



GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Fakultät für
Physik 

Master's Thesis

Entwicklung mit dem aktualisierten USBPix Test System für die ATLAS Pixel Front-end Elektronik

Development with the upgraded USBPix Test System for the ATLAS Pixel Front-end Electronics

prepared by

Eric Buschmann

from Krefeld

at the II. Physikalischen Institut

Thesis number: II.Physik-UniGö-MSc-2016/03

Thesis period: 19th October 2015 until 30th April 2016

First referee: Prof. Dr. Arnulf Quadt

Second referee: Priv.Do. Dr. Jörn Große-Knetter

Contents

1. Introduction	1
2. The Large Hadron Collider	3
2.1. Overview	3
2.2. The Standard Model of Particle Physics	3
2.3. The ATLAS Detector	5
2.3.1. Inner Detector	6
2.3.2. Calorimeters	7
2.3.3. Muon Spectrometer	7
2.4. Pixel Detector Upgrades	8
2.4.1. IBL Upgrade	8
2.4.2. HL-LHC Upgrade	8
3. Semiconductor Pixel Detectors	11
3.1. Interaction of Particles with Matter	11
3.2. Semiconductor Detectors	14
4. The New Front-End Readout Chip FE-I4	17
4.1. Motivation	17
4.2. Design of the FE-I4 Chip	18
4.2.1. Analog Pixel	20
4.3. Operation	21
4.3.1. Registers	21
4.3.2. Commands	21
4.3.3. Output Format	22
4.4. Data Acquisition	23
4.4.1. Triggering	24
4.4.2. Pulse Generator	24
5. The USBpix System	27
5.1. Hardware Overview	27

5.2. Readout Software	29
5.2.1. STcontrol	29
5.2.2. pyBAR	29
5.3. USBpix 3	30
5.3.1. Comparison of USBpix and USBpix 3	31
6. USBpix 3 and STcontrol	33
6.1. STcontrol	33
6.1.1. Software Structure	33
6.2. FPGA Firmware	34
6.2.1. Firmware Considerations	34
6.2.2. PyBAR FPGA Firmware	35
6.2.3. Software Interface	37
6.3. USBpix Protocol	39
6.4. USBpix 3 Integration	41
6.4.1. USBpix3 Library	41
6.4.2. USB3PixController	44
7. Verification and Results	49
7.1. Basic Tests	49
7.2. Analog and Digital Test	50
7.3. Tuning	51
7.3.1. Tuning Results	52
7.3.2. Crosscheck	54
7.4. Source Scans	55
7.4.1. Background Measurement	57
7.4.2. ^{241}Am Source	58
7.4.3. ^{90}Sr Source	63
7.5. Performance Measurements	70
8. Summary and Outlook	73
A. Appendix	75

1. Introduction

The *Standard Model of Particle Physics* describes all known elementary particles and three of the four fundamental forces. While it is a very successful and thoroughly tested description of particle physics and even further substantiated by the recent discovery of a Higgs Boson, it still leaves many questions unanswered. Other theories aim to fill these gaps, but ultimately, it requires experimental discoveries to select and test viable extensions or alternatives to the Standard Model. The success of the experiments in turn depends on powerful particle accelerators and the continuous improvement of particle detectors.

Interesting processes can be very rare and to make precise measurements, a large number of events has to be collected, which requires high luminosity and exposes the detector to large amounts of radiation. This especially prompts the development of radiation-hard sensors and readout chips that can withstand the high fluences close to the interaction point and can provide the bandwidth necessary for the high occupancy and trigger rate.

The *Large Hadron Collider* (LHC) continues to define the frontier of high energy particle physics by reaching unprecedented energies and luminosities, therefore allowing the experiments to explore new regions of physics. The LHC and the ATLAS detector are described in Chapter 2. The innermost part of the ATLAS detector, the *Pixel Detector* provides the high precision tracking required for momentum and vertex measurements close to the interaction point. It consists of semiconductor pixel detector modules (see Chapter 3) connected to radiation-hard readout chips.

During the 2013-2015 shutdown, the LHC was upgraded and ATLAS was equipped with an additional pixel layer, the *Insertable B-Layer* (IBL). To meet the higher requirements for the new pixel layer, a new readout chip, the FE-I4 was developed, which is described in Chapter 4.

USBpix is a USB-based test and readout system for ATLAS pixel, supporting both the FE-I4 and the previous generation, the FE-I3. Limitations in bandwidth and available hardware resources led to the development of the improved *USBpix 3* system for the FE-I4, which is also intended as a test system for new readout chips in the future *High Luminosity LHC* upgrade. Both systems are described in Chapter 5.

1. Introduction

To operate the new hardware, an integration into the existing software is required. To this end, two new components for the *STcontrol* test and readout software were developed, which add support for the USBpix 3 hardware (see Chapter 6).

Many measurements with the old and the new system were performed to verify the proper operation of the new software and to compare the performance of both systems. The measurements and results are presented in Chapter 7.

2. The Large Hadron Collider

2.1. Overview

The *Large Hadron Collider* (LHC) is the world's largest and most powerful particle accelerator, designed for proton-proton collisions at a centre of mass energy up to 14 TeV and a peak luminosity of $10^{34} \text{ cm}^{-2} \text{ s}^{-1}$. While proton-proton collisions are the primary mode of operation, proton-ion or ion-ion collisions are also possible. It provides collisions to the four large experiments ALICE, ATLAS, CMS and LHCb and is located at CERN inside the 27 km long tunnel previously occupied by LEP. ATLAS (A Toroidal Lhc ApparatuS) and CMS (Compact Muon Solenoid) are the two general-purpose detectors, which are used in a wide range of searches for new physics such as supersymmetry and extra dimensions. Their operation led to the discovery of a Higgs-like boson in 2012 [1, 2]. ALICE (A Large Ion Collider Experiment) is designed to study heavy-ion collisions and quark-gluon plasma. LHCb (Large Hadron Collider beauty) investigates the decay of B-mesons and CP-violation.

In 2011, the LHC operated at a centre-of-mass energy of $\sqrt{s} = 7 \text{ TeV}$ with ATLAS recording an integrated luminosity of about 5 fb^{-1} . Operation resumed after the end-of-year shutdown in 2012 with increased energy of $\sqrt{s} = 8 \text{ TeV}$ and approximately 20 fb^{-1} of data were recorded by ATLAS at the end of the run. February 2013 marked the beginning of the first long shutdown, necessary to enable proton-proton collisions at $\sqrt{s} = 14 \text{ TeV}$ and to reach the design luminosity. This was accompanied by upgrades to the experiments to cope with the new running conditions.

The second long shutdown is scheduled for 2018 and will increase the peak luminosity to $2\text{--}3 \times 10^{34} \text{ cm}^{-2} \text{ s}^{-1}$. The third long shutdown in 2022 is dedicated to the *High Luminosity-LHC* (HL-LHC) upgrade, further increasing the luminosity to $5\text{--}7 \times 10^{34} \text{ cm}^{-2} \text{ s}^{-1}$.

2.2. The Standard Model of Particle Physics

The *Standard Model of Particle Physics* [3] contains all known particles and is a unification of the *electromagnetic interaction*, the *weak interaction*, and the *strong interaction*.

2. The Large Hadron Collider

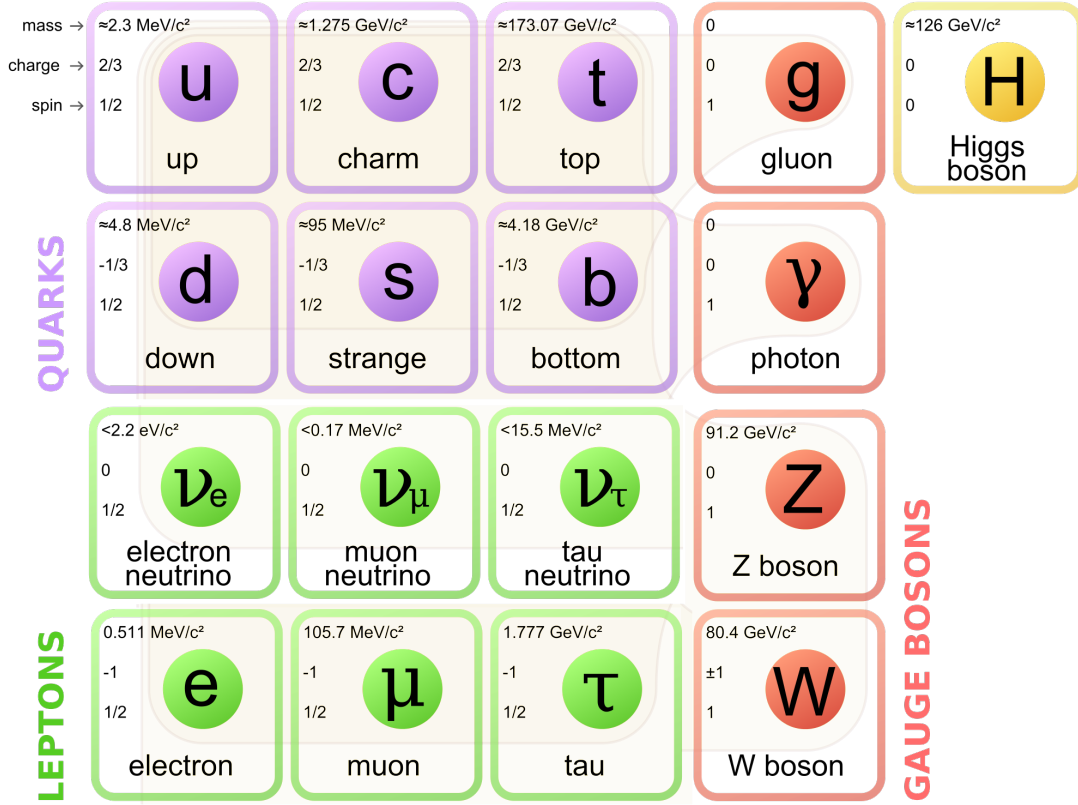


Figure 2.1.: The particles described by the Standard Model of Particle Physics.

The Standard Model contains spin- $1/2$ particles, the *fermions*, and integer-spin particles, the *bosons* (see Figure 2.1). The fermions in turn are divided into the *leptons* and the *quarks*, which are arranged in three generations with two leptons and two quarks each. The first generation contains the lightest particles which therefore cannot decay and are stable. They are the constituents of ordinary matter. The masses increase with the generation, which makes the particles in the second and third generation unstable. The exception are the *neutrinos*, which are assumed to be massless in the Standard Model.

Leptons carry a charge of -1 or 0 , quarks carry an electric charge of $2/3$ or $-1/3$ and are associated with a *colour charge* of *red*, *green*, or *blue*. To each particle corresponds an antiparticle, which has the same mass as the particle but the opposite charge.

The interactions between particles are mediated by four spin-1 *gauge bosons*. The strong force is mediated by eight massless *gluons* (g), which differ in their colour charge and couple to all particles with colour charge, namely the quarks and the gluons themselves. The electromagnetic force is mediated by the massless *photon* (γ), which couples to all electrically charged particles. The massive *intermediate vector bosons*, the charged $W^\pm$ boson and the neutral Z boson, are the force carriers of the weak interaction. They couple to all fermions as well as to themselves. Due to its charge, the photon also couples to the

$W^\pm$ boson.

The Higgs boson is the only scalar spin-0 particle in the Standard Model, unlike the fermions and gauge bosons. It is an excitation of the Higgs field, which has a non-zero vacuum expectation value. By interacting with this field, the massive gauge bosons acquire their mass.

2.3. The ATLAS Detector

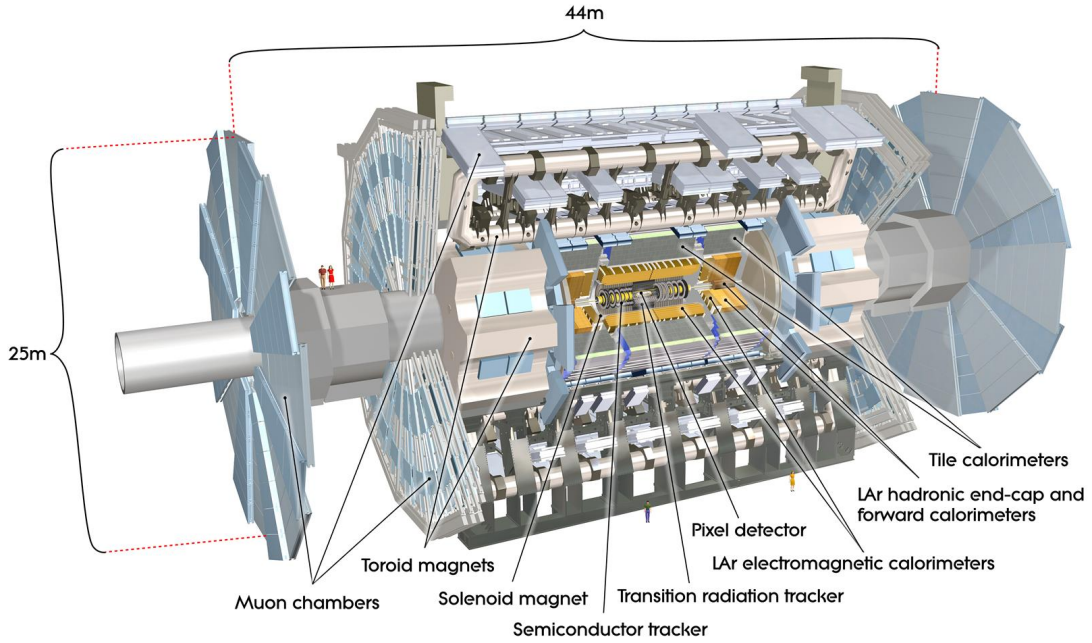


Figure 2.2.: Overview of the ATLAS detector [4].

With 44 m length and 25 m diameter, ATLAS is the largest detector at the LHC. The requirements for the detector design are dictated by the physical processes that are of interest or could indicate new physics. This includes the mechanism of spontaneous electroweak symmetry-breaking and the Higgs boson in the Standard Model as well as Supersymmetric (SUSY) extensions, searches for SUSY particles, top-quark physics and B physics [5].

The basic design criteria of the detector include very good electromagnetic calorimetry and full-coverage hadronic calorimetry, accurate jet and missing transverse energy measurements, high-precision muon momentum measurements, efficient tracking at high luminosity for high- p_T lepton-momentum measurements, electron and photon identification, τ -lepton and heavy-flavour identification and full event reconstruction capability at lower

2. The Large Hadron Collider

luminosity. Large acceptance in pseudorapidity with almost full azimuthal angle coverage as well as triggering and measurements of particles at low- p_T thresholds are also required.

The overall layout of the ATLAS detector is shown in Figure 2.2. The *Inner Detector* (ID) is the detector closest to the interaction point. It is surrounded by a thin superconducting solenoid, followed by the calorimeter system, which is enclosed by the toroid magnets and the Muon chambers [6].

2.3.1. Inner Detector

The *Inner Detector* is 7 m long and has a radius of 1.15 m. It contains three different detector technologies and is used for pattern recognition, momentum and vertex measurements and electron identification. Closest to the beam pipe is the *Pixel Detector*, followed by the *Semiconductor Tracker* (SCT) and the *Transition Radiation Tracker* (TRT).

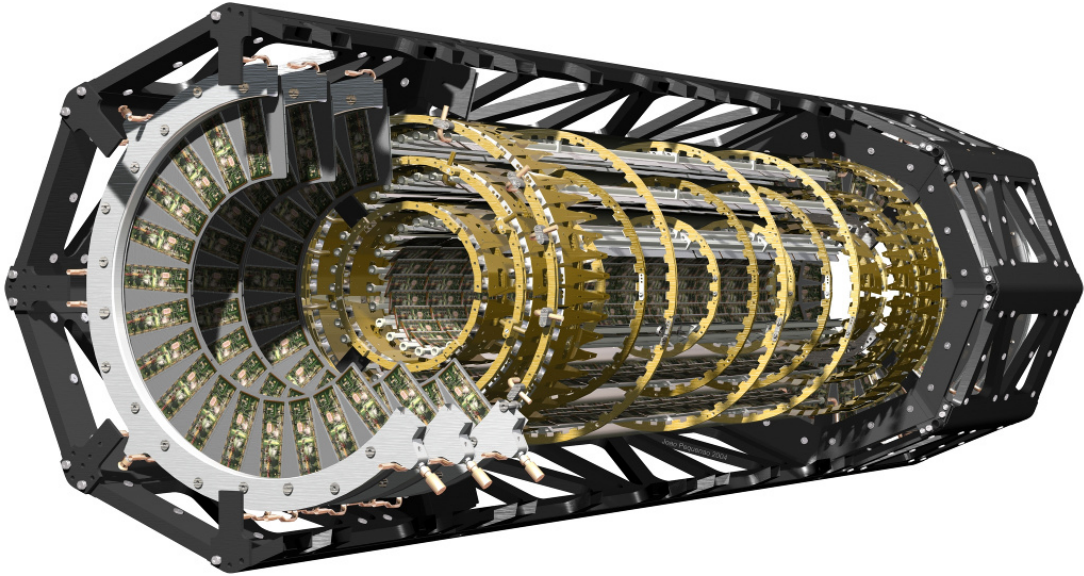


Figure 2.3.: The ATLAS pixel detector with 3 barrel layers and 3 disk layers at each end.

The *Pixel Detector* [7] consists of three barrel layers and two endcaps with three disks on each side (see Figure 2.3). It offers the highest granularity, as required for momentum and vertex measurements close to the interaction point. Typically, each track crosses three pixel layers. The detector contains identical pixel modules in the barrel and the disks. 1456 pixel modules are used in the barrel and 288 modules in the disks. Each module is composed of a silicon sensor with 47,232 pixels connected to 16 front-end chips (FE-I3) with 2880 pixel cells each. Also included is the *module control chip* (MCC) and

a flexible printed circuit to route signals and power. The detector has an active area of about 1.7 m^2 .

The SCT [4] is arranged in four barrel layers and two endcaps of nine disks and is designed to provide eight measurements per track. The SCT uses single-sided p-on-n silicon microstrip detectors. Two detectors are glued together back-to-back at a 40 mrad angle to obtain measurements in the direction of the beam pipe.

The TRT is based on the use of proportional counters. The TRT consists of 370,000 tubes (straws) filled with a gas mixture and containing a sense wire at the centre. The TRT barrel has three layers of 32 modules each, with the straws parallel to the beam. In the end-caps, they are arranged radially with a total of 224 layers of straws on each side. The layers are interleaved with the radiators. The straws are operated as drift tubes, while the radiators produce detectable X-rays when traversed by electrons [8]. The detector can discriminate between tracking hits and transition-radiation hits, thus providing excellent electron identification.

2.3.2. Calorimeters

The ID is enclosed by the *electromagnetic calorimeter* and the *hadronic calorimeter*, which measure the energy of particles leaving the ID.

The electromagnetic calorimeter is a sampling calorimeter with lead absorbers and liquid argon. It has a thickness of over 22 radiation lengths in the barrel and over 24 radiation lengths in the end-caps [4].

The *hadronic calorimeter* consists of the hadronic barrel calorimeter, a sampling calorimeter with plastic scintillator tiles interleaved with iron absorbers, the hadronic end-cap calorimeter with copper and liquid argon, and the high density forward calorimeter with copper and tungsten absorbers and liquid argon. The hadronic calorimeter is thick enough to provide good containment for hadronic showers and to reduce punch-through into the muon system to a minimum.

2.3.3. Muon Spectrometer

The *Muon Spectrometer* uses the large barrel and two smaller end-cap toroid magnets to bend muon tracks and four different chamber technologies to measure them. In the barrel region, the chambers are arranged in three cylindrical layers around the beam axis. In the end-caps, the chambers are installed vertically, again in three layers. The track coordinates are measured by *Monitored Drift Tubes* (MDTs) and *Cathode Strip Chambers* (CSCs). The trigger system uses *Resistive Plate Chambers* (RPCs) in the barrel and *Thin*

Gap Chambers (TGCs) in the end-cap regions.

2.4. Pixel Detector Upgrades

2.4.1. IBL Upgrade

The *Insertable B-Layer* (IBL) is an additional pixel layer installed between the current inner pixel layer (B-layer) and a new beam-pipe. It was inserted into the detector during the 2013-2015 LHC shutdown. By adding an additional layer, the degrading tracking efficiency of the existing detector due to failing modules and radiation damage can be compensated and the tracking precision is improved.

For the design of the IBL, several new constraints had to be taken into account. The smaller radius required development of a more radiation hard technology and placed restrictions on the arrangement of modules and support structures. The IBL consists of 14 staves with 32 modules each, arranged around the beam pipe and tilted with a slight overlap. This gives full coverage in ϕ , but there is not enough space to overlap modules along z , which leaves gaps in z direction. The material in IBL is reduced to almost half the radiation length of the existing B-layer, reaching 1.54% of X_0 [9]. Two sensor technologies are used in the modules: planar sensors with two front end chips and one sensor per module (2-chip module), or 3D sensors with a single chip and sensor (1-chip module). Planar sensors usually consist of a lowly doped substrate with highly doped implants on the surface, while the electrode structure of 3D sensors extends well into the substrate.

The FE-I3 front end chips used in the present detector do not have a sufficient hit rate capability and radiation hardness and the active fraction of the chip is too small to build a compact layer with high geometric acceptance. This led to the development of the FE-I4, which is described in more detail in Chapter 4.

2.4.2. HL-LHC Upgrade

To cope with the increase in radiation and occupancy in the HL-LHC, a complete replacement of the ID with an all-silicon design is proposed [10]. The design consists of cylinders with four pixel layers followed by three short-strip layers and two long-strip layers. The forward regions are covered by six pixel disks and seven strip disks. The sensors are of finer granularity to deal with the very high pile-up and to be able to reconstruct tracks in high-energy jets. Technologies under consideration include existing solutions such as

planar sensors, 3D sensors, and diamond sensors, which use diamond instead of silicon as semiconductor material.

Especially the innermost layer has to withstand the high fluences close to the interaction point. The 3D sensors require a relatively low depletion voltage even after high irradiation dose and the diamond detectors offer high radiation hardness, but both technologies are still expensive. Planar sensors can be mass produced by multiple vendors in high quality with high yields and low cost, which makes them interesting for the large area required for the outer layers.

Readout of the sensors also entails the need for new pixel readout chips beyond the FE-I4 to accommodate for both the smaller pixels as well as the higher bandwidth and new DAQ and trigger protocols.

3. Semiconductor Pixel Detectors

3.1. Interaction of Particles with Matter

To register particles traversing a detector, they have to interact with the material in a way that induces a measurable signal in the detector. For semiconductor detectors, this is the creation of electron-hole pairs due to ionisation, which in turn are separated by an externally applied electric field and cause a measurable current.

Depending on the type of particles, the interaction processes differ. They can be grouped into light charged particles, the electron and positron, heavy charged particles such as muons, protons or α -particles, and photons. Neutral particles are not considered here.

Heavy Charged Particles

The predominant interactions of heavy charged particles with matter are inelastic collisions with electrons of the material, and elastic scattering from nuclei, while the emission of Cherenkov radiation, nuclear reactions, and bremsstrahlung are extremely rare processes [11].

The average energy loss per distance of a heavy charged particle is given by the *Bethe-Bloch formula*

$$-\left\langle \frac{dE}{dx} \right\rangle = 2\pi N_a r_e^2 m_e c^2 \rho \frac{Z}{A} \frac{z^2}{\beta^2} \left[\ln \left(\frac{2m_e \gamma^2 v^2 W_{max}}{I^2} \right) - 2\beta^2 - \delta - 2\frac{C}{Z} \right]$$

with the symbols listed in Table 3.1. The formula remains valid down to $\beta \simeq 0.1$ and up to $\beta \simeq 1000$, where radiative losses start to dominate. This is shown in Figure 3.1 for muons in copper.

For $\beta\gamma \approx 3.5$, the energy loss reaches a minimum where $-\left\langle \frac{dE}{dx} \right\rangle_{mip} = 1.5 \frac{\text{GeV}}{\text{g cm}^{-2}}$ is a good approximation for most materials. Particles at this point are called *minimum ionizing particles* (mip) and are useful to estimate the smallest signal of a charged particle to be expected in a detector.

Table 3.1.: Symbols in the Bethe-Bloch formula [11].

r_e	classical electron radius
m_e	electron mass
N_a	Avogadro's number
I	mean excitation potential
Z	atomic number of absorbing material
A	atomic weight of absorbing material
ρ	density of absorbing material
z	charge of incident particle in units of e
β	v/c of the incident particle
γ	$1/\sqrt{1-\beta^2}$
δ	density correction
C	shell correction
W_{max}	maximum energy transfer in a single collision

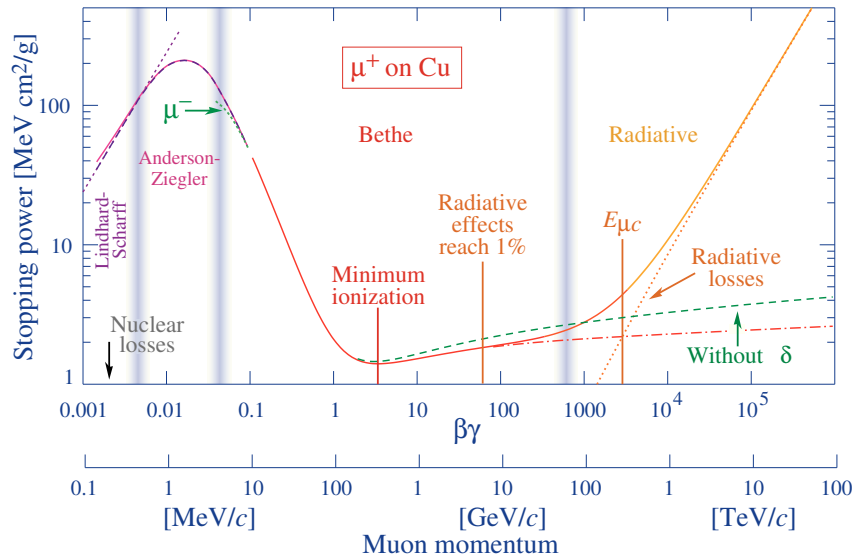


Figure 3.1.: Energy loss for muons in copper over nine orders of magnitude in momentum [12].

Electrons

Electrons and positrons lose energy via similar mechanisms as heavy charged particles, but due to their low mass, bremsstrahlung is much more dominant and the scattering on shell electrons has different kinematic constraints. At low energies, ionisation is still the dominant process, but above the *critical energy* of approximately [13]

$$E_c \approx \frac{800}{Z} \text{ MeV},$$

with the charge Z of the nucleus, bremsstrahlung becomes dominant.

For high energy electrons and photons, the electromagnetic interactions are characterised by the *radiation length* X_0 , which is the mean distance after which the energy is reduced by a factor of $1/e$. It is approximated by [13]

$$X_0 \approx \frac{1}{4\alpha n Z^2 r_e^2 \ln(287/Z^{1/2})},$$

where n is the number density of nuclei and r_e is the classical electron radius. For silicon, the radiation length is 9.36 cm.

Photons

The interactions of photons are very different to charged particles, partly due to the lack of an electric charge. To detect photons, they first have to create charged particles in the material, which in turn cause ionisation and a measurable signal. Photons are either completely absorbed or scattered under relatively large angles, which is in contrast to the behaviour of charged particles. This leads to the intensity of a beam passing through a material decreasing exponentially with the distance x [13]

$$I = I_0 e^{-\mu x},$$

where I_0 is the initial intensity and μ is the material specific attenuation coefficient.

The predominant interactions of photons with matter for increasing energy are:

1. Photoelectric effect
2. Compton scattering
3. Pair production

The cross sections for photons in carbon are shown in Figure 3.2. For low photon energies, the photoelectric effect is the dominant process. The photon is absorbed by

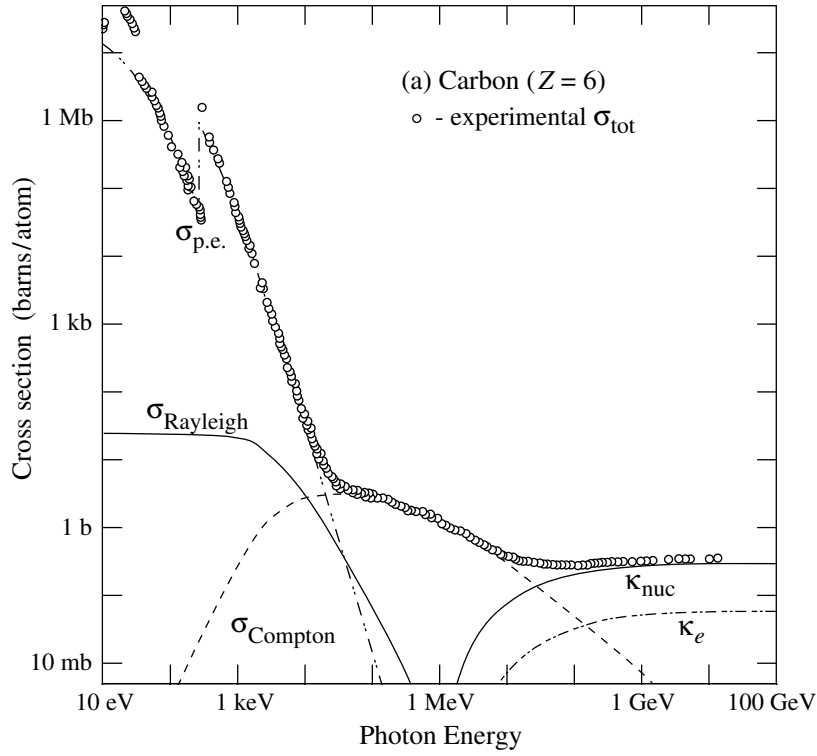


Figure 3.2.: Photon cross section in carbon with the contributing cross sections $\sigma_{\text{p.e.}}$ for the photoelectric effect, σ_{Rayleigh} for Rayleigh scattering, σ_{Compton} for Compton scattering, and κ_{nuc} and κ_e for pair production in the nuclear respective in the electron field [12].

an electron, which is then ejected from the atom and carries the energy of the photon reduced by the binding energy. For higher energies, photons can scatter on electrons in the material instead of being absorbed. At sufficiently large photon energies, the electrons can be considered essentially free and the process is described by Compton scattering. Pair creation becomes possible above energies of twice the electron mass. A photon can be converted into an electron-positron pair, which requires the presence of a nucleus or an electron to satisfy conservation of momentum. Here, the radiation length corresponds to $7/9$ of the mean free path for pair production.

3.2. Semiconductor Detectors

The detection mechanism for particles in semiconductors is based on the creation of electron-hole pairs. For silicon, an average energy of 3.61 eV is required to create one electron-hole pair.

pn Semiconductor Junction

A pn-junction in a semiconductor forms between n-type and p-type material, where the n-type material has been doped with *donor* atoms that have one additional valence electron and the p-type has been doped with *acceptor* atoms that have one valence electron less than the material. For silicon with four valence electrons, typical elements used as p-type dopant are gallium, aluminium, indium and boron, while phosphorus and arsenic are typical dopants for n-type materials. Pn-junctions are the basic building block of semiconductor detectors.

Because the concentrations of electrons and holes between both materials differ, holes diffuse towards the n-region and electrons diffuse towards the p-region, which then recombine and form the *depletion zone*. As both regions were initially neutral and the atoms remain stationary, an electric field is formed which eventually halts the diffusion process. The depletion zone is devoid of free charge carriers and has a very high resistivity compared to the n-type and p-type materials alone.

The potential across the depletion zone makes it sensitive to electron-hole pairs created inside, but the width is determined by the doping concentration and the low intrinsic potential offers poor charge collection. To increase the sensitive volume of the detector and provide efficient charge collection, the pn-junction is operated with reverse bias, which increases the potential across the junction. By applying a voltage across the junction with the negative terminal on the p-region and the positive terminal on the n-region, holes from the p-region and electrons from the n-region are attracted towards the contacts and the depletion zone widens. In contrast to a pn-junction with forward bias which becomes conductive, the pn-junction with reverse bias conducts very little current. Inside the depletion zone, electron-hole pairs are separated by the electric field and drift towards the electrodes, which causes a measurable current according to the *Shockley–Ramo theorem* [14, 15]. The pairs are created by the energy deposition of particles, but a current is also caused by thermal effects and by diffusion from the undepleted volume, which is the undesired *leakage current* that contributes to the noise of the detector.

For sufficiently large voltages, the depletion zone can be extended to cover most of the detector volume and is no longer determined by the doping concentration. The achievable width is limited by the breakdown voltage of the junction. The width of the depletion zone is [13]

$$d = x_n + x_p = \sqrt{(V_0 + V_B) \frac{2\varepsilon (N_A + N_D)}{e N_A N_D}},$$

where d is the total width and x_n and x_p are the widths on the n- and p-side. N_A and N_D are the acceptor and donor concentrations and V_0 and V_B are the potential of the

3. Semiconductor Pixel Detectors

junction itself and the external bias voltage. When the concentrations are very different, the depletion zone extends mostly into the lower doped side of the junction. Together with a depletion voltage well above the built-in potential, the width is approximated by

$$d \approx \sqrt{\frac{2\varepsilon V_B}{eN_D}}.$$

Hybrid Pixel Detectors

For use in a tracking detector, the sensor has to provide spatial resolution for tracks passing through it. To this end, the sensor is segmented into pixels which consist of individual pn-junctions and have to be read out separately. The segmentation is usually achieved by introducing small implants with opposite doping type into the wafer.

With the number of pixels in the order of several ten thousands per sensor not being uncommon, the readout poses a significant challenge. The two usual approaches are *monolithic sensors*, where the readout electronics are integrated on the same chip as the sensor, and *hybrid sensors*, where the readout is performed by a separate chip which has to be connected to the sensor.

The ATLAS pixel detector uses hybrid modules consisting of sensors bump-bonded to the FE-I3 or FE-I4 readout chips. A schematic for a single pixel is shown in Figure 3.3. The pixels are connected to the readout chip with small conductive bump balls. The sensor is operated with reverse bias and the electron-hole pairs created by particles are separated and drift toward the electrodes. The readout chip processes the induced current signal for all individual pixels. Each channel consists of an analog part, which uses an amplifier and a discriminator to generate a pulse with a length proportional to the charge signal, and a digital part, which measures the pulse width and passes the data to a buffer shared between several pixels until readout. Deviations between channels can be compensated by adjusting the discriminator threshold and other settings individually. The parallel readout is necessary in order to achieve the high bandwidth required for the ATLAS detector.

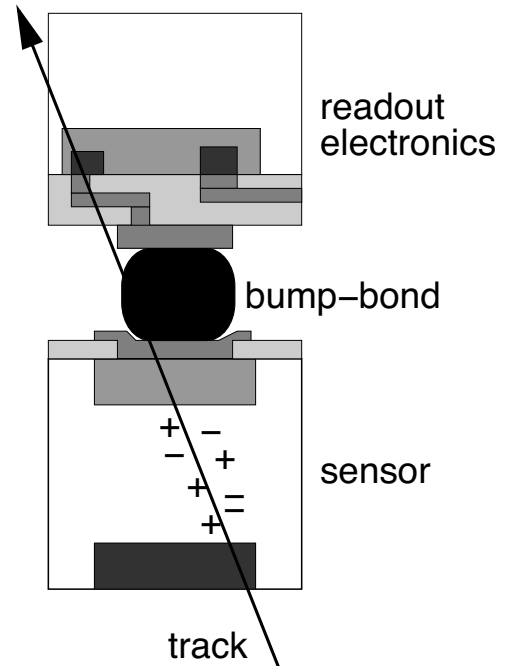


Figure 3.3.: Cross section of a single pixel of a hybrid pixel detector [7].

4. The New Front-End Readout Chip FE-I4

4.1. Motivation

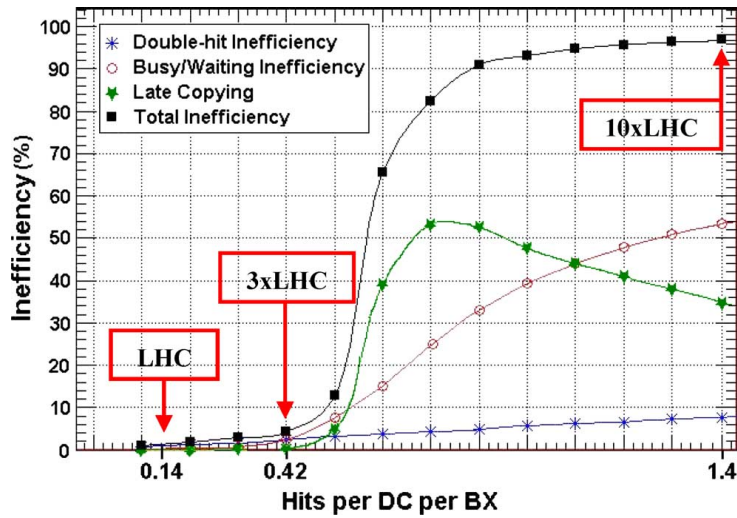


Figure 4.1.: Inefficiencies of the FE-I3 with increasing hit-rate [16].

The planned upgrades to the LHC luminosity exceed the capabilities of the front-end chips currently used for pixel readout. Figure 4.1 shows the inefficiency of the FE-I3 as function of the hit rate. At around three times the LHC luminosity, the inefficiency starts to increase significantly. Double-hit inefficiency occurs, when a pixel is hit twice in fast succession and the second hit is not recognized as separate event. The busy/waiting inefficiency is caused by the read-out architecture of the pixel array, which uses a shared bus inside each double-column. Every registered hit in the FE-I3 is transferred over this bus and buffered. As the bus is shared between all pixels in a double-column, but can only be used for one transfer at a time, the waiting time increases with the hit rate. As the pixels can only store a single hit, subsequent hits during this time are lost. This effect increases as the bus becomes saturated. A high waiting time can also cause the hit data to arrive too late for readout, which is the late copying inefficiency.

4. The New Front-End Readout Chip FE-I4

The increasing hit rates lead to problems both for the readout efficiency and the additional requirements for radiation hardness. The FE-I4 is the first step in addressing these issues, as it is designed to meet the requirements for operating closer to the interaction point in the IBL upgrade, and it could still be adapted for use in the outer layers of the pixel detector, when the luminosity is increased even further in a subsequent upgrade [16].

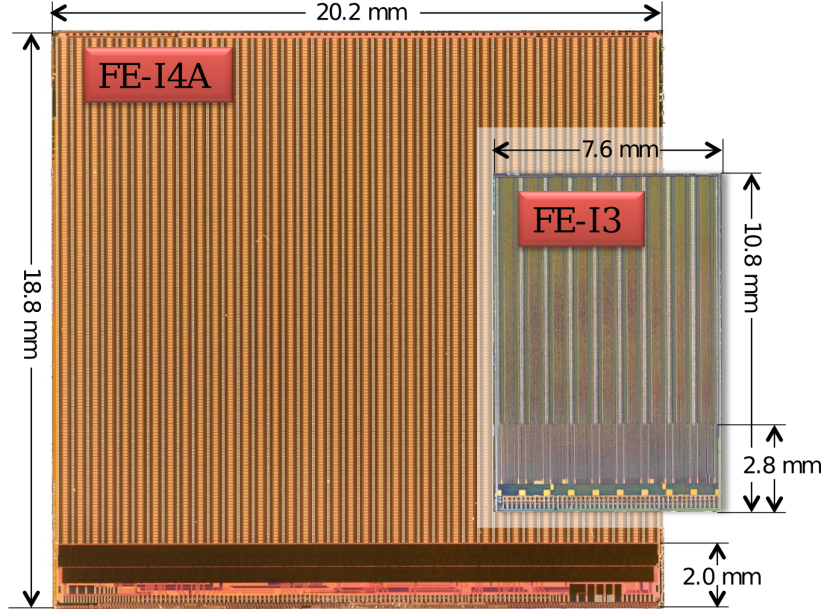


Figure 4.2.: Comparison of the FE-I4A and the FE-I3 at same scale [17].

4.2. Design of the FE-I4 Chip

The FE-I4 is manufactured in a 130 nm process with a reduced pixel size of $50\text{ }\mu\text{m} \times 250\text{ }\mu\text{m}$ compared to the FE-I3 with $50\text{ }\mu\text{m} \times 400\text{ }\mu\text{m}$ pixels and a 250 nm process. This helps to reduce the double-hit inefficiency and increases the spatial resolution. Figure 4.2 shows the FE-I4 in comparison to the FE-I3. With $20.2\text{ mm} \times 18.8\text{ mm}$, the FE-I4 is significantly larger than the FE-I3 with only $7.6\text{ mm} \times 10.8\text{ mm}$. The pixel array of the FE-I4 takes up 89% of the chip, while the active area of the FE-I3 is only 74%.

Figure 4.3 shows a schematic overview of the chip. The FE-I4 pixel array is comprised of 26880 pixels, organized in 336 rows and 80 columns. Two columns each are grouped together to form 40 double-columns. The double-columns in turn are divided into 2×2 pixel regions with four separate analogue pixels and one common *Pixel Digital Region* (PDR). Hits are stored in the PDR alongside timing information. An external trigger can then select events with the matching trigger latency, which are transferred to the *End of*

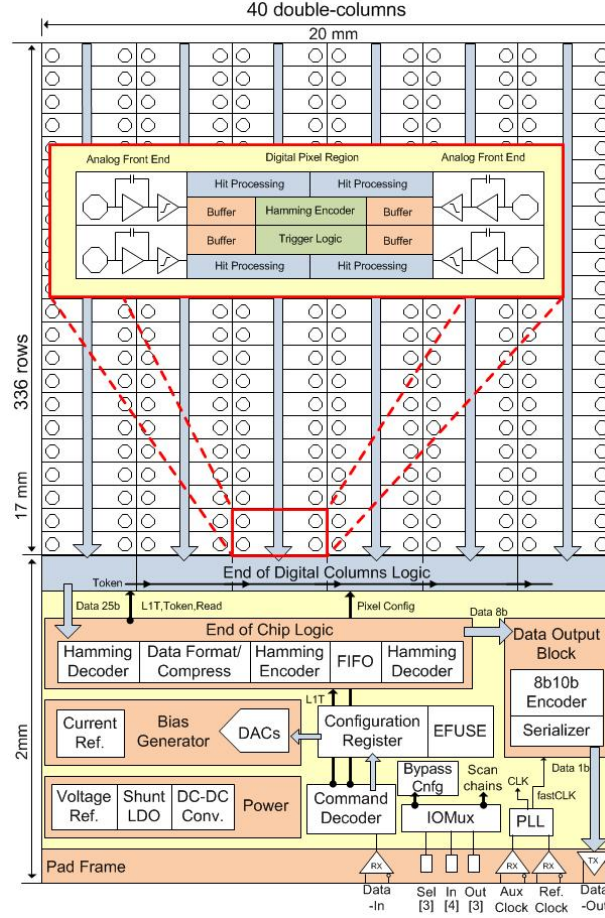


Figure 4.3.: FE-I4 chip diagram [18].

Digital Column Logic (EODCL). Hits which exceed the trigger latency are discarded. By only transferring the selected events, the bus is not congested by unwanted events, thus avoiding this issue of the FE-I3. The double-hit inefficiency is addressed by the reduced pixel size, which automatically reduces the pixel hit probability, and a faster return to baseline, which reduces the dead time of the analog pixel.

The lower edge of the chip contains the *End of Chip Logic* (EOCHL), which is responsible for processing the hit data and triggers and inserting service records and register readbacks into the data stream. The *Data Output Block* (DOB) then encodes the data stream and sends it to the transmitter. The *Command Decoder* (CMD) receives commands and generates the appropriate signals to control the other components of the chip. All global parameters of the FE-I4 are stored in the *Configuration Memory* (CNFGMEM). The *Phase Locked Loop* (PLL) is locked to the 40 MHz clock input and provides multiples (1x, 2x, 4x and 8x) of the clock input to the data transmitter and other blocks. The FE-I4 also contains several voltage and current regulators, some of which are programmable.

4. The New Front-End Readout Chip FE-I4

Many internal signals can be routed to pads on the edge of the chip. With this, parts of the chip can be bypassed or accessed directly. This is useful for testing, but not necessary for operating the FE-I4 under normal circumstances.

There are two revisions of the FE-I4, namely the prototype FE-I4A and the FE-I4B, the improved revision used in the IBL detector.

4.2.1. Analog Pixel

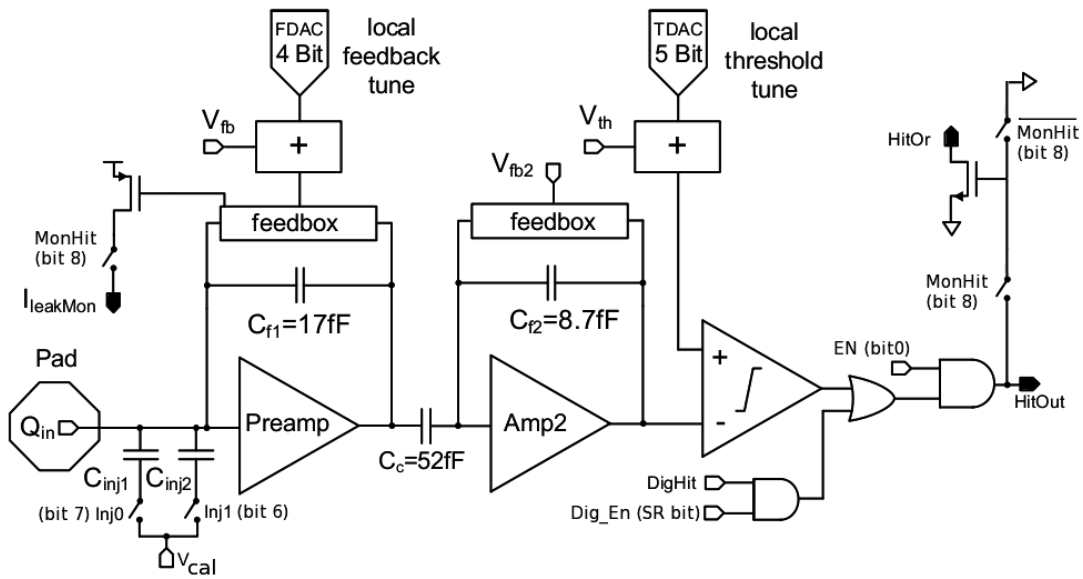


Figure 4.4.: Schematic diagram of an analog pixel [18].

Figure 4.4 illustrates the inner structure of an analog pixel. The sensor is connected to the input pad on the left, which feeds into the first stage of the two-stage amplifier, followed by the discriminator. The capacitors for charge injection are also connected to the amplifier. They allow the injection of a known amount of charge into the pixel, independent from the sensor. This is used for testing and characterisation of the pixels, where the response to the injected charge is measured.

The discriminator converts the analog charge signal into a digital square wave. The output of the discriminator is zero as long as the input is below the threshold, and otherwise one. The return to baseline of the amplifier and the discriminator threshold are adjustable for each pixel. The amplifier has a nearly linear return to baseline, which leads to a pulse width of the discriminator proportional to the input charge. This in turn can be measured as the *Time over Threshold* (ToT) of the discriminator.

4.3. Operation

4.3.1. Registers

The FE-I4 contains programmable registers which control certain aspects of the chip. There are two types of registers. The global registers (GR) are common to the whole front-end and control parameters that affect the entire chip, while each pixel has its own independent set of pixel registers, which directly control the pixel electronics. The global registers are organized and accessed as 36 words with 16 bit each and subdivided into configuration parameters of one bit to 40 bit size.

Each pixel has six pixel registers with a combined size of 13 bit. The pixel registers are accessed over the pixel shift register. This register is 672 bit long, directly corresponding to the 672 pixels in a double column. The pixel registers are shown in Figure 4.4, labelled as *FDAC*, *TDAC*, *Inj0*, *Inj1*, *MonHit* and *EN*. The pixel shift register is also connected to the *Dig_En* input. *FDAC* controls the feedback current of the preamplifier and *TDAC* adjusts the threshold of the discriminator. *Inj0*, *Inj1* and *Dig_En* are used to inject signals into the analog and digital part of the pixel respectively, which is explained in more detail in Chapter 4.4.2. The *EN* register enables the pixel for readout.

All registers are designed to be tolerant to *Single Event Upsets* (SEU), as bit errors can disturb the operation of the chip. SEUs are unexpected changes of state caused by ionising particles in semiconductor circuits. They do not damage the chip, but interfere with the electronics and can cause undefined or erratic behaviour. The tolerance is achieved by triplicating all registers and using error-correcting codes[18].

4.3.2. Commands

The FE-I4 receives commands over a simple serial link, typically at 40 MHz clock speed. Commands are divided into the three classes *Trigger*, *Fast* and *Slow*. The chip can either be in *Run Mode*, where it only responds to triggers and fast commands, or in *Slow Mode*, where it only executes slow commands. The only exception is the *RunMode* command, which switches between the two modes and therefore has to be executed in both modes. Table 4.1 shows the trigger command, the fast commands and the unique header that identifies slow commands. The trigger command is the shortest command and initiates the acquisition of hit data. The *BCR* commands resets the bunch counter of the FE-I4, which counts the number of 40 MHz pulses from the external clock, corresponding to the bunch crossing rate at the LHC. The *ECR* command clears the data path of the frontend and can be issued to restore synchronisation across several chips. The *CAL* command

4. The New Front-End Readout Chip FE-I4

generates calibration pulses which can be used to inject signals into the digital or analog parts of the pixel array.

The slow commands are listed in Table 4.2. They are longer than the other commands and include a chip identification to address a specific FE-I4. This allows to share the same command link between up to eight chips while being able to address each of them individually. One bit in the identifier indicates that all FE-I4 should execute the command. The slow commands *RdRegister* and *WrRegister* are responsible for reading and writing the global registers. The *WrFrontEnd* command writes 672 bits into the shift registers of the selected double-columns. Issuing the *GlobalReset* command resets the chip to its initial state and clears the content of the whole frontend. The *GlobalPulse* command generates a pulse which can be routed to different parts of the chip to invoke special behaviour such as copying the content of the pixel shift register into the pixel registers. The *RunMode* commands selects RunMode when field 5 is set to 111000 and ConfMode for 000111.

Table 4.1.: Trigger and Fast Commands [18].

Name	Field 1	Field 2	Description
size (bits):	5	4	
LV1	11101	-	Level 1 Trigger
BCR	10110	0001	Bunch Counter Reset
ECR	10110	0010	Event Counter Reset
CAL	10110	0100	Calibration Pulse
Slow	10110	1000	Slow command header

Table 4.2.: Slow Commands [18].

Name	Field 3	Field 4	Field 5	Field 6	Description
size (bits):	4	4	6		
RdRegister	0001	ChipId	Address	-	Read addressed global memory register
WrRegister	0010	ChipId	Address	Data	Write into addressed global memory register
WrFrontEnd	0100	ChipId	xxxxxx	Data	Write conf data to selected shift register(s)
GlobalReset	1000	ChipId	-	-	Reset command; Puts the chip in its idle state
GlobalPulse	1001	ChipId	Width	-	Has variable pulse width and functionality
RunMode	1010	ChipId	Mode	-	Sets RunMode or ConfMode

4.3.3. Output Format

In contrast to the control link, which receives the commands unencoded at 40 Mbit/s, the data link transmits data with 8b/10b encoding at a configurable rate between 40 Mbit/s

and 320 Mbit/s. While 160 MHz is the default setting and is currently used in ATLAS, the clock can be selected to be 1x, 2x, 4x or 8x the 40 MHz clock input.

8b/10b encoding translates each 8-bit byte into a 10-bit sequence with desirable properties to facilitate clock recovery, AC-coupling and error detection. As only a subset of the 1024 possible 10-bit sequences is used for encoding, additional symbols are introduced to mark the start and end of packets or to indicate that the transmitter is idle.

Before encoding the data are organized in 24-bit records as shown in Table 4.3. The *Empty Record* is only transmitted when 8b/10b encoding is disabled. All records except for the *Data Record* and the *Empty Record* start with the pattern 11101, followed by a 3-bit identifier and the 16-bit payload.

The start of pixel data is marked by the *DataHeader*, which contains a trigger counter and a BCID counter. The actual pixel data consisting of the column, row and ToT of a hit is stored in a *DataRecord*. In the case of two hits in neighbouring pixels of the same column, both hits are stored in the two ToT fields of the same record. *AddressRecords* and *ValueRecords* are used to read global registers and pixel registers. The address of the register is stored in an *AddressRecord*, followed by the value of the register in a *ValueRecord*. *ServiceRecords* indicate errors or additional information and hold a code that expresses the kind of condition alongside with a 10-bit value.

Table 4.3.: The six 24-bit Record Words [18].

Record Word	Field 1	Field 2	Field 3	Field 4	Field 5
Data Header (DH)	11101	001	Flag	LV1ID [4:0]	bcID [9:0]
Data Record (DR)	Column [6:0]	Row [8:0]	ToT(1) [3:0]	ToT(2) [3:0]	
Address Record (AR)	11101	010	Type	Address [14:0]	
Value Record (VR)	11101	100	Value [15:0]		
Service Record (SR)	11101	111	Code [5:0]	Number [9:0]	
Empty Record (ER)	abcdefgh	abcdefgh	abcdefgh		

4.4. Data Acquisition

On power-up, all registers are cleared and the FE-I4 is left in *ConfMode*. For data acquisition, an appropriate configuration has to be written to the registers and the chip has to be placed in *RunMode*. Signals from a connected sensor or from the internal pulse generator are registered and stored as hits in the corresponding pixel, as long as they exceed the discriminator threshold. They can then be read-out by issuing a trigger command.

4.4.1. Triggering

When a hit is detected, it is stored locally alongside with an 8-bit counter that measures the time since the hit was registered. The counter starts at 255 and counts down until it reaches the trigger latency stored in a global register. If a trigger is issued, all regions that have a counter matching this latency value are flagged for read-out. Otherwise, the hits are discarded and the counters are reset.

Triggers can be supplied externally by executing the *LV1* command, or generated internally by enabling the *HitBus*, which is the logical *or* of all selected discriminator outputs. By setting a pixel register, individual pixels can be connected to the *HitBus*. When one of the connected pixels detects a hit, this becomes visible on the *HitBus* and a trigger command is issued.

4.4.2. Pulse Generator

The programmable pulse generator of the FE-I4 can be used to inject signals into both the analog and the digital part of each pixel. This is called *Analog Injection* or *Digital Injection* and allows to produce hits even when no sensor is connected, which is used to test various parts of the pixel by injecting a known signal and measuring the response.

Figure 4.5 shows the analog and digital waveforms generated by the pulse generator in response to a *CAL* command. Almost all parameters of the signal, such as the delay and the pulse width are configurable via global registers. The pulse generator can also be activated by executing a *GlobalPulse* command and routing the global pulse to the pulse generator.

For *Analog Injection*, each pixel has two capacitors, C_{inj1} and C_{inj2} , which can be selected by the corresponding pixel registers. They are connected in parallel to the preamplifier and the calibration voltage V_{cal} , as can be seen in Figure 4.4. The shape of the applied signal is controlled by the pulse generator. The output of the pulse generator is at V_{cal} and after a certain delay after receiving the *CAL* command, the output is pulled to ground and held there for the duration t_{w2} . Afterwards, the voltage increases over the duration t_R and returns to V_{cal} . The falling edge injects a negative charge into the amplifier, possibly generating a hit.

Digital Injection has to be enabled globally in a global register and for individual pixels in the pixel shift register. The pulse generator will then generate a hit in the selected pixels, completely bypassing the amplifier and the discriminator.

When selecting double-columns for injection, the behaviour between analog and digital

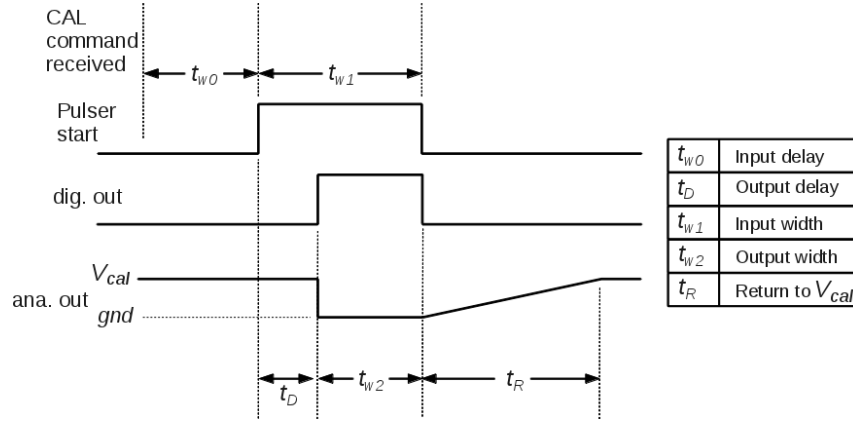


Figure 4.5.: Outputs of the pulse generator [18].

injection differs slightly. For *Digital Injection*, selecting¹ the double-column n selects the columns $2n + 1$ and $2n + 2$, while for *Analog Injection*, the columns $2n$ and $2n + 1$ are selected and selecting double-column 0 only selects column 1, but selecting double-column 39 selects the columns 78, 79 and 80.

¹Columns are numbered from 1 to 80 while double-columns are numbered from 0 to 39.

5. The USBpix System

The *USBpix System* was developed as a USB-based test system for ATLAS hybrid pixel detectors containing the new FE-I4 readout chip [19]. It is compatible to the FE-I3 and the FE-I4 and uses an FPGA (Field Programmable Gate Array), which can be reprogrammed as needed and allows to adapt the system to new requirements without changes to the hardware.

A readout software such as *STcontrol* is used on the PC to control the USBpix board over USB and to retrieve the measured data. The USBpix system can be used together with a variety of additional devices, as often required in more sophisticated test setups. This includes a *Trigger Logic Unit* (TLU), which processes triggers from external sources such as scintillators in a testbeam setup, or a multimeter to measure analog parameters during a scan.

5.1. Hardware Overview

The USBpix system consists of the Multi-IO Board and several extension cards which connect to the FEs or provide additional functionality. A typical setup is shown in Figure 5.1.

Multi-IO Board (MIO)

The *S3 Multi IO USB Board* contains a USB 2.0 microcontroller, a Xilinx Spartan-3 FPGA and 2 MiB SRAM [20]. It has several LEMO connectors for triggering, one RJ45 port for connecting to a TLU, and connects to extension cards via a 100-pin KEL-connector. The microcontroller is connected over USB 2.0 to the PC and handles the communication with the host via the USBpix protocol described in Chapter 6.3. The FPGA sends commands to the FE, receives data and does processing and buffering using the SRAM. Figure 5.2 gives a detailed overview of the functional blocks and connections on the MIO.

5. The USBpix System

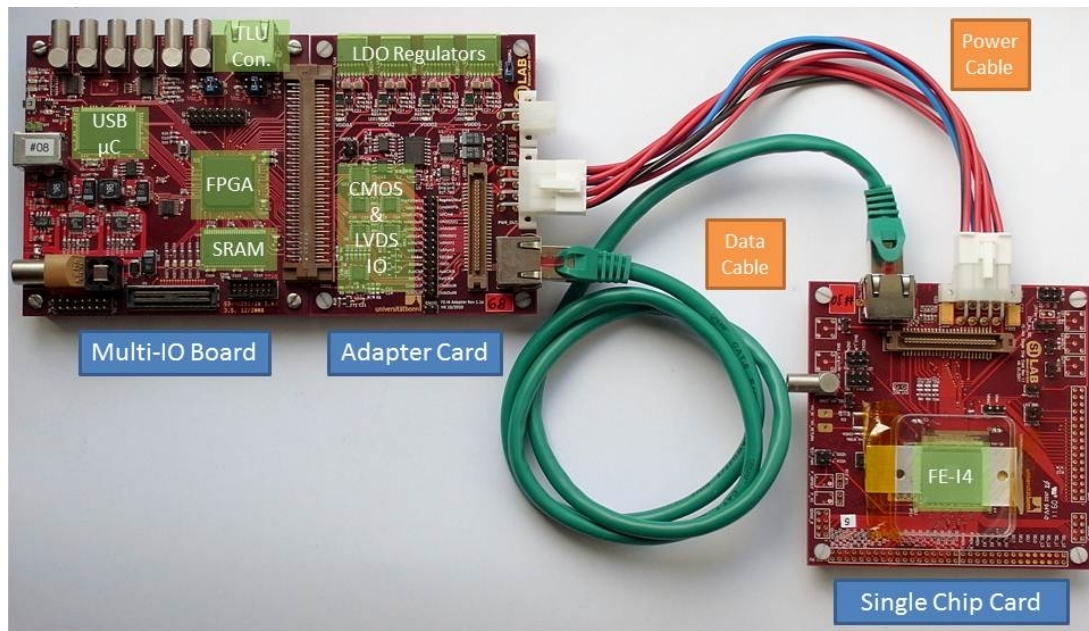


Figure 5.1.: A typical setup with the *Multi-IO Board* and the *Adapter Card* connected to a single FE-I4.

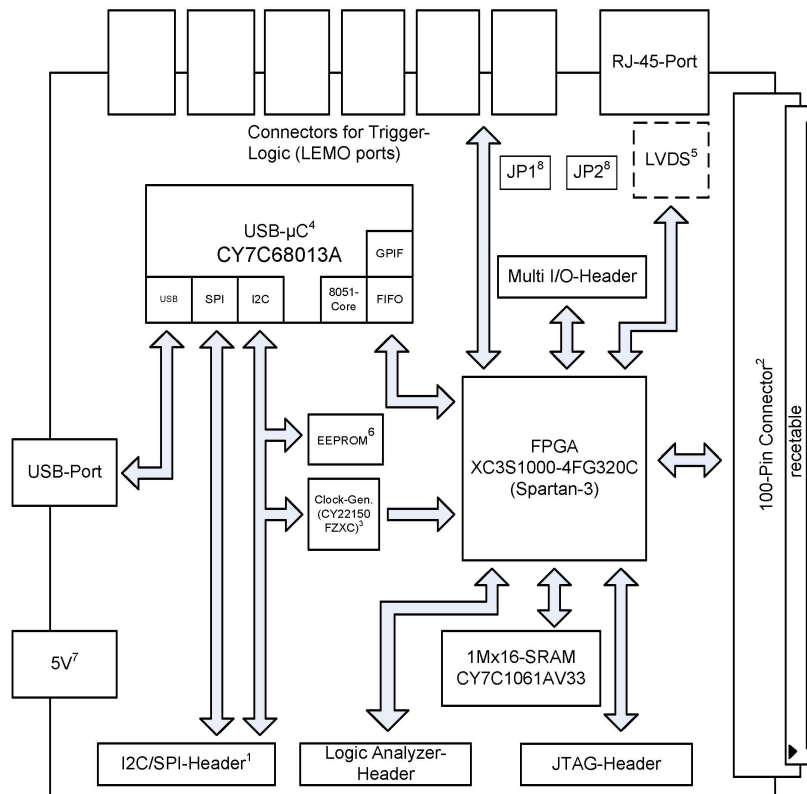


Figure 5.2.: Functional block diagram of the S3 Multi IO System [20].

Adapter Card for FE-I4

The *Adapter Card* is connected to the MIO using the 100-pin KEL-connector and to one FE via the RJ45 port or a ribbon cable. It provides voltage regulators and LVDS transceivers, which convert between the single ended signals used on the MIO and the differential signals used by the FEs.

Burn-In Card (BIC)

The *Burn-In Card* is a replacement for the *Adapter Card* and can operate up to four FEs. It uses the same 100-pin connector, but has four RJ45 ports accompanied by four MOLEX power connectors [21].

General Purpose Analog Card (GPAC)

The *General Purpose Analog Card* extends the digital IO capabilities of the MIO by analog blocks such as power supplies, voltage and current sources and fast ADC. It also provides several LVDS channels and low voltage CMOS level shifters. The GPAC has two 100-pin connectors, one connects to the MIO and one to the Device-Under-Test (DUT) [22].

5.2. Readout Software

5.2.1. STcontrol

STcontrol is a readout and test software for the USBpix board. It is written in C++, using the Qt framework and ROOT. STcontrol is based on the ATLAS *PixLib* package, a collection of C++ classes originally developed to provide access to the ATLAS Pixel detector read-out devices (RODs) [23]. It provides a graphical user interface (GUI) to configure the connected hardware, perform scans, and review and analyse the collected data.

It is described in more detail in Chapter 6.1.

5.2.2. pyBAR

PyBAR (Bonn ATLAS Readout in Python and C++) is a readout and test software similar to STcontrol [24]. Most of the software is written in Python with performance critical parts written in C++. In contrast to STcontrol, it does not offer a GUI but uses a collection of Python scripts to perform scans. PyBAR comes with its own FPGA firmware based on the *basil* framework [25].

5. The USBpix System

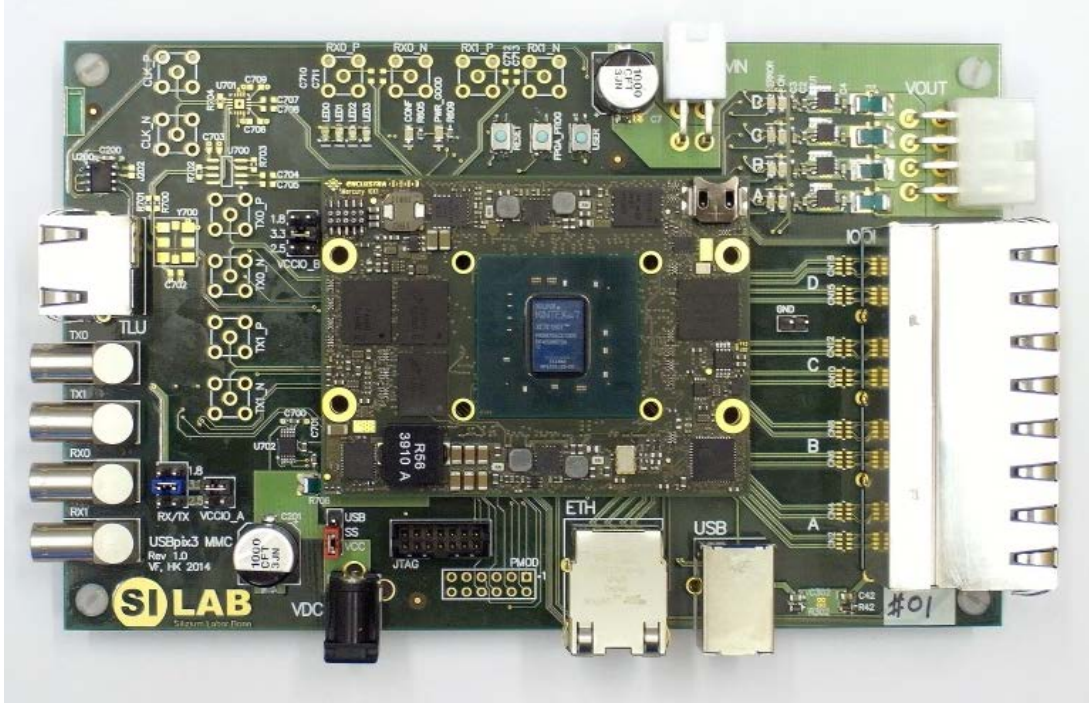


Figure 5.3.: The MMC3 with eight RJ45 ports, USB 3.0 and connectors for power, TLU and trigger.

5.3. USBpix 3

The *USBpix 3* boards are the next generation of USB-based test and read-out systems for ATLAS FEs. They offer higher transfer rates over USB 3.0 to the PC and significantly more powerful hardware on the board. There are two new USBpix systems: the MIO3 (Multi-IO 3), which is similar to the MIO board and uses the same 100-pin KEL-connector to attach to add-on cards, and the MMC3 (Multi-chip Module Card 3), which hosts eight RJ45 ports and can be directly connected to the FEs. The MMC3 is shown in Figure 5.3.

Both USBpix 3 boards are built around the Mercury KX1 FPGA module manufactured by Enclustra. It contains a Xilinx Kintex-7 FPGA, a Cypress FX3 USB 3 microcontroller, 1024 MiB + 256 MiB DDR3 SDRAM, 32 MiB Flash, and Gigabit Ethernet as well as 8 differential analog input channels [26]. The microcontroller handles the communication between the PC and the board, while the FPGA handles the communication with the FE and stores data internally until it is read-out. The FPGA module is mounted on the USBpix board, which provides power to the module and the connectors for USB, RJ45 for connecting the FEs and the TLU, four LEMO ports, Ethernet, a JTAG header for programming and debugging the FPGA, and several supply connectors.

Table 5.1.: Comparison of USBpix and USBpix 3.

	USBpix	USBpix 3
theoretical transfer rate (USB)	60 MB/s	500 MB/s
sustained transfer rate	≈ 15 MB/s	≈ 200 MB/s
number of FEs	up to 4	up to 8

5.3.1. Comparison of USBpix and USBpix 3

The biggest differences between both systems are the upgraded microcontroller and FPGA. The Xilinx Spartan-3 FPGA (XC3S1000) is replaced by a Xilinx Kintex-7 FPGA (XC7K160T) and the Cypress FX2 USB microcontroller (CY7C68013A) is replaced by a Cypress FX3 USB microcontroller (CYUSB3014).

Both microcontrollers consist of a generic microprocessor coupled to special-purpose hardware for USB support. The FX2 is equipped with an 8-bit processor based on the 8051 architecture running at up to 48 MHz and is USB 2.0 compatible [27]. It is connected to the FPGA over two 8-bit data buses for slow and fast transfers and a 16-bit address bus as well as several control signals. In contrast, the FX3 houses a 32-bit ARM microprocessor running at 200 MHz and supports USB 3.0 [28]. The FPGA is connected over a single 32-bit data bus operating with up to 100 MHz.

The Kintex-7 FPGA has 162,240 logic cells while the Spartan-3 only has 17,280 [29, 30]. While these values might not be directly comparable due to the different architectures of both FPGAs, it still serves as an indication of the increase in resources. The Kintex-7 also has additional dedicated hardware for arithmetic, 11,700 Kibit Block RAM compared to the 432 Kibit of the Spartan-3, and several multi-gigabit transceivers. Table 5.1 summarises the differences between both USBpix systems.

The USBpix 3 system has vastly more powerful hardware and still has reserves for subsequent generations of frontend chips. The increased bandwidth allows to operate several FE-I4 in parallel without saturating the connection and is indispensable for future upgrades, where a bandwidth in the order of several Gbit/s is required, far surpassing the capabilities of the USB 2.0 system.

6. USBpix 3 and STcontrol

To add support for the USBpix 3 boards to STcontrol, the differences in hardware and firmware to the Multi-IO board have to be taken into account and knowledge of the internal structure of STcontrol and the various components of the USBpix system is required.

6.1. STcontrol

6.1.1. Software Structure

STcontrol has three software layers, which are the graphical user interfaces of the STcontrol applications, the intermediate *PixLib* library, and the low-level hardware specific libraries. *PixLib* builds on the hardware specific libraries to implement the functionality required to perform measurements, including the configuration and control of the connected hardware and frontends, and data readout. It uses abstract classes to hide implementation details and provide a common interface for all supported systems. This is shown in Figure 6.1. Of particular interest is the *PixController* class, which is the abstract class that provides support for the Multi-IO board with the FE-I3 in form of the *USBI3PixController* class, and for the Multi-IO board with the FE-I4 in form of the *USBpixController* class.

The hardware specific libraries are the lowest layer of the STcontrol software stack and interface directly with the hardware. This is the *SiLibUSB* library, which implements the USBpix protocol to communicate via USB with the hardware, and the *USBpixI3dll* library and *USBpixI4dll* library, which control the Multi-IO board and the connected FE-I3 or FE-I4.

On the hardware side, both the USB-microcontroller and the FPGA are programmable. The USB-microcontroller performs the communication between the PC and the FPGA over the USBpix protocol, and is usually operated with a fixed firmware stored in non-volatile memory. This is necessary to allow the operation of the hardware without special programming devices or drivers. The FPGA on the other hand requires the loading of a firmware every time the board is initialised. The FPGA firmware is also very specific to

6. USBpix 3 and STcontrol

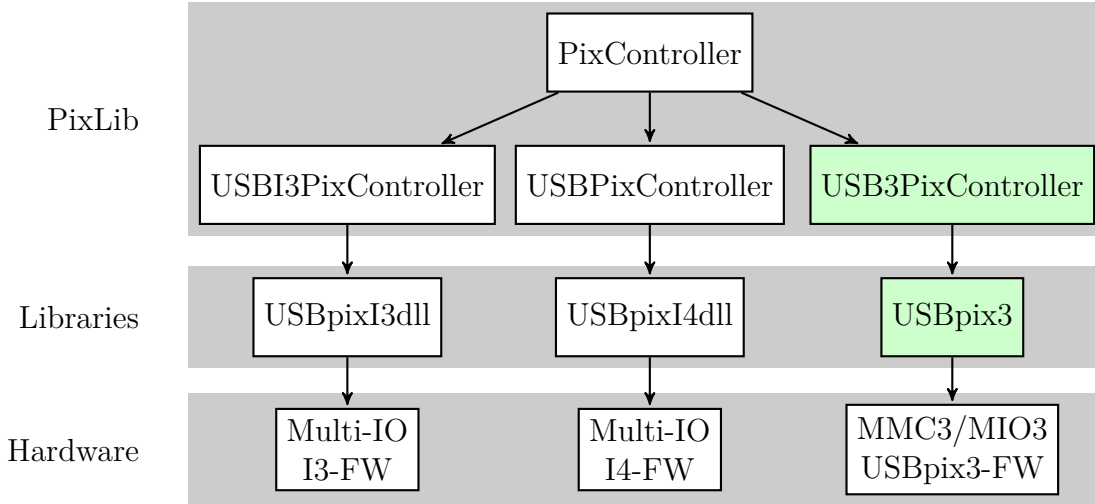


Figure 6.1.: Lower layers of STcontrol and the proposed new components to support USBpix 3.

the hardware setup and different versions are required for the FE-I3 and the FE-I4, which are all provided by STcontrol.

6.2. FPGA Firmware

6.2.1. Firmware Considerations

Integrating USBpix 3 into STcontrol could be done in two disjunct ways: adapting the STcontrol FPGA firmware used for the Multi-IO board to the new hardware, or using the existing PyBAR FPGA firmware for the USBpix 3 boards and adapting STcontrol. Both approaches have their advantages and disadvantages.

Adapting the existing STcontrol firmware would keep compatibility with USBpixI4dll and require relatively few changes to STcontrol. On the other hand, extensive changes to the firmware would be required. The interface between the USB microcontroller and the FPGA was changed substantially to meet the high bandwidth requirements, and the STcontrol firmware is tightly coupled to the external memory interface, which would either have to be upgraded to support the external DDR3 memory or be replaced with the internal but smaller memory of the FPGA, which is too small to support the buffering and onboard histogramming in its current form, especially for eight instead of four frontends.

Using the PyBAR firmware would require no changes to the firmware itself, but USBpixI4dll was written specifically for the STcontrol firmware, therefore requiring a new library and PixController. The PyBAR software can already operate the USBpix 3 boards, which makes evaluating the PyBAR firmware straightforward and allows to incre-

mentally implement support for the firmware and verify it by using PyBAR as reference. It also has a very modular architecture and supports the new high-bandwidth interface between the USB microcontroller and the FPGA.

The achievable performance is important and comparative measurements show that the PyBAR firmware on the Multi-IO board, which uses the same basic firmware blocks as on the USBpix 3 boards, accomplishes significantly shorter scan durations [31]. The PyBAR firmware has the added benefit of continuous readout, while the STcontrol firmware buffers everything and stops readout during transfer to the PC, which reduces the effective bandwidth. The STcontrol firmware also performs histogramming in hardware, which in principle unburdens the computer, but is limited by the available memory on the hardware and makes the analysis less flexible, as the raw data of strobe scans is discarded after processing and never transferred to the PC.

Overall, the PyBAR firmware promises improved performance, the modular architecture makes interfacing with the firmware relatively simple, and the correct operation can be verified using the existing PyBAR system. This makes the PyBAR firmware an obvious choice for adding USBpix 3 support to STcontrol.

6.2.2. PyBAR FPGA Firmware

The PyBAR FPGA firmware consists of several functional blocks from the basil framework which are tied together by a module that is specific to the USBpix 3 board. The firmware is written in the hardware description language Verilog.

Figure 6.2 shows the overall structure as well as the flow of commands to the FEs and the flow of data back to the host.

FEI4 Command Sequencer

The *Command Sequencer* (CMD) allows to control all connected FE-I4 and can be used to perform all necessary actions to operate the front ends, such as configuration, register access and triggering. Aside from its control registers, the *Command Sequencer* contains about 2 KiB of *Command Memory*, which can store an arbitrary sequence of FE-I4 commands as described in Chapter 4.3.2. Upon activation of the sequencer, the commands stored in the memory are transmitted over the serial link to all connected FE-I4. The transmitted sequence can be precisely controlled and automatically repeated. This allows to execute long command sequences without any intervention or overhead from the host software, which is essential for writing fast and efficient scan loops.

Together with the *TLU Controller*, which is described later, trigger commands can be

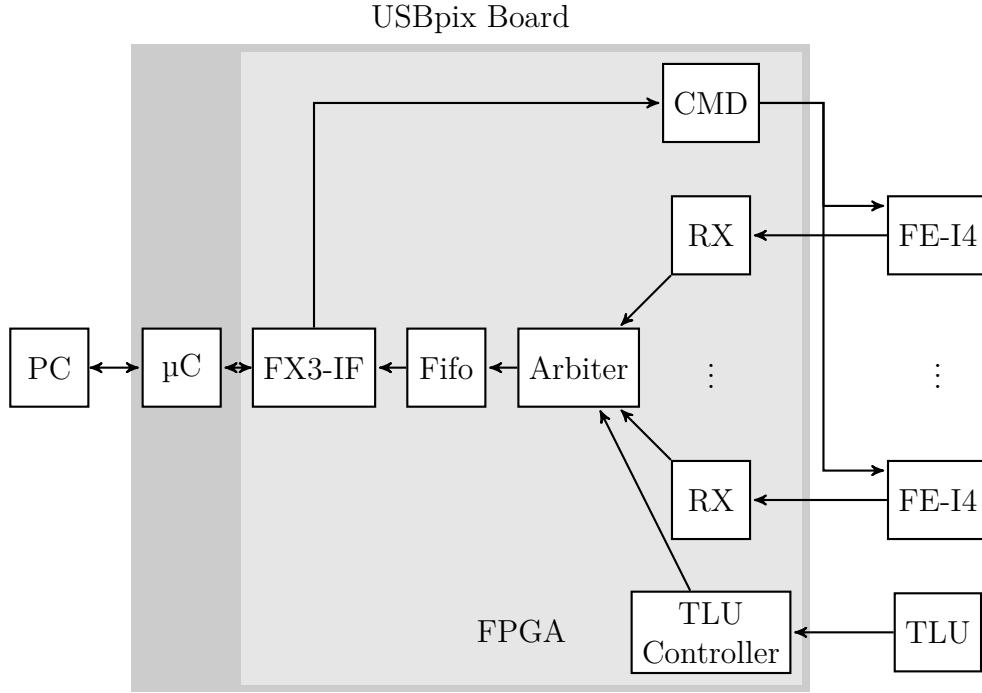


Figure 6.2.: Structure of the PyBAR FPGA firmware on the MMC3. The arrows indicate the flow of commands to the FE-I4 and the data returned to the host.

generated from external sources.

FEI4 Receiver

The *FE-I4 Receiver* (RX) is connected to the data output of an FE-I4. It decodes the received data, performs clock recovery and frame alignment, and detects transmission errors. Received data are buffered and forwarded to the *Arbiter*.

Arbiter

The *Arbiter* is situated between the receivers and the *FIFO* buffer and combines the data streams from the individual frontends into a single stream for readout. The FE-I4 records are preceded with an additional byte to identify the corresponding frontend. This makes all entries 32-bit wide, which is a convenient size for further processing and directly maps the 32-bit wide bus to the microcontroller.

When running with the TLU, the *Arbiter* also inserts the trigger numbers into the stream and ensures that they always precede the associated records.

FIFO

The *FIFO* (First In, First Out) buffers the data stream from the *Arbiter* in a 1 MiB large region of the internal BRAM (Block RAM) of the FPGA. It is visible to the host software as one continuous block of memory which is accessed over the fast 32-bit wide interface. Reading from the *FIFO* automatically removes the read data.

TLU Controller

The *TLU Controller* receives triggers from the LEMO or the RJ45 ports and issues trigger commands using the Command Sequencer.

FX3 Interface

The *FX3 Interface* (FX3-IF) connects to the Cypress FX3 USB 3.0 controller over a 32-bit wide bus and several control lines, decodes incoming read and write requests and forwards them over an internal bus to the other parts of the FPGA firmware.

GPIO

The *GPIO* (General-purpose input/output) provides channels which can be configured as input or output and can be read in the case of an input or written in the case of an output. The channels are connected to other parts of the firmware, for example to control individual channels of the *Command Sequencer*, or to external hardware such as the *Burn-In Card* to enable or disable the voltage regulators and transceivers or to read the overcurrent indicators.

6.2.3. Software Interface

The USBpix 3 board is accessed over the USBpix Protocol as described in Chapter 6.3. Only a small subset of the commands are actually used and in particular, only the *SUR_TYPE_GPIFBLOCK* command is available to access the FPGA. From the host's point of view, the FPGA is visible as a continuous block of memory which contains the control and status registers of the previously described firmware blocks. The memory layout for the MMC3 board is shown in Figure 6.3. The address range is divided into two halves: the region from 0x00000000 to 0x7FFFFFFF mainly holds the control and status registers and is byte-accessible, while the region from 0x80000000 to 0xFFFFFFFF contains the FIFO buffer and is accessed over the fast 32-bit interface.

6. USBpix 3 and STcontrol

0xFFFF FFFF	-	} 32-bit interface
0x800F FFFF	FIFO Data	
0x8000 0000		
0x7FFF FFFF	-	
0x0000 96FF	RX5	} 8-bit interface
	⋮	
0x0000 9300	RX8	
	-	
0x0000 8AFF	GPIO	
0x0000 8900		
	-	
0x0000 86FF	RX1	
	⋮	
0x0000 8300	RX4	
0x0000 82FF	TLU	
0x0000 8200		
0x0000 81FF	FIFO	
0x0000 8100	-	
0x0000 7FFF		
0x0000 0000	CMD	

Figure 6.3.: Memory layout of the FPGA firmware on the MMC3.

As an example, Figure 6.4 shows the status and control registers of the FE-I4 receiver block. Every firmware block has a *Reset/Version* register, which reinitialises the block on a write access and returns a version number on a read access. The *RX Reset* register reinitialises the receiver, the *Ready* register indicates that the receiver is synchronised to the FE-I4. Enabling *Invert RX* negates all received bits before decoding. *FIFO Size* is the number of records which have been buffered but not yet read by the Arbiter. The *Decoder Error Counter* counts 8b/10b decoding errors, while the *Lost Data Counter* counts records lost as a result of a completely filled FIFO.

Registers are between 1-bit and 32-bit wide. Due to the byte-wise memory access, this requires a read-modify-write cycle when writing small or unaligned registers.

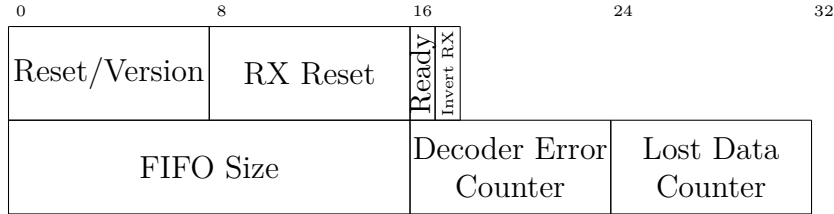


Figure 6.4.: Register layout of the FE-I4 receiver.

6.3. USBpix Protocol

The PC and the USBpix board communicate over USB using the *USBpix Protocol*. The commands are executed by the USB-capable microcontroller, which implements several *USB endpoints* for different purposes. USB endpoints are the source or sink of independent data streams in the communication between host and device. A connection from the host to an endpoint is called a *pipe*. They provide the abstraction of independent channels with different operation modes and transfer speeds. There are four types of transfers, namely the *Control Transfer*, which is used to configure a device, the *Bulk Transfer*, which can transmit large bursts of data but has no guarantees on bandwidth or latency, the *Interrupt transfer* with bounded latency, and the *Isochronous Transfer* with guaranteed constant data rate and bounded latency [32].

USB endpoints are identified by a number and have a specified transfer direction. The USBpix system only provides bulk endpoints and the control endpoint mandated by the USB standard:

- 0. Default control endpoint (In/Out)
- 1. USBPix communication (In/Out)
- 2. GPIF (Out)
- 6. GPIF (In)

The GPIF (General Purpose Interface) is a flexible 8-bit or 16-bit parallel interface driven by a user-programmable finite state machine [27]. It is a feature specific to the USB microcontroller, can be used to implement arbitrary parallel interfaces and can transfer data without interaction from the microcontroller core, which unburdens the controller and increases the transfer speed. In case of the USBpix 3 boards, the 32-bit GPIF II is used instead, but this difference is mostly transparent to the software.

The USBpix protocol uses a 10-byte header and an optional data payload as shown in Figure 6.5. The *Type* field indicates the kind of command that is executed. The second

6. USBpix 3 and STcontrol

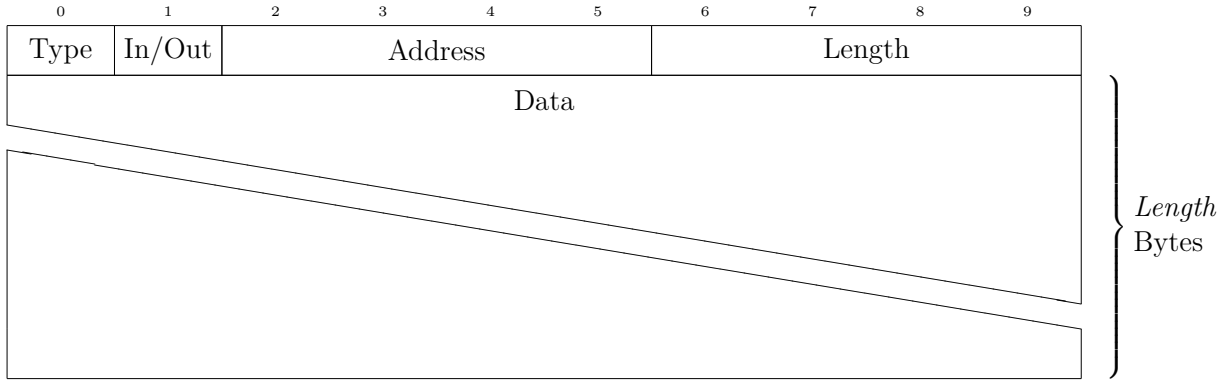


Figure 6.5.: Structure of the USBpix protocol header.

field is either zero for a write access or one for a read access. *Address* and *Length* are used to address memory regions for the transfer. Requests are always initiated from the host by sending the header with a bulk transfer to endpoint 1, followed by the payload to endpoint 1 or 2, depending on the type of the request.

The commands used by USBpix are:

SUR_TYPE_8051 Accesses the hardware registers of the 8051 USB microcontroller.

SUR_TYPE_XILINX Accesses the FPGA without incrementing the address after each byte. This is used to upload the firmware to the FPGA.

SUR_TYPE_EXTERNAL Accesses the FPGA and increments the address after each byte. This is used to read and write registers of the FPGA.

SUR_TYPE_I2C Accesses devices connected to the I²C (Inter-Integrated Circuit) bus of the microcontroller.

SUR_TYPE_EEPROM Accesses the external 24LC128 EEPROM (Electrically Erasable Programmable Read-Only Memory).

SUR_TYPE_FWVER Returns the microcontroller firmware version.

SUR_TYPE_GPIFBLOCK Fast transfer between the host and the FPGA without interaction of the microcontroller core via the GPIF. The request is transmitted to the regular endpoint, but the payload is transferred via the fast GPIF endpoints.

6.4. USBpix 3 Integration

To add support for the USBpix 3 boards to STcontrol, two new software components are developed. Both modules are written from scratch and aim to be easily maintainable by being small and modular and using many new features from the C++11 standard, including smart pointers and STL containers for memory management, as well as new multithreading and synchronisation facilities.

6.4.1. USBpix3 Library

The new *USBpix3* library implements similar functionality for the USBpix 3 boards as does the *USBpixI4dll* for the Multi-IO board, but due to the very different structure of the FPGA firmware and the monolithic architecture of the *USBpixI4dll*, no code is shared between both libraries and the *USBpix3* library is developed from the ground up for the new system.

The USBpix3 library can be regarded as providing three separate categories of functionality, namely the hardware interface, which is the lowest layer of the library and allows access to the FPGA firmware over USB, the control interface, which controls the connected frontends and provides a higher-level interface to the hardware, and the readout interface, which receives and decodes the raw data and performs histogramming.

Hardware Interface

The *USB* class implements the low-level communication with the hardware over the USBpix protocol described in Chapter 6.3. Each instance of the class corresponds to one connected board and allows read and write access to the FPGA firmware.

The *FwDevice* class builds on top of the USB interface and describes the abstract concept of an FPGA firmware block consisting of several registers. It translates read and write accesses to registers of the firmware to accesses to the FPGA memory map. For each firmware block described in Section 6.2.2, a new class is derived from the *FwDevice* class, which is equipped with a list of the available registers and functions specific to the block. This includes the *CmdSeq* class, which controls the command sequencer and has functions to fill the FPGA command buffer and transfer it to the connected frontends, the *FEI4rx* class, which initialises and controls the data receivers, the *Fifo* class, which accesses the data buffer on the FPGA and retrieves the raw data received from the frontends, the *GPIO* class, which enables or disables the channels of the Burn-In Card, and the *TLU* class, which selects an external trigger source and enables triggering of the command sequencer. Upon initialisation, each class verifies the version of the firmware block and

6. USBpix 3 and STcontrol

reports an error if it does not match. This ensures that the FPGA is programmed with the right firmware.

Control

All classes derived from the *FwDevice* class are encapsulated by the *Board* class, which corresponds to a USBpix 3 board connected to the computer. It coordinates the initialisation of the FPGA firmware blocks, provides functions to transmit commands to the frontends and read the results, and selects the trigger mode.

The *CommandGenerator* class implements the FE-I4 specific commands described in 4.3.2. It generates the corresponding bit patterns and stores them in an instance of the *Command* class. Several instances can be concatenated to generate command sequences with bit-exact granularity as required for the correct timing of injection and trigger sequences. The *CommandBuffer* class is a more generic approach to storing and executing command sequences, when the exact timing is not important. It stores an arbitrarily long list of commands which, upon execution, is split into chunks according to the available command memory on the FPGA.

The *Frontend* class provides functions to control the connected frontends, building on the *CommandGenerator* and *CommandBuffer* classes. Each instance corresponds to one frontend and the type of frontend, either FE-I4A or FE-I4B, is specified upon creation. It holds a copy of all global registers and pixel registers and provides the mapping between the logical and the physical layout of the registers. All accesses to registers are performed on the copy and when required, the changes can be written into a *CommandBuffer*. This allows to keep the number of written registers and therefore the number of generated commands to a minimum by condensing several writes to logical registers into one write of a physical register. For example, eight of the logical registers controlling the FE-I4 pulse generator share the same physical register.

Building on the ability to manipulate frontend registers and generate command sequences, several higher-level operations were implemented. This includes functions to read and write the pixel registers, which consists of disabling the pulse generator, selecting the right double-column and the right address mode, writing the desired bit pattern into the shift register, latching the shift register into the pixel register, and restoring all global registers to their previous values.

The *Frontend* class also tries to hide the differences between the FE-I4A and FE-I4B. On the lowest level, both differ slightly in the available registers and some of them have different behaviour, but the higher-level functions take this into account and work equally on both types.

The *RegisterProxy* class is a helper class used by the *FwDevice* and *Frontend* classes to provide transparent access to hardware registers in a way that behaves like a regular variable and hides the access mechanism. This is achieved by overloading an assignment operator for write access and a conversion operator to integer for read access. The operations are then delegated to the respective read and write functions.

Readout

The *Readout* class realises continuous readout by invoking a readout function in fixed intervals in a separate thread. Upon creation of a class instance, the readout function is specified and the thread is in a paused state. The readout thread can then be resumed, paused again, or completely stopped.

To decode the received data, the *DataDecoder* class implements the record format described in Chapter 4.3.3. Each record is decoded, checked for validity and passed to the *Histogrammers*, which can process all received records or records from individual channels. The *Histogrammer* class is a virtual class where every type of record has a function that adds it to the histogram. All kinds of histogrammers can be created by deriving a new class from the *Histogrammer* class and overriding the functions. Two general template classes, the *Histogrammer1D* and *Histogrammer2D* classes cover most of the required cases. Being implemented as template classes allows to specify the size of the histograms as needed.

Derived from the *Histogrammer1D* class are the *BCIDHistogrammer*, where the BCID counter indicates the number of 40 MHz pulses received up to the given event, the *LV1IDHistogrammer*, where the LV1ID counts the number of received LV1 trigger commands, the *LVL1Histogrammer*, which records the delay between the LV1 trigger command and the registration of the event, and the *SrHistogrammer*, which decodes service records. The *HitHistogrammer* is derived from the *Histogrammer2D* class and generates occupancy histograms, where each entry indicates the number of recorded hit for an individual pixel.

Other histogrammer classes include the *ToTHistogrammer2D*, which creates 16 two-dimensional ToT histograms, each for one specific ToT value over all pixels. The *RegisterDecoder*, *PixelRegisterDecoder* and *RawFileWriter* classes create no actual histograms, but nevertheless use the same interface. The *RegisterDecoder* and *PixelRegisterDecoder* classes decode global register values or pixel register values returned by the frontend and create a list of the register addresses and values.

The *RawFileWriter* is mainly used for source scans, but generally works for all scan modes. It converts the records into the plaintext format used by STcontrol and writes

6. USBpix 3 and STcontrol

them to a file. The records are grouped and buffered by channel and written in chunks to the file. Trigger numbers are inserted for all channels.

6.4.2. USB3PixController

The *USB3PixController* is derived from the generic *PixController* class in *PixLib*. It uses the *USBpix3* library to implement similar functionality for the USBpix 3 boards, as the *USBPixController* class does for the Multi-IO board using the *USBpix4Idll* library. Due to the polymorphic properties of the *PixController* classes, all hardware-specific details are hidden from the higher software layers of *STcontrol* and all required changes are confined to this class.

Hardware Detection and Initialisation

When the *USB3PixController* is instructed to initialise the hardware, a new instance of the *Board* class is created or the existing instance is reused. The connected board is detected over USB and if successful, the FPGA firmware blocks are initialised and settings from the *PixController* configuration are written.

Frontend Configuration

To properly initialise the frontends, a complete configuration consisting of global register and pixel register values has to be written to the frontend. A *Frontend* instance is created for each frontend, the configuration is extracted from the *PixLib* configuration and stored in the instance, which then generates the appropriate commands to initialise the frontend.

Frontend Autodetect

After configuration, the mapping of the physical channels on the board to the IDs of the connected frontend has to be determined. This could either be achieved by doing the assignment by hand, or by performing an automated detection. To automatically detect the frontends, a read request is transmitted to each frontend ID in the configuration, one at a time, and the response determines the corresponding channel. The detection procedure for a frontend fails when there is no response, for example because the frontend is not properly configured or connected, or when there are multiple responses, which indicates several frontends with the same ID.

The autodetection is implemented using a *Frontend* instance to generate the requests and a *RegisterDecoder* to decode the responses.

Global Register Test

A global register test verifies the operation of the global registers by reading back all registers and comparing them to the expected values. This test verifies the values of the 16 bit registers, which is sufficient to test the frontend itself, but does not find errors in the mapping between global and logical registers, as this is performed in software. During development, the mapping has to be verified separately.

The register values are decoded using a *RegisterDecoder*.

Pixel Register Test

The pixel register test is similar to the global register test, but instead accesses the pixel registers. Again, the correct mapping of the registers has to be verified separately, including the correct addressing of the pixels.

A *PixelRegisterDecoder* is used to extract the register values from the received records.

Strobe Scans

Strobe scans measure properties of the frontend by varying frontend settings and generating injection signals on the frontend instead of relying on external sources.

Strobe scans usually perform several nested loops, in which signals are injected into the analog or digital part of the pixels. The innermost loops are executed directly on the FPGA or in the PixController for performance reasons. The injection loop is processed on the FPGA. It executes a calibration command to inject into the selected pixels, followed by a delay and a trigger command to initiate readout. After an additional delay to limit the injection rate, the procedure is repeated to reach the desired number of injections. The command sequence is generated by the PixController and transferred to the FPGA, where the actual loop is performed. The PixController then loops over all 40 double-columns and selects one at a time for injection. To limit the number of simultaneously injected pixels, a mask is used to select only every n -th pixel. This mask is then shifted and the inner loop repeated to cover all pixels. Finally, the PixController can loop over values of a global register and scan the frontend several times with different settings.

This is the underlying principle of all strobe scans, on top of which STcontrol implements different scans by selecting the injection mechanism, the pixel masks, the loop settings, and the histogramming and analysis of the returned data. Some scans also add additional loops, which are not processed by the PixController.

The USB3PixController has to process the scan configuration, translate scan settings to frontend settings, generate the pixel masks, and implement the inner scan loops as

6. USBpix 3 and STcontrol

well as the readout, decoding and histogramming. Some scan settings correspond directly to frontend registers, mainly regarding trigger and injection settings. The mask settings contain the distance between selected pixels, the step size of the mask shifts, the number of mask shifts, and the pixel registers which should be masked. The mask patterns are calculated from those settings, taking special flags for some scans into account, and written to the frontends during the scan. The scan settings also specify which histograms should be created from the raw data.

The USB3PixController organises scans in a *Scan* structure, which holds details of the currently running scan including a list of *ScanLoop* structures, which hold the *Histogrammer* classes described in Chapter 6.4.1, which are linked to the *DataDecoder* to create histograms for the active scan loop. Upon completion of the scan, the histograms can be requested from the PixController and are extracted for further processing.

The simplest scans are the *Analog Test* and *Digital Test*. With default settings, they inject 200 times into the analog or the digital part of each pixel respectively. The *Digital Test* injects into every sixth pixel and shifts both the mask for digital injection and the mask for enabling readout. The *Analog Test* injects into every third pixel and shifts the masks of the injection capacitors and the readout mask. The hits are then histogrammed per pixel to produce occupancy histograms and each properly working pixel should show 200 hits.

The *Threshold Scan* is an example for a scan that loops over a frontend register. It performs analog scans for *PlsrDAC* values from 0 to 200, which controls the voltage applied to the injection capacitors and therefore the injected charge. The occupancies for each iteration are recorded and fitted per-pixel to an error function, which determines the threshold and the noise.

The *Crosstalk Scan* is an example for a scan that requires special mask processing. It enables every eighth pixel for readout and then injects into the two neighbouring pixels in the column. Similar to the *Threshold Scan*, it loops over *PlsrDAC* and performs the same analysis. The result is then the threshold charge at which one pixel is affected by the injection into the neighbouring pixels.

Source Scans

Source scans are used to read out a frontend with sensor that is placed under a radioactive source, in a particle beam, or another source of radiation.

To read out a hit, a trigger has to be issued. This can either be an external signal on a LEMO connector or on the RJ45 port from a TLU, or the internally generated *frontend selftrigger*, which causes the frontend to execute a trigger command when any of the

enabled pixels register a hit.

Upon receiving a trigger signal from an external source, the *TLU Controller* signals the *Command Sequencer* to transmit the commands stored in the command memory of the FPGA. The commands are generated once by the PixController, taking several settings for trigger timing into account, and transferred to the FPGA. The sequence of commands is a delay to match the trigger to the latency counter of the frontend, followed by the *LV1* trigger command and another delay to limit the trigger rate.

The *frontend selftrigger* is enabled by connecting pixels to the *HitBus* and enabling the *GateHitOr* register. After placing the frontend in *Run Mode*, no further interaction is required until the scan is completed.

The scan configuration for source scans in the USB3PixController is performed identically to the strobe scans, including the translation of scan settings to frontend registers, the mask settings, and the histogramming. The scan also uses the same *Scan* and *Scan-Loop* structures with the same readout, decoding, and histogramming mechanisms. A *RawFileWriter* is added to the list of histogrammers in the *DataDecoder* to write the raw data received during the scan to a file.

Once active, the scan runs until it is cancelled or a predefined condition is met. These are target values for the scan duration, the number of triggers, or the number of received data headers or data records. The USB3PixController checks the condition at regular intervals and terminates the scan.

Differences between MMC3 and MIO3

To support both the MMC3 and the MIO3, some of the differences between both boards have to be taken into account. Firmware-wise, both boards behave nearly identical and the only differences are the four channels on the MIO3 compared to the eight channels on the MMC3, which is reflected by the number of FE-I4 receiver blocks, and the GPIO interface, which enables individual channels on the card connected to the MIO3. A minimal working solution is a switch in the *Board* class to select between both boards and enable all channels of the MIO3 by default.

A proper solution requires additional support for the *Adapter Card*, *Burn In Card* and *General Purpose Analog Card*, because all cards have programmable voltage regulators and other devices connected to the I2C bus of the USB microcontroller. They are not controlled by the FPGA firmware but over the *SUR_TYPE_I2C* request of the USBpix protocol. Only the *Burn In Card* is functional without initialising the regulators. The code of the *USBpixI4dll* could be adapted to add support for the I2C devices, as this component is relatively compact and independent from other parts of the library.

7. Verification and Results

To verify the proper operation of the new USBpix3 library and the new USB3PixController, several tests and measurements are performed, ranging from simple tests of the low level functionality to complex tuning procedures and measurements with radioactive sources. The measurements are performed on both the Multi-IO board and the new MMC3 and the Multi-IO is used as a reference.

Finally, the performance of both systems is compared including scan duration, frontend configuration, and bandwidth.

7.1. Basic Tests

Initialisation and Configuration

After power up, the FE-I4 does not transmit a signal on its data output. To receive any data, several parts of the chip have to be initialised and enabled. This includes enabling the PLL and clocks, selecting the right clock source, and enabling the LVDS output driver. If these steps are performed correctly, the frontend will start transmitting 8b10b encoded data. Figure 7.1 shows the clock input and data output of the FE-I4 after initialisation. It is transmitting the K.28.1 comma which indicates an idle state [18]. The comma is encoded as 00 11111 00 1 or as 11 00000 11 0 with both encodings being alternated to match the parity. This verifies that the frontend has received and executed the commands and that the registers are properly initialised.

Global and Pixel Register Test

A *Global Register Test* and a *Pixel Register Test* are performed to verify the proper reading and writing of the global registers and pixel registers. The registers can be set to the current configuration, all bits to zero or one, or random values. In case of the global registers, extra care has to be taken to keep the frontend in a working configuration and therefore, the overwriting can be limited to smaller subsections of the available registers.

Both tests show that the basic reading and writing of global and pixel registers of the

7. Verification and Results

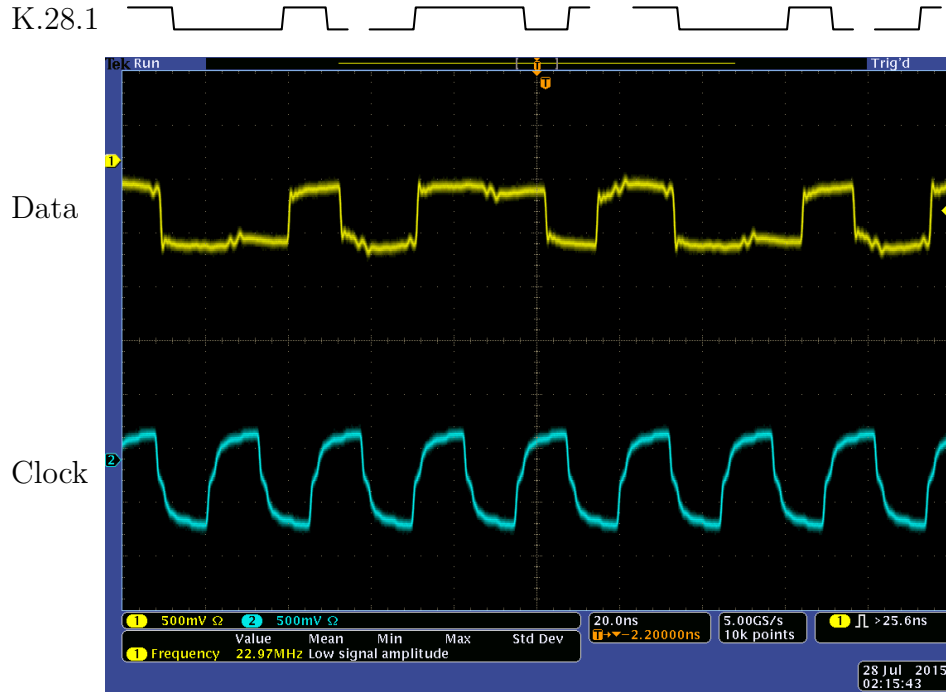


Figure 7.1.: Clock input and data output of the FE-I4 after initialisation.

new PixController is working. The proper addressing of the individual pixels has to be verified separately using analog or digital injection.

7.2. Analog and Digital Test

The digital and analog scans are the simplest strobe scans and allow the verification of the injection mechanism, mask writing, and pixel addressing of the USB3PixController.

The pixel mask can be made visible by decreasing the number of mask iterations so that not every pixel is enabled for injection. This is shown in the occupancy histogram in Figure 7.2a, where only every sixth pixel is enabled. The histogram records the number of registered hits for each individual pixel, which match the expectation of 200 hits for the enabled pixels. Visible is also the double-column wise addressing, which causes the pattern in every second column to be reversed with respect to the other columns.

The pixel addressing can be verified by writing a test pattern into a pixel register and performing a scan. By performing an analog test with a fixed mask for the injection capacitors, the resulting occupancy histogram should match the mask and the reported hits confirm the pixel addressing. This is shown in Figure 7.2b. The mask demonstrates the correct writing of the pixel register, including the correct column and pixel addressing as well as the overall orientation of the frontend.

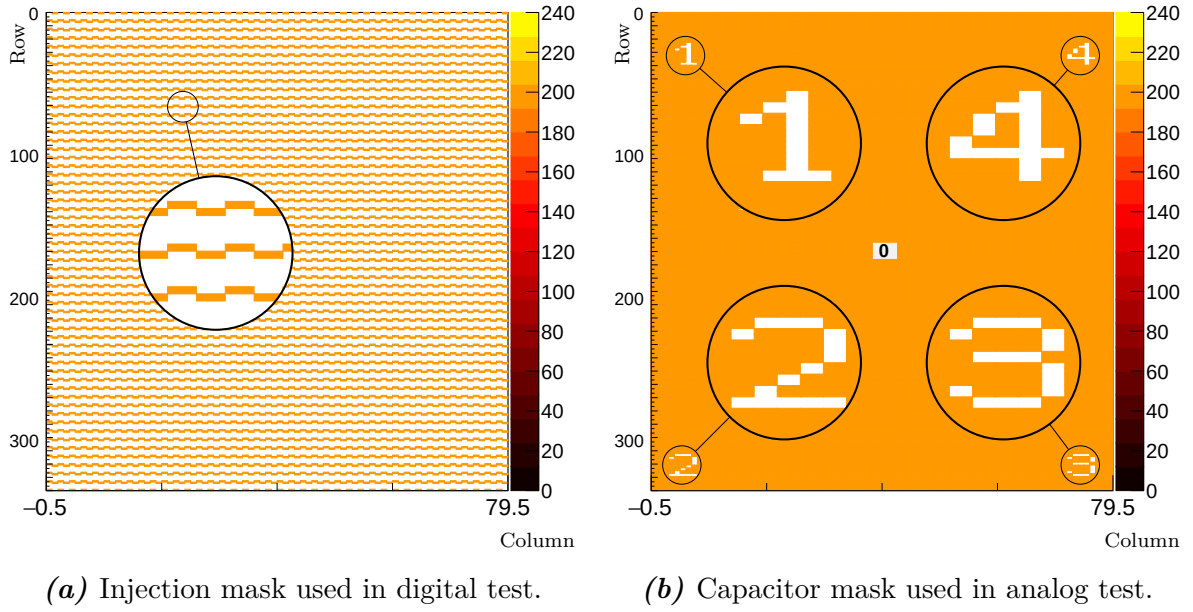


Figure 7.2.: Digital and analog scans to verify the mask writing, injection mechanism, and pixel addressing.

7.3. Tuning

A frontend is tuned to reach uniform threshold and ToT responses for all pixels. This is an important prerequisite for source scans and also serves as an extensive test of the strobe scan implementation of the USB3PixController.

Both threshold and ToT tunings are performed in two steps. The global registers *Vthin_Alt_Coarse* and *Vthin_Alt_Fine* control the threshold settings for all pixels on the frontend. The *GDAC Fast Tune* adjusts these parameters to reach the target threshold value. In the next step, the *TDAC Fast Tune* is used to adjust the local threshold setting of each pixel to compensate deviations from the target value for individual pixels, therefore decreasing the overall threshold dispersion. *GDAC* and *TDAC Fast Tune* loop over their respective registers, inject into all pixels, measure the resulting occupancy, and perform a binary search to determine the register settings for the target threshold. This covers a wide range of functions from the new PixController.

ToT tunings operate in a similar fashion. The *IF Tune* adjusts the global feedback setting *PrmpVbpf*, while the *FDAC Tune* calibrates the separate pixels. For both tunings, the step size of the per-pixel parameter can be set via the *TdacVbp* and *FdacVbp* registers. Choosing those parameters is a tradeoff between the accuracy of the per-pixel corrections and the largest offset that can be compensated. An indication of a suitable step size is the width of the TDAC and FDAC distributions, which should cover the available range.

7. Verification and Results

A smaller distribution indicates too large a step size and suboptimal accuracy, while too small a step size will lead to pixels which can not reach the target value even with the minimal or maximal setting.

A complete tuning is performed by alternating threshold and ToT tunings, because both tunings influence each other, leading to a shifted threshold after a ToT tuning and vice versa. In this case, a threshold tuning, followed by a ToT tuning and another threshold tuning is performed.

All tunings are based on the default STcontrol register settings with the exception of *TdacVbp*, which was lowered from 255 to 100 to achieve lower threshold dispersion. The target threshold value is 3000 e, the target ToT response is a ToT Code of 6 for an injected charge of 20 000 e. The frontend chip used to test the tuning procedure is part of the 66-W10-IBL3 four chip module.

To verify the tuning, the procedure is performed independently on the Multi-IO and the MMC3 with the same FE-I4 and the threshold and ToT response are measured after each step.

7.3.1. Tuning Results

During the measurements, a shift of threshold, ToT, and noise values depending on the temperature of the module is observed. To mitigate the influence of this effect on the results, all measurements are performed after the module has reached a stable temperature. As the module is not actively cooled or otherwise contained, some fluctuations in the temperature still occur. A stable operating temperature is assumed when the leakage current over the sensor has stabilised.

Table 7.1.: Measured threshold before and after warm-up of the module.

	Threshold [e]	Dispersion [e]	Noise [e]
Cold	5593	378	120
Warm	5718	370	305

To estimate the influence of the operating temperature, the threshold and ToT response are measured shortly after turning the module on, and again after equilibrium is reached. This is summarised in Table 7.1 for the untuned configuration. After warming up, the threshold has increased by about 100 e and the noise by about 200 e. The width of the threshold distribution remains unaffected. The following measurements are performed

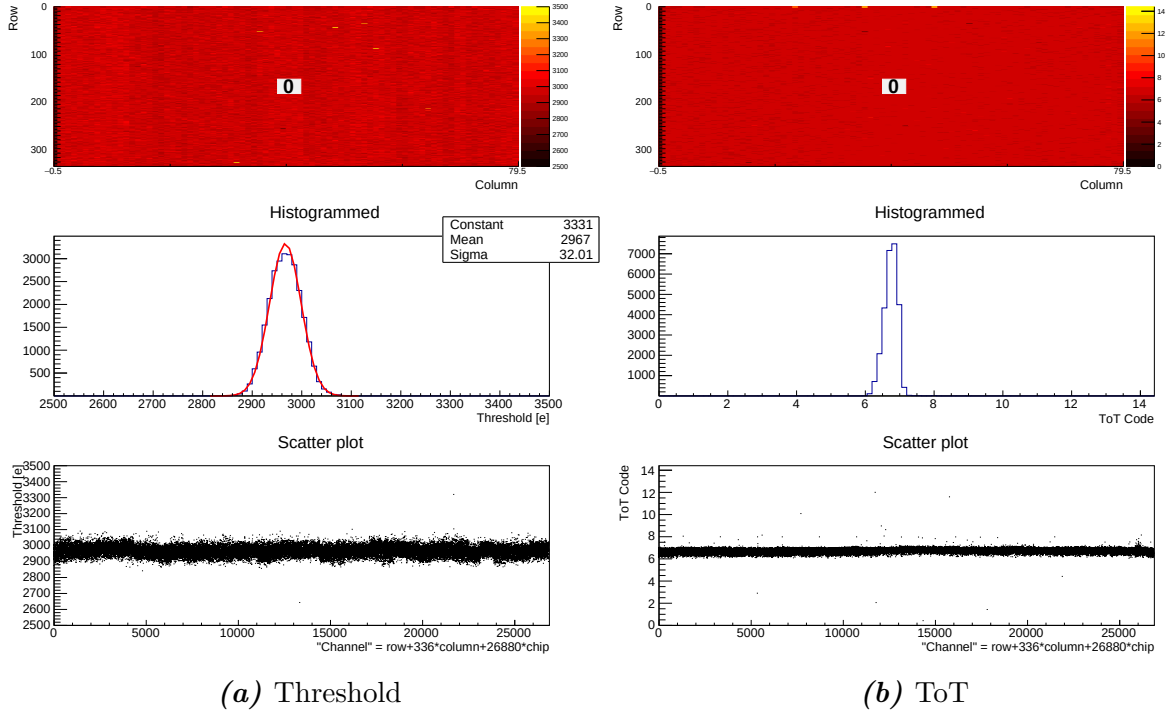


Figure 7.3.: Threshold and ToT after the complete tuning procedure on the MMC3.

after an ample warm-up period, which should keep the threshold and noise relatively stable.

The measured thresholds for both boards after each step of the tuning process are listed in Table 7.2. The threshold distribution and threshold dispersion are plotted in Figure A.1 for the untuned module, in Figure A.3 after the first threshold tuning, in Figure A.5 after threshold and ToT tuning, and in Figure A.7 after the complete tuning procedure. The top of the plots shows a map of the individual pixels with their values, which are also histogrammed and displayed as a scatter plot. This is illustrated in Figure 7.3, which shows the threshold and ToT after tuning the module with the MMC3.

Table 7.2.: Measured threshold after each tuning step for both boards.

Tuning Step	MIO			MMC3		
	Threshold [e]	Dispersion [e]	Noise [e]	Threshold [e]	Dispersion [e]	Noise [e]
Untuned	5682	369	316	5718	370	305
Threshold Tuning	2932	25	331	2978	22	311
Threshold & ToT	4184	224	425	4244	230	397
Full Tuning	2983	29	392	2967	32	405

7. Verification and Results

The average ToT responses and widths are shown in Figure A.2 before the tuning, in Figure A.4 after the first threshold tuning, in Figure A.6 after the threshold and ToT tuning, and in Figure A.8 after the complete tuning. The values are encoded as *ToT Code*, where values above 12 have a special meaning. The time over threshold is measured in 25 ns steps, corresponding to the external 40 MHz clock and bunch crossing rate at the LHC. The global register *HitDiscCnfg* sets the minimal ToT above which a signal is considered as hit. All hits that would have a ToT Code above 12 are encoded with a ToT Code of 13, a ToT below the minimal ToT next to a registered hit is encoded with a ToT Code of 14. With *HitDiscCnfg* set to 0, ToT Code 14 should not appear and ToT Code 0 corresponds to a duration of 25 ns.

For the untuned module, the measured values are determined by the default register settings, which are not aimed at specific threshold and ToT values. All pixel registers are initialised to the default value at the centre of the available range, which does not yet compensate for the individual pixels and leads to relatively wide distributions.

After the first tuning step, the measured threshold is very close to the target value of 3000 e and the dispersion has decreased significantly. The ToT response is also affected by the threshold tuning and is shifted to larger values, giving a ToT Code of 13 for most pixels.

The second tuning step brings the ToT response close to a ToT Code of 6 for the injected charge of 20 000 e, but influences the threshold tuning and increases the threshold by about 1200 e and the dispersion by a factor of ten. The noise is also slightly increased by about 100 e.

The final tuning step brings the threshold again close to the desired value and decreases the dispersion. The ToT response is slightly shifted to values between 6 and 7 and becomes wider, but the deviations are still acceptable.

During the tuning, several pixels close to the right edge of the frontend become visible, which have significantly larger fluctuations in their ToT response than the rest of the module. This effect is visible on both the Multi-IO board and the MMC3, which indicates that there are faulty pixels on the frontend and it is not caused by the software. Overall, the tuning procedure yields very similar results on both boards and verifies the strobe scan implementation of the USB3PixController.

7.3.2. Crosscheck

A further verification is achieved by performing a cross-check between both boards. The tuning obtained from one board is loaded using the other board which is then used to measure the threshold and ToT response. In Table 7.3, the threshold tuning obtained

with the Multi-IO board is compared to the verification measurements on the MMC3. Table 7.4 shows the tuning obtained with the MMC3 and the verification on the Multi-IO board. Both measurements are in very good agreement and the measured thresholds differ by less than one percent.

Table 7.3.: Verification of the tuning obtained with the Multi-IO board with the MMC3.

Tuning Step	Tuning (MIO)			Verification (MMC3)		
	Threshold [e]	Dispersion [e]	Noise [e]	Threshold [e]	Dispersion [e]	Noise [e]
Threshold Tuning	2932	25	331	2943	25	317
Threshold & ToT	4183	226	425	4187	227	399
Full Tuning	2983	29	392	2974	28	395

Table 7.4.: Verification of the tuning obtained with the MMC3 with the Multi-IO board.

Tuning Step	Tuning (MMC3)			Verification (MIO)		
	Threshold [e]	Dispersion [e]	Noise [e]	Threshold [e]	Dispersion [e]	Noise [e]
Threshold Tuning	2978	22	311	2973	22	315
Threshold & ToT	4245	230	397	4248	230	414
Full Tuning	2967	32	405	2974	32	390

The tuning procedure and especially the crosscheck also serves to verify the correct mapping of the pixel registers and the correct addressing of pixels, as both have visible impact on the threshold and ToT distributions.

7.4. Source Scans

Several measurements with radioactive sources are performed with both the Multi-IO and the MMC3 board to verify the source scan implementation of the USB3PixController.

Due to high leakage currents of the 66-W10-IBL3 module in the order of 100 μA and high noise, a different four chip module with a leakage current of below 10 μA at a bias voltage of 100 V is used. Out of the four FE-I4, three are operational and two are used for the measurement. On the module is a 200 μm thick planar silicon sensor. A tuning procedure similar to 7.3 is used. Starting from the default register settings, the frontends are tuned to a target threshold of 3000 e and a ToT response of 10 for an injected charge of 16 000 e.

7. Verification and Results

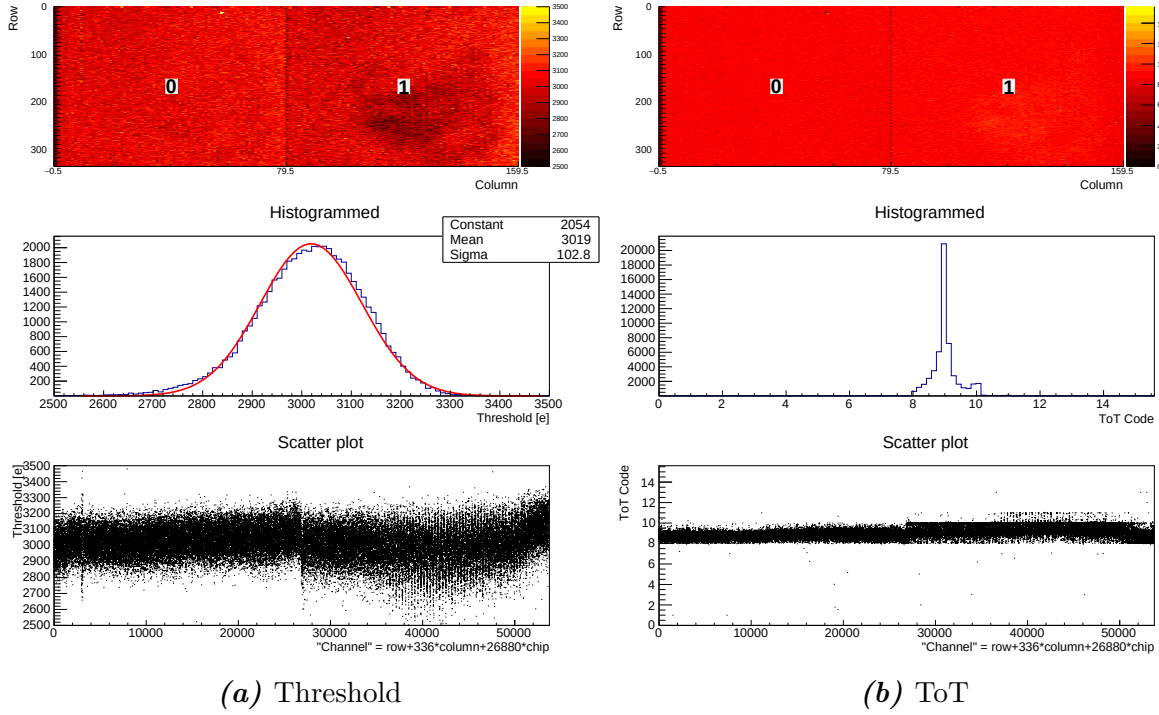


Figure 7.4.: Verification of the threshold and ToT tuning used for the source scans.

The result of the tuning is shown in Figure 7.4. The threshold is at 3019 e with a dispersion of 103 e. The ToT distribution is centred around ToT Code 9, which is slightly below the target of 10 and is again shifted by the threshold tuning.

A mask is used to disable defective or noisy pixels which could otherwise disturb the measurement. The mask is created by combining several scans. In a first step, all pixels that do not show the expected 200 hits during a Digital or Analog test are disabled. Also, pixels that show large amounts of hits during a source scan without source are considered noisy and are disabled. This is repeated until only a few random hits remain which are likely caused by natural sources. Most pixels in the last column of the first frontend and the first column of the second frontend show an excessive amount of hits during a scan with source and the columns are therefore disabled. The sensor pixels for those columns are larger than the regular $50\text{ }\mu\text{m} \times 250\text{ }\mu\text{m}$ pixels, which leads to more hits and also increases the noise. A *Noise Occupancy* and a *Stuck Pixel* scan are performed to identify additional noisy pixels or pixels that keep the *HitBus* constantly active, but no such pixels are observed.

Altogether, 691 of the 53760 pixels are masked, most of them in the two central columns and 19 all across the frontends. This is shown in Figure 7.5.

The recorded data includes the raw data received from the frontends, occupancy histograms, ToT histograms, and LV1 histograms. Also, a clustering of the hits is performed.

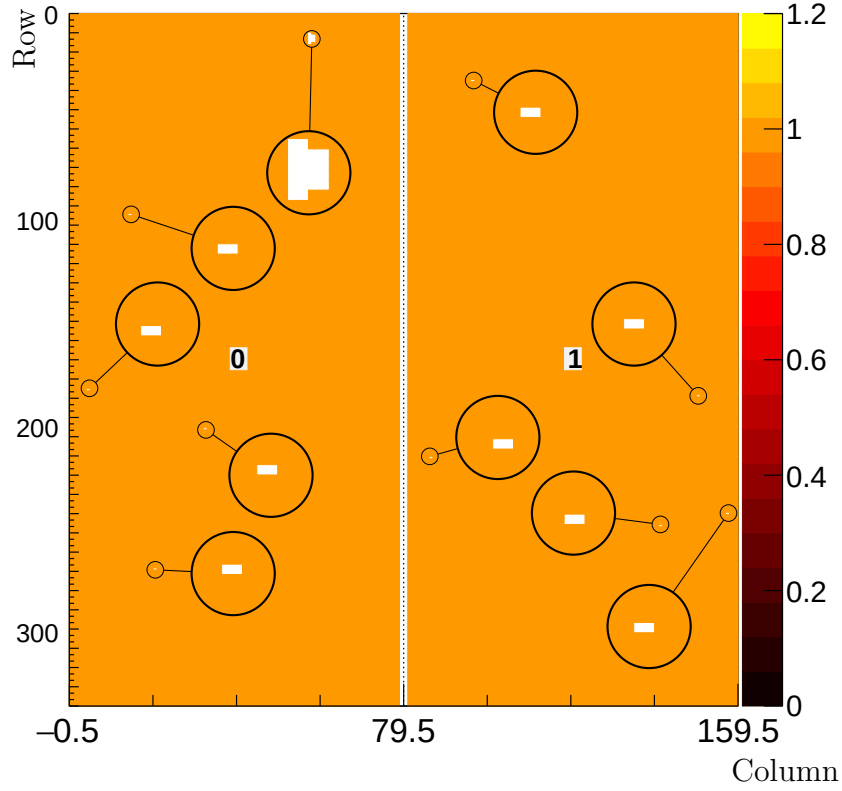


Figure 7.5.: The mask used for all source scans. Disabled pixels are shown in white, enabled pixels in orange.

A hit is considered part of the cluster if it is in a five bunchcrossing wide window and is adjacent to a pixel in the cluster. The ToT values of the hits are summed to determine the total deposited charge.

The measurements are performed with both boards and two radioactive sources and different triggering modes. The four chip module is placed inside a shielded box. Above the module is a metal plate on which a radioactive source can be mounted. The plate has a round hole that is aligned with the two used frontends. Below the module is a scintillator which is used as an external trigger source during the measurement. The sensor is biased with 100 V and the module is cooled with air.

7.4.1. Background Measurement

A first measurement is performed without a source but still inside the containment box using the frontend selftrigger. The results of the scans are summarised in Table 7.5, listing the number of recorded data headers, data records, and hit pixels. Both scans have a duration of about five minutes and a comparable amount of recorded hits. The

7. Verification and Results

Table 7.5.: Amount of data recorded during the background measurement.

	MIO	MMC3
Duration	304 s	303 s
#DH	1 248	1 377
#DR	202	179
Hits	293	240
Hit rate	0.96 Hz	0.79 Hz

deviations can be attributed to statistical fluctuations and the overall low amount of data.

Figure 7.6 shows the occupancy histograms for both boards, in which the locations of the hits are visible. The pixels show at most two hits, while almost all pixels have at most one hit. The hits form clusters or short tracks on the sensor. There is no spatial correlation between the hits on both boards which indicates physical hits from natural sources instead of noisy pixels.

The ToT distributions in Figure 7.7 translate to the charge that is collected in individual pixels for one hit and use the same ToT encoding as the ToT measurements during the tuning. *HitDiscCnfg* is again set to zero, therefore excluding ToT Code 14. Both measurements show a peak at ToT Code 1. The MMC3 shows a broader peak, but the total number of hits is very low and charge sharing between pixels as well as different incident particles or different energies can alter the distribution.

The time difference between a trigger being issued and a hit being registered is recorded in the LV1 histograms in Figure 7.8. Almost all hits are registered at the same time as the trigger. This is expected because the selftrigger issues a trigger for each registered hit. Higher values indicate charge sharing between pixels, where smaller signals take longer to be registered. This effect is called *timewalk* and causes some hits to appear later in the LV1 histogram.

Overall, both source scans are in good agreement and verify the basic operation of the source scan implementation of the new PixController. The observed hits are likely to be caused by cosmic radiation or other natural sources which are not completely shielded by the containment box.

7.4.2. ²⁴¹Am Source

²⁴¹Am decays with a half-life of 432.2 a and an energy of 5.486 MeV (84.8%) or 5.443 MeV (13.1%) into ²³⁷Np. The α -decay is accompanied by the emission of photons with the highest intensities at 13.9 keV and 59.54 keV. The photons are completely absorbed by

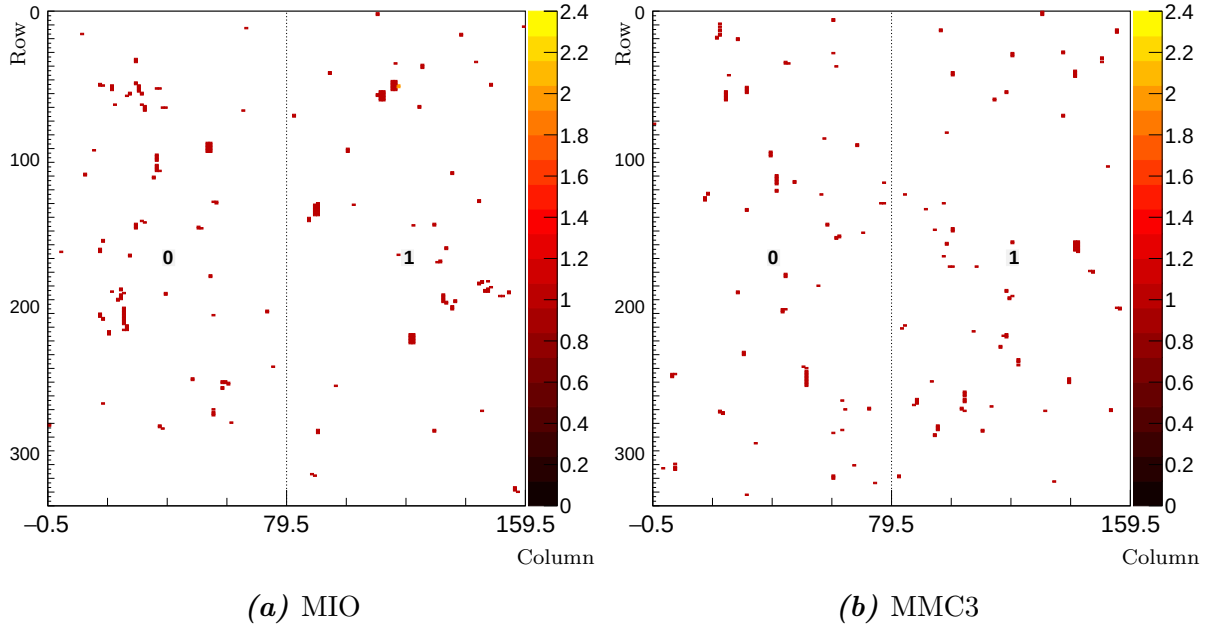


Figure 7.6.: Occupancy histograms showing the number of hits per pixel for a measurement without radioactive source obtained with the Multi-IO board and the MMC3.

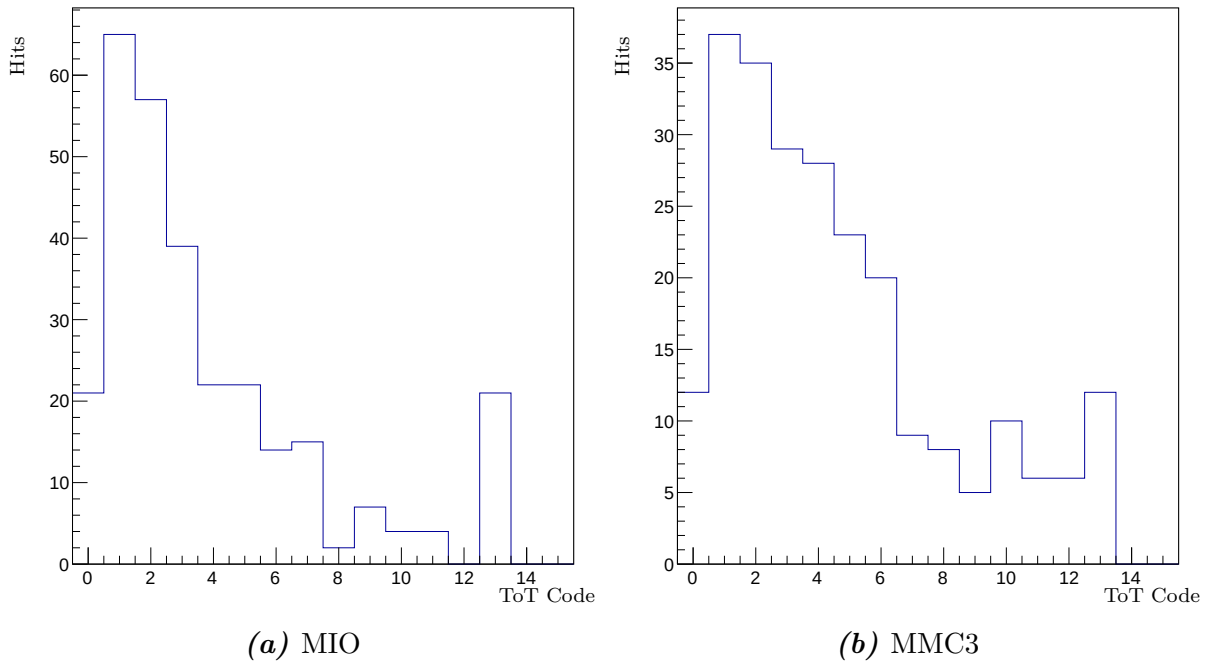


Figure 7.7.: Unclustered ToT histograms showing the charge spectrum for a measurement without radioactive source obtained with the Multi-IO board and the MMC3.

7. Verification and Results

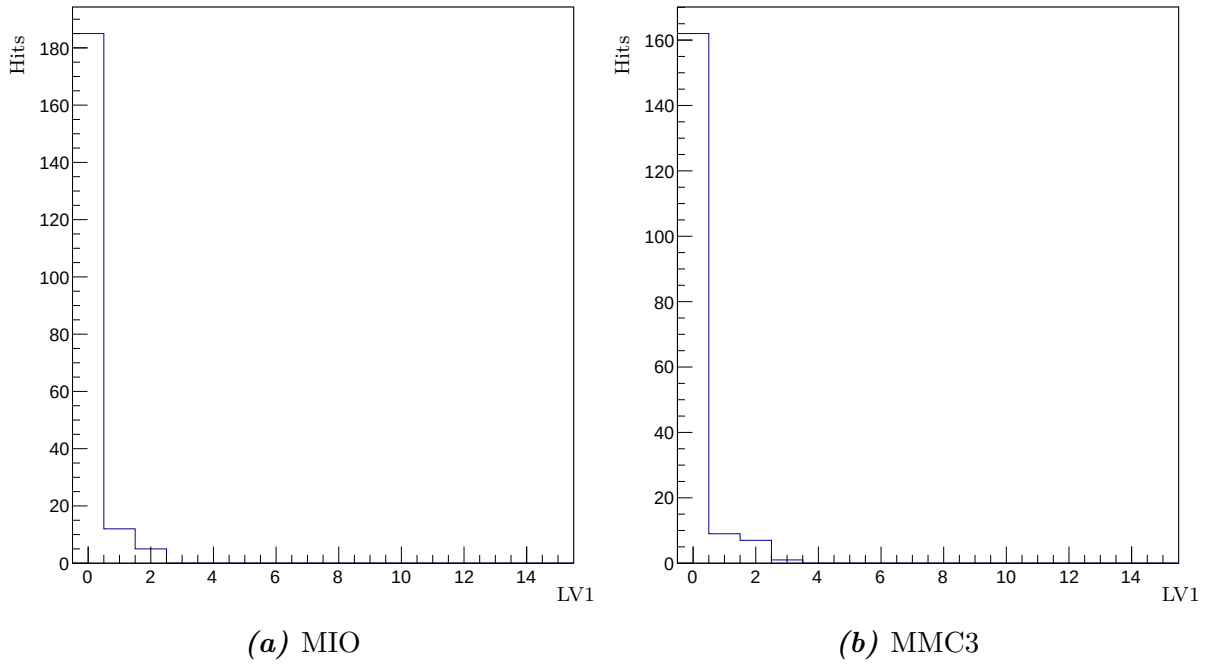


Figure 7.8.: LV1 histograms showing the delay between trigger and hit for a measurement without radioactive source obtained with the Multi-IO board and the MMC3.

the sensor, which allows to compare the measured ToT values to the expected energies, but limits the measurement to the frontend selftrigger, as no photons would hit both the sensor and the scintillator.

The occupancy histograms are shown in Figure 7.9. The position of the source above both frontends is clearly visible in the gradient of the number of hits over the active sensor surface. The results are summarised in Table 7.6. Both measurements have a duration of about 34 minutes. The number of data headers, data records, and hit pixels for the MMC3 is about 7% higher compared to the Multi-IO board. This can be attributed to the stopless readout of the USB3PixController, which does not pause the data taking when transferring data from the board to the PC.

The measured ToT spectrum is shown in Figure 7.10. The shape is identical for both measurements and the two photon energies are visible as peaks around a ToT Code of 2 and 6.

Figure 7.12 shows the clustered ToT spectrum. The spectrum is fitted with the sum of two Gaussians to determine the position and width of the peaks, which are summarised in Table 7.7. Both measurements are in very good agreement. The clustered and unclustered ToT spectra are very similar and the clustered spectrum has a better separation of both peaks. This is expected as the clustering compensates for charge sharing across several

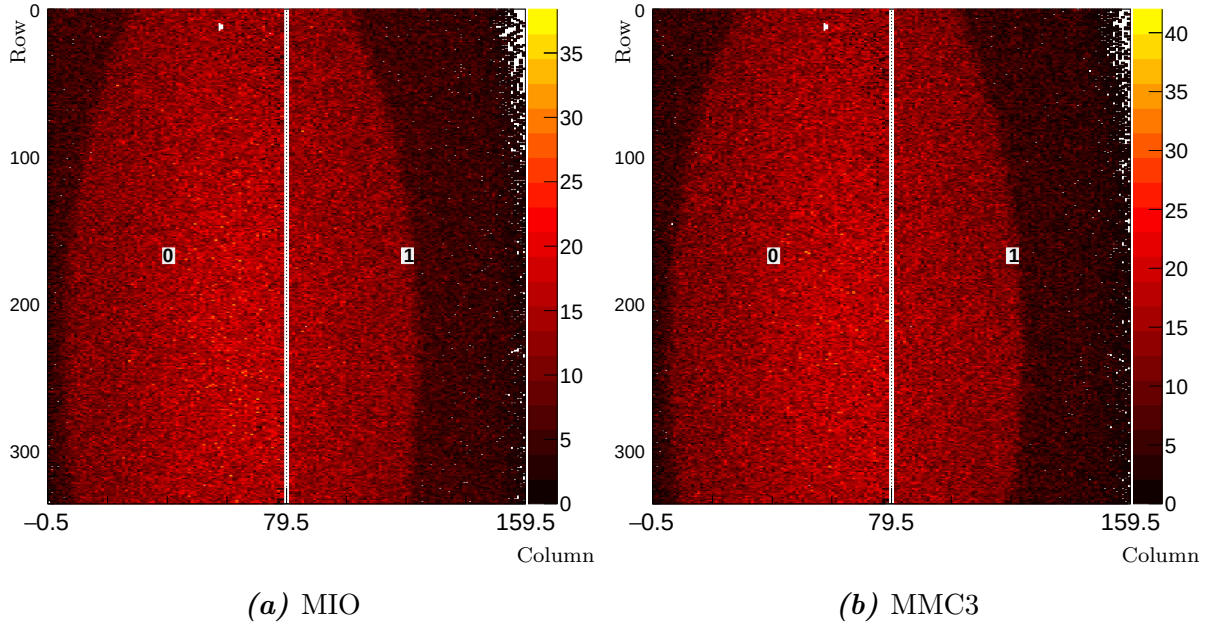


Figure 7.9.: Occupancy histograms showing the number of hits per pixel for the measurement with the ^{241}Am source obtained with the Multi-IO board and the MMC3.

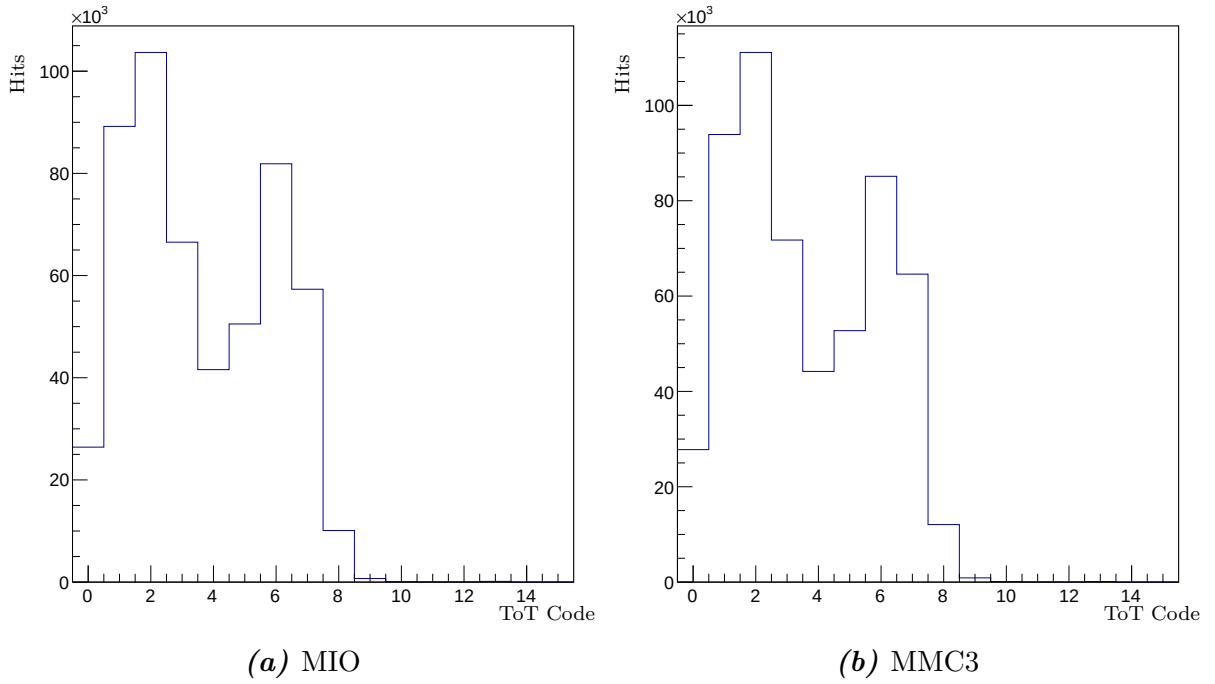


Figure 7.10.: Unclustered ToT histograms showing the charge spectrum for the measurement with the ^{241}Am source obtained with the Multi-IO board and the MMC3.

7. Verification and Results

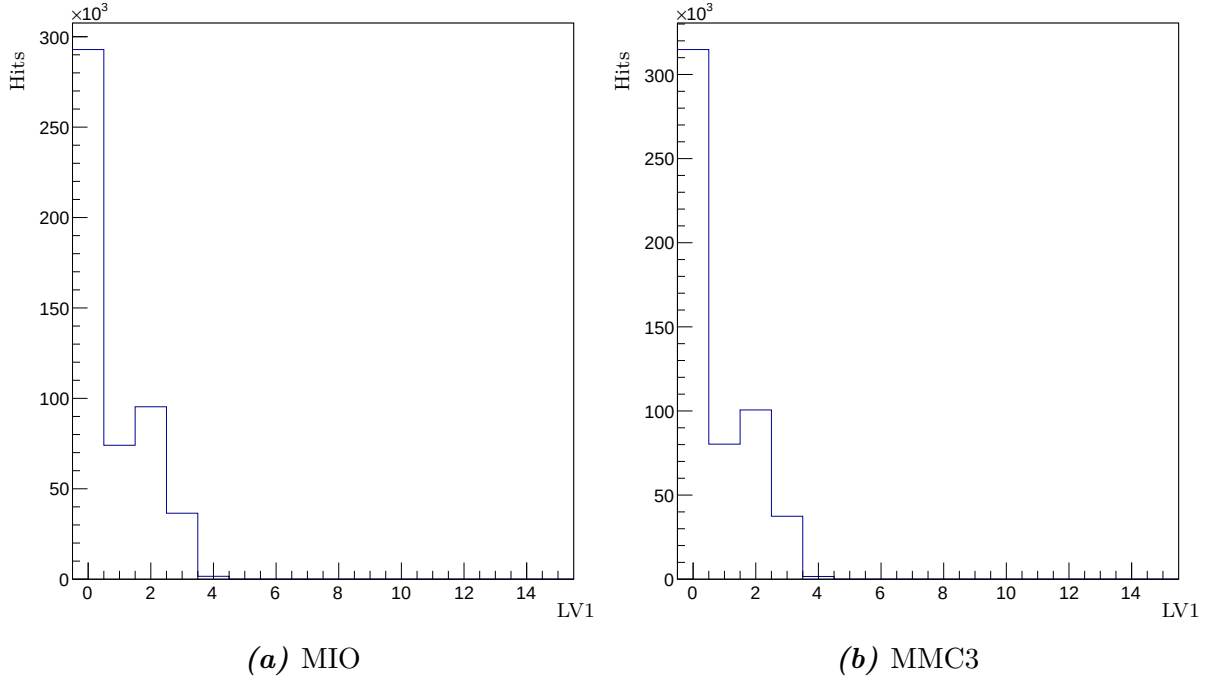


Figure 7.11.: LV1 histograms showing the delay between trigger and hit for the measurement with the ^{241}Am source obtained with the Multi-IO board and the MMC3.

pixels, which is especially the case for the 59.54 keV peak.

A 59.54 keV photon is expected to produce about 16500 electron-hole pairs, close to the ToT tuning target with a ToT Code of 9 for 16 000 e. The measured peak is clearly lower at ToT Code 6.7 with a width of 1.1. This can be attributed to uncertainties on the capacitance of the injection capacitors and on the relation between *PlsrDAC* and the actual injection voltage, which are set to the default STcontrol values and can deviate from the real values. This can lead to a systematic deviation between the calculated injected charge used for the ToT tuning and the actually deposited charge.

Table 7.6.: Amount of data recorded during the measurement with the ^{241}Am source.

	MIO	MMC3
Duration	2037 s	2038 s
#DH	7 672 104	8 199 666
#DR	500 406	534 728
Hits	528 235	564 429
Hit rate	259 Hz	277 Hz

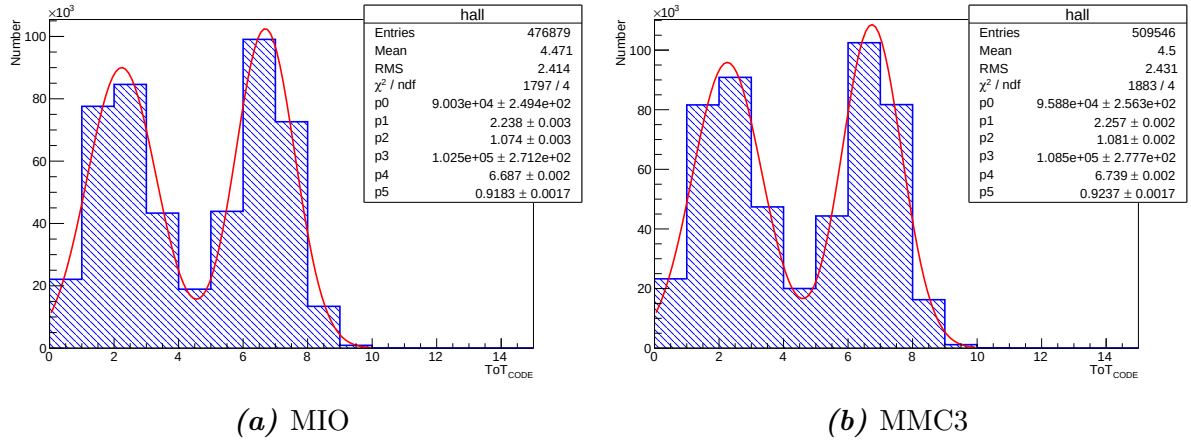


Figure 7.12.: Clustered ToT spectrum for the measurement with the ^{241}Am source obtained with the Multi-IO board and the MMC3.

Table 7.7.: Peaks from the clustered ^{241}Am spectrum. Values are in ToT Code.

Peak	MIO		MMC3	
	Center	Width	Center	Width
13.9 keV	2.2	1.1	2.3	1.1
59.54 keV	6.7	0.9	6.7	0.9

7.4.3. ^{90}Sr Source

^{90}Sr decays with a half-life of 28.79 a and an energy of 546.0 keV into ^{90}Y . ^{90}Y decays with a half-life of 64.00 h and an energy of 2280.1 keV into ^{90}Zr , which is stable. Both are β^- -decays.

The measurement is performed with the scintillator and the frontend selftrigger.

External Trigger

The occupancy histograms are shown in Figure 7.13. The ^{90}Sr source is clearly visible as a spot across both frontends. The number of triggers, data headers, data records, and hit pixels is summarised in Table 7.8. Both measurements have a duration of about 100 minutes. Comparing the Multi-IO and the MMC3, the rates on the MMC3 are 40% to 50% higher, which can again be attributed to the continuous data taking on the MMC3.

The ToT histograms are shown in Figure 7.14. The Multi-IO board shows a small peak around ToT Code 2 and a bigger peak around ToT Code 6. For the MMC3, the peak at 2 is no longer visible and the peak around 6 has widened, especially to lower ToT values. The ToT spectra are again clustered and fitted with the convolution of a

7. Verification and Results

Table 7.8.: Amount of data recorded during the measurement with the ^{90}Sr source and the scintillator.

	MIO	MMC3
Duration	6433 s	6303 s
Trigger	1 032 385	1 445 350
#DH	32 568 561	45 632 672
#DR	810 899	1 083 987
Hits	902 058	1 317 859
Hit rate	140 Hz	209 Hz

Landau distribution with a Gaussian, which gives the position of the peak and is shown in Figure 7.15. The first peak in the spectrum of the Multi-IO board has disappeared and the fitted peaks are in agreement with a most probable value of ToT Code 6.3 and a width of 1.3 for the Multi-IO board and a most probable value of ToT Code 6.0 and a width of 1.2 for the MMC3. The most probable energy loss in 200 μm silicon is 55.3 keV [33], corresponding to about 15 300 electron-hole pairs. Using the ^{241}Am spectrum and assuming a linear relation between energy and ToT, a peak position of $\text{ToT Code } 6.3 \pm 1.0$ is expected, matching the measured peaks.

Both LV1 distributions in Figure 7.16 show a small constant background. This can be caused by coincidental hits from the 546 keV decay, where the electrons do not reach the scintillator. An attempt to mask pixels based on the LV1 distribution is unsuccessful, as they appear to be random hits and not noisy pixels.

Most of the hits are registered with a delay of 5 bunch crossings from the trigger command. This is an arbitrary delay caused by the external triggering mechanism. The delay can be adjusted in the frontend settings and by the FPGA triggering mechanism to keep the hits in the 16 bunch crossing wide window after the trigger. Compared to the Multi-IO, the MMC3 shows less hits with a LV1 delay of 6 and no visible excess for a LV1 delay of 7. This could be caused by a slightly different timing of the trigger processing between both boards, which is supported by the fact that this deviation is only observed when operating with an external triggering source.

Frontend Selftrigger

With the frontend selftrigger, the trigger rate is no longer limited by the scintillator and the recorded amount of data over the same period of time as well as the trigger rate between both boards has increased by a factor of 11, as can be seen in Table 7.9. The

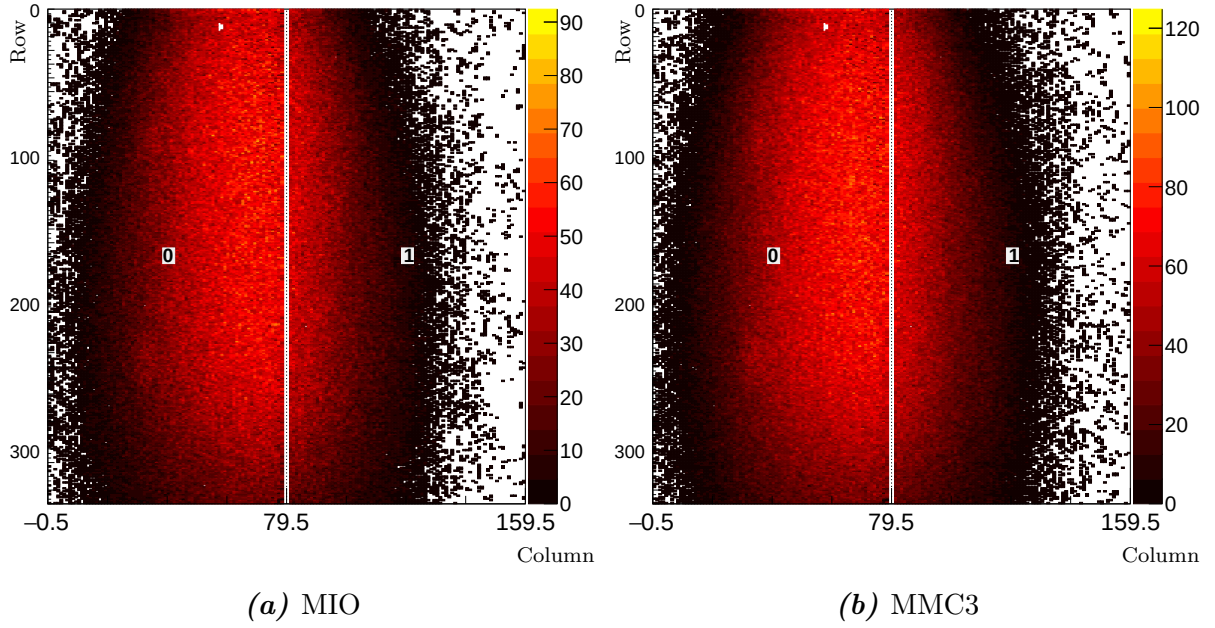


Figure 7.13.: Occupancy histograms showing the number of hits per pixel for the measurement with the ^{90}Sr source and external triggering obtained with the Multi-IO board and the MMC3.

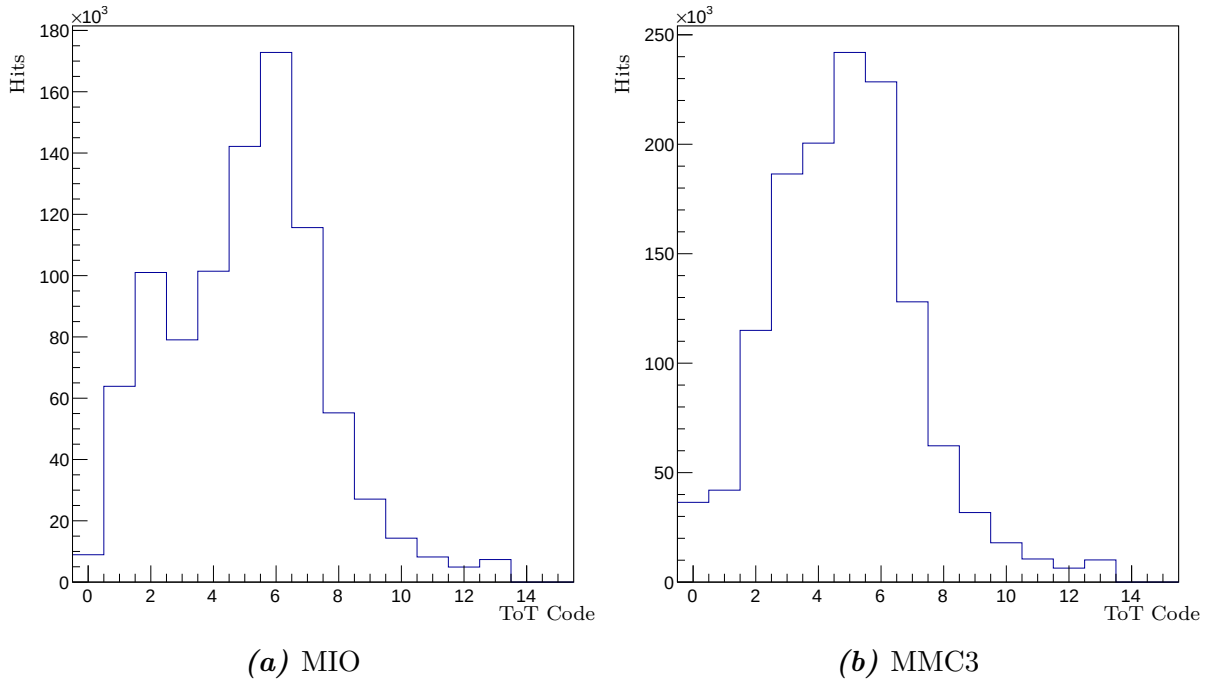


Figure 7.14.: Unclustered ToT histograms showing the charge spectrum for the measurement with the ^{90}Sr source and external triggering obtained with the Multi-IO board and the MMC3.

7. Verification and Results

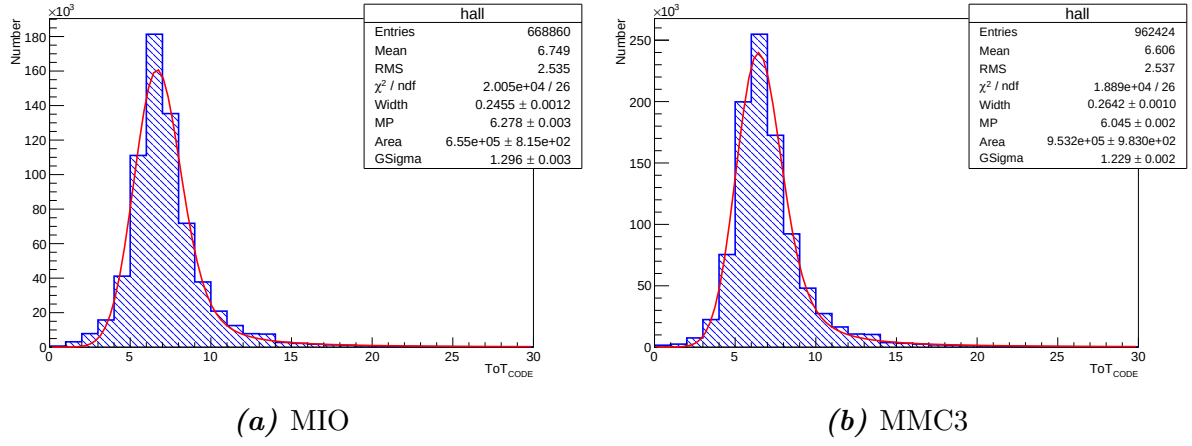


Figure 7.15.: Clustered ToT spectrum for the measurement with the ^{90}Sr source and external triggering obtained with the Multi-IO board and the MMC3.

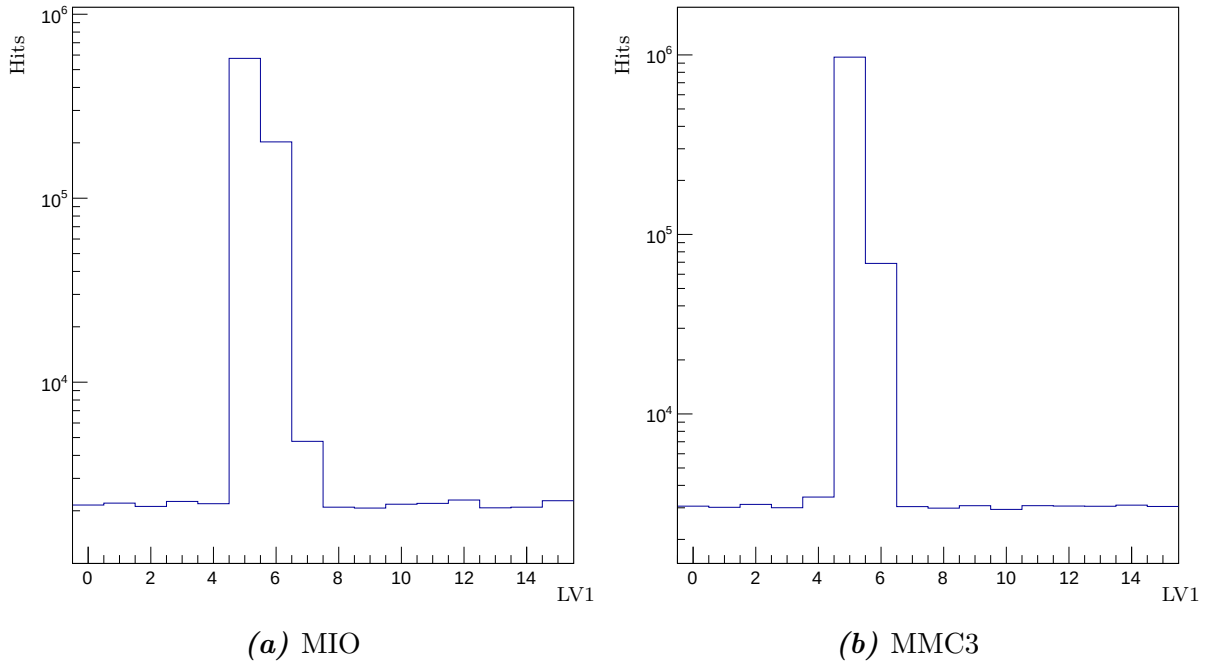


Figure 7.16.: LV1 histograms showing the delay between trigger and hit for the measurement with the ^{90}Sr source and external triggering obtained with the Multi-IO board and the MMC3.

Table 7.9.: Amount of data recorded during the measurement with the ^{90}Sr source and the selftrigger.

	MIO	MMC3
Duration	279 s	302 s
#DH	17 144 918	206 013 720
#DR	2 095 399	25 287 980
Hits	2 799 539	33 734 035
Hit rate	10 034 Hz	111 702 Hz

increase in data is also visible in the occupancy histograms in Figure 7.17, where the intensity profile of the ^{90}Sr source becomes much clearer for the MMC3 compared to the Multi-IO board.

Both ToT distributions in Figure 7.18 are in good agreement and show a peak around ToT Code 3. They are again clustered and fitted (see Figure 7.19), which yields a most probable value of ToT Code 6.0 and a width of 1.8 for the Multi-IO board and a most probable value of ToT Code 6.2 and a width of 1.8 for the MMC3. The peak position again matches the expectation.

The frontend selftrigger causes almost all hits to be registered in the same bunchcrossing as the triggers are issued (see Figure 7.20).

7. Verification and Results

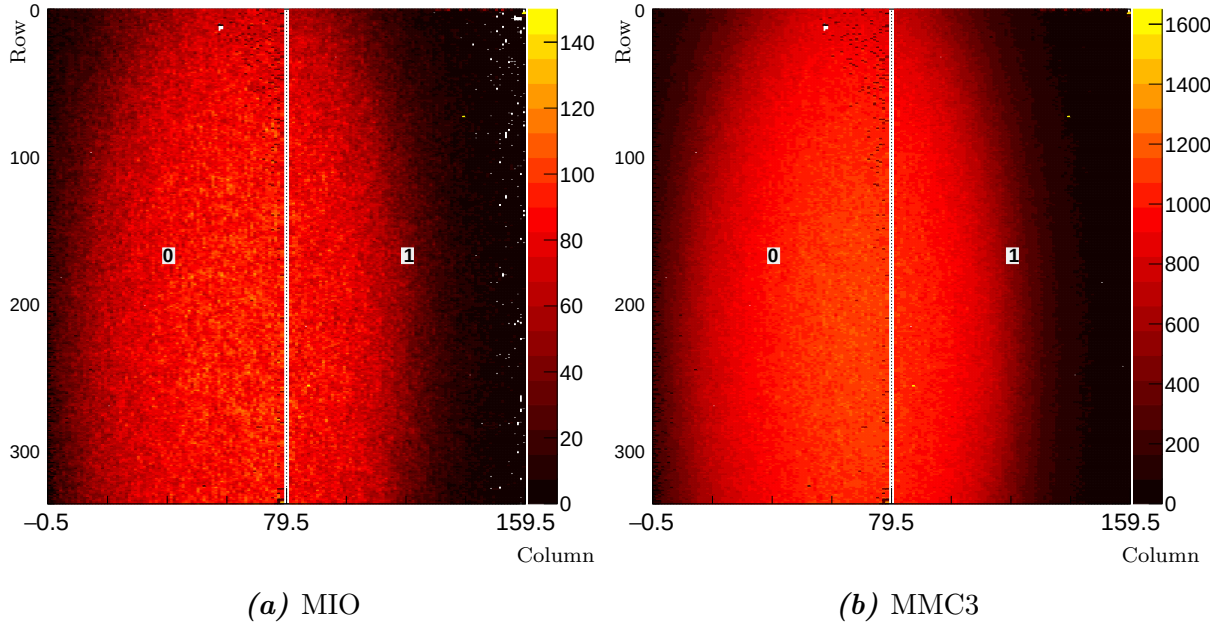


Figure 7.17.: Occupancy histograms showing the number of hits per pixel for the measurement with the ^{90}Sr source and the frontend selftrigger obtained with the Multi-IO board and the MMC3.

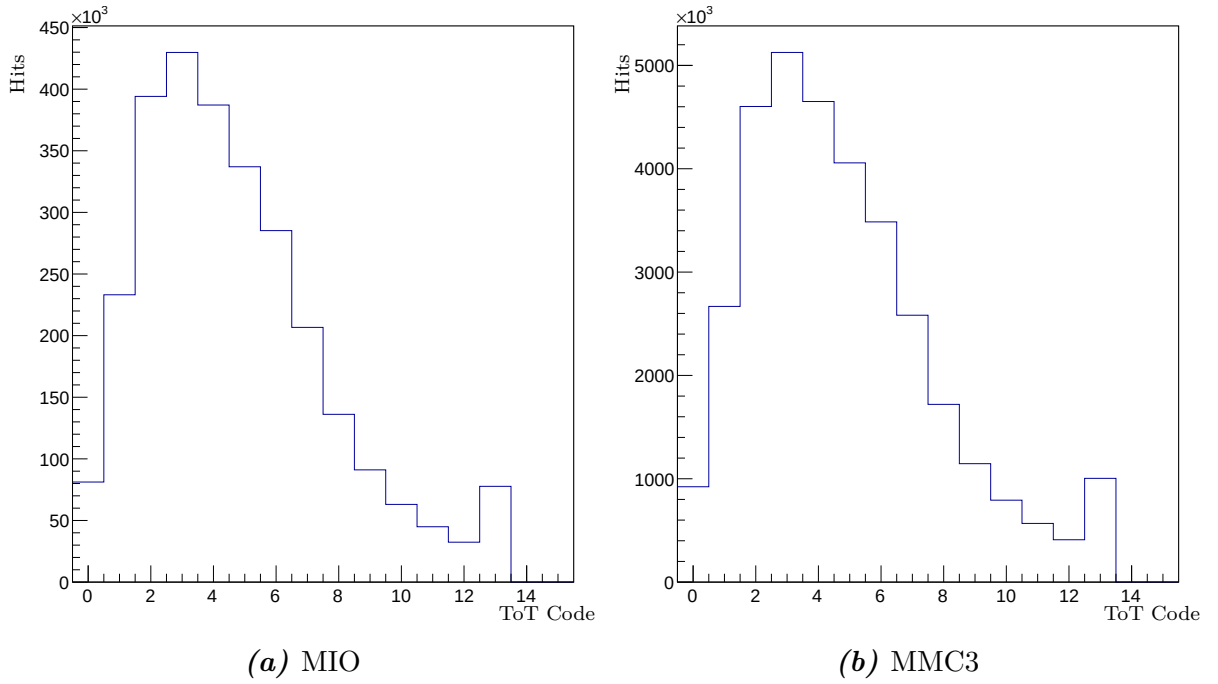


Figure 7.18.: Unclustered ToT histograms showing the charge spectrum for the measurement with the ^{90}Sr source and the frontend selftrigger obtained with the Multi-IO board and the MMC3.

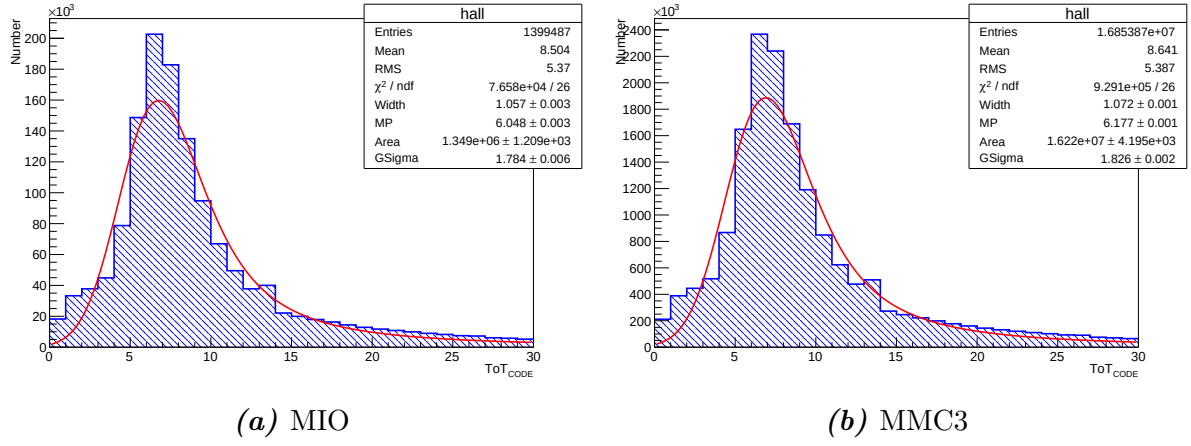


Figure 7.19.: Clustered ToT spectrum for the measurement with the ^{90}Sr source and the frontend selftrigger obtained with the Multi-IO board and the MMC3.

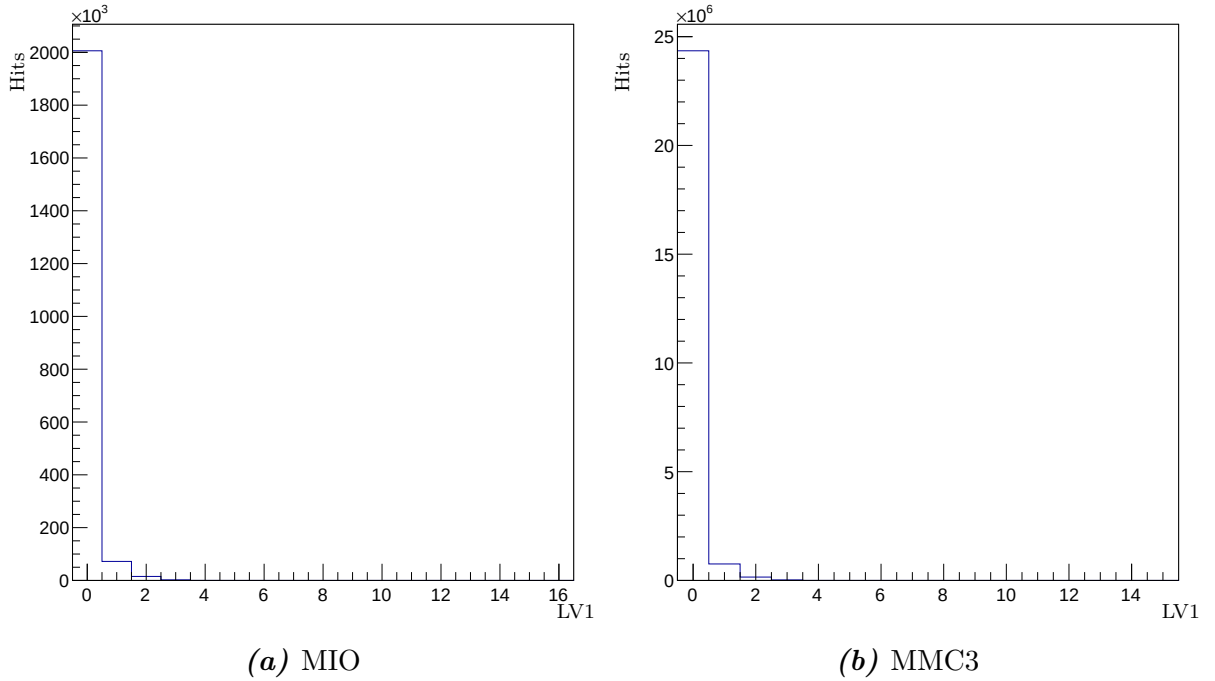


Figure 7.20.: LV1 histograms showing the delay between trigger and hit for the measurement with the ^{90}Sr source and the frontend selftrigger obtained with the Multi-IO board and the MMC3.

7.5. Performance Measurements

Besides verifying the correct operation of the USBpix 3 integration into STcontrol, a comparison of the performance is also of interest. The duration of scans is compared between the Multi-IO board and the MMC3, and a detailed analysis of the differences during frontend configuration is performed.

Scan Duration

A wide range of scans which were performed during the previous tuning are compared. The scan duration is determined from the time-stamps stored by STcontrol. The time-stamps have a resolution of one second, which dominates the uncertainty for short scans. The shortest scans, the Analog Test and Digital Test on the MMC3 have a duration of five seconds.

The duration of longer scans can fluctuate by several seconds. Therefore, all values are rounded to five seconds. The results are summarised in Table 7.10. All scans perform significantly faster on the MMC3, often by a factor of three to four, some even faster.

Table 7.10.: Comparison of scan and tuning durations on the Multi-IO board and the MMC3.

Scan	MIO [mm:ss]	MMC3 [mm:ss]	Speedup factor
Analog Test	0:20	0:05	4.0
Digital Test	0:20	0:05	4.0
Threshold Scan	13:25	5:20	2.5
ToT Verify	0:20	0:05	4.0
GDAC Fast Tune	3:45	0:40	5.6
TDAC Fast Tune	2:20	0:45	3.1
IF Tune	2:45	0:50	3.3
FDAC Tune	2:40	0:35	4.6

Frontend Configuration

The frontend configuration from the previous source measurements, which consists of global and pixel register values for two frontends, is written to the frontends and the USB traffic is recorded.

Each USB request consists of the transfer initiated by the host and the response of the USBpix board. The duration is determined from the recorded timestamps of the first and

Table 7.11.: Comparison of the duration and number of USB requests for writing a frontend configuration.

	MIO	MMC3
Duration	13.402 s	0.053 s
USB requests	147 459	542
Request rate	11 003 Hz	10 226 Hz

last USB transfer. The results are summarised in Table 7.11.

Writing the configuration takes the Multi-IO about 250 times longer than the MMC3 and generates about 270 times more USB requests. The request rates for both boards are comparable, with the Multi-IO having an approximately 10% higher rate. This indicates that the difference in speed is not caused by the difference in available bandwidth between USB 2.0 and 3.0, but by the high number of requests, which accumulate some latency for each request. Therefore, an adaption of the Multi-IO firmware to the MMC3 would have been likely to experience the same issue.

The USBPixController issues many small requests when communicating with the frontend, while the new controller buffers commands and executes them in larger chunks. On the MMC3, the complete configuration is transferred in 53 packets with an average payload of 1886 bytes, which is close to the size of the FPGA command memory buffer of 2032 bytes. The remaining requests are used to access the FPGA firmware. Overall, about 1 MB of commands are transferred to the frontends.

On the Multi-IO board, the pixel registers are written in 2242 requests with an average payload of 43 bytes. The *WrFrontEnd* command to write one pixel register double column is 87 bytes large, but gets split across two packets. All accesses to global registers result in 9834 packets with an payload of 5 bytes each, which is exactly the size of one *WrRegister* command. The remaining transfers again control the FPGA firmware. The commands executed on the frontends amount to about 1.5 MB, which is comparable to the MMC3, but is spread over about 230 times more USB requests, leading to the much longer duration.

8. Summary and Outlook

The new generation of USBpix boards meets the increased demands on bandwidth and resources to use the new FE-I4 to its full potential and has reserves for the following generations of readout chip that are under development for the HL-LHC upgrade. Adding support for USBpix 3 to STcontrol combines the improved hardware with the tested and tried readout software and allows to maintain a familiar workflow while benefiting from the capabilities of the new boards.

To support the new MMC3 and MIO3 boards in STcontrol, a new hardware specific library and a new PixController class were developed and a wide range of measurements was performed to compare the old and the new USBpix systems and to validate the correct operation of the new software. Overall, the comparison measurements show very good agreement between both systems and the proper operation of the new library and PixController for USBpix 3 has been verified.

By using the PyBAR FPGA firmware, several limitations of the FPGA firmware used by STcontrol for the USB 2.0 system are avoided. This includes the high number of USB requests, which especially limits the rate at which commands can be transferred to the frontend, and the non-continuous data taking, which reduces the average data acquisition rate. Therefore, the new library and PixController offer a vastly improved performance compared to the old system.

STcontrol now supports most strobe scans and source scan modes on the MMC3 and MIO3 boards, including the frontend selftrigger and external triggering, and can operate up to eight frontends in parallel. It can also process triggers from a Trigger Logic Unit, which is commonly used in testbeam setups. For complete integration into data acquisition frameworks such as EUDAQ, the new PixController still requires some changes to forward the recorded data to the framework.

The MIO3 is currently limited to operating with the Burn-In Card, but the existing classes in the USBpixI4dll for controlling the additional hardware on the Adapter Card and the General Purpose Analog Card could be adapted to provide full support.

A. Appendix

A. Appendix

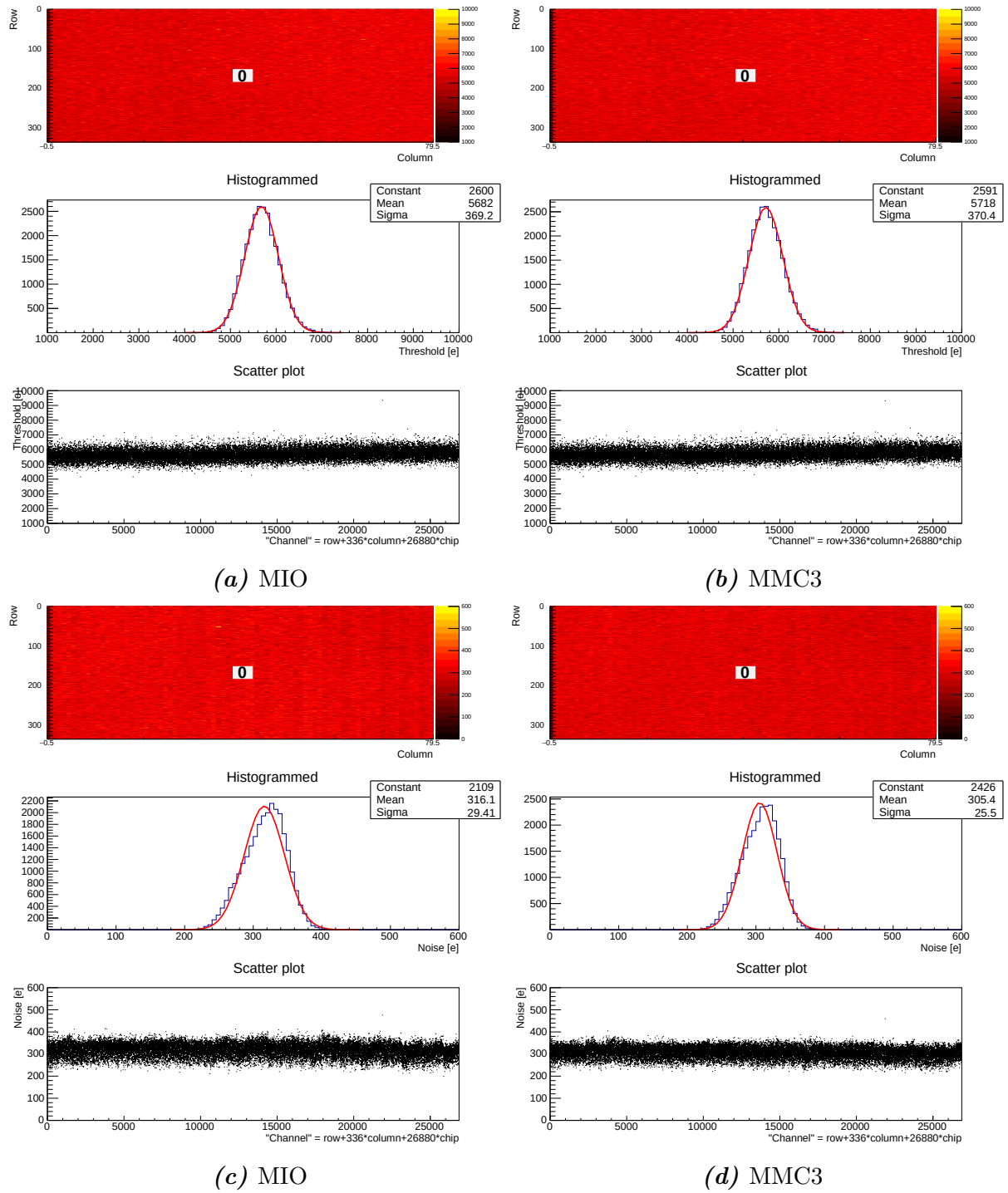
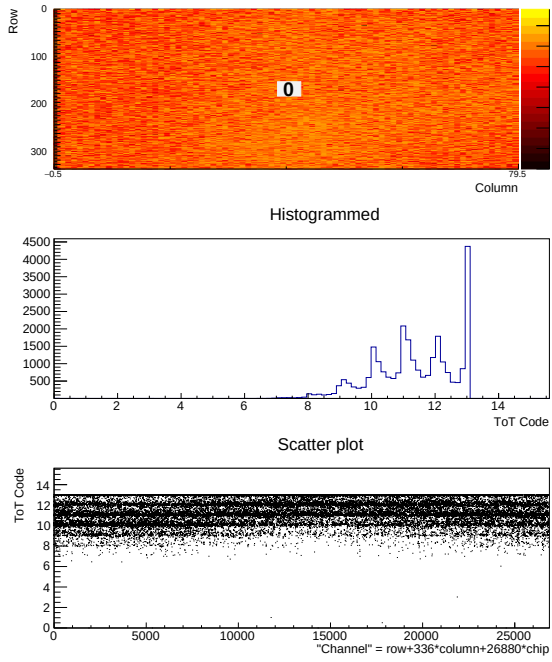
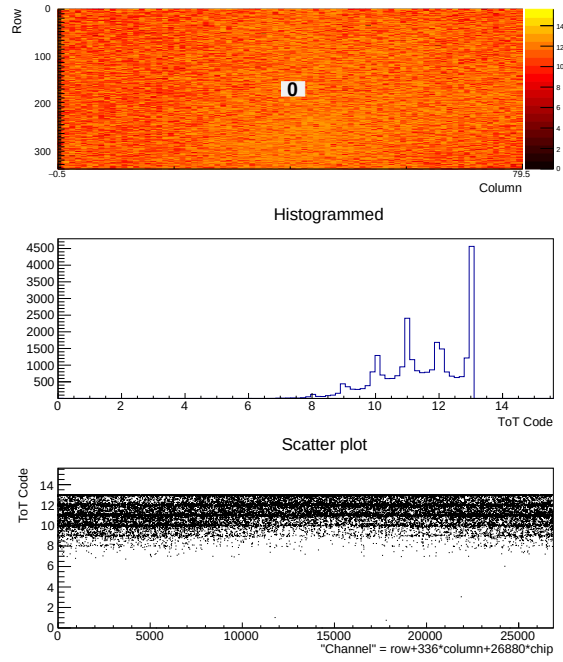


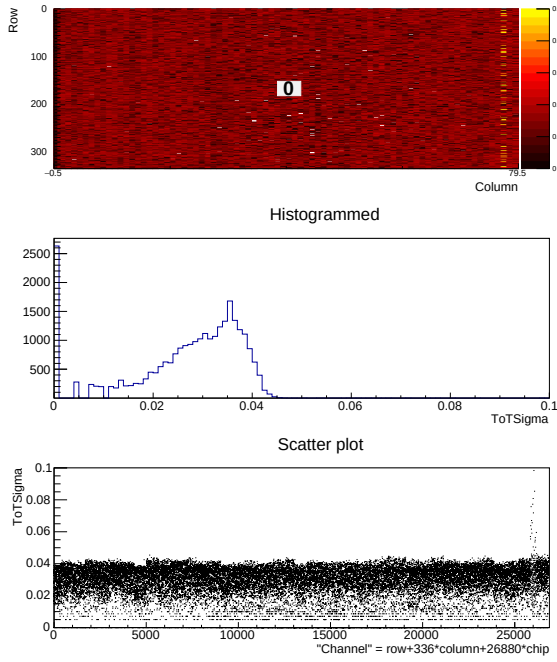
Figure A.1.: Measured threshold (a,b) and noise (c,d) for the untuned 66-W10-IBL3 module with the Multi-IO and the MMC3.



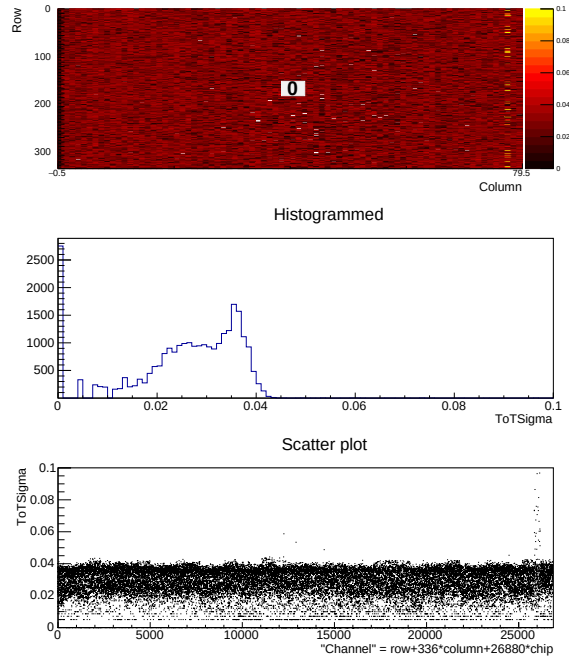
(a) MIO



(b) MMC3



(c) MIO



(d) MMC3

Figure A.2.: Mean ToT (a,b) and standard deviation (c,d) for the untuned 66-W10-IBL3 module with the Multi-IO and the MMC3.

A. Appendix

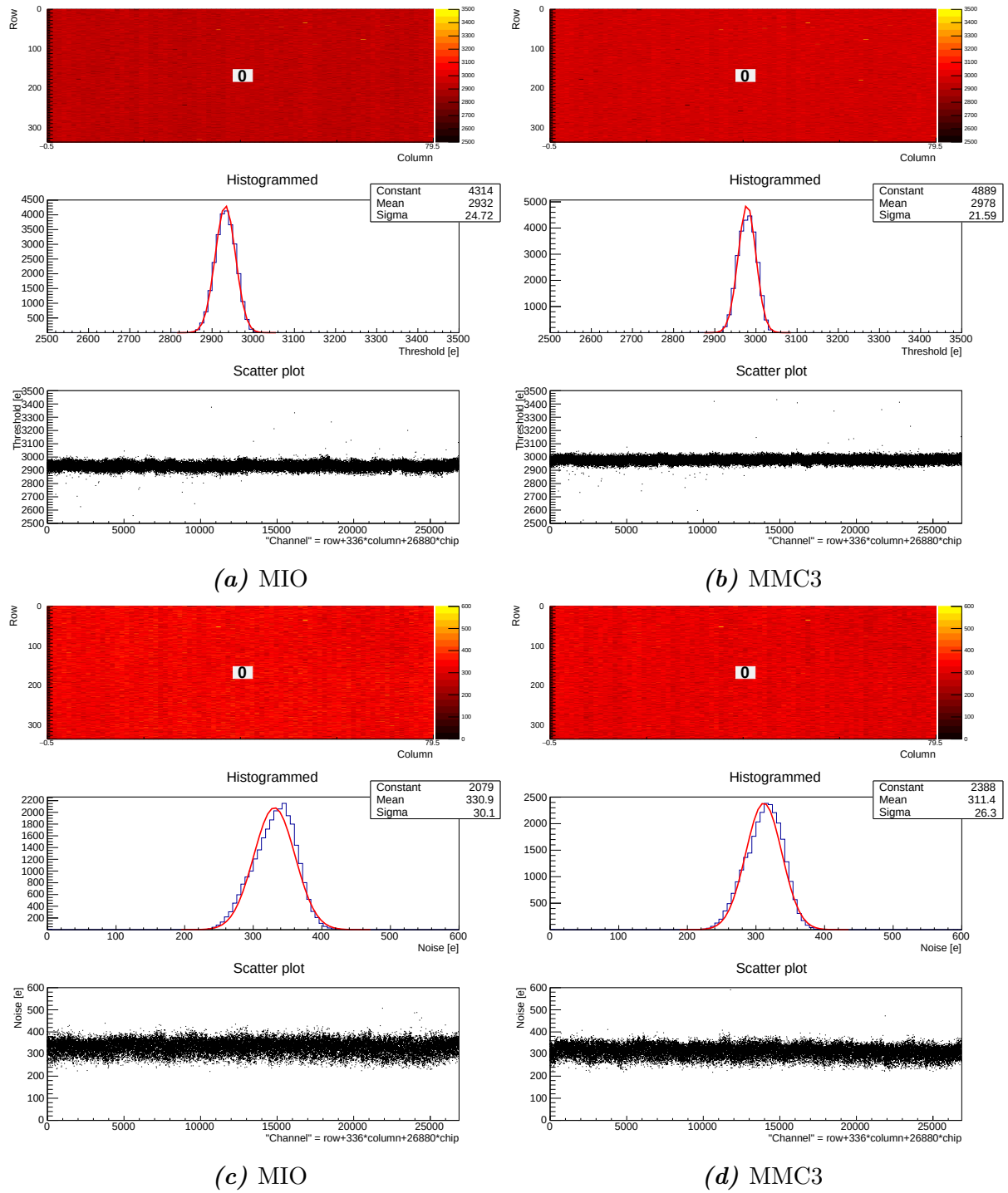


Figure A.3.: Measured threshold (a,b) and noise (c,d) for the 66-W10-IBL3 module after the threshold tuning with the Multi-IO and the MMC3.

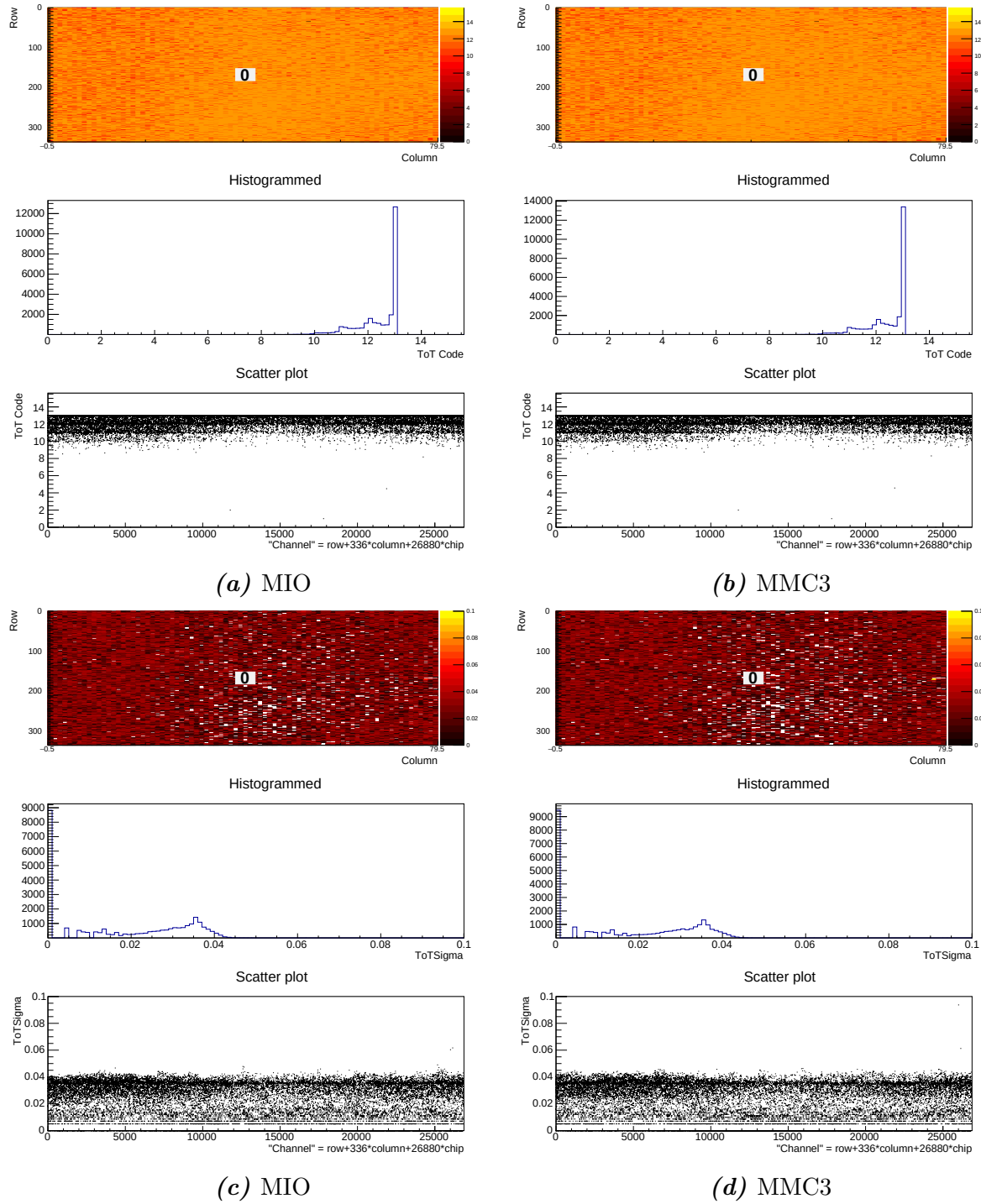


Figure A.4.: Mean ToT (a,b) and standard deviation (c,d) for the 66-W10-IBL3 module after the threshold tuning with the Multi-IO and the MMC3.

A. Appendix

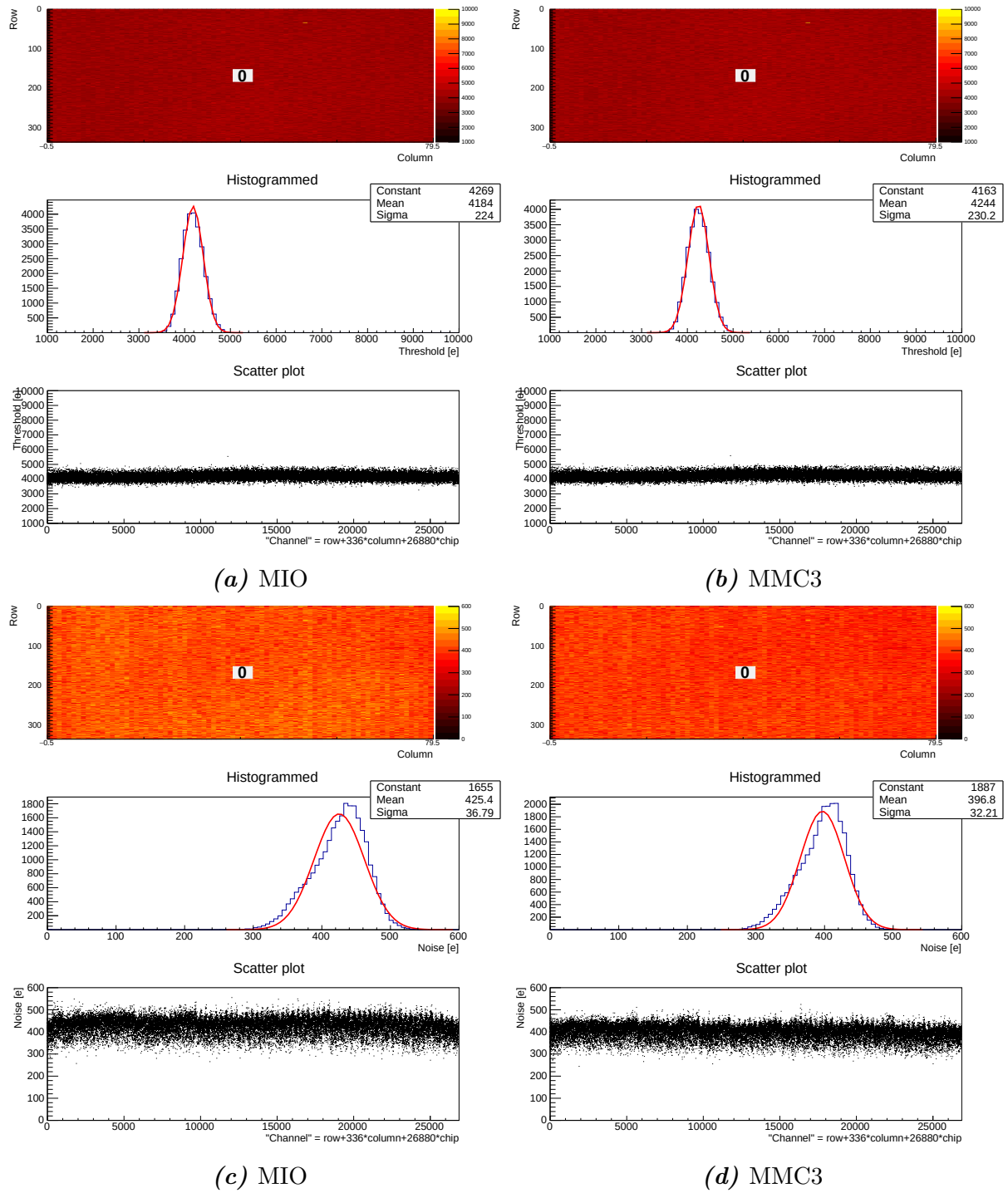


Figure A.5.: Measured threshold (a,b) and noise (c,d) for the 66-W10-IBL3 module after the threshold and ToT tuning with the Multi-IO and the MMC3.

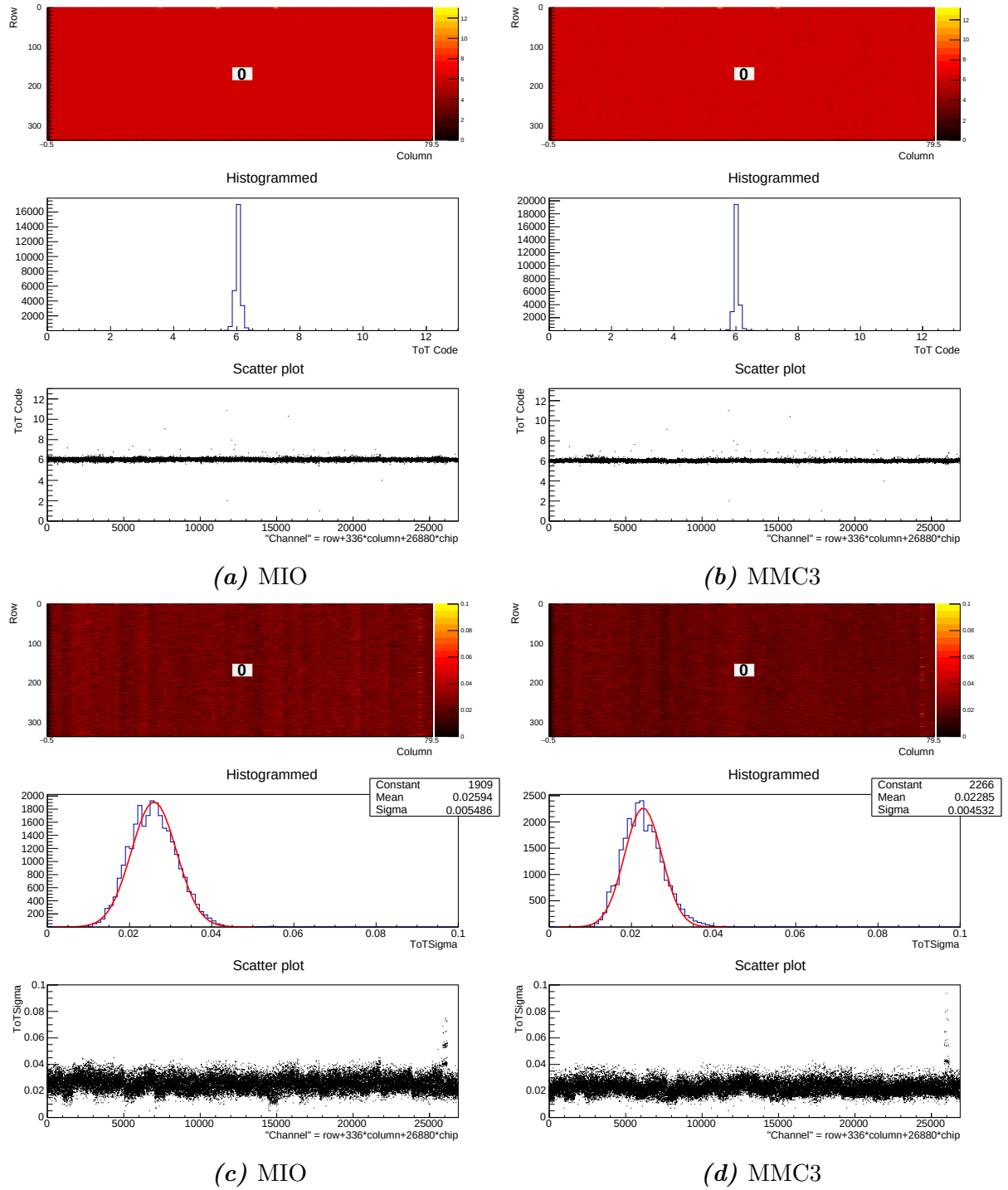


Figure A.6.: Mean ToT (a,b) and standard deviation (c,d) for the 66-W10-IBL3 module after the threshold and ToT tuning with the Multi-IO and the MMC3.

A. Appendix

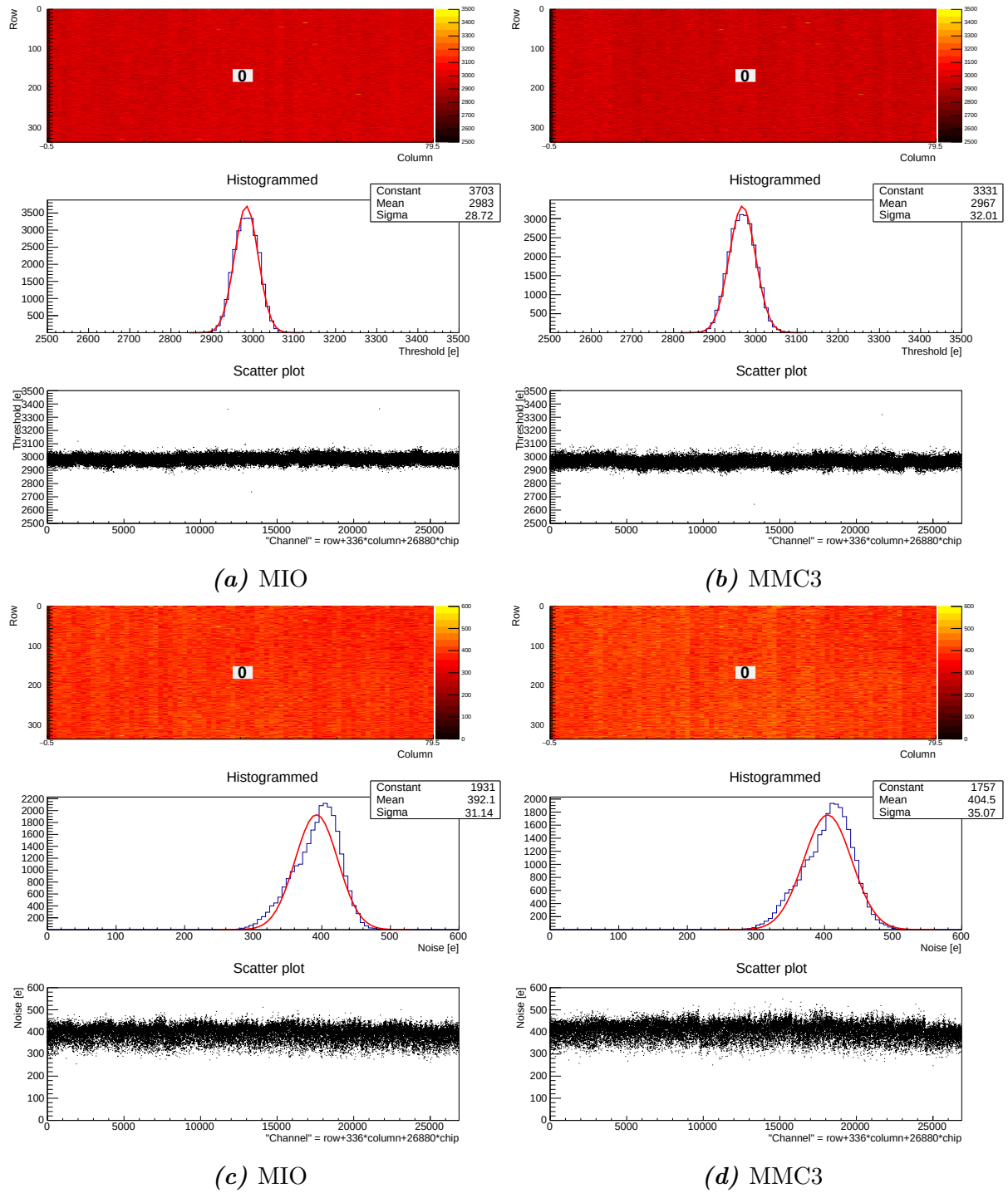
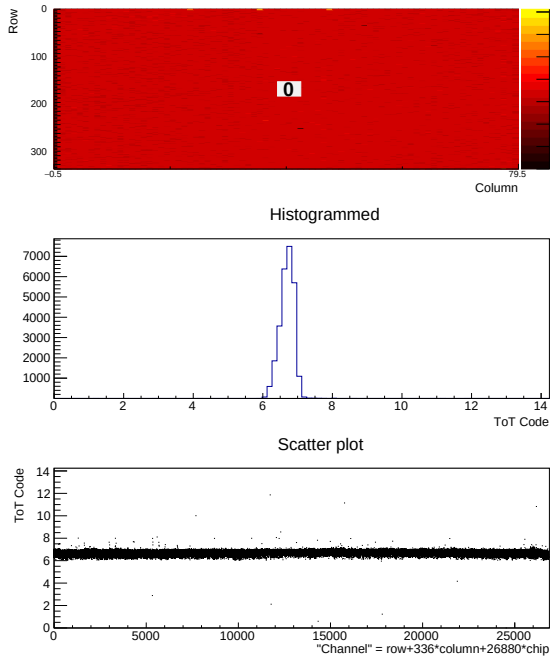
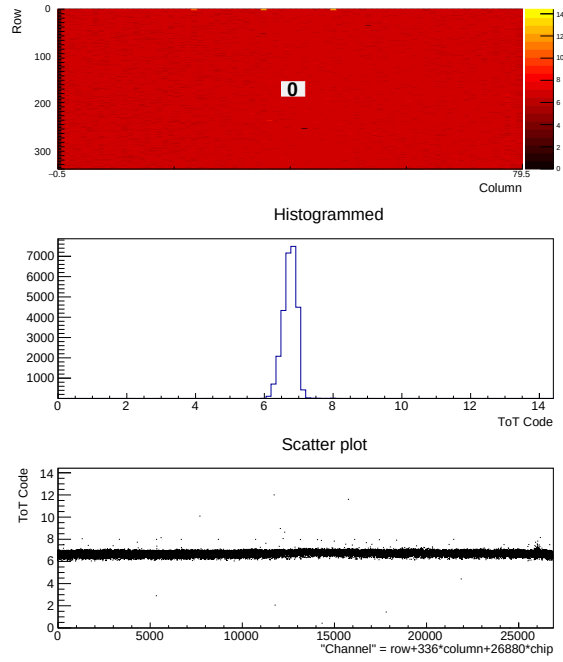


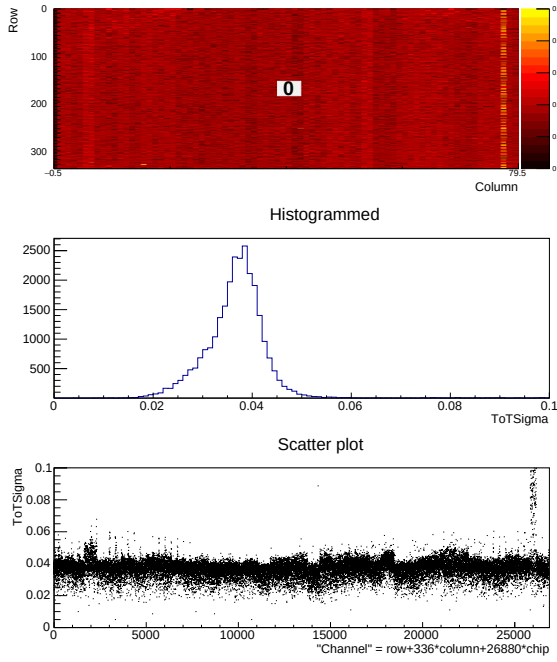
Figure A.7.: Measured threshold (a,b) and noise (c,d) for the 66-W10-IBL3 module after the full tuning with the Multi-IO and the MMC3.



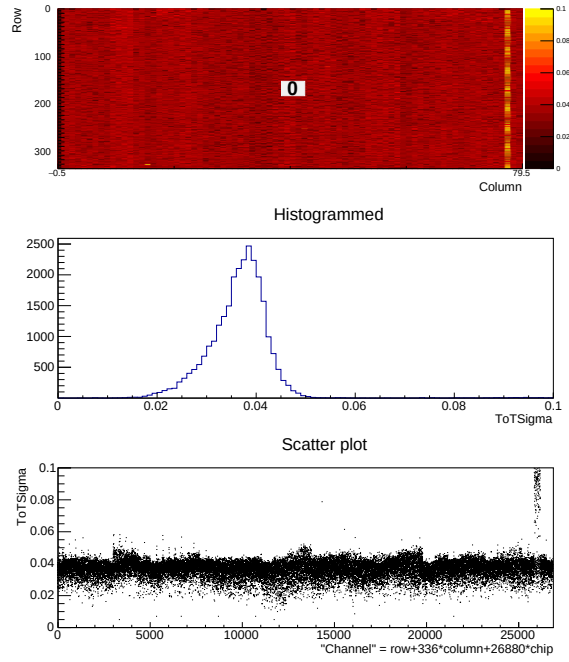
(a) MIO



(b) MMC3



(c) MIO



(d) MMC3

Figure A.8.: Mean ToT (a,b) and standard deviation (c,d) for the 66-W10-IBL3 module after the full tuning with the Multi-IO and the MMC3.

Bibliography

- [1] ATLAS Collaboration, *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 (2012)
- [2] CMS Collaboration, *Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC*, Phys. Lett. **B716**, 30 (2012)
- [3] D. Griffiths, *Introduction to Elementary Particles*, Wiley, second edition (2008)
- [4] ATLAS Collaboration, *The ATLAS experiment at the CERN large hadron collider*, JINST **3(08)**, S08003 (2008)
- [5] ATLAS Collaboration, *ATLAS: Technical proposal for a general-purpose p p experiment at the Large Hadron Collider at CERN* (1994)
- [6] ATLAS Collaboration, *ATLAS detector and physics performance: Technical Design Report, 1*, Technical Design Report ATLAS, CERN, Geneva (1999)
- [7] ATLAS Collaboration, *ATLAS pixel detector electronics and sensors*, JINST **3(07)**, P07007 (2008)
- [8] V. A. Mitsou (ATLAS TRT), *The ATLAS transition radiation tracker*, in *Astroparticle, particle and space physics, detectors and medical physics applications. Proceedings, 8th Conference, ICATPP 2003, Como, Italy, October 6-10, 2003*, pages 497–501 (2003)
- [9] M. Capeans, et al., *ATLAS Insertable B-Layer Technical Design Report*, Technical Report CERN-LHCC-2010-013. ATLAS-TDR-19, CERN, Geneva (2010)
- [10] ATLAS Collaboration, *Letter of Intent for the Phase-II Upgrade of the ATLAS Experiment*, Technical Report CERN-LHCC-2012-022. LHCC-I-023, CERN, Geneva (2012)
- [11] W. R. Leo, *Techniques for nuclear and particle physics experiments: a how-to approach*, Springer Science & Business Media (1994)

Bibliography

- [12] K. Olive, P. D. Group, *Review of Particle Physics*, Chin. Phys. C **38(9)**, 090001 (2014)
- [13] L. Rossi, *Pixel Detectors: From Fundamentals to Applications*, Particle Acceleration and Detection, Springer (2006)
- [14] W. Shockley, *Currents to conductors induced by a moving point charge*, J. Appl. Phys. **9(10)**, 635 (1938)
- [15] S. Ramo, *Currents induced by electron motion*, Proc. Ire. **27**, 584 (1939)
- [16] D. Arutinov, et al., *Digital Architecture and Interface of the New ATLAS Pixel Front-End IC for Upgraded LHC Luminosity*, IEEE Trans. Nucl. Sci. **56(2)**, 388 (2009)
- [17] The ATLAS IBL collaboration, *Prototype ATLAS IBL modules using the FE-I4A front-end readout chip*, JINST **7(11)**, P11010 (2012)
- [18] ATLAS Collaboration, *The FE-I4B integrated circuit guide* (2012), internal document
- [19] M. Backhaus, et al., *Development of a versatile and modular test system for ATLAS hybrid pixel detectors*, Nucl. Instrum. Meth. A **650(1)**, 37 (2011)
- [20] J.Schneider, H.Krüger, *S3 Multi IO System - S3 Multi IO USB Card Version V1.03* (2010), internal document
- [21] H.Krüger, *Burn-In Card for FE-I4B - User Manual* (2012), internal document
- [22] H.Krüger, *GPAC - General Purpose Analog Card* (2014), internal document
- [23] *USBpix - USB based readout system for ATLAS FE-I3 and FE-I4*, visited on 2015-09-25, URL <http://icwiki.physik.uni-bonn.de/twiki/bin/view/Systems/UsbPix>
- [24] *pyBAR - Bonn ATLAS Readout in Python and C++*, visited on 2015-09-25, URL <https://github.com/SiLab-Bonn/pyBAR>
- [25] *Basil - a data acquisition system and system testing framework in Python*, visited on 2015-09-25, URL <https://github.com/SiLab-Bonn/basil>
- [26] Enclustra, *Mercury KX1 User Manual V1.02* (2015)
- [27] Cypress Semiconductor Corporation, *CY7C68013A, CY7C68014A, CY7C68015A, CY7C68016A EZ-USB FX2LP USB Microcontroller High-Speed USB Peripheral Controller Datasheet* (2015)

- [28] Cypress Semiconductor Corporation, *CYUSB301X, CYUSB201X EZ-USB FX3 SuperSpeed USB Controller Datasheet* (2015)
- [29] Xilinx, *Spartan-3 FPGA Family Data Sheet* (2013)
- [30] Xilinx, *7 Series FPGAs Overview* (2014)
- [31] T. W. G. Westphal, *Test measurements with active pixel sensors for the upgrade of the ATLAS pixel detector*, Bachelor's thesis, Georg-August-Universität Göttingen (2015), II.Physik-UniGö-BSc-2015/08
- [32] Compaq Computer Corporation, et al., *Universal Serial Bus Specification Revision 2.0* (2000)
- [33] H. Bichsel, *Straggling in thin silicon detectors*, Rev. Mod. Phys. **60**, 663 (1988)

Danksagung

Ich bedanke mich bei Prof. Dr. Arnulf Quadt, der mir die Möglichkeit gegeben hat diese Masterarbeit anzufertigen und Priv.Do. Dr. Jörn Große-Knetter, der immer für jegliche Fragen zur Verfügung stand.

Außerdem gilt mein Dank Dr. Jens Weingarten für seine Hilfe bei vielen Messungen und technischen Problemen und für das Korrekturlesen meiner Arbeit, sowie Julia Rieger für viele hilfreiche Hinweise zu USBpix und STcontrol.

Ich danke allen, die mich unterstützt und motiviert haben und dem ganzen Hardware-Team für viele Kicker-Partien und viele unterhaltsame Kaffeepausen.

Erklärung

nach §17(9) der Prüfungsordnung für den Bachelor-Studiengang Physik und den Master-Studiengang Physik an der Universität Göttingen:

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe.

Darüberhinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, im Rahmen einer nichtbestandenenen Prüfung an dieser oder einer anderen Hochschule eingereicht wurde.

Göttingen, den 21. April 2016

(Eric Buschmann)