Georg-August-Universität
Göttingen
Zentrum für Informatik

ISSN 1612-6793
II.Physik-UniGö-MSc-2014/05

Masterarbeit

im Studiengang ”Angewandte Informatik”

Development Of A Faster Test System For ATLAS Pixel Front End Electronics

Johannes Agricola

II. Physikalisches Institut

Bachelor- und Masterarbeiten
des Zentrums für Informatik
an der Georg-August-Universität Göttingen

26. September 2014

Georg-August-Universität Göttingen
Zentrum für Informatik

Lotzestraße 16-18
37083 Göttingen
Germany

Tel. +49 (5 51) 39-1 44 14
Fax +49 (5 51) 39-1 44 15
Email office@informatik.uni-goettingen.de
WWW www.informatik.uni-goettingen.de

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Göttingen, den 26. September 2014

Master Thesis

Development Of A Faster Test System For ATLAS Pixel Front End Electronics

Johannes Agricola

September 26th, 2014

Betreut durch
Prof. Dr. Arnulf Quadt
und
PD Dr. Jörn Große-Knetter
II. Physikalisches Institut
Georg-August-Universität Göttingen

Contents

Abstract

In the ATLAS Collaboration, plans for upgrades to the ATLAS Pixel Detector take shape. These include pixel layers using read out chips similar to the recently developed FE-I4, which was used in the Insertable B-Layer. In contrast to the latter, not only layers directly at the beam pipe, but also at greater distances are required, which exhibit a lower occupancy and larger areas per barrel structure. As the per area cost of larger modules is less than in smaller ones, four FE-I4 chips per module are being taken into consideration. The lower occupancy reduces the data rate per link and thus the efficiency, which motivates the usage of link sharing techniques. This thesis implements parallel read out of modules with four FE-I4 chips with the small-scale read out system USBpix. Pixel detector data is interleaved in a specially created set-up and deinterleaved on USBpix, which has been prepared to handle future sharing techniques. The complete implementation is evaluated using a four chip module.

1 Introduction

The ATLAS Collaboration, together with the CMS Collaboration, announced the discovery of the Higgs Boson on 4th of July, 2012 [12, 13]. This was made possible by the ATLAS (A Toroidal LHC AppartuS) experiment, a multi-purpose particle detector located at a collision point of the Large Hadron Collider (LHC). The LHC is a proton-proton collider designed for a center of mass energy of $\sqrt{s} = 14$ TeV as well as a heavy ion collider.

ATLAS detects the particles created in these collisions with several layers of different detectors. The inner detector performs tracking and vertexing of electromagnetically interacting particles using a pixel detector, a silicon strip detector and a transition radiation detector. The first and innermost in this list is built from three layers of silicon pixels. With $80 \cdot 10^6$ read out channels spread over an area of $2.2\,\mathrm{m}^2$ it is both highly integrated and extending to a large range of pseudorapidities $|\eta| < 2.5$. The pseudorapidity is $\eta = -\ln\tan(\theta/2)$ with the polar angle θ between particle momentum and beam axis.

With the Insertable B-Layer (IBL) [8] an additional layer is planned to be inserted between the innermost pixel detector layer and a new beam pipe to improve tracking performance at increased luminosities. For this project, a particularly radiation-hard read out chip with increased resolution, the FE-I4, was devised.

Still, through prolonged exposure to the high doses of radiation right next to an interaction point of a high energy hadronic collider and increased pile-up due to higher luminosities, the efficiency of the complete inner detector is going to decline. These are not only results from ageing devices but actual technological limitations. That motivates to opt for an upgrade to more advanced methods, such as the replacement of the transition radiation tracker with silicon strips and the inner layers of the silicon strip detector in turn with pixel detector layers, during the next upgrade phases [10, 11].

Modern pixel detector technologies, especially the most commonly used hybrid pixel detectors, which are built from a solid-state sensor material with attached read out electronics, are sufficiently radiation hard. While hybrid detectors constructed from a separate pixel sensor and read out electronics yield a highly flexible sensor choice from different silicon sensor types to diamond, it also results in a high area cost compared to silicon strips, which makes it hard to justify the gain of replacing these with pixel detectors. It is therefore subject of ongoing research to investigate monolithic pixel detectors, such as HVCMOS (High Voltage Complementary Metal Oxide Semiconductor).

Alternative approaches are looking into reducing the cost of manufacturing hybrid pixel detectors. One of these is to capacitively couple the signal from the sensor into the analog front-end of the read out chip. This eliminates the usually performed bump-bonding, where an ohmic connection is created through solder balls between the sensor and the chip. The relative cost of this process itself can be reduced by increasing the size of the objects that are bonded at once, which is traditionally called a *module*.

While on the IBL a maximum of two FE-I4 were joined to form a module, this thesis focuses on the read out of FE-I4 four-chip modules as the next step in module sizes. The system used for this is USBpix, a calibration and test read out system already used for development and production of the IBL. This system along with the required experimental and formal background will be introduced in the next chapter. Chapter 3 will extend this focus to the currently used USBpix FPGA configuration, which describes the processing and logic performed on the USBpix hardware. This is required to motivate the design choices, presented in chapter 4, which ultimately are supposed to lead to parallel four chip read out capabilities in USBpix. In Chapter 5 the implementation of the new FPGA configuration is detailed and justified. The new configuration is validated and the results are presented in chapter 6.

2 Experimental Setup

2.1 Large Hadron Collider

Built in the tunnel of the Large Electron Positron Collider (LEP) at CERN, on average 100 m below the ground and with a circumference of 27 km, the Large Hadron Collider (LHC) [6] is a proton-proton collider specialized on reaching unprecedented center-of-mass energies of up to $\sqrt{s} = 14$ TeV at high luminosity.

The protons, but also heavy ions, are injected into the LHC from a chain of preaccelerators. Starting in a linear accelerator, three synchrotrons (Booster, Proton Synchrotron and Super Proton Syncrotron) increase the energy of these particles to 450 GeV. In the LHC, 16 superconducting radio frequency cavities continue increasing their energy. While to this date, the particle energies have been kept at 4 TeV, the machine is designed to handle beams of up to 7 TeV.

The fully accelerated bunches collide in $n = 2808$ interaction points at a rate of up to $f = 40$ MHz. With a design RMS beam size of $\sigma = 16.7$ µm at two of these four points, $N = 1.15 \cdot 10^{11}$ particles per bunch, and the revolution frequency f_{rev}, this gives an estimate for the peak luminosity of

$$\mathcal{L} = \frac{n \cdot N^2 \cdot f_{\mathrm{rev}}}{4\pi\sigma^2} \approx 10^{34}\ \mathrm{cm}^{-2}\mathrm{s}^{-1}.$$

ATLAS is located at one of these points and enabled by the high luminosity to collect data about normally rare events. Examples are the production of Higgs-Bosons but also the search for physics beyond the Standard Model and others.

2.2 The ATLAS Experiment

ATLAS measures the outcome of these collisions using a stack of several subdetectors, covering a solid angle of almost 4π around the collision point [29]. The layers are typically divided into a barrel section and an end-cap and forward section which are the inner and outer area of the disk-shaped sides of the barrel. Together, these form the almost cylindrical experiment. An overview drawing is shown in Figure 2.1.

A particle originating in the interaction first traverses the inner detector with the pixel detector forming its innermost part, followed by a silicon strip detector and a segmented transition radiation detector. The complete inner detector is contained in a magnetic field with a nominal field strength of 2 T, enabling it to measure the momentum of electromagnetically charged particles.

The inner detector is enclosed by the electromagnetic calorimeter, a sampling calorimeter with alternating layers of lead as absorber and liquid argon as active material, which

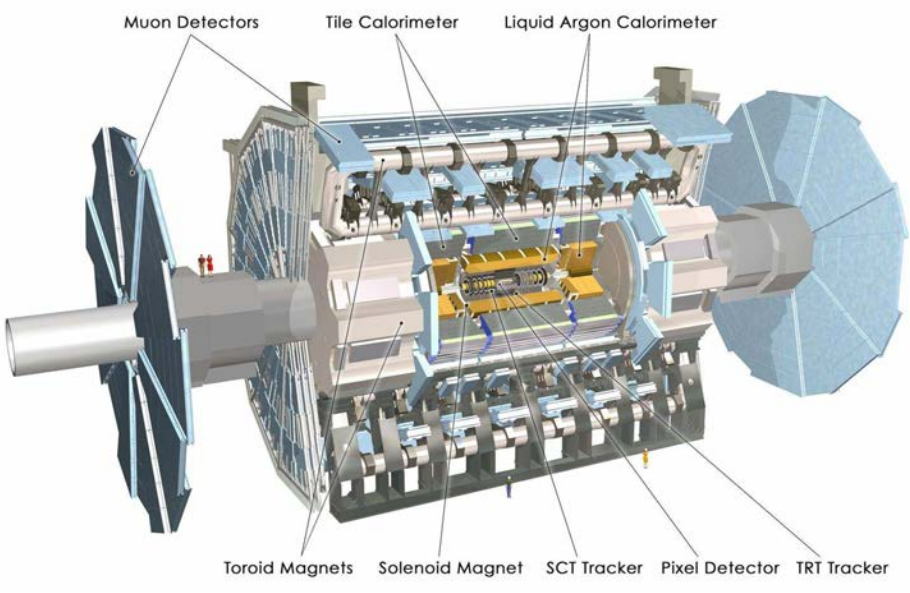


Figure 2.1: Drawing of the ATLAS Experiment.

is laid out in an accordion shape to enable fast read out. It spans a thickness of more than 25 radiation lengths.

The next layer is formed by the hadronic calorimeters. In the barrel region, energy is absorbed by iron tiles, which alternate with a scintillating material. In the end-cap and forward regions, liquid argon calorimeters are used again, absorbing with copper plates in the end-cap and, depending on the distance from the interaction point, also tungsten absorbers in the forward region.

The outermost part of the detector is the muon spectrometer. The magnetic field with a nominal field strength of 3.9 T to 4.1 T for bending is provided by two additional magnet systems; eight air-core toroids are placed radially in the barrel region and two additional toroids are inserted into the barrel in the end-cap region. The tracks are recorded using Monitored Drift Tubes and Cathode Strip Chambers at $2 < |\eta| < 2.7$ to cope with the higher detection rate in this region [1]. In addition, for $|\eta| < 2.4$, Resistive Plate Chambers (barrel region) and Thin Gap Chambers (end cap region) are used to provide fast muon trigger data.

2.2.1 The ATLAS Pixel Detector

The first part in the previously described stack is the ATLAS Pixel Detector [9] (see Figure 2.2). In the barrel region it reaches from an inner radius of 50.5 mm, formed by the innermost layer, called the B-Layer, to its outer active layer at a 122.5 mm radial distance with another layer in between at 88.5 mm.

The closeness of the B-Layer to the interaction point is crucial in tagging of jets originating from a b-quark. These often have a relatively long lifetime in the order of 1 ps, such as the B-meson with $\tau_{B^{\pm}} = 1.641 \pm 0.008$ ps [5]. The vertex of their decay products

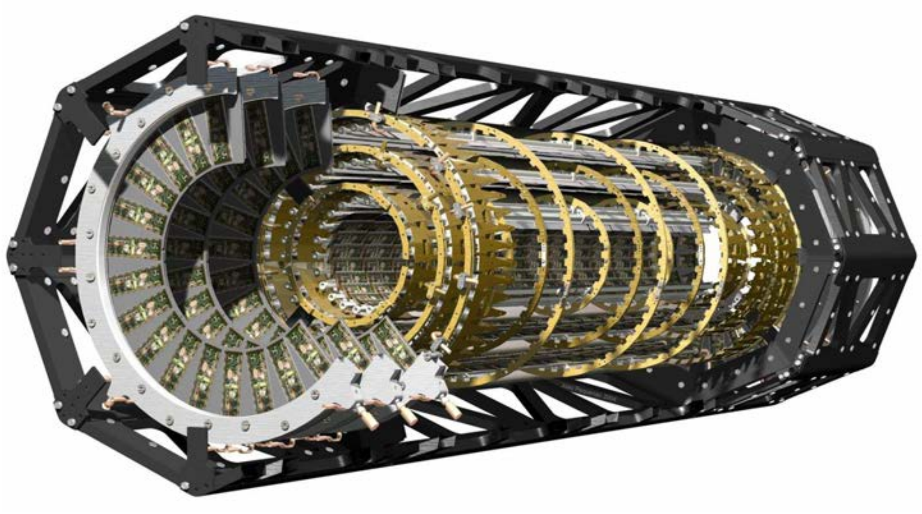


Figure 2.2: Drawing of the ATLAS Pixel Detector, showing the detector modules and support structures.

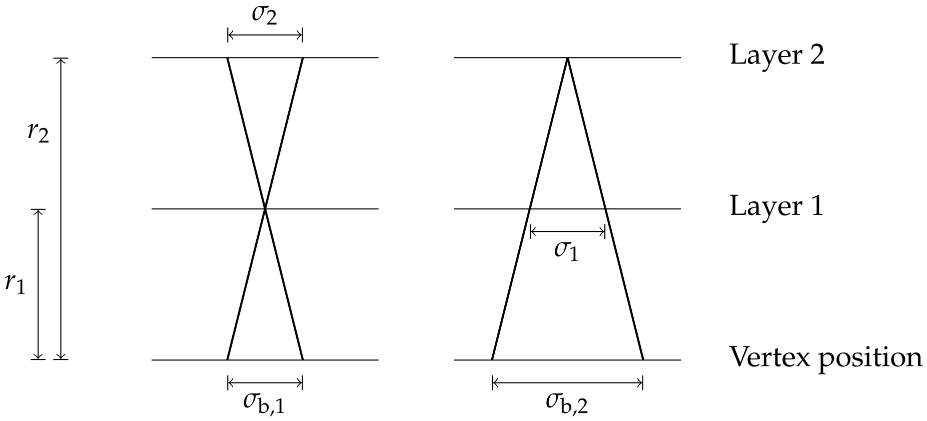


Figure 2.3: Error contribution from radii and resolution from pixel layers to vertex reconstruction.

is therefore displaced from the original interaction point. Thus, by following the tracks of several particles to a common point one is able to label these jets. This requires an accurate measurement of the vertex origin.

Assuming a two-layered pixel detector with planes at distances r_1 and r_2 from the interaction point with resolutions σ_1 and σ_2 yields error contributions by each plane of

$$\sigma_{b,1} = \frac{r_1}{r_2 - r_1}\sigma_2 \qquad \sigma_{b,2} = \frac{r_2}{r_2 - r_1}\sigma_1$$

(see Figure 2.3) [21]. The total resolution for vertexing σ_b, is therefore given by adding in quadrature, as contributions are uncorrelated:

$$\sigma_b^2 = \left(\frac{r_1}{r_2 - r_1}\sigma_2\right)^2 + \left(\frac{r_2}{r_2 - r_1}\sigma_1\right)^2 + \sigma_{\mathrm{MS}}^2, \tag{2.1}$$

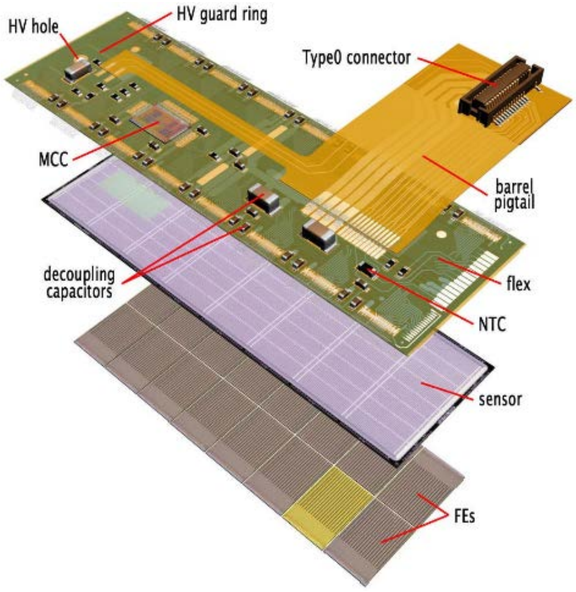


Figure 2.4: ATLAS Pixel Detector Module

where σ_{MS} is the multiple scattering contribution. It becomes clear that r_1 should be as small as possible for a good vertexing resolution, explaining the importance of the B-Layer. Another factor is the resolution of each detector $\sigma_{1,2,...}$. This resolution is dominated by the pixel pitch.

Each layer is divided into modules (Figure 2.4), each consisting of one silicon pixel sensor and several read out chips. The pixels span $50 \times 400\ \mu m^2$. Where the read out chips meet, the pixels are elongated to $50 \times 600\ \mu m^2$ along one axis and ganged together on the other. A turbine arrangement created by tilting the modules in the barrel and staggering on the end-cap disks ensures full coverage even with dead areas at the module borders. Aside from that, tilting the modules to the Lorentz Angle also (over-)compensates the motion of charges generated in the pixel detector due to the magnetic field.

The silicon sensors consist of n^+-implants in an *n*-type bulk. A pn-junction is established by the p-doped backside. Metal contacts above these implants allow for connection to a read out chip via solder bumps in a process called bump-bonding. Because of this separation, this type of pixel detector is referred to as a *hybrid pixel detector*.

The read out chip, often referred to as front end chip, is called *FE-I3* [25]. It offers a matching set of contacts to a charge sensitive amplifier and digitization circuitry for 18×160 pixels. 16 read out chips are bump-bonded to one sensor chip. Digitization is done by shaping the signal from the charge sensitive amplifier and measuring the time this signal is above a tunable threshold, called the Time over Threshold (ToT) which is proportional to the amount of deposited charge. To achieve a well-defined and uniform response, the threshold and the shaping time can be tuned. This is enabled by calibration circuitry for injecting known charges. The chip serial data outputs, together with control signals and connections for powering, are wire-bonded to a PCB on the back-side of the sensor which holds the Module Control Chip (MCC). There the data is aggregated and forwarded at

160 MBit/s (B-Layer), 80 Mbit/s (Layer 1) and 40 MBit/s (Layer 2).

2.2.2 Insertable B-Layer

The first upgrade to the ATLAS Pixel Detector is the Insertable B-Layer (IBL) [8]. Another layer is inserted between a new, smaller beam pipe and the existing B-Layer. The IBL was planned as it was expected that the performance of the pixel detector would degrade significantly until after the second long shutdown in 2016, where the IBL was supposed to be installed. Reasons for this include failures of pixel modules in the existing B-Layer and an exceedingly large event pile-up due to increasing luminosity. Additionally, another pixel detector closer to the beam pipe would increase the impact parameter resolution. As it became obvious that it was possible to finish IBL earlier, its installation was moved to the first long shutdown in 2013/14.

Apart from evaluating new sensor designs [19], such as 3D silicon sensors, in which the electrodes are built from columns vertically tranversing chip, in addition to the previously used planar sensors, the new read out chip *FE-I4* was developed [14]. It uses several techniques to increase radiation hardness, such as using an inherently radiation hard 130 nm CMOS process and triplicated digital electronics such as registers and logic.

Much more obvious is the size change from 7.8×10.8 mm^2 to 20.2×19.0 mm^2 while the pixel size is reduced to 50×250 µm^2 in an array of 336×80 pixels. At the same time, the inactive edge containing the end of chip logic and wire bond pads shrank from a width of 2.8 mm to 2.0 mm, increasing the active fraction of the chip from 0.74 to 0.89.

While the chip size increased, only two chips for planar sensors or one chip for 3D sensors form a module. This is due to the small radius and space available for the IBL, as it has to fit between the existing B-Layer at a radius of 5.5 mm and the new beam pipe and the already significantly increased width of the FE-I4. Additionally, there is no MCC anymore on the modules. The single data output link of each front end chip, which is now specified to operate at up to 160 Mbit/s and uses 8B/10B (see section 3.3.1) as channel coding, is directly routed off the module.

2.2.3 Upgrades

While this pixel detector configuration is expected to perform well over the LHC lifetime, the high luminosity LHC (HL-LHC) upgrades [27] will bring new challenges. The associated ATLAS upgrade plans [11] (Phase-II upgrade) for the inner detector foresee issues due to radiation damage, especially in the silicon strip tracker and bandwidth saturation in both the strip and the pixel detector. Especially in high transverse momentum jets, the silicon strip tracker and the transition radiation tracker will not be able to separate the jet contents.

Therefore, the inner detector is planned to be replaced in the Phase-II upgrade by an all-silicon tracker, with the inner layers provided by pixel, the outer layers by silicon strip detectors. The number of layers that use pixel detectors, set by [11] to four, is primarily limited by their cost. While strip detectors can be built from two separate sensor chips with read out electronics on their side, a pixel detector requires read out for every single pixel. This is done in hybrid pixel detectors using the rather expensive bump bonding process.

Pixel detectors outperform strip detectors in many aspects[1]. Thus, reducing the pixel detector construction cost is essential to increase the number of pixel detector channels is essential. There are several approaches, for example monolithic (or active) pixel detectors [24, 32], which move amplification (and processing) into the silicon sensor. Alternatively, this also allows to capacitively couple a signal into a separate read out chip, and thus avoid at least bump-bonding, which would be hard without an amplification on the sensor, due to the small signal amplitude and capacitance relationships between charge-sensitive amplifier and coupling capacitor.

The approach that leads to this thesis is to increase the module size. As the area that can be bump-bonded at once increases, the cost per area decreases and thus, larger modules also increase the cost efficiency. Currently, modules containing four FE-I4 chips are evaluated.

2.3 Beam tests

Performance tests are essential for all particle detector technologies, which of course includes four-chip modules. This can be done in a test beam [31] in which a pixel detector is tested under conditions replicate those in the experiment the detector is planned to be used in. A particle beam passes through the device under test (DUT) and induces a signal there. Additionally, it is tracked using a well-understood tracking detector, for example the EUDET telescope [7] consisting of 6 planes, each equipped with an active pixel sensor [22] achieving a spatial resolution of more than 4 µm and an impact resolution at the DUT of 2 µm (at DESY II, 6 GeV electron beam). By analysing the correlations between the fitted tracks and the DUT signal one is able to reconstruct performance parameters of the DUT, such as resolution and detection efficiency.

[1] Such as noise, as the noise at the output of a charge sensitive amplifier grows with the sensor capacitance, which is much larger for a strip than a pixel. Also, the performance under high occupancies is better, as pixel detectors do not suffer from ghosting effects.

3 Detector Read Out

Naturally, the DUT has to be read out during a beam test. As pixel detectors for high-energy physics are often designed to be read out from within a larger detector, read out schemes are non-trivial. On the other hand, due to its complexity, the final read out infrastructure is either not available in the prototyping phase or complicated to set up, which is not compatible with the limited time the test beam can be allocated for such tests at the respective facilities. Similar problems arise in other stages of development and production of a pixel detector, from wafer testing to commissioning of detector parts.

3.1 USBpix

To bridge the gap between development and final read out, laboratory read out systems are used. They are smaller and more comprehensible than the complete read out chain, support a subset of the features of the full read out system and read out only small portions of the full detector, for example one module. One of these systems is USBpix [4].

A representative USBpix set-up, such as shown in Figure 3.1, consists of a Multi-IO board, an adapter card and the DUT, mounted on a carrier card or contacted via a needle card during wafer probing. The Multi-IO board [2] contains the USB 2.0 capable microcontroller *CY7C68013A*, which establishes a USB connection to a host PC, controlling the USBpix unit and performing higher-level operations. The load on this link is reduced by an FPGA (Field Programmable Gate Array, see section 3.2) of type *XC3S1000* [16] which is able to perform time-critical tasks, such as histogramming, trigger logic and clocking of the DUT. Data-intensive operations are supported by an asynchronous static random-access memory (SRAM) of type *CY7C1061AV33* [15] adjacent to the FPGA, providing 2 MByte of memory for histogramming and buffering. The Multi-IO board also provides connections for receiving trigger data from the aforementioned EUDET telescope and the exchange of trigger pulses with other devices.

The adapter card, which is directly plugged into the Multi-IO board, provides electronic functions specific to the DUT. This includes powering the DUT, but also logic level shifting and conversion and analog to digital conversion. Apart from the adapter card shown in Figure 3.1, which supports one FE-I4 chip, another card, called *burn-in adapter card* is available (see Figure 3.2). It supports the simultaneous connection of up to four FE-I4 read out chips which all share the same clock and command outputs from the Multi-IO board, but the data received from each of them is handled separately (see Figure 3.3).

On the PC connected via USB, *STcontrol* controls together with the several libraries, such as the interface library *USBpixI4dll*, the higher level functions of USBpix. This refers mostly to storage of chip configurations, control of scan loops and calibration. Several different types of scans are provided there, which enable calibration and allow to tune the chip

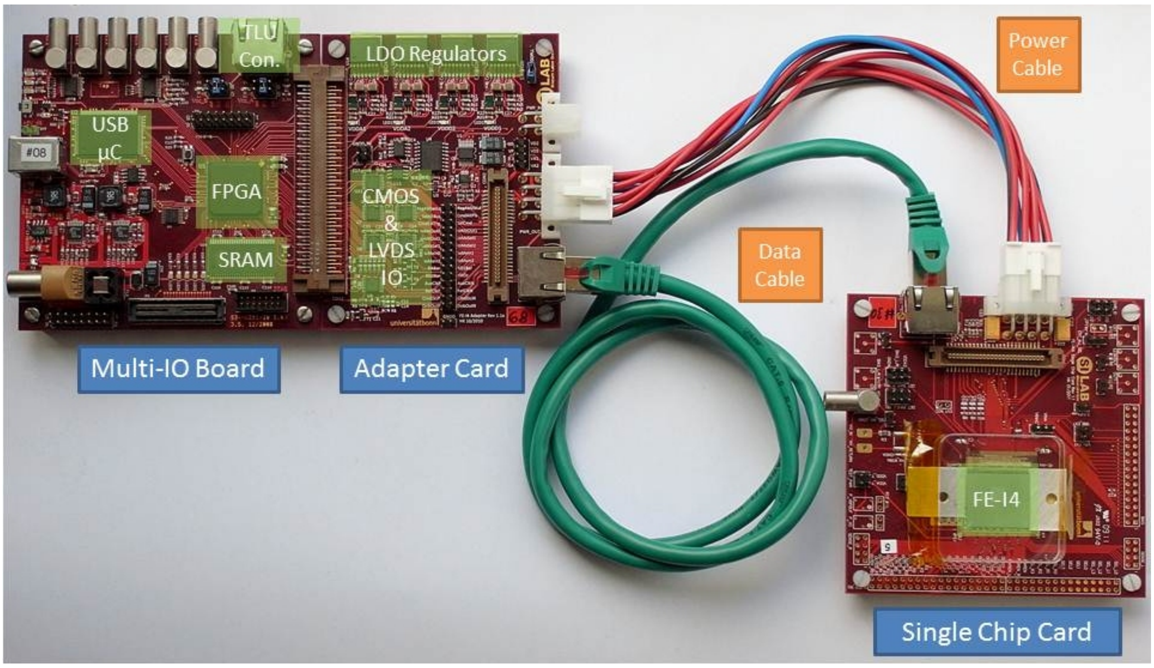


Figure 3.1: A representative USBpix set-up: On the top left is the Multi-IO board, providing USB uplink to a host PC and connects via an adapter card to a DUT, in this case a FE-I4, mounted on a single chip carrier card.

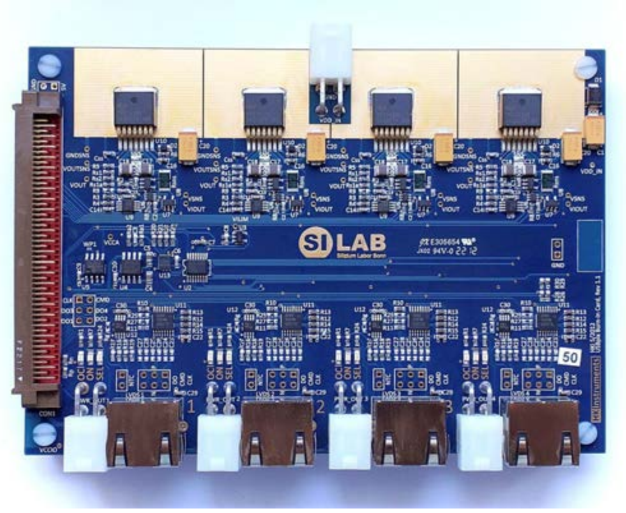


Figure 3.2: The burn-in adapter card, supporting connection of up to four FE-I4 chips simultaneously.

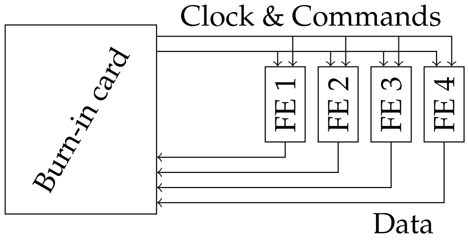


Figure 3.3: Burn-in card connection diagram: Clock and command lines to four FE chips are shared, data return lines are individual.

configuration as well as check for certain chip defects, such as problems resulting from defects during bump bonding. While many of these scans use the chips' internal mechanism to inject a signal into each pixel for calibration, others control the read out while data is generated by a physical signal such as from radioactive sources or a test beam.

Even though an adapter card to connect four read out chips is provided, parallel read out capabilities are limited. The original purpose of the burn-in card was to test multiple chips in a row without human interaction during IBL production, by only testing one chip at a time and disabling all others. For modules containing two chips, as they were already used in IBL, two USBpix units have to be used, each reading data from one chip, while one of them, the master, is responsible for sending clock and command to both chips. As a result, the data received by the slave is synchronised to the master's clock and the clocks of both USBpix units have to be synchronized.

3.2 Programmable Logic

The Field Programmable Gate Array (FPGA) working on the MultiIO board can be programmed to implement a large set of logic functions and save their results. As FPGAs are integrated circuits (IC), they can easily be able to operate at relatively high frequencies in the order of several 100 MHz[1]. The programmed configuration can be changed quickly, which makes the FPGA an excellent tool for prototyping high-speed logic.

Typically, an advanced FPGA offers a large set of different resources. These shall be discussed on the example provided by the FPGA used in USBpix, a Spartan 3 FPGA manufactured by Xilinx.

3.2.1 Registered Logic

The central resources in FPGAs are registers and look-up tables (LUTs). Look-up tables map a set of logic inputs $i_0, \cdots, i_{n-1}$ to one logic output q and are typically able to implement any logic function with n inputs in this way. Such a LUT is shown in Figure 3.4, along with a register. The register here is a D-type flip-flop, which stores the value of its input D when

[1] The clock frequency is limited by the dynamic interconnect, which significantly increases the propagation delay of signals compared to a static interconnect in an application-specific IC.

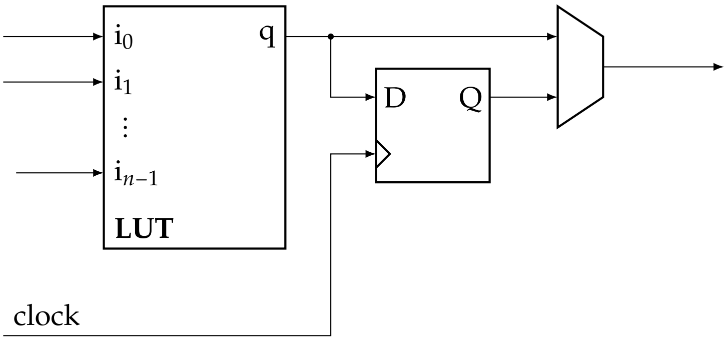


Figure 3.4: LUT with associated register and output multiplexer.

a rising edge on the clock input (marked with an arrow) occurs. Its output Q represents this state.

LUTs and registers are often paired up in this way in an FPGA, as this allows for fast storage of intermediate results without the need of routing them to a far register. It is clear, that such a block can easily be repurposed to serve as memory, by configuring the LUT to simply pass through one of its inputs. On the other hand, a multiplexer before the output of this block allows to circumvent the register and enables cascading of several LUTs to implement logic functions with more than n inputs.

Figure 3.5 shows how this is implemented on the Spartan 3 architecture. One can clearly make out two LUTs, with four inputs A[4:1] and two registers which can be fed with the outputs of these LUTs. In addition, a large set of additional specialized functions are provided, such as turning the LUT into a memory cell or shift register, carry logic which connects to neighbouring blocks (CIN, COUT) and a large set of multiplexers to choose from these different configurations. Also, the register has additional inputs, such as CE (clock enable), which makes it possible to disable updating the register state for selected clock cycles, and an SR (set, reset) input which allows to asynchronously[2] change the state of the register. It is worth noting that only some of the multiplexers have a control input, such as F5MUX, CYMUXF. This distinguishes elements which are configured while programming the FPGA and elements that can be controlled by logic values while the FPGA is active.

3.2.2 Interconnect and Specialized Resources

The in- and outputs of several slices are connected to a switch matrix, which is programmed by the FPGA configuration. The switch matrix itself is also connected to other switch matrices. Routing of logic signals inside the FPGA fabric is then implemented by connecting two ports of a switch matrix.

At the edge of this interconnect, special input- and output blocks (IOBs) contain the necessary circuitry to receive and transmit signals from the FPGA periphery. Often, as it is the case with the Spartan 3 architecture, these IOBs are versatile and support a multitude of different signalling standards, which include single-ended signalling such as LVTTL (Low

[2] not synchronized to a clock edge

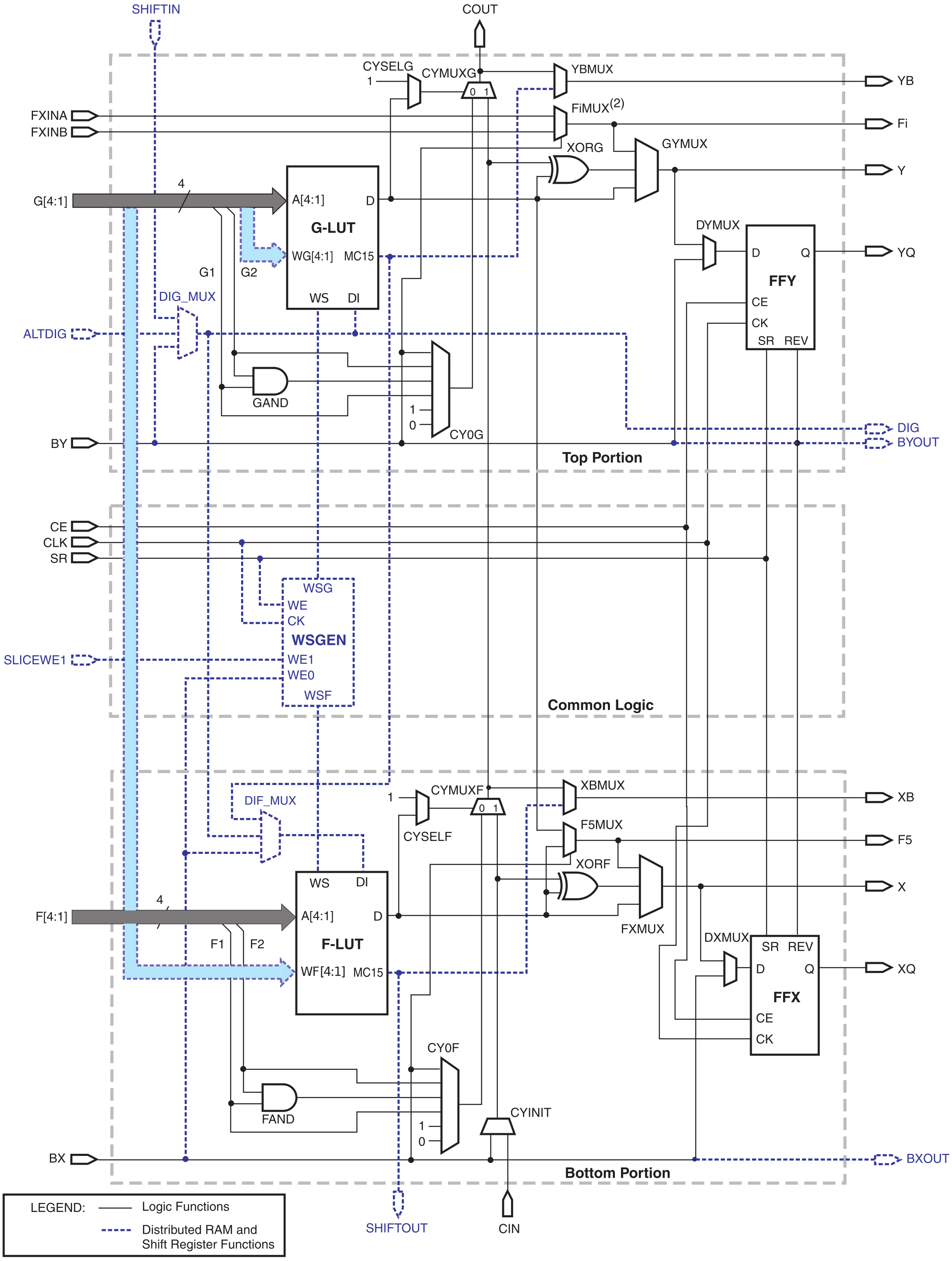


Figure 3.5: Two LUT-Register pairs, called a slice, as they are provided on a Spartan-3 FPGA [16].

Voltage Transistor Transistor Logic) and LVCMOS (Low Voltage Complementary Metal Oxide Semiconductor) at different reverence levels, but also many differential standards, such as LVDS (see section 3.3). In addition they contain a set of registers allowing for double data rate (DDR) operation.

While routing data signals via several switch matrices to their destination and thereby introducing a noticable routing delay is required to provide versatile routing capabilities on confined space, this is not the case for clocks; usually a large set of registers receives the same clock signal. Therefore, FPGAs such as the Spartan 3 architecture offer specialized clock interconnect which minimizes the clock skew between two elements receiving the same clock. However, as there clearly is a certain distance between two edges of the FPGA and the large fan-out of the clock signal requires several stages of amplification, this can only be achieved at the cost of introducing a propagation delay, which is very large in comparison to the delay of the general purpose interconnect.

There are two reasons this does not pose a problem. First of all, the clock signals are periodic. Thus, for register transfers inside the FPGA, it does not matter, which of the clock edges arrive, as long as it is predictable when they do. Secondly, FPGAs tend to provide clock management facilities: The XC3S1000 at hand contains four Digital Clock Managers (DCMs). Each DCM is able to feed several lines of the global clock network with a periodic signal while compensating for its delay by receiving a feedback signal, which is the same signal that the registers receive. Thereby, the delay of a clock network can be completely ignored and instantaneous transmission of a clock signal from the clock input to the registers be assumed.

Another very important feature of these DCMs are the clock synthesis capabilities. Each DCM is able to generate phase-shifted copies of the received clock, first of all at fixed intervals of 90°, 180° and 270°, but also at finer phase shift steps between 30 ps and 60 ps, where the amount of such steps can be modified by the general purpose logics while the FPGA is active. In addition, it is able to generate a new clock signal (CLKFX) with frequency f_g derived from an existing clock with frequency f_o as long as the frequencies are related as

$$f_\mathrm{g} = \frac{M}{D} f_\mathrm{o}$$

with $M \in \{2, 3, \cdots, 32\}$, $D \in \{1, 2, \cdots, 32\}$.

A last specialized block to be introduced here is block RAM. This refers to sections of the FPGA with addressed read and write access to memory cells, which can be accessed with much smaller delays than external memory. FPGA vendors also provide readily designed logic that work with block RAM (or alternatively general-purpose logic cells) to generate memories of different sizes and data widths. This especially includes FIFO (First In First Out) memories, which are often used in FPGA configurations for buffering data. Another purpose of FIFOs is to provide a clean transition between clock-domains of unknown or complicated clock phase relationships, where it must be ensured that data is handed over from one register to another in a stable fashion.

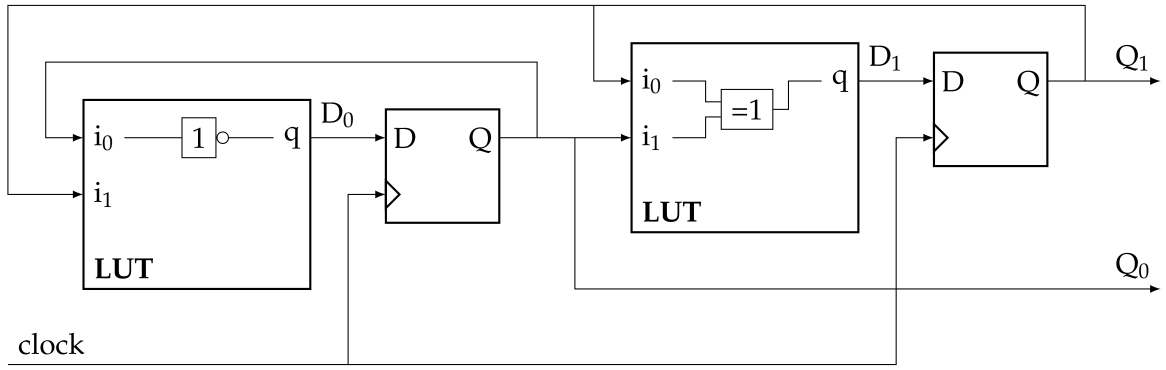


Figure 3.6: A synchronous counter with LUTs and registers to demonstrate an RTL design.

```
D0 <= not Q0;                          1
D1 <= Q0 xor Q1;                       2

process (clock)                        4
begin                                  5
    if rising_edge(clock) then         6
        Q0 <= D0;                      7
        Q1 <= D1;                      8
    end if;                            9
end process;                           10
```

Listing 3.1: Example for a circuit description in VHDL on the RTL. Declarations and module context have been left out.

3.2.3 FPGA Design

Digital systems are often highly abstracted. On the electronic level, they are based on the operational characteristics of transistors. From there, transistors are aggregated into gates, such as AND or NOT. While gates are fine for logic functions, states are created by feedback of gate outputs. These constructs are abstracted into registers, such as the D-type flip-flop used above. Further abstraction yields isolated functional blocks.

FPGA design is often done on the last three of these and higher levels of abstraction. Most prevalent is programming on the *Register Transfer Level* (RTL). Between rising edges of a common clock, the result of a logic function of a set of inputs from registers are computed and saved to another register. An example for this using the resources available on an FPGA is shown in Figure 3.6. Depicted is a synchronous counter: On every clock edge, the state (Q_1,Q_0) counts through $(0,0),(0,1),(1,0),(1,1),(0,0),\cdots$.

A common way to describe such an RTL design is a hardware description language (HDL) such as Verilog or VHDL (Very High Speed Integrated Circuit HDL). Listing 3.1 shows a VHDL code fragment generating the counter from the example above. In lines 1 and 2, the values D0 and D1, corresponding to the diagram, are computed from Q0 and

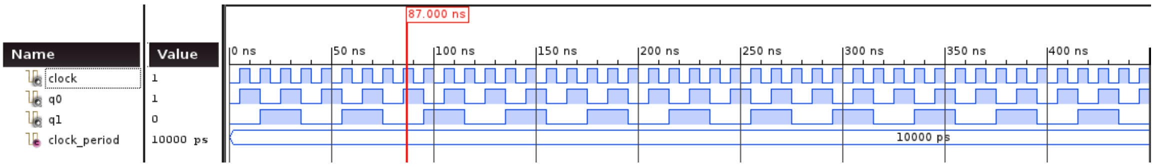


Figure 3.7: Simulation of the counter example.

Q1. Once a rising edge on the clock occurs (line 6), the D0 and D1 become the new values of Q0 and Q1 (lines 8 and 9) and D0 and D1 is computed anew. Having a description in a formal language allows not only to synthesize the counter to an FPGA configuration, but also to simulate it. This is shown in Figure 3.7.

While creating FPGA designs on the RTL and higher levels of abstraction is comfortable, it masks the delays introduced by routing and active elements inside the FPGA. These are problematic, as one assumes that all calculations are instantaneous during behavioural design of logic elements. Thus, it has to be ensured that the logic functions do not become too complex to be completed within one clock period. More stringently, registers require a stable signal over a certain period of time around a nominal rising edge. This period is influenced by the set-up and hold times, the time a signal must be valid before the clock edge and has to be kept stable afterwards, which are intrinsic properties of the register, and the period jitter of the clock source. By providing a clock period and period jitter, the toolchain responsible for compiling the code above into an FPGA configuration, keeps track of these timing requirements and warns if any of the given constraints could not be fulfilled.

3.3 FE-I4 read out

The FE-I4 read out chip [14] has already been introduced in section 2.2.2 as the front end chip used in the Insertable B-Layer upgrade and more importantly also employed in the four-chip modules considered for the Phase-II upgrade.

The FE-I4 chip is read out using three serial links: A 40 MHz reference clock, a common command and trigger link and a data output. The first two are generated by the read out system and sent to the FE-I4 while the latter contains data and responses to commands. All three links use differential signalling, in particular oriented at LVDS (Low Voltage Differential Signalling), which is supposed to be operating at $V_{\mathrm{CM}} = 1.2$ V common mode voltage and a differential current of ±3.5 mA for signalling. However, as a single digital voltage supply close to V_{CM} is used for the FE-I4, the actual common-mode voltage is significantly lower.

Commands are sent synchronously to the clock at 40 MHz without any specific channel coding. They are divided into two groups. The first contains *slow* commands, which are used to access registers on the FE chip and reset. These commands typically carry a large amount of data and contain a chip address field, which can be used to select one chip, if several chips share one command link. The other group contains *fast and trigger* commands, which transmit trigger and calibration signals to the read out chip and synchronize

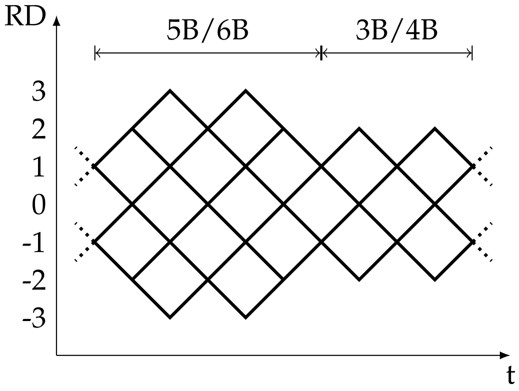


Figure 3.8: Possible values for the running disparity during transmission of a 8B/10B symbol.

its internal counters, which are used to identify events after read out. These commands are much shorter and neither carry an address nor data as payload.

Between commands, zeroes are transmitted as idle bits. While this is fine for DC-coupled transmission, if AC-coupling through capacitors or a transformer is used, a continuous transmission of the same bit, called a *run*, results in subtraction of this value as the DC offset. AC coupling is typically used in advanced powering schemes, such as serial powering, where the reference potential differs between two modules, but both modules have to be handled by the same read out system. It may also be required by the physical transmission scheme.

As the sampling point is well-defined and the data rate is only 40 MBit/s, Manchester encoding can be used. This means, that each clock period is split up into a first half, in which the actual bit is transmitted and sampled by the read out chip, and a second half, in which the inverted bit is transmitted, but ignored by the read out chip. This way, not only the run length is kept minimal, but also the DC portion of the signal is kept centered between the high and low signal. If this is the case, a transmission is called *DC balanced*, which ensures that the signal can be transmitted via AC-coupled elements, as long as the data rate is sufficiently high as compared to the time constant of the coupling elements.

3.3.1 8B/10B

Manchester coding the transmission is practical for the command and trigger line, where only small amounts of data have to be transmitted or the timing requirements are relaxed, as it effectively requires the transmission of two bits for every bit to be transmitted.

For the data line, however, a high net data rate is desirable, as the read out chip only has limited buffering capabilities. As a data rate of 160 Mbit/s is already used, increasing it further to 320 Mbit/s or reducing the net data rate to 80 Mbit/s, as it would be necessary for using Manchester code, is not acceptable.

One of many alternatives to this is provided by the 8B/10B channel code [33]. Here each block of 8 bits is encoded as a block of 10 bits. DC balance could be achieved in this scheme by only taking those 10 bit blocks, which have an equal number of zeroes c_0 and

symbol name	code for		FE-I4 usage
	$RD = -1$	$RD = +1$	
K.28.1	0011111001	1100000110	idle
K.28.5	0011111010	1100000101	end of frame
K.28.7	0011111000	1100000111	start of frame

Table 3.1: The 8B/10B comma symbols, their code representations and their usage by the FE-I4 read out chip.

ones c_1. As with in the Manchester code, this would ensure DC balance after a full symbol is transmitted. As there are only $\binom{10}{5} = 252 < 256 = 2^8$ such blocks, 8B/10B also has to take blocks with a *disparity* of $c_1 - c_0 = \pm 2$ into account. Two such symbols are provided for each source symbol, one with $c_1 - c_0 = 2$ (positive disparity) and one with $c_1 - c_0 = -2$ (negative disparity). The 8B/10B encoder then keeps track of these symbols in form of the sum of all ones minus the sum of all zeroes minus one[3], called the running disparity:

$$RD = -1 + \sum_{\text{symbols } i} c_1(i) - c_0(i)$$

If a source symbol has to be encoded and $RD = -1$, the symbol with positive disparity is used as the code symbol, leaving $RD = +1$ after encoding and vice versa.

A side effect of the extension to code words with non-zero disparity is that an additional $\binom{10}{4} = 210$ code words are available. This already includes that two code symbols are required for encoding, one with positive and one with negative disparity. By careful selection from the complete set of 462 possible words, one can additionally limit the running disparity even within code words and simplify the encoding process by splitting the code into a 3B4B and 5B6B code. With 8B/10B, RD is effectively limited to ±3, which is shown in Figure 3.8.

In 8B/10B, this leaves twelve symbols unassociated. These are referred to as control symbols and can be used for in-band control signalling, such as beginning and end of valid data, a process called framing, and the transmission of idle symbols in between. A subset of these symbols, called comma symbols, can be used by the receiver to *align* to the 10 bit boundary that separates symbols, as they are chosen such that there is no way to detect them as part of an overlap of two other symbols. A list of the comma symbols can be found in Table 3.1. The comma symbol K.28.7 has a special role: If it is used, alignment is only possible on two subsequent comma symbols and repetitive use of K.28.7 has to be forbidden, as it would be possible to incorrectly detect K.28.7 in an overlap region between K.28.7 and other symbols[4].

On the data line, the FE-I4 uses 8B/10B encoding with all three available comma symbols for signalling. While no data is available, a K.28.1 symbol is transmitted. Each block of data is announced by K.28.7 as *start of frame* (SOF) and concludes with a K.25.5 as *end of frame* (EOF). Two consecutive commas, such as two idle words, idle + SOF or EOF + SOF can be used for alignment.

[3] This is an arbitrary start value for the running disparity to centre it around 0 and could as well be +1

[4] This can be easily seen when concatenating K.28.7 and K.28.7, which results in the bit stream *00111110000011111000*. When now searching for K.28.7, it is possible to find one offset by 0, 5 or 10 bits from the start of this sequence. This clearly does not identify a 10 bit boundary.

Record Type	Field 1	Field 2	Field 3	Field 4	Field 5
DH	11101	001	Flag	LV1ID	BCID
DR	Column		Row	ToT 1	ToT 2
AR	11101	010	Type	Address	
VR	11101	100	Values		
SR	11101	111	Code	Number	

Table 3.2: The set of records available in 8B/10B framed transmission (Field widths do not align within columns).

3.3.2 Data Protocol

The underlying data protocol is grouped into records of 24 bits, thus each record can be encoded in three 8B/10B symbols. Five different record types are used. The list of records is shown in Table 3.2.

A characteristic data frame consists of a data header (DH) with an arbitrary number of data records (DR) and a small set of service records (SR). The data header identifies the bunch crossing and trigger id all subsequent data records are assigned to. The data records then contain the column and row address of a hit, together with a ToT value of this pixel. In addition, if a hit in the adjacent pixel with increased row number has been observed, this ToT value is also included[5]. Service records are used to inform about errors or provide additional information. Address (AR) and value records (VR) do not occur during normal detector operation and deliver the address and values of configuration registers.

Records are uniquely identifiable by their first two fields. All records except the data record start with 11101. The data record directly starts with the 7 bit column address. The maximum column address is 80 which is encoded as 10100 00, where the first five bits align to the identifier for the other records. As these can never reach a value of 11101 data records do not need a distinct preamble, which is reasonable as they will be the major record exchanged during operation. Distinguishing between the other records is then done based on the identifying value in the second field.

3.3.3 Protocol Optimizations for Low Data Rates

Modules containing four FE-I4 chips are especially planned for layers that are not particularly close to the interaction point. As the occupancy drops as the radius squared, the per area data rates between inner and outer layers are going to vary significantly, while the number of links in the same area is fixed to one link per FE-I4. This leads to a significant mismatch between link capacity and net data rates and thus additional unnecessary material inside the detector.

Previously, with the FE-I3 and MCC solution, the amount of links could be reduced from two to one in the outermost layer. This is not the case with FE-I4, as no device which could aggregate any data is foreseen with the FE-I4 architecture. Alternative approaches are therefore discussed.

[5] This is a feature called ϕ-pairing, which results from the non-uniform aspect ratio of the pixels, which are shorter in ϕ than in z, as seen in the detector coordinate system.

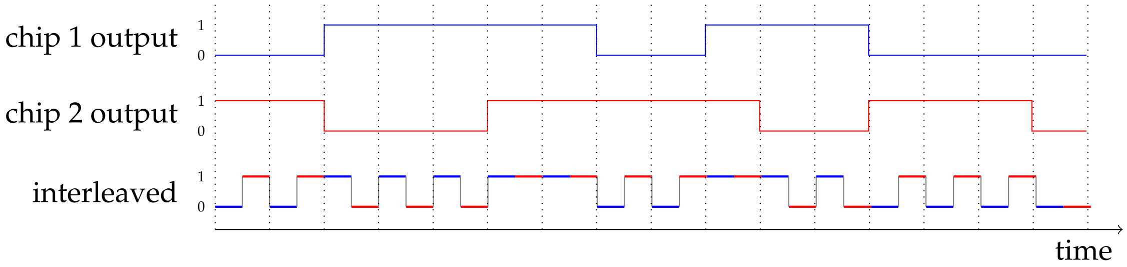


Figure 3.9: Interleaving two chips to one line. The chip data is differently coloured, blue show data from chip 1, red from chip 2.

The most simplistic of these is interleaving the data of several chip on one data line. This is shown as an example for two chips in Figure 3.9. In each original bit period, separated by dotted lines, two bits, one from each chip is transmitted. As the average of two DC-balanced signals is DC-balanced, the interleaved output is again DC-balanced.

There are several approaches to doing interleaving. First of all, the outputs of several FE-I4 chips could be shorted while only one of the chips enables its transmitter at a time. Alternatively, the interleaving logic can be separated and positioned, like the MCC, on the module.

As this is one of the methods considered for the Phase-II upgrade, it will have to be implemented in USBpix.

4 Stable USBpix Discussion

In preparation for any discussion about improvements to USBpix, the relevant parts of the stable USBpix implementation shall be discussed. The hardware is already sufficient to handle four chips with the burn-in card providing the appropriate connections. The next chapters will also show that the performance of the other components are sufficient.

The software can be divided into the USB microcontroller firmware, the FPGA configuration and the PC software. From a read out perspective the microcontroller is only responsible for providing a bulk transfer interface for SRAM data and writing into the FPGA's configuration register. As this is not supposed to change, the firmware can remain in its current state. The PC software has already been able to handle modules with many chips, as this was required for the FE-I3 modules. Also, some support for multiple FE-I4 chips is available for modules containing two FE-I4 chips, even though the concept of using two USBpix units is different from using only one unit for four chips. Still, the changes required in the host software are rather minute or straightforward and therefore not going to be discussed here.

Major changes are required in the remaining FPGA configuration. The main blocks in the top module of this configuration are the USB interface, clock generation, the read out block, control strobe and configuration state machines and trigger id control. These are supplemented by additional smaller blocks like the SRAM interface and the command Manchester encoder. Trigger id control, handling trigger-related work, like inserting a trigger-id into the FE data stream, and the control strobe and configuration state machines, responsible for configuring and sending triggers to the FEs, are rather unrelated to the problem at hand. The reason for this is that a full module is still triggered the same way as a single chip and strobes are sent to all chips on a single command bus as they do not contain addressing information. Due to the common command bus, configuration can not be parallelised with the burn-in card.

The most important parts for this thesis are the clock generation and read out chain. A block diagram of the latter is shown in Figure 4.1. After an input multiplexer, which exclusively selects one of the FE data outputs, synchronization, decoding and histogramming follow. These blocks shall now be explained in detail.

4.1 Input Selection and Synchronization

All four FE data signals from the burn-in card are received by the FPGA. The FPGA configuration selects exactly one of these for further processing using a multiplexer controlled by the host software.

The first task performed on the selected signal in the read out chain is synchronization, a process in which a data signal, which was synchronous to some clock at its source, has

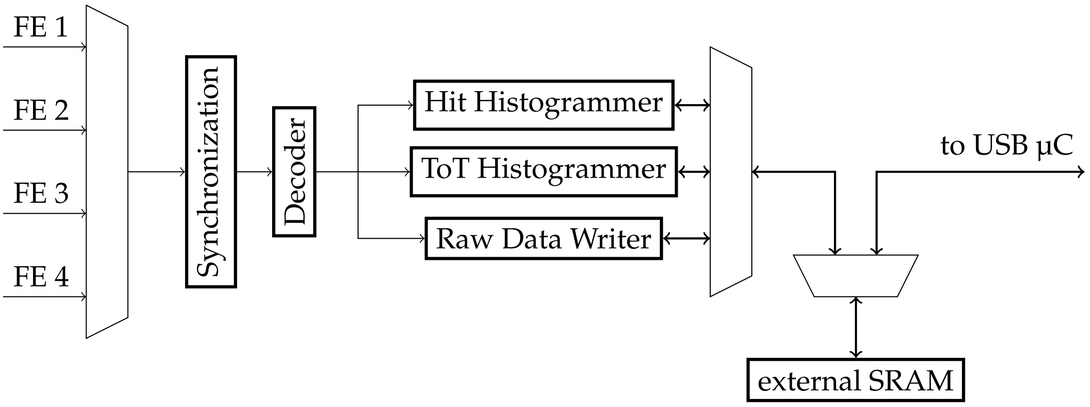


Figure 4.1: Read out Chain Block Diagram

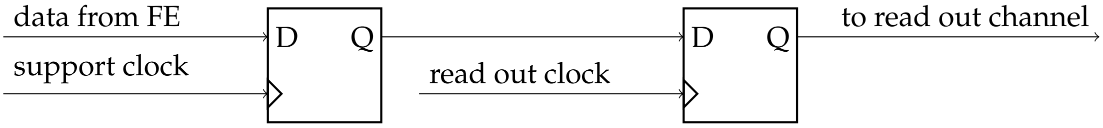


Figure 4.2: Sampling with support clock

to be sampled on its destination, usually without knowing more than the clock frequency. Synchronization is done statically with a phase parameter measured once after chip configuration when the chip enables its LVDS drivers: An additional support clock with a stable phase relationship to the read out clock and thus also to the data is generated. This clock is used to sample the data once, before it is handed over to the subsequent part of the read out channel (see Figure 4.2). By changing the phase shift of this clock, one is able to also delay the data over a notable range.

By adjusting the support clock phase while the chip transmits a known pattern on the data line and checking the received pattern against the expectation one can obtain a relationship between the amount of transmission errors and the clock phase of the support clock. Two peaks are expected in this relation, at the point where the rising edge of the support clock coincides with the edges of the data. The phase of the support clock is, with an offset determined by the flip-flop performance, also the phase of the delayed data. Thus, a second peak arises once the shifted data edges coincide with the read out clock. Knowing this relationship one can place the phase of the support clock in the center of the widest error-free region.

This approach is only successful if the clock at the data source, the FE-I4 clock, is phase-stable to the USBpix clocks. With only one chip and one USBpix unit this is the case, as the FE-I4 derives the data clock with integer multiples of the 40 MHz reference clock from USBpix. Also, this is the reason why in the two-USBpix-unit configuration to read out modules with two chips, the clock sources of both USBpix units have to be synchronised.

4.2 Decoder Structure

The synchronized data is processed next by a set of decoders. The stable USBpix FPGA configuration contains a set of four of these. Two decoders are able to decode 8B/10B, two just decode the FE-I4 protocol [14]. Each of these two groups contain one decoder for 40 Mbit/s and 160 Mbit/s operation. This is what allows decoding the different speed and encoding modes.

If 8B/10B is enabled, the decoder deserializes the data into a 20 bit wide shift register. The contents of the shift register are then continuously checked for two consecutive 8B/10B idle words (*K.28.1*). If this combination is found, it is used to align the subsequent data stream. This means that every 10 received bits after this event, the shift register contents are examined further. First of all, if a multiple of 10 bits have been deserialized and the shift register contains a start-of-frame word (*K.28.7*), the following words are considered to belong to a frame which has to be ended with an end-of-frame word (*K.28.5*).

Secondly, the frame data is decoded from the 10 bit 8B/10B symobols by a purely combinatorial 8B/10B decoder, which is constantly decoding the most recent ten bits in the shift register. Thus (from a behavioural point of view) the received data is correctly decoded each ten clock cycles counting from the start-of-frame detection. The decoded bytes are packed into groups of three which constitute one record of the FE-I4 protocol. These are then handled by the subsequent read out modules.

While this worked in practice, it will later be important that for 160 Mbit/s operation the decoder fails the timing simulation by nearly one additional clock period. The cause for this lies in the amount of levels of the logic required to decode 8B/10B purely combinatorially.

While operating without 8B/10B, frame detection is performed differently: A frame is considered to start whenever the bit sequence 11101_2 is received (see Table 3.2). This is sensible as all records start with that sequence, except for the data record which is not allowed at the start of a frame. A frame ends, whenever a (hypothetical) record starting with 0000000_2 is received, as this is always forbidden in valid records and thus indicates the omission of further transmission. Apart from that difference, three consecutive bytes are again grouped to form FE-I4 protocol records.

4.3 Histogrammers and event read out

Finally, the FE-I4 protocol records are processed by one of the read out modules. The choice which one is used is based on the data required for the respective scan. Two of the read out modules are histogrammers: A hit histogrammer (*scan_readout*) creates a histogram of the hit counts per pixel. Alternatively, a Time-over-Threshold histogrammer divides the pixel-bins further into ToT bins. Being able to directly create a hit histogram, which is just a projection of the ToT histogram, avoids having to perform the projection on the PC.

The third read out module directly writes the decoded FE-I4 data stream to the SRAM, while inserting trigger data when a trigger is received. The interpretation of this data stream is then performed in the USBpixI4DLL or specialised tools located further up-stream. This is generally required for continuous scans like source scans or test beam measurements. It is also required if additional data apart from hit and ToT information is required, for example while reading back values of chip registers.

While the data input to the histogrammers is trivial and consists of just a 3 byte input and a flag signalling valid data, the memory access implementation is rather peculiar. The SRAM and FPGA I/O tristate control lines are routed via a bidirectional multiplexer to each of the read out modules. One of the read out modules is then selected and has sole control over the SRAM. Alternatively, the multiplexing structure is also able to allow access from the USB microcontroller interface, which is how bulk data like histograms and raw data are transferred to the PC.

4.4 Clock Generation

The USBpix FPGA configuration requires a small set of clocks, all of which are derived from a 48 MHz clock generated by the USB microcontroller. Apart from this 48 MHz clock for synchronous communication with the microcontroller, a 40 MHz clock is required. This clock is first of all sent to the FE, but also used for large parts of the FPGA, for example some sections of the read out modules and the 40 Mbit/s-mode decoders. Additionally two 160 MHz clocks are required; one static and one which can be phase-shifted with respect to the former. The static clock is mainly used to decode data when the FE transmits at 160 Mbit/s. The dynamic clock is required for the synchronization scheme detailed above.

Figure 4.3 shows a simplified schematic of the clock generator block. The incoming clock with frequency $f_{\mathrm{FCLK\_IN}} = 48\,\mathrm{MHz}$ is directly routed from the I/O pad to a DCM. This DCM is responsible for feeding the clock into a global clock network labeled FCLK and due to the feedback also eliminate the distribution delay of this network. This is also done with all other clocks mentioned in this section. So as long as no dynamic phase shifting is used, all clocks are phase-synchronized. The DCM is also configured with a division of 6 and multiplication of 5 for the CLKFX output, which yields a new clock labeled XCK with

$$f_{\mathrm{XCK}} := f_{\mathrm{CLKFX}} = \frac{5}{6} \cdot f_{\mathrm{FCLK\_IN}} = 40\,\mathrm{MHz}.$$

Starting from the FCLK clock, a second DCM derives with appropriate multiplication the static FCK clock with

$$f_{\mathrm{FCK}} := f_{\mathrm{CLKFX}} = \frac{3}{10} \cdot f_{\mathrm{FCLK}} = 160\,\mathrm{MHz}.$$

For generating the phase-shiftable 160 MHz clock labelled SYNC_CLK, FCK is used as the input for another DCM. This DCM exposes its dynamic phase shift interface to a phase shift controller which is in turn controlled by the PC.

It should additionally be noted that all three DCM implementations have the reset line constantly pulled down. As a result, the DCMs rely on the global reset of the FPGA, which is released directly after configuration. This is not a problem for the first DCM, as the FPGA is typically configured by the microcontroller which also generates FCLK_IN.

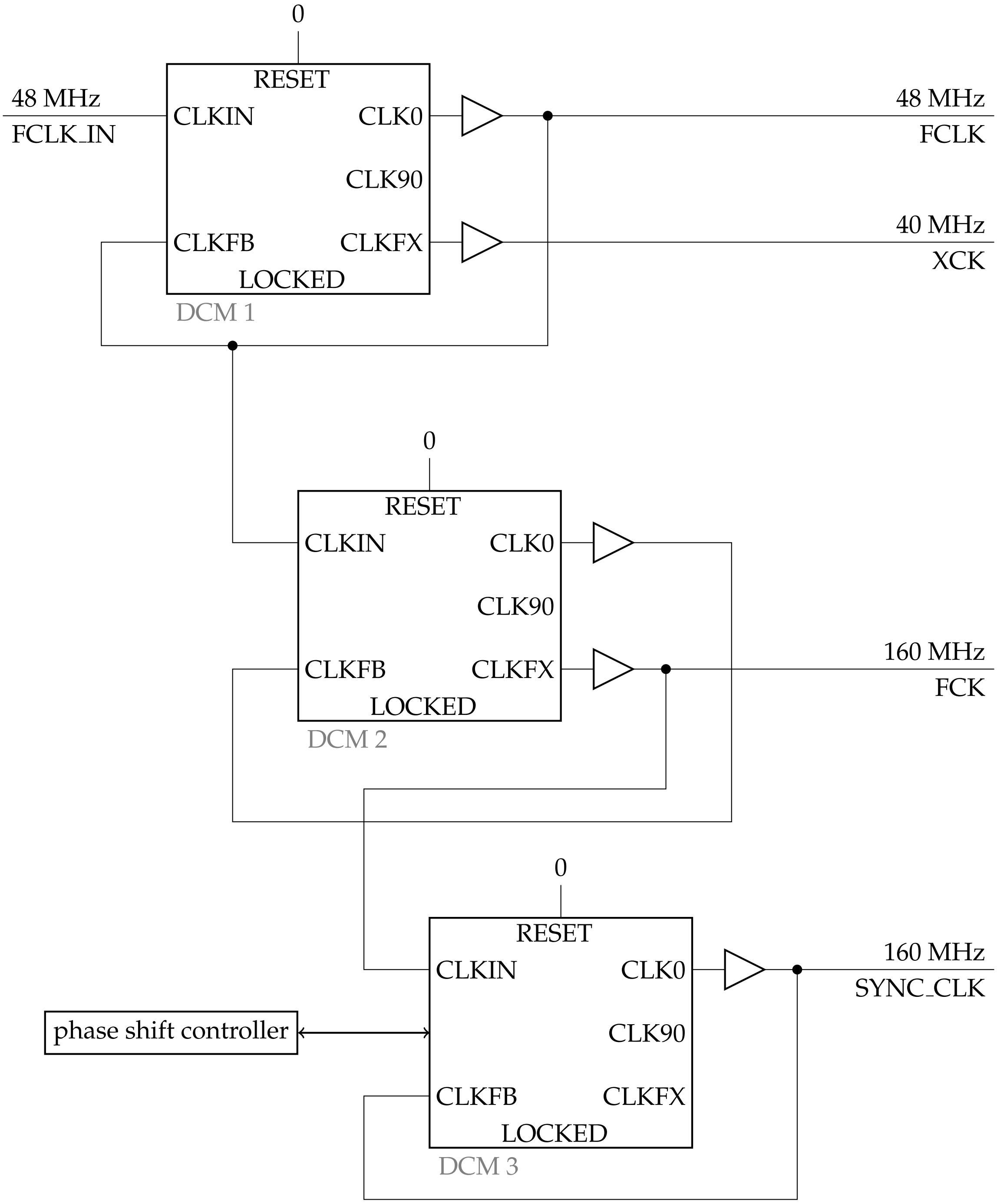


Figure 4.3: Clock generator block of the stable firmware. Three chained DCMs generate three fixed and one phase-shiftable clocks.

On the other hand this means for the other two DCMs that their reset is released while there is no stable clock at their input as the previous DCMs have not yet locked. This is a non-optimal state for the DCMs, as will be presented later.

5 Design

The overview of the existing configuration now enables the creation of a set of required improvements. This chapter will again focus on the read out chain, as the rest of the configuration does not require major improvements. The new configuration shall first of all be able to process data at full rate from four chips in parallel. As single-chip read out is already supported at that data rate, this involves mostly replicating the existing scheme and adapting it where simple replication is not possible. Additionally, the design shall involve the ability to process data on shared channels. In particular, bit-by-bit deinterleaving should be possible.

An overview over the proposed design to match these requirements is shown in Figure 5.1. Compared to the stable USBpix configuration (see Figure 4.1) the changes seem minute, as they are hidden inside each block, and the structural differences involve only three areas. The most obvious change is that three (mostly) identical read out chains are available. To maximize channel independence, an external memory arbiter has been added to the design which merges the outputs of the read out chains and handles SRAM access. In addition, a switch has been inserted. This allows for one channel to send data for processing to other channels, which is required for shared channel operation and for deinterleaving in particular. Apart from these structural changes, some modules will also be replaced. Static synchronization will be replaced by directly recovering the data. For reasons explained later in this chapter, this recovery scheme prompts a rework of the subsequent decoder section. Also, the read out modules are stripped of all direct memory functions, which are moved into the memory arbiter.

5.1 Synchronization

The static recovery scheme explained in section 4.1 is obviously problematic. Replicating it completely would require four independently phase-shiftable clocks. This is not achievable, as only one additional DCM is available and each DCM can only generate one of these clocks. An alternative approach would be to generate four clocks with 0°, 90°, 180° and 270° phase shift with the same dynamic phase shift added to each of these clocks. This would be possible without the usage of additional DCMs, the error-free region could be investigated for each of these inputs with the dynamic phase shift and one of the clocks mentioned above will be somewhere in an error-free region. However, this scheme is still static and requires a complicated initialization procedure. Also, data without a stable phase-relationship with the clock USBpix sends to the FEs still would not be sampled correctly.

The approach chosen for the new USBpix FPGA configuration is data recovery, a scheme that is dynamic and comes completely without any initialization procedure. The key con-

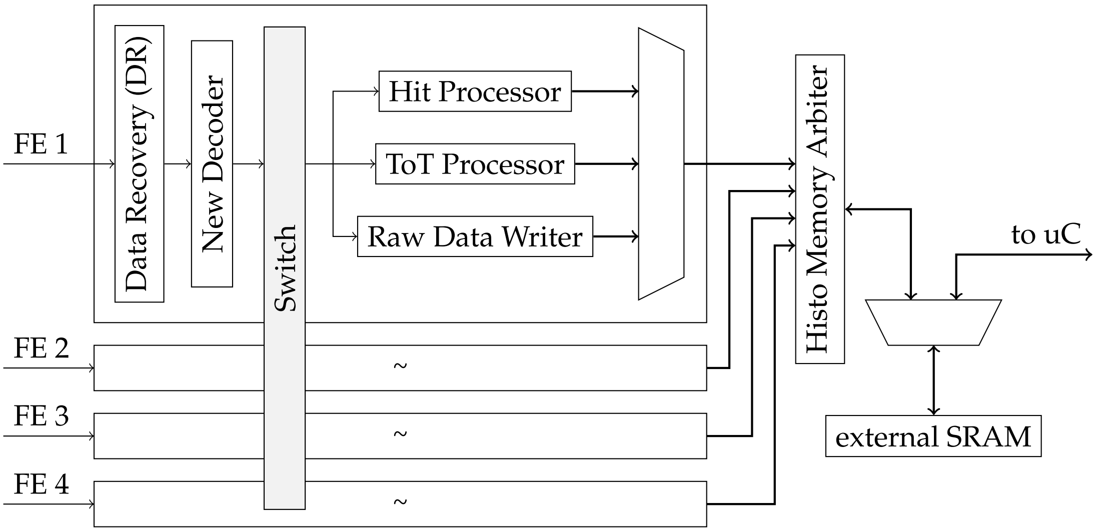


Figure 5.1: Block overview of the proposed new read out channel configuration. The read out sections for FEs 2 to 4 are the same as the one assigned to FE 1.

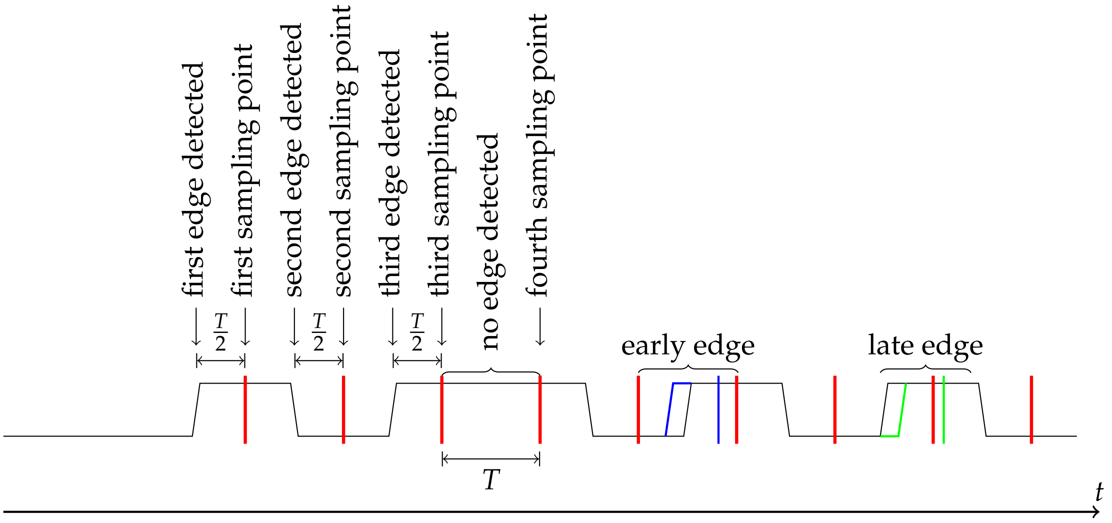


Figure 5.2: Data Recovery Scheme: The first three samples are fixed to half a period T after the last edge, the fourth sample is not directly preceded by an edge and thus is taken one period after the last sample. On the left, the effect of early and late edges is shown in blue or green, respectively; the sample points are then shifted accordingly.

cept of data recovery is to search for edges in the data stream. Once an edge was detected, a certain amount of time, ideally half a clock period, is let pass to be certain to have left the data line transition region, before a data sample is forwarded to the output of the data recovery module. If no edges are detected, a data sample is taken every time another full clock period after the last sample has passed. An example waveform is displayed in Figure 5.2.

Transferring this scheme to an FPGA requires a synchronous implementation. For detecting edges synchronously, one just takes two consecutive samples and calculates the XOR (exclusive or) between them. This limits the granularity of the edge detection to the sampling period T_s. Thus we have to take at least two samples per bit. An additional sample is required which is not one of the other two samples that contains the valid data. Additionally taking a clock period jitter j into account constraints the sampling frequency in respect to the data rate r as

$$T_\mathrm{s} < \frac{1\,\mathrm{bit}}{3 \cdot r} - j.$$

j is the quadratic mean of clock and data jitter. For a data rate of 160 Mbit/s, this results in a sampling frequency $f_s \gg 480$ MHz. However, increasing the sampling frequency further gives more freedom in choosing the correct point to read the data sample. Therefore a data recovery implementation with 4× oversampling will be presented in section 6.1.1.

A side effect of this scheme is that the distance between the samples is allowed to vary if edges are not detected at the expected rate but more or less often. This could either be instantaneous as a result of clock or data jitter or the FE clock may not be phase-stable to the USBpix clocks. Both situations occur in USBpix operation and thus have to be handled. While the effect of a data rate that is slightly off does not seem large this manifests as an instantaneous change of the relative data rate from $1\,\frac{\mathrm{bit/s}}{\mathrm{Hz}}$ to $0\,\frac{\mathrm{bit/s}}{\mathrm{Hz}}$ or $2\,\frac{\mathrm{bit/s}}{\mathrm{Hz}}$, respectively. This has to be handled appropriately, for example by outputting two bits of data with two associated bits that mark data bits as valid.

5.2 Divider and Switch

While data recovery replaced a previous synchronization module, the input divider and switch are introduced to provide new functions. First of all, the divider is responsible for enabling operation at data rates less than 160 MBit/s. While for different data rates, in the stable USBpix FPGA configuration, different decoders operating at different clock frequencies were provided, this is not the case with the configuration described here.

If data rates are varying, but the logic is clocked at the same rate, the result of each logic block must either be invariant under reduction of the data rate, which is for example the case for data recovery, or clock cycles must be skipped intentionally when there are clock cycles without new data. The latter is compatible with the output format of the data recovery module, as each bit is accompanied by an additional bit indicating its validity. However, the decoder logic following the block described in this section, does not behave correctly if the data rate is simply reduced. This would result in single bits being inflated to two or four bits. Therefore, for 80 Mbit/s and 40 Mbit/s operation, only every second or fourth bit, respectively, is used and the others are removed from the data stream.

However, instead of ignoring the removed bits one can use these for deinterleaving. For dividing the data rate by n, out of every group of n subsequent bits only one bit is selected, as described previously. If instead of a single data stream with reduced data rate, several interleaved streams are received, by taking a second, third, ..., n-th bit out of this n-bit group, one obtains the contents of the previously interleaved data streams.

For interleaving operation, an additional switch after the divider is inserted to route the data from the divider output of one channel to the read out channel input of another channel. Alternatively, one could have added a set of up to 3 additional logical read out channels per physical channel. This was not done as the limited histogramming memory would not be sufficient. Thus, the read out capabilities of one USBpix unit will be limited to four chips in parallel, even when using multiplexing.

5.3 Decoder

A rather intricate part of the read out chain is the decoder block. Its tasks have not changed in comparison to the decoder from the stable FPGA configuration, while the operating parameters have: First of all, data recovery requires that it has to be able to (at least instantaneously) handle data rates between 0 MBit/s and 320 Mbit/s with nominal data rates at 40 MBit/s, 80 Mbit/s and 160 Mbit/s. Also, from a verification standpoint, it is required that the decoder is able to operate fast enough to fulfil the timing constraints.

As it was the case for the original decoder, first of all the data has to be deserialized (typically using a shift register). This time, between 0 and 2 bits may be pushed into the shift register at each clock cycle. As a result, it is not sufficient to just compare the complete shift register for consecutive idle words to align to the 8B/10B data, but if the amount of bits shifted was two, also the shift register content offset by one has to be taken into account. Alignment has to be kept stable during the absence of idle words, which also has to respect the number of transmitted bits.

Once alignment is established, 8B/10B can be decoded. For this, the latest 8B/10B symbol has to be selected and forwarded to an 8B/10B decoder. Depending on the amount of bits in the latest deserializing steps, the range of the deserialized data selected for decoding has to be offset. The same 8B/10B decoder as in the stable USBpix FPGA configuration will be used. It has been mentioned in section 4.2 that this decoder fails to meet the timing constraints by nearly an additional clock period. This issue is solved by delaying sampling the decoder output and thereby giving the decoder an additional clock cycle.

Frame detection, i.e. the scanning for start-of-frame and end-of-frame comma symbols, can be either performed on the 8B/10B-encoded data or after the decoder, as each comma symbol has a corresponding data byte and the decoder marks these with an additional comma flag. It requires less logic to perform this comparison on the decoded data, as only 9 bits (1 comma flag and 8 data bits) have to be matched to one pattern as compared to 10 8B/10B bits having to be matched to 2 patterns, one for positive and one for negative running disparity. After frame detection, FE-I4 protocol words are built by aggregating three data bytes. Each word is then sent to the subsequent read out processors.

5.4 Read out processors

At last, the signal received from the FEs has been completely decoded to separate data words. These are processed further by histogrammers or written in sequence to the memory. All read out modules operate on the external SRAM, since for histogramming, as well as for the raw data, more memory is required than is available on the FPGA. Previously, as explained in section 4.3, this was done by directly routing all SRAM control, address, and data lines through a multiplexer to the read out modules. As every read out chain has its own read out processor, creating an arbiter for this is rather complicated. Instead, SRAM-specific memory functions are transferred to a separate memory arbiter, which will be described in the next section. The read out processors then send memory commands to the memory arbiter which are then executed there. To further streamline the histogrammers, an *add* operation is provided in addition to a *write* command,

which completely moves incrementing the value of a bin to the memory arbiter.

With this setup, the design of the histogrammers is trivial. The decoded data words are loaded and interpreted. The only relevant word is the *data record*, which contains the row and column address a_{row} and a_{column} and the measured ToT for one or two pixels $t_{\text{ot}\,1}$, $t_{\text{ot}\,2}$. Given this and the read out channel id k, it is easy to calculate a memory address

$$a_1 = k \cdot 2^{19} + t_{\text{ot}\,1} \cdot 2^{15} + a_{\text{row}} \cdot 80 + a_{\text{column}} \qquad \text{or} \tag{5.1}$$

$$a_2 = k \cdot 2^{19} + t_{\text{ot}\,2} \cdot 2^{15} + (a_{\text{row}} + 1) \cdot 80 + a_{\text{column}} \tag{5.2}$$

for a specific bin. The generated memory layout is shown in Figure 5.3. From $a_{\text{column}} \in [1;80]$, $a_{\text{row}} \in [1;336]$ we get for the bit width of the last two terms

$$\log_2 (a_{\text{row}} \cdot 80 + a_{\text{column}}) \le \log_2 (336 \cdot 80 + 80) = \log_2 26960 \approx \boxed{14.7 < 15},$$

$t_{\text{ot}\,1,2} \in [0;15]$ gives for the bits of the inner term

$$\log_2 t_{\text{ot}\,1,2} < 4 = 19 - 15$$

and $k = 0,1,2,3$ gives $\log_2 k < 2$. Thus each address is unique and fits into a memory providing $2^{2+4+15} = 2^{21}$ bytes. The histogrammers then just send a histogramming command containing the calculated address.

For directly writing the received data to the memory in sequence, the memory is segmented into 4 blocks of equal size, one for each of the read out channels. The responsible read out processor iterates over its assigned memory block while writing the incoming data. If the end of a memory block is reached, a flag is set indicating to the host software that no more data can be written.

5.5 Memory Arbiter

The memory arbiter implements the aforementioned operations: writing and histogramming. Histogramming can be split up into an SRAM read, add and SRAM *write* operation. Thus, while there is no return channel to the read out modules, on the memory interface, *read* and *write* operations must be implemented.

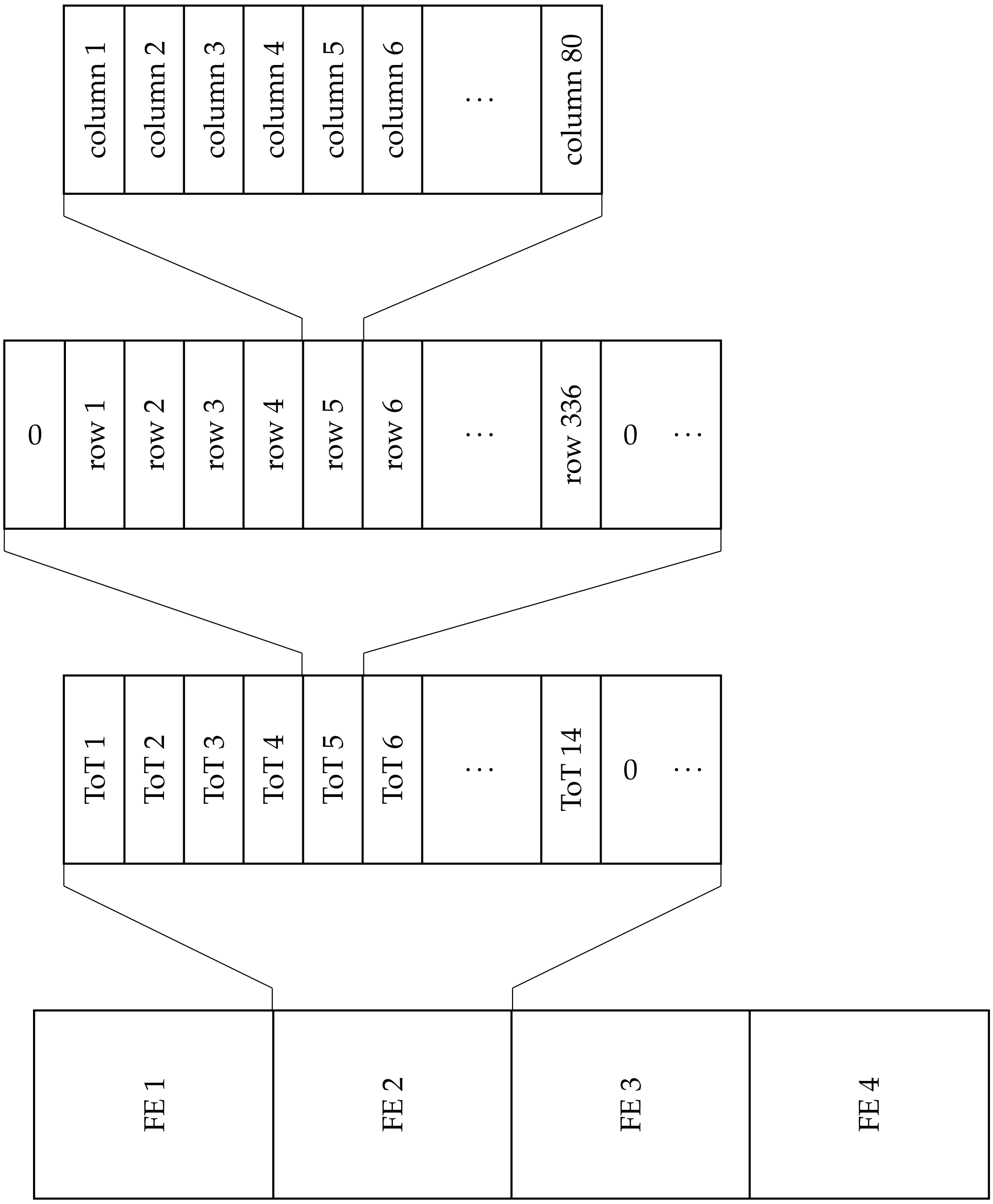


Figure 5.3: Memory layout for ToT histogramming.

The memory arbiter design depends heavily on the required performance. By assuming four connected FEs sending 8B/10B data at the full rate of 160 Mbit/s one can derive an upper limit for the time each operation is allowed to take. First, the record rate is

$$r_{\text{record}} = 4\,\text{channels} \cdot \frac{1}{3}\,\frac{\text{record}}{\text{8B/10B symbol}} \cdot \frac{1}{10}\,\frac{\text{8B/10B symbol}}{\text{bit}} \cdot 160\,\frac{\text{Mbit/s}}{\text{channel}} = \frac{64}{3}\,\frac{\text{record}}{\mu\text{s}}.$$

For histogramming, each data record contains up to two ToT values for different pixels, thus two histogramming operations may have to be performed resulting in the overall available time

$$t_{\text{histo max}} = \frac{1}{r_{\text{record}} \cdot \frac{2}{\text{record}}} \approx 23.4\,\text{ns}. \tag{5.3}$$

Directly writing the received data to the memory yields three *write*s per record, which gives the upper limit on the write time

$$t_{\text{write max}} = \frac{1}{r_{\text{record}} \cdot \frac{3}{\text{record}}} \approx 15.6\,\text{ns}. \tag{5.4}$$

On the other hand, the SRAM datasheet [15] gives lower limits for the *read* and *write* times. Write cycle time t_{WC} and *read cycle* time t_{RC} are both

$$t_{\text{RC}} = t_{\text{WC}} = 10\,\text{ns}.$$

At first glance, these limits seem to be compatible. The first obvious choice for clocking the memory arbiter would be the fastest clock available, which is FCK with a frequency of 160 MHz or a clock period of 6.25 ns. It is then clear, that a *write* as well as a *read* will take two clock cycles, or 12.5 ns. This, however violates the $t_{\text{histo max}} \approx 23.4\,\text{ns} \ngtr 25\,\text{ns}$ limit. Thus, a faster clock has to be generated to achieve these timing values. If we assume that fitting *read* and *write* operations in two clock cycles each, the imposed period limit by the histogramming operation is

$$T_{\text{SRAM\_CLK}} < \frac{t_{\text{histo max}}}{4} \approx 5.86\,\text{ns}. \tag{5.5}$$

This rather abstract timing calculation only shows the general feasibility of performing individual *reads* and *writes* at the required rates. In an actual implementation, addresses to and data from and to the SRAM must pass at least one synchronization stage before they can be processed. These not only delay the presence of data but also hinder fast switching between reading and writing as data is transmitted via a bi-directional data bus on which switching on the tristate-element of one device requires the other device to have switched off to avoid short circuits.

While this is not a problem for consecutive *write* operations, for the histogramming mode it is not possible to alternate single *read* and *write* operations at full rate. Instead, *read* and *write* operations must be grouped. This requires a FIFO at the input of the memory arbiter to buffer incoming histogramming operations while the arbiter is in a write phase. Once a new read phase starts, an address is read from the input FIFO and a read from this

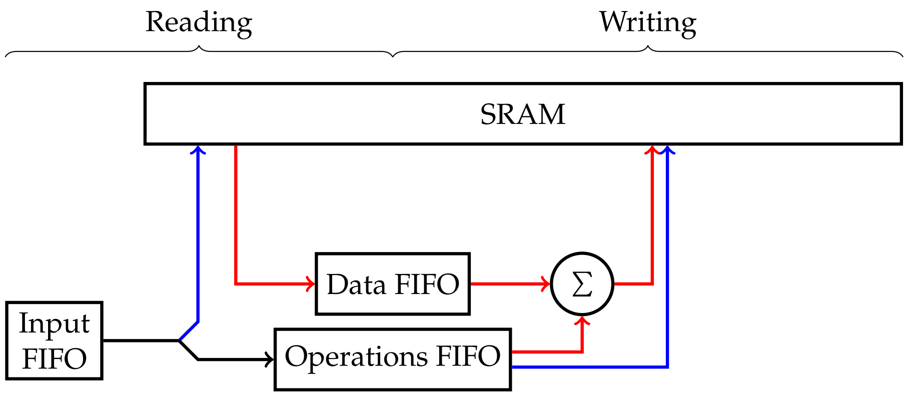


Figure 5.4: Memory arbiter data flow for histogramming operations. Black lines represent full memory operations, blue only their address content, red denotes bin data. The left part of the diagram shows data flow during reading, the right part during writing.

address is requested from the SRAM. The command is stored in another FIFO which contains all recently (pre-)processed commands. Additionally, a countdown is started which signals once valid data is received from the SRAM. This data is stored in yet one more FIFO, containing all recently read memory cells. This last step is independent of requesting new reads, as the former only involves the data lines, the latter only the address lines to the SRAM. Thus, while data reading may take some time, as already explained, new bytes can already be requested. This is also depicted in Figure 5.4.

As soon as the input FIFO is completely emptied, all *read*s are finished and the memory arbiter begins to write the new values to the memory. To do this, a memory command is fetched from the FIFO of recently preprocessed commands. As for each preprocessed command, the according value has been written to the FIFO of read values, this is fetched, too. The new value, the previously read bin content and the additional bin content from the memory command are then written to the SRAM. This is repeated until both FIFOs are empty, after which the arbiter starts a new read cycle.

6 Implementation

The implementations done in this thesis include the FPGA configuration described in the previous chapter. Additionally, a test set-up for interleaved read out is presented.

6.1 USBpix FPGA Configuration

6.1.1 Data Recovery

The data recovery follows Xilinx Application Note 224 [28] with minor changes. As explained in section 5.1, data recovery requires oversampling of the received data. The chosen oversampling rate is 4× the data rate, as this results in stable and simple decision logic. At 160 Mbit/s this results in sampling frequencies of 640 MHz. These frequencies are far outside of the spectrum at which the FPGA at hand can be operated [17]. A solution provided by the application note is to generate four clocks at 0°, 90°, 180° and 270° phase shifts. These clocks are then used to separately take a sample of the data. The samples are then carefully moved to the 0° phase shift clock domain by up to three $\frac{1}{4}$ period phase shifts.

Spartan 3 FPGAs allow to optimize this scheme, as it supports negation of the clock at the input and the 180° phase shifted clock is at 50 % duty cycle close to the negated 0° clock. The same is true for the 90° and 270° clock. The inverter inserts a phase shift of 400 ps [30], which is the major difference between the previously suggested shifted clocks and the clock derived from negation. Thus, only two clocks have to be generated and distributed, which in comparison to static synchronization, is only one additional clock. They are labelled SYNC_CLK and SYNC_CLK_90.

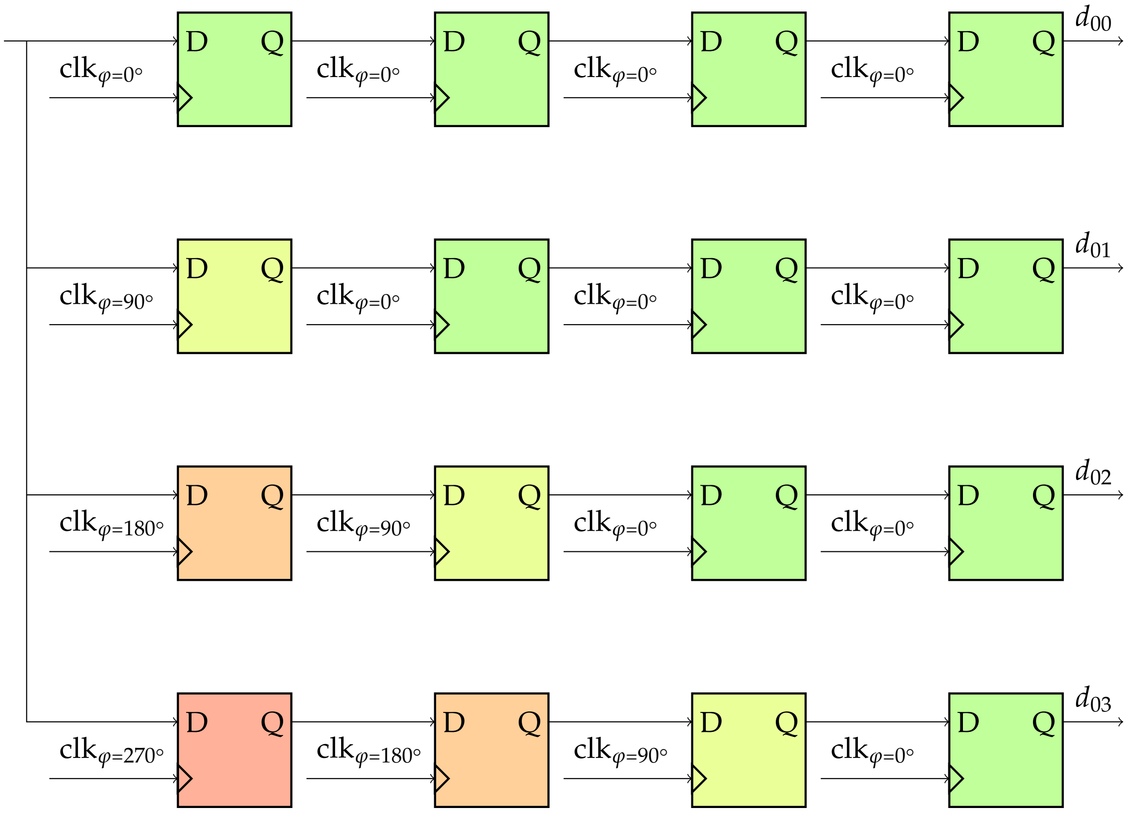
While more modern FPGA generations have circuits for deserializing with schemes like this as independent circuits, this is not the case with the FPGA at hand. Thus, the circuit has to be implemented with the general-purpose logic resources. In placing the first row of flip-flops, special care has to be taken to maximize the distance between the sampling points, so they get as close as possible to a distance of a quarter period $\frac{T_s}{4}$. Especially it should be very improbable that subsequent samples are re-ordered due to clock skew and jitter. Snippet 1 shows the Verilog code responsible for instantiating these flip-flops. The RLOC constraints lock the placement of these flip-flops in reference to a per-instance location constraint. By complementing a global location constraint on these RLOCs, this places the flip-flops in a way that the data routing delay to the flip-flops clocked by the inverted clocks is delayed by a similar value as the clocks are themselves.

The logic to do this is shown in Figure 6.1. At its output, the lines marked d_{0k}, $k = 0, 1, 2, 3$, contain the sampled data with $\frac{1}{4}$ period sampling point distance starting with d_{00}.

Feeding the d_{0k} through additional flip-flops also clocked with SYNC_CLK gives delayed samples d_{1k}. By looking for changes between d_{0k} and d_{1k} one can find out if an edge

```
(* RLOC = ”X−0Y−2”, IOB = ”FALSE” *)
  FDC ff_a0 (.D(D), .C(SYNC_CLK),      .CLR(RST), .Q(az[0]));
(* RLOC = ”X0Y−0”,  IOB = ”FALSE” *)
  FDC ff_b0 (.D(D), .C(SYN_CLK_90),    .CLR(RST), .Q(bz[0]));
(* RLOC = ”X−0Y−3”, IOB = ”FALSE” *)
  FDC ff_c0 (.D(D), .C(~SYNC_CLK),     .CLR(RST), .Q(cz[0]));
(* RLOC = ”X0Y−1”,  IOB = ”FALSE” *)
  FDC ff_d0 (.D(D), .C(~SYNC_CLK_90), .CLR(RST), .Q(dz[0]));
```

Snippet 1: Flip-flop instantiation for data recovery oversampling.



D
Q
$\mathrm{clk}_{\varphi=0°}$
$\mathrm{clk}_{\varphi=90°}$
$\mathrm{clk}_{\varphi=180°}$
$\mathrm{clk}_{\varphi=270°}$
d_{00}
d_{01}
d_{02}
d_{03}

Figure 6.1: Oversampling logic used by data recovery. The color changes for increasing clock phase from green to red.

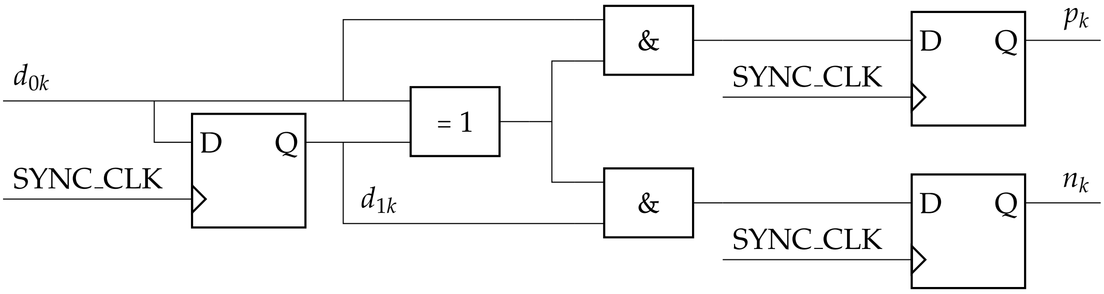


Figure 6.2: Data recovery edge detection. The XOR ensures that the two samples differ. Together with the dually connected ANDs this also decides whether the event is a negative edge n_k or a positive edge p_k.

has occured in the period offset by $\frac{k}{4}$ of a clock period. The corresponding logic circuit can be found in Figure 6.2. The events are stored by setting p_k for a positive or n_k for a negative edge to 1. If $p_0 = p_1 = p_2 = p_3 = 1$ or $n_0 = n_1 = n_2 = n_3 = 1$, the edge occured between the sample by the 270° clock and the 0° clock. Thus valid data would be available on the 90° or 180° sample, d_{02} is chosen. Observing $p_0 = p_1 = p_2 = 1, p_3 = 0$ or $n_0 = n_1 = n_2 = 1, n_3 = 0$, the edge occured between the 180° sample and the 270° sample. Thus valid data can be sampled from the 0° and 90° sample; d_{01} is chosen. Continuing gives the sampling decisions for d_{00} and d_{03}. The logic circuit for this decision can be found in Figure 6.3 and the decision results will be labeled u_{0k} for choosing d_{0k}. Additionally, e_u is one if and only if an edge was detected at all.

While the samples chosen above can be directly used as one of the outputs of the data recovery, three special cases have to be handled. The first is that $n_k = p_k = 0 \forall k$. In this case, no edge has been detected at all, the same sampling point chosen in the previous clock cycle is used again. Of course, this assumption requires the data transmission to contain a minimum number of edges, when the clock and data phases are not statically aligned. Figure 6.4 shows how the bit d_0 is thus recovered. First, u_{0k} is routed to flip-flops that save the last edge detected into u_{1k}. As only one edge position can be detected at a time, a one-hot-4-to-1-multiplexer gives the required value for d_0.

The other two cases are related to moving edges. Whenever an edge moves backward with respect to the sampling phase, i.e. the data is transmitted faster than the sampling clock takes samples, and a complete period of phase difference has been accumulated, two bits have to be sampled. This case has already been explained before and is implemented by providing an additional data output d_1 with an associated validity bit v_1. The third case is an edge moving forward with respect to the sampling phase. Here, less data is transmitted than sampled and thus, sometimes a valid bit is available to be written to the output. To handle that, the first output bit also has an associated validity bit v_0 that becomes 0, once this event occurs. Figure 6.4 shows the symmetry in v_0 and v_1.

6.1.2 Clock Generation

The additional clock required for data recovery can easily be derived by adapting the existing clock generation module. In addition to the clock with phase 0°, each DCM also provides the clock with different phase shifts. One of these is the same clock with 90° phase

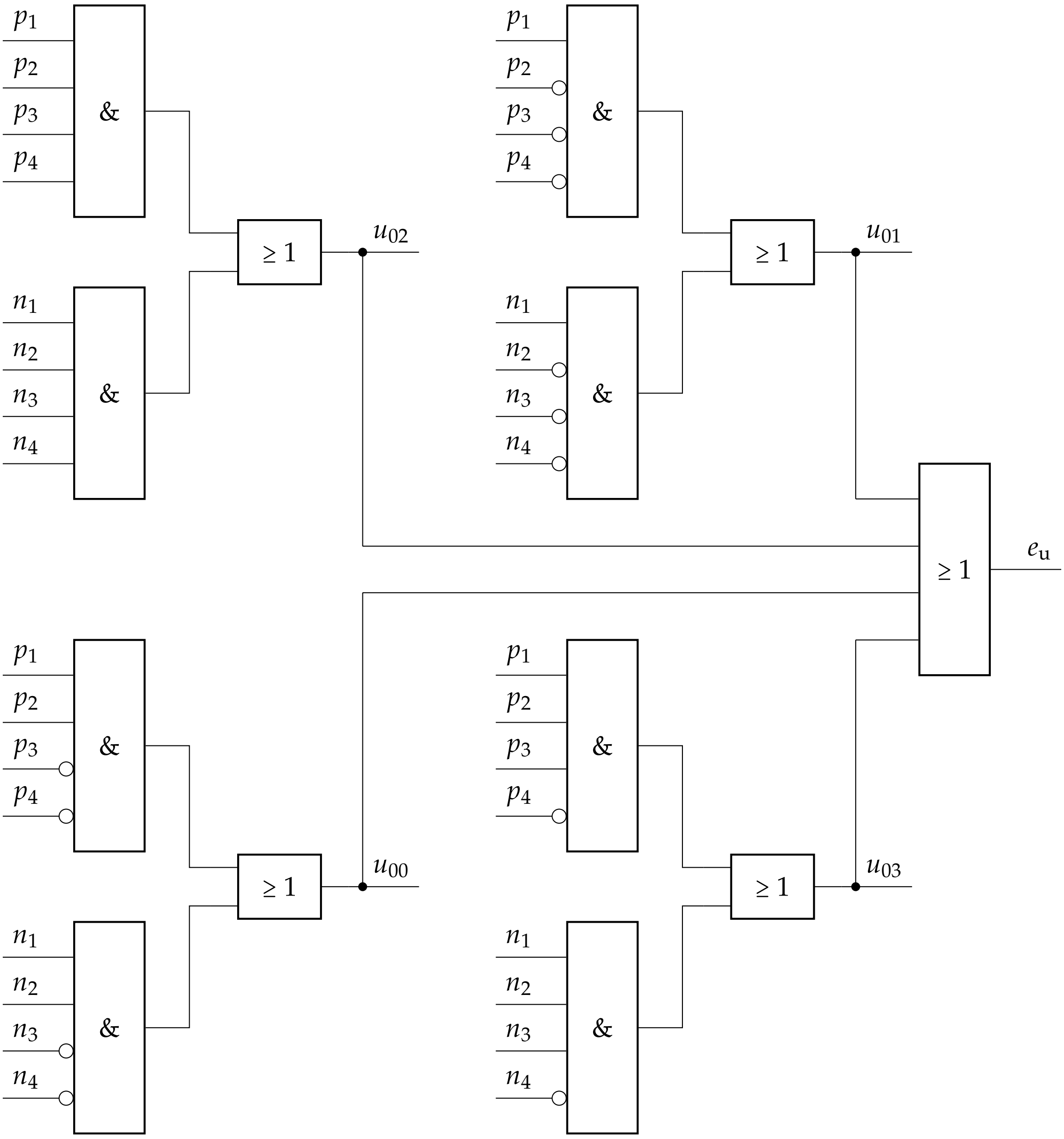


Figure 6.3: Sample decision logic

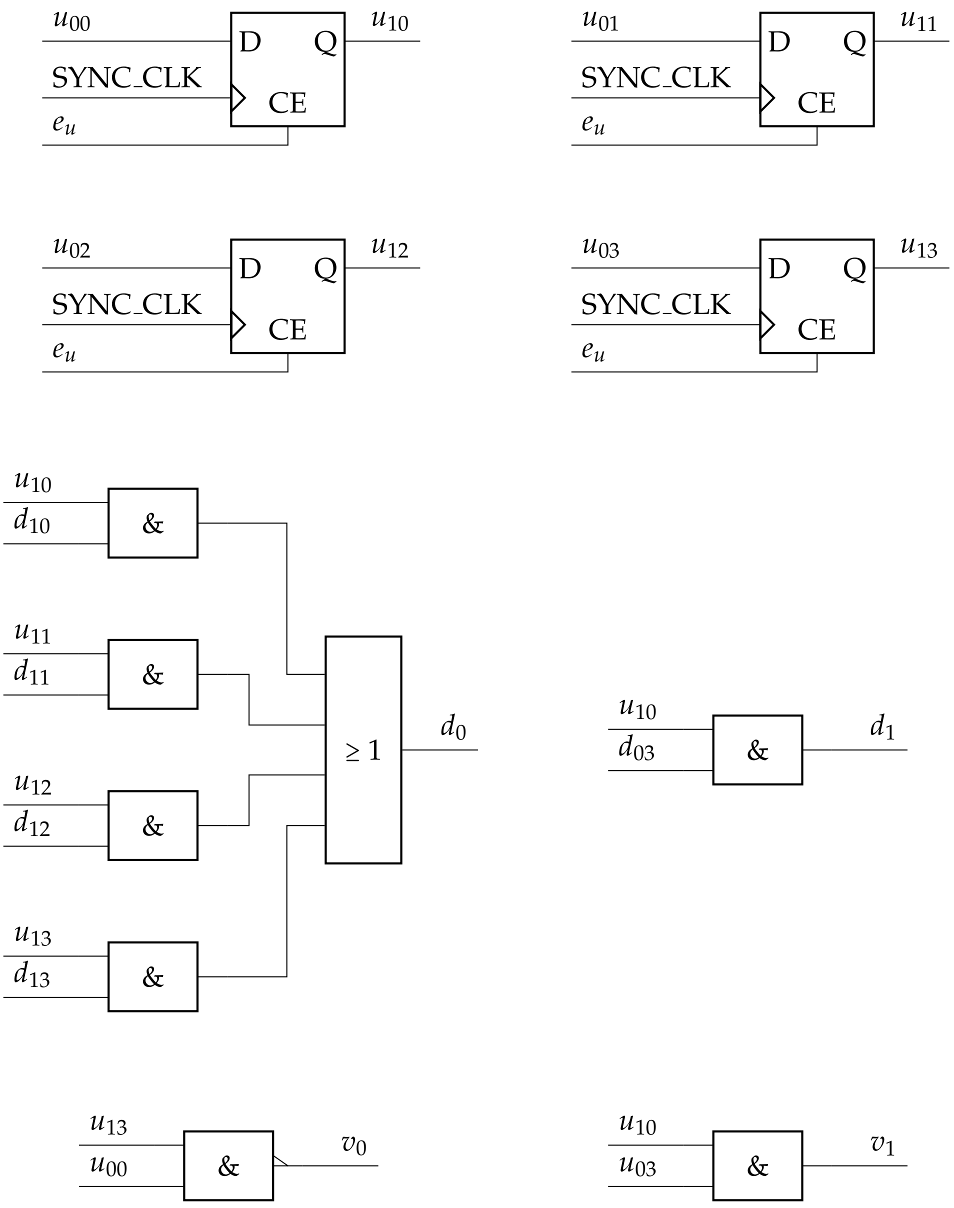


Figure 6.4: First output bit of data recovery.

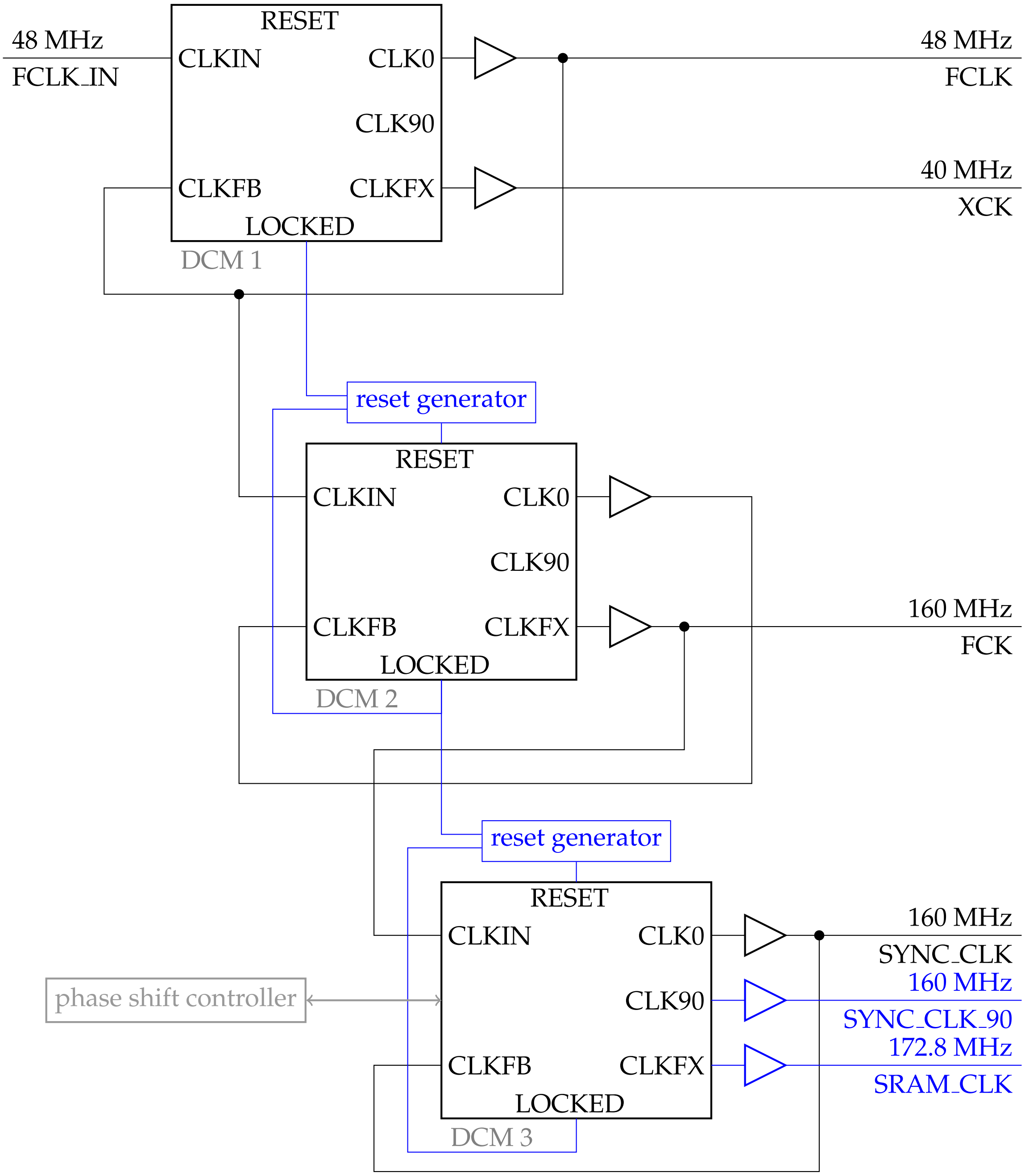


Figure 6.5: Updated clock generator block. Additions in comparison to Figure 4.3 are shown in blue, inactive parts in grey. Reset generators have been added, two additional clocks are generated, and the phase shift controller is not used anymore.

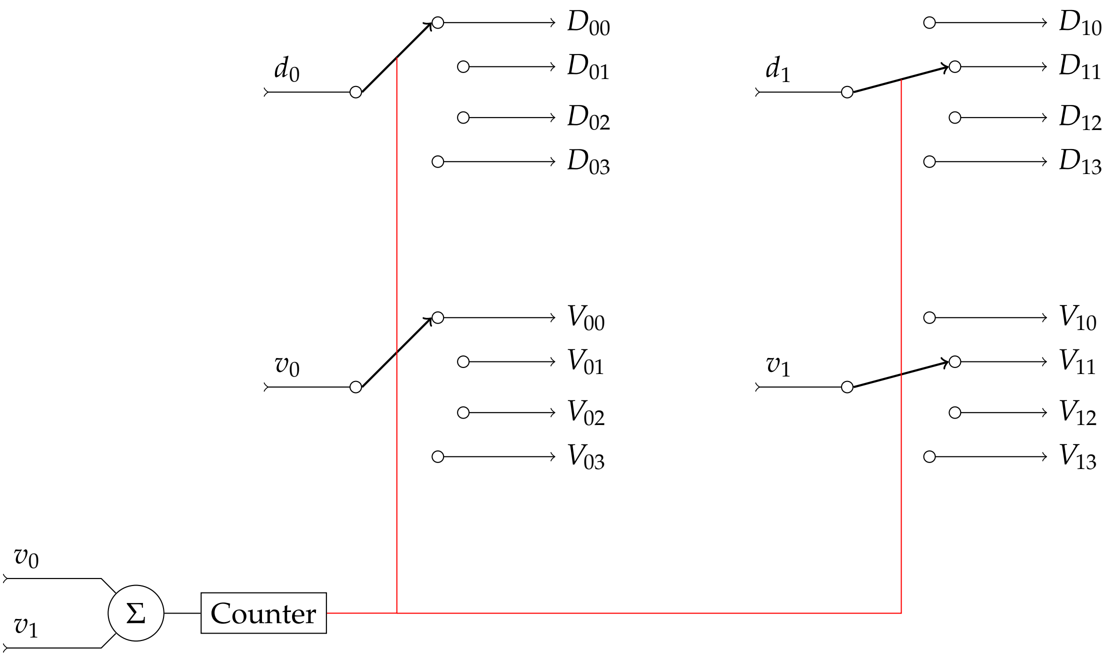


Figure 6.6: Divider model: For dividing by g, d_k, v_k are routed to D_{kl}, V_{kl} depending of the counter value r with $l = r$ for $k = 0$ or $l = r + 1 \mod g$ for $k = 1$. The counter itself counts the number of available bits in $\mathbb{Z}/g\mathbb{Z}$.

shift. Therefore, generating SYNC_CLK_90 requires only the instantiation of an additional global clock buffer (see Figure 6.5). Following up on the jitter discussion it becomes clear however, that the chaining scheme used in the stable USBpix configuration does not yield minimum achievable jitter, as the DCM reset is disabled before a stable clock is available at the two DCMs which follow later in the chain and thus the locking algorithm can not find the minimum jitter state [30]. Therefore, additional reset generators are added to these two DCMs, generating a sufficiently long reset pulse if either the DCM itself or its predecessor lost its lock.

The second change on this module is to generate a new clock SRAM_CLK with a period $T_{\text{SRAM\_CLK}} < 5.86$ ns (see Equation 5.5). Generating this clock requires a CLKFX output, which is available on the DCM generating SYNC_CLK and SYNC_CLK_90. Using a multiplier/divider setting of $\frac{27}{25}$ gives

$$f_{\text{SRAM\_CLK}} = \frac{27}{25} \cdot f_{\text{FCK}} = \frac{27}{25} \cdot 160 \text{ MHz} = 172.8 \text{ MHz}$$

or a period of $T_{\text{SRAM\_CLK}} = 5.79 \text{ ns} < 5.86 \text{ ns}$.

6.1.3 Input Divider and Switch

A schematic of the divider is shown in Figure 6.6 with demultiplexers depicted as switches to improve comprehensibility. A counter keeps track of the number of transmitted bits per clock cycle and steers the multiplexers. For a division by g, a counter with value r counts from 0 to $g - 1$ and wraps around. d_0 from data recovery is demultiplexed to D_{0r}

depending on the value of r. To keep validity information, v_0 is also demultiplexed to a V_{0r} associated to D_{0r}. Looking only at V_{00} and D_{00} and ignoring cases where $v_1 = 1$ makes it already clear that V_{00} will assume a value of 1 every g clock cycles. As the mechanisms for V_{00} and D_{00} are the same, D_{00} will contain valid data at exactly that clock cycle. Also, V_{01}, V_{02} and V_{03} contain (depending on the divisor) the samples from the other time slots which can be used for deinterleaving.

For $v_1 = 1$, the bit succeeding the one in d_0 is already in d_1 instead of the d_0 of the next clock cycle, as it would be the case for $v_1 = 0$. Thus, a set of additional bins D_{1s}, V_{1s}, $s = 1, \cdots, 4$ are used to demultiplex d_1 and v_1. The multiplexing process is the same as previously, but as the bit does not belong to the current counter value, for a counter value of r, the s^{th} bin with $s = r + 1 \mod g$ is used.

D_{1s} and D_{0r} then have to be merged, which has to be done depending on the divider value. For $g = 1$, so no division, D_{00} and D_{10} are not merged at all and directly forwarded. This is a special case, as only for $g = 1$ the data rate may exceed the clock frequency. For $g > 1$, D_{0r} and D_{1s} are merged by ORing these for $s = r$, because the D_{1s} only contain data when D_{0r} for $r = s - 1 \mod g$ also contained data and thus it is sufficient to transmit at most one bit per clock cycle. Proceeding equally with V completes the divider logic.

The switch is the only intermediate read out chain element that spans all read out chains to allow moving data from one divider to the decoder of another read out chain. For operation without division or demultiplexing, only forwarding to the decoder of the same read out chain is supported. The D_{00} and D_{10} are then used as the inputs of the decoder continuing the pattern from the previous paragraph. In case of $g > 1$, still $D_{00} \vee D_{10}$ has to be the input of the first decoder, but other decoders may be assigned $D_{01} \vee D_{11}$ and higher time slots as input if the data is supposed to be deinterleaved. Again, the scheme is completed by proceeding equally with V.

6.1.4 Decoder

At the input of the decoder block, the data streams have been successfully separated and 8B/10B encoded FE-I4 protocol data for exactly one FE-I4 chip can be expected. This data is provided using the four outputs from the switch block, two data signals and two associated validity indicators. The data is inserted into a shift register, always shifting the indicated number of bits. The shift register has its register values exposed as a parallel output, which is used for pattern matching for 8B/10B symbol alignment and parts of it are also forwarded to the decoder. As the uniquely identifiable idle pattern consists of two consecutive comma symbols, 20 consecutive bits have to be available at the shift register output. This number is increased to 21 to also cover the case where two bits are shifted into the shift register at once. Let the output of the shift register be $d_{\text{SR}}[20:0]$.

Four cases have to be taken into account for idle pattern detection. An idle pattern might occur starting at $RD = +1$ or $RD = -1$. Independently, it might be detected in $d_{\text{SR}}[19:0]$ or $d_{\text{SR}}[20:1]$. Thus, four match units with 20 bits width each are required. Given the 4-input LUT architecture, it becomes clear that this would require 5 LUTs, each processing 4 bits, for matching individual bits and then an addition of two consecutive LUTs for reducing the match result to one bit. Placing and routing is complicated due to the high degree of connectivity required between the shift registers and the matching LUTs. Providing simple match units at the high operating frequencies is therefore unfeasible and the idle pattern matcher has to be pipelined.

For this, the initial problem is first divided into detecting single 8B/10B idle commas. This increases the number of cases of this first stage from 4 to 8. Matching only 10 instead of 20 bit reduces the problem to three LUTs, two of them matching the first 8 bit, the third reducing their result together with the remaining two bit to a single stage output bit. Having reduced the data at hand this far, it is feasible to generate a signal, indicating the detection of an alignment pattern.

While this construct continually checks for new alignment data, a counter counts the number of valid bits down from 9 to 0 and wraps around. This counter thus keeps track of alignment, while no alignment patterns are detected. The counter is monitored for two cases: Either one bit is shifted into the shift register and the counter reaches the value 0. In this case, the value $d_{\mathrm{SR}}[9:0]$ contains a valid 8B/10B symbol. The other case is that the counter just wrapped around and thus has the value 9 and two bits have been shifted into the shift register at once. In that case, $d_{\mathrm{SR}}[10:1]$ contains a valid 8B/10B symbol. A flag indicating which is the case is set. In the next clock cycle, the appropriate range of d is selected and written to D_{sel}. For this, also the number of just received bits has to be taken into account and the range shifted further, depending on that value, to up to $d_{\mathrm{SR}}[12:3]$. Additionally a flag e_s indicating valid data is set to 1 for one clock cycle.

The *alignment detected* signal is used to reset the counter to its correct value, every time one is detected. However, from the time the last bit of the alignment pattern was received by the decoder to the valid alignment signal, four[1] clock cycles have passed. Therefore, the counter is not reset to 0, but 9 minus the number of valid bits that have been received within these four clock cycles.

Generally, repeated alignments should not change the counting behaviour of the counter, as the counter itself keeps track of correct 8B/10B symbol alignment. Only once, while a new chip is connected, the alignment is supposed to set the counter to its correct value. If another re-alignment is detected, this typically indicates an error and is thus signalled to the PC software.

Once the data D_{sel} has been selected, it is forwarded to the 8B/10B decoder. In parallel, the decoder's disparity output is written to its disparity input and e_{s} is delayed by two clock cycles to e_{s}''. By using e_{s}'' to indicate valid decoder output, the decoder is allowed to take two clock cycles to complete its operation. This is shown in Figure 6.7. The 8B/10B decoder error outputs set flags that can be subsequently read by the host software. e_{s}''' indicates validity of D_{dec}.

D_{dec} is 9 bit wide, 8 of which are data, one is signalling if the decoded symbol was a comma data symbol. As D_{dec} uniquely identifies comma symbols, it is now less complicated to trigger for start-of-frame and end-of-frame symbols. The data between these symbols is grouped into records of three bytes.

[1] One for shifting into the shift register, one for the pattern detection stage, one for the combining stage, one for generating the alignment signal.

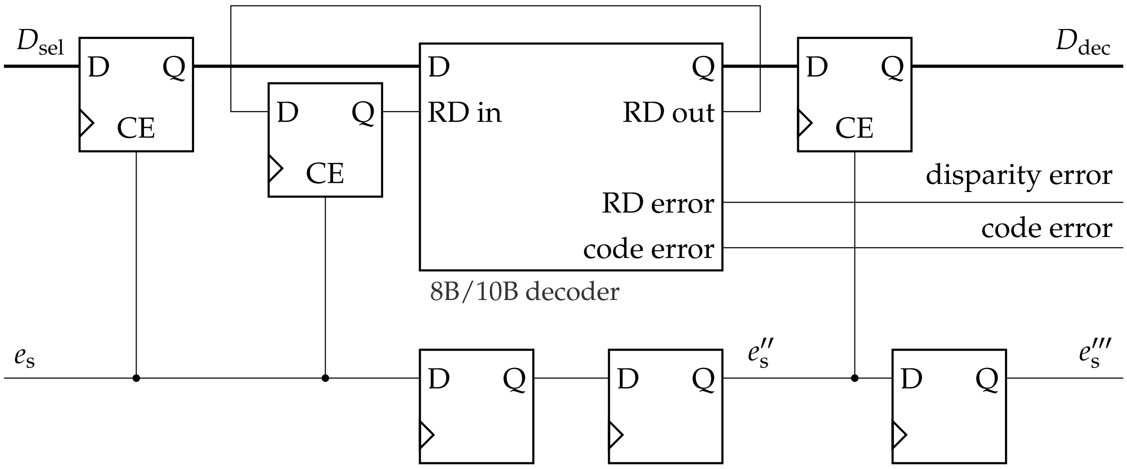


Figure 6.7: 8B/10B decoder periphery. All registers are clocked by FCK and enabled with by a e_{s} or e_{s}''. The top shows the data flow with the associated disparity feedback, the flip-flops on the bottom show how e_{s} is delayed.

6.1.5 Read out modules

With readily grouped FE-I4 protocol records received from the decoder and most memory functions moved to the memory arbiter, the implementation of the read out modules does not require much effort. The interface to the memory arbiter is given through memory commands. As only two types of commands exist, one bit in the command interface is sufficient to distinguish between these. An additional bit on the interface signals that the current command is valid and to be processed. As both commands refer to a single memory address, 21 bit of address information are sufficient. Apart from that, the command also contains 8 bit of payload. For a write command, this payload contains the value that is supposed to be written to the memory. For a histogram command, the payload contains the value which should be added to it.

The hit processor is just the ToT processor with all ToT bins projected to the first one, thus it is sufficient to describe the latter here. The ToT histogrammer contains a finite state machine (FSM) with two states. The first state s_0 is the idle state. Whenever a record is received from the decoder which is a data record, a memory command with the address from Equation 5.1 using the first ToT value, a value of 1 and the histogramming flags set is sent to the memory arbiter. The state is changed to the second state s_1. As the paired pixel might not have a hit at all, the ToT field is checked to be less than 15 (which corresponds to a measured ToT of more than 0) and if this is the case another memory command is issued using the address from Equation 5.2 and the second ToT field value.

Implementing the raw data writer requires a little more effort. This is rooted in the requirement, that in addition to the FE data, trigger data has to be merged into the data stream. As the FE-I4 data is associated with a valid bit, the same is the case with the trigger data. Both are expected to come at a very low rate, so merging is simply implemented by providing an additional register for both which is written to every time new data on the respective channel arrives. Each register is accompanied by a flag that is set once the register is written and reset once it is read (see Figure 6.8). Analog to the ToT histogram-

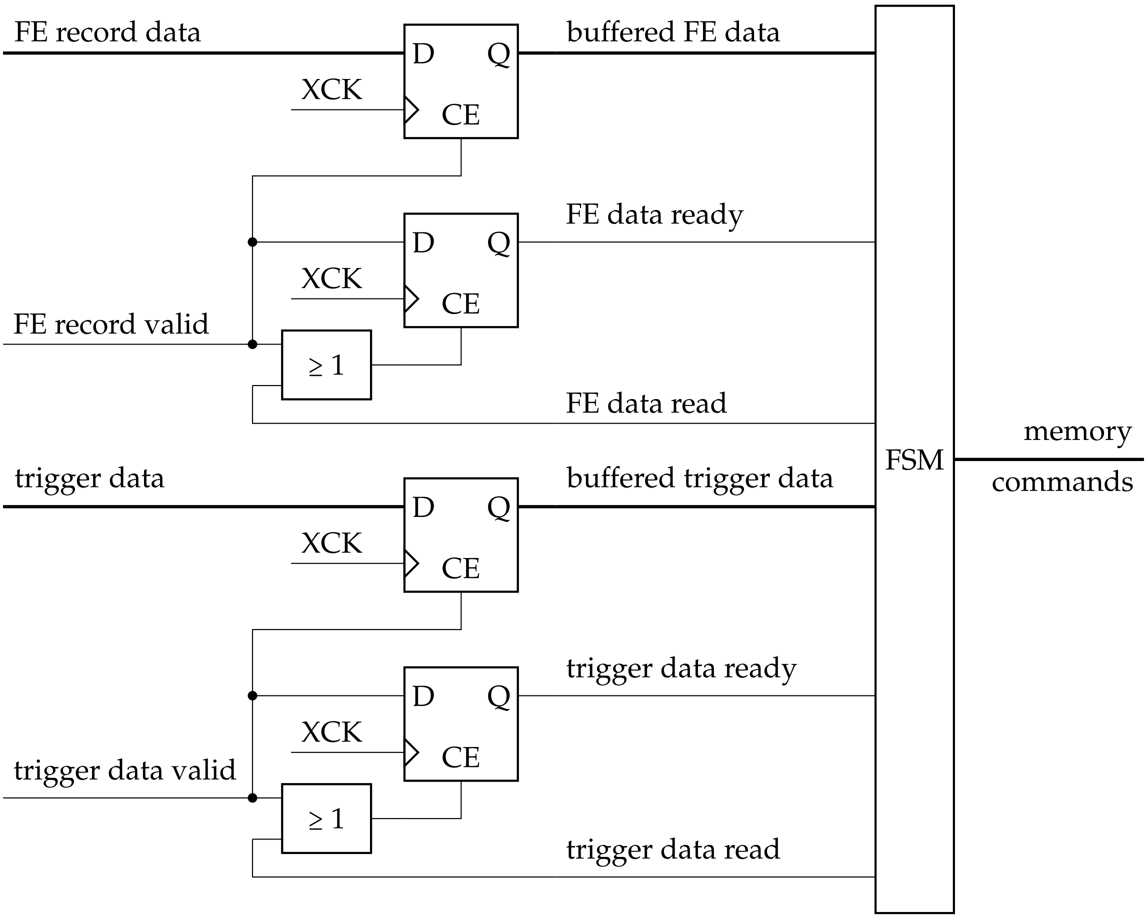


Figure 6.8: Raw data writer schematic: The FSM transforming data into memory commands is supported by two registers with flags indicating updated contents. Even though only one flip-flop is displayed, this is actually replicated for each bit in the FE record and trigger data.

mer, a three-state FSM sends three memory commands containing the three bytes of the record, once one of the register flag is set to 1. The memory addresses are simply incremented starting at 0, adding $2^{19} \cdot k$ for channel k. Once the memory address reaches the lower bound of the next channel or the maximum memory address, a flag indicating that one of the read out modules completely filled its reserved memory is set, which is then interpreted by the host software.

6.1.6 Memory Arbiter

Through the careful design of the memory arbiter commands, the arbiting stage is reduced to merging the four command streams. This is done by instantiating four FIFOs, one for each channel. By using independent clocks for the read and write port, in addition to buffering, the data is moved from the XCK to the SRAM_CLK clock domain. The SRAM_CLK is the clock motivated in section 5.5, which is used to meet the SRAM timing at the required data rates. A circular shift register of size 4, which is initialized with the pattern 0001_2 is used to select one FIFO. The read input of this fifo is set to 1, thereby telling the FIFO to output one memory arbiter command.

While a read may not succeed when a selected FIFO is empty, successful reads are marked by the FIFO by setting its *valid* output to 1 for one clock cycle. This bit, which can only be one for one output at a time, since the delay between a read request and valid data is fixed, is then used to merge the FIFO outputs. As using a single-stage merge logic turned out to strain the timing constraints at the high SRAM_CLK clock frequency notably, it has been pipelined such that two stages first perform a pairwise merge and then a final merge in another clock cycle. The output of this merge logic is written to the input FIFO of the memory arbiter.

Having only one stream of memory commands to handle the actual memory handling has to be implemented. The most delicate task here is to provide stable memory access for histogramming. In this case, as per section 5.5, read and write operations are supposed to be grouped. For this, first of all the three buffer FIFOs are instantiated:

- The *input FIFO* is responsible for buffering incoming memory commands during write operations. Its width is the same as the width of a memory command as it needs to store them completely.
- The (preprocessed) *operations FIFO* contains all memory commands for which a read request was issued to the SRAM. In addition, for write instead of histogramming memory commands, commands are simply written to this FIFO without issuing a read. Obviously, its width is the same as the input FIFO.
- The (read) *data FIFO* contains the data that has been read back from the SRAM. Its width is one byte, the data width of the SRAM. Additional addressing information is not required, as the FIFO is ordered in the same way the operations FIFO is ordered.

All are first word fall-through FIFOs. Such FIFOs exhibit the behaviour that a *valid* flag signals the validity of the data on the FIFO's data output. This is the case as soon as valid data is available and it is not necessary, as with traditional FIFOs, to explicitly perform a read operation. Instead, the read port is used to signal that the data at the FIFO output has been evaluated by the successive logic and it may be replaced by the next element waiting in the FIFO.

Reading from and writing to these FIFOs as well as the SRAM I/O is controlled by a FSM which is connected to the FIFOs as shown in Figure 6.9. All FSM outputs are additionally registered within the FSM, even those with external flip-flops. On the left hand side of the figure, the FIFOs and their connections are displayed. The input FIFO is written externally from the merge logic and only exposes its read interface with the corresponding data output Q, validity and empty indicators, and the read input which is used to request reads. The same is the case for the operations FIFO, however, it is directly written by the FSM. The data FIFO is obviously connected to the data output from the SRAM and a shift register delaying any write signal from the FSM by 5 clock cycles.

The SRAM connections on the right hand side start on the top with the SRAM address written by the FSM via a SRAM_A. As the SRAM does contain 2^{21} bytes, these are addressed in 16 bit groups. Thus, only 20 bits of SRAM_A are actually routed to the SRAM's address input. The remaining bit controls a bit selecting which of the two bytes in the aforementioned group is asked for. This is done by two lines called BLE (byte low enable) and BHE (byte high enable). Directly below the address lines, the logic for sending write enable (WE) signals is shown. As a finer granularity is required here, the DDR functionality of the FPGA is used to generate WE pulses with a granularity of $T_{\mathrm{SRAM\_CLK}}/2$. The SRAM data uses a bidirectional bus. Thus SRAM_D and the data return to the data FIFO connect via registers and a tristate buffer to the same pin. The SRAM_IOSEL signal is used to disable the buffer when data is supposed to be received by the FPGA and enables it, when data is supposed to be written. SRAM_OE (output enable) is used to transmit that information to the SRAM, which uses a similar construct to drive its data lines.

Figure 6.10 shows its state transition diagram. Two of the states (left) are responsible for reading, another two (right) for writing. Two additional states (top and bottom) manage transitioning between these two groups.

FSM operation

For a detailed description of its operation, assume a write group was just finished and the FSM entered the state R_1. This means that both the data FIFO and the operations FIFO are empty. The FSM turns to the Input FIFO: If its valid output is zero, the FIFO is empty and the FSM remains in R_1 to await incoming memory commands. If the valid bit is 1, the next memory command is available at the data output (Q) of the FIFO. Without regard of the type of command (write or add), SRAM_A is set to the address referenced in the command and the full command is written to the operations FIFO. In addition, SRAM_IOSEL is set to 0 to disable the FPGA output buffer on the SRAM data lines. Setting SRAM_OE to 1 enables the SRAM data output and a value of 1 instead of the default value 0 is shifted into the shift register connected to the WRITE port of the data FIFO. This will issue a write to the data FIFO five clock cycles later, when the bit has been completely shifted through. The address requires about two clock cycles until it has reached the SRAM, the SRAM_D outputs are valid at most 10 ns or two clock cycles later, and the data has to pass another flip-flop. This corresponds to the maximum time required for the read data to be available at the FIFO. As the data may be available one cycle before and the SRAM read cycle time is t_{RC} = 10 ns, which spans almost two clock cycles, the FSM transitions into the pause state R_2. This is the sole purpose of R_2 and the state becomes R_1 in the next clock cycle.

Back in R_1 this is repeated until the input FIFO is empty. If this is the case, the FSM transitions to the $R \rightarrow W$ state. The purpose of the intermediate state before performing actual writes is to make sure that all reads performed above are finalized. This is not the case while the shift register connected to the data FIFO WRITE port is not completely filled with zeroes. Therefore the FSM resides in $R \rightarrow W$ until this is not the case any more and the new state becomes W_1.

While the FSM is in state W_1, two tasks have to be performed. One is to execute the SRAM writes as they are read from the commands FIFO and data FIFO. For this, SRAM_IOSEL is set to 1 and SRAM_OE is set to 0, which inverts the direction of the data lines. Next, the command available at the output of the operations FIFO is checked for its type. If it is a histogramming command, the sum of the value available from the data FIFO output and the value contained in the command are added and written to SRAM_D. Otherwise,

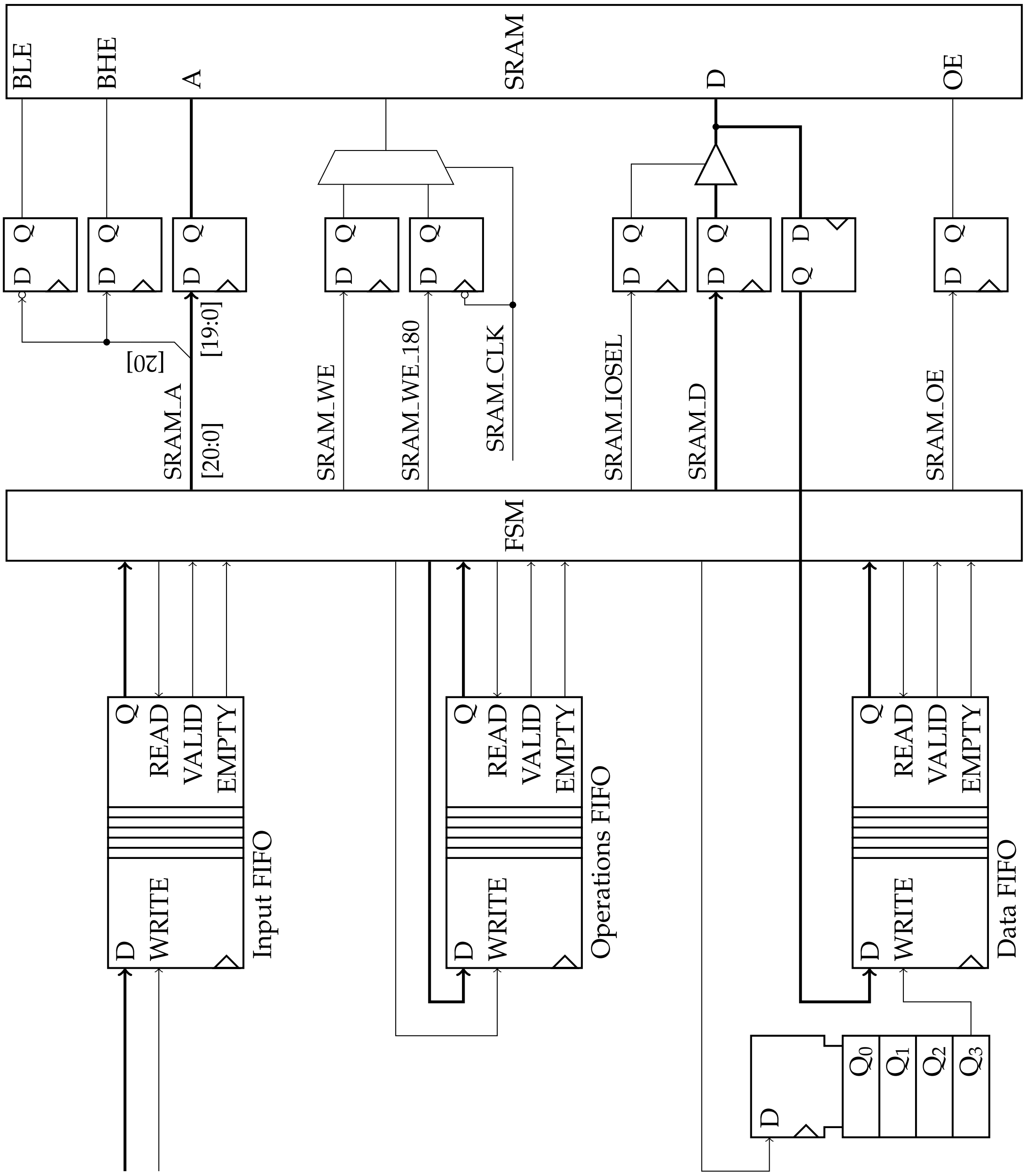


Figure 6.9: Memory arbiter FSM connection diagram. All FSM outputs are registered (additionally) within the FSM. All registers are clocked with SRAM_CLK except where explicitly shown.

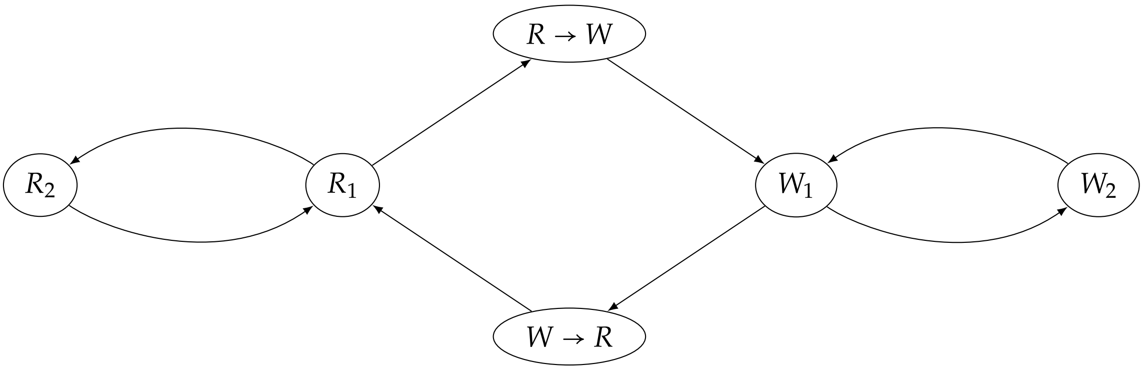


Figure 6.10: Memory arbiter FSM

only the value contained in the command is used as the new value of SRAM_D. In addition, a write pulse is initiated by setting SRAM_WE and SRAM_WE_180 to 1. The FSM transitions into W_2, where the write pulse is continued, but only for half a clock cycle by setting SRAM_WE to 0 and SRAM_WE_180 to 1 and the FSM returns to W_1.

While still in W_1, the FSM also checks the current output of the input FIFO. If there is a valid command and the command type is not add but write, the command is written to the operations FIFO and the next one is requested. In addition, a dummy value is written to the data FIFO to keep it synchronized. This allows for faster processing of write commands without switching back to $\{R_1, R_2\}$.

A write group is finished once one of the two intermediate FIFOs does not have valid data at its output. The FSM state becomes $W \to R$ which is used to make sure that the SRAM has completely finished the write cycle before the direction of the data lines is reversed again in the next state R_1.

FIFO depth constraints

Special care has to be taken while choosing the depths of the FIFOs. The time required to read a group of v bytes is the same as the time to write v bytes. This dictates that all three FIFOs have the same depths. The depth itself is given by the maximum size of each read/write group, as it has to be possible to buffer one of these completely. Again it is sufficient to focus on the add command timing.

The time required for completely histogramming a group of size v is

$$\begin{aligned} t_{\text{histo group}}(v) &= (\underbrace{2 \cdot v}_{\text{reading}} + \overbrace{2 \cdot v}^{\text{writing}} + \underbrace{5}_{R \to W} + \overbrace{1}^{W \to R}) \cdot T_{\text{SRAM\_CLK}} \\ &= (4v + 6) \cdot T_{\text{SRAM\_CLK}}. \end{aligned}$$

Whereas the maximum time allowed for processing this group is given by the incoming data rate (see Equation 5.3)

$$t_{\text{histo group max}}(v) = v \cdot t_{\text{histo max}} \approx v \cdot 23.4\ \text{ns} \approx 4.05 \cdot v \cdot T_{\text{SRAM\_CLK}}.$$

with $T_{\text{SRAM\_CLK}} \approx 5.79$ ns.

The minimum v is then obtained from

$$\begin{aligned} t_{\text{histo group}}\left(\underline{v}\right) &= t_{\text{histo group max}}\left(\underline{v}\right) \\ \rightarrow \left(4\underline{v}+6\right) \cdot T_{\text{SRAM\_CLK}} &\approx 4.05 \cdot \underline{v} \cdot T_{\text{SRAM\_CLK}} \\ \rightarrow \underline{v} &\approx 120. \end{aligned}$$

This is comfortably close to a power of 2, namely 128, which is chosen for the maximum group size and thus the FIFO depths.

6.1.7 Other Improvements

Other parts of the FPGA configuration were also updated. Previously, clearing the SRAM could only be done by writing zeroes to it using the USB SRAM bulk interface. This has been supplemented by an additional pseudo read out module, which resets the complete SRAM to zero.

The Manchester encoder for the command line was unified with the clock output to a single encoder, able to treat both outputs in parallel. This was done to prepare for plans to unify the clock and command links to a single link. Also, the encoding process was simplified and restructured.

The FPGA output drivers for the clock and command link were incorrectly configured for load of the four buffers on the burn-in card plus the termination resistor, which resulted in clock duty cycles between 55% and more than 60% at the output of the burn-in card. As there are plans to do interleaving using the clock to select between two chip outputs on the module, this is not acceptable. Increasing the *drive strength* (maximum current out of and into the pin) and the *slew rate* (inclination of the signal at edges) of these outputs will result in duty cycles closer to 50%.

6.2 Interleaving Test Devices

Before this thesis, no test setup for interleaving FE-I4 data streams existed. A test device was created which allows to connect two FE-I4 chips, multiplexes the data received from both using the presented interleaving scheme and sends the result to a read out system. To have maximal flexibility during testing and the possibility to extend its features, an FPGA-based solution was chosen.

The major factors that influenced the FPGA device selection were the time it would take to create the hard- and software around it, as it was supposed to be quickly available for testing, and the size constraints on the final circuit, as placing a similar but smaller device directly on a four-chip module was considered to be a possibility during an upcoming prototyping phase. Historically, FPGAs have grown to be rather complicated devices which need a variety of external components, for example for providing a multitude of several different operating voltages and configuration storage, as most FPGAs themselves store the configuration within volatile SRAM cells, which have to be updated every time the FPGA is reset.

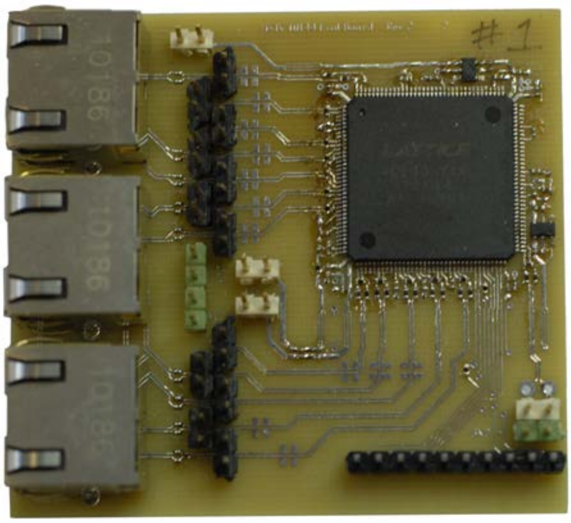


Figure 6.11: Photograph of the HX1K PCB.

Recently, some FPGAs have been released which do not suffer from these limitations. Among these are for example the *igloo nano* family, which sadly does not offer LVDS I/O [3], or the *iCE40* family [23]. While both are quite similar, due to the LVDS support, an iCE40 FPGA was chosen. They offer very small devices[2] down to $2.5 \times 2.5\,\mathrm{mm}^2$ at low power (21 µW standby) and only two voltages are required, 1.2 V for the core logic and 2.5 V for I/O. A phase-locked loop (PLL) offers clock conditioning inside the FPGA, DDR registers reduce the required clock frequencies. A one time programmable non-volatile memory on the FPGA could allow for permanent configuration storage, without the need for external memory.

Two similar devices were selected, a low performance LP1K and high performance HX1K. Both are very similar in features, but while the HX1K is characterized to support higher clock frequencies and comes in a prototyping friendly QFP (Quad Flat Pack, surface-mountable IC package with easily accessible pins on all four sides) package, the LP1K requires less power and is in a small $3 \times 3\,\mathrm{mm}^2$ BGA (Ball Grid Array, solder balls are placed in a grid below the device, similarly to bump-bonding) package.

The HX1K was put together with two voltage regulators, debugging pins and 8P8C modular connectors, which are also used by the USBpix burn in card, onto a printed circuit board (PCB), which is shown in Figure 6.11. The on-board PLL was used to increase the reference clock from USBpix from 40 MHz to 80 MHz, thus enabling the device to output the full 160 MBit/s when using DDR registers. While the source code is already prepared to handle four chips, only two connections to front end chips are physically implemented and their 40 MBit/s data outputs are sampled synchronously to the reference clock. These are then written to the output in an interleaved mode.

[2] Interestingly, even down to $1.4 \times 1.48\,\mathrm{mm}^2$, but with rather severe limitations.

7 Results

In parallel to this thesis, the first four chip modules were manufactured and became available in time for testing. Therefore, many results will reference these modules, one of which is shown in Figure 7.1. They exhibit a 2 × 2 front end geometry bump-bonded to a common planar silicon pixel sensor. The chips on the south and on the north face each other at their top edge in the coordinates of the read out chip. The wire bonds from each read out chip's protruding bottom edge can be seen at the north and south of the module. Every time a result from this module is shown, the plots will have the same 2 × 2 structure with the top read out chips rotated by 180°.

7.1 Readout and tuning a four chip module

A first test for the read out of four chips in parallel is given by a global register test, in which every register of the front end chip is set to a known value and then read back. Registers which would influence the data transmission are excluded, such as the selection of the transmission rate. This verifies the correct configuration of global registers and the raw data read out processor for relatively low data rates. The test was able to successfully read and write all tested global registers.

A next step is to test histogramming. This can be done with a digital test scan, which bypasses the analog front end of the read out chip and digitally injects signals into the read out logic for selected pixels. 200 hits are injected into each pixel. The test result in Figure 7.2 shows successful histogramming on all four modules. It also verifies the operation of the scan loop controlling the injection of signals for four chips. This in turn requires successful configuration of single pixels, which has not yet been tested in the global register test.

Completely tuning the module further increases the test coverage. The tunables for the analog front end of the read out chips are adjusted such that a test pulse of well-defined charge results in a nominal ToT response, which is uniform over the whole module. The top plot in Figure 7.3 shows the non-uniformity in the mean ToT response before tuning. The bottom plot shows the mean ToT after tuning to a threshold of 3000 e^-. The results look promising, although there is clearly an anomaly visible on the south east chip. Tests with a different read out system revealed the same problem there. This test verified that ToT histogramming works properly and the original tuning procedures correctly handle four chip modules.

At last, a source scan was performed to provide additional verification for the raw data read out processor. 10000 triggers on signals from a ^{90}Sr source are shown in Figure 7.4. Even though the FE-I4 is able to internally generate triggers, this mode could not be used, as the chip reproducibly entered a state in which it consumed a large amount of current

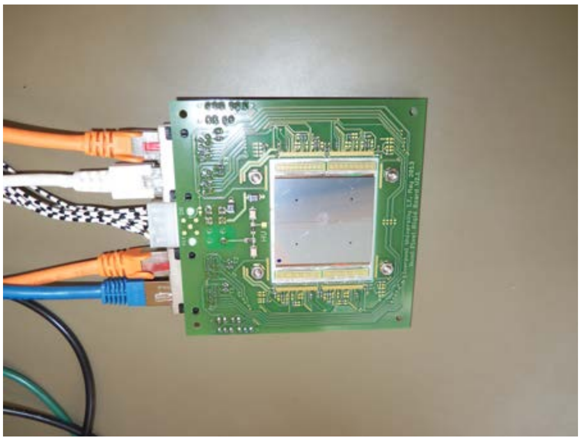


Figure 7.1: A four-chip module from the sensor side. Four FE-I4 chips are arranged in a 2 × 2 *pattern on the backside. The wire bonds to the carrier PCB can be seen on the top and bottom edge.*

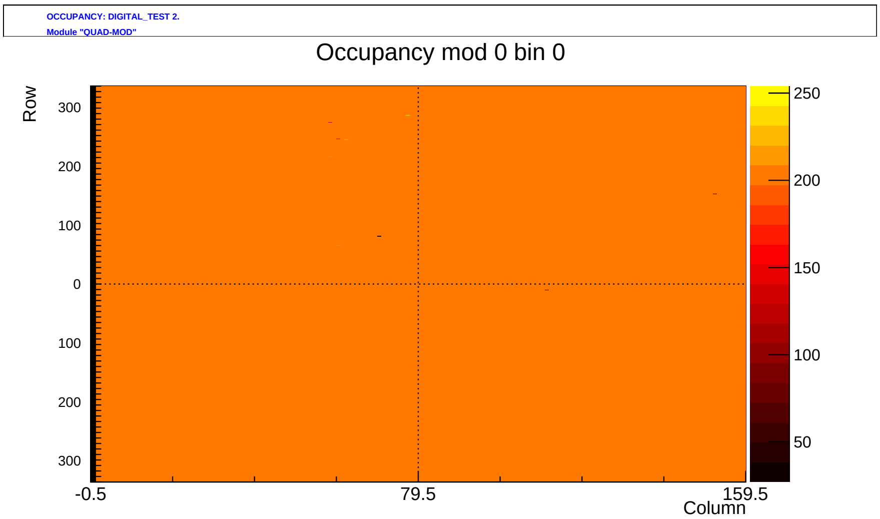


Figure 7.2: Digital test scan of the four chip module [20].

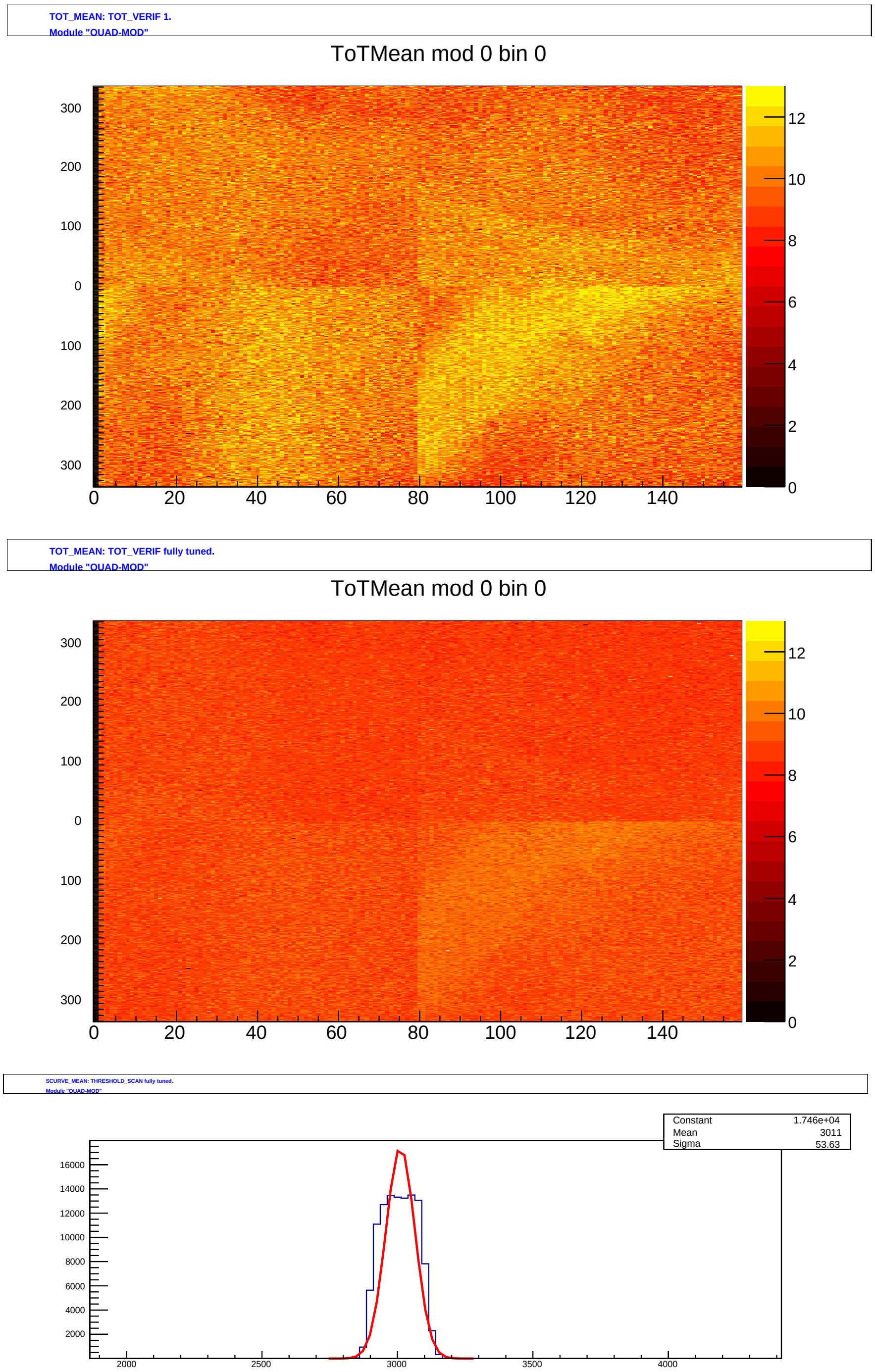


Figure 7.3: The mean ToT of a calibration pulse before (top) and after (middle) tuning of the four-chip module. The bottom plot shows the ToT distribution over the whole chip [20].

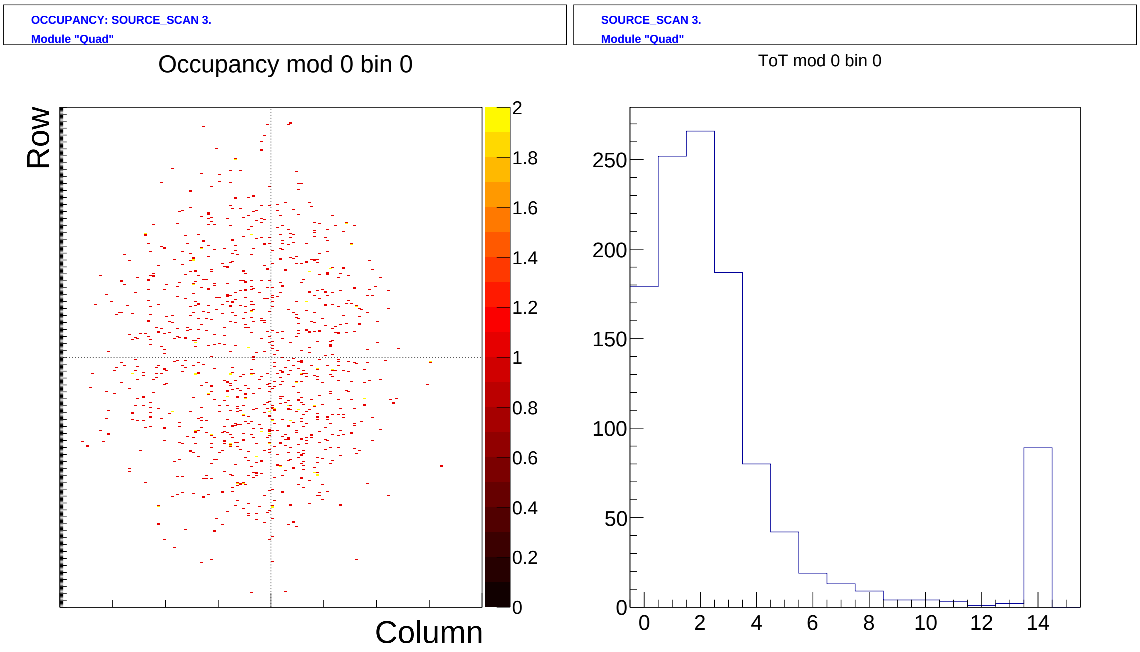


Figure 7.4: 10^5 *triggers with a* ^{90}Sr *source, triggered by an external scintillator [26]. The histogram is a hit map, the right shows the distribution of ToT values cumulated over all pixels.*

and seized to operate correctly. With an external trigger source, however, the problem faded and a normal source scan could be performed. Cross-checks with another read out system did not show a similar behaviour. So far further checks failed to unveil the cause for this issue.

7.2 Data Recovery

For the four chip module, all read out chips were clocked by the USBpix clock and thus the previous methods offer limited value for the data recovery validity test. A much more expressive result is obtained by switching back to operating a module with two chips using two USBpix units. While this previously required the oscillators of both devices to be linked to provide a constant phase between the two clocks, this is no longer the case when using data recovery. Letting the oscillator of the slave board which does not send its clock to any of the chips run free, results in a large number of clock cycles with instantaneously varying output data rates of the data recovery block. Figure 7.5 shows an analog test, in which signals are injected into the analog front end of both read out chips. The pixels that do not show the nominal 200 hits result from actual physical defects.

Another test vector is to measure the effects of routing on the data recovery block. This is based on the idea that for data recovery with four samples it could be potentially fatal if it is possible to interchange two samples. In that case, one might not take a sample on a plateau of the signal but rather directly on the edge. This would for example be the

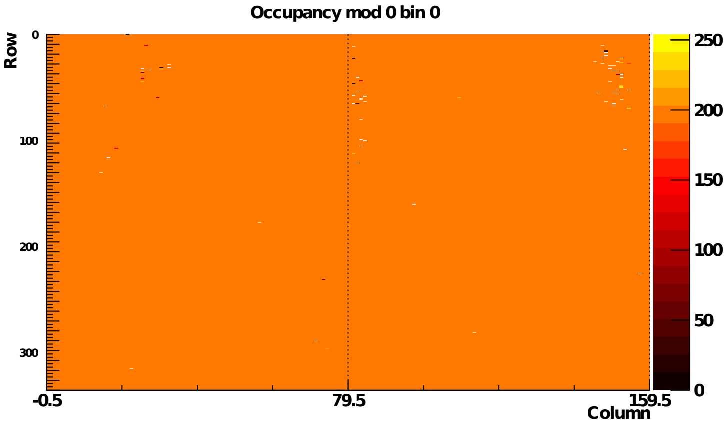


Figure 7.5: Analog test of a module with two chips and two separate USBpix units, each with a normal adapter card with only one channel but without synchronized oscillators.

Sample	Data offset Δt_d	Clock offset Δt_c	$\Delta t_\mathrm{d} - \Delta t_\mathrm{c}$	Sample position skew	Slack
0°	1.119 ns	0.831 ns	288 ps	28 ps	1.18 ns
90°	1.112 ns	0.852 ns	260 ps	372 ps	0.84 ns
180°	1.119 ns	0.831 ns + 0.4 ns	−112 ps	28 ps	1.18 ns
270°	1.112 ns	0.852 ns + 0.4 ns	−140 ps	−428 ps	0.78 ns

Table 7.1: Values of the clock t_c and data offsets t_d for one data recovery channel obtained from the FPGA configuration. Using the difference between these shifts, which give the actual position relative to the signal, one can obtain the actual distance between the samples. The value given is the skew of the next sample position in reference to the current sample. The last value is the slack given the slack based on $\tau_\mathrm{max} = 1.21$ ns.

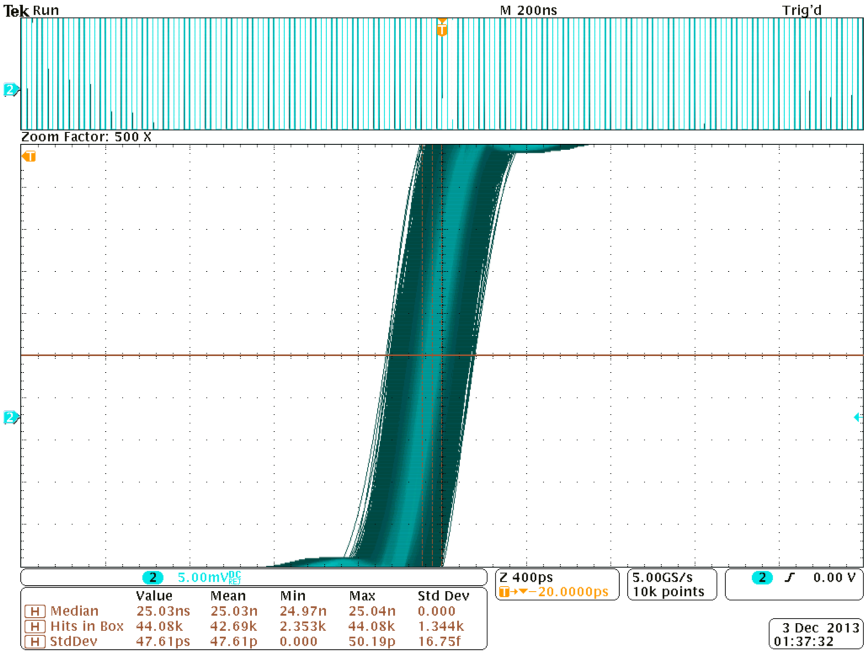
case if one data signal is delayed by more than $\frac{1}{4}T_\mathrm{s}$ (sampling clock), for example by a bad routing decision made by the FPGA toolchain.

Additionally, one has to take period jitter of the clock into account. As period jitter results from noise in electronic circuits, it is a statistical effect and a gaussian distribution is assumed. Typically, period jitter is given as at least 7σ, where σ^2 is the variance of the normal distribution fitted to the period jitter [34]. Figure 7.6 shows the measurement for the period jitter on one of the clock outputs of the burn-in card, which are directly derived from the sampling clock. A value of $\sigma = 47.61 \pm 0.02$ ps is obtained and thus a value of $j = 350$ ps $\geq 7\sigma$ is used.

Having the period jitter j and the sampling clock T_s, the gap between two sampling points available for delay mismatch introduced by routing differences is

$$\tau_\mathrm{max} = \frac{1}{4}T_s - j = \frac{1}{4} \cdot 6.25\ \mathrm{ns} - 350\ \mathrm{ps} \approx 1.21\ \mathrm{ns},$$

as four samples per clock cycle are recorded and the period jitter measures the jitter between two clock edges. It is convenient, that FPGAs are very thoroughly characterized in regards of propagation delays. A figure for the delay offset between the four sampling flip



Tek Run M 200ns Trig'd

Zoom Factor: 500 X

2 5.00mV DC REJ

Z 400ps
T→▼−20.0000ps

5.00GS/s
10k points

2 ʃ 0.00 V

	Value	Mean	Min	Max	Std Dev
H Median	25.03ns	25.03n	24.97n	25.04n	0.000
H Hits in Box	44.08k	42.69k	2.353k	44.08k	1.344k
H StdDev	47.61ps	47.61p	0.000	50.19p	16.75f

3 Dec 2013
01:37:32

Figure 7.6: Period jitter measurement at the reference clock output of one burn-in card channel.

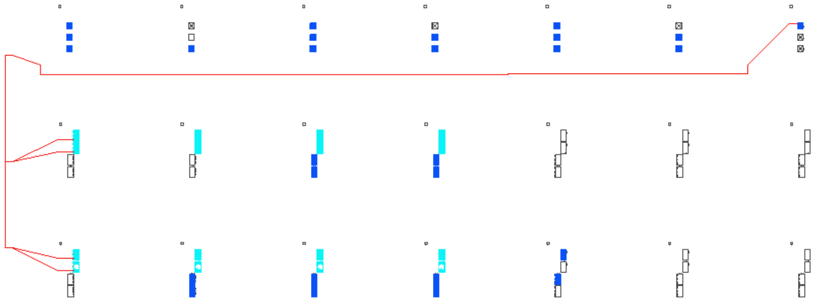


Figure 7.7: Routing of the data signal from the I/O pad (top right) to the four input registers (bottom left).

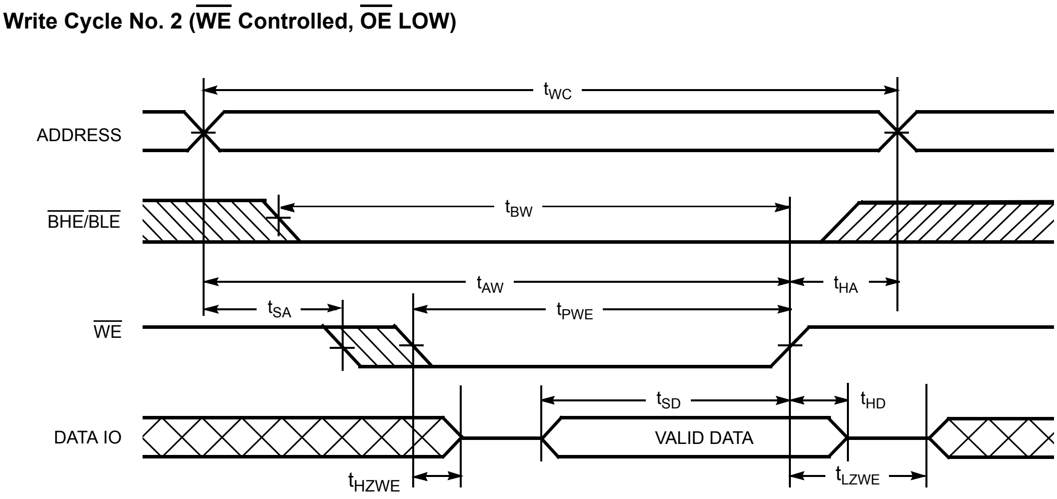


Figure 7.8: SRAM write wave form using WE for signalling [15]. Notice that WE, BHE and BLE are inverted.

flops can be obtained directly from the final configuration data. Figure 7.7 shows the routing for one link from its input pad to the data recovery registers. The values are shown in Table 7.1 together with the resulting effective shift of the sampling point and the remaining available slack for one example channel. A comfortable amount of slack is observed.

7.3 Memory Arbiter

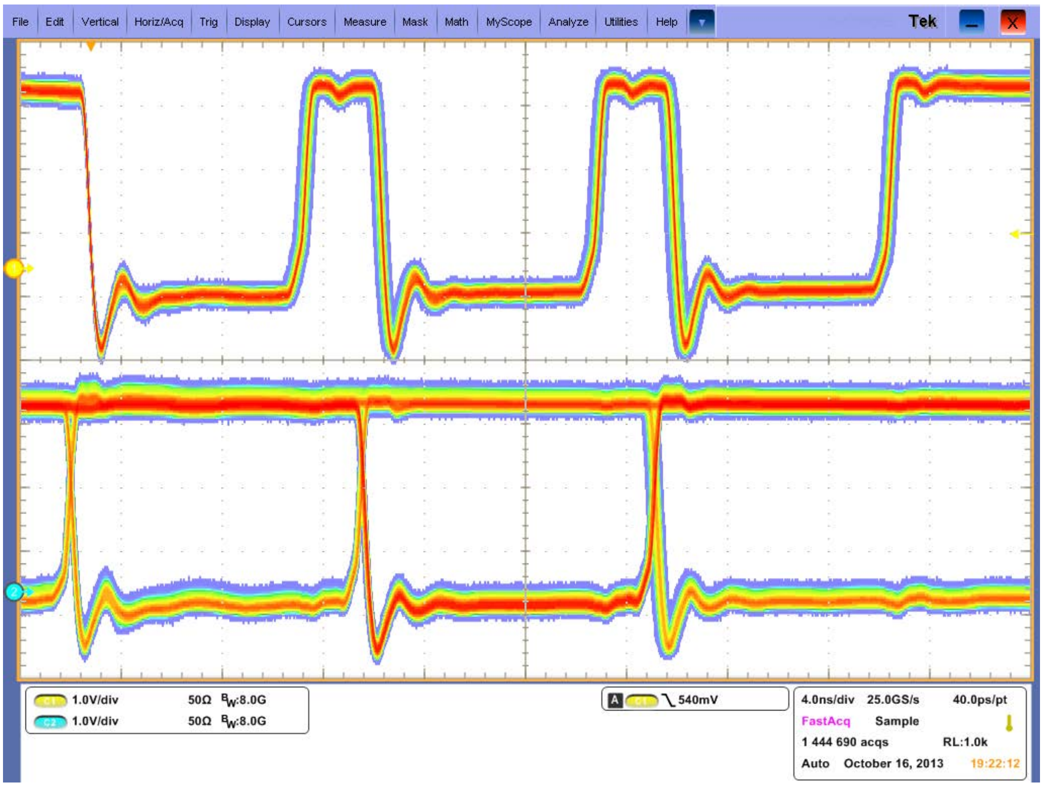
Testing the memory arbiter has been done in detail. This includes thorough verification of the configuration using a logic analyzer embedded into the FPGA configuration. The primary external vectors are of course all previously performed tests, as they thoroughly tested all required functions. In addition to the resulting histograms and raw data, the FIFOs inside the memory arbiter were configured to issue an overflow signal, which is checked by the host software. During testing, no overflow was observed.

By modifying the read out processors, a continuous stream of either add or write operations were created at the rate equal to the theoretical maximum calculated in section 5.5. Both operations were handled correctly.

In addition, the signals as they arrive at the SRAM have been analysed. The SRAM datasheet [15] specifies their timing accurately. This is shown in Figure 7.8 and Table 7.2. Some of the signals, WE, BLE and BHE are inverted before transmission. One that has to be taken very seriously is the correct application of write enable (WE) pulses. This pulse has to be at least 7 ns long and shall not occur more than once within a 10 ns window. The address lines have to be valid as soon as the WE pulse starts and stay in this state until it ends. The data lines are less restricted and only are not allowed to change any more after 5.5 ns to the end of the WE pulse. Figure 7.9 shows a measurement of the inverted WE

Symbol	Description	Minimum
t_{WC}	Write Cycle Time	10 ns
t_{AW}	Address Setup to Write End	7 ns
t_{HA}	Address Hold From Write End	0 ns
t_{SA}	Address Setup to Write Start	0 ns
t_{PWE}	WE Pulse Width	7 ns
t_{SD}	Data Setup to Write End	5.5 ns
t_{HD}	Data Hold From Write End	0 ns

Table 7.2: The minimum values for the symbols in Figure 7.8 [15].



C1 1.0V/div 50Ω B_W:8.0G
C2 1.0V/div 50Ω B_W:8.0G

A C1 ╲ 540mV

4.0ns/div 25.0GS/s 40.0ps/pt
FastAcq Sample
1 444 690 acqs RL:1.0k
Auto October 16, 2013 19:22:12

Figure 7.9: SRAM WR signal (top, inverted) in comparison to a random address line (bottom).

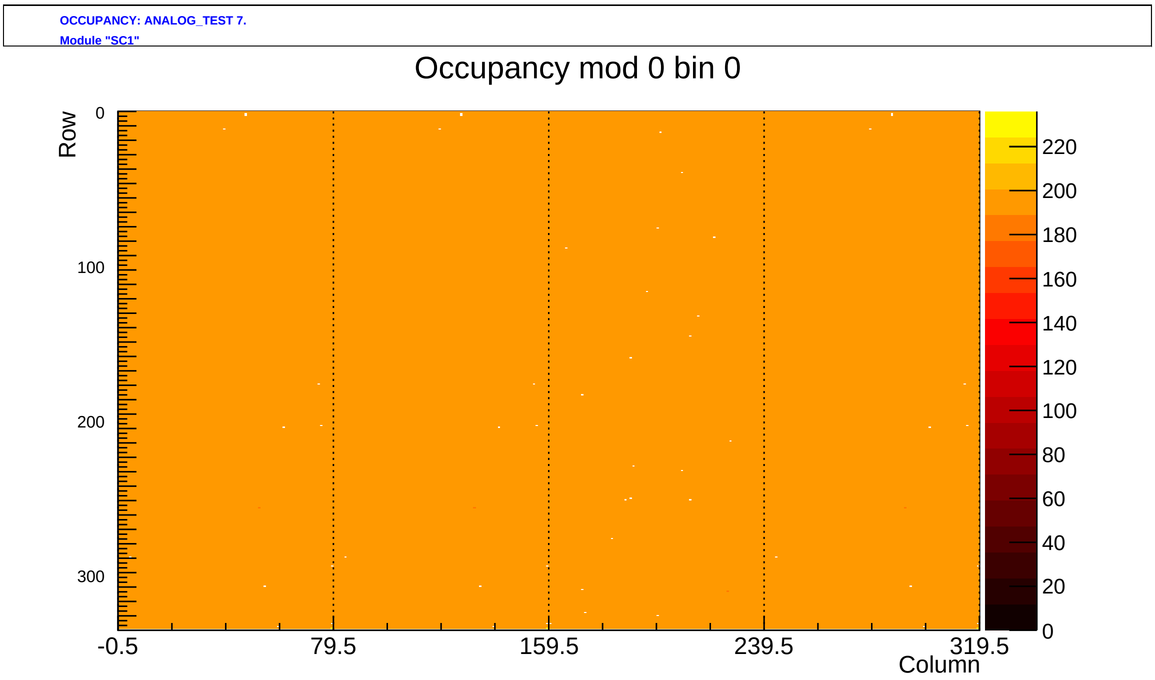


Figure 7.10: Analog test of two single FE-I4 chips transmitting at 40 MBit/s *via one data link decoded at* 160 MBit/s. *All four decoded time slots are lined up from right to left separated by dashed lines.*

pulse (top) and a random address line (bottom). All conditions are met.

7.4 Interleaving

To test deinterleaving, the presented HX1K test board was connected to two FE-I4 chips. Both chips were configured to transmit data at 40 Mbit/s. The HX1K test board was configured to create a 160 Mbit/s data stream from this by replicating the data of one chip to three time slots and transferring the data of the remaining chip in the remaining time slot. Figure 7.10 shows an analog test scan with this configuration. In contrast to the four chip module, all chips are displayed in one row. One can see that for the replicated chips, all histograms have the same bin content.

It was mentioned that the clock duty cycle needed to be corrected. The result of this is shown in Figure 7.11. From top to bottom, the four outputs with the duty cycle correction are shown. Best channel has a duty cycle of 0.511 ± 0.003 %, the worst 0.540 ± 0.001 %.

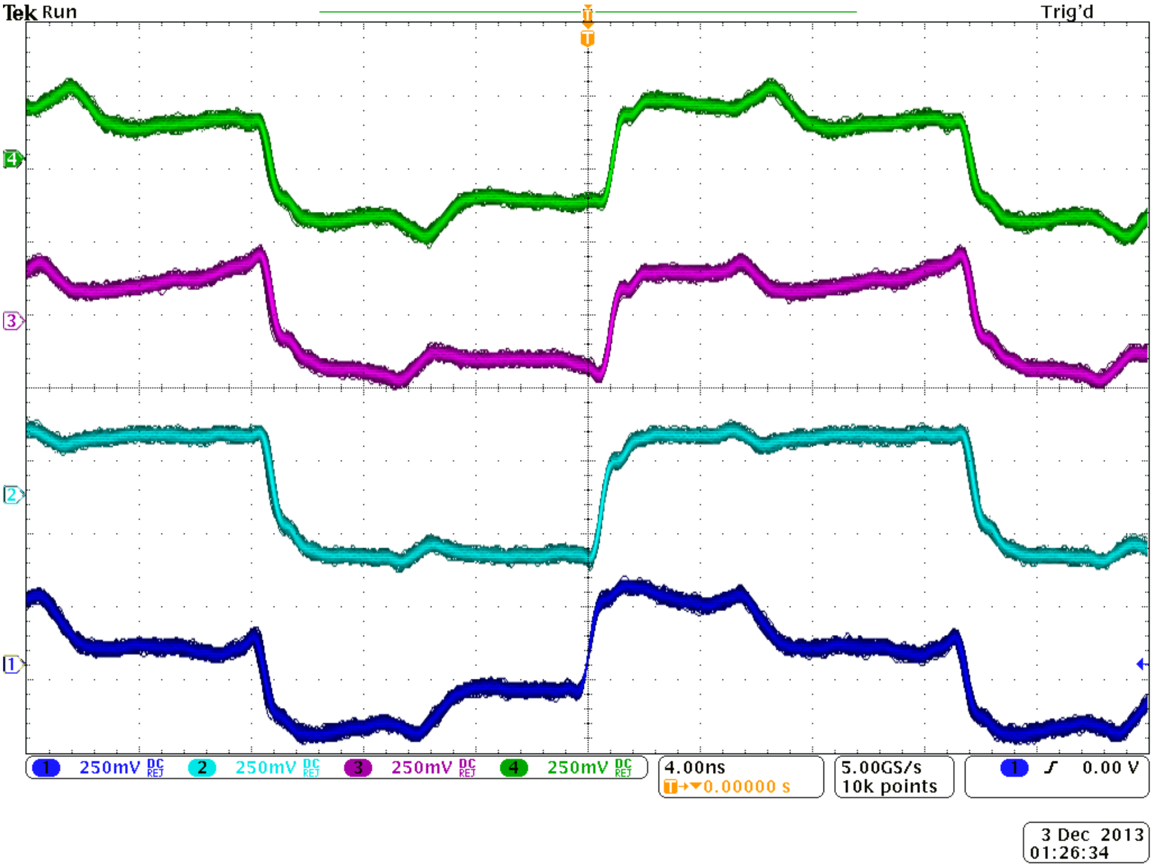


Figure 7.11: Oscilloscope capture of the four clock outputs of the burn-in card.

8 Conclusion and Outlook

The upgrade of the existing USBpix software to support parallel read out of up to four FE-I4 chips has been presented. The design choices have been explained and evaluated based on the most recent stable USBpix software release. Based on the sketch delivered by the design, an implementation was devised and justified.

The resulting changes have been validated in a multitude of experimental set-ups, especially tuning of a four-chip module, and deinterleaving tests, but also the functionality of two critical blocks, the memory arbiter and data recovery. While most aspects of the configuration could be validated successfully, an issue with the FE Self Trigger mode, which can only be reproduced with USBpix, remained. This issue will be investigated further.

While the software upgrade enabled fast access to four chip module prototyping with existing and in the ATLAS pixel community commonly available hardware, higher data rates are not achievable with a similar scheme, as the configuration comes very close to the maximum throughput of the SRAM. Additionally, many parts of the FPGA configuration are operating close to their maximum clock frequency. Larger data rates on the input channels will hardly be achievable with the currently used XC3S1000 FPGA. Thus, work on a hardware upgrade is ongoing at the Universität Bonn.

A future focus is given by implementing further link sharing mechanisms. This includes the possibility of more complicated, for example packet oriented, multiple access methods for the data link, which could further increase the efficiency by dynamically sharing several links as it is done in switched networks. Also, using separate clock and command links is rather inefficient, as, with an appropriate channel coding, the clock can be recovered from the command link. Preliminary tests already indicated promising results. In that instance, pulse width encoding of the command line was used. A 25% duty cycle pulse corresponded to a zero and a 75% duty cycle pulse to a one. The clock was recovered by simple division of this signal and conditioned using the PLL on the HX1K FPGA. With this scheme, a four chip module could be operated with just a common clock and command link and one data link. Other more complicated implementations, which require less bandwidth and provide DC balance, for example by recovering a clock from an 8B/10B signal, are also being discussed [18].

Additionally, as the HX1K board has proven the feasibility of testing these channel sharing techniques with the iCE40 FPGA family, a smaller version of the board using a iCE40 LP1K FPGA is planned to be manufactured. The board, as shown in Figure 8.1, is supposed to be directly glued and wire bonded to the PCB on the back side of a four chip module. This board supports the required connections for interleaving four 40 MBit/s data streams to one data link and on-module clock recovery.

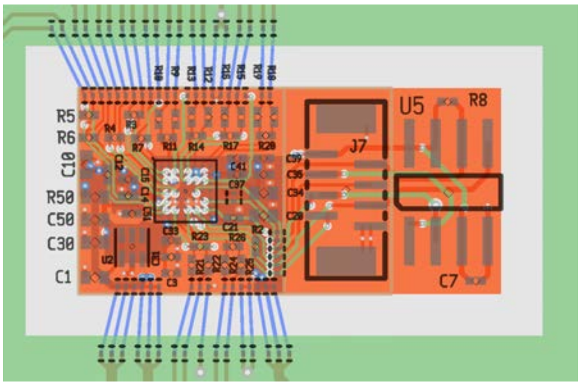


Figure 8.1: Layout of the proposed LP1K PCB (orange). On the left, a 1 × 1 cm^2 square contains the FPGA and necessary passive components as well as a voltage regulator. An optional zero insertion force programming header and a flash memory for storing reprogrammable configurations are placed on the right.

Bibliography

[1] *ATLAS muon spectrometer: Technical Design Report*. CERN, Geneva, 1997. ATLAS-TDR-10. CERN-LHCC-97-022.

[2] Multi-IO board schematic and layout prints, Jan 2010. `icwiki.physik.uni-bonn.de/twiki/bin/view/Systems/UsbPix`.

[3] IGLOO nano low power flash FPGAs, Nov 2013. Datasheet. `www.microsemi.com/`.

[4] M. Backhaus et al. Development of a versatile and modular test system for ATLAS hybrid pixel detectors. *Nucl.Instrum.Meth.*, A650(1):37 – 40, 2011.

[5] J. Beringer et al. Review of particle physics. *Phys. Rev. D*, 86:010001, Jul 2012.

[6] O. S. Brüning et al. *LHC Design Report*. CERN, Geneva, 2004. CERN-2004-003-V-1.

[7] A. Bulgheroni. Results from the EUDET telescope with high resolution planes. *Nucl.Instrum.Meth.*, A623:399–401, 2010.

[8] M. Capeans et al. *ATLAS Insertable B-Layer Technical Design Report*. Geneva, Sep 2010. CERN-LHCC-2010-013. ATLAS-TDR-019.

[9] The ATLAS Collaboration. *ATLAS Pixel Detector: Technical Design Report*. CERN, Geneva, May 1998. CERN-LHCC-98-13.

[10] The ATLAS Collaboration. *Letter of Intent for the Phase-I Upgrade of the ATLAS Experiment*. Geneva, Nov 2011. CERN-LHCC-2011-012. LHCC-I-020.

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

[12] The ATLAS Collaboration. Observation of a new particle in the search for the standard model higgs boson with the ATLAS detector at the LHC. *Physics Letters B*, 716(1):1 – 29, 2012.

[13] The CMS Collaboration. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Physics Letters B*, 716(1):30 – 61, 2012.

[14] The FE-I4 Collaboration. The FE-I4B integrated circuit guide. 2012. Version 2.3.

[15] CY7C1061AV33 - 16-Mbit (1 M × 16) Static RAM, 2011. Datasheet.

[16] DS099: Spartan-3 FPGA family data sheet, Jun 2013. `www.xilinx.com`.

[17] DS099: Spartan-3 FPGA family data sheet: DC and switching characteristics, Jun 2013. `www.xilinx.com`.

[18] M. Garcia-Sciveres. Private communications.

[19] P Grenier. Silicon sensor technologies for the ATLAS IBL upgrade. Technical Report ATL-INDET-PROC-2011-006, CERN, Geneva, Sep 2011.

[20] J. Große-Knetter. Private communications.

[21] J. Große-Knetter. *Vertex Measurement at a Hadron Collider - The ATLAS Pixel Detector*. Habilitation thesis, Nov 2007.

[22] C. Hu-Guo et al. A ten thousand frames per second readout MAPS for the EUDET beam telescope. 2009.

[23] iCE40 (TM) LP/HX family data sheet, Oct 2013.

[24] I. Peric. Active pixel sensors in high-voltage cmos technologies for atlas. *Journal of Instrumentation*, 7(08):C08002, 2012.

[25] I. Perić et al. The FEI3 readout chip for the ATLAS pixel detector. *Nucl.Instrum.Meth.*, A565(1):178 – 187, 2006.

[26] J. Rieger. Private communications.

[27] L. Rossi and O. S. Brüning. High Luminosity Large Hadron Collider — A description for the European Strategy Preparatory Group. Technical report, CERN, Geneva, Aug 2012. CERN-ATS-2012-236.

[28] Nick Sawyer. *XAPP224: Data Recovery*. Xilinx, 2005. `www.xilinx.com`,.

[29] The ATLAS Collaboration. *ATLAS detector and physics performance: Technical Design Report, Volume 1*. Technical Design Report ATLAS. CERN, Geneva, 1999.

[30] UG331: Spartan-3 generation FPGA user guide, Jun 2011. `www.xilinx.com`.

[31] J. Weingarten et al. Planar pixel sensors for the atlas upgrade: beam tests results. *Journal of Instrumentation*, 7(10):P10028, 2012.

[32] N. Wermes. Pixel detectors for charged particles. *Nucl.Instrum.Meth.*, A604(1-2):370–379, 2009.

[33] A.X. Widmer and P.A. Franaszek. A dc-balanced, partitioned-block, 8b/10b transmission code. *IBM Journal of research and development*, 27(5):440–451, 1983.

[34] XAPP462: using digital clock managers (DCMs) in Spartan-3 fpgas., Jun 2011. pgs. 108, 117. `www.xilinx.com`.

Acknowledgements

I thank Arnulf Quadt and Jörn Große-Knetter for making this thesis possible, their outstanding organizational efforts and professional supervision. They and Maurice Garcia-Sciveres made it possible to visit the Lawrence Berkeley National Laboratory during this thesis. Working with Maurice Garcia-Sciveres provided me with unprecedented insights into read out developments, techniques and the currently ongoing research. I also extend thanks to the whole hardware group for moral support, in particular Jens Weingarten, Julia Rieger and again Jörn Große-Knetter for prompt testing and providing me with their knowledge.

Finally I would like to express my gratitude towards Sabine Wolff, my family and friends for supporting me, which allowed me to write this thesis in the first place.