wong, and M. Jackson.
Survey of existing tools for formal verification.
12 2014.

[28] J. Stephenson et al.
Understanding metastability in fpgas.
Altera Corp., San Jose, CA, USA, 2009.
White Paper WP-01082-1.2.

[29] P. Alfke.
Efficient shift registers, lfsr counters, and long pseudo-random sequence generators.
Xilinx Application Note, 1996.

[30] Texas Instruments.
Fractional/Integer-N PLL Basics.
Technical Brief SWRA029, 1999.
http://www.ti.com/lit/an/swra029/swra029.pdf.

[31] Arria V Device Overview.
Intel Corp., 2020.
https://www.intel.com/content/www/us/en/docs/programmable/683440.

[32] Fpga architecture.
Altera Corp., San Jose, CA, USA, 2006.
White Paper WP-01003-1.0.

[33] Arria V Device Handbook Volume 1: Device Interfaces and Integration.
Altera Corp., San Jose, CA, USA, 2019.
https://www.intel.com/content/www/us/en/docs/programmable/683213.

[34] Intel Quartus User Guide, Design Optimization.
Intel Corp., 2021.
https://www.intel.com/content/www/us/en/docs/programmable/683641.

[35] Max 10 FPGA Device Overview.
Intel Corp., 2022.
https://www.intel.com/content/www/us/en/docs/programmable/683658.

[36] R. Kumar and V. Kursun.
Reversed temperature-dependent propagation delay characteristics in nanometer cmos circuits.
IEEE Transactions on Circuits and Systems II: Express Briefs, 53(10):1078–1082, 2006.

[37] Intel Quartus User Guide, Design Recommendations.
Intel Corp., 2021.
https://www.intel.com/content/www/us/en/docs/programmable/683082.

[38] H.W. Johnson and M. Graham.
High-speed Digital Design: A Handbook of Black Magic.
Prentice Hall Modern Semiconductor Design. Prentice Hall, 1993.

[39] H.W. Johnson and M. Graham.
High-speed Signal Propagation: Advanced Black Magic.
Prentice Hall PTR Signal Integrity Library. Prentice Hall PTR, 2003.

[40] A. X. Widmer P. A. Franaszek.
Byte oriented DC balanced (0,4) 8B/10B partitioned block transmission code.
US patent US4486739A, 1982.

[41] W. W. Peterson; E. J. Weldon.
Error-correcting codes.
MIT Pr., Cambridge, Mass. [u.a.], 2. ed. edition, 1972.

[42] Avalon Interface Specifications.
Intel Corp., 2022.
https://www.intel.com/content/www/us/en/docs/programmable/683091.

[43] AMBA AXI Protocol.
ARM Limited, 2004.
https://developer.arm.com/documentation/ihi0022/latest/.

[44] Arria V Device Handbook Volume 2: Transceivers.
Altera Corp., San Jose, CA, USA, 2019.
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-v/av_5v3.pdf.

[45] Texas Instruments.
To Measure PCI-e Reference Clock With Multiplexers.
https://www.ti.com/lit/rr/slat161/slat161.pdf
.

[46] Xillybus Ltd.
Down to the TLP: How PCI express devices talk (Part I).
http://xillybus.com/tutorials/pci-express-tlp-pcie-primer-tutorial-guide-1.

[47] M. Köppel.
Data Flow in the Mu3e Filter Farm.
Master Thesis, JGU Mainz, 2019.
https://www.psi.ch/en/mu3e/theses.

[48] Pci configuration space — Wikipedia, the free encyclopedia, 2024.

[49] FS.com Inc.
advantages-and-disadvantages-of-multimode-fiber.
2024.
https://community.fs.com/article/advantages-and-disadvantages-of-multimode-fiber.html.

[50] D.J. Richardson, J. Fini, and L. Nelson.
Space division multiplexing in optical fibres.
Nature Photonics, 7:354–362, 05 2013.

[51] M. Müller.
A Control System for the Mu3e Data Acquisition.
Master Thesis, JGU Mainz, 2019.

[52] N. Berger.
https://www.flickr.com/photos/nberger.

[53] P. Durante, N. Neufeld, R. Schwemmer, G. Balbi, and U. Marconi.
100 gbps PCI-express readout for the LHCb upgrade.
Journal of Instrumentation, 10(04):C04018–C04018, apr 2015.

[54] Avago Technologies.
MiniPOD^{TM} AFBR-812VxyZ, AFBR-822VxyZ Product Brief.
AV02-4038EN, 2013.

[55] V. Henkys, B. Schmidt, and N. Berger.
Online event selection for mu3e using gpus.
In *2022 21st International Symposium on Parallel and Distributed Computing (ISPDC)*, volume 150, page 17–24. IEEE, July 2022.

[56] MIDAS Wiki.
TRIUMF.
https://midas.triumf.ca/MidasWiki, Accessed: 05.10.2019.

[57] Samtec.
FIREFLY OPTICAL TRANSCEIVERS.
https://www.samtec.com/optics/systems/firefly/

.

[58] Skyworks Solutions Inc.
Si5345/44/42 Rev D Data Sheet, 2021.
https://www.skyworksinc.com/en/Products/Timing/High-Performance-Jitter-Attenuators/Si5345A.

[59] T. Wagner.
Clock Transmission for the Data Acquisition System of the Mu3e Experiment.
Bachelor Thesis, JGU Mainz, 2019.
https://www.psi.ch/sites/default/files/2019-05/BachelorWagner.pdf.

[60] Run Start and Reset Protocol.
Mu3e Internal Note 0046.

[61] LVDS SERDES Transmitter / Receiver IP Cores User Guide.
Altera Corp., San Jose, CA, USA, 2017.
https://www.intel.com/content/www/us/en/docs/programmable/683062.

[62] M. Lipiński, T. Włostowski, J. Serrano, and P. Alvarez.
White rabbit: a ptp application for robust sub-nanosecond synchronization.
In *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 25–30, 2011.

[63] D. Calvet.
Clock-centric serial links for the synchronization of distributed readout systems.
IEEE Transactions on Nuclear Science, 67(8):1912–1919, 2020.

[64] Y. Okazaki.
Status of muon g-2/edm experiment at j-parc.
NuFact 2023.

[65] P. Moreira et al.
The GBT Project.
In *Topical Workshop on Electronics for Particle Physics*. CERN, 2009.

[66] R. Giordano and A. Aloisio.
Fixed-latency, multi-gigabit serial links with xilinx fpgas.
IEEE Transactions on Nuclear Science, 58(1):194–201, 2011.

[67] S. Ritt.
Midas Slow Control Bus (MSCB).
PSI, 2001.
https://elog.psi.ch/mscb.

[68] M. Heppener and Y. Witzky.
Analysis of Max 10 FPGA Board.
2019.
student lab project, JGU Mainz.

[69] N. Berger.
Pixel Hit Time Sorting.
Internal Mu3e Meeting, Wengen, 2020.

[70] M. Köppel.
Private communication.

[71] M. Köppel.
Data Flow in the Mu3e Data Acquisition System.
PhD. Thesis, JGU Mainz, 2024.
in preparation, will be available here: https://www.psi.ch/en/mu3e/theses.

[72] H. A. Murugan.
PhD. Thesis, JGU Mainz.
in preparation, will be available here: https://www.psi.ch/en/mu3e/theses.

[73] S. Knapen, K. Langhoff, T. Opferkuch, and D. Redigolo.
A Robust Search for Lepton Flavour Violating Axions at Mu3e.
arXiv:2311.17915 [hep-ph], 2023.

[74] A.-K. Perrevoort.
Prospects for Dark Photon Searches in Mu3e.
2021.

[75] T. Hume.
Multiple scattering of positrons for the muon Electric Dipole Moment (muEDM) Experiment.
SPS Annual Meeting, 2022.
https://indico.cern.ch/event/1119258/contributions/4868737/.

[76] H. Backe, W. Lauth, P. Drexler, P. Heil, P. Klag, B. Ledroit, and F. Stieler.
Design study for a 500 MeV positron beam at the Mainz Microtron MAMI.
Eur. Phys. J. D, 76(8):150, 2022.

[77] J. Petersen.
PhD. Thesis, JGU Mainz.
in preparation.

[78] D. Becker et al.
The P2 experiment: A future high-precision measurement of the weak mixing angle at low momentum transfer.
The European Physical Journal A, 54(11), November 2018.

[79] N. Berger et al.
Measuring the weak mixing angle with the P2 experiment at MESA.
J. Univ. Sci. Tech. China, 46(6):481–487, 2016.

[80] U. Hartenstein.
Track Based Alignment for the Mu3e Pixel Detector.
PhD. Thesis, JGU Mainz, 2019.
https://www.psi.ch/sites/default/files/2019-05/DissertationHartenstein.pdf.

[81] G. Stanic.
A Camera Alignment System for the Mu3e Experiment.
Master Thesis, JGU Mainz.
https://www.psi.ch/en/mu3e/theses.

[82] S. Gagneur.
PhD. Thesis, JGU Mainz.
in preparation, will be available here: https://www.psi.ch/en/mu3e/theses.

[83] P. Freundlieb.
FPGA-based Camera Readout for the Mu3e Experiment.
Bachelor Thesis, JGU Mainz.
https://www.psi.ch/en/mu3e/theses.

[84] Arria10 Core Fabric and General Purpose I/Os Handbook.
Intel Corp., 2024.
https://www.intel.com/content/www/us/en/docs/programmable/683461.

[85] K. Neureither.
Towards an Online Reconstruction of Cosmic Muons for Mu3e using Hardware-Based Pattern Recognition.
Bachelor Thesis, Heidelberg University.
https://www.psi.ch/en/mu3e/theses.

[86] M. Köppel.
Mu3e integration run 2021, 2022.
arXiv:2203.07855 [physics.ins-det].

[87] L. Fuchs.
Development of a Quality Control Procedure for MuPix11 Pixel Sensors for the Mu3e Vertex Detector.
Bachelor Thesis, Heidelberg University, 2023.

[88] B. Gayther.
Preparations for Phase I of the Mu3e experiment.
PhD. Thesis, University College London.
in preparation, will be available here: https://www.psi.ch/en/mu3e/theses.

Acknowledgements

I would like to thank everyone who has supported me during this thesis. First of all, I would like to thank Prof. Niklaus Berger for the things i have learned in his group and for the opportunity to contribute to the Mu3e project. I would also like to thank Prof. Böser, who agreed to take over the role of the second referee.

Furthermore, I would like to thank all members of the AG Berger and the Mu3e collaboration for an amazing work atmosphere and many interesting discussions. I am thankful to everyone who proof-read this thesis.

Lastly, I would like to thank my family and friends who accompanied and supported me during my time in Mainz.

Curriculum Vitae

Martin Müller
Karl-Benz Straße 2a
79761 Waldshut-Tiengen

Date of birth:	30. July 1995
Place of birth:	Boppard
Nationality:	German

Ph.D.

since January 2020	Ph.D. student at the Johannes Gutenberg-University Mainz. Developement of an FPGA-based readout system for the Mu3e Experiment
January 2020 - December 2023	Employed as Research assistant at the JGU Mainz

Education

December 2019	Degree M.Sc. in Physics with a thesis on the subject „A Control System for the Mu3e Data Acquisition“
July 2017	Degree B.Sc. in Physics with a thesis on the subject „Effizienz eines HV-MAP Sensors auf niederenergetische Photonen“
2014 - 2019	Study of physics at the JGU Mainz
April 2014	German „Abitur“, Gymnasium auf der Karthause, Koblenz

Internships, Student-jobs, Voluntary Activities

October 2017- September 2018	Student assistant at the Institute for nuclear Physics, JGU Mainz
August 2017- September 2017	„Practical Course in Particle Physics“at PSI (3 weeks)

2017-2020	Volunteering as Webmaster of the jDPG Mainz

Languages

German	First language
English	Fluent, C1 level language certificate
Latin	7 years of school-education

List of publications

Public Presentation	„*Status of Mu3e Phase 1*“ at the „NuFact 2023“ in Seoul, Korea, August 2023
Public Presentation	„*Tests of the Mu3e DAQ in the Cosmic run 2022*“ at the „DPG spring meeting“ in Dresden, March 2023
Technical Design Report (co-author)	„*Technical Design of the Phase I Mu3e Experiment*“ in Nuclear Instruments and Methods Sect. A, August 2021
Article (co-author)	„*The Mu3e Data Acquisition*“ in „IEEE Transactions on Nuclear Science“, January 2021
Article (co-author)	„*Performance of the large scale HV-CMOS pixel sensor MuPix8*“ in „Journal of Instrumentation“, October 2019
Public Presentation	„*A Control System for the Mu3e Data Acquisition*“ at the „DPG spring meeting“ in Aachen, March 2019
Public Presentation	„*The Search for extremely rare processes with the Mu3e experiment*“ at the „4th interdisciplinary symposium“ in Mainz, April 2019

Mainz, 23. September 2024