GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Fakultät für
Physik

Master's Thesis

Entwicklung eines Datennahmesystems für Teststrahlmessungen mit der ATLAS Pixel Front-End Elektronik

Development of a Data Acquisition System for Testbeam Measurements with the ATLAS Pixel Front End Electronics

prepared by

Björn Klaas

from Detmold

at the II. Physikalisches Institut

Thesis number:	II.Physik-UniGoe-MSc-2015/06
Thesis period:	23rd April 2015 until 23rd October 2015
First referee:	Prof. Dr. Arnulf Quadt
Second referee:	Priv.Doz. Dr. Jörn Große-Knetter

Abstract

The search for unknown particles and new physics continues at increasingly powerful particle accelerators. The most energetic to date is the LHC at CERN, aiming for a centre-of-mass energy of 14 TeV at an instantaneous luminosity of $10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ by 2018. Then, and again in 2022, it will be shut down for upgrades to more than quadruple its currently achievable luminosity

The collisions are recorded by large detectors, such as the ATLAS experiment. To continuously provide precision data at high efficiencies a huge R&D effort is necessary, developing new detector technologies and designing, building and testing prototypes. Evaluation is performed using several test systems, such as the USBpix system for ATLAS Pixel Detector sensors and front-end electronics.

This thesis describes the current state of the USBpix system and the implementation of an improved read-out scheme. The new scheme transitions from on-board data storage with stopping read-out to on-board data buffering with continuous read-out, greatly improving the achievable average data acquisition rate. Changes made to the hardware interface libraries of the host software and to the USBpix FPGA firmware are presented.

Keywords: LHC, ATLAS, USBpix, Firmware, FPGA

Contents

1 Introduction

The scientific search for the fundamental particles of the universe and their interactions resulted in the *Standard Model of Particle Physics*. This collection of well-tested theories describes most physical phenomena observed to date. Even though the model contains all observed particles, as well as three out of four fundamental interactions between them, many unanswered questions remain. These lead physicists to push technological boundaries ever further to obtain data from energy regions inaccessible so far.

The current frontier is set by the *Large Hadron Collider* (LHC), the world's largest and most energetic particle collider. It is located at the *European Centre for Nuclear Research* (CERN) at Geneva, Switzerland. During its first run (Run I, 2010–2013) it operated at centre-of-mass energies of 7 TeV and 8 TeV, and reached a peak luminosity of $6 \cdot 10^{33}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$. During a first long shut-down from 2013 to 2015, the LHC received a complete overhaul, preparing it for Run II at centre-of-mass energies of 13 TeV and 14 TeV, with a peak luminosity of $10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$. A second and third shut-down, to further increase the luminosity, are planned for 2018 and 2022 (HL-LHC, see chapter 2).

The spray of particles generated in the LHC's high energy collisions is recorded by four large physics experiments, built around the interaction points. The data collected by the two general-purpose experiments, ATLAS and CMS, allowed them to already discover a Higgs boson-like particle, increase the precision on many cross section measurements, and push the exclusion boundaries of many theories further out. The increased luminosity during future LHC runs will allow them to collect an even larger data set, and thus increase the likelihood of discovering new physics. The greater number and intensity of generated particles also increases the strain on the detectors, making upgrades and replacements imperative. The chosen path for ATLAS includes replacement of the entire Inner Detector (ID) by the new *ATLAS Inner Tracker* (ITk), scheduled for the 2022 HL-LHC upgrade (see chapter 3).

Selecting technologies for these upgrades is a delicate affair, requiring extensive

testing and analysis of proposed designs. Similar tests are performed to carefully characterize each part of the current detector, so all gathered data can be interpreted correctly. This multitude of precision measurements is carried out at laboratories and testbeam facilities around the world. A major test system for ATLAS Pixel Detector components is the USBpix test system (see chapter 4). This flexible system provides a framework for testing several generations of various detector components, implementing numerous technologies.

A large part of the system's versatility stems from on-board programmable logic, provided by a *Xilinx Spartan-3A Field Programmable Gate Array* (FPGA). The FPGA handles most on-board logic concerned with device configuration, triggering and data processing. It can be reconfigured via firmware updates, to handle new devices and requirements. An analysis of the FPGA firmware and the hardware interface library of the host software was performed, identifying room for improvement on the current USBpix configuration. Based on this, a set of changes necessary to increase the USBpix data acquisition rate is described, and a possible implementation proposed (see chapter 5). The implementation is evaluated by comparison of several measurements, performed with the current as well as the presented USBpix configuration (see chapter 6).

2 The Large Hadron Collider

Decades of intense research in particle physics resulted in a set of concise and well-tested theories. They describe most of the observable universe through several elementary particles and interactions, introduced below. To explore what lies beyond, and find answers to some yet unsolved problems, the *Large Hadron Collider* (LHC) was built at the *European Centre for Nuclear Research* (CERN), located near Geneva, Switzerland. It is the world's largest and most energetic particle accelerator, and involved more then 10 000 scientists and engineers from over 100 countries in design and construction. After almost ten years of assembly the 27 km long machine was turned on in 2008, achieved first beam in 2009, and started physics operation in 2010. Below the LHC design and upgrade path are outlined.

2.1 The Standard Model of Particle Physics

The *Standard Model of Particle Physics* (SM, Fig. 2.1) was formed by combining the aforementioned theories into a unified model, describing all known elementary particles and three out of four interactions they engage in. The SM contains twelve spin-$^1/_2$ matter particles, called *fermions*, and five full-integer spin *bosons*. The fermions are divided into two groups of six particles each, the *leptons* and *quarks* [**?**].

Leptons and quarks are organized in three *generations*, combining two particles of each group. The first generation contains the lowest mass particles, which cannot decay and are hence called *stable*. They make up almost all observable matter. Each particle has a partner with equal mass but opposite charge signs, called an antiparticle.

The four spin-1 particles, labelled *gauge bosons*, mediate the three fundamental interactions described by the SM. These are the *electromagnetic force*, mediated by photons, the *weak force*, mediated by the vector bosons Z and $W^{\pm}$, and the *strong force*, mediated by eight gluons of different colour charge. Which particles

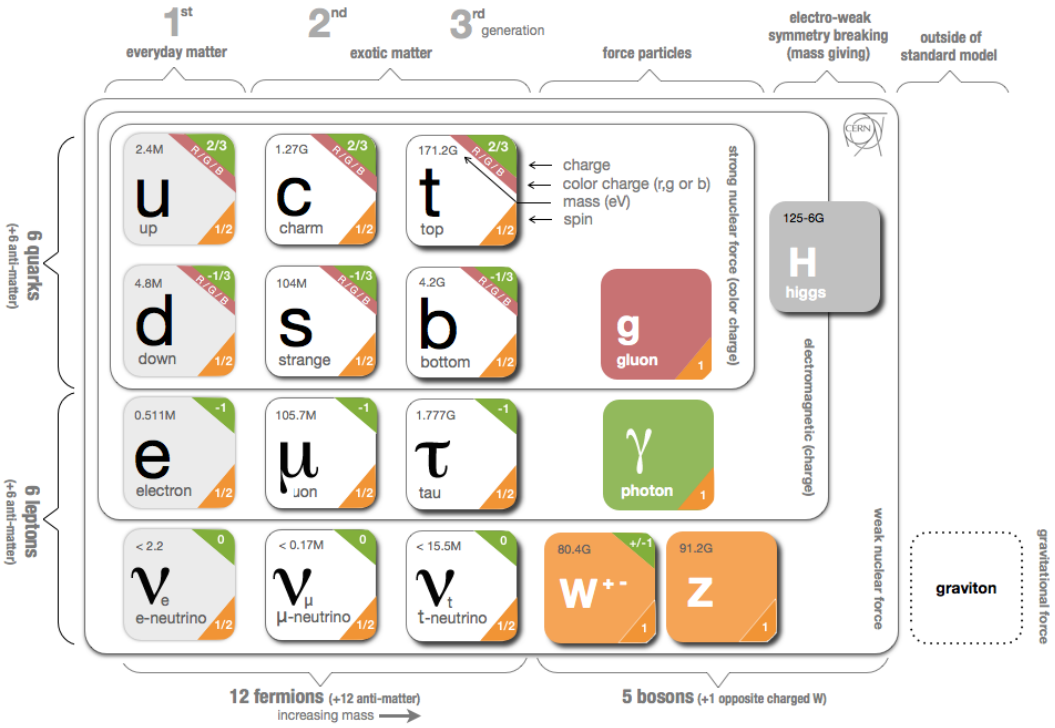


Figure 2.1: Overview of the particles described by the Standard Model of Particle Physics.

are affected by a certain force depends on the respective gauge boson's characteristics. Photons couple only to electromagnetically charged particles, and gluons to particles carrying colour charge, while Z and $W^{\pm}$ couple to all fermions. The ranges of the interactions differ greatly, as well as the particles they interact with. Since the photon is massless, the range of the electromagnetic interaction is infinite, while the strong interaction is severely limited by colour confinement, and the weak interaction due to the decay of the Z and $W^{\pm}$. These decays occur because Z and $W^{\pm}$ are massive particles, measuring $m_Z = (91.1876 \pm 0.0021)\,\mathrm{GeV/c^2}$ and $m_W = (80.385 \pm 0.015)\,\mathrm{GeV/c^2}$ [**?**].

Those masses require an extension of the SM, since it originally needed massless gauge bosons to ensure local gauge invariance. The spin-0 Higgs boson was proposed, as part of the *Higgs mechanism*, providing the most elegant way of keeping the model consistent [**? ?**]. Finding it was one of the main goals of the LHC and the ATLAS and CMS experiments, and was achieved in July 2012 [**? ?**]. The discovery was a

great success for particle physics and rewarded with a Nobel Prize for the founders of the Higgs theory, Peter Higgs and Francois Englert. The ATLAS and CMS collaborations received honorary mentions during the ceremony and in the laudatio, for finally discovering the elusive particle. The search had taken many decades, since theory could not predict the mass of and no decay exclusive to the Higgs. It was eventually found at $m_H = (125.6 \pm 0.3)\,\mathrm{GeV/c^2}$ [**?**], an energy only accessible with very limited luminosity prior to the LHC.

Many more theories and extensions of the SM were proposed, trying to answer the questions left open by the current SM. So far, no evidence could be found for any of them. On the contrary, many theories and versions thereof were already excluded by ATLAS and CMS. The recently started Run II of the LHC will allow to expand the search into new energy regions.

2.2 LHC Design

The LHC was designed for a centre-of-mass energy of $\sqrt{s} = 14\,\mathrm{TeV}$ and a luminosity of $10^{34}\,\mathrm{cm^{-2}\,s^{-1}}$, thought to provide the necessary energy and statistical significance to discover the Higgs boson and new physics phenomena [**?**].

The high target luminosity required a proton-proton collider, since currently achievable anti-proton production rates are too small to sustain the targeted collision rate of 40 MHz of bunches containing $1.1 \cdot 10^{11}$ particles each. To reach this rate, 2808 bunches need to circulate in two separate beampipes, required to keep equally charged particles, moving in opposite directions, on the same trajectory. The number of particles per bunch and the beam focusing facilities were chosen to produce at least 25 inelastic interactions per bunch crossing, generating more than 1000 charged particles every 25 ns.

At the four interaction points, the large LHC experiments ATLAS, CMS, ALICE and LHCb are located (Fig. 2.2). ATLAS and CMS are general-purpose experiments, designed to gather information on all physics phenomena examined at the LHC, while LHCb focuses on b-physics and ALICE on heavy ion collisions, occurring when the LHC is circulating lead ions.

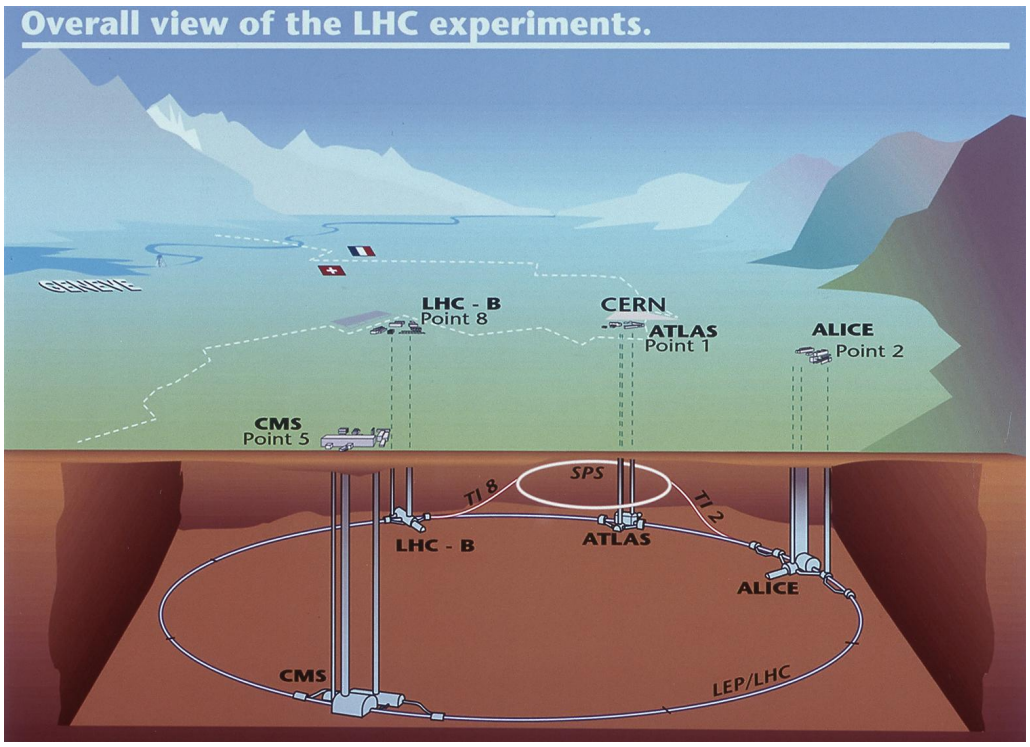


Figure 2.2: The LHC, shown are the locations of the four large experiments and part of the injection chain.

2.3 LHC Runs and Upgrades

An incident upon first start-up delayed the LHC Run I, which occured at a centre-of-mass energy of 7 TeV and 8 TeV from 2010–2013. During the run the LHC delivered an integrated luminosity of approximately $25\,\mathrm{fb}^{-1}$ [**?**]. In early 2013 it was shut down for upgrade and consolidation work, to allow operation at nominal centre-of-mass energy and luminosity upon resuming operation. The following Run II was started in 2015 and is scheduled to end in 2018, when further upgrades to the LHC are due. Initial Run II collisions occurred at a centre-of-mass energy of $\sqrt{s} = 13\,\mathrm{TeV}$ with a bunch spacing of 50 ns. The bunch spacing was switched to 25 ns shortly after, matching the nominal collision rate of 40 MHz, and immediately doubling the instantaneous luminosity.

The LHC is meant to reach its final collision energy of $\sqrt{s} = 14\,\mathrm{TeV}$ at a luminosity of $10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ before the 2018 shut-down. During this next long operational break (LS2, $\approx$ one year) enhancements are to be implemented to raise the luminosity to twice the previous value.

Research and development for a third upgrade named *High Luminosity-LHC* (HL-

LHC) has already begun. The upgrade is scheduled for 2022 and will further increase the LHC's instantaneous luminosity to $5 \cdot 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$. This increase will not only provide more data to the experiments, but also vastly increase the occupancy of the detectors and the radiation damage inflicted on them. The reward for operation under such strenuous conditions will be a data set large enough for many additional precision measurements, outlined below.

Physics at HL-LHC

Increasing the luminosity to HL-LHC values will widen the energy scales under investigation in high energy boson-boson scattering, allow the study of electroweak symmetry breaking (EWSB) and extend the search for SUSY and extra dimensions into the multi-TeV region [**?**]. In the search for new physics, signatures of high-mass gauge bosons, complex SUSY cascade decays, and resonances in $t\bar{t}$-pairs are of special interest. The large amount of data will also greatly increase the precision of Higgs measurements and allow the observation of rare channels, such as $H \to \mu\mu$, vector boson fusion production of $H \to \gamma\gamma$ and $H \to \tau\tau$, and the production with a top-pair $t\bar{t}H$, with $H \to \gamma\gamma$ [**?**]. It will also allow to study Higgs self-coupling in channels such as $HH \to \tau\tau bb$ and $HH \to \gamma\gamma bb$.

3 The ATLAS Experiment

The ATLAS experiment proofed to be a great success during LHC Run I. It very reliably produced a continuous stream of precision data at high efficiencies, achieving its original design goals stated in [**?**]. Nevertheless careful evaluation of Run I performance and damages identified several components not suited for operation under LHC Run II conditions. These are either not capable of handling the expected occupancies and trigger rates, or prone to eventually break under the prolonged exposure to intense radiation. Affected components will be upgraded or replaced in a series of upgrades, planned for the 2018 and 2022 LHC shut-downs.

3.1 Original Detector Design

A set of basic design criteria was established for ATLAS, to ensure high efficiencies for most physics processes of interest at the LHC. These included very good electromagnetic and full-coverage hadronic calorimetry, high-precision muon momentum measurements, efficient tracking for high-p_{T} lepton-momentum measurements, electron and photon identification, τ-lepton and heavy-flavour identification, and full event reconstruction capability at lower luminosity. Other criteria were large acceptance in pseudo-rapidity with almost full azimuthal angle coverage everywhere, and triggering and measurements of particles at low-p_{T} thresholds.

The overall detector layout chosen to accomplish these goals is shown in Fig. 3.1. Its a cylindrical design with several layers of subdetectors and magnets. The innermost layer is the *Inner Detector* (ID), extending to 1.15 m from the beampipe, and surrounded by a thin superconducting solenoid. The solenoid itself is surrounded by the calorimeter system, extending to a 4.25 m radius. The calorimeters are enclosed by the muon system, containing a large superconducting air-core toroid magnet, consisting of independent coils arranged in an eight-fold symmetry. It provides the magnetic field for the muon system and extends to 11 m from the beampipe [**?**].

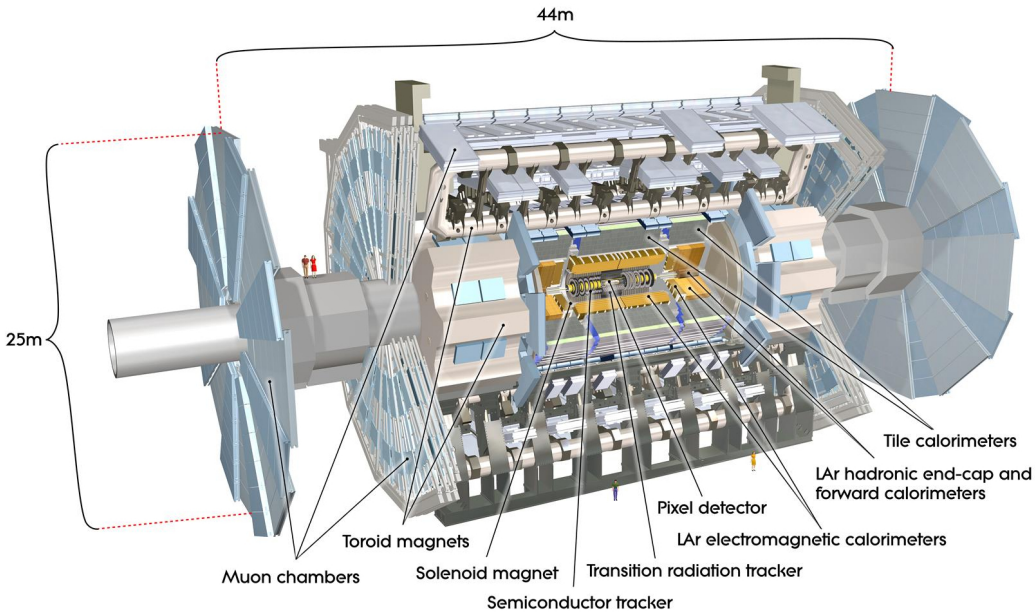


Figure 3.1: Overall layout of the ATLAS general-purpose detector.

3.1.1 The Inner Detector

The purpose of the Inner Detector is precise tracking of particle trajectories and vertex reconstruction. To achieve this, the ID was placed at the centre of ATLAS, directly adjacent to the beampipe. It consists of three subdetectors, each containing several barrel layers and endcaps, chosen to provide the best balance between precision, spatial coverage and cost efficiency. The innermost subdetector, providing the best spatial resolution, is the *Pixel detector* (APD). It is surrounded by the *Semiconductor Tracker* (SCT), itself encased by the *Transition Radiation Tracker* (TRT). Combined, the three subdetectors have a length of 7 m, a diameter of 2.3 m and provide a relative momentum resolution of $\sigma/p = (4.83 \pm 0.16) \cdot 10^{-4}\,\mathrm{GeV}^{-1} \cdot p_\mathrm{T}$ [**?**].

The APD (Fig. 3.2) consists of three barrel layers and two endcaps. The endcaps consist of three disks each and are located on both sides of the interaction point in beampipe direction. This assembly yields three space points per particle on average, very close to the interaction point, and thus providing very good vertex reconstruction. The APD main components are 1744 identical sensor-chip-hybrid modules, totalling approximately $8 \cdot 10^7$ pixels. Each module contains a silicon sensor subdivided into 47 232 pixels, mostly $50 \times 400\,\mu\mathrm{m}^2$ in size. Each pixel is connected individually to one of 2880 analogue pixels, located on one of the 16 Front-End read-out chips (FE-I3) per module.

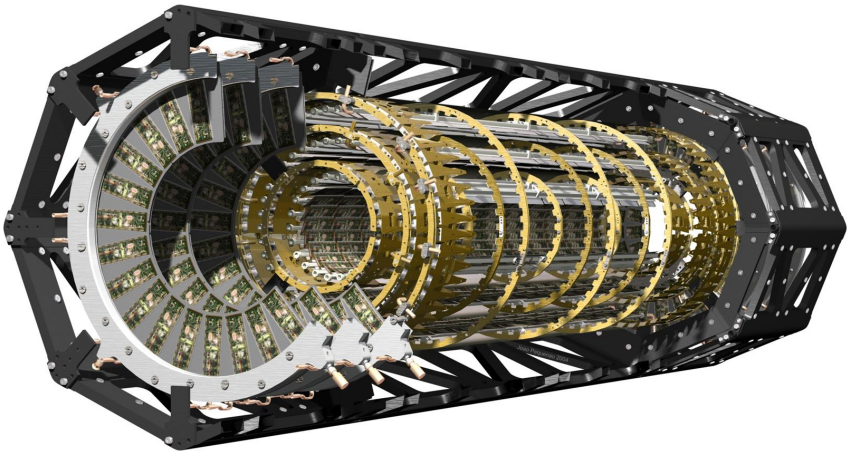


Figure 3.2: Engineering drawing of the original ATLAS Pixel detector in its global support frame, prior to the IBL upgrade.

The SCT consists of four barrel layers and two endcaps with nine disks each. The layers and disks are made of single sided p-in-n microstrip sensors glued back to back. They are rotated by 40 mRad to provide three-dimensional hit information, and are read out via $6.2 \cdot 10^6$ channels. This provides a resolution of 16 µm in the $R\phi$ plane and 580 µm in direction of the beampipe.

The TRT is made of straw tubes, arranged in parallel to the beampipe in the barrel layers, and perpendicular in the endcaps. It provides a resolution of 170 µm per straw. Each tube contains a sense wire and is filled with a gas mixture. Particles traversing the tubes ionize the gas mixture and are detected by collecting the produced free electrons at the sense wire. When light particles, such as electrons, cross the walls of the tubes photons are emitted. Measuring this transient radiation helps to distinguish these particles from heavier ones. Approximately 36 hits are recorded per particle, and read out via the TRT's 420 000 read-out channels.

All ID subdetectors are under the influence of the 2 T magnetic field created by the superconducting solenoid magnet surrounding the ID. It measures 5.6 m in length and 2.4 m across, and is supplied with a current of 7600 A.

3.1.2 The Calorimeter System

The calorimeter system is a two part system, designed to precisely measure the energy of particles leaving the ID, and to stop all particles except for the close to unstoppable muons and neutrinos.

The inner part of the calorimeter system is the *electromagnetic calorimeter*, a

sampling calorimeter made of lead absorbers and liquid Argon. The outer part is the *hadronic calorimeter*, also a sampling calorimeter, but made of steel absorbers and plastic scintillators. To stop almost all particles both calorimeters are approximately 25 radiation lengths thick.

3.1.3 The Muon System

Since the calorimeters cannot stop muons, they were encased with a third detector, the *muon system*. It performs unambiguous identification of muons and provides space points far away from the interaction point, allowing very precise track reconstruction when combined with the ID space points. To measure muon energy a magnetic field is generated, penetrating the muon system. It originates from a toroid magnet, consisting of eight superconducting air-coils and two endcaps, creating ATLAS' characteristic shape. The coils are 25.3 m long and extend from a radius of 9.4 m to 20.1 m, producing a magnetic field of 3.9 T peak strength. The endcaps extend from a radius of 1.65 m to 10.7 m, and produce a magnetic field of 4.1 T peak strength.

The muon system has four parts. *Resistive Plate Chambers* (RPC) in avalanche mode in the barrel regions, and *Thin Gap Chambers* (TGC) in the endcap regions, handle triggering. *Drift Tubes* (MDT), made of aluminium and filled with a gas mixture, and *Cathode Strip Chambers* (CSC), made of multi-wire proportional chambers, perform precision measurements. They provide a spatial resolution of approximately 50 µm.

3.2 Inner Detector Upgrades

Many parts of ATLAS receive periodic upgrades to ensure reliable and highly efficient data acquisition in increasingly harsh conditions. These conditions worsen with each LHC upgrade and are accompanied by years of intense radiation exposure. A first large upgrade of the ATLAS Pixel Detector was performed during the LHC shut-down in 2013–2015, when a new pixel layer was successfully inserted into the existing APD. The next large upgrade is planned for the long shut-down prior to the start of HL-LHC, and foresees the replacement of the entire Inner Detector with a new, all-silicon *ATLAS Inner Tracker* (ITk).

3.2.1 Phase I: Insertable B-Layer

The *Insertable B-Layer* (IBL) project was an effort to prepare the ATLAS Pixel Detector for the increased irradiation and occupancy resulting from the LS1 and LS2 LHC upgrades, so it could keep providing precision data at high efficiencies until deployment of the ITk. It resulted in the insertion of a fourth and innermost pixel layer into the existing APD, during the LS1 shut-down from 2013–2015 [**?**].

The IBL Design

In between the original APD and the beampipe existed an 8.5 mm radial gap, which was enlarged to 12.5 mm by reducing the radius of the beampipe [**?**]. The enlarged free space allowed the installation of said fourth pixel layer. The layer provides full coverage in the $R\phi$ plane and only small gaps in z direction, unavoidable due to the tight space.

The layer consists of 14 staves, equipped with 32 modules each. The staves are enclosed by the IBL Support Tube (IST), stabilizing the layer during installation and operation. All parts were thinned as much as possible to reach the target radiation length of $0.015\,X_0$ [**?**].

The majority of IBL modules are double-chip modules with one large n-in-n planar silicon sensor and two newly developed FE-I4 Front-End read-out chips [**?**]. The outermost modules of each stave are composed of 3D silicon sensors and four FE-I4 chips. A third type of module, a single-chip module with a diamond sensor, was developed for the IBL but not deployed due to changes in the IBL time table. They are thus used in the *Diamond Beam Monitor* (DBM) exclusively, a telescope-style detector system meant to provide high precision luminosity measurements. It allows testing of the new technology in a non-critical application [**?**].

The Read-out Electronics

A new generation of front-end electronics, the FE-I4 read-out chip, was developed to fit the conditions of the IBL. It employs several technological enhancements to be more radiation hard and operable under higher occupancies than the previous FE-I3 of the original APD. It contains per pixel analogue and digital circuits for processing and storing charge information from an attached sensor, and periphery circuits handling off-chip communication.

The FE-I4 (Fig. 3.3) is one of the largest ASIC designed for a HEP application to

date, measuring $18.8 \times 20.2\,\mathrm{mm}^2$ when diced [**?**]. This allows for a large 80×336 pixel array and several other size related advantages, such as an improved active over total area ratio. The pixel and feature size were shrunk to $250 \times 50\,\mu\mathrm{m}^2$ pixels and an inherently radiation hard 130 nm CMOS process. The increased density allowed increased complexity of the on-chip digital circuits and thus processing power. The new CMOS process and careful design increased the total ionizing dose (TID) radiation hardness to 250 MRad.

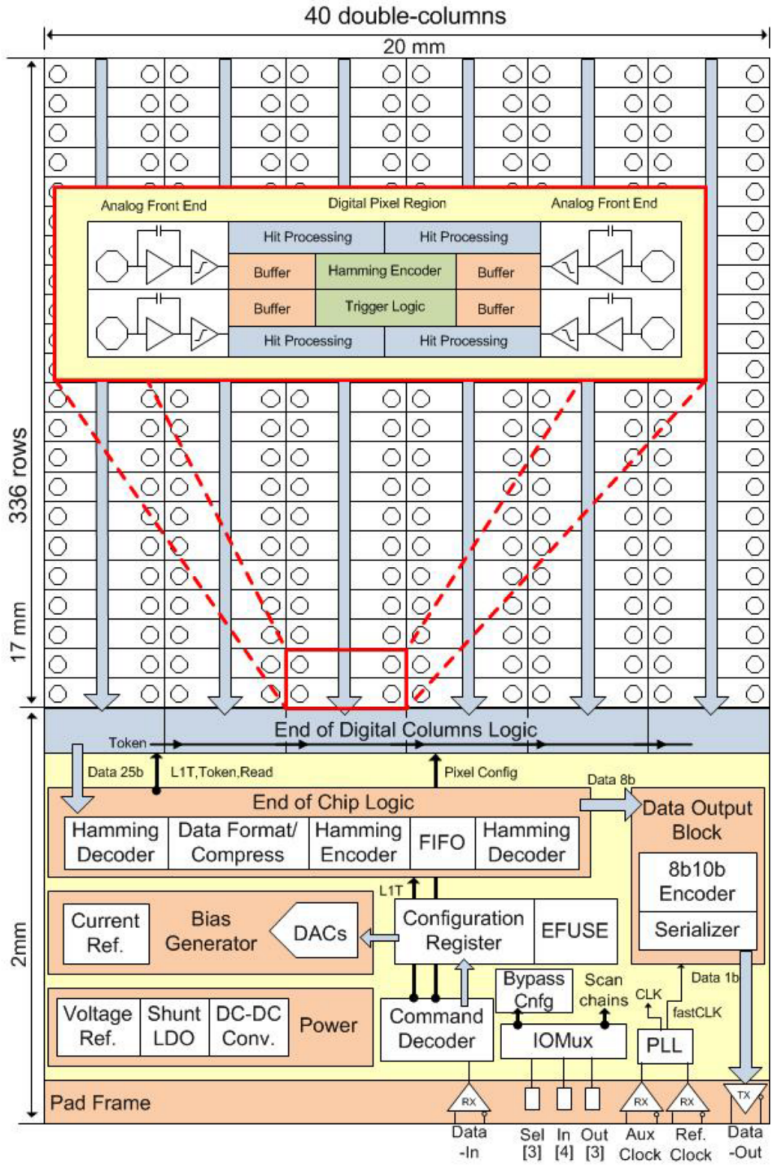


Figure 3.3: Block schematic of the FE-I4 read-out chip, enlarged are four analogue pixels connected to a 4-Pixel Digital Region [**?**].

A new pixel matrix architecture was employed, storing data locally at the pixel

level until triggered. This ensures FE read-out to be efficient up to a luminosity of $3 \cdot 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ at the IBL radius, sufficient up to the HL-LHC upgrade.

3.2.2 Phase II: ATLAS Inner Tracker

When starting operation around 2025 the HL-LHC will provide a levelled instantaneous luminosity of $5 \cdot 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$, far beyond the capabilities of the ID, even after the IBL upgrade. To ensure reliable operation during an envisioned 10 year run, providing a $2500\,\mathrm{fb}^{-1}$ data set [**?**], multiple upgrades and modifications are necessary.

The HL-LHC upgrade will increase the average number of interactions per bunch crossing from currently about 25 to 140. This dramatically increases the received radiation dose, occupancy and pile-up, neither of which can be handled by any subdetector of the current ID [**?**]. This motivates the complete replacement of the ID with a new Inner Tracker (ITk), an all-silicon design built on the experience gained from the ID.

The inner four or five ITk layers will be pixel modules, providing pattern recognition and vertex detection, the outer layers will be strip modules, allowing precise tracking at lower cost. This choice preserves and partially improves the tracking performance, while keeping the material budget at a minimum. Various layouts and pixel and strip technologies fulfilling the ITk requirements have been proposed and are currently under investigation (Fig. 3.4). Good candidates for pixel sensors are the planar and 3D silicon, and diamond sensor technologies, currently used in APD, IBL and DBM. 3D silicon is considered for the inner layers due to low depletion voltage after irradiation, and planar silicon for the larger outer layers, due to high yield and low cost. Additional technologies, such as active CMOS sensors, are tested as well, promising lower cost and smaller pixel sizes than traditional hybrid designs.

For high-speed detector read-out, matching the high luminosity and thus increased trigger rate, a new Front-End chip is necessary. At the inner layers data rates of $5\,\mathrm{Gbit}\,\mathrm{s}^{-1}$ per FE are expected, far beyond the capabilities of the current FE-I4. Testing of FEs delivering such rates is also beyond current test systems, making upgrades to them inevitable.

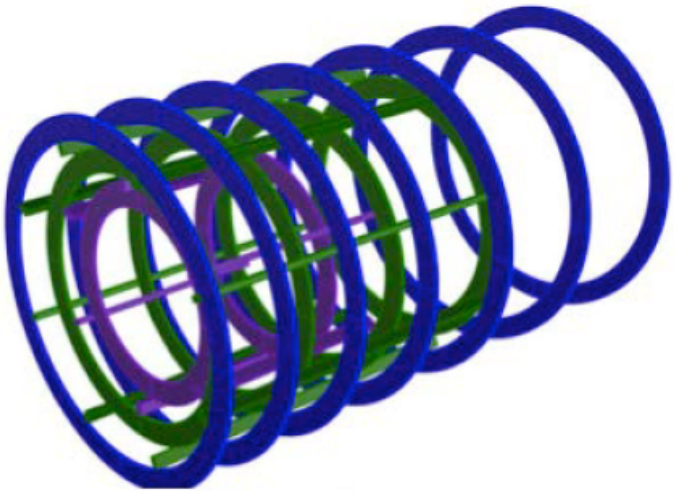


Figure 3.4: 3D view of the *open rings layout* considered for the ITk pixel endcaps, as opposed to conventional solid endcaps. The open rings promise better cable routing, cooling and easier construction [**?**].

3.2.3 Further Upgrades

A major upgrade to the trigger system is necessary to provide sufficient bandwidth for the HL-LHC hit rates. A two-stage hardware trigger will be used to reduce the initial rate of 1 MHz to 400 kHz, still more than current read-out electronics can handle. Therefore the calorimeter and muon system read-out electronics must be replaced as well as the pixel electronics. This will be used to provide the full calorimeter granularity and improved muon p_T resolution to the trigger system. The calorimeter itself will also be upgraded, especially the forward part, as will be the muon system and ATLAS computing.

4 Detector Read-out

During detector development and testing a portable, quickly installable and self-contained read-out system is needed. The large scale ATLAS read-out system does not match this description, so smaller systems, suitable for laboratory and testbeam environments, were designed. A widely used system for ATLAS Pixel Detector development is the USBpix system, currently capable of testing FE-I3 and FE-I4 read-out chips and matching sensors. In the sections below the core components of the system and a subset of the exchangeable parts are introduced, before examining the programmable logic provided on the main PCB in greater detail.

4.1 The USBpix Read-out System

The USBpix system [**?**] was developed to cater to the needs of a large community, involved in many different projects based around various pieces of hardware. To support as many of these devices as possible, both the USBpix hardware and software were designed to be modular and adaptable. The system subsequently found its way into laboratories and testbeam facilities, is used for testing front-end electronics and sensors, and for interfacing many supplementary devices, such as external trigger sources.

4.1.1 USBpix Hardware

The core component of the USBpix hardware is the *Multi-IO board*, handling communication between the host software, the device under test (DUT), and any supplementary devices. It connects to the host PC via a USB 2.0 port, to the DUT via one of several *adapter cards*, and to any other devices via LEMO or RJ-45 ports. The DUTs are placed on *carrier cards*, dedicated to a specific type and number of DUTs, which interface with a matching *adapter card*. An exemplary hardware assembly suited for testing a single FE-I4 is shown in Fig. 4.1.

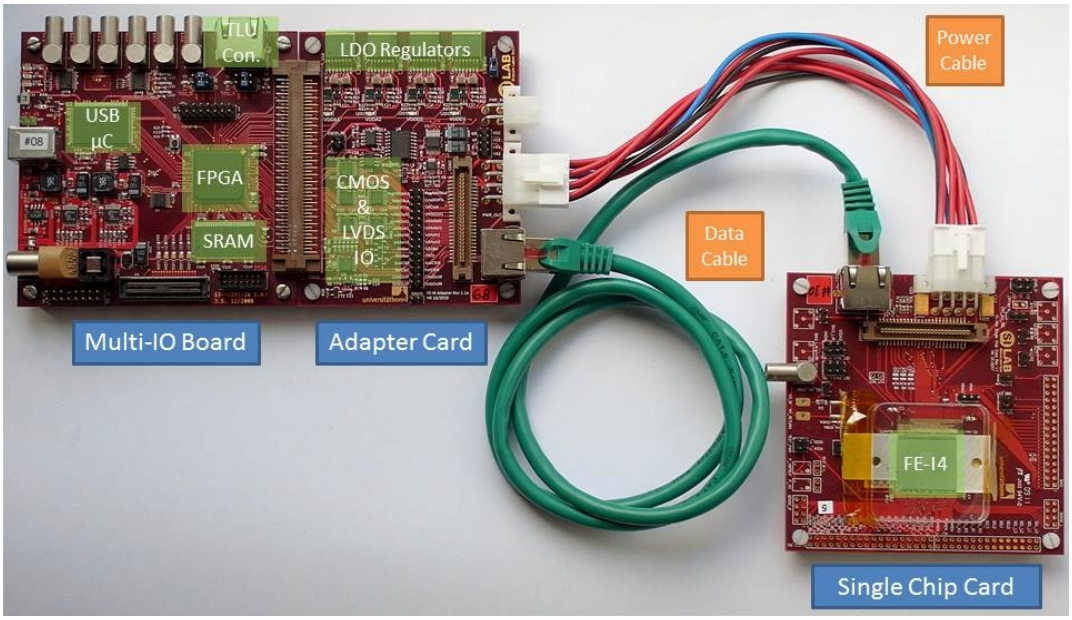


Figure 4.1: USBpix set-up for testing a single FE-I4: Multi-IO board with *Single Chip Adapter card* and attached *Single Chip Card* with FE-I4.

The Multi-IO Board

The *S3 Multi-IO board* [**?**] is equipped with a USB 2.0 capable *CY7C68013A* microcontroller (µC) [**?**], a *XC3S1000* Xilinx Spartan-3A Field Programmable Gate Array (FPGA) [**?**], and a *CY7C1061AV33* asynchronous static random-access memory (SRAM).

The µC handles all communication between the host and the on-board logic, via the USB Full Speed and USB High Speed protocols. Using USB Full Speed single bytes paired with a 16 bit addresses are transferred, used for setting and reading configuration and status registers. In USB High Speed mode large blocks of data can be quickly transmitted, used for sending data received from the DUT to the host. Time critical tasks, such as FE clocking, data recovery, data processing, and trigger logic, are handled on-board by the Spartan-3A FPGA (see sections 4.2 & 5). For data intensive tasks the FPGA relies on the 2 MB SRAM for increased storage capacity. Communication with all devices aside from the host is also handled by FPGA, via the aforementioned LEMO and RJ-45 ports. A block diagram showing the board layout is presented in Fig. 4.2.

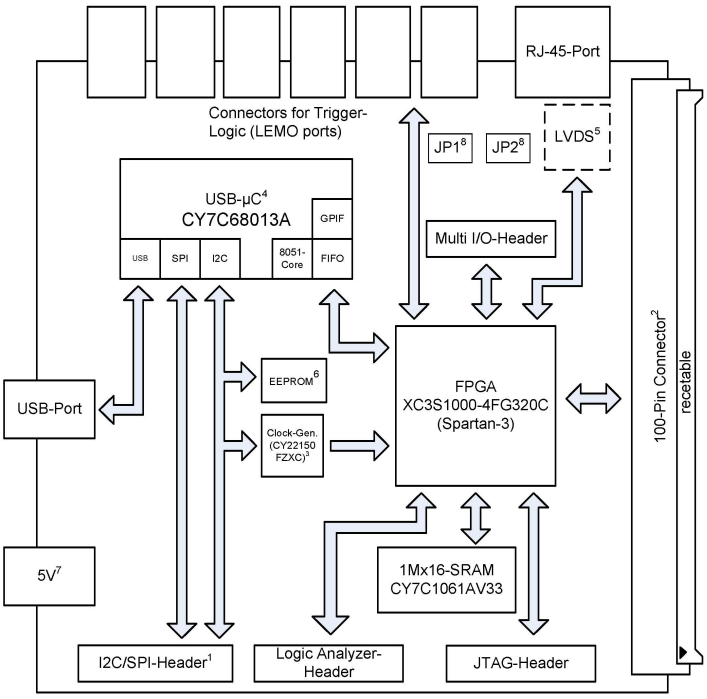


Figure 4.2: Functional block diagram of the S3 Multi-IO board, showing the µC, FPGA, SRAM and interconnects [**?**].

FE-I4 Adapter Cards

A range of *adapter cards* allows the Multi-IO board to communicate with various DUTs. Each card connects to the Multi-IO board via a 100 pin connector, and receives signals directly from the FPGA. The main task of most adapter cards is converting these single-ended Multi-IO/FPGA signals to the mostly differential signals used by the DUTs. Most cards provide pin headers for debugging signals and DUT specific additional functionality.

The standard card for interfacing a FE-I4 is the *Single Chip Adapter Card* (SCA), capable of passing signals between a single FE-I4 and the Multi-IO board. The *Burn-In Card* (BIC) is a more sophisticated model, allowing to connect up to four FE-I4s simultaneously. It still receives only a single pair of FE clock and command signals from the Multi-IO board, which it routes to all connected FE-I4s. The FEs return data via a separate data line per FE, which are all passed to the Multi-IO board individually and in parallel.

FE Carrier Cards

The DUTs are mounted on *carrier cards*, which route power and communication signals to the DUTs. The cards typically connect to a matching adapter card using an RJ-45 port and CAT cable. This form of connection allows them to be placed several meters away from the core parts of the USBpix system, e. g. in the path of a particle beam.

The counterpart of the SCA is the *Single Chip Card* (SCC), holding a single wire-bonded FE-I4. A BIC is usually paired with a *4-chip-module card*, holding four FE-I4s. Both carrier cards offer bias voltage ports to deplete any sensor that may be attached to the FEs. The SCC has a dedicated LEMO port for biasing, the 4-chip module card receives the bias voltage on the same 8 pin MOLEX as the FE supply voltages.

Front-End communication

The FE-I4 has three differential serial links, a *clock*, a *command*, and a *data line*. The clock line supplies the FE with a reference clock, needed for setting the bunch crossing id and counting charge and trigger information. The uni-directional command line transmits slow configuration commands and fast trigger commands to the FE. Slow commands usually contain an address and a data payload, and are used to access specific registers on the FE and set them to the provided value. The fast trigger command is a short bit sequence without payload, instructing the FE to transmit all stored data via the data line. Data transfer occurs in *frames*, typically consisting of one *data header* (DH), up to three *service records* (SR), and an arbitrary number of *data records* (DR), enclosed in *start of frame* and *end of frame* comma symbols.

The DH specifies the bunch crossing and trigger ids for all following DR. Each DR contains a pixel location and the charge information collected for that and an adjacent pixel, during the given bunch crossing. Charge is measured in *Time-over-Threshold* (ToT), which is proportional to the observed charge. Combining the ToT information of neighbouring pixels is called ϕ*-paring* and reduces the FE data rate, since pixel are often hit in pairs. The SRs contain status and error information. Each record is 24 bit long and send out in three, sequentially transmitted 8b/10b symbols.

4.1.2 USBpix Software

The USBpix hardware is controlled by the *STcontrol* software application. It provides a graphical user interface (GUI), using ROOT and Qt, for performing measurements and analysing obtained data. It builds on the functionality of a customized *ATLAS PixLib* library, a collection of *C++* classes originally used to access the ATLAS Pixel Detector read-out devices (RODs). The STcontrol PixLib implementation is supplemented by several hardware specific libraries for interacting with FE-I3s and FE-I4s via the USBpix hardware. Several abstraction layer and abstract classes allow adding additional hardware interface libraries, to easily widen the pool of supported DUTs. To work seamlessly within larger testbeam set-ups STcontrol integrates with the EUDAQ software suite [**?**] over a TCP connection.

4.1.3 Scans

The STcontrol *PixScan panel* offers a large number of scans. Among them are scans to test the electronics of an FE, calibrate its charge threshold, determine cross talk, gather data from an attached sensor, and many more. Two groups of scans can be distinguished by their underlying mechanisms. The larger group of *calibration scans* includes all scans relying on the internal charge injection of the FE. The second group measures charges injected into a sensor by an external source, typically a laser, particle accelerator or radioactive source.

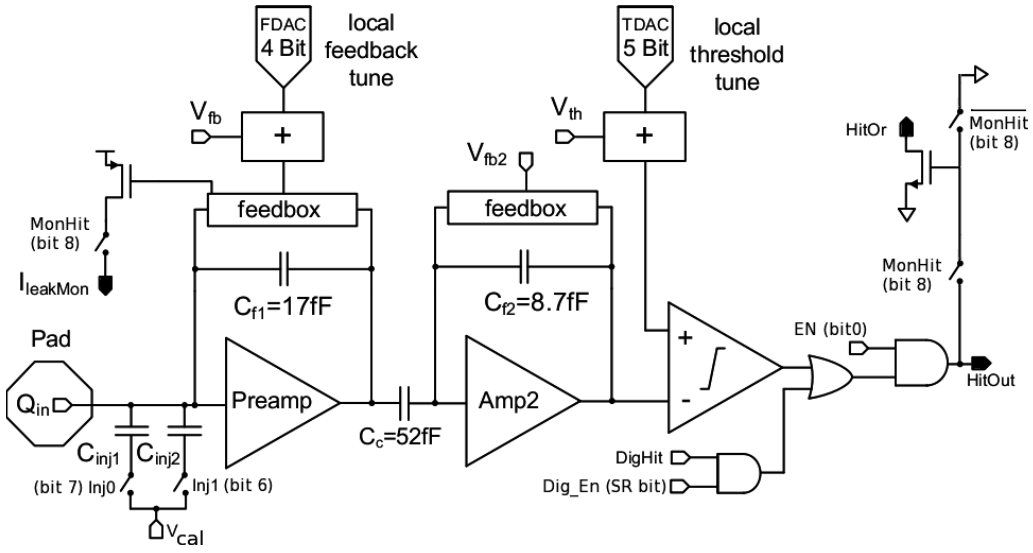


Figure 4.3: The analogue pixel electronics of the FE-I4, showing the V_{cal} (lower left corner) and *DigHit* (lower right) analogue and digital injection pads [**?**].

Internal injections can occur at the analogue or the digital pixel electronics, and are controlled by the V_{cal} and *DigHit* parameters [**?**] (see Fig. 4.3). The *Analog Scan* and *Digital Scan* instruct the FE to perform the injection corresponding to the scan title and record the registered hits in occupancy histograms. All other scans of the group rely on the same injection mechanisms and are thus derivatives of the *Analog* and *Digital Scan*. For example, should the users choose to perform a *Threshold Scan*, to determine the amount of charge necessary for a pixel to register a hit, the system performs a set of *Analog Scans*, injecting various charges, and calculates the threshold from the *hit-or-no-hit* information gained from the individual *Analog Scans*. To validate any scan of this group on the hardware and hardware interface level it is thus sufficient to verify the *Analog* and *Digital Scans*. When only altering the FE read-out, it would even be sufficient to perform just one of the scans, since even these two differ only in their injection parameters.

The second group consists of merely the *Source Scan* and its testbeam derivative. These rely on external charge injection and triggering. The data returned from the FEs is again the same, regardless of injection mechanism, but triggering and scan duration are very different for these scans. During calibration scans the USBpix system issues a predefined number of injections and generates a read-out trigger after each one. *Source Scans* know a large number of trigger mechanism, internal and external, which have to be translated and counted by the USBpix system. Scans can also run for many hours and produce vast amounts of data. Therefore this group has to be validated individually.

4.2 Programmable Logic

The Multi-IO board is equipped with a programmable logic chip, allowing swift system upgrades without the need for manufacturing and distributing new ASICs and PCBs. The *Xilinx Spartan-3A FPGA* chosen for the USBpix system can handle a wide range of logic functions and signalling standards, at clock rates up to 333 MHz. It can be reprogrammed at any time by pushing a new firmware file to the chip, making it a great choice for ever evolving, high frequency applications such as USBpix. The architecture of modern FPGAs involves a diverse set of specialized resources, requiring special design paradigms to be followed during firmware development, for optimal utilization.

4.2.1 FPGA Architecture

At the lowest level, FPGAs consist of a large array of interconnected transistors, often times several billion. They are arranged to form logic gates, many of which are combined to more complex structures, such as *Look-Up Tables* (LUTs), registers and multiplexers. The named structures are sufficient for implementing a wide range of logic functions, and capable of emulating many other structures. They have therefore become the main resources on most modern FPGAs,

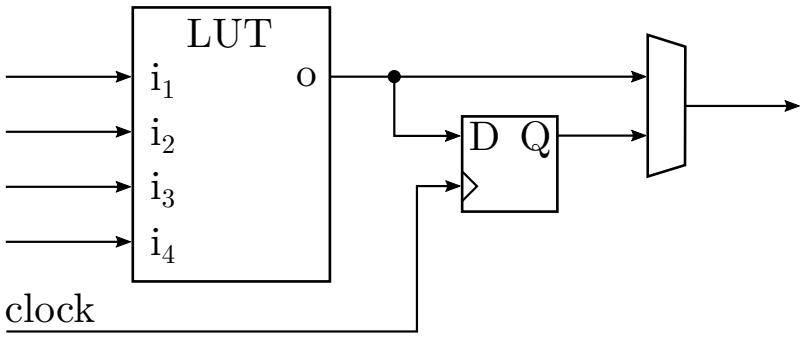


Figure 4.4: A LUT connected to a register and a multiplexer, representing the basic logic block common to most FPGAs.

A LUT maps n logic inputs to a single logic output, and is typically capable of implementing any logic function with n parameters. In combination with a register, usually a D-flip-flop, and a multiplexer it becomes usable for both synchronous and asynchronous logic. Therefore the LUT output gets connected to both the register and multiplexer. The register stores the output value and makes it available on its own output during the next clock cycle, which is connected to the other input of the multiplexer (see figure 4.4). This way the arrangement can be used for synchronous logic, when selecting the register output, and asynchronous logic, when selecting the LUT output at the multiplexer.

On the Spartan-3A the described cells are combined in pairs of two and supplemented with additional elements to form *slices*, shown in Fig. 4.5. The extra elements allow chaining of LUTs, to construct larger logic functions or structures like memory cells and shift registers.

Specialized Resources

The Spartan-3A contains five fundamental programmable functional elements [**?**], their placement on the FPGA is laid out in Fig. 4.6. As shown, most of the

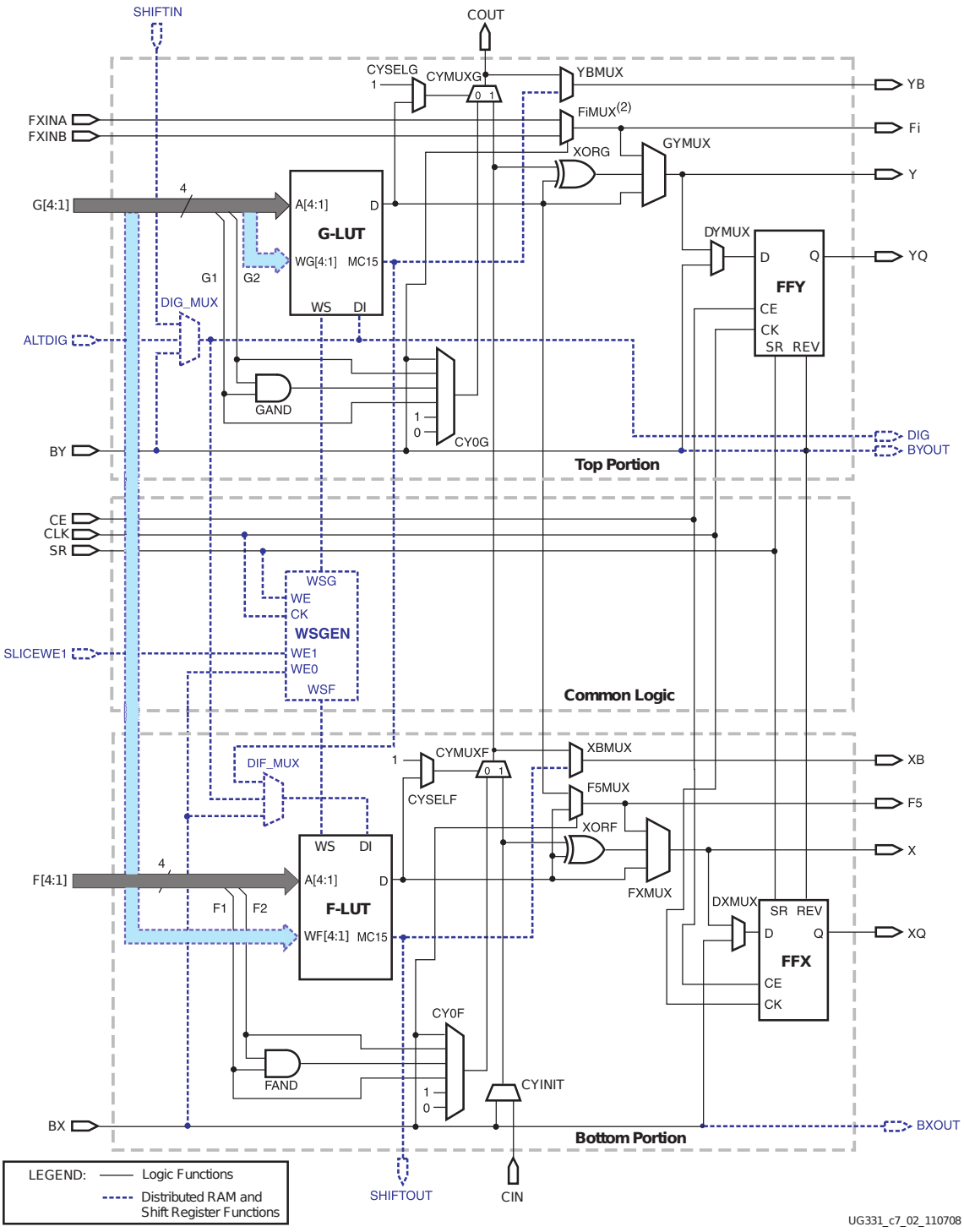


Figure 4.5: Simplified diagram of the contents of a Xilinx Spartan-3A slice [**?**].

Spartan-3A's area is taken up by *Configurable Logic Blocks* (CLBs), which combine four slices each, and implement most of the logic on the FPGA [**?**]. All LUTs are 4-to-1 LUTs and can thus be configured as 16 × 1 memory cells or 16-bit shift registers.

Knowledge of these exact numbers is important to utilize the CLBs optimally in the design.

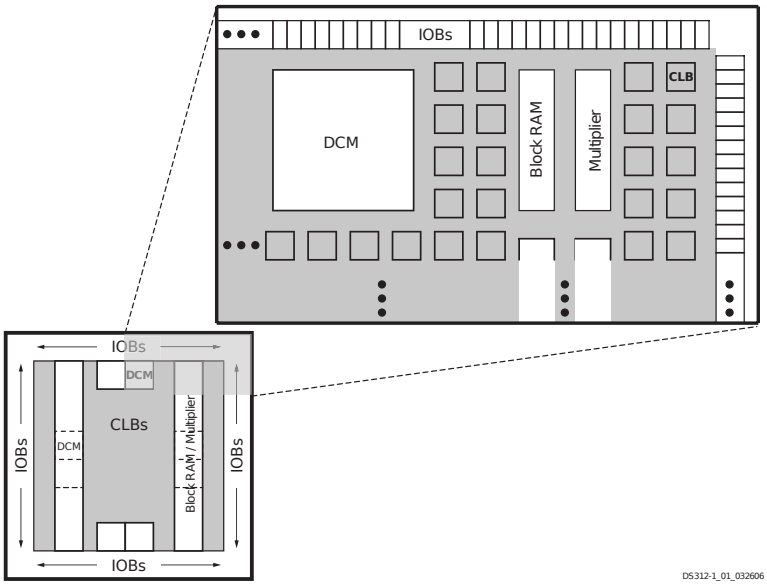


Figure 4.6: Architecture of a Xilinx Spartan-3A, shown are the fundamental building blocks and their locations [**?**].

Off-chip communication is handled by *Input/Output Blocks* (IOBs), arranged along the edges of the Spartan-3A. These perform encoding and decoding of a large number of signalling standards, allowing to connect the FPGA directly to most other system components, without the need for external decoders. The IOBs are capable of bi-directional tri-state operation, necessary for using the USB and SRAM data channels. They also permit double data rate operation (DDR), allowing SRAM writes on consecutive clock cycles.

The Spartan-3A provides two columns of 18-Kbit dual-port RAM blocks, allowing efficient implementation of *first-in-first-out queues* (FIFOs). These are often used for buffering data on-chip for a small number of clock cycles, and passing data from one clock domain to another. Both are very common tasks and handled by FIFOs with either the same or independent read and write clocks. Using the block RAM, *first-word-fall-through* (FWFT) FIFOs can be implemented as well. These make the first word pushed in immediately available on their output, enabling designs with minimal delay.

Another major resource are *Digital Clock Managers* (DCMs), necessary for proper generation and distribution of clocks to synchronous logic. Ideally a clock signal would be supplied to all connected logic at the exact same time, impossible due

to routing delays on the large FPGA area. These unwanted delays can be compensated, because clocks are periodic signals, and only the arrival of *an* edge at a predictable time is important, not the arrival of any specific edge. Therefore the DCMs can calibrate themselves to simulate instantaneous transmission of a clock to all registers via use of a *feedback clock*. DCMs can also generate new clocks by dividing, multiplying and phase shifting an input clock. Designs often require multiple clocks, when communicating with devices operating at different frequencies, such as the µC and DUTs, or when oversampling is necessary to recover data from external sources.

Interconnects

The FPGA contains several networks of traces for routing clock, configuration and logic signals. To receive or send these signals all functional elements connect to a subset of the networks via switch matrices, associated with the individual elements.

The clock nets are carefully designed to have the same delay, called *clock skew*, at as many endpoints as possible. To achieve this a form of tree structure is used, first distributing the signal evenly to all areas of the FPGA, before connecting to the individual registers. The large fanout requires many stages of amplification, also evenly distributed across the FPGA. To allow optimal operation of the clock net great care must be taken, to only use proper clock signals to trigger synchronous logic. Otherwise logic signals may be inserted into the clock net, hindering performance.

Separate nets are provided for routing logic signals. Flexibility in routing is provided by transistors in the nets and the switch matrices of the functional blocks. These get set, to pass through or block signals, during configuration of the FPGA. Signal routing introduces delays, which must be considered in any design, just as the delay from the logic operations. Their sum needs to be small enough to fit within one clock period. Routing delay can be optimized by placing connected elements as close to one another as possible, and thus minimizing the length of signal paths.

4.2.2 FPGA Design

When designing an FPGA configuration it is obviously not feasible to set the state of each transistor or even logic gate individually. Instead, designing is done on a higher abstraction layer, usually the *Register Transfer Level* (RTL). On this level,

registers exist as the primary storage for logic levels. At each rising edge of a clock, a set of registers is read. A logic computation then generates new logic levels from the read registers' states. This result is then stored in another set of registers, and made available at the next rising clock edge. The source, destination and logic function are commonly described using a *Hardware Description Language*, such as *VHDL* (Very High Speed Integrated Circuit Hardware Description Language) or *Verilog*. These languages allow control of hardware specifics, such as signal timing and asynchronous and synchronous parallel command execution.

Designs are split in *modules*, or *entities*, designed for specific tasks, and connecting to one another using a specified set of input, output and bi-directional ports [**?**]. The body of an module contains asynchronous signal assignments, synchronous processes, and instances of other modules. This level of abstraction simplifies the design process, but obscures the underlying complexity and requirements of the hardware. Especially the set-up and hold times of signals before and after a clock edge, necessary for safe operation, are hidden from the designer. They need to be specified in dedicated *constraints files*, read by the place and route tools when mapping and routing a design on an FPGA. The tools inform the designer if any constraint could not be met for the design at hand, requiring design optimization or even careful, manual placement of registers.

5 USBpix Configuration

Before any update to the USBpix system can be discussed the current configuration must be examined, and room for improvement identified. Then the feasibility of the chosen upgrade path needs be shown, and finally the implementation described.

Since the goal of this thesis is the implementation of a new read-out scheme, the following discussion focusses solely on the read-out related parts of FPGA firmware and USBpix software. The new read-out implementation is supposed to allow data transfer to the host PC in a continuous stream, without the current need for frequent stopping of ongoing scans. It is motivated by an expected large increase in average data acquisition rate. Also unified data handling for all measurements is desired, enabling the system to store the complete and unaltered raw data regardless of scan type.

5.1 Stable USBpix discussion

The modifiable parts of USBpix are the host software, the microcontroller firmware, and the FPGA firmware. Since the microcontroller firmware is closed source, and already provides modes for bulk data transfer and addressable single byte transfer, it will remain as is. *STcontrol* will require only minute changes, since it is well separated from the actual scan processes by the software abstraction layers. The underlying *PixLib* and *USBpixI4dll* libraries on the other hand require major changes to read-out functions and procedure, while the configuration functions can remain unchanged. The same holds for the FPGA firmware, since only the last two of the major blocks *USB Full Speed interface*, *clock generation*, *control strobe*, *configuration state machine*, *trigger id control*, *read-out block*, and *SRAM control*, are directly involved in board read-out only these need modification. A sketch of the current read-out scheme is shown in Fig. 5.1, offering some guidance during the following discussion.

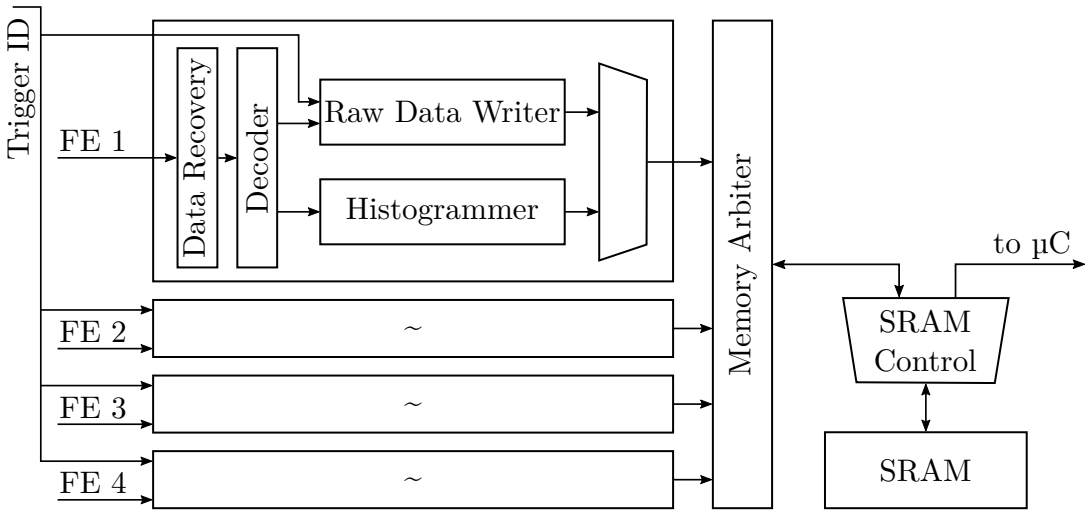


Figure 5.1: The read-out chain of the stable USBpix firmware starts with a dedicated *data recovery* module for each FE. The data streams are passed through decoders and a hit processor and are combined in the *memory arbiter* module. The data is then buffered in the SRAM and send to the host via the µC.

5.1.1 Data Recovery & Decoding

The serial data streams received from the FEs are first sent to the data recovery and decoder blocks, for synchronization, deserialization and decoding. These actions are performed for all data streams in parallel by multiple instances of the described blocks.

The data recovery modules search for edges, i. e. logic level transitions, on the FE data link. Whenever such an edge is detected the data stream is sampled to retrieve the currently send bit. In between the edge and the sampling point a short amount of time, optimally half a clock cycle, is waited to ensure the link settled on a stable level. If no new edge is detected for a full clock cycle after the previous sampling point, the next sample is taken at the beginning of the clock cycle following the latest sampling. This approach requires at least three samples per bit, with more samples improving the granularity of the edge detection. Using an even number simplifies the design, hence four times oversampling was chosen for implementation. The needed clock frequency is achieved by using two 90° phase shifted clocks and their inverse, in combination delivering the 640 MHz clock needed at maximum FE data rate. Further details are found in Xilinx Application Note 224 [**?**].

The decoder receives the serial bit stream from the data recovery and pushes it into a shift register. The register is constantly checked for FE protocol commata, marking the beginning of a new word. Once found the following data is aligned and

the packages send to the *8b/10b decoder*. The returned bytes are checked for *start of frame* markers, and the following bytes combined into 24 bit FE protocol words and flagged with a *valid* bit.

The described *data recovery* and *decoder* modules handle parallel data streams from up to four FEs very efficiently and reliably. The resulting data format of fully decoded FE protocol words with *valid* bit is well suited for the subsequent processing and transmission, so both modules can remain unchanged.

5.1.2 Read-out Modules

Two types of read out modules are implemented, one for on-board histogramming during calibration scans, and one for writing raw data during source scans. The *histogrammer* module offers two modes, one for creating per pixel ToT, and one for creating per pixel occupancy histograms. Since the occupancy histogram is simply a ToT histogram with all ToT bins combined in the first bin, both histograms are handled by the same module. Both module types create memory commands, which are send to the *memory arbiter* for execution. The arbiter applies the commands to the external SRAM, since the FPGA does not provide sufficient memory capacity for handling such data intensive tasks on-chip.

The commands begin with a single bit, identifying the type of command and thus triggering either the arbiter's *add* function (histogram memory command) or a *write* function (raw data writer memory command). The second bit marks a command as valid and ensures every command is executed only once. The next 21 bit are the SRAM memory cell address the command is to operate on, while the last 8 bit contain the data, which is either to be added or written to the given memory cell.

The *histogrammer* extracts the pixel location and both ToTs from each data record. It then computes the memory addresses corresponding to the histogram bins the hits are to be added to, and issues a memory command for each valid hit. The *raw data writer* performs no analysis of the FE data, instead it just adds sequential addresses to the words and passes them along. Additionally, it merges any trigger ids it receives into the FE data streams. To keep the multiple FE data streams separate, the *raw data writer* assigns one SRAM segment to each FE and matches data and addresses accordingly. Once a segment is full, the data writer raises a flag and the current measurement halts.

Neither of these read-out modules is suited for a unified continuous read-out. The *histogrammer* discards all data after extracting a selected piece of information and

is thus out of the question. The *raw data writer* preserves all data, but lacks the memory management capabilities required for continuous read-out. Adding them to the existing module would be cumbersome and put many constraints on the implementation of downstream logic and interfaces. Removing both modules and passing the FE words along on a 24 bit parallel bus allows for a much more resource efficient design.

5.1.3 Memory Arbiter

The *memory arbiter* merges the memory commands received from the up to four read-out channels into one command stream and executes the *write* and *add* commands on the SRAM. Since an *add* requires reading the current cell value from the SRAM, adding the received byte and writing the new value back to the SRAM, the arbiter implements a bi-directional SRAM interface. An analysis of read and write frequencies required for four FEs at maximum data rate showed the necessity for grouping read and write operations, and performing them at a finely tuned SRAM clock [**?**].

The first stage of the arbiter consists of four FIFOs, buffering the memory commands from one channel each, until they are merged into the common arbiter input FIFO. They simultaneously pass the commands into the SRAM clock domain. Merging is performed in two stages, to meet timing at the high SRAM clock frequency. *Raw data writer* commands received by the arbiter are passed to the *operations FIFO* and executed. *histogrammer* commands are held in the input FIFO until the arbiter begins a read cycle. Then the address from the first command in the FIFO is read, the retrieved data stored in the *read FIFO*, and the histogramming command stored in the *operations FIFO*. The process repeats until the read cycle is over. Then the data payload of the first command in the *operations FIFO* is added to the first data byte in the *read FIFO*, and the calculated byte written to the address given by the histogramming command.

The merging stage of the arbiter will remain necessary for any read-out scheme, since only one SRAM data bus exists, onto which four read-out channels have to be merged. Without the *raw data writer* merging the trigger number into the FE data streams trigger merging will have to be handled at the arbiter stage. Without on-board histogramming no high frequency switching between read and write operations will occur, allowing to operate the modified arbiter at a more moderate clock frequency. The only entity reading the SRAM will be the host software, which

performs reads in bulk and with comparatively long time intervals in between. This also removes the need for the *read* and *operations FIFOs*, and the two-stage merging, and a two-way SRAM interface.

5.1.4 SRAM control

All SRAM interface signals are handled by the *SRAM control* module. When SRAM access is requested by the memory arbiter, the module mostly provides tri-state buffers for the bi-directional SRAM data bus. When accessed by the host via the USB High Speed interface, it becomes responsible for making the requested SRAM data available on the USB data bus. The *High Speed* interface provides no address signal, so the start address for a read operation is delivered to the FPGA via the *Full Speed* interface beforehand, and must be decoded by the *SRAM control*. Whenever a USB read is performed, the *read strobe* signal is raised for one clock cycle by the µC, causing the *SRAM control* to pull the data from the next SRAM address and make it available on the USB data bus. When the host is not reading the USB bus, the *SRAM control* holds it in a high impedance state. Additionally, the module handles domain crossing between the SRAM interface and USB interface, via several sets of registers.

The *SRAM control* requires significant changes to handle continuous read-out. It will have to keep track of the SRAM addresses last written to and read from, and buffer incoming FE data while SRAM writes are blocked by SRAM read issued by the host. It also needs to calculate the amount of buffered data, so the host can query the number of read strobes to issue.

5.1.5 PixLib & USBpixI4dll

All scans are initiated from a *control thread* within the *STcontrol* application. The thread keeps running during the entire scan process, printing status information, and aborting the scan should the user choose to do so. After the scan the thread performs any necessary post-processing of the scan results. The specific actions taken upon starting a scan depend on the type of scan, which is either *calibration scan* or a *source scan*.

Calibration scans require frequent interaction with the FE, to control the injection mechanism and alternate the pixels receiving the injections. Therefore a dedicated *scan thread* is started, performing all necessary actions. Once the scan is done

the finished histogram is retrieved from the hardware, copied to the corresponding histogram object and presented to the user, upon which the *scan* and *control threads* return.

The *Source Scan* is performed without an additional thread, since it requires no interaction with the FE once a scan was started. The read-out system however needs attention, since it relies on the host software for instruction, when to read out the SRAM. Among the status information pulled from the board by the *control thread* is the fill level of the SRAM. Once received, it is compared to a threshold defined in the scan parameters, and, if matched, the instruction to hold the scan and retrieve all data from the SRAM is issued. The data is transferred byte by byte to an array equal to the SRAM in capacity. The bytes received are recombined to FE protocol words and triggers numbers and stored on disk. The hardware is then reset and the scan resumed, until the threshold is reached again.

The start scan and control procedures are mostly independent of the read-out and can therefore remain unaltered. Only the conditional statement, triggering read-out based on the SRAM fill level during *source scans*, must be removed. The function reading the SRAM needs to be modified to be able to read only a select portion of the SRAM. Additionally, some mechanism must be implemented to periodically call the function during all scan types, and to perform processing of received data in parallel, so data polling is not slowed by data processing.

5.2 Stopless USBpix configuration

After identifying the parts of the USBpix system, needing changes to achieve a higher average data acquisition rate and unify the data processing across scan types, the feasibility and implementation of the required improvements can be analysed and laid out. Since the changes involve mainly the read-out block of the FPGA firmware and the *PixLib* and *USBpixI4dll* libraries the discussion will again focus on these parts.

On the host side the functions for stopping the measurement and reading the entire SRAM are replaced with two threads, one of which constantly polls data from the board, while the other processes the available data. The use of a dedicated thread for data polling allows near continuous read-out of the USBpix hardware and thus minimizes the risk for buffer overflows on the board.

On the FPGA side the read-out modules are replaced with data buffers and id

taggers. The taggers implement a revised data-to-FE matching scheme, which replaces the no longer applicable *id-by-SRAM-address* scheme. The *memory arbiter* is replaced by a *data arbiter*, and the *SRAM control* module by an *SRAM FIFO* module. The *data arbiter* is much simpler than its predecessor, since it does not handle direct memory access, but only selects the data packet which is to be sent to the *SRAM FIFO* next. The *SRAM FIFO* module then handles memory access and management, by implementing a circular buffer in the SRAM, a common approach for streaming applications. Fig. 5.2 presents an overview of the proposed new scheme.

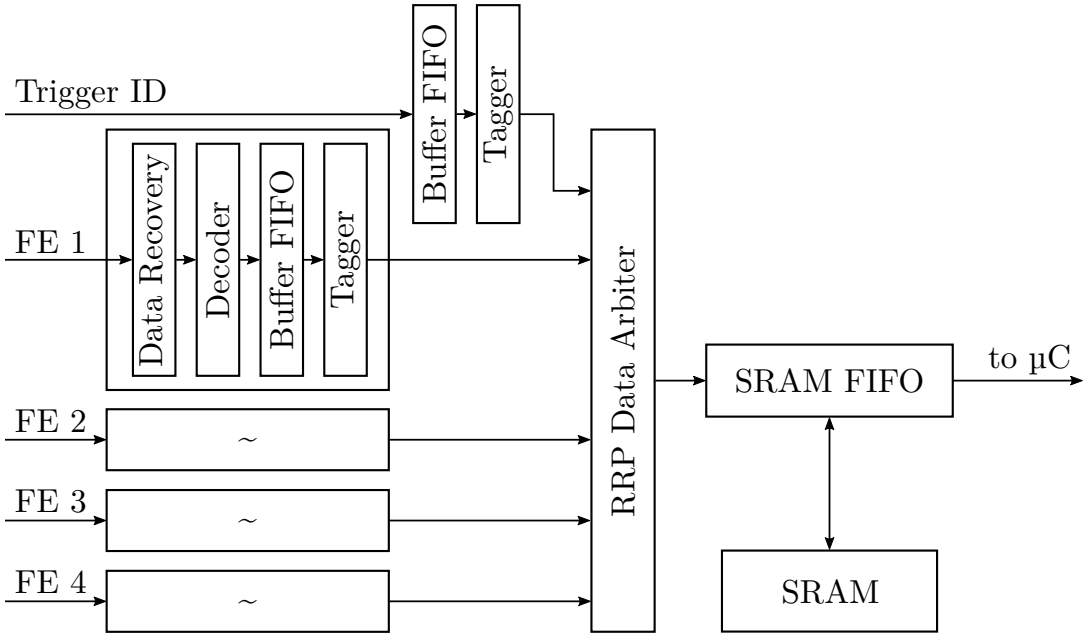


Figure 5.2: The proposed USBpix firmware keeps the data recovery blocks, but replaces the processor blocks with buffer and tagger modules. The memory arbiter is split into a round-robin with priority data arbiter and the SRAM FIFO module, which absorbs the SRAM I/O multiplexer.

Feasibility Study

The USBpix system needs to read out four FE-I4s in parallel at most, which can send data at a maximum clock frequency of 160 MHz each. The 24 bit FE protocol words are split in 8 bit words and 8b/10b encoded (see section 4.1.1), resulting in 30 bit sent sequentially for each word. This means at peak data rate the USBpix

system receives

$$160\,\frac{\text{Mbit}}{\text{s}} \times \frac{1}{30}\frac{\text{word}}{\text{bit}} \times 4\,\text{FEs} = 21.4 \cdot 10^6\,\frac{\text{words}}{\text{s}} = 512\,\frac{\text{Mbit}}{\text{s}}\,. \tag{5.1}$$

The system is connected to the host PC via USB 2.0 and capable of Full Speed and High Speed transmissions. The faster High Speed specification is defined with a bus rate of $480\,\text{Mbit}\,\text{s}^{-1}$. This rate results in a maximum effective data rate of $424\,\text{Mbit}\,\text{s}^{-1}$, due to encoding and transfer protocol overhead [**?**]. The data rate is further limited by the USBpix implementation of the specification. The microcontroller on the Multi-IO board operates at 48 MHz and issues a read strobe to the FPGA every three clock cycles, reading 8 bit in parallel at each strobe. This results in a data rate of

$$\frac{1}{3}\,\text{strobe} \times 48\,\text{MHz} \times 8\,\frac{\text{bit}}{\text{strobe}} = 128\,\frac{\text{Mbit}}{\text{s}}\,. \tag{5.2}$$

The calculations show that simultaneous, continuous read-out of four FEs at peak data rate is not possible with the available hardware. Especially considering, that the calculated transfer rate is only theoretical and in practice diminished by other transfers occurring on the USB bus and host software processing times. Fortunately the peak data rate is only seldom reached and only for very short times, if at all, during real world operation. It is also unlikely, that all four FEs reach their maximum rate at the same time. Thus simultaneous, continuous read-out remains possible, as long as sufficient storage and mechanisms for data buffering are included in the design.

The maximum sustainable trigger rate for source and testbeam measurements can be obtained by estimating the maximum number of pixels to read out for each event. The number of pixels hit per event is usually a single digit number. When working with a bad module about one hundred noise hits must be added on. Thus two hundred hit pixels resulting in two hundred data records accompanied by 16 data headers and a few service records per FE per event is a safe upper limit. This yields a worst case trigger rate of

$$\left(128\,\frac{\text{Mbit}}{\text{s}} \times 0.9\right) \times \frac{1}{220}\frac{\text{event}}{\text{words}} \times \frac{1}{4\,\text{FEs}} \times \frac{1}{24}\frac{\text{word}}{\text{bit}} \approx 5.5\,\text{kHz}\,, \tag{5.3}$$

when working with four noisy modules. A realistic calculation with one noisy and

three good modules yields a far more satisfying sustainable trigger rate of

$$\left(128\,\frac{\text{Mbit}}{\text{s}}\times 0.9\right)\times\left(\frac{1}{220}\frac{\text{event}}{\text{words}}\times\frac{1}{1\,\text{FE}}+\frac{1}{25}\frac{\text{event}}{\text{words}}\times\frac{1}{3\,\text{FEs}}\right)\times\frac{1}{24}\frac{\text{word}}{\text{bit}}\approx 16.3\,\text{kHz}\,. \tag{5.4}$$

The same calculation for a single good module gives a rate of

$$\left(128\,\frac{\text{Mbit}}{\text{s}}\times 0.9\right)\times\frac{1}{25}\frac{\text{event}}{\text{words}}\times 1\,\text{FE}\times\frac{1}{24}\frac{\text{word}}{\text{bit}}\approx 192\,\text{kHz}\,. \tag{5.5}$$

The calculated worst case trigger rate is achieved in some testbeams, while the realistic rate lies beyond observed rates [**?**]. The single module rate lies way beyond the rates achieved in any testbeams, but still gives a handy upper limit for the multitude of single chip laboratory set-ups. The calculations show the proposed upgrade surpassing the real world requirements, and thus support its feasibility. For a comparison to the current stopping read-out implementation and experimentally determined real world performance refer to chapter 6.

5.2.1 Read-out Module

The first two blocks processing incoming FE data, the *data recovery* and *decoder* described in section 5.1.1, efficiently turn unsynchronised serial bit streams into parallelized FE protocol words with accompanying *valid* bits. This data needs to be buffered, moved to the microcontroller clock domain and tagged with an *id byte*, before it can be passed to the *data arbiter*. These tasks are performed by a new read-out module, shown in Fig. 5.3. An instance of the module is created per FE and trigger channel, to simultaneously apply the operation to all data streams.

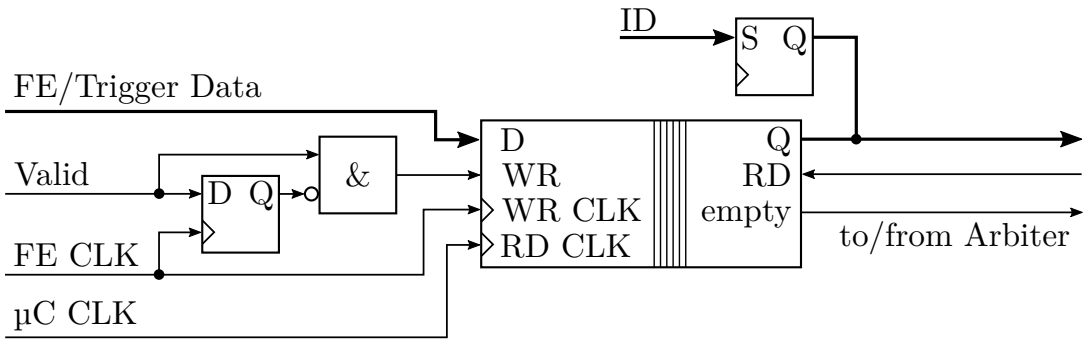


Figure 5.3: Buffering and clock domain crossing is achieved by using a FIFO with independent read and write clocks. The D-flip-flop and AND gate on the *write* port (WR) ensure every word is written only once.

Buffering and clock domain crossing are achieved most easily using a FIFO with independent read and write clocks. A first-word-fall-through FIFO (FWFT FIFO) was chosen, to minimize the delay introduced by the module, and simplify the downstream logic. The received data and *valid* bit are synchronized to the 40 MHz FE clock, which thus serves as the FIFO write clock. Since the microcontroller, and subsequently the USB bus, operate on a 48 MHz clock this clock is chosen for the remainder of the read-out chain and routed to the FIFO read clock port. The data is directly fed to the FIFO, since any data on the input is ignored as long as no *write* signal is received. The write command is generated from the *valid* signal, by feeding it to a register, and the inverted register output and the *valid* signal itself to an AND gate. This ensures that the *write* signal is only HIGH for one clock cycle per *valid* signal, without delaying the signal. This way the *valid* bit still marks the data on the correct clock cycle, and every incoming FE word is ensured to be written to the FIFO only once.

At the output of the FIFO a byte with constant bit pattern is added to the data. The pattern uniquely identifies the words read from this read-out module instance, and is later used to match the data to the different FEs or identify trigger words. Each trigger word is tagged with a 0x80, each FE word with a (0x01 + FE_ID). Adding the id after the FIFO keeps the width of the FIFO at a minimum. The FIFO depth is rather unimportant, since each read-out module is checked for data at least every sixth µC clock cycle, and data arrives at maximum every seventh FE clock cycle. It can therefore be chosen to match the remaining resources on the FPGA.

5.2.2 Data Arbiter

The data supplied by the five read-out modules must be muxed to the single data bus of the downstream module. Valid words on the arbiter output must be marked as such, and the downstream module needs to be offered a way to hold the arbiter when it cannot digest further data. Upstream communication to the read-out modules must include *read* and *data available* signals. A schematic of the arbiter is shown in Fig. 5.4.

A *round-robin* schedule was chosen for input multiplexing, since it minimizes the risk of buffer overflows on any given channel. A priority option was added to the *trigger number* input, because the host software expects the trigger number of a given event to always arrive before the FE data of the same event. The *round-robin with priority* (RRP) block thus always checks the trigger input first, and, if no

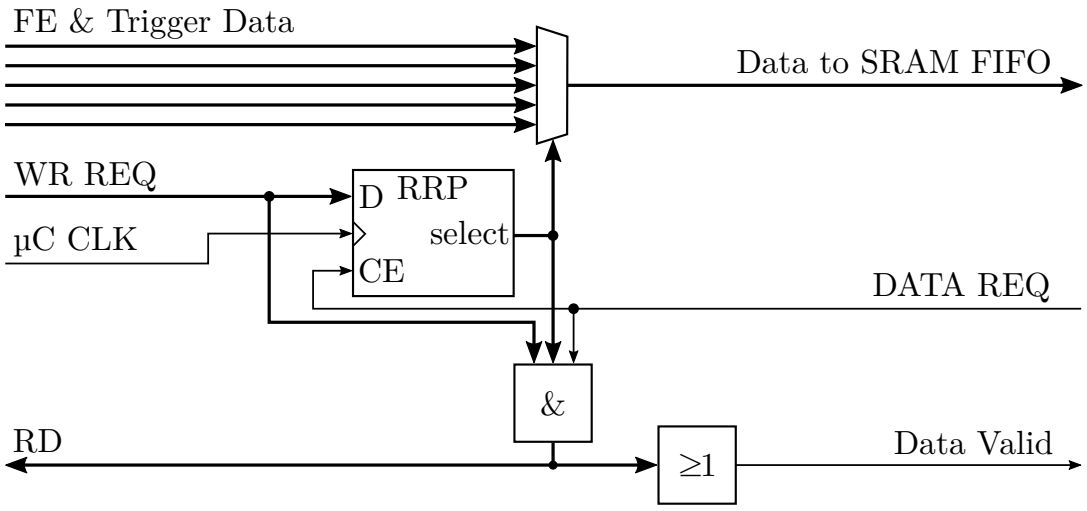


Figure 5.4: The arbiter muxes multiple input data buses to one data output bus using a round-robin scheme. Priority is given to the trigger data bus.

trigger word is available, proceeds to sequentially checking the FE inputs, starting with input following the one selected last. The input of the RRP block is composed of the five inverted *empty* signals of the read-out modules. These signal valid data at the output of their respective FIFO and thus function as *write request* signals. The RRP block is enabled by a *data request* signal from the downstream module and put on hold when this signal goes LOW. While enabled the block iterates over all input channels following the aforementioned scheme and sets a *select* signal to the first channel with a *write request* it finds.

The *select* signal is passed to the multiplexer, choosing the matching input for pass-through, and to the upstream *read* and downstream *valid* outputs of the arbiter. The upstream port passes the information which channel was selected to the read-out modules, and the corresponding module clears the read data from its FIFO. The downstream module has no use for this information, so the bits of the *select* signal are ORed to a single *valid* bit. By ANDing the *select* signal with the *write request* and *data request* signals, before passing it to the outputs, the *read* and *valid* signals are guaranteed to be HIGH for only one clock cycle per valid data word. As long as data is requested and available, the *select* signal takes care of this. Should one of the conditions no longer hold, the corresponding signal does so. Raising the *read* signal for just one cycle is necessary to not skip any words at the read-out module FIFOs, and raising the *valid* signal for just one cycle to write each word only once.

5.2.3 SRAM FIFO

The SRAM FIFO module is a more delicate and complex affair than the upstream modules. It needs to manage simultaneous read and write requests, adhere to the SRAM interface timings, and communicate various status signals to the host software. Additionally, it needs to break down the 32 bit data words, received from the arbiter, to the 16 bit SRAM memory cells, and finally the 8 bit USB bus. Since correct timing of the SRAM and USB interfaces is crucial for successful implementation, these requirements are discussed in detail, before presenting the implementation of the SRAM FIFO.

SRAM Timing

The SRAM data sheet specifies a minimum read and write cycle time of 10 ns [**?**], close to half a clock period of the µC 48 MHz clock. Since only one such operation needs to be performed per µC clock cycle, these constraints are easily met.

More careful investigation of the write cycle shows, that the *address* signal should be provided at the same time or before the *write* signal, and the *write* signal before the *data* signal. For the latter two, a time difference $t_{\mathrm{HZWE}} = 5\,\mathrm{ns}$ is given. This is the maximum time it takes the SRAM to switch its bi-directional *data* bus to a *high impedance* state, after the *write* signal was received. If the *data* bus is driven before the switch occurred, the SRAM and FPGA ports short-circuit and potentially damage the devices. After this time the *data* bus can be safely written. It then needs to be held steadily for at least $t_{\mathrm{SD}} = 5.5\,\mathrm{ns}$ for the write to complete successfully.

These specifications are met by supplying the *address* and *write* signals at a rising edge of the µC clock, and *data* at the falling edge of the clock, a little more than 10 ns later. All three signals are held for the remaining 10 ns of the clock cycle, and removed at the next rising edge. The state of the *read* signal is irrelevant during this process, since it is negated by the *write* signal.

When performing a read operation, this means first of all removing the *write* signal. Reads can then be controlled by switching *address* to point to the desired memory cell, and the *read* signal to HIGH in arbitrary order. The *data* bus is guaranteed to be valid $t_{\mathrm{AA}} = 10\,\mathrm{ns}$ after switching *address* and $t_{\mathrm{DOE}} = 5\,\mathrm{ns}$ after switching the *read* signal. Thus, when setting both at the rising edge of a clock cycle, the *data* bus can be safely sampled at the following falling edge or any edge after, for as long as the *address* and *read* signals remain unchanged. Since it is beneficial for

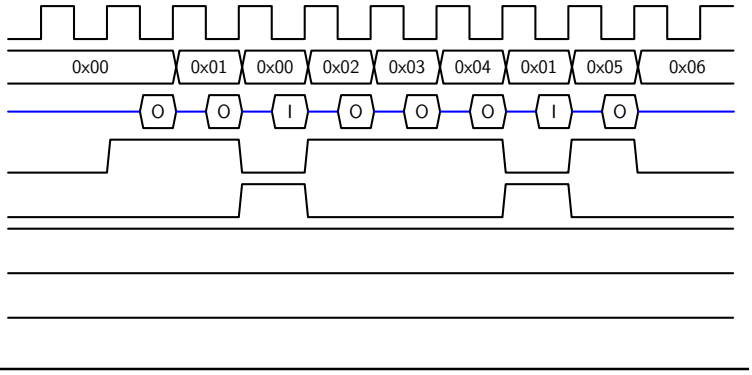
Signal	Timing
µC clock	
SRAM address	
SRAM I/O	
SRAM write	
SRAM read	
SRAM BHE	
SRAM BLE	
SRAM CE1	

Table 5.1: Timing table of the SRAM interface of the *SRAM FIFO* module, matching the signal timings specified in the SRAM data sheet. The letters on the *SRAM I/O* bus denote outgoing (O) and incoming (I) data.

the FPGA design to perform as many operations as possible on the same clock edge, the following rising edge was chosen as the sampling point. Tab. 5.1 shows the signal timing achieved by the *SRAM FIFO* module, adhering to the stated specifications. The *byte high enable* (BHE), *byte low enable* (BLE), and *chip enable* (CE1) interface signals are not needed in the design and held at a constant value.

Consecutive USB reads were found to be performed every third clock cycle, reading 8 bit each. Since 16 bit are retrieved from the SRAM during each read, and buffered inside the FPGA, it is sufficient to perform an SRAM read every other USB read. Thus, at least six clock cycles pass in between any two SRAM reads, by far sufficient for coordinating quasi-simultaneous read and write operations on the SRAM.

SRAM FIFO Implementation

A common type of intermediary storage for streaming applications is a *circular buffer*. Such a structure is created by generating a *write pointer* and a *read pointer*, respectively storing the next memory address to be written to or read from. When consequently incrementing the corresponding pointer after each read or write, and having them wrap around at the end of the address space, a behaviour resembling a FIFO is produced. Since exactly this is desired, and a circular buffer maps well to the FPGA/SRAM combination at hand, it was chosen for the implementation of the *SRAM FIFO* module. A schematic illustrating the below description is presented in Fig. 5.5.

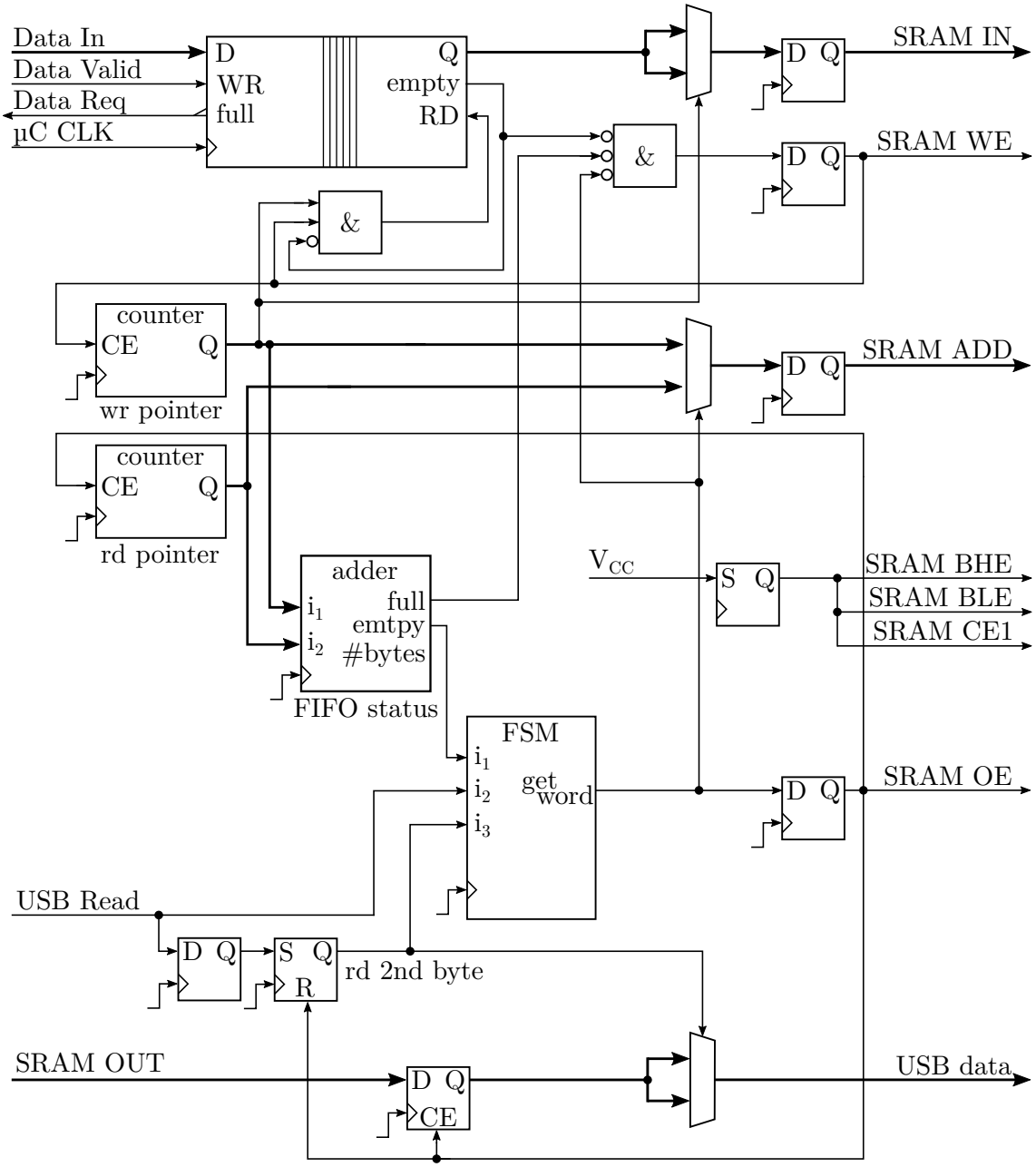


Figure 5.5: Schematic of the SRAM FIFO module. SRAM reads are performed every other USB read, writes whenever new data arrives.

Since the 48 MHz µC clock was shown to work well with the SRAM timing requirements, and the USB interface operates at this very clock, it was chosen for all synchronous module logic. To minimize skew on the SRAM interface, each signal is routed through a register at the FPGA IOBs. The internally received data is similarly buffered by passing it through a FWFT FIFO. This allows controlling the

data stream, necessary for delaying it during SRAM reads. The *valid* bit generated by the data arbiter serves as the *write* signal to the FIFO. The inverted *full* signal provides feedback to the arbiter via its *DATA REQ* input.

SRAM FIFO Write Operation

The output of the buffer FIFO is passed to the SRAM I/O bus via a multiplexer (MUX) and a subset of the aforementioned output registers. At the MUX the 32 bit bus gets split into two 16 bit buses, each matching the width of the I/O bus and the SRAM memory cells. Which half is passed to the SRAM depends on the least significant bit of the write pointer. Writing all lower halves to the even addresses and upper halves to the odd addresses, ensures all words are stored in consecutive memory cells. Since the address space begins with an even address, a word is fully written when the write pointer switches from an odd address back to an even one. By setting the FIFO *read* signal not on this transition, but already when the write pointer is set to an odd address and a write is starting, the next word becomes available on the FIFO output as soon as the write finishes. This saves one clock cycle, and allows for continuous SRAM writes without any idle and thus waisted clock cycles in between.

The read and write pointers are implemented by inferring a counter with clock enable for each of them. These counters increment their stored value once per clock cycle, as long as they are supplied with a logic HIGH on the clock enable (CE) port. Otherwise the current value is retained. Since the respective pointer should move to the next address after a successful read or write operation, the SRAM *read* (SRAM OE) and *write* (SRAM WE) signals are connected to the corresponding CE ports. The pointers are routed to another MUX, putting one of them through to the SRAM *address* bus (SRAM ADD). This is usually the write pointer, unless the SRAM *read* signal, connected to the *select* port of the MUX, becomes HIGH. Both pointers are also connected to an arithmetic block, calculating the current fill level of the SRAM FIFO and providing a *full* and *empty* flag.

The *write* signal is generated by inverting the buffer FIFO *empty* port, SRAM FIFO *full* flag and SRAM *read* signals, and ANDing them. This way the SRAM is written to, whenever data is available on the input FIFO, the SRAM has available memory cells, and is currently not being read. Allowing SRAM read operations to block writes is necessary, since write operations can be buffered by storing data in the input FIFO, whereas read operations are instructed by the host and carried

out by the microcontroller without any question or remorse. Should a read come in between the two write operations for the first and second halves of a word, the second half is simply written after the read was performed. By blocking the *write* signal the SRAM read also inhibits the *read* signal to the buffer FIFO, so the current word remains available.

SRAM FIFO Read Operation

As mentioned above, read operations are received from the microcontroller and retrieve the data available on the 8 bit USB data bus. Since twice that many bits are read from the SRAM simultaneously, the module must keep track of the number of USB reads that occurred since the last SRAM read. The counting is done by a *finite state machine* (FSM), taking the SRAM FIFO *empty* flag, USB *read* signal, and *read second byte* (R2B) signal as inputs. From these values and its three states it determines when the next SRAM read is due. It then generates a *get word* instruction, which is received by the SRAM *write*, *address* and *read* signal logics and switches them to *read* mode.

The initial state of the FSM is the *try-sram-read* state. While in this state the FSM checks the *empty* flag every clock cycle, until it is removed. Once it goes LOW the FSM transitions to the *read-sram* state and issues the *get word* command in the process. At the next clock cycle the FSM moves to the third state, *await-usb-read*. Here it remains until it detects the USB *read* and R2B signals being HIGH at the same time. This indicates a second USB read being made, requiring a new word from the SRAM FIFO afterwards. To get the next word the FSM transitions back to the *read-sram* state, again issuing a *get word* command, and again moving to the *await-usb-read* state one clock cycle later.

As the name suggests the R2B flag is not only responsible for informing the FSM if a given USB read is a second read, but also for putting the proper half of the latest word retrieved from the SRAM onto the USB data bus. This bus splitting is once more achieved by routing the R2B signal to the *select* port of a MUX, and the high and low bytes of the SRAM data bus to the inputs of the MUX. Before being send to the MUX the SRAM data is stored in a register with CE, to remain available after the SRAM switched back to writing. The CE port is supplied with the SRAM *read* signal, causing the register to store the next value exactly one clock cycle after the *read* (and thus *address*) signal was send to the SRAM. At this point the data on the I/O bus is guaranteed to be valid by the SRAM manufacturer.

Generation of the R2B flag itself relies on the SRAM and USB *read* signals. Its value is stored in a register with *set* and *reset* ports. The *reset* is connected to the SRAM *read* signal, so it gets reset after each SRAM read, when a new word is available and its first byte is to be read. The *set* port is connected to the delayed USB *read* signal, so that the first USB read, after a new word arrived, sets R2B to HIGH. Setting R2B to HIGH must be delayed by one clock cycle to make sure the first USB read will always retrieve the first byte.

Tri-state and further control signals

So far the SRAM I/O data bus has been described as two separate buses, one SRAM OUT and one SRAM IN bus. In reality, there is only one bi-directional data bus, passing data to and from the SRAM. For successful communication only one end of the bus is allowed to be driven at any time, the device on the other end must either read or ignore the bus. Merging the SRAM IN and SRAM OUT buses to the single SRAM I/O bus and controlling read and write operation is handled by inferring tri-state buffers at the SRAM I/O pins in the IOBs. Such a buffer is shown in Fig. 5.6.

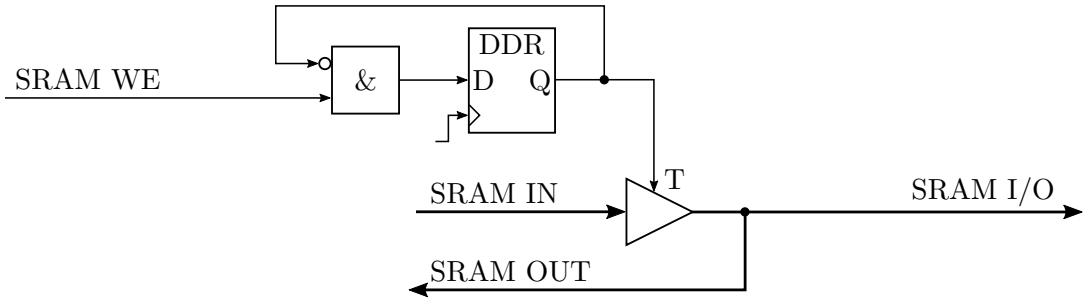


Figure 5.6: Schematic of the tri-state buffers and tri-state controls of the SRAM FIFO module.

The tri-state connects the SRAM OUT wire, used for reading from the bus, directly to the bi-directional SRAM I/O bus. The SRAM IN bus is connected via an output buffer (OBUF) with active high tri-state control. This means, that the bi-directional bus is only driven with the SRAM IN bus data, when the tri-state port (T) is HIGH. Otherwise the output of the OBUF is in high impedance state, and thus basically disconnected from the bus. To ensure the OBUF is only activated after the SRAM had sufficient time to set its own tri-states to high impedance, a double data rate (DDR) register is used. It activates the OBUF half a clock cycle

after SRAM *write* was issued. The inverted output of the DDR register is fed back to its input, by ANDing it with the *write* signal. This deactivates the tri-state half a clock cycle after it was activated, just at the same moment when other interface signals such as *read*, *write*, or *address* may change. The DDR keeps toggling states at both rising and falling clock edges for as long as *write* is HIGH, producing the pattern shown in table 5.1, and allowing safe I/O operation.

The SRAM offers three more control signals, the *byte high enable* (BHE), *byte low enable* (BLE), and *chip enable* (CE1). These are all constantly held HIGH, which permanently activates the SRAM (CE1), and causes I/O operations to be performed on all 16 bit of the SRAM memory cells simultaneously. This obviously doubles the achieved data throughput per clock cycle, compared to alternating activation of the bytes or using just one of them.

5.2.4 PixLib & USBpixI4dll

The data provided on the USB bus has to be frequently polled by the host, and the retrieved words reconstructed, histogrammed, and stored. These tasks are slow compared to the required polling rate, and would thus hinder proper board read-out when performed sequentially for each retrieved data set. Offloading the processing steps to a second thread allows to perform them in parallel to data polling, and reach the polling rates required to avoid buffer overflows and thus data loss on the hardware. The two-threaded design is visualized in Fig. 5.7.

The read-out threads and functions were placed in the *USBpixI4dll* hardware interface library, minimizing latency and strengthening the abstraction layer separating the hardware interface from the higher level classes. The involvement of the higher level libraries is subsequently reduced to starting and stopping scans, all further actions are handled internally by the *USBpixI4dll*. Upon start of a measurement it starts both threads, thenceforth continuously polling and processing data. When the measurement is stopped, first the polling thread is instructed to exit, following the order once all data was retrieved from the hardware. Then the processing thread is notified, and exits once all data has been processed.

Data Polling

Data is polled by a dedicated function incessantly called by the polling thread. The function requests the number of currently buffered words from the FPGA and

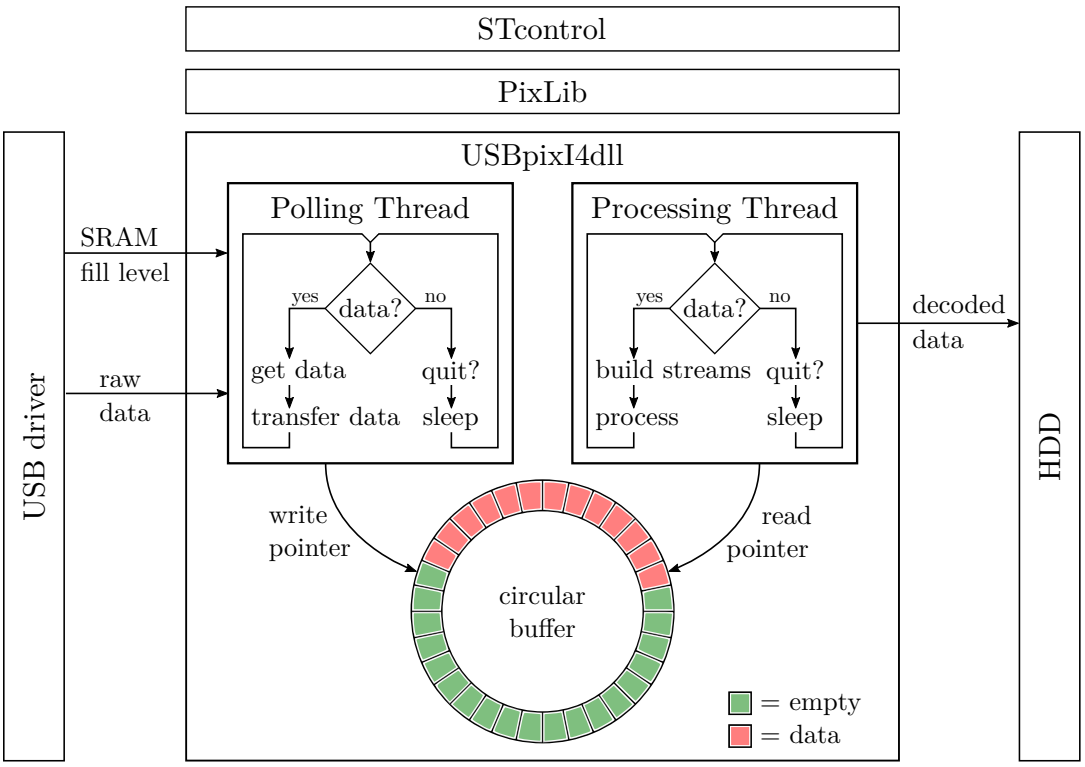


Figure 5.7: Outline of the *USBpixI4dll* structure. The two-threaded approach allows for high frequency data polling by offloading time consuming tasks to the second thread. Both threads operate on a common memory structure.

instructs the USB driver to issue the corresponding number of USB reads. The driver returns the gathered data, which is transferred to a circular buffer. Once all data has been written the function returns *true*, and is called again by the thread. In case no buffered data is available the function immediately returns *false*, whereupon the thread checks its *terminate* flag and exits if it is set.

Storing the retrieved data in a circular buffer allows data sets of varying sizes to be passed efficiently from the polling thread to the processing thread, without placing any constraints on the order or frequency of execution of the functions writing and reading the data. Absence of such constraints is important, since both parameters depend on the instantaneous data rate and the operating system's scheduler, and cannot be predicted.

Data Processing

Data processing is organized in a *reconstruction* and *data manager* function. The reconstruction function is called first by the processing thread. It checks the circular

buffer for unprocessed data, returning *false* if none was found. In this case the thread checks if the polling thread is still running and, if so, again calls the reconstruction function. Otherwise the thread exits.

If unprocessed data is available, the reconstruction function proceeds by loading the next four bytes, bit shifting them to their proper positions, and ORing them together. The words are then added to the *record streams* matching their id byte. The streams are vector data types designed for storing 24 bit FE protocol words and trigger numbers. FE words are added to dedicated streams, trigger numbers to all of them. Once all available data was processed the function returns *true*.

This causes the processing thread to call the data manager function. The manager starts either one or both of the *raw data histogrammer* and *raw data file writer* routines, based scan type and user settings. The histogrammer was designed for creating histograms from record streams, was used for *source scan* histogramming by the stopping configuration. Due to unification of the read-out process all scans now produce record streams, so the histogrammer can be used for creating all stopless configuration histograms.

The file writer also operates on record streams, but needed some modification before being usable. The comparatively low write frequency of the stopping read-out scheme allowed creation and subsequent deletion of new *file writer* and *ofstream* objects for each write cycle. The higher write frequency introduced by more frequent updates to the record streams demand a more efficient approach. Therefore the writer was modified to keep the necessary objects alive and the destination file opened during the entire measurement. Periodic updates to the file are ensured by flushing the *ofstream* after each write cycle. Once histogrammer and file writer return, the manager function returns, and the thread resumes by calling the reconstruction function once more.

Histogramming and long-term storage are the final steps in the proposed upgrade to the USBpix read-out chain. All remaining tasks, such as presenting the histograms to the user, are handled by higher level classes. These remained unaltered, since all added and upgraded functions were fitted to the existing interfaces between the higher level and hardware interface libraries. The system is thus complete and ready for validation.

6 Measurements

The proposed stopless read-out scheme should provide the same results as the stable USBpix configuration across all scan types. The only visible difference should be a greatly improved data acquisition rate during *Source* and *testbeam measurements*. The full data retention for all scan types was, as of this writing, only implemented in hardware and the *USBpixI4dll* hardware interface library, but lacked support from higher level classes. Hence the results and histograms produced per scan type are the same for both configurations, and compared below. Once the higher level classes are adapted to the proposed read-out scheme, the user can be presented with options for raw data storage and creation of additional histograms.

6.1 Fundamental Tests

Before tests involving large volumes of data can be performed, the system has to be examined on a *per-word* level. This is achieved by performing an *Analog Scan* with a single injection in a single pixel, and observing the data stream on the FPGA and the data received by the host. The injections were limited to a single pixel by deactivating all others in the *CAP0* and *CAP1* masks of the FE. Read-out was limited to the same pixel by activating only the chosen pixel in the *ENABLE* mask. Fig. 6.1 shows the used *ENABLE* mask as an example for all three identical masks. The signals on the FPGA were examined using the *Xilinx Chipscope* utility. The captured waveform shows the data bus and *valid* signal at the *data arbiter* output. It is presented in Fig. 6.2. The occupancy plot generated from the data received by the host is shown in Fig. 6.1, next to the *ENABLE* map.

The scan was performed with a *Single Chip Adapter Card* (SCA) and a *4-chip module*. The test pixel was randomly chosen to be the pixel in row 92 and column 46. The waveform captured the *data record* (DR) containing the hit information, amidst several of the 16 *data headers* (DH) generated per event. The DR shows the bit pattern 0x015E5D6F, comprised of 8 bit channel tag, 9 bit row number, 7 bit

column number, 4 bit TOT1 and 4 bit TOT2. Decoding this shows a hit with ToT code 6 in pixel 92/46 of FE 1, and no hit in the adjacent pixel, just as expected. The waveform thus shows the correct hit information being properly tagged and available at the data arbiter output. The occupancy plot shows the same pixel having registered exactly one hit, while all other pixel have none. This confirms, that the DR was received and decoded correctly by the host.

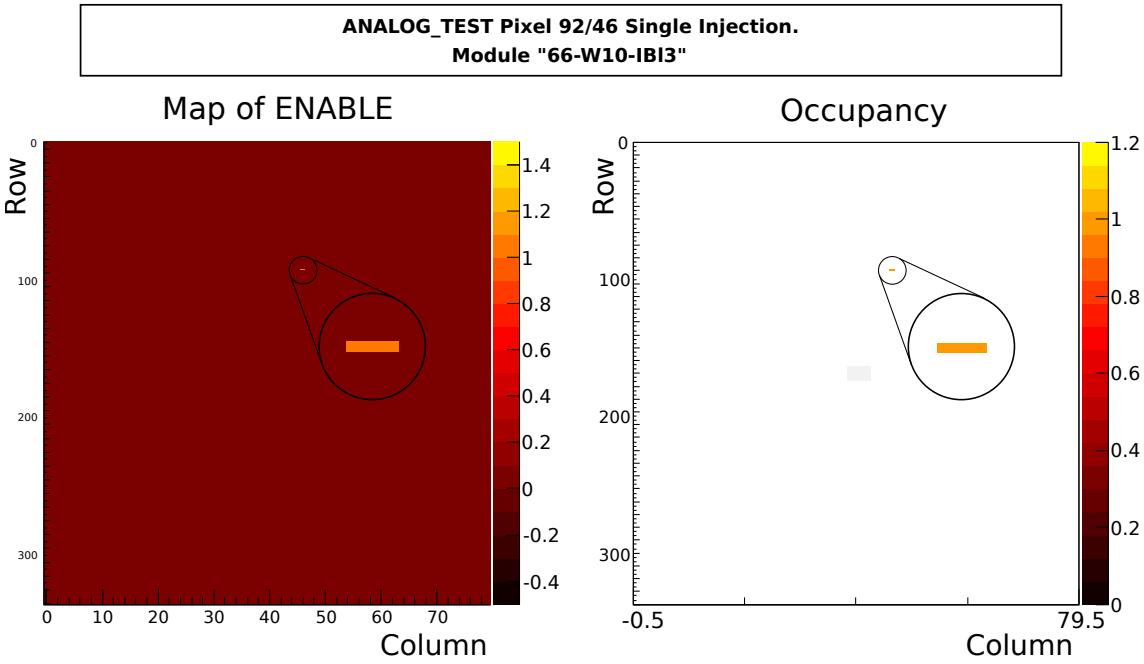


Figure 6.1: The *ENABLE* mask for a single pixel *Analog Scan* and the occupancy plot generated from the received data.

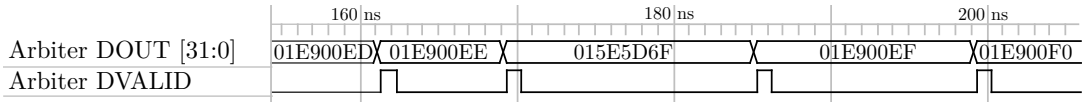


Figure 6.2: Waveform showing the single *data record* produced by the scan at the *data arbiter* output.

6.2 Calibration measurements

As described in section 4.1.3, the *Analog Scan* and *Digital Scan* fully involve the FE-I4's internal testing mechanisms. Performing these scans is thus sufficient for validating any calibration measurement on the hardware and hardware interface levels. For the initial measurements a *Single Chip Adapter* (SCA) and *Single Chip*

Carrier (SCC) card were chosen, since the combination is simpler in design and handling than the alternative *Burn In Card* (BIC) and *4-chip module* combination. Both scans were successfully performed using the stopping and stopless USBpix configurations, and are presented below.

Afterwards the SCA and SCC were swapped against a BIC and 4-chip module. Both scans were again performed for both configurations, but all results were unclear due to difficulties in the operation of the prototype 4-chip module. Without a proper baseline measurement from the known-good stable configuration, any attempt at a meaningful analysis of the proposed configuration would be futile. Therefore the four-chip measurements had to be postponed until all issues with the available 4-chip module have been resolved.

6.2.1 Analog Scans

The results of the *Analog Scans* are presented in colour-coded occupancy histograms, showing how many hits each FE pixel registered. Both configurations properly returned a histogram, shown in Fig. 6.3. A typical *Analog Scan* injects the same charge 200 times into each pixel, so 200 hits are expected in each bin of the histograms.

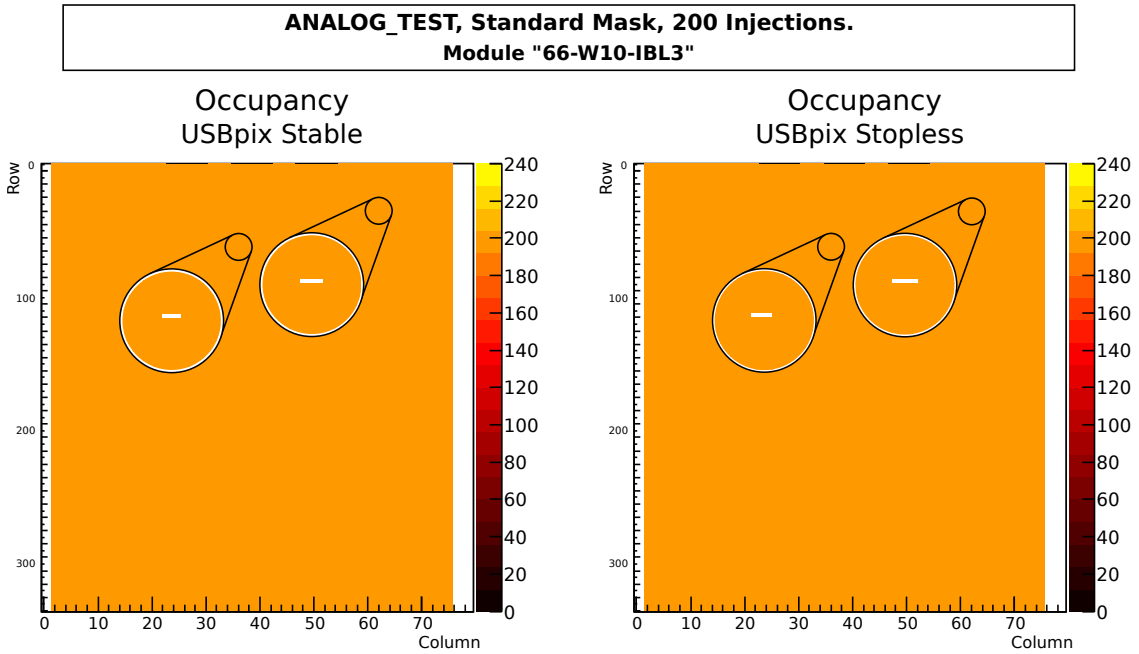


Figure 6.3: *Analog Scans* performed using the stable and stopless USBpix configurations. Both show the same two dead pixels.

This result is observed for both configurations for all but two FE pixels, which

are known to be dead and thus registered no hits. The first and last few columns also show no hits, which is expected as well. These columns were disabled in the scan configuration, since the FE at hand is a prototype model FE-I4A, containing test pixels in the intentionally skipped columns.

6.2.2 Digital Scans

Since the *Digital Scan* is very similar to the *Analog Scan* its results are presented using the same type of histogram. The only difference between the scans is the point of the injection, located at the digital pixel electronics for the *Digital Scan*. As for the *Analog Scan* 200 injections were made per pixel, so again 200 hits are expected per pixel. Both histograms (Fig. 6.4) show all pixels registering all 200 hits, even the dead pixels identified during the *Analog Scan*. Hence the point of failure is narrowed down to the analogue electronics of both pixels.

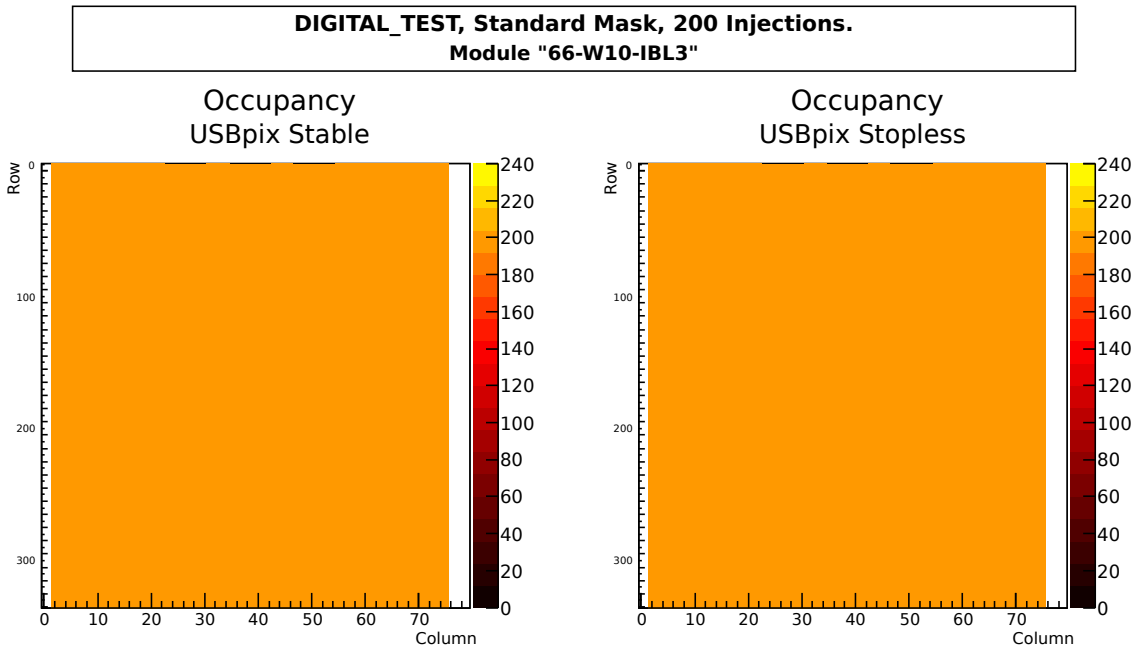


Figure 6.4: *Digital Scans* performed using the stable and stopless USBpix configurations. Both show the same perfect results.

The *Analog* and *Digital Scans* working perfectly is a strong indication towards the validity of all calibration scans. Individual verification of each scan type remains to be done, due to the limited time available.

6.3 Source measurements

Both configurations produced very good results during the calibration scans. The FE at hand thus seems to be working properly and is suitable for the following source measurements. The *Source Scan* mode used to produce the following measurements is a very versatile scan type. Depending on which parts of the test set-up are well understood and which are supposed to be investigated, it can provide information about the sensor, the injection mechanism, the FE under test, or the test system itself. In this set-up, the three former parts are well known, so the measurement can be used to characterize the proposed USBpix configuration exclusively.

Two sets of measurements are presented below, a noise measurement without any intentional injections, and a source measurement with injections originating from a radioactive source. For each set a reference measurement is performed using the stable USBpix configuration, before repeating the measurement with the stopless configuration.

6.3.1 Set-Up

For the following measurements the same SCA/SCC/FE combination was used as for the calibration scans. Charge injections were made into the n-in-n planar silicon sensor bump-bonded to the FE, using a radioactive Strontium source. The source and FE were placed in a containment box to prohibit radioactive contamination of the laboratory environment. The sensor was depleted by applying a bias voltage of $-50\,\mathrm{V}$.

The Strontium source contains the Strontium isotope $^{90}\mathrm{Sr}$. This isotope decays to $^{90}\mathrm{Y}$ and further to $^{90}\mathrm{Zr}$, both via β^--decays. It has a half-life of 28.78 a and emits electrons with an endpoint energy of 0.546 MeV. The Yttrium isotope decays with a half-life of 64.1 h, and emits 2.282 MeV electrons. The initial activity of the source was 18 MBq when manufactured.

FE read-out was triggered using the FE self-trigger mechanism, which generates a trigger signal whenever any of the FE pixels registers a hit. Each measurement was restricted to 2 min, from the time the *Start Measurement* button to the time the *Stop Measurement* button was clicked.

Configuration:	stopping	stopless
Duration	108 s	108 s
#DH	448	544
#DR	40	54
#Hits	62	77
Hit rate	0.57 Hz	0.71 Hz

Table 6.1: Comparison of the amount of data received during the noise measurements with the stable and proposed USBpix configurations.

6.3.2 Noise Measurements

The noise measurements were performed without the Strontium source, but still inside the containment box and with bias voltage applied. The timestamps generated at the beginning and end of the actual scans showed a scan duration of 108 s for both configurations.

The number of collected FE words and encoded hits is summarized in table 6.1, which shows comparable numbers for both configurations. The observed deviations are well within the limits of statistical fluctuations. Occupancy histograms were produced for both scans, showing the location and hit count for each pixel. Both configurations recorded hits spread evenly over the FE, with two hits per pixel at most (Fig. 6.5). The *Time-over-Threshold* (ToT) histograms (Fig. 6.6) present the recorded numbers for each possible ToT code, translating to the amount of charge collected by the FE per hit. The bins 0 through 12 encode charge intervals in ascending order, while ToT code 13 contains all hits that would have a ToT code larger than 12, and ToT code 14 all those with a ToT code smaller than 0. Both configurations generated similar distributions, with peaks around ToT codes 2 and 3. The LV1 histograms (Fig. 6.7) reveal the time that passed in between a trigger being issued and a hit being registered. Thus hits in LV1 bin 0 are likely to be hits that caused a trigger. The hits in the following few bins mostly originate from charge sharing or a particle only grazing a pixel while traversing the sensor. Such small hits take longer to be registered and thus show up later in the LV1 histograms, an effect called *timewalk*.

All three sets of histograms show very similar results for both configurations, further supporting the validity of the proposed stopless configuration. The number of noise hits is higher than expected for a good module in a closed containment box with bias voltage applied, but still easily explainable. Even under such controlled

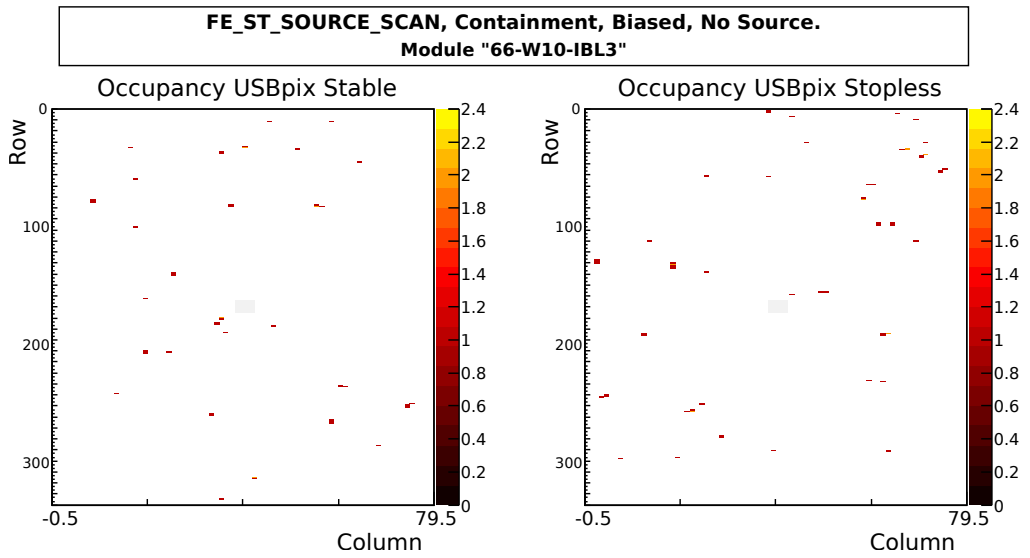


Figure 6.5: Occupancy histogram showing the number of hits per pixel during the noise measurement with the stable and proposed USBpix configurations.

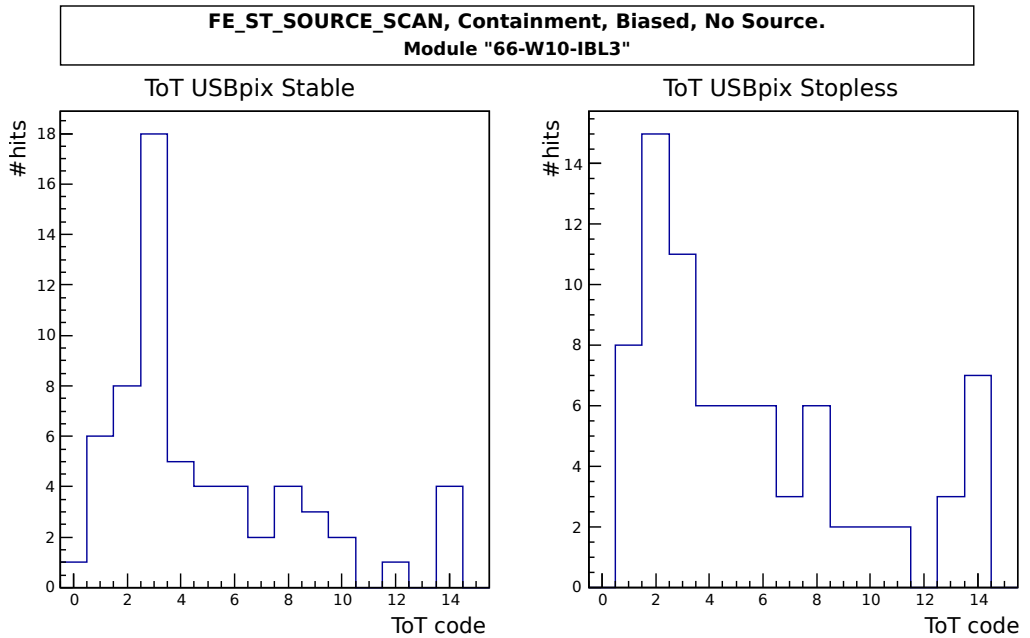


Figure 6.6: Time-over-Threshold histogram showing the charge distribution of hits during the noise measurement with the stable and proposed USBpix configurations.

conditions many causes for noise hits remain, including cosmic and stray radiation, sub-optimal FE tuning, high leakage currents in the sensor, noisy pixels, and insufficient bias voltage for complete depletion. The collected data does not provide enough information to narrow down the possible origins or rank them according to their significance.

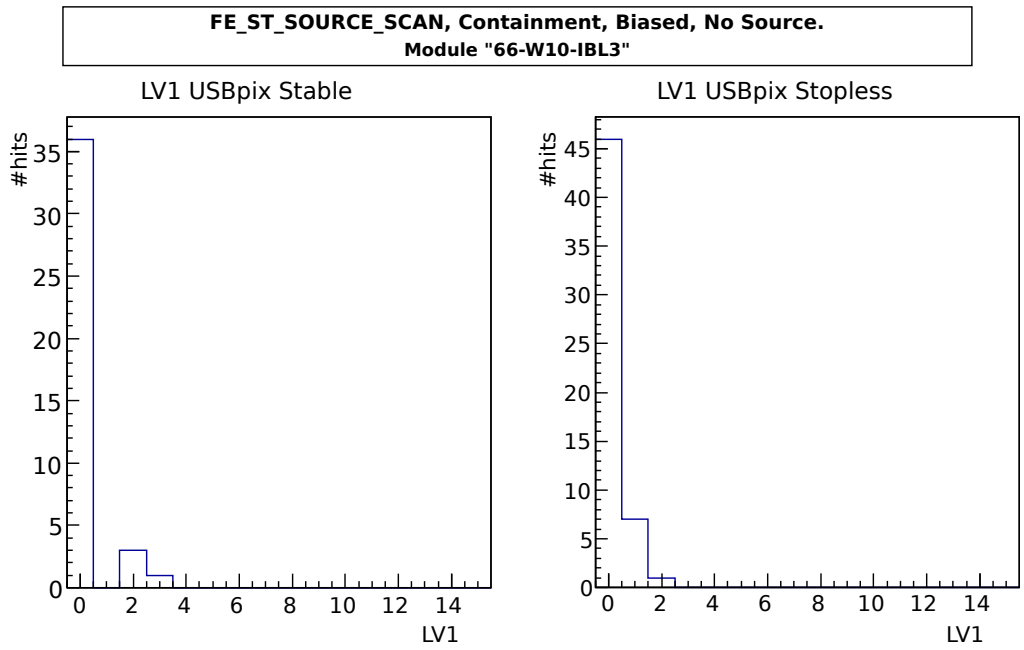


Figure 6.7: LV1 histogram showing the time delay between trigger and hits during the noise measurement with the stable and proposed USBpix configurations.

Configuration:	**stopping**	**stopless**
Duration	110 s	108 s
#DH	4 813 021	45 604 075
#DR	779 423	7 382 864
#Hits	1 166 141	11 046 473
Hit rate	10.6 kHz	102.3 kHz

Table 6.2: Comparison of the amount of data received during the Strontium source measurements with the stable and proposed USBpix configurations.

6.3.3 Strontium Measurements

After completing the noise measurement the Strontium source was placed in the containment box and aligned with the FE. Then the same measurement procedure was repeated as for the noise measurement, subsequently producing the same type of data and histograms as explained above.

While the noise measurement showed very similar numbers of recorded hits and subsequently hit rates for both configurations, table 6.2 presents a very different picture for the source measurement. The stable configuration recorded 1 166 141 hits during a 110 s measurement, while the stopless configuration recorded almost ten times the number of hits in two seconds less. This immense difference leads

to a hit rate of 102.3 kHz for the stopless configuration, 9.7 times higher than that observed using the stopping configuration.

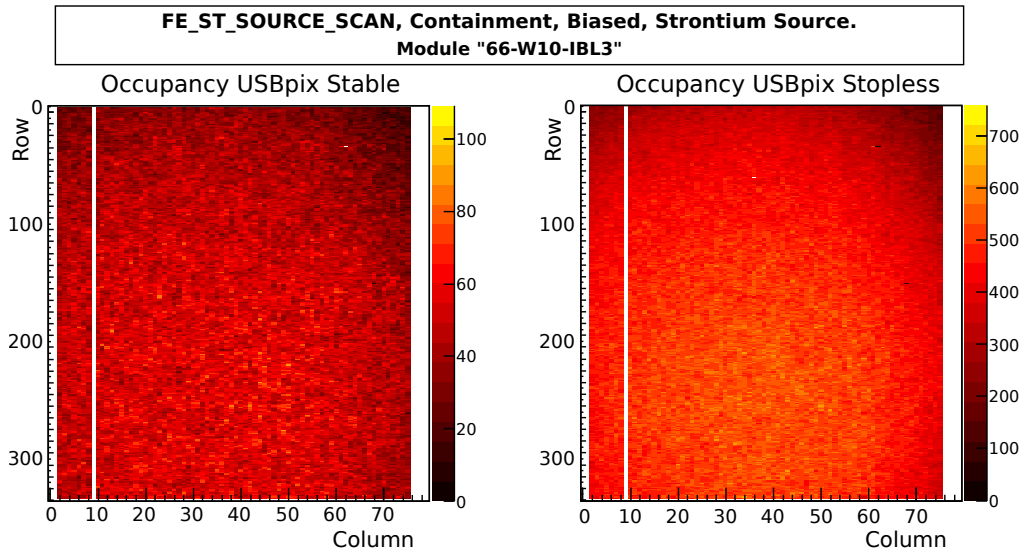


Figure 6.8: Occupancy histogram showing the number of hits per pixel during the Strontium source measurement with the stable and proposed USBpix configurations.

The produced histograms (figures 6.8, 6.9, 6.10) verify, that the additional data originates from the Strontium source and is not an artefact of the stopless configuration. The histograms of all three sets appear almost identical to one another in shape, but with a tenfold increase of the scales for the stopless read-out histograms. The occupancy plots both show hits in all but the masked test pixels and the two dead pixels identified during the *Analog Scan*. Additionally no hits appear in column 9, which was masked due to a high number of noisy pixels. The histogram from the stable configuration hints at the position of the centre spot of the Strontium source, whereas the stopless histogram clearly shows its location to the bottom left of the chip's centre.

The ToT histograms show the Landau distribution expected for the energy loss of ionizing particles traversing a thin material [**?**]. The observed rise and large peak at the tail of the distribution is due to the special purposes of ToT codes 13 and 14. The LV1 histograms again show almost all hits in LV1 bin 0 and only few late hits, which are likely caused by *timewalk*.

All obtained results strongly support the validity of the stopless read-out scheme and the proposed implementation. The benefits to the data acquisition rate are clearly shown by this measurement.

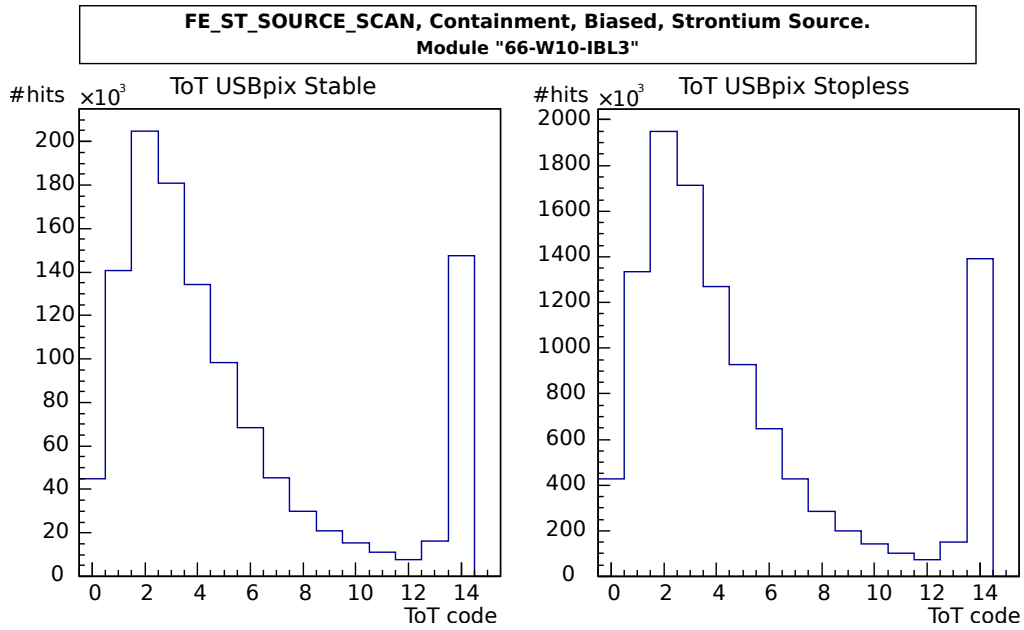


Figure 6.9: Time-over-Threshold histogram showing the charge distribution of hits during the Strontium source measurement with the stable and proposed USBpix configurations.

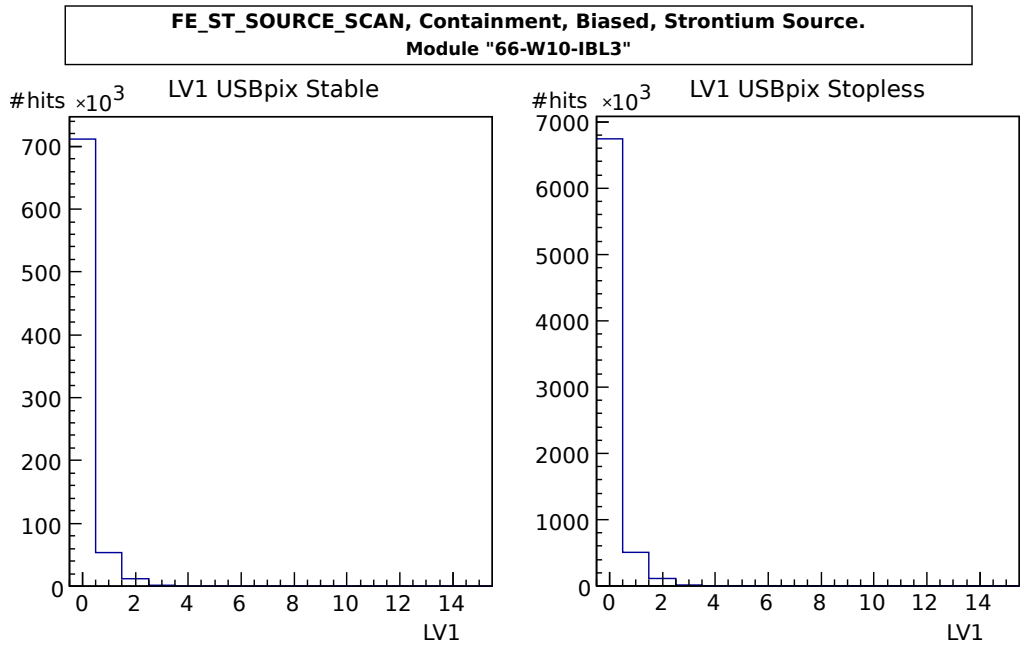


Figure 6.10: LV1 histogram showing the time delay between trigger and hits during the Strontium source measurement with the stable and proposed USBpix configurations.

7 Summary

An analysis of the stable USBpix configuration showed the stopping read-out scheme to severely limit the achievable average data acquisition rate of the system. The employed scheme required frequent interruptions of ongoing measurements for transferring data from the hardware to the host. An alternative *stopless* read-out scheme was presented, allowing board read-out during a running measurement by managing simultaneous read and write access to the data buffer. An implementation was proposed and necessary changes to the hardware interface library and the FPGA firmware were motivated and described.

Measurements performed with both the stable and proposed USBpix configurations were shown to support the validity of the stopless configuration. All obtained results matched the expectations and reference measurements perfectly. The achieved average data acquisition rate of the stopless configuration was found to be almost ten times the rate achieved by the stopping configuration. The observed data rate was likely limited by the activity of the available Strontium source, hinting at a total increase in data acquisition rate even larger than the determined factor.

Several calibration scans remain to be verified individually. Additional source and testbeam measurements need to be performed to fully characterize the proposed upgrade and ensure its stability. Several minor updates to the FPGA firmware are yet to be implemented as well. They aim to provide a more robust and precise read-out of the SRAM FIFO fill level and better handling of unlikely yet possible buffer overflows.

While a further increase in data acquisition rate may be achievable by rewriting the microcontroller firmware, it is unlikely to provide any real-world benefits. The rate achieved using the presented stopless configuration was shown to be sufficient for testbeam measurements and laboratory applications. A jump in data rate large enough to support either more FEs in parallel or FEs of the forthcoming HL-LHC generation is prohibited by the hardware itself.

An upgraded Multi-IO board is developed as part of the *USBpix 3.0* system,

which supports the current USB 3.0 SuperSpeed specification, offering data rates up to 5 Gbit s^{-1}. Porting the USBpix FPGA firmware to the new system and integrating it into *STcontrol* will be the next great challenges.

Bibliography

[1] D. Griffiths, *Introduction to Elementary Particles*, Wiley-VCH, second edition (2004)

[2] K.A. Olive et al. (Particle Data Group), *Review of Particle Physics*, Chin. Phys. C **38**, 090001 (2014)

[3] P. W. Higgs, *Broken Symmetries, Massless Particles and Gauge Fields*, Phys. Lett. **12**, 132 (1964)

[4] F. Englert, R. Brout, *Broken Symmetry and the Mass of Gauge Vector Mesons*, Phys. Rev. Lett. **13**, 321 (1964)

[5] 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)

[6] S. Chatrchyan, et al. (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)

[7] S. Brüning et al., *LHC Design Report*, CERN-ATS-2012-236, CERN, Geneva (2004)

[8] ATLAS Collaboration, *Measurements of Higgs boson production and couplings in diboson final states with the ATLAS detector at the LHC* **(CERN-PH-EP-2013-103)** (2013)

[9] ATLAS Collaboration, *Letter of Intent for the Phase-II Upgrade of the ATLAS Experiment*, CERN-LHCC-2012-022. LHCC-I-023, Geneva (2012)

[10] ATLAS Collaboration, *Physics at a High-Luminosity LHC with ATLAS* **(ATL-PHYS-PUB-2012-004)** (2012)

[11] ATLAS Collaboration (ATLAS), *ATLAS Letter of Intent for a General-Purpose pp Experiment at the Large Hadron Collider at CERN*, CERN-LHCC-92-4, LHCC-I 2 (1992)

[12] ATLAS Collaboration (ATLAS), *ATLAS DETECTOR AND PHYSICS PERFORMANCE Technical Design Report Volume I*, Technical Report ATLAS TDR 14, CERN-LHCC 99-14, CERN (1999)

[13] G. Aad et al. (ATLAS), *The ATLAS Inner Detector commissioning and calibration*, Eur. Phys. J. **C70**, 787 (2010)

[14] J. Große-Knetter (ATLAS), *Overview of the ATLAS Insertable B-Layer (IBL) Project*, Technical Report ATL-INDET-PROC-2011-022 (2011)

[15] IBL Community (ATLAS IBL), *ATLAS Insertable B-Layer Technical Design Report*, Technical Report ATLAS TDR 19, CERN-LHCC 2010-013, CERN (2010)

[16] IBL collaboration (ATLAS IBL), *Prototype ATLAS IBL modules using the FE-I4A front-end readout chip*, J. Inst. **7(11)**, P11010 (2012)

[17] M. Barbero et al. (ATLAS IBL), *The FE-I4 Pixel Readout Chip and the IBL Module*, Technical Report PoS(Vertex 2011)038, CERN (2011)

[18] M. Červ, *The ATLAS Diamond Beam Monitor*, J. Inst. **9(02)**, C02026 (2014)

[19] L. Rossi, O. Brüning, *High Luminosity Large Hadron Collider A description for the European Strategy Preparatory Group*, Technical Report CERN-ATS-2012-236, CERN, Geneva (2012)

[20] S. McMahon et al., *Initial Design Report of the ITk*, Technical Report ATL-COM-UPGRADE-2014-029, CERN, Geneva (2014)

[21] M. Backhaus et al., *Development of a versatile and modular test system for ATLAS hybrid pixel detectors*, NIM A **650(1)**, 37 (2011)

[22] J. Schneider, H. Krüger, *S3 Multi IO System – S3 Multi IO USB Card Version V1.03*, Technical report, Universität Bonn (2010)

[23] Cypress, *CY7C68013A, CY7C68014A, CY7C68015A, CY7C68016A EZ-USB FX2LP USB Microcontroller High-Speed USB Peripheral Controller*, No. 38-08032, Revised January 15, 2015, datasheet

[24] Xilinx, *Spartan-3 Generation FPGA User Guide* (2011), UG331, v1.8

[25] E. Corrin, *EUDAQ Software User Manual*, EUDET-Memo-2010-01 (2010)

[26] ATLAS Collaboration, *The FE-I4B Integrated Circuit Guide* (2012)

[27] Xilinx, *Spartan-3 FPGA Family Data Sheet*, Technical Report DS099, v3.1 (2013)

[28] J. Reichardt, B. Schwarz, *VHDL-Synthese: Entwurf digitaler Schaltungen und Systeme*, Oldenbourg Wissenschaftsverlag (2012)

[29] N. Sawyer (Xilinx), *Data Recovery* (2005), XAPP224 (v2.5)

[30] J. Agricola, *Development Of A Faster Test System For ATLAS Pixel Front End Electronics* (2014), ISSN 1612-6793, II.Physik-UniGö-MSc-2014/05

[31] R. Murphy, *USB 101: An Introduction To Universal Serial Bus 2.0*, AN57294, Document No. 001-57294 Rev. *F

[32] J. Weingarten, *private conversation*

[33] Cypress, *CY7C1061AV33 16-Mbit (1Mx16) Static RAM*, No. 38-05256, Revised January 16, 2015, datasheet

[34] L. Landau, *On the energy loss of fast particles by ionization*, J. Phys.(USSR) **8**, 201 (1944)

Acknowledgements

I thank all people involved in the making and successful completion of this thesis. First of all Arnulf Quadt, for allowing me to write this thesis as a member of his group, and Jörn Große-Knetter, for his professional supervision. I further thank Jens Weingarten for answering many questions, guidance throughout the making of this thesis, and proofreading it.

I also extend thanks to Julia Rieger and Lars Graber for many helpful insides into the quirks of the hardware at hand, and Johannes Agricola for the introduction to FPGA design. Lastly I thank all members of the II. Institute for their support and encouragement.

Erklärung nach §18(8) 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 nichtbestandenen Prüfung an dieser oder einer anderen Hochschule eingereicht wurde.

Göttingen, den 16. Oktober 2015

(Björn Klaas)